```
**********************************************************
   17173 Wed Aug 13 19:32:51 2014
new/usr/src/lib/openssl/include/bn_lcl.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/bn/bn_lcl.h */
   2 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
   3  * All rights reserved.
   4  *
   5  * This package is an SSL implementation written
   6  * by Eric Young (eay@cryptsoft.com).
   7  * The implementation was written so as to conform with Netscapes SSL.
   8  *
   9  * This library is free for commercial and non-commercial use as long as
  10  * the following conditions are aheared to.  The following conditions
  11  * apply to all code found in this distribution, be it the RC4, RSA,
  12  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
  13  * included with this distribution is covered by the same copyright terms
  14  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  15  *
  16  * Copyright remains Eric Young's, and as such any Copyright notices in
  17  * the code are not to be removed.
  18  * If this package is used in a product, Eric Young should be given attribution
  19  * as the author of the parts of the library used.
  20  * This can be in the form of a textual message at program startup or
  21  * in documentation (online or textual) provided with the package.
  22  *
  23  * Redistribution and use in source and binary forms, with or without
  24  * modification, are permitted provided that the following conditions
  25  * are met:
  26  * 1. Redistributions of source code must retain the copyright
  27  *    notice, this list of conditions and the following disclaimer.
  28  * 2. Redistributions in binary form must reproduce the above copyright
  29  *    notice, this list of conditions and the following disclaimer in the
  30  *    documentation and/or other materials provided with the distribution.
  31  * 3. All advertising materials mentioning features or use of this software
  32  *    must display the following acknowledgement:
  33  *    "This product includes cryptographic software written by
  34  *     Eric Young (eay@cryptsoft.com)"
  35  *    The word 'cryptographic' can be left out if the rouines from the library
  36  *    being used are not cryptographic related :-).
  37  * 4. If you include any Windows specific code (or a derivative thereof) from
  38  *    the apps directory (application code) you must include an acknowledgement:
  39  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  40  *
  41  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
  42  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  43  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  44  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
  45  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  46  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  47  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  48  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  49  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  50  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  51  * SUCH DAMAGE.
  52  *
  53  * The licence and distribution terms for any publically available version or
  54  * derivative of this code cannot be changed.  i.e. this code cannot simply be
  55  * copied and put under another distribution licence
  56  * [including the GNU Public Licence.]
  57  */
  58 /* ====================================================================
  59  * Copyright (c) 1998-2000 The OpenSSL Project.  All rights reserved.
  60  *
  61  * Redistribution and use in source and binary forms, with or without
```



```
  62  * modification, are permitted provided that the following conditions
  63  * are met:
  64  *
  65  * 1. Redistributions of source code must retain the above copyright
  66  *    notice, this list of conditions and the following disclaimer.
  67  *
  68  * 2. Redistributions in binary form must reproduce the above copyright
  69  *    notice, this list of conditions and the following disclaimer in
  70  *    the documentation and/or other materials provided with the
  71  *    distribution.
  72  *
  73  * 3. All advertising materials mentioning features or use of this
  74  *    software must display the following acknowledgment:
  75  *    "This product includes software developed by the OpenSSL Project
  76  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
  77  *
  78  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  79  *    endorse or promote products derived from this software without
  80  *    prior written permission. For written permission, please contact
  81  *    openssl-core@openssl.org.
  82  *
  83  * 5. Products derived from this software may not be called "OpenSSL"
  84  *    nor may "OpenSSL" appear in their names without prior written
  85  *    permission of the OpenSSL Project.
  86  *
  87  * 6. Redistributions of any form whatsoever must retain the following
  88  *    acknowledgment:
  89  *    "This product includes software developed by the OpenSSL Project
  90  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
  91  *
  92  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ''AS IS'' AND ANY
  93  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  94  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  95  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  96  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  97  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  98  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  99  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 100  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 101  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 102  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 103  * OF THE POSSIBILITY OF SUCH DAMAGE.
 104  * ====================================================================
 105  *
 106  * This product includes cryptographic software written by Eric Young
 107  * (eay@cryptsoft.com).  This product includes software written by Tim
 108  * Hudson (tjh@cryptsoft.com).
 109  *
 110  */

 112 #ifndef HEADER_BN_LCL_H
 113 #define HEADER_BN_LCL_H

 115 #include <openssl/bn.h>

 117 #ifdef  __cplusplus
 118 extern "C" {
 119 #endif


 122 /*
 123  * BN_window_bits_for_exponent_size -- macro for sliding window mod_exp function
 124  *
 125  *
 126  * For window size 'w' (w >= 2) and a random 'b' bits exponent,
 127  * the number of multiplications is a constant plus on average
```

```
 *
 *    2^(w-1) + (b-w)/(w+1);
 *
 * here  2^(w-1)  is for precomputing the table (we actually need
 * entries only for windows that have the lowest bit set), and
 * (b-w)/(w+1)  is an approximation for the expected number of
 * w-bit windows, not counting the first one.
 *
 * Thus we should use
 *
 *    w >= 6  if        b > 671
 *     w = 5  if  671 > b > 239
 *     w = 4  if  239 > b >  79
 *     w = 3  if   79 > b >  23
 *    w <= 2  if   23 > b
 *
 * (with draws in between).  Very small exponents are often selected
 * with low Hamming weight, so we use  w = 1  for b <= 23.
 */
#if 1
#define BN_window_bits_for_exponent_size(b) \
                ((b) > 671 ? 6 : \
                 (b) > 239 ? 5 : \
                 (b) >  79 ? 4 : \
                 (b) >  23 ? 3 : 1)
#else
/* Old SSLeay/OpenSSL table.
 * Maximum window size was 5, so this table differs for b==1024;
 * but it coincides for other interesting values (b==160, b==512).
 */
#define BN_window_bits_for_exponent_size(b) \
                ((b) > 255 ? 5 : \
                 (b) > 127 ? 4 : \
                 (b) >  17 ? 3 : 1)
#endif



/* BN_mod_exp_mont_conttime is based on the assumption that the
 * L1 data cache line width of the target processor is at least
 * the following value.
 */
#define MOD_EXP_CTIME_MIN_CACHE_LINE_WIDTH      ( 64 )
#define MOD_EXP_CTIME_MIN_CACHE_LINE_MASK       (MOD_EXP_CTIME_MIN_CACHE_LINE_WI

/* Window sizes optimized for fixed window size modular exponentiation
 * algorithm (BN_mod_exp_mont_consttime).
 *
 * To achieve the security goals of BN_mode_exp_mont_consttime, the
 * maximum size of the window must not exceed
 * log_2(MOD_EXP_CTIME_MIN_CACHE_LINE_WIDTH).
 *
 * Window size thresholds are defined for cache line sizes of 32 and 64,
 * cache line sizes where log_2(32)=5 and log_2(64)=6 respectively. A
 * window size of 7 should only be used on processors that have a 128
 * byte or greater cache line size.
 */
#if MOD_EXP_CTIME_MIN_CACHE_LINE_WIDTH == 64

#  define BN_window_bits_for_ctime_exponent_size(b) \
                ((b) > 937 ? 6 : \
                 (b) > 306 ? 5 : \
                 (b) >  89 ? 4 : \
                 (b) >  22 ? 3 : 1)
#  define BN_MAX_WINDOW_BITS_FOR_CTIME_EXPONENT_SIZE    (6)
```



```
#elif MOD_EXP_CTIME_MIN_CACHE_LINE_WIDTH == 32

#  define BN_window_bits_for_ctime_exponent_size(b) \
                ((b) > 306 ? 5 : \
                 (b) >  89 ? 4 : \
                 (b) >  22 ? 3 : 1)
#  define BN_MAX_WINDOW_BITS_FOR_CTIME_EXPONENT_SIZE    (5)

#endif


/* Pentium pro 16,16,16,32,64 */
/* Alpha       16,16,16,16.64 */
#define BN_MULL_SIZE_NORMAL                     (16) /* 32 */
#define BN_MUL_RECURSIVE_SIZE_NORMAL            (16) /* 32 less than */
#define BN_SQR_RECURSIVE_SIZE_NORMAL            (16) /* 32 */
#define BN_MUL_LOW_RECURSIVE_SIZE_NORMAL        (32) /* 32 */
#define BN_MONT_CTX_SET_SIZE_WORD               (64) /* 32 */

#if !defined(OPENSSL_NO_ASM) && !defined(OPENSSL_NO_INLINE_ASM) && !defined(PEDA
/*
 * BN_UMULT_HIGH section.
 *
 * No, I'm not trying to overwhelm you when stating that the
 * product of N-bit numbers is 2*N bits wide:-) No, I don't expect
 * you to be impressed when I say that if the compiler doesn't
 * support 2*N integer type, then you have to replace every N*N
 * multiplication with 4 (N/2)*(N/2) accompanied by some shifts
 * and additions which unavoidably results in severe performance
 * penalties. Of course provided that the hardware is capable of
 * producing 2*N result... That's when you normally start
 * considering assembler implementation. However! It should be
 * pointed out that some CPUs (most notably Alpha, PowerPC and
 * upcoming IA-64 family:-) provide *separate* instruction
 * calculating the upper half of the product placing the result
 * into a general purpose register. Now *if* the compiler supports
 * inline assembler, then it's not impossible to implement the
 * "bignum" routines (and have the compiler optimize 'em)
 * exhibiting "native" performance in C. That's what BN_UMULT_HIGH
 * macro is about:-)
 *
 *                                      <appro@fy.chalmers.se>
 */
# if defined(__alpha) && (defined(SIXTY_FOUR_BIT_LONG) || defined(SIXTY_FOUR_BIT
#  if defined(__DECC)
#   include <c_asm.h>
#   define BN_UMULT_HIGH(a,b)   (BN_ULONG)__asm__("umulh %a0,%a1,%v0",(a),(b))
#   define BN_UMULT_HIGH(a,b)   (BN_ULONG)asm("umulh %a0,%a1,%v0",(a),(b))
#  elif defined(__GNUC__) && __GNUC__>=2
#   define BN_UMULT_HIGH(a,b)   ({      \
        register BN_ULONG ret;          \
        __asm__ ("umulh %1,%2,%0"       \
        asm ("umulh     %1,%2,%0"       \
             : "=r"(ret)                \
             : "r"(a), "r"(b));         \
        ret;                    })
#  endif        /* compiler */
# elif defined(_ARCH_PPC) && defined(__64BIT__) && defined(SIXTY_FOUR_BIT_LONG)
#  if defined(__GNUC__) && __GNUC__>=2
#   define BN_UMULT_HIGH(a,b)   ({      \
        register BN_ULONG ret;          \
        __asm__ ("mulhdu        %0,%1,%2"       \
        asm ("mulhdu    %0,%1,%2"       \
             : "=r"(ret)                \
             : "r"(a), "r"(b));         \
        ret;                    })
```

```
257 #  endif        /* compiler */
258 # elif (defined(__x86_64) || defined(__x86_64__)) && \
259       (defined(SIXTY_FOUR_BIT_LONG) || defined(SIXTY_FOUR_BIT))
260 #  if defined(__GNUC__) && __GNUC__>=2
261 #   define BN_UMULT_HIGH(a,b)   ({      \
262         register BN_ULONG ret,discard;  \
263         __asm__ ("mulq  %3"             \
263         asm ("mulq      %3"             \
264              : "=a"(discard),"=d"(ret)  \
265              : "a"(a), "g"(b)           \
266              : "cc");                   \
267         ret;                    })
268 #   define BN_UMULT_LOHI(low,high,a,b)  \
269         __asm__ ("mulq  %3"             \
269         asm ("mulq      %3"             \
270                 : "=a"(low),"=d"(high)  \
271                 : "a"(a),"g"(b)         \
272                 : "cc");
273 #  endif
274 # elif (defined(_M_AMD64) || defined(_M_X64)) && defined(SIXTY_FOUR_BIT)
275 #  if defined(_MSC_VER) && _MSC_VER>=1400
276     unsigned __int64 __umulh    (unsigned __int64 a,unsigned __int64 b);
277     unsigned __int64 _umul128   (unsigned __int64 a,unsigned __int64 b,
278                                  unsigned __int64 *h);
279 #   pragma intrinsic(__umulh,_umul128)
280 #   define BN_UMULT_HIGH(a,b)           __umulh((a),(b))
281 #   define BN_UMULT_LOHI(low,high,a,b)  ((low)=_umul128((a),(b),&(high)))
282 #  endif
283 # elif defined(__mips) && (defined(SIXTY_FOUR_BIT) || defined(SIXTY_FOUR_BIT_LON
284 #  if defined(__GNUC__) && __GNUC__>=2
285 #   if __GNUC__>=4 && __GNUC_MINOR__>=4 /* "h" constraint is no more since 4.4 *
286 #    define BN_UMULT_HIGH(a,b)         (((__uint128_t)(a)*(b))>>64)
287 #    define BN_UMULT_LOHI(low,high,a,b) ({      \
288       __uint128_t ret=(__uint128_t)(a)*(b);   \
289       (high)=ret>>64; (low)=ret;       })
290 #   else
291 #    define BN_UMULT_HIGH(a,b) ({       \
292       register BN_ULONG ret;            \
293       __asm__ ("dmultu        %1,%2"          \
293       asm ("dmultu    %1,%2"          \
294              : "=h"(ret)                \
295              : "r"(a), "r"(b) : "l");   \
296         ret;                    })
297 #    define BN_UMULT_LOHI(low,high,a,b)\
298       __asm__ ("dmultu        %2,%3"          \
298       asm ("dmultu    %2,%3"          \
299              : "=l"(low),"=h"(high)     \
300              : "r"(a), "r"(b));
301 #    endif
302 #  endif
303 # endif         /* cpu */
304 #endif          /* OPENSSL_NO_ASM */

306 /*********************************************************
307  * Using the long long type
308  */
309 #define Lw(t)    (((BN_ULONG)(t))&BN_MASK2)
310 #define Hw(t)    (((BN_ULONG)((t)>>BN_BITS2))&BN_MASK2)

312 #ifdef BN_DEBUG_RAND
313 #define bn_clear_top2max(a) \
314         { \
315         int      ind = (a)->dmax - (a)->top; \
316         BN_ULONG *ftl = &(a)->d[(a)->top-1]; \
317         for (; ind != 0; ind--) \
318                 *(++ftl) = 0x0; \
```



```
319         }
320 #else
321 #define bn_clear_top2max(a)
322 #endif

324 #ifdef BN_LLONG
325 #define mul_add(r,a,w,c) { \
326         BN_ULLONG t; \
327         t=(BN_ULLONG)w * (a) + (r) + (c); \
328         (r)= Lw(t); \
329         (c)= Hw(t); \
330         }

332 #define mul(r,a,w,c) { \
333         BN_ULLONG t; \
334         t=(BN_ULLONG)w * (a) + (c); \
335         (r)= Lw(t); \
336         (c)= Hw(t); \
337         }

339 #define sqr(r0,r1,a) { \
340         BN_ULLONG t; \
341         t=(BN_ULLONG)(a)*(a); \
342         (r0)=Lw(t); \
343         (r1)=Hw(t); \
344         }

346 #elif defined(BN_UMULT_LOHI)
347 #define mul_add(r,a,w,c) {              \
348         BN_ULONG high,low,ret,tmp=(a);  \
349         ret =  (r);                     \
350         BN_UMULT_LOHI(low,high,w,tmp);  \
351         ret += (c);                     \
352         (c) =  (ret<(c))?1:0;           \
353         (c) += high;                    \
354         ret += low;                     \
355         (c) += (ret<low)?1:0;           \
356         (r) =  ret;                     \
357         }

359 #define mul(r,a,w,c)    {               \
360         BN_ULONG high,low,ret,ta=(a);   \
361         BN_UMULT_LOHI(low,high,w,ta);   \
362         ret =  low + (c);               \
363         (c) =  high;                    \
364         (c) += (ret<low)?1:0;           \
365         (r) =  ret;                     \
366         }

368 #define sqr(r0,r1,a)    {               \
369         BN_ULONG tmp=(a);               \
370         BN_UMULT_LOHI(r0,r1,tmp,tmp);   \
371         }

373 #elif defined(BN_UMULT_HIGH)
374 #define mul_add(r,a,w,c) {              \
375         BN_ULONG high,low,ret,tmp=(a);  \
376         ret =  (r);                     \
377         high=  BN_UMULT_HIGH(w,tmp);    \
378         ret += (c);                     \
379         low =  (w) * tmp;               \
380         (c) =  (ret<(c))?1:0;           \
381         (c) += high;                    \
382         ret += low;                     \
383         (c) += (ret<low)?1:0;           \
384         (r) =  ret;                     \
```

```
385         }

387 #define mul(r,a,w,c)    {               \
388         BN_ULONG high,low,ret,ta=(a);   \
389         low =  (w) * ta;                \
390         high=  BN_UMULT_HIGH(w,ta);     \
391         ret =  low + (c);               \
392         (c) =  high;                    \
393         (c) += (ret<low)?1:0;           \
394         (r) =  ret;                     \
395         }

397 #define sqr(r0,r1,a)    {               \
398         BN_ULONG tmp=(a);               \
399         (r0) = tmp * tmp;               \
400         (r1) = BN_UMULT_HIGH(tmp,tmp);  \
401         }

403 #else
404 /*************************************************************
405  * No long long type
406  */

408 #define LBITS(a)        ((a)&BN_MASK2l)
409 #define HBITS(a)        (((a)>>BN_BITS4)&BN_MASK2l)
410 #define L2HBITS(a)      (((a)<<BN_BITS4)&BN_MASK2)

412 #define LLBITS(a)       ((a)&BN_MASKl)
413 #define LHBITS(a)       (((a)>>BN_BITS2)&BN_MASKl)
414 #define LL2HBITS(a)     ((BN_ULLONG)((a)&BN_MASKl)<<BN_BITS2)

416 #define mul64(l,h,bl,bh) \
417         { \
418         BN_ULONG m,m1,lt,ht; \
419  \
420         lt=l; \
421         ht=h; \
422         m =(bh)*(lt); \
423         lt=(bl)*(lt); \
424         m1=(bl)*(ht); \
425         ht =(bh)*(ht); \
426         m=(m+m1)&BN_MASK2; if (m < m1) ht+=L2HBITS((BN_ULONG)1); \
427         ht+=HBITS(m); \
428         m1=L2HBITS(m); \
429         lt=(lt+m1)&BN_MASK2; if (lt < m1) ht++; \
430         (l)=lt; \
431         (h)=ht; \
432         }

434 #define sqr64(lo,ho,in) \
435         { \
436         BN_ULONG l,h,m; \
437  \
438         h=(in); \
439         l=LBITS(h); \
440         h=HBITS(h); \
441         m =(l)*(h); \
442         l*=l; \
443         h*=h; \
444         h+=(m&BN_MASK2h1)>>(BN_BITS4-1); \
445         m =(m&BN_MASK2l)<<(BN_BITS4+1); \
446         l=(l+m)&BN_MASK2; if (l < m) h++; \
447         (lo)=l; \
448         (ho)=h; \
449         }
```



```
451 #define mul_add(r,a,bl,bh,c) { \
452         BN_ULONG l,h; \
453  \
454         h= (a); \
455         l=LBITS(h); \
456         h=HBITS(h); \
457         mul64(l,h,(bl),(bh)); \
458  \
459         /* non-multiply part */ \
460         l=(l+(c))&BN_MASK2; if (l < (c)) h++; \
461         (c)=(r); \
462         l=(l+(c))&BN_MASK2; if (l < (c)) h++; \
463         (c)=h&BN_MASK2; \
464         (r)=l; \
465         }

467 #define mul(r,a,bl,bh,c) { \
468         BN_ULONG l,h; \
469  \
470         h= (a); \
471         l=LBITS(h); \
472         h=HBITS(h); \
473         mul64(l,h,(bl),(bh)); \
474  \
475         /* non-multiply part */ \
476         l+=(c); if ((l&BN_MASK2) < (c)) h++; \
477         (c)=h&BN_MASK2; \
478         (r)=l&BN_MASK2; \
479         }
480 #endif /* !BN_LLONG */

482 #if defined(OPENSSL_DOING_MAKEDEPEND) && defined(OPENSSL_FIPS)
483 #undef bn_div_words
484 #endif

486 void bn_mul_normal(BN_ULONG *r,BN_ULONG *a,int na,BN_ULONG *b,int nb);
487 void bn_mul_comba8(BN_ULONG *r,BN_ULONG *a,BN_ULONG *b);
488 void bn_mul_comba4(BN_ULONG *r,BN_ULONG *a,BN_ULONG *b);
489 void bn_sqr_normal(BN_ULONG *r, const BN_ULONG *a, int n, BN_ULONG *tmp);
490 void bn_sqr_comba8(BN_ULONG *r,const BN_ULONG *a);
491 void bn_sqr_comba4(BN_ULONG *r,const BN_ULONG *a);
492 int bn_cmp_words(const BN_ULONG *a,const BN_ULONG *b,int n);
493 int bn_cmp_part_words(const BN_ULONG *a, const BN_ULONG *b,
494         int cl, int dl);
495 void bn_mul_recursive(BN_ULONG *r,BN_ULONG *a,BN_ULONG *b,int n2,
496         int dna,int dnb,BN_ULONG *t);
497 void bn_mul_part_recursive(BN_ULONG *r,BN_ULONG *a,BN_ULONG *b,
498         int n,int tna,int tnb,BN_ULONG *t);
499 void bn_sqr_recursive(BN_ULONG *r,const BN_ULONG *a, int n2, BN_ULONG *t);
500 void bn_mul_low_normal(BN_ULONG *r,BN_ULONG *a,BN_ULONG *b, int n);
501 void bn_mul_low_recursive(BN_ULONG *r,BN_ULONG *a,BN_ULONG *b,int n2,
502         BN_ULONG *t);
503 void bn_mul_high(BN_ULONG *r,BN_ULONG *a,BN_ULONG *b,BN_ULONG *l,int n2,
504         BN_ULONG *t);
505 BN_ULONG bn_add_part_words(BN_ULONG *r, const BN_ULONG *a, const BN_ULONG *b,
506         int cl, int dl);
507 BN_ULONG bn_sub_part_words(BN_ULONG *r, const BN_ULONG *a, const BN_ULONG *b,
508         int cl, int dl);
509 int bn_mul_mont(BN_ULONG *rp, const BN_ULONG *ap, const BN_ULONG *bp, const BN_U

511 #ifdef  __cplusplus
512 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4432 Wed Aug 13 19:32:51 2014
new/usr/src/lib/openssl/include/cryptlib.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/cryptlib.h */
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */

#ifndef HEADER_CRYPTLIB_H
#define HEADER_CRYPTLIB_H
```



```
#include <stdlib.h>
#include <string.h>

#include "e_os.h"

#ifdef OPENSSL_USE_APPLINK
#define BIO_FLAGS_UPLINK 0x8000
#include "ms/uplink.h"
#endif

#include <openssl/crypto.h>
#include <openssl/buffer.h>
#include <openssl/bio.h>
#include <openssl/err.h>
#include <openssl/opensslconf.h>

#ifdef  __cplusplus
extern "C" {
#endif

#ifndef OPENSSL_SYS_VMS
#define X509_CERT_AREA          OPENSSLDIR
#define X509_CERT_DIR           OPENSSLDIR "/certs"
#define X509_CERT_FILE          OPENSSLDIR "/cert.pem"
#define X509_PRIVATE_DIR        OPENSSLDIR "/private"
#else
#define X509_CERT_AREA          "SSLROOT:[000000]"
#define X509_CERT_DIR           "SSLCERTS:"
#define X509_CERT_FILE          "SSLCERTS:cert.pem"
#define X509_PRIVATE_DIR        "SSLPRIVATE:"
#endif

#define X509_CERT_DIR_EVP        "SSL_CERT_DIR"
#define X509_CERT_FILE_EVP       "SSL_CERT_FILE"

/* size of string representations */
#define DECIMAL_SIZE(type)      ((sizeof(type)*8+2)/3+1)
#define HEX_SIZE(type)          (sizeof(type)*2)

void OPENSSL_cpuid_setup(void);
extern unsigned int OPENSSL_ia32cap_P[];
void OPENSSL_showfatal(const char *fmta,...);
void *OPENSSL_stderr(void);
extern int OPENSSL_NONPIC_relocated;

void illumos_locking_setup();

#endif /* ! codereview */
#ifdef  __cplusplus
}
#endif

#endif
```

```
**********************************************************
   13700 Wed Aug 13 19:32:52 2014
new/usr/src/lib/openssl/include/des_locl.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/des/des_locl.h */
/* Copyright (C) 1995-1997 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */

#ifndef HEADER_DES_LOCL_H
#define HEADER_DES_LOCL_H
```



```
#include <openssl/e_os2.h>

#if defined(OPENSSL_SYS_WIN32)
#ifndef OPENSSL_SYS_MSDOS
#define OPENSSL_SYS_MSDOS
#endif
#endif

#include <stdio.h>
#include <stdlib.h>

#ifndef OPENSSL_SYS_MSDOS
#if !defined(OPENSSL_SYS_VMS) || defined(__DECC)
#ifdef OPENSSL_UNISTD
# include OPENSSL_UNISTD
#else
# include <unistd.h>
#endif
#include <math.h>
#endif
#endif
#include <openssl/des.h>

#ifdef OPENSSL_SYS_MSDOS                /* Visual C++ 2.1 (Windows NT/95) */
#include <stdlib.h>
#include <errno.h>
#include <time.h>
#include <io.h>
#endif

#if defined(__STDC__) || defined(OPENSSL_SYS_VMS) || defined(M_XENIX) || defined
#include <string.h>
#endif

#ifdef OPENSSL_BUILD_SHLIBCRYPTO
# undef OPENSSL_EXTERN
# define OPENSSL_EXTERN OPENSSL_EXPORT
#endif

#define ITERATIONS 16
#define HALF_ITERATIONS 8

/* used in des_read and des_write */
#define MAXWRITE        (1024*16)
#define BSIZE           (MAXWRITE+4)

#define c2l(c,l)        (l =((DES_LONG)(*((c)++)))    , \
                         l|=((DES_LONG)(*((c)++)))<< 8L, \
                         l|=((DES_LONG)(*((c)++)))<<16L, \
                         l|=((DES_LONG)(*((c)++)))<<24L)

/* NOTE - c is not incremented as per c2l */
#define c2ln(c,l1,l2,n) { \
                        c+=n; \
                        l1=l2=0; \
                        switch (n) { \
                        case 8: l2 =((DES_LONG)(*(--(c))))<<24L; \
                        case 7: l2|=((DES_LONG)(*(--(c))))<<16L; \
                        case 6: l2|=((DES_LONG)(*(--(c))))<< 8L; \
                        case 5: l2|=((DES_LONG)(*(--(c))));     \
                        case 4: l1 =((DES_LONG)(*(--(c))))<<24L; \
                        case 3: l1|=((DES_LONG)(*(--(c))))<<16L; \
                        case 2: l1|=((DES_LONG)(*(--(c))))<< 8L; \
                        case 1: l1|=((DES_LONG)(*(--(c))));     \
                                } \
                        }
```

```
129 #define l2c(l,c)        (*((c)++)=(unsigned char)(((l)     )&0xff), \
130                          *((c)++)=(unsigned char)(((l)>> 8L)&0xff), \
131                          *((c)++)=(unsigned char)(((l)>>16L)&0xff), \
132                          *((c)++)=(unsigned char)(((l)>>24L)&0xff))

134 /* replacements for htonl and ntohl since I have no idea what to do
135  * when faced with machines with 8 byte longs. */
136 #define HDRSIZE 4

138 #define n2l(c,l)        (l =((DES_LONG)(*((c)++)))<<24L, \
139                          l|=((DES_LONG)(*((c)++)))<<16L, \
140                          l|=((DES_LONG)(*((c)++)))<< 8L, \
141                          l|=((DES_LONG)(*((c)++))))

143 #define l2n(l,c)        (*((c)++)=(unsigned char)(((l)>>24L)&0xff), \
144                          *((c)++)=(unsigned char)(((l)>>16L)&0xff), \
145                          *((c)++)=(unsigned char)(((l)>> 8L)&0xff), \
146                          *((c)++)=(unsigned char)(((l)     )&0xff))

148 /* NOTE - c is not incremented as per l2c */
149 #define l2cn(l1,l2,c,n) { \
150                         c+=n; \
151                         switch (n) { \
152                         case 8: *(--(c))=(unsigned char)(((l2)>>24L)&0xff); \
153                         case 7: *(--(c))=(unsigned char)(((l2)>>16L)&0xff); \
154                         case 6: *(--(c))=(unsigned char)(((l2)>> 8L)&0xff); \
155                         case 5: *(--(c))=(unsigned char)(((l2)     )&0xff); \
156                         case 4: *(--(c))=(unsigned char)(((l1)>>24L)&0xff); \
157                         case 3: *(--(c))=(unsigned char)(((l1)>>16L)&0xff); \
158                         case 2: *(--(c))=(unsigned char)(((l1)>> 8L)&0xff); \
159                         case 1: *(--(c))=(unsigned char)(((l1)     )&0xff); \
160                                 } \
161                         }

163 #if (defined(OPENSSL_SYS_WIN32) && defined(_MSC_VER)) || defined(__ICC)
164 #define ROTATE(a,n)     (_lrotr(a,n))
165 #elif defined(__GNUC__) && __GNUC__>=2 && !defined(__STRICT_ANSI__) && !defined(
166 # if defined(__i386) || defined(__i386__) || defined(__x86_64) || defined(__x86_
167 #  define ROTATE(a,n)   ({ register unsigned int ret;   \
168                                 __asm__ ("rorl %1,%0"   \
168                                 asm ("rorl %1,%0"       \
169                                         : "=r"(ret)     \
170                                         : "I"(n),"0"(a) \
171                                         : "cc");        \
172                            ret;                         \
173                         })
174 # endif
175 #endif
176 #ifndef ROTATE
177 #define ROTATE(a,n)     (((a)>>(n))+((a)<<(32-(n))))
178 #endif

180 /* Don't worry about the LOAD_DATA() stuff, that is used by
181  * fcrypt() to add it's little bit to the front */

183 #ifdef DES_FCRYPT

185 #define LOAD_DATA_tmp(R,S,u,t,E0,E1) \
186         { DES_LONG tmp; LOAD_DATA(R,S,u,t,E0,E1,tmp); }

188 #define LOAD_DATA(R,S,u,t,E0,E1,tmp) \
189         t=R^(R>>16L); \
190         u=t&E0; t&=E1; \
191         tmp=(u<<16); u^=R^s[S  ]; u^=tmp; \
192         tmp=(t<<16); t^=R^s[S+1]; t^=tmp
```



```
193 #else
194 #define LOAD_DATA_tmp(a,b,c,d,e,f) LOAD_DATA(a,b,c,d,e,f,g)
195 #define LOAD_DATA(R,S,u,t,E0,E1,tmp) \
196         u=R^s[S  ]; \
197         t=R^s[S+1]
198 #endif

200 /* The changes to this macro may help or hinder, depending on the
201  * compiler and the architecture.  gcc2 always seems to do well :-).
202  * Inspired by Dana How <how@isl.stanford.edu>
203  * DO NOT use the alternative version on machines with 8 byte longs.
204  * It does not seem to work on the Alpha, even when DES_LONG is 4
205  * bytes, probably an issue of accessing non-word aligned objects :-( */
206 #ifdef DES_PTR

208 /* It recently occurred to me that 0^0^0^0^0^0^0 == 0, so there
209  * is no reason to not xor all the sub items together.  This potentially
210  * saves a register since things can be xored directly into L */

212 #if defined(DES_RISC1) || defined(DES_RISC2)
213 #ifdef DES_RISC1
214 #define D_ENCRYPT(LL,R,S) { \
215         unsigned int u1,u2,u3; \
216         LOAD_DATA(R,S,u,t,E0,E1,u1); \
217         u2=(int)u>>8L; \
218         u1=(int)u&0xfc; \
219         u2&=0xfc; \
220         t=ROTATE(t,4); \
221         u>>=16L; \
222         LL^= *(const DES_LONG *)(des_SP      +u1); \
223         LL^= *(const DES_LONG *)(des_SP+0x200+u2); \
224         u3=(int)(u>>8L); \
225         u1=(int)u&0xfc; \
226         u3&=0xfc; \
227         LL^= *(const DES_LONG *)(des_SP+0x400+u1); \
228         LL^= *(const DES_LONG *)(des_SP+0x600+u3); \
229         u2=(int)t>>8L; \
230         u1=(int)t&0xfc; \
231         u2&=0xfc; \
232         t>>=16L; \
233         LL^= *(const DES_LONG *)(des_SP+0x100+u1); \
234         LL^= *(const DES_LONG *)(des_SP+0x300+u2); \
235         u3=(int)t>>8L; \
236         u1=(int)t&0xfc; \
237         u3&=0xfc; \
238         LL^= *(const DES_LONG *)(des_SP+0x500+u1); \
239         LL^= *(const DES_LONG *)(des_SP+0x700+u3); }
240 #endif
241 #ifdef DES_RISC2
242 #define D_ENCRYPT(LL,R,S) { \
243         unsigned int u1,u2,s1,s2; \
244         LOAD_DATA(R,S,u,t,E0,E1,u1); \
245         u2=(int)u>>8L; \
246         u1=(int)u&0xfc; \
247         u2&=0xfc; \
248         t=ROTATE(t,4); \
249         LL^= *(const DES_LONG *)(des_SP      +u1); \
250         LL^= *(const DES_LONG *)(des_SP+0x200+u2); \
251         s1=(int)(u>>16L); \
252         s2=(int)(u>>24L); \
253         s1&=0xfc; \
254         s2&=0xfc; \
255         LL^= *(const DES_LONG *)(des_SP+0x400+s1); \
256         LL^= *(const DES_LONG *)(des_SP+0x600+s2); \
257         u2=(int)t>>8L; \
258         u1=(int)t&0xfc; \
```

```
259         u2&=0xfc; \
260         LL^= *(const DES_LONG *)(des_SP+0x100+u1); \
261         LL^= *(const DES_LONG *)(des_SP+0x300+u2); \
262         s1=(int)(t>>16L); \
263         s2=(int)(t>>24L); \
264         s1&=0xfc; \
265         s2&=0xfc; \
266         LL^= *(const DES_LONG *)(des_SP+0x500+s1); \
267         LL^= *(const DES_LONG *)(des_SP+0x700+s2); }
268 #endif
269 #else
270 #define D_ENCRYPT(LL,R,S) { \
271         LOAD_DATA_tmp(R,S,u,t,E0,E1); \
272         t=ROTATE(t,4); \
273         LL^= \
274         *(const DES_LONG *)(des_SP      +((u     )&0xfc))^ \
275         *(const DES_LONG *)(des_SP+0x200+((u>> 8L)&0xfc))^ \
276         *(const DES_LONG *)(des_SP+0x400+((u>>16L)&0xfc))^ \
277         *(const DES_LONG *)(des_SP+0x600+((u>>24L)&0xfc))^ \
278         *(const DES_LONG *)(des_SP+0x100+((t     )&0xfc))^ \
279         *(const DES_LONG *)(des_SP+0x300+((t>> 8L)&0xfc))^ \
280         *(const DES_LONG *)(des_SP+0x500+((t>>16L)&0xfc))^ \
281         *(const DES_LONG *)(des_SP+0x700+((t>>24L)&0xfc)); }
282 #endif

284 #else /* original version */

286 #if defined(DES_RISC1) || defined(DES_RISC2)
287 #ifdef DES_RISC1
288 #define D_ENCRYPT(LL,R,S) {\
289         unsigned int u1,u2,u3; \
290         LOAD_DATA(R,S,u,t,E0,E1,u1); \
291         u>>=2L; \
292         t=ROTATE(t,6); \
293         u2=(int)u>>8L; \
294         u1=(int)u&0x3f; \
295         u2&=0x3f; \
296         u>>=16L; \
297         LL^=DES_SPtrans[0][u1]; \
298         LL^=DES_SPtrans[2][u2]; \
299         u3=(int)u>>8L; \
300         u1=(int)u&0x3f; \
301         u3&=0x3f; \
302         LL^=DES_SPtrans[4][u1]; \
303         LL^=DES_SPtrans[6][u3]; \
304         u2=(int)t>>8L; \
305         u1=(int)t&0x3f; \
306         u2&=0x3f; \
307         t>>=16L; \
308         LL^=DES_SPtrans[1][u1]; \
309         LL^=DES_SPtrans[3][u2]; \
310         u3=(int)t>>8L; \
311         u1=(int)t&0x3f; \
312         u3&=0x3f; \
313         LL^=DES_SPtrans[5][u1]; \
314         LL^=DES_SPtrans[7][u3]; }
315 #endif
316 #ifdef DES_RISC2
317 #define D_ENCRYPT(LL,R,S) {\
318         unsigned int u1,u2,s1,s2; \
319         LOAD_DATA(R,S,u,t,E0,E1,u1); \
320         u>>=2L; \
321         t=ROTATE(t,6); \
322         u2=(int)u>>8L; \
323         u1=(int)u&0x3f; \
324         u2&=0x3f; \
```



```
325         LL^=DES_SPtrans[0][u1]; \
326         LL^=DES_SPtrans[2][u2]; \
327         s1=(int)u>>16L; \
328         s2=(int)u>>24L; \
329         s1&=0x3f; \
330         s2&=0x3f; \
331         LL^=DES_SPtrans[4][s1]; \
332         LL^=DES_SPtrans[6][s2]; \
333         u2=(int)t>>8L; \
334         u1=(int)t&0x3f; \
335         u2&=0x3f; \
336         LL^=DES_SPtrans[1][u1]; \
337         LL^=DES_SPtrans[3][u2]; \
338         s1=(int)t>>16; \
339         s2=(int)t>>24L; \
340         s1&=0x3f; \
341         s2&=0x3f; \
342         LL^=DES_SPtrans[5][s1]; \
343         LL^=DES_SPtrans[7][s2]; }
344 #endif

346 #else

348 #define D_ENCRYPT(LL,R,S) {\
349         LOAD_DATA_tmp(R,S,u,t,E0,E1); \
350         t=ROTATE(t,4); \
351         LL^=\
352                 DES_SPtrans[0][(u>> 2L)&0x3f]^ \
353                 DES_SPtrans[2][(u>>10L)&0x3f]^ \
354                 DES_SPtrans[4][(u>>18L)&0x3f]^ \
355                 DES_SPtrans[6][(u>>26L)&0x3f]^ \
356                 DES_SPtrans[1][(t>> 2L)&0x3f]^ \
357                 DES_SPtrans[3][(t>>10L)&0x3f]^ \
358                 DES_SPtrans[5][(t>>18L)&0x3f]^ \
359                 DES_SPtrans[7][(t>>26L)&0x3f]; }
360 #endif
361 #endif

363         /* IP and FP
364          * The problem is more of a geometric problem that random bit fiddling.
365          0  1  2  3  4  5  6  7      62 54 46 38 30 22 14  6
366          8  9 10 11 12 13 14 15      60 52 44 36 28 20 12  4
367         16 17 18 19 20 21 22 23      58 50 42 34 26 18 10  2
368         24 25 26 27 28 29 30 31  to  56 48 40 32 24 16  8  0

370         32 33 34 35 36 37 38 39      63 55 47 39 31 23 15  7
371         40 41 42 43 44 45 46 47      61 53 45 37 29 21 13  5
372         48 49 50 51 52 53 54 55      59 51 43 35 27 19 11  3
373         56 57 58 59 60 61 62 63      57 49 41 33 25 17  9  1

375         The output has been subject to swaps of the form
376         0 1 -> 3 1 but the odd and even bits have been put into
377         2 3    2 0
378         different words.  The main trick is to remember that
379         t=((l>>size)^r)&(mask);
380         r^=t;
381         l^=(t<<size);
382         can be used to swap and move bits between words.

384         So l =  0  1  2  3  r = 16 17 18 19
385                 4  5  6  7      20 21 22 23
386                 8  9 10 11      24 25 26 27
387                12 13 14 15      28 29 30 31
388         becomes (for size == 2 and mask == 0x3333)
389            t =   2^16  3^17 -- --   l =  0  1 16 17  r =  2  3 18 19
390                  6^20  7^21 -- --        4  5 20 21       6  7 22 23
```

```
391                 10^24 11^25 -- --        8  9 24 25      10 11 24 25
392                 14^28 15^29 -- --       12 13 28 29      14 15 28 29

394         Thanks for hints from Richard Outerbridge - he told me IP&FP
395         could be done in 15 xor, 10 shifts and 5 ands.
396         When I finally started to think of the problem in 2D
397         I first got ~42 operations without xors.  When I remembered
398         how to use xors :-) I got it to its final state.
399         */
400 #define PERM_OP(a,b,t,n,m) ((t)=((((a)>>(n))^(b))&(m)),\
401         (b)^=(t),\
402         (a)^=((t)<<(n)))

404 #define IP(l,r) \
405         { \
406         register DES_LONG tt; \
407         PERM_OP(r,l,tt, 4,0x0f0f0f0fL); \
408         PERM_OP(l,r,tt,16,0x0000ffffL); \
409         PERM_OP(r,l,tt, 2,0x33333333L); \
410         PERM_OP(l,r,tt, 8,0x00ff00ffL); \
411         PERM_OP(r,l,tt, 1,0x55555555L); \
412         }

414 #define FP(l,r) \
415         { \
416         register DES_LONG tt; \
417         PERM_OP(l,r,tt, 1,0x55555555L); \
418         PERM_OP(r,l,tt, 8,0x00ff00ffL); \
419         PERM_OP(l,r,tt, 2,0x33333333L); \
420         PERM_OP(r,l,tt,16,0x0000ffffL); \
421         PERM_OP(l,r,tt, 4,0x0f0f0f0fL); \
422         }

424 extern const DES_LONG DES_SPtrans[8][64];

426 void fcrypt_body(DES_LONG *out,DES_key_schedule *ks,
427                  DES_LONG Eswap0, DES_LONG Eswap1);

429 #ifdef OPENSSL_SMALL_FOOTPRINT
430 #undef DES_UNROLL
431 #endif
432 #endif
```

```
**********************************************************
   12520 Wed Aug 13 19:32:52 2014
new/usr/src/lib/openssl/include/md32_common.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/md32_common.h */
/* ====================================================================
 * Copyright (c) 1999-2007 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 */

/*
 * This is a generic 32 bit "collector" for message digest algorithms.
 * Whenever needed it collects input character stream into chunks of
 * 32 bit values and invokes a block function that performs actual hash
 * calculations.
 *
 * Porting guide.
 *
 * Obligatory macros:
 *
```



```
 * DATA_ORDER_IS_BIG_ENDIAN or DATA_ORDER_IS_LITTLE_ENDIAN
 *      this macro defines byte order of input stream.
 * HASH_CBLOCK
 *      size of a unit chunk HASH_BLOCK operates on.
 * HASH_LONG
 *      has to be at lest 32 bit wide, if it's wider, then
 *      HASH_LONG_LOG2 *has to* be defined along
 * HASH_CTX
 *      context structure that at least contains following
 *      members:
 *              typedef struct {
 *                      ...
 *                      HASH_LONG       Nl,Nh;
 *                      either {
 *                      HASH_LONG       data[HASH_LBLOCK];
 *                      unsigned char   data[HASH_CBLOCK];
 *                      };
 *                      unsigned int    num;
 *                      ...
 *                      } HASH_CTX;
 *      data[] vector is expected to be zeroed upon first call to
 *      HASH_UPDATE.
 * HASH_UPDATE
 *      name of "Update" function, implemented here.
 * HASH_TRANSFORM
 *      name of "Transform" function, implemented here.
 * HASH_FINAL
 *      name of "Final" function, implemented here.
 * HASH_BLOCK_DATA_ORDER
 *      name of "block" function capable of treating *unaligned* input
 *      message in original (data) byte order, implemented externally.
 * HASH_MAKE_STRING
 *      macro convering context variables to an ASCII hash string.
 *
 * MD5 example:
 *
 *      #define DATA_ORDER_IS_LITTLE_ENDIAN
 *
 *      #define HASH_LONG               MD5_LONG
 *      #define HASH_LONG_LOG2          MD5_LONG_LOG2
 *      #define HASH_CTX                MD5_CTX
 *      #define HASH_CBLOCK             MD5_CBLOCK
 *      #define HASH_UPDATE             MD5_Update
 *      #define HASH_TRANSFORM          MD5_Transform
 *      #define HASH_FINAL              MD5_Final
 *      #define HASH_BLOCK_DATA_ORDER   md5_block_data_order
 *
 *                                      <appro@fy.chalmers.se>
 */

#if !defined(DATA_ORDER_IS_BIG_ENDIAN) && !defined(DATA_ORDER_IS_LITTLE_ENDIAN)
#error "DATA_ORDER must be defined!"
#endif

#ifndef HASH_CBLOCK
#error "HASH_CBLOCK must be defined!"
#endif
#ifndef HASH_LONG
#error "HASH_LONG must be defined!"
#endif
#ifndef HASH_CTX
#error "HASH_CTX must be defined!"
#endif

#ifndef HASH_UPDATE
#error "HASH_UPDATE must be defined!"
```

```
#endif
#ifndef HASH_TRANSFORM
#error "HASH_TRANSFORM must be defined!"
#endif
#ifndef HASH_FINAL
#error "HASH_FINAL must be defined!"
#endif

#ifndef HASH_BLOCK_DATA_ORDER
#error "HASH_BLOCK_DATA_ORDER must be defined!"
#endif

/*
 * Engage compiler specific rotate intrinsic function if available.
 */
#undef ROTATE
#ifndef PEDANTIC
# if defined(_MSC_VER) || defined(__ICC)
#  define ROTATE(a,n)   _lrotl(a,n)
# elif defined(__MWERKS__)
#  if defined(__POWERPC__)
#   define ROTATE(a,n)  __rlwinm(a,n,0,31)
#  elif defined(__MC68K__)
    /* Motorola specific tweak. <appro@fy.chalmers.se> */
#   define ROTATE(a,n)  ( n<24 ? __rol(a,n) : __ror(a,32-n) )
#  else
#   define ROTATE(a,n)  __rol(a,n)
#  endif
# elif defined(__GNUC__) && __GNUC__>=2 && !defined(OPENSSL_NO_ASM) && !defined(
  /*
   * Some GNU C inline assembler templates. Note that these are
   * rotates by *constant* number of bits! But that's exactly
   * what we need here...
   *                                    <appro@fy.chalmers.se>
   */
#  if defined(__i386) || defined(__i386__) || defined(__x86_64) || defined(__x86
#   define ROTATE(a,n)  ({ register unsigned int ret;   \
                                __asm__ (                       \
                                asm (                   \
                                "roll %1,%0"            \
                                : "=r"(ret)             \
                                : "I"(n), "0"((unsigned int)(a))        \
                                : "cc");                \
                            ret;                        \
                         })
#  elif defined(_ARCH_PPC) || defined(_ARCH_PPC64) || \
        defined(__powerpc) || defined(__ppc__) || defined(__powerpc64__)
#   define ROTATE(a,n)  ({ register unsigned int ret;   \
                                __asm__ (                       \
                                asm (                   \
                                "rlwinm %0,%1,%2,0,31"  \
                                : "=r"(ret)             \
                                : "r"(a), "I"(n));      \
                            ret;                        \
                         })
#  elif defined(__s390x__)
#   define ROTATE(a,n) ({ register unsigned int ret;    \
                                __asm__ ("rll %0,%1,%2" \
                                asm ("rll %0,%1,%2"     \
                                : "=r"(ret)             \
                                : "r"(a), "I"(n));      \
                              ret;                      \
                         })
#  endif
# endif
#endif /* PEDANTIC */
```



```
#ifndef ROTATE
#define ROTATE(a,n)     (((a)<<(n))|(((a)&0xffffffff)>>(32-(n))))
#endif

#if defined(DATA_ORDER_IS_BIG_ENDIAN)

#ifndef PEDANTIC
# if defined(__GNUC__) && __GNUC__>=2 && !defined(OPENSSL_NO_ASM) && !defined(OP
#  if ((defined(__i386) || defined(__i386__)) && !defined(I386_ONLY)) || \
      (defined(__x86_64) || defined(__x86_64__))
#   if !defined(B_ENDIAN)
    /*
     * This gives ~30-40% performance improvement in SHA-256 compiled
     * with gcc [on P4]. Well, first macro to be frank. We can pull
     * this trick on x86* platforms only, because these CPUs can fetch
     * unaligned data without raising an exception.
     */
#   define HOST_c2l(c,l)        ({ unsigned int r=*((const unsigned int *)(c));
                                   __asm__ ("bswapl %0":"=r"(r):"0"(r));
                                   asm ("bswapl %0":"=r"(r):"0"(r));    \
                                   (c)+=4; (l)=r;                       })
#   define HOST_l2c(l,c)        ({ unsigned int r=(l);                  \
                                   __asm__ ("bswapl %0":"=r"(r):"0"(r));
                                   asm ("bswapl %0":"=r"(r):"0"(r));    \
                                   *((unsigned int *)(c))=r; (c)+=4; r; })
#   endif
#  endif
# endif
#endif
#if defined(__s390__) || defined(__s390x__)
# define HOST_c2l(c,l) ((l)=*((const unsigned int *)(c)), (c)+=4, (l))
# define HOST_l2c(l,c) (*((unsigned int *)(c))=(l), (c)+=4, (l))
#endif

#ifndef HOST_c2l
#define HOST_c2l(c,l)   (l =(((unsigned long)(*((c)++)))<<24),          \
                         l|=(((unsigned long)(*((c)++)))<<16),          \
                         l|=(((unsigned long)(*((c)++)))<< 8),          \
                         l|=(((unsigned long)(*((c)++)))    ),          \
                         l)
#endif
#ifndef HOST_l2c
#define HOST_l2c(l,c)   (*((c)++)=(unsigned char)(((l)>>24)&0xff),      \
                         *((c)++)=(unsigned char)(((l)>>16)&0xff),      \
                         *((c)++)=(unsigned char)(((l)>> 8)&0xff),      \
                         *((c)++)=(unsigned char)(((l)    )&0xff),      \
                         l)
#endif

#elif defined(DATA_ORDER_IS_LITTLE_ENDIAN)

#ifndef PEDANTIC
# if defined(__GNUC__) && __GNUC__>=2 && !defined(OPENSSL_NO_ASM) && !defined(OP
#  if defined(__s390x__)
#   define HOST_c2l(c,l)        ({ __asm__ ("lrv        %0,%1"
#   define HOST_c2l(c,l)        ({ asm ("lrv    %0,%1"                  \
                                   :"=d"(l) :"m"(*(const unsigned int *)(c)));\
                                   (c)+=4; (l);                         })
#   define HOST_l2c(l,c)        ({ __asm__ ("strv       %1,%0"
#   define HOST_l2c(l,c)        ({ asm ("strv   %1,%0"                  \
                                   :"=m"(*(unsigned int *)(c)) :"d"(l));\
                                   (c)+=4; (l);                         })
#  endif
# endif
#endif
```

```
253 #if defined(__i386) || defined(__i386__) || defined(__x86_64) || defined(__x86_6
254 # ifndef B_ENDIAN
255    /* See comment in DATA_ORDER_IS_BIG_ENDIAN section. */
256 #  define HOST_c2l(c,l) ((l)=*((const unsigned int *)(c)), (c)+=4, l)
257 #  define HOST_l2c(l,c) (*((unsigned int *)(c))=(l), (c)+=4, l)
258 # endif
259 #endif

261 #ifndef HOST_c2l
262 #define HOST_c2l(c,l)   (l =(((unsigned long)(*((c)++)))    ),          \
263                          l|=(((unsigned long)(*((c)++)))<< 8),          \
264                          l|=(((unsigned long)(*((c)++)))<<16),          \
265                          l|=(((unsigned long)(*((c)++)))<<24),          \
266                          l)
267 #endif
268 #ifndef HOST_l2c
269 #define HOST_l2c(l,c)   (*((c)++)=(unsigned char)(((l)    )&0xff),      \
270                          *((c)++)=(unsigned char)(((l)>> 8)&0xff),      \
271                          *((c)++)=(unsigned char)(((l)>>16)&0xff),      \
272                          *((c)++)=(unsigned char)(((l)>>24)&0xff),      \
273                          l)
274 #endif

276 #endif

278 /*
279  * Time for some action:-)
280  */

282 int HASH_UPDATE (HASH_CTX *c, const void *data_, size_t len)
283         {
284         const unsigned char *data=data_;
285         unsigned char *p;
286         HASH_LONG l;
287         size_t n;

289         if (len==0) return 1;

291         l=(c->Nl+(((HASH_LONG)len)<<3))&0xffffffffUL;
292         /* 95-05-24 eay Fixed a bug with the overflow handling, thanks to
293          * Wei Dai <weidai@eskimo.com> for pointing it out. */
294         if (l < c->Nl) /* overflow */
295                 c->Nh++;
296         c->Nh+=(HASH_LONG)(len>>29);    /* might cause compiler warning on 16-bi
297         c->Nl=l;

299         n = c->num;
300         if (n != 0)
301                 {
302                 p=(unsigned char *)c->data;

304                 if (len >= HASH_CBLOCK || len+n >= HASH_CBLOCK)
305                         {
306                         memcpy (p+n,data,HASH_CBLOCK-n);
307                         HASH_BLOCK_DATA_ORDER (c,p,1);
308                         n      = HASH_CBLOCK-n;
309                         data  += n;
310                         len   -= n;
311                         c->num = 0;
312                         memset (p,0,HASH_CBLOCK);       /* keep it zeroed */
313                         }
314                 else
315                         {
316                         memcpy (p+n,data,len);
317                         c->num += (unsigned int)len;
318                         return 1;
```



```
319                         }
320                 }

322         n = len/HASH_CBLOCK;
323         if (n > 0)
324                 {
325                 HASH_BLOCK_DATA_ORDER (c,data,n);
326                 n    *= HASH_CBLOCK;
327                 data += n;
328                 len  -= n;
329                 }

331         if (len != 0)
332                 {
333                 p = (unsigned char *)c->data;
334                 c->num = (unsigned int)len;
335                 memcpy (p,data,len);
336                 }
337         return 1;
338         }


341 void HASH_TRANSFORM (HASH_CTX *c, const unsigned char *data)
342         {
343         HASH_BLOCK_DATA_ORDER (c,data,1);
344         }


347 int HASH_FINAL (unsigned char *md, HASH_CTX *c)
348         {
349         unsigned char *p = (unsigned char *)c->data;
350         size_t n = c->num;

352         p[n] = 0x80; /* there is always room for one */
353         n++;

355         if (n > (HASH_CBLOCK-8))
356                 {
357                 memset (p+n,0,HASH_CBLOCK-n);
358                 n=0;
359                 HASH_BLOCK_DATA_ORDER (c,p,1);
360                 }
361         memset (p+n,0,HASH_CBLOCK-8-n);

363         p += HASH_CBLOCK-8;
364 #if   defined(DATA_ORDER_IS_BIG_ENDIAN)
365         (void)HOST_l2c(c->Nh,p);
366         (void)HOST_l2c(c->Nl,p);
367 #elif defined(DATA_ORDER_IS_LITTLE_ENDIAN)
368         (void)HOST_l2c(c->Nl,p);
369         (void)HOST_l2c(c->Nh,p);
370 #endif
371         p -= HASH_CBLOCK;
372         HASH_BLOCK_DATA_ORDER (c,p,1);
373         c->num=0;
374         memset (p,0,HASH_CBLOCK);

376 #ifndef HASH_MAKE_STRING
377 #error "HASH_MAKE_STRING must be defined!"
378 #else
379         HASH_MAKE_STRING(c,md);
380 #endif

382         return 1;
383         }
```

```
385 #ifndef MD32_REG_T
386 #if defined(__alpha) || defined(__sparcv9) || defined(__mips)
387 #define MD32_REG_T long
388 /*
389  * This comment was originaly written for MD5, which is why it
390  * discusses A-D. But it basically applies to all 32-bit digests,
391  * which is why it was moved to common header file.
392  *
393  * In case you wonder why A-D are declared as long and not
394  * as MD5_LONG. Doing so results in slight performance
395  * boost on LP64 architectures. The catch is we don't
396  * really care if 32 MSBs of a 64-bit register get polluted
397  * with eventual overflows as we *save* only 32 LSBs in
398  * *either* case. Now declaring 'em long excuses the compiler
399  * from keeping 32 MSBs zeroed resulting in 13% performance
400  * improvement under SPARC Solaris7/64 and 5% under AlphaLinux.
401  * Well, to be honest it should say that this *prevents*
402  * performance degradation.
403  *                              <appro@fy.chalmers.se>
404  */
405 #else
406 /*
407  * Above is not absolute and there are LP64 compilers that
408  * generate better code if MD32_REG_T is defined int. The above
409  * pre-processor condition reflects the circumstances under which
410  * the conclusion was made and is subject to further extension.
411  *                              <appro@fy.chalmers.se>
412  */
413 #define MD32_REG_T int
414 #endif
415 #endif
```

```
**********************************************************
    3636 Wed Aug 13 19:32:53 2014
new/usr/src/lib/openssl/include/modes_lcl.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* ====================================================================
 * Copyright (c) 2010 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use is governed by OpenSSL license.
 * ====================================================================
 */

#include <openssl/modes.h>


#if (defined(_WIN32) || defined(_WIN64)) && !defined(__MINGW32__)
typedef __int64 i64;
typedef unsigned __int64 u64;
#define U64(C) C##UI64
#elif defined(__arch64__)
typedef long i64;
typedef unsigned long u64;
#define U64(C) C##UL
#else
typedef long long i64;
typedef unsigned long long u64;
#define U64(C) C##ULL
#endif

typedef unsigned int u32;
typedef unsigned char u8;

#define STRICT_ALIGNMENT 1
#if defined(__i386)     || defined(__i386__)    || \
    defined(__x86_64)   || defined(__x86_64__)  || \
    defined(_M_IX86)    || defined(_M_AMD64)    || defined(_M_X64) || \
    defined(__s390__)   || defined(__s390x__)
# undef STRICT_ALIGNMENT
#endif

#if !defined(PEDANTIC) && !defined(OPENSSL_NO_ASM) && !defined(OPENSSL_NO_INLINE
#if defined(__GNUC__) && __GNUC__>=2
# if defined(__x86_64) || defined(__x86_64__)
#  define BSWAP8(x) ({  u64 ret=(x);                    \
                        __asm__ ("bswapq %0"            \
                        asm ("bswapq %0"                \
                        : "+r"(ret));   ret;            })
#  define BSWAP4(x) ({  u32 ret=(x);                    \
                        __asm__ ("bswapl %0"            \
                        asm ("bswapl %0"                \
                        : "+r"(ret));   ret;            })
# elif (defined(__i386) || defined(__i386__)) && !defined(I386_ONLY)
#  define BSWAP8(x) ({  u32 lo=(u64)(x)>>32,hi=(x);     \
                        __asm__ ("bswapl %0; bswapl %1" \
                        asm ("bswapl %0; bswapl %1"     \
                        : "+r"(hi),"+r"(lo));           \
                        (u64)hi<<32|lo;                 })
#  define BSWAP4(x) ({  u32 ret=(x);                    \
                        __asm__ ("bswapl %0"            \
                        asm ("bswapl %0"                \
                        : "+r"(ret));   ret;            })
# elif (defined(__arm__) || defined(__arm)) && !defined(STRICT_ALIGNMENT)
#  define BSWAP8(x) ({  u32 lo=(u64)(x)>>32,hi=(x);     \
                        __asm__ ("rev %0,%0; rev %1,%1" \
                        asm ("rev %0,%0; rev %1,%1"     \
                        : "+r"(hi),"+r"(lo));           \
```



```
                        (u64)hi<<32|lo;                 })
#  define BSWAP4(x) ({  u32 ret;                        \
                        __asm__ ("rev %0,%1"            \
                        asm ("rev %0,%1"                \
                        : "=r"(ret) : "r"((u32)(x)));   \
                        ret;                            })
# endif
#elif defined(_MSC_VER)
# if _MSC_VER>=1300
#  pragma intrinsic(_byteswap_uint64,_byteswap_ulong)
#  define BSWAP8(x)     _byteswap_uint64((u64)(x))
#  define BSWAP4(x)     _byteswap_ulong((u32)(x))
# elif defined(_M_IX86)
   __inline u32 _bswap4(u32 val) {
        _asm mov eax,val
        _asm bswap eax
   }
#  define BSWAP4(x)     _bswap4(x)
# endif
#endif
#endif

#if defined(BSWAP4) && !defined(STRICT_ALIGNMENT)
#define GETU32(p)       BSWAP4(*(const u32 *)(p))
#define PUTU32(p,v)     *(u32 *)(p) = BSWAP4(v)
#else
#define GETU32(p)       ((u32)(p)[0]<<24|(u32)(p)[1]<<16|(u32)(p)[2]<<8|(u32)(p)
#define PUTU32(p,v)     ((p)[0]=(u8)((v)>>24),(p)[1]=(u8)((v)>>16),(p)[2]=(u8)((
#endif

/* GCM definitions */

typedef struct { u64 hi,lo; } u128;

#ifdef  TABLE_BITS
#undef  TABLE_BITS
#endif
/*
 * Even though permitted values for TABLE_BITS are 8, 4 and 1, it should
 * never be set to 8 [or 1]. For further information see gcm128.c.
 */
#define TABLE_BITS 4

struct gcm128_context {
        /* Following 6 names follow names in GCM specification */
        union { u64 u[2]; u32 d[4]; u8 c[16]; size_t t[16/sizeof(size_t)]; }
          Yi,EKi,EK0,len,Xi,H;
        /* Relative position of Xi, H and pre-computed Htable is used
         * in some assembler modules, i.e. don't change the order! */
#if TABLE_BITS==8
        u128 Htable[256];
#else
        u128 Htable[16];
        void (*gmult)(u64 Xi[2],const u128 Htable[16]);
        void (*ghash)(u64 Xi[2],const u128 Htable[16],const u8 *inp,size_t len);
#endif
        unsigned int mres, ares;
        block128_f block;
        void *key;
};
```

*unchanged_portion_omitted_*

```
**********************************************************
   40611 Wed Aug 13 19:32:53 2014
new/usr/src/lib/openssl/include/openssl/engine.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* openssl/engine.h */
/* Written by Geoff Thorpe (geoff@geoffthorpe.net) for the OpenSSL
 * project 2000.
 */
/* ====================================================================
 * Copyright (c) 1999-2004 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */
/* ====================================================================
 * Copyright 2002 Sun Microsystems, Inc. ALL RIGHTS RESERVED.
 * ECDH support in OpenSSL originally developed by
 * SUN MICROSYSTEMS, INC., and contributed to the OpenSSL project.
```



```
 */

#ifndef HEADER_ENGINE_H
#define HEADER_ENGINE_H

#include <openssl/opensslconf.h>

#ifdef OPENSSL_NO_ENGINE
#error ENGINE is disabled.
#endif

#ifndef OPENSSL_NO_DEPRECATED
#include <openssl/bn.h>
#ifndef OPENSSL_NO_RSA
#include <openssl/rsa.h>
#endif
#ifndef OPENSSL_NO_DSA
#include <openssl/dsa.h>
#endif
#ifndef OPENSSL_NO_DH
#include <openssl/dh.h>
#endif
#ifndef OPENSSL_NO_ECDH
#include <openssl/ecdh.h>
#endif
#ifndef OPENSSL_NO_ECDSA
#include <openssl/ecdsa.h>
#endif
#include <openssl/rand.h>
#include <openssl/ui.h>
#include <openssl/err.h>
#endif

#include <openssl/ossl_typ.h>
#include <openssl/symhacks.h>

#include <openssl/x509.h>

#ifdef  __cplusplus
extern "C" {
#endif

/* These flags are used to control combinations of algorithm (methods)
 * by bitwise "OR"ing. */
#define ENGINE_METHOD_RSA               (unsigned int)0x0001
#define ENGINE_METHOD_DSA               (unsigned int)0x0002
#define ENGINE_METHOD_DH                (unsigned int)0x0004
#define ENGINE_METHOD_RAND              (unsigned int)0x0008
#define ENGINE_METHOD_ECDH              (unsigned int)0x0010
#define ENGINE_METHOD_ECDSA             (unsigned int)0x0020
#define ENGINE_METHOD_CIPHERS           (unsigned int)0x0040
#define ENGINE_METHOD_DIGESTS           (unsigned int)0x0080
#define ENGINE_METHOD_STORE             (unsigned int)0x0100
#define ENGINE_METHOD_PKEY_METHS        (unsigned int)0x0200
#define ENGINE_METHOD_PKEY_ASN1_METHS   (unsigned int)0x0400
/* Obvious all-or-nothing cases. */
#define ENGINE_METHOD_ALL               (unsigned int)0xFFFF
#define ENGINE_METHOD_NONE              (unsigned int)0x0000

/* This(ese) flag(s) controls behaviour of the ENGINE_TABLE mechanism used
 * internally to control registration of ENGINE implementations, and can be set
 * by ENGINE_set_table_flags(). The "NOINIT" flag prevents attempts to
 * initialise registered ENGINEs if they are not already initialised. */
#define ENGINE_TABLE_FLAG_NOINIT        (unsigned int)0x0001

/* ENGINE flags that can be set by ENGINE_set_flags(). */
```

```
/* #define ENGINE_FLAGS_MALLOCED        0x0001 */ /* Not used */

/* This flag is for ENGINEs that wish to handle the various 'CMD'-related
 * control commands on their own. Without this flag, ENGINE_ctrl() handles these
 * control commands on behalf of the ENGINE using their "cmd_defns" data. */
#define ENGINE_FLAGS_MANUAL_CMD_CTRL    (int)0x0002

/* This flag is for ENGINEs who return new duplicate structures when found via
 * "ENGINE_by_id()". When an ENGINE must store state (eg. if ENGINE_ctrl()
 * commands are called in sequence as part of some stateful process like
 * key-generation setup and execution), it can set this flag - then each attempt
 * to obtain the ENGINE will result in it being copied into a new structure.
 * Normally, ENGINEs don't declare this flag so ENGINE_by_id() just increments
 * the existing ENGINE's structural reference count. */
#define ENGINE_FLAGS_BY_ID_COPY         (int)0x0004

/* This flag if for an ENGINE that does not want its methods registered as
 * part of ENGINE_register_all_complete() for example if the methods are
 * not usable as default methods.
 */

#define ENGINE_FLAGS_NO_REGISTER_ALL    (int)0x0008

/* ENGINEs can support their own command types, and these flags are used in
 * ENGINE_CTRL_GET_CMD_FLAGS to indicate to the caller what kind of input each
 * command expects. Currently only numeric and string input is supported. If a
 * control command supports none of the _NUMERIC, _STRING, or _NO_INPUT options,
 * then it is regarded as an "internal" control command - and not for use in
 * config setting situations. As such, they're not available to the
 * ENGINE_ctrl_cmd_string() function, only raw ENGINE_ctrl() access. Changes to
 * this list of 'command types' should be reflected carefully in
 * ENGINE_cmd_is_executable() and ENGINE_ctrl_cmd_string(). */

/* accepts a 'long' input value (3rd parameter to ENGINE_ctrl) */
#define ENGINE_CMD_FLAG_NUMERIC         (unsigned int)0x0001
/* accepts string input (cast from 'void*' to 'const char *', 4th parameter to
 * ENGINE_ctrl) */
#define ENGINE_CMD_FLAG_STRING          (unsigned int)0x0002
/* Indicates that the control command takes *no* input. Ie. the control command
 * is unparameterised. */
#define ENGINE_CMD_FLAG_NO_INPUT        (unsigned int)0x0004
/* Indicates that the control command is internal. This control command won't
 * be shown in any output, and is only usable through the ENGINE_ctrl_cmd()
 * function. */
#define ENGINE_CMD_FLAG_INTERNAL        (unsigned int)0x0008

/* NB: These 3 control commands are deprecated and should not be used. ENGINEs
 * relying on these commands should compile conditional support for
 * compatibility (eg. if these symbols are defined) but should also migrate the
 * same functionality to their own ENGINE-specific control functions that can be
 * "discovered" by calling applications. The fact these control commands
 * wouldn't be "executable" (ie. usable by text-based config) doesn't change the
 * fact that application code can find and use them without requiring per-ENGINE
 * hacking. */

/* These flags are used to tell the ctrl function what should be done.
 * All command numbers are shared between all engines, even if some don't
 * make sense to some engines.  In such a case, they do nothing but return
 * the error ENGINE_R_CTRL_COMMAND_NOT_IMPLEMENTED. */
#define ENGINE_CTRL_SET_LOGSTREAM               1
#define ENGINE_CTRL_SET_PASSWORD_CALLBACK       2
#define ENGINE_CTRL_HUP                         3 /* Close and reinitialise any
                                                     handles/connections etc. */
#define ENGINE_CTRL_SET_USER_INTERFACE          4 /* Alternative to callback */
#define ENGINE_CTRL_SET_CALLBACK_DATA           5 /* User-specific data, used
                                                     when calling the password
```



```
                                                     callback and the user
                                                     interface */
#define ENGINE_CTRL_LOAD_CONFIGURATION          6 /* Load a configuration, given
                                                     a string that represents a
                                                     file name or so */
#define ENGINE_CTRL_LOAD_SECTION                7 /* Load data from a given
                                                     section in the already load
                                                     configuration */

/* These control commands allow an application to deal with an arbitrary engine
 * in a dynamic way. Warn: Negative return values indicate errors FOR THESE
 * COMMANDS because zero is used to indicate 'end-of-list'. Other commands,
 * including ENGINE-specific command types, return zero for an error.
 *
 * An ENGINE can choose to implement these ctrl functions, and can internally
 * manage things however it chooses - it does so by setting the
 * ENGINE_FLAGS_MANUAL_CMD_CTRL flag (using ENGINE_set_flags()). Otherwise the
 * ENGINE_ctrl() code handles this on the ENGINE's behalf using the cmd_defns
 * data (set using ENGINE_set_cmd_defns()). This means an ENGINE's ctrl()
 * handler need only implement its own commands - the above "meta" commands will
 * be taken care of. */

/* Returns non-zero if the supplied ENGINE has a ctrl() handler. If "not", then
 * all the remaining control commands will return failure, so it is worth
 * checking this first if the caller is trying to "discover" the engine's
 * capabilities and doesn't want errors generated unnecessarily. */
#define ENGINE_CTRL_HAS_CTRL_FUNCTION           10
/* Returns a positive command number for the first command supported by the
 * engine. Returns zero if no ctrl commands are supported. */
#define ENGINE_CTRL_GET_FIRST_CMD_TYPE          11
/* The 'long' argument specifies a command implemented by the engine, and the
 * return value is the next command supported, or zero if there are no more. */
#define ENGINE_CTRL_GET_NEXT_CMD_TYPE           12
/* The 'void*' argument is a command name (cast from 'const char *'), and the
 * return value is the command that corresponds to it. */
#define ENGINE_CTRL_GET_CMD_FROM_NAME           13
/* The next two allow a command to be converted into its corresponding string
 * form. In each case, the 'long' argument supplies the command. In the NAME_LEN
 * case, the return value is the length of the command name (not counting a
 * trailing EOL). In the NAME case, the 'void*' argument must be a string buffer
 * large enough, and it will be populated with the name of the command (WITH a
 * trailing EOL). */
#define ENGINE_CTRL_GET_NAME_LEN_FROM_CMD       14
#define ENGINE_CTRL_GET_NAME_FROM_CMD           15
/* The next two are similar but give a "short description" of a command. */
#define ENGINE_CTRL_GET_DESC_LEN_FROM_CMD       16
#define ENGINE_CTRL_GET_DESC_FROM_CMD           17
/* With this command, the return value is the OR'd combination of
 * ENGINE_CMD_FLAG_*** values that indicate what kind of input a given
 * engine-specific ctrl command expects. */
#define ENGINE_CTRL_GET_CMD_FLAGS               18

/* ENGINE implementations should start the numbering of their own control
 * commands from this value. (ie. ENGINE_CMD_BASE, ENGINE_CMD_BASE + 1, etc). */
#define ENGINE_CMD_BASE                         200

/* NB: These 2 nCipher "chil" control commands are deprecated, and their
 * functionality is now available through ENGINE-specific control commands
 * (exposed through the above-mentioned 'CMD'-handling). Code using these 2
 * commands should be migrated to the more general command handling before these
 * are removed. */

/* Flags specific to the nCipher "chil" engine */
#define ENGINE_CTRL_CHIL_SET_FORKCHECK          100
        /* Depending on the value of the (long)i argument, this sets or
         * unsets the SimpleForkCheck flag in the CHIL API to enable or
```

```
260          * disable checking and workarounds for applications that fork().
261          */
262 #define ENGINE_CTRL_CHIL_NO_LOCKING            101
263         /* This prevents the initialisation function from providing mutex
264          * callbacks to the nCipher library. */

266 /* If an ENGINE supports its own specific control commands and wishes the
267  * framework to handle the above 'ENGINE_CMD_***'-manipulation commands on its
268  * behalf, it should supply a null-terminated array of ENGINE_CMD_DEFN entries
269  * to ENGINE_set_cmd_defns(). It should also implement a ctrl() handler that
270  * supports the stated commands (ie. the "cmd_num" entries as described by the
271  * array). NB: The array must be ordered in increasing order of cmd_num.
272  * "null-terminated" means that the last ENGINE_CMD_DEFN element has cmd_num set
273  * to zero and/or cmd_name set to NULL. */
274 typedef struct ENGINE_CMD_DEFN_st
275         {
276         unsigned int cmd_num; /* The command number */
277         const char *cmd_name; /* The command name itself */
278         const char *cmd_desc; /* A short description of the command */
279         unsigned int cmd_flags; /* The input the command expects */
280         } ENGINE_CMD_DEFN;

282 /* Generic function pointer */
283 typedef int (*ENGINE_GEN_FUNC_PTR)(void);
284 /* Generic function pointer taking no arguments */
285 typedef int (*ENGINE_GEN_INT_FUNC_PTR)(ENGINE *);
286 /* Specific control function pointer */
287 typedef int (*ENGINE_CTRL_FUNC_PTR)(ENGINE *, int, long, void *, void (*f)(void)
288 /* Generic load_key function pointer */
289 typedef EVP_PKEY * (*ENGINE_LOAD_KEY_PTR)(ENGINE *, const char *,
290         UI_METHOD *ui_method, void *callback_data);
291 typedef int (*ENGINE_SSL_CLIENT_CERT_PTR)(ENGINE *, SSL *ssl,
292         STACK_OF(X509_NAME) *ca_dn, X509 **pcert, EVP_PKEY **pkey,
293         STACK_OF(X509) **pother, UI_METHOD *ui_method, void *callback_data);
294 /* These callback types are for an ENGINE's handler for cipher and digest logic.
295  * These handlers have these prototypes;
296  *   int foo(ENGINE *e, const EVP_CIPHER **cipher, const int **nids, int nid);
297  *   int foo(ENGINE *e, const EVP_MD **digest, const int **nids, int nid);
298  * Looking at how to implement these handlers in the case of cipher support, if
299  * the framework wants the EVP_CIPHER for 'nid', it will call;
300  *   foo(e, &p_evp_cipher, NULL, nid);    (return zero for failure)
301  * If the framework wants a list of supported 'nid's, it will call;
302  *   foo(e, NULL, &p_nids, 0); (returns number of 'nids' or -1 for error)
303  */
304 /* Returns to a pointer to the array of supported cipher 'nid's. If the second
305  * parameter is non-NULL it is set to the size of the returned array. */
306 typedef int (*ENGINE_CIPHERS_PTR)(ENGINE *, const EVP_CIPHER **, const int **, i
307 typedef int (*ENGINE_DIGESTS_PTR)(ENGINE *, const EVP_MD **, const int **, int);
308 typedef int (*ENGINE_PKEY_METHS_PTR)(ENGINE *, EVP_PKEY_METHOD **, const int **,
309 typedef int (*ENGINE_PKEY_ASN1_METHS_PTR)(ENGINE *, EVP_PKEY_ASN1_METHOD **, con
310 /* STRUCTURE functions ... all of these functions deal with pointers to ENGINE
311  * structures where the pointers have a "structural reference". This means that
312  * their reference is to allowed access to the structure but it does not imply
313  * that the structure is functional. To simply increment or decrement the
314  * structural reference count, use ENGINE_by_id and ENGINE_free. NB: This is not
315  * required when iterating using ENGINE_get_next as it will automatically
316  * decrement the structural reference count of the "current" ENGINE and
317  * increment the structural reference count of the ENGINE it returns (unless it
318  * is NULL). */

320 /* Get the first/last "ENGINE" type available. */
321 ENGINE *ENGINE_get_first(void);
322 ENGINE *ENGINE_get_last(void);
323 /* Iterate to the next/previous "ENGINE" type (NULL = end of the list). */
324 ENGINE *ENGINE_get_next(ENGINE *e);
325 ENGINE *ENGINE_get_prev(ENGINE *e);
```



```
326 /* Add another "ENGINE" type into the array. */
327 int ENGINE_add(ENGINE *e);
328 /* Remove an existing "ENGINE" type from the array. */
329 int ENGINE_remove(ENGINE *e);
330 /* Retrieve an engine from the list by its unique "id" value. */
331 ENGINE *ENGINE_by_id(const char *id);
332 /* Add all the built-in engines. */
333 void ENGINE_load_openssl(void);
334 void ENGINE_load_dynamic(void);
335 #ifndef OPENSSL_NO_STATIC_ENGINE
336 void ENGINE_load_4758cca(void);
337 void ENGINE_load_aep(void);
338 void ENGINE_load_atalla(void);
339 void ENGINE_load_chil(void);
340 void ENGINE_load_cswift(void);
341 void ENGINE_load_nuron(void);
342 void ENGINE_load_sureware(void);
343 void ENGINE_load_ubsec(void);
344 void ENGINE_load_padlock(void);
345 void ENGINE_load_capi(void);
346 #ifndef OPENSSL_NO_GMP
347 void ENGINE_load_gmp(void);
348 #endif
349 #ifndef OPENSSL_NO_GOST
350 void ENGINE_load_gost(void);
351 #endif
352 #endif
353 void ENGINE_load_cryptodev(void);
354 void ENGINE_load_pk11(void);
355 #endif /* ! codereview */
356 void ENGINE_load_rsax(void);
357 void ENGINE_load_rdrand(void);
358 void ENGINE_load_builtin_engines(void);

360 /* Get and set global flags (ENGINE_TABLE_FLAG_***) for the implementation
361  * "registry" handling. */
362 unsigned int ENGINE_get_table_flags(void);
363 void ENGINE_set_table_flags(unsigned int flags);

365 /* Manage registration of ENGINEs per "table". For each type, there are 3
366  * functions;
367  *   ENGINE_register_***(e) - registers the implementation from 'e' (if it has o
368  *   ENGINE_unregister_***(e) - unregister the implementation from 'e'
369  *   ENGINE_register_all_***() - call ENGINE_register_***() for each 'e' in the
370  * Cleanup is automatically registered from each table when required, so
371  * ENGINE_cleanup() will reverse any "register" operations. */

373 int ENGINE_register_RSA(ENGINE *e);
374 void ENGINE_unregister_RSA(ENGINE *e);
375 void ENGINE_register_all_RSA(void);

377 int ENGINE_register_DSA(ENGINE *e);
378 void ENGINE_unregister_DSA(ENGINE *e);
379 void ENGINE_register_all_DSA(void);

381 int ENGINE_register_ECDH(ENGINE *e);
382 void ENGINE_unregister_ECDH(ENGINE *e);
383 void ENGINE_register_all_ECDH(void);

385 int ENGINE_register_ECDSA(ENGINE *e);
386 void ENGINE_unregister_ECDSA(ENGINE *e);
387 void ENGINE_register_all_ECDSA(void);

389 int ENGINE_register_DH(ENGINE *e);
390 void ENGINE_unregister_DH(ENGINE *e);
391 void ENGINE_register_all_DH(void);
```

```
393 int ENGINE_register_RAND(ENGINE *e);
394 void ENGINE_unregister_RAND(ENGINE *e);
395 void ENGINE_register_all_RAND(void);

397 int ENGINE_register_STORE(ENGINE *e);
398 void ENGINE_unregister_STORE(ENGINE *e);
399 void ENGINE_register_all_STORE(void);

401 int ENGINE_register_ciphers(ENGINE *e);
402 void ENGINE_unregister_ciphers(ENGINE *e);
403 void ENGINE_register_all_ciphers(void);

405 int ENGINE_register_digests(ENGINE *e);
406 void ENGINE_unregister_digests(ENGINE *e);
407 void ENGINE_register_all_digests(void);

409 int ENGINE_register_pkey_meths(ENGINE *e);
410 void ENGINE_unregister_pkey_meths(ENGINE *e);
411 void ENGINE_register_all_pkey_meths(void);

413 int ENGINE_register_pkey_asn1_meths(ENGINE *e);
414 void ENGINE_unregister_pkey_asn1_meths(ENGINE *e);
415 void ENGINE_register_all_pkey_asn1_meths(void);

417 /* These functions register all support from the above categories. Note, use of
418  * these functions can result in static linkage of code your application may not
419  * need. If you only need a subset of functionality, consider using more
420  * selective initialisation. */
421 int ENGINE_register_complete(ENGINE *e);
422 int ENGINE_register_all_complete(void);

424 /* Send parametrised control commands to the engine. The possibilities to send
425  * down an integer, a pointer to data or a function pointer are provided. Any of
426  * the parameters may or may not be NULL, depending on the command number. In
427  * actuality, this function only requires a structural (rather than functional)
428  * reference to an engine, but many control commands may require the engine be
429  * functional. The caller should be aware of trying commands that require an
430  * operational ENGINE, and only use functional references in such situations. */
431 int ENGINE_ctrl(ENGINE *e, int cmd, long i, void *p, void (*f)(void));

433 /* This function tests if an ENGINE-specific command is usable as a "setting".
434  * Eg. in an application's config file that gets processed through
435  * ENGINE_ctrl_cmd_string(). If this returns zero, it is not available to
436  * ENGINE_ctrl_cmd_string(), only ENGINE_ctrl(). */
437 int ENGINE_cmd_is_executable(ENGINE *e, int cmd);

439 /* This function works like ENGINE_ctrl() with the exception of taking a
440  * command name instead of a command number, and can handle optional commands.
441  * See the comment on ENGINE_ctrl_cmd_string() for an explanation on how to
442  * use the cmd_name and cmd_optional. */
443 int ENGINE_ctrl_cmd(ENGINE *e, const char *cmd_name,
444         long i, void *p, void (*f)(void), int cmd_optional);

446 /* This function passes a command-name and argument to an ENGINE. The cmd_name
447  * is converted to a command number and the control command is called using
448  * 'arg' as an argument (unless the ENGINE doesn't support such a command, in
449  * which case no control command is called). The command is checked for input
450  * flags, and if necessary the argument will be converted to a numeric value. If
451  * cmd_optional is non-zero, then if the ENGINE doesn't support the given
452  * cmd_name the return value will be success anyway. This function is intended
453  * for applications to use so that users (or config files) can supply
454  * engine-specific config data to the ENGINE at run-time to control behaviour of
455  * specific engines. As such, it shouldn't be used for calling ENGINE_ctrl()
456  * functions that return data, deal with binary data, or that are otherwise
457  * supposed to be used directly through ENGINE_ctrl() in application code. Any
```



```
458  * "return" data from an ENGINE_ctrl() operation in this function will be lost -
459  * the return value is interpreted as failure if the return value is zero,
460  * success otherwise, and this function returns a boolean value as a result. In
461  * other words, vendors of 'ENGINE'-enabled devices should write ENGINE
462  * implementations with parameterisations that work in this scheme, so that
463  * compliant ENGINE-based applications can work consistently with the same
464  * configuration for the same ENGINE-enabled devices, across applications. */
465 int ENGINE_ctrl_cmd_string(ENGINE *e, const char *cmd_name, const char *arg,
466                            int cmd_optional);

468 /* These functions are useful for manufacturing new ENGINE structures. They
469  * don't address reference counting at all - one uses them to populate an ENGINE
470  * structure with personalised implementations of things prior to using it
471  * directly or adding it to the builtin ENGINE list in OpenSSL. These are also
472  * here so that the ENGINE structure doesn't have to be exposed and break binary
473  * compatibility! */
474 ENGINE *ENGINE_new(void);
475 int ENGINE_free(ENGINE *e);
476 int ENGINE_up_ref(ENGINE *e);
477 int ENGINE_set_id(ENGINE *e, const char *id);
478 int ENGINE_set_name(ENGINE *e, const char *name);
479 int ENGINE_set_RSA(ENGINE *e, const RSA_METHOD *rsa_meth);
480 int ENGINE_set_DSA(ENGINE *e, const DSA_METHOD *dsa_meth);
481 int ENGINE_set_ECDH(ENGINE *e, const ECDH_METHOD *ecdh_meth);
482 int ENGINE_set_ECDSA(ENGINE *e, const ECDSA_METHOD *ecdsa_meth);
483 int ENGINE_set_DH(ENGINE *e, const DH_METHOD *dh_meth);
484 int ENGINE_set_RAND(ENGINE *e, const RAND_METHOD *rand_meth);
485 int ENGINE_set_STORE(ENGINE *e, const STORE_METHOD *store_meth);
486 int ENGINE_set_destroy_function(ENGINE *e, ENGINE_GEN_INT_FUNC_PTR destroy_f);
487 int ENGINE_set_init_function(ENGINE *e, ENGINE_GEN_INT_FUNC_PTR init_f);
488 int ENGINE_set_finish_function(ENGINE *e, ENGINE_GEN_INT_FUNC_PTR finish_f);
489 int ENGINE_set_ctrl_function(ENGINE *e, ENGINE_CTRL_FUNC_PTR ctrl_f);
490 int ENGINE_set_load_privkey_function(ENGINE *e, ENGINE_LOAD_KEY_PTR loadpriv_f);
491 int ENGINE_set_load_pubkey_function(ENGINE *e, ENGINE_LOAD_KEY_PTR loadpub_f);
492 int ENGINE_set_load_ssl_client_cert_function(ENGINE *e,
493                                              ENGINE_SSL_CLIENT_CERT_PTR loadssl_f);
494 int ENGINE_set_ciphers(ENGINE *e, ENGINE_CIPHERS_PTR f);
495 int ENGINE_set_digests(ENGINE *e, ENGINE_DIGESTS_PTR f);
496 int ENGINE_set_pkey_meths(ENGINE *e, ENGINE_PKEY_METHS_PTR f);
497 int ENGINE_set_pkey_asn1_meths(ENGINE *e, ENGINE_PKEY_ASN1_METHS_PTR f);
498 int ENGINE_set_flags(ENGINE *e, int flags);
499 int ENGINE_set_cmd_defns(ENGINE *e, const ENGINE_CMD_DEFN *defns);
500 /* These functions allow control over any per-structure ENGINE data. */
501 int ENGINE_get_ex_new_index(long argl, void *argp, CRYPTO_EX_new *new_func,
502                             CRYPTO_EX_dup *dup_func, CRYPTO_EX_free *free_func);
503 int ENGINE_set_ex_data(ENGINE *e, int idx, void *arg);
504 void *ENGINE_get_ex_data(const ENGINE *e, int idx);

506 /* This function cleans up anything that needs it. Eg. the ENGINE_add() function
507  * automatically ensures the list cleanup function is registered to be called
508  * from ENGINE_cleanup(). Similarly, all ENGINE_register_*** functions ensure
509  * ENGINE_cleanup() will clean up after them. */
510 void ENGINE_cleanup(void);

512 /* These return values from within the ENGINE structure. These can be useful
513  * with functional references as well as structural references - it depends
514  * which you obtained. Using the result for functional purposes if you only
515  * obtained a structural reference may be problematic! */
516 const char *ENGINE_get_id(const ENGINE *e);
517 const char *ENGINE_get_name(const ENGINE *e);
518 const RSA_METHOD *ENGINE_get_RSA(const ENGINE *e);
519 const DSA_METHOD *ENGINE_get_DSA(const ENGINE *e);
520 const ECDH_METHOD *ENGINE_get_ECDH(const ENGINE *e);
521 const ECDSA_METHOD *ENGINE_get_ECDSA(const ENGINE *e);
522 const DH_METHOD *ENGINE_get_DH(const ENGINE *e);
523 const RAND_METHOD *ENGINE_get_RAND(const ENGINE *e);
```

```
524 const STORE_METHOD *ENGINE_get_STORE(const ENGINE *e);
525 ENGINE_GEN_INT_FUNC_PTR ENGINE_get_destroy_function(const ENGINE *e);
526 ENGINE_GEN_INT_FUNC_PTR ENGINE_get_init_function(const ENGINE *e);
527 ENGINE_GEN_INT_FUNC_PTR ENGINE_get_finish_function(const ENGINE *e);
528 ENGINE_CTRL_FUNC_PTR ENGINE_get_ctrl_function(const ENGINE *e);
529 ENGINE_LOAD_KEY_PTR ENGINE_get_load_privkey_function(const ENGINE *e);
530 ENGINE_LOAD_KEY_PTR ENGINE_get_load_pubkey_function(const ENGINE *e);
531 ENGINE_SSL_CLIENT_CERT_PTR ENGINE_get_ssl_client_cert_function(const ENGINE *e);
532 ENGINE_CIPHERS_PTR ENGINE_get_ciphers(const ENGINE *e);
533 ENGINE_DIGESTS_PTR ENGINE_get_digests(const ENGINE *e);
534 ENGINE_PKEY_METHS_PTR ENGINE_get_pkey_meths(const ENGINE *e);
535 ENGINE_PKEY_ASN1_METHS_PTR ENGINE_get_pkey_asn1_meths(const ENGINE *e);
536 const EVP_CIPHER *ENGINE_get_cipher(ENGINE *e, int nid);
537 const EVP_MD *ENGINE_get_digest(ENGINE *e, int nid);
538 const EVP_PKEY_METHOD *ENGINE_get_pkey_meth(ENGINE *e, int nid);
539 const EVP_PKEY_ASN1_METHOD *ENGINE_get_pkey_asn1_meth(ENGINE *e, int nid);
540 const EVP_PKEY_ASN1_METHOD *ENGINE_get_pkey_asn1_meth_str(ENGINE *e,
541                                       const char *str, int len);
542 const EVP_PKEY_ASN1_METHOD *ENGINE_pkey_asn1_find_str(ENGINE **pe,
543                                       const char *str, int len);
544 const ENGINE_CMD_DEFN *ENGINE_get_cmd_defns(const ENGINE *e);
545 int ENGINE_get_flags(const ENGINE *e);

547 /* FUNCTIONAL functions. These functions deal with ENGINE structures
548  * that have (or will) be initialised for use. Broadly speaking, the
549  * structural functions are useful for iterating the list of available
550  * engine types, creating new engine types, and other "list" operations.
551  * These functions actually deal with ENGINEs that are to be used. As
552  * such these functions can fail (if applicable) when particular
553  * engines are unavailable - eg. if a hardware accelerator is not
554  * attached or not functioning correctly. Each ENGINE has 2 reference
555  * counts; structural and functional. Every time a functional reference
556  * is obtained or released, a corresponding structural reference is
557  * automatically obtained or released too. */

559 /* Initialise a engine type for use (or up its reference count if it's
560  * already in use). This will fail if the engine is not currently
561  * operational and cannot initialise. */
562 int ENGINE_init(ENGINE *e);
563 /* Free a functional reference to a engine type. This does not require
564  * a corresponding call to ENGINE_free as it also releases a structural
565  * reference. */
566 int ENGINE_finish(ENGINE *e);

568 /* The following functions handle keys that are stored in some secondary
569  * location, handled by the engine.  The storage may be on a card or
570  * whatever. */
571 EVP_PKEY *ENGINE_load_private_key(ENGINE *e, const char *key_id,
572         UI_METHOD *ui_method, void *callback_data);
573 EVP_PKEY *ENGINE_load_public_key(ENGINE *e, const char *key_id,
574         UI_METHOD *ui_method, void *callback_data);
575 int ENGINE_load_ssl_client_cert(ENGINE *e, SSL *s,
576         STACK_OF(X509_NAME) *ca_dn, X509 **pcert, EVP_PKEY **ppkey,
577         STACK_OF(X509) **pother,
578         UI_METHOD *ui_method, void *callback_data);

580 /* This returns a pointer for the current ENGINE structure that
581  * is (by default) performing any RSA operations. The value returned
582  * is an incremented reference, so it should be free'd (ENGINE_finish)
583  * before it is discarded. */
584 ENGINE *ENGINE_get_default_RSA(void);
585 /* Same for the other "methods" */
586 ENGINE *ENGINE_get_default_DSA(void);
587 ENGINE *ENGINE_get_default_ECDH(void);
588 ENGINE *ENGINE_get_default_ECDSA(void);
589 ENGINE *ENGINE_get_default_DH(void);
```



```
590 ENGINE *ENGINE_get_default_RAND(void);
591 /* These functions can be used to get a functional reference to perform
592  * ciphering or digesting corresponding to "nid". */
593 ENGINE *ENGINE_get_cipher_engine(int nid);
594 ENGINE *ENGINE_get_digest_engine(int nid);
595 ENGINE *ENGINE_get_pkey_meth_engine(int nid);
596 ENGINE *ENGINE_get_pkey_asn1_meth_engine(int nid);

598 /* This sets a new default ENGINE structure for performing RSA
599  * operations. If the result is non-zero (success) then the ENGINE
600  * structure will have had its reference count up'd so the caller
601  * should still free their own reference 'e'. */
602 int ENGINE_set_default_RSA(ENGINE *e);
603 int ENGINE_set_default_string(ENGINE *e, const char *def_list);
604 /* Same for the other "methods" */
605 int ENGINE_set_default_DSA(ENGINE *e);
606 int ENGINE_set_default_ECDH(ENGINE *e);
607 int ENGINE_set_default_ECDSA(ENGINE *e);
608 int ENGINE_set_default_DH(ENGINE *e);
609 int ENGINE_set_default_RAND(ENGINE *e);
610 int ENGINE_set_default_ciphers(ENGINE *e);
611 int ENGINE_set_default_digests(ENGINE *e);
612 int ENGINE_set_default_pkey_meths(ENGINE *e);
613 int ENGINE_set_default_pkey_asn1_meths(ENGINE *e);

615 /* The combination "set" - the flags are bitwise "OR"d from the
616  * ENGINE_METHOD_*** defines above. As with the "ENGINE_register_complete()"
617  * function, this function can result in unnecessary static linkage. If your
618  * application requires only specific functionality, consider using more
619  * selective functions. */
620 int ENGINE_set_default(ENGINE *e, unsigned int flags);

622 void ENGINE_add_conf_module(void);

624 /* Deprecated functions ... */
625 /* int ENGINE_clear_defaults(void); */

627 /**************************/
628 /* DYNAMIC ENGINE SUPPORT */
629 /**************************/

631 /* Binary/behaviour compatibility levels */
632 #define OSSL_DYNAMIC_VERSION            (unsigned long)0x00020000
633 /* Binary versions older than this are too old for us (whether we're a loader or
634  * a loadee) */
635 #define OSSL_DYNAMIC_OLDEST             (unsigned long)0x00020000

637 /* When compiling an ENGINE entirely as an external shared library, loadable by
638  * the "dynamic" ENGINE, these types are needed. The 'dynamic_fns' structure
639  * type provides the calling application's (or library's) error functionality
640  * and memory management function pointers to the loaded library. These should
641  * be used/set in the loaded library code so that the loading application's
642  * 'state' will be used/changed in all operations. The 'static_state' pointer
643  * allows the loaded library to know if it shares the same static data as the
644  * calling application (or library), and thus whether these callbacks need to be
645  * set or not. */
646 typedef void *(*dyn_MEM_malloc_cb)(size_t);
647 typedef void *(*dyn_MEM_realloc_cb)(void *, size_t);
648 typedef void (*dyn_MEM_free_cb)(void *);
649 typedef struct st_dynamic_MEM_fns {
650         dyn_MEM_malloc_cb                       malloc_cb;
651         dyn_MEM_realloc_cb                      realloc_cb;
652         dyn_MEM_free_cb                         free_cb;
653         } dynamic_MEM_fns;
654 /* FIXME: Perhaps the memory and locking code (crypto.h) should declare and use
655  * these types so we (and any other dependant code) can simplify a bit?? */
```

```
656 typedef void (*dyn_lock_locking_cb)(int,int,const char *,int);
657 typedef int (*dyn_lock_add_lock_cb)(int*,int,int,const char *,int);
658 typedef struct CRYPTO_dynlock_value *(*dyn_dynlock_create_cb)(
659                                         const char *,int);
660 typedef void (*dyn_dynlock_lock_cb)(int,struct CRYPTO_dynlock_value *,
661                                         const char *,int);
662 typedef void (*dyn_dynlock_destroy_cb)(struct CRYPTO_dynlock_value *,
663                                         const char *,int);
664 typedef struct st_dynamic_LOCK_fns {
665         dyn_lock_locking_cb                     lock_locking_cb;
666         dyn_lock_add_lock_cb                    lock_add_lock_cb;
667         dyn_dynlock_create_cb                   dynlock_create_cb;
668         dyn_dynlock_lock_cb                     dynlock_lock_cb;
669         dyn_dynlock_destroy_cb                  dynlock_destroy_cb;
670         } dynamic_LOCK_fns;
671 /* The top-level structure */
672 typedef struct st_dynamic_fns {
673         void                                    *static_state;
674         const ERR_FNS                           *err_fns;
675         const CRYPTO_EX_DATA_IMPL               *ex_data_fns;
676         dynamic_MEM_fns                         mem_fns;
677         dynamic_LOCK_fns                        lock_fns;
678         } dynamic_fns;

680 /* The version checking function should be of this prototype. NB: The
681  * ossl_version value passed in is the OSSL_DYNAMIC_VERSION of the loading code.
682  * If this function returns zero, it indicates a (potential) version
683  * incompatibility and the loaded library doesn't believe it can proceed.
684  * Otherwise, the returned value is the (latest) version supported by the
685  * loading library. The loader may still decide that the loaded code's version
686  * is unsatisfactory and could veto the load. The function is expected to
687  * be implemented with the symbol name "v_check", and a default implementation
688  * can be fully instantiated with IMPLEMENT_DYNAMIC_CHECK_FN(). */
689 typedef unsigned long (*dynamic_v_check_fn)(unsigned long ossl_version);
690 #define IMPLEMENT_DYNAMIC_CHECK_FN() \
691         OPENSSL_EXPORT unsigned long v_check(unsigned long v); \
692         OPENSSL_EXPORT unsigned long v_check(unsigned long v) { \
693                 if(v >= OSSL_DYNAMIC_OLDEST) return OSSL_DYNAMIC_VERSION; \
694                 return 0; }

696 /* This function is passed the ENGINE structure to initialise with its own
697  * function and command settings. It should not adjust the structural or
698  * functional reference counts. If this function returns zero, (a) the load will
699  * be aborted, (b) the previous ENGINE state will be memcpy'd back onto the
700  * structure, and (c) the shared library will be unloaded. So implementations
701  * should do their own internal cleanup in failure circumstances otherwise they
702  * could leak. The 'id' parameter, if non-NULL, represents the ENGINE id that
703  * the loader is looking for. If this is NULL, the shared library can choose to
704  * return failure or to initialise a 'default' ENGINE. If non-NULL, the shared
705  * library must initialise only an ENGINE matching the passed 'id'. The function
706  * is expected to be implemented with the symbol name "bind_engine". A standard
707  * implementation can be instantiated with IMPLEMENT_DYNAMIC_BIND_FN(fn) where
708  * the parameter 'fn' is a callback function that populates the ENGINE structure
709  * and returns an int value (zero for failure). 'fn' should have prototype;
710  *    [static] int fn(ENGINE *e, const char *id); */
711 typedef int (*dynamic_bind_engine)(ENGINE *e, const char *id,
712                                    const dynamic_fns *fns);
713 #define IMPLEMENT_DYNAMIC_BIND_FN(fn) \
714         OPENSSL_EXPORT \
715         int bind_engine(ENGINE *e, const char *id, const dynamic_fns *fns); \
716         OPENSSL_EXPORT \
717         int bind_engine(ENGINE *e, const char *id, const dynamic_fns *fns) { \
718                 if(ENGINE_get_static_state() == fns->static_state) goto skip_cbs \
719                 if(!CRYPTO_set_mem_functions(fns->mem_fns.malloc_cb, \
720                         fns->mem_fns.realloc_cb, fns->mem_fns.free_cb)) \
721                         return 0; \
```



```
722                 CRYPTO_set_locking_callback(fns->lock_fns.lock_locking_cb); \
723                 CRYPTO_set_add_lock_callback(fns->lock_fns.lock_add_lock_cb); \
724                 CRYPTO_set_dynlock_create_callback(fns->lock_fns.dynlock_create_
725                 CRYPTO_set_dynlock_lock_callback(fns->lock_fns.dynlock_lock_cb);
726                 CRYPTO_set_dynlock_destroy_callback(fns->lock_fns.dynlock_destro
727                 if(!CRYPTO_set_ex_data_implementation(fns->ex_data_fns)) \
728                         return 0; \
729                 if(!ERR_set_implementation(fns->err_fns)) return 0; \
730         skip_cbs: \
731                 if(!fn(e,id)) return 0; \
732                 return 1; }

734 /* If the loading application (or library) and the loaded ENGINE library share
735  * the same static data (eg. they're both dynamically linked to the same
736  * libcrypto.so) we need a way to avoid trying to set system callbacks - this
737  * would fail, and for the same reason that it's unnecessary to try. If the
738  * loaded ENGINE has (or gets from through the loader) its own copy of the
739  * libcrypto static data, we will need to set the callbacks. The easiest way to
740  * detect this is to have a function that returns a pointer to some static data
741  * and let the loading application and loaded ENGINE compare their respective
742  * values. */
743 void *ENGINE_get_static_state(void);

745 #if defined(__OpenBSD__) || defined(__FreeBSD__) || defined(HAVE_CRYPTODEV)
746 void ENGINE_setup_bsd_cryptodev(void);
747 #endif

749 /* BEGIN ERROR CODES */
750 /* The following lines are auto generated by the script mkerr.pl. Any changes
751  * made after this point may be overwritten when the script is next run.
752  */
753 void ERR_load_ENGINE_strings(void);

755 /* Error codes for the ENGINE functions. */

757 /* Function codes. */
758 #define ENGINE_F_DYNAMIC_CTRL                           180
759 #define ENGINE_F_DYNAMIC_GET_DATA_CTX                   181
760 #define ENGINE_F_DYNAMIC_LOAD                           182
761 #define ENGINE_F_DYNAMIC_SET_DATA_CTX                   183
762 #define ENGINE_F_ENGINE_ADD                             105
763 #define ENGINE_F_ENGINE_BY_ID                           106
764 #define ENGINE_F_ENGINE_CMD_IS_EXECUTABLE               170
765 #define ENGINE_F_ENGINE_CTRL                            142
766 #define ENGINE_F_ENGINE_CTRL_CMD                        178
767 #define ENGINE_F_ENGINE_CTRL_CMD_STRING                 171
768 #define ENGINE_F_ENGINE_FINISH                          107
769 #define ENGINE_F_ENGINE_FREE_UTIL                       108
770 #define ENGINE_F_ENGINE_GET_CIPHER                      185
771 #define ENGINE_F_ENGINE_GET_DEFAULT_TYPE                177
772 #define ENGINE_F_ENGINE_GET_DIGEST                      186
773 #define ENGINE_F_ENGINE_GET_NEXT                        115
774 #define ENGINE_F_ENGINE_GET_PKEY_ASN1_METH              193
775 #define ENGINE_F_ENGINE_GET_PKEY_METH                   192
776 #define ENGINE_F_ENGINE_GET_PREV                        116
777 #define ENGINE_F_ENGINE_INIT                            119
778 #define ENGINE_F_ENGINE_LIST_ADD                        120
779 #define ENGINE_F_ENGINE_LIST_REMOVE                     121
780 #define ENGINE_F_ENGINE_LOAD_PRIVATE_KEY                150
781 #define ENGINE_F_ENGINE_LOAD_PUBLIC_KEY                 151
782 #define ENGINE_F_ENGINE_LOAD_SSL_CLIENT_CERT            194
783 #define ENGINE_F_ENGINE_NEW                             122
784 #define ENGINE_F_ENGINE_REMOVE                          123
785 #define ENGINE_F_ENGINE_SET_DEFAULT_STRING              189
786 #define ENGINE_F_ENGINE_SET_DEFAULT_TYPE                126
787 #define ENGINE_F_ENGINE_SET_ID                          129
```

```
#define ENGINE_F_ENGINE_SET_NAME                        130
#define ENGINE_F_ENGINE_TABLE_REGISTER                  184
#define ENGINE_F_ENGINE_UNLOAD_KEY                      152
#define ENGINE_F_ENGINE_UNLOCKED_FINISH                 191
#define ENGINE_F_ENGINE_UP_REF                          190
#define ENGINE_F_INT_CTRL_HELPER                        172
#define ENGINE_F_INT_ENGINE_CONFIGURE                   188
#define ENGINE_F_INT_ENGINE_MODULE_INIT                 187
#define ENGINE_F_LOG_MESSAGE                            141

/* Reason codes. */
#define ENGINE_R_ALREADY_LOADED                         100
#define ENGINE_R_ARGUMENT_IS_NOT_A_NUMBER               133
#define ENGINE_R_CMD_NOT_EXECUTABLE                     134
#define ENGINE_R_COMMAND_TAKES_INPUT                    135
#define ENGINE_R_COMMAND_TAKES_NO_INPUT                 136
#define ENGINE_R_CONFLICTING_ENGINE_ID                  103
#define ENGINE_R_CTRL_COMMAND_NOT_IMPLEMENTED           119
#define ENGINE_R_DH_NOT_IMPLEMENTED                     139
#define ENGINE_R_DSA_NOT_IMPLEMENTED                    140
#define ENGINE_R_DSO_FAILURE                            104
#define ENGINE_R_DSO_NOT_FOUND                          132
#define ENGINE_R_ENGINES_SECTION_ERROR                  148
#define ENGINE_R_ENGINE_CONFIGURATION_ERROR             102
#define ENGINE_R_ENGINE_IS_NOT_IN_LIST                  105
#define ENGINE_R_ENGINE_SECTION_ERROR                   149
#define ENGINE_R_FAILED_LOADING_PRIVATE_KEY             128
#define ENGINE_R_FAILED_LOADING_PUBLIC_KEY              129
#define ENGINE_R_FINISH_FAILED                          106
#define ENGINE_R_GET_HANDLE_FAILED                      107
#define ENGINE_R_ID_OR_NAME_MISSING                     108
#define ENGINE_R_INIT_FAILED                            109
#define ENGINE_R_INTERNAL_LIST_ERROR                    110
#define ENGINE_R_INVALID_ARGUMENT                       143
#define ENGINE_R_INVALID_CMD_NAME                       137
#define ENGINE_R_INVALID_CMD_NUMBER                     138
#define ENGINE_R_INVALID_INIT_VALUE                     151
#define ENGINE_R_INVALID_STRING                         150
#define ENGINE_R_NOT_INITIALISED                        117
#define ENGINE_R_NOT_LOADED                             112
#define ENGINE_R_NO_CONTROL_FUNCTION                    120
#define ENGINE_R_NO_INDEX                               144
#define ENGINE_R_NO_LOAD_FUNCTION                       125
#define ENGINE_R_NO_REFERENCE                           130
#define ENGINE_R_NO_SUCH_ENGINE                         116
#define ENGINE_R_NO_UNLOAD_FUNCTION                     126
#define ENGINE_R_PROVIDE_PARAMETERS                     113
#define ENGINE_R_RSA_NOT_IMPLEMENTED                    141
#define ENGINE_R_UNIMPLEMENTED_CIPHER                   146
#define ENGINE_R_UNIMPLEMENTED_DIGEST                   147
#define ENGINE_R_UNIMPLEMENTED_PUBLIC_KEY_METHOD        101
#define ENGINE_R_VERSION_INCOMPATIBILITY                145

#ifdef  __cplusplus
}
#endif
#endif
```

```
**********************************************************
   10126 Wed Aug 13 19:32:53 2014
new/usr/src/lib/openssl/include/openssl/opensslconf.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* opensslconf.h */
/* WARNING: Generated automatically from opensslconf.h.in by Configure. */

/* OpenSSL was configured with the following options: */
#ifndef OPENSSL_DOING_MAKEDEPEND


#ifndef OPENSSL_NO_EC
# define OPENSSL_NO_EC
#endif
#endif /* ! codereview */
#ifndef OPENSSL_NO_EC_NISTP_64_GCC_128
# define OPENSSL_NO_EC_NISTP_64_GCC_128
#endif
#ifndef OPENSSL_NO_ECDH
# define OPENSSL_NO_ECDH
#endif
#ifndef OPENSSL_NO_ECDSA
# define OPENSSL_NO_ECDSA
#endif
#endif /* ! codereview */
#ifndef OPENSSL_NO_GMP
# define OPENSSL_NO_GMP
#endif
#ifndef OPENSSL_NO_GOST
# define OPENSSL_NO_GOST
#endif
#ifndef OPENSSL_NO_HW_4758_CCA
# define OPENSSL_NO_HW_4758_CCA
#endif
#ifndef OPENSSL_NO_HW_AEP
# define OPENSSL_NO_HW_AEP
#endif
#ifndef OPENSSL_NO_HW_ATALLA
# define OPENSSL_NO_HW_ATALLA
#endif
#ifndef OPENSSL_NO_HW_CHIL
# define OPENSSL_NO_HW_CHIL
#endif
#ifndef OPENSSL_NO_HW_CSWIFT
# define OPENSSL_NO_HW_CSWIFT
#endif
#ifndef OPENSSL_NO_HW_GMP
# define OPENSSL_NO_HW_GMP
#endif
#ifndef OPENSSL_NO_HW_NCIPHER
# define OPENSSL_NO_HW_NCIPHER
#endif
#ifndef OPENSSL_NO_HW_NURON
# define OPENSSL_NO_HW_NURON
#endif
#ifndef OPENSSL_NO_HW_PADLOCK
# define OPENSSL_NO_HW_PADLOCK
#endif
#ifndef OPENSSL_NO_HW_SUREWARE
# define OPENSSL_NO_HW_SUREWARE
#endif
#ifndef OPENSSL_NO_HW_UBSEC
# define OPENSSL_NO_HW_UBSEC
#endif
#ifndef OPENSSL_NO_IDEA
```



```
# define OPENSSL_NO_IDEA
#endif
#endif /* ! codereview */
#ifndef OPENSSL_NO_JPAKE
# define OPENSSL_NO_JPAKE
#endif
#ifndef OPENSSL_NO_KRB5
# define OPENSSL_NO_KRB5
#endif
#ifndef OPENSSL_NO_MDC2
# define OPENSSL_NO_MDC2
#endif
#ifndef OPENSSL_NO_RC3
# define OPENSSL_NO_RC3
#ifndef OPENSSL_NO_MD2
# define OPENSSL_NO_MD2
#endif
#ifndef OPENSSL_NO_RC5
# define OPENSSL_NO_RC5
#endif
#ifndef OPENSSL_NO_RFC3779
# define OPENSSL_NO_RFC3779
#endif
#ifndef OPENSSL_NO_SCTP
# define OPENSSL_NO_SCTP
#endif
#ifndef OPENSSL_NO_SEED
# define OPENSSL_NO_SEED
#endif
#endif /* ! codereview */
#ifndef OPENSSL_NO_STORE
# define OPENSSL_NO_STORE
#endif
#ifndef OPENSSL_NO_UNIT_TEST
# define OPENSSL_NO_UNIT_TEST
#endif
#ifndef OPENSSL_NO_WHIRLPOOL
# define OPENSSL_NO_WHIRLPOOL
#endif
#ifndef OPENSSL_NO_WHRLPOOL
# define OPENSSL_NO_WHRLPOOL
#endif
#endif /* ! codereview */

#endif /* OPENSSL_DOING_MAKEDEPEND */

#ifndef OPENSSL_THREADS
# define OPENSSL_THREADS
#endif
#ifndef OPENSSL_NO_STATIC_ENGINE
# define OPENSSL_NO_STATIC_ENGINE
#ifndef OPENSSL_NO_DYNAMIC_ENGINE
# define OPENSSL_NO_DYNAMIC_ENGINE
#endif

/* The OPENSSL_NO_* macros are also defined as NO_* if the application
   asks for it.  This is a transient feature that is provided for those
   who haven't had the time to do the appropriate changes in their
   applications.  */
#ifdef OPENSSL_ALGORITHM_DEFINES
# if defined(OPENSSL_NO_EC) && !defined(NO_EC)
#  define NO_EC
# endif
#endif /* ! codereview */
# if defined(OPENSSL_NO_EC_NISTP_64_GCC_128) && !defined(NO_EC_NISTP_64_GCC_128)
#  define NO_EC_NISTP_64_GCC_128
```

```
124 # endif
125 # if defined(OPENSSL_NO_ECDH) && !defined(NO_ECDH)
126 #  define NO_ECDH
127 # endif
128 # if defined(OPENSSL_NO_ECDSA) && !defined(NO_ECDSA)
129 #  define NO_ECDSA
130 # endif
131 #endif /* ! codereview */
132 # if defined(OPENSSL_NO_GMP) && !defined(NO_GMP)
133 #  define NO_GMP
134 # endif
135 # if defined(OPENSSL_NO_GOST) && !defined(NO_GOST)
136 #  define NO_GOST
137 # endif
138 # if defined(OPENSSL_NO_HW_4758_CCA) && !defined(NO_HW_4758_CCA)
139 #  define NO_HW_4758_CCA
140 # endif
141 # if defined(OPENSSL_NO_HW_AEP) && !defined(NO_HW_AEP)
142 #  define NO_HW_AEP
143 # endif
144 # if defined(OPENSSL_NO_HW_ATALLA) && !defined(NO_HW_ATALLA)
145 #  define NO_HW_ATALLA
146 # endif
147 # if defined(OPENSSL_NO_HW_CHIL) && !defined(NO_HW_CHIL)
148 #  define NO_HW_CHIL
149 # endif
150 # if defined(OPENSSL_NO_HW_CSWIFT) && !defined(NO_HW_CSWIFT)
151 #  define NO_HW_CSWIFT
152 # endif
153 # if defined(OPENSSL_NO_HW_GMP) && !defined(NO_HW_GMP)
154 #  define NO_HW_GMP
155 # endif
156 # if defined(OPENSSL_NO_HW_NCIPHER) && !defined(NO_HW_NCIPHER)
157 #  define NO_HW_NCIPHER
158 # endif
159 # if defined(OPENSSL_NO_HW_NURON) && !defined(NO_HW_NURON)
160 #  define NO_HW_NURON
161 # endif
162 # if defined(OPENSSL_NO_HW_PADLOCK) && !defined(NO_HW_PADLOCK)
163 #  define NO_HW_PADLOCK
164 # endif
165 # if defined(OPENSSL_NO_HW_SUREWARE) && !defined(NO_HW_SUREWARE)
166 #  define NO_HW_SUREWARE
167 # endif
168 # if defined(OPENSSL_NO_HW_UBSEC) && !defined(NO_HW_UBSEC)
169 #  define NO_HW_UBSEC
170 # endif
171 # if defined(OPENSSL_NO_IDEA) && !defined(NO_IDEA)
172 #  define NO_IDEA
173 # endif
174 #endif /* ! codereview */
175 # if defined(OPENSSL_NO_JPAKE) && !defined(NO_JPAKE)
176 #  define NO_JPAKE
177 # endif
178 # if defined(OPENSSL_NO_KRB5) && !defined(NO_KRB5)
179 #  define NO_KRB5
180 # endif
181 # if defined(OPENSSL_NO_MDC2) && !defined(NO_MDC2)
182 #  define NO_MDC2
183 # endif
184 # if defined(OPENSSL_NO_RC3) && !defined(NO_RC3)
185 #  define NO_RC3
 29 # if defined(OPENSSL_NO_MD2) && !defined(NO_MD2)
 30 #  define NO_MD2
186 # endif
187 # if defined(OPENSSL_NO_RC5) && !defined(NO_RC5)
```



```
188 #  define NO_RC5
189 # endif
190 # if defined(OPENSSL_NO_RFC3779) && !defined(NO_RFC3779)
191 #  define NO_RFC3779
192 # endif
193 # if defined(OPENSSL_NO_SCTP) && !defined(NO_SCTP)
194 #  define NO_SCTP
195 # endif
196 # if defined(OPENSSL_NO_SEED) && !defined(NO_SEED)
197 #  define NO_SEED
198 # endif
199 #endif /* ! codereview */
200 # if defined(OPENSSL_NO_STORE) && !defined(NO_STORE)
201 #  define NO_STORE
202 # endif
203 # if defined(OPENSSL_NO_UNIT_TEST) && !defined(NO_UNIT_TEST)
204 #  define NO_UNIT_TEST
205 # endif
206 # if defined(OPENSSL_NO_WHIRLPOOL) && !defined(NO_WHIRLPOOL)
207 #  define NO_WHIRLPOOL
208 # endif
209 # if defined(OPENSSL_NO_WHRLPOOL) && !defined(NO_WHRLPOOL)
210 #  define NO_WHRLPOOL
211 # endif
212 #endif /* ! codereview */
213 #endif

215 #define OPENSSL_CPUID_OBJ

217 #endif /* ! codereview */
218 /* crypto/opensslconf.h.in */

220 #include <openssl/sunw_prefix.h>

222 #endif /* ! codereview */
223 /* Generate 80386 code? */
224 #undef I386_ONLY

226 #if !(defined(VMS) || defined(__VMS)) /* VMS uses logical names instead */
227 #if defined(HEADER_CRYPTLIB_H) && !defined(OPENSSLDIR)
228 #if defined(__x86_64)
229 #define ENGINESDIR "/usr/lib/openssl/engines/64"
230 #else
231 #define ENGINESDIR "/usr/lib/openssl/engines"
232 #endif
233 #define OPENSSLDIR "/etc/openssl"
 41 #define ENGINESDIR "/usr/local/ssl/lib/engines"
 42 #define OPENSSLDIR "/usr/local/ssl"
234 #endif
235 #endif

237 #undef OPENSSL_UNISTD
238 #define OPENSSL_UNISTD <unistd.h>

240 #undef OPENSSL_EXPORT_VAR_AS_FUNCTION

242 #if defined(HEADER_IDEA_H) && !defined(IDEA_INT)
243 #define IDEA_INT unsigned int
244 #endif

246 #if defined(HEADER_MD2_H) && !defined(MD2_INT)
247 #define MD2_INT unsigned int
248 #endif

250 #if defined(HEADER_RC2_H) && !defined(RC2_INT)
251 /* I need to put in a mod for the alpha - eay */
```

```
252 #define RC2_INT unsigned int
253 #endif

255 #if defined(HEADER_RC4_H)
256 #if !defined(RC4_INT)
257 /* using int types make the structure larger but make the code faster
258  * on most boxes I have tested - up to %20 faster. */
259 /*
260  * I don't know what does "most" mean, but declaring "int" is a must on:
261  * - Intel P6 because partial register stalls are very expensive;
262  * - elder Alpha because it lacks byte load/store instructions;
263  */
264 #define RC4_INT unsigned int
265 #endif
266 #if !defined(RC4_CHUNK)
267 /*
268  * This enables code handling data aligned at natural CPU word
269  * boundary. See crypto/rc4/rc4_enc.c for further details.
270  */
271 #if defined(__x86_64)
272 #define RC4_CHUNK unsigned long
273 #else
274 #endif /* ! codereview */
275 #undef RC4_CHUNK
276 #endif  /* __x86_64 */
277 #endif /* ! codereview */
278 #endif
279 #endif

281 #if (defined(HEADER_NEW_DES_H) || defined(HEADER_DES_H)) && !defined(DES_LONG)
282 /* If this is set to 'unsigned int' on a DEC Alpha, this gives about a
283  * %20 speed up (longs are 8 bytes, int's are 4). */
284 #ifndef DES_LONG
285 #if defined(__x86_64)
286 #define DES_LONG unsigned int
287 #else
288 #endif /* ! codereview */
289 #define DES_LONG unsigned long
290 #endif
291 #endif
292 #endif
293 #endif /* ! codereview */

295 #if defined(HEADER_BN_H) && !defined(CONFIG_HEADER_BN_H)
296 #define CONFIG_HEADER_BN_H
297 #if defined(__x86_64)
298 #endif /* ! codereview */
299 #undef BN_LLONG
300 #else
301 #define BN_LLONG
302 #endif
303 #endif /* ! codereview */

305 /* Should we define BN_DIV2W here? */

307 /* Only one for the following should be defined */
308 #if defined(__x86_64)
309 #define SIXTY_FOUR_BIT_LONG
310 #undef THIRTY_TWO_BIT
311 #else
312 #endif /* ! codereview */
313 #undef SIXTY_FOUR_BIT_LONG
314 #undef SIXTY_FOUR_BIT
315 #define THIRTY_TWO_BIT
316 #endif
317 #undef SIXTY_FOUR_BIT
```



```
318 #endif
319 #endif /* ! codereview */

321 #if defined(HEADER_RC4_LOCL_H) && !defined(CONFIG_HEADER_RC4_LOCL_H)
322 #define CONFIG_HEADER_RC4_LOCL_H
323 /* if this is defined data[i] is used instead of *data, this is a %20
324  * speedup on x86 */
325 #if defined(__x86_64)
326 #endif /* ! codereview */
327 #undef  RC4_INDEX
328 #else
329 #define RC4_INDEX
330 #endif /* __x86_64 */
331 #endif /* ! codereview */
332 #endif

334 #if defined(HEADER_BF_LOCL_H) && !defined(CONFIG_HEADER_BF_LOCL_H)
335 #define CONFIG_HEADER_BF_LOCL_H
336 #define BF_PTR
 80 #undef BF_PTR
337 #endif /* HEADER_BF_LOCL_H */

339 #if defined(HEADER_DES_LOCL_H) && !defined(CONFIG_HEADER_DES_LOCL_H)
340 #define CONFIG_HEADER_DES_LOCL_H
341 #ifndef DES_DEFAULT_OPTIONS
342 /* the following is tweaked from a config script, that is why it is a
343  * protected undef/define */
344 #ifndef DES_PTR
345 #if defined(__x86_64)
346 #endif /* ! codereview */
347 #undef DES_PTR
348 #else
349 #define DES_PTR
350 #endif /* __x86_64 */
351 #endif /* ! codereview */
352 #endif

354 /* This helps C compiler generate the correct code for multiple functional
355  * units.  It reduces register dependancies at the expense of 2 more
356  * registers */
357 #ifndef DES_RISC1
358 #if defined(__x86_64)
359 #endif /* ! codereview */
360 #undef DES_RISC1
361 #else
362 #define        DES_RISC1
363 #endif /* __x86_64 */
364 #endif /* ! codereview */
365 #endif

367 #ifndef DES_RISC2
368 #undef DES_RISC2
369 #endif

371 #if defined(DES_RISC1) && defined(DES_RISC2)
372 YOU SHOULD NOT HAVE BOTH DES_RISC1 AND DES_RISC2 DEFINED!!!!!
373 #endif

375 /* Unroll the inner loop, this sometimes helps, sometimes hinders.
376  * Very mucy CPU dependant */
377 #ifndef DES_UNROLL
378 #define DES_UNROLL
 89 #undef DES_UNROLL
379 #endif

381 /* These default values were supplied by
```

```
 382  * Peter Gutman <pgut001@cs.auckland.ac.nz>
 383  * They are only used if nothing else has been defined */
 384 #if !defined(DES_PTR) && !defined(DES_RISC1) && !defined(DES_RISC2) && !defined(
 385 /* Special defines which change the way the code is built depending on the
 386    CPU and OS.  For SGI machines you can use _MIPS_SZLONG (32 or 64) to find
 387    even newer MIPS CPU's, but at the moment one size fits all for
 388    optimization options.  Older Sparc's work better with only UNROLL, but
 389    there's no way to tell at compile time what it is you're running on */

 391 #if defined( sun )              /* Newer Sparc's */
 392 #  define DES_PTR
 393 #  define DES_RISC1
 394 #  define DES_UNROLL
 395 #elif defined( __ultrix )       /* Older MIPS */
 396 #  define DES_PTR
 397 #  define DES_RISC2
 398 #  define DES_UNROLL
 399 #elif defined( __osf1__ )       /* Alpha */
 400 #  define DES_PTR
 401 #  define DES_RISC2
 402 #elif defined ( _AIX )          /* RS6000 */
 403   /* Unknown */
 404 #elif defined( __hpux )         /* HP-PA */
 405   /* Unknown */
 406 #elif defined( __aux )          /* 68K */
 407   /* Unknown */
 408 #elif defined( __dgux )         /* 88K (but P6 in latest boxes) */
 409 #  define DES_UNROLL
 410 #elif defined( __sgi )          /* Newer MIPS */
 411 #  define DES_PTR
 412 #  define DES_RISC2
 413 #  define DES_UNROLL
 414 #elif defined(i386) || defined(__i386__)        /* x86 boxes, should be gcc */
 415 #  define DES_PTR
 416 #  define DES_RISC1
 417 #  define DES_UNROLL
 418 #endif /* Systems-specific speed defines */
 419 #endif

 421 #endif /* DES_DEFAULT_OPTIONS */
 422 #endif /* HEADER_DES_LOCL_H */
```

```
**********************************************************
    9275 Wed Aug 13 19:32:54 2014
new/usr/src/lib/openssl/include/pkcs11.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
/* pkcs11.h include file for PKCS #11. */
/* $Revision: 1.4 $ */

/* License to copy and use this software is granted provided that it is
 * identified as "RSA Security Inc. PKCS #11 Cryptographic Token Interface
 * (Cryptoki)" in all material mentioning or referencing this software.

 * License is also granted to make and use derivative works provided that
 * such works are identified as "derived from the RSA Security Inc. PKCS #11
 * Cryptographic Token Interface (Cryptoki)" in all material mentioning or
 * referencing the derived work.

 * RSA Security Inc. makes no representations concerning either the
 * merchantability of this software or the suitability of this software for
 * any particular purpose. It is provided "as is" without express or implied
 * warranty of any kind.
 */

#ifndef _PKCS11_H_
#define _PKCS11_H_ 1

#ifdef __cplusplus
extern "C" {
#endif

/* Before including this file (pkcs11.h) (or pkcs11t.h by
 * itself), 6 platform-specific macros must be defined.  These
 * macros are described below, and typical definitions for them
 * are also given.  Be advised that these definitions can depend
 * on both the platform and the compiler used (and possibly also
 * on whether a Cryptoki library is linked statically or
 * dynamically).
 *
 * In addition to defining these 6 macros, the packing convention
 * for Cryptoki structures should be set.  The Cryptoki
 * convention on packing is that structures should be 1-byte
 * aligned.
 *
 * If you're using Microsoft Developer Studio 5.0 to produce
 * Win32 stuff, this might be done by using the following
 * preprocessor directive before including pkcs11.h or pkcs11t.h:
 *
 * #pragma pack(push, cryptoki, 1)
 *
 * and using the following preprocessor directive after including
 * pkcs11.h or pkcs11t.h:
 *
 * #pragma pack(pop, cryptoki)
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to produce Win16 stuff, this might be done by using
 * the following preprocessor directive before including
 * pkcs11.h or pkcs11t.h:
 *
 * #pragma pack(1)
 *
 * In a UNIX environment, you're on your own for this.  You might
 * not need to do (or be able to do!) anything.
 *
 *
 * Now for the macros:
```



```
 *
 *
 * 1. CK_PTR: The indirection string for making a pointer to an
 * object.  It can be used like this:
 *
 * typedef CK_BYTE CK_PTR CK_BYTE_PTR;
 *
 * If you're using Microsoft Developer Studio 5.0 to produce
 * Win32 stuff, it might be defined by:
 *
 * #define CK_PTR *
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to produce Win16 stuff, it might be defined by:
 *
 * #define CK_PTR far *
 *
 * In a typical UNIX environment, it might be defined by:
 *
 * #define CK_PTR *
 *
 *
 * 2. CK_DEFINE_FUNCTION(returnType, name): A macro which makes
 * an exportable Cryptoki library function definition out of a
 * return type and a function name.  It should be used in the
 * following fashion to define the exposed Cryptoki functions in
 * a Cryptoki library:
 *
 * CK_DEFINE_FUNCTION(CK_RV, C_Initialize)(
 *   CK_VOID_PTR pReserved
 * )
 * {
 *   ...
 * }
 *
 * If you're using Microsoft Developer Studio 5.0 to define a
 * function in a Win32 Cryptoki .dll, it might be defined by:
 *
 * #define CK_DEFINE_FUNCTION(returnType, name) \
 *   returnType __declspec(dllexport) name
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to define a function in a Win16 Cryptoki .dll, it
 * might be defined by:
 *
 * #define CK_DEFINE_FUNCTION(returnType, name) \
 *   returnType __export _far _pascal name
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_DEFINE_FUNCTION(returnType, name) \
 *   returnType name
 *
 *
 * 3. CK_DECLARE_FUNCTION(returnType, name): A macro which makes
 * an importable Cryptoki library function declaration out of a
 * return type and a function name.  It should be used in the
 * following fashion:
 *
 * extern CK_DECLARE_FUNCTION(CK_RV, C_Initialize)(
 *   CK_VOID_PTR pReserved
 * );
 *
 * If you're using Microsoft Developer Studio 5.0 to declare a
 * function in a Win32 Cryptoki .dll, it might be defined by:
 *
```

```
 * #define CK_DECLARE_FUNCTION(returnType, name) \
 *   returnType __declspec(dllimport) name
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to declare a function in a Win16 Cryptoki .dll, it
 * might be defined by:
 *
 * #define CK_DECLARE_FUNCTION(returnType, name) \
 *   returnType __export _far _pascal name
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_DECLARE_FUNCTION(returnType, name) \
 *   returnType name
 *
 *
 * 4. CK_DECLARE_FUNCTION_POINTER(returnType, name): A macro
 * which makes a Cryptoki API function pointer declaration or
 * function pointer type declaration out of a return type and a
 * function name.  It should be used in the following fashion:
 *
 * // Define funcPtr to be a pointer to a Cryptoki API function
 * // taking arguments args and returning CK_RV.
 * CK_DECLARE_FUNCTION_POINTER(CK_RV, funcPtr)(args);
 *
 * or
 *
 * // Define funcPtrType to be the type of a pointer to a
 * // Cryptoki API function taking arguments args and returning
 * // CK_RV, and then define funcPtr to be a variable of type
 * // funcPtrType.
 * typedef CK_DECLARE_FUNCTION_POINTER(CK_RV, funcPtrType)(args);
 * funcPtrType funcPtr;
 *
 * If you're using Microsoft Developer Studio 5.0 to access
 * functions in a Win32 Cryptoki .dll, in might be defined by:
 *
 * #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
 *   returnType __declspec(dllimport) (* name)
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to access functions in a Win16 Cryptoki .dll, it might
 * be defined by:
 *
 * #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
 *   returnType __export _far _pascal (* name)
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
 *   returnType (* name)
 *
 *
 * 5. CK_CALLBACK_FUNCTION(returnType, name): A macro which makes
 * a function pointer type for an application callback out of
 * a return type for the callback and a name for the callback.
 * It should be used in the following fashion:
 *
 * CK_CALLBACK_FUNCTION(CK_RV, myCallback)(args);
 *
 * to declare a function pointer, myCallback, to a callback
 * which takes arguments args and returns a CK_RV.  It can also
 * be used like this:
 *
 * typedef CK_CALLBACK_FUNCTION(CK_RV, myCallbackType)(args);
 * myCallbackType myCallback;
```



```
 *
 * If you're using Microsoft Developer Studio 5.0 to do Win32
 * Cryptoki development, it might be defined by:
 *
 * #define CK_CALLBACK_FUNCTION(returnType, name) \
 *   returnType (* name)
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to do Win16 development, it might be defined by:
 *
 * #define CK_CALLBACK_FUNCTION(returnType, name) \
 *   returnType _far _pascal (* name)
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_CALLBACK_FUNCTION(returnType, name) \
 *   returnType (* name)
 *
 *
 * 6. NULL_PTR: This macro is the value of a NULL pointer.
 *
 * In any ANSI/ISO C environment (and in many others as well),
 * this should best be defined by
 *
 * #ifndef NULL_PTR
 * #define NULL_PTR 0
 * #endif
 */


/* All the various Cryptoki types and #define'd values are in the
 * file pkcs11t.h. */
#include "pkcs11t.h"

#define __PASTE(x,y)      x##y


/* ==============================================================
 * Define the "extern" form of all the entry points.
 * ==============================================================
 */

#define CK_NEED_ARG_LIST  1
#define CK_PKCS11_FUNCTION_INFO(name) \
  extern CK_DECLARE_FUNCTION(CK_RV, name)

/* pkcs11f.h has all the information about the Cryptoki
 * function prototypes. */
#include "pkcs11f.h"

#undef CK_NEED_ARG_LIST
#undef CK_PKCS11_FUNCTION_INFO


/* ==============================================================
 * Define the typedef form of all the entry points.  That is, for
 * each Cryptoki function C_XXX, define a type CK_C_XXX which is
 * a pointer to that kind of function.
 * ==============================================================
 */

#define CK_NEED_ARG_LIST  1
#define CK_PKCS11_FUNCTION_INFO(name) \
  typedef CK_DECLARE_FUNCTION_POINTER(CK_RV, __PASTE(CK_,name))

/* pkcs11f.h has all the information about the Cryptoki
```

```
 * function prototypes. */
#include "pkcs11f.h"

#undef CK_NEED_ARG_LIST
#undef CK_PKCS11_FUNCTION_INFO


/* ==============================================================
 * Define structed vector of entry points.  A CK_FUNCTION_LIST
 * contains a CK_VERSION indicating a library's Cryptoki version
 * and then a whole slew of function pointers to the routines in
 * the library.  This type was declared, but not defined, in
 * pkcs11t.h.
 * ==============================================================
 */

#define CK_PKCS11_FUNCTION_INFO(name) \
  __PASTE(CK_,name) name;

struct CK_FUNCTION_LIST {

  CK_VERSION    version;  /* Cryptoki version */

/* Pile all the function pointers into the CK_FUNCTION_LIST. */
/* pkcs11f.h has all the information about the Cryptoki
 * function prototypes. */
#include "pkcs11f.h"

};

#undef CK_PKCS11_FUNCTION_INFO


#undef __PASTE

#ifdef __cplusplus
}
#endif

#endif
#endif /* ! codereview */
```

```
**********************************************************
   28338 Wed Aug 13 19:32:54 2014
new/usr/src/lib/openssl/include/pkcs11f.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* pkcs11f.h include file for PKCS #11. */
/* $Revision: 1.4 $ */

/* License to copy and use this software is granted provided that it is
 * identified as "RSA Security Inc. PKCS #11 Cryptographic Token Interface
 * (Cryptoki)" in all material mentioning or referencing this software.

 * License is also granted to make and use derivative works provided that
 * such works are identified as "derived from the RSA Security Inc. PKCS #11
 * Cryptographic Token Interface (Cryptoki)" in all material mentioning or
 * referencing the derived work.

 * RSA Security Inc. makes no representations concerning either the
 * merchantability of this software or the suitability of this software for
 * any particular purpose. It is provided "as is" without express or implied
 * warranty of any kind.
 */

/* This header file contains pretty much everything about all the */
/* Cryptoki function prototypes.  Because this information is */
/* used for more than just declaring function prototypes, the */
/* order of the functions appearing herein is important, and */
/* should not be altered. */

/* General-purpose */

/* C_Initialize initializes the Cryptoki library. */
CK_PKCS11_FUNCTION_INFO(C_Initialize)
#ifdef CK_NEED_ARG_LIST
(
  CK_VOID_PTR   pInitArgs  /* if this is not NULL_PTR, it gets
                            * cast to CK_C_INITIALIZE_ARGS_PTR
                            * and dereferenced */
);
#endif


/* C_Finalize indicates that an application is done with the
 * Cryptoki library. */
CK_PKCS11_FUNCTION_INFO(C_Finalize)
#ifdef CK_NEED_ARG_LIST
(
  CK_VOID_PTR   pReserved  /* reserved.  Should be NULL_PTR */
);
#endif


/* C_GetInfo returns general information about Cryptoki. */
CK_PKCS11_FUNCTION_INFO(C_GetInfo)
#ifdef CK_NEED_ARG_LIST
(
  CK_INFO_PTR   pInfo  /* location that receives information */
);
#endif


/* C_GetFunctionList returns the function list. */
CK_PKCS11_FUNCTION_INFO(C_GetFunctionList)
#ifdef CK_NEED_ARG_LIST
(
  CK_FUNCTION_LIST_PTR_PTR ppFunctionList  /* receives pointer to
```



```
                                           * function list */
);
#endif



/* Slot and token management */

/* C_GetSlotList obtains a list of slots in the system. */
CK_PKCS11_FUNCTION_INFO(C_GetSlotList)
#ifdef CK_NEED_ARG_LIST
(
  CK_BBOOL       tokenPresent,  /* only slots with tokens? */
  CK_SLOT_ID_PTR pSlotList,     /* receives array of slot IDs */
  CK_ULONG_PTR   pulCount       /* receives number of slots */
);
#endif


/* C_GetSlotInfo obtains information about a particular slot in
 * the system. */
CK_PKCS11_FUNCTION_INFO(C_GetSlotInfo)
#ifdef CK_NEED_ARG_LIST
(
  CK_SLOT_ID       slotID,  /* the ID of the slot */
  CK_SLOT_INFO_PTR pInfo    /* receives the slot information */
);
#endif


/* C_GetTokenInfo obtains information about a particular token
 * in the system. */
CK_PKCS11_FUNCTION_INFO(C_GetTokenInfo)
#ifdef CK_NEED_ARG_LIST
(
  CK_SLOT_ID        slotID,  /* ID of the token's slot */
  CK_TOKEN_INFO_PTR pInfo    /* receives the token information */
);
#endif


/* C_GetMechanismList obtains a list of mechanism types
 * supported by a token. */
CK_PKCS11_FUNCTION_INFO(C_GetMechanismList)
#ifdef CK_NEED_ARG_LIST
(
  CK_SLOT_ID            slotID,          /* ID of token's slot */
  CK_MECHANISM_TYPE_PTR pMechanismList,  /* gets mech. array */
  CK_ULONG_PTR          pulCount         /* gets # of mechs. */
);
#endif


/* C_GetMechanismInfo obtains information about a particular
 * mechanism possibly supported by a token. */
CK_PKCS11_FUNCTION_INFO(C_GetMechanismInfo)
#ifdef CK_NEED_ARG_LIST
(
  CK_SLOT_ID            slotID,  /* ID of the token's slot */
  CK_MECHANISM_TYPE     type,    /* type of mechanism */
  CK_MECHANISM_INFO_PTR pInfo    /* receives mechanism info */
);
#endif


/* C_InitToken initializes a token. */
```

```
128 CK_PKCS11_FUNCTION_INFO(C_InitToken)
129 #ifdef CK_NEED_ARG_LIST
130 /* pLabel changed from CK_CHAR_PTR to CK_UTF8CHAR_PTR for v2.10 */
131 (
132   CK_SLOT_ID      slotID,    /* ID of the token's slot */
133   CK_UTF8CHAR_PTR pPin,      /* the SO's initial PIN */
134   CK_ULONG        ulPinLen,  /* length in bytes of the PIN */
135   CK_UTF8CHAR_PTR pLabel     /* 32-byte token label (blank padded) */
136 );
137 #endif


140 /* C_InitPIN initializes the normal user's PIN. */
141 CK_PKCS11_FUNCTION_INFO(C_InitPIN)
142 #ifdef CK_NEED_ARG_LIST
143 (
144   CK_SESSION_HANDLE hSession,  /* the session's handle */
145   CK_UTF8CHAR_PTR   pPin,      /* the normal user's PIN */
146   CK_ULONG          ulPinLen   /* length in bytes of the PIN */
147 );
148 #endif


151 /* C_SetPIN modifies the PIN of the user who is logged in. */
152 CK_PKCS11_FUNCTION_INFO(C_SetPIN)
153 #ifdef CK_NEED_ARG_LIST
154 (
155   CK_SESSION_HANDLE hSession,  /* the session's handle */
156   CK_UTF8CHAR_PTR   pOldPin,   /* the old PIN */
157   CK_ULONG          ulOldLen,  /* length of the old PIN */
158   CK_UTF8CHAR_PTR   pNewPin,   /* the new PIN */
159   CK_ULONG          ulNewLen   /* length of the new PIN */
160 );
161 #endif



165 /* Session management */

167 /* C_OpenSession opens a session between an application and a
168  * token. */
169 CK_PKCS11_FUNCTION_INFO(C_OpenSession)
170 #ifdef CK_NEED_ARG_LIST
171 (
172   CK_SLOT_ID            slotID,        /* the slot's ID */
173   CK_FLAGS              flags,         /* from CK_SESSION_INFO */
174   CK_VOID_PTR           pApplication,  /* passed to callback */
175   CK_NOTIFY             Notify,        /* callback function */
176   CK_SESSION_HANDLE_PTR phSession      /* gets session handle */
177 );
178 #endif


181 /* C_CloseSession closes a session between an application and a
182  * token. */
183 CK_PKCS11_FUNCTION_INFO(C_CloseSession)
184 #ifdef CK_NEED_ARG_LIST
185 (
186   CK_SESSION_HANDLE hSession  /* the session's handle */
187 );
188 #endif


191 /* C_CloseAllSessions closes all sessions with a token. */
192 CK_PKCS11_FUNCTION_INFO(C_CloseAllSessions)
193 #ifdef CK_NEED_ARG_LIST
```



```
194 (
195   CK_SLOT_ID     slotID  /* the token's slot */
196 );
197 #endif


200 /* C_GetSessionInfo obtains information about the session. */
201 CK_PKCS11_FUNCTION_INFO(C_GetSessionInfo)
202 #ifdef CK_NEED_ARG_LIST
203 (
204   CK_SESSION_HANDLE   hSession,  /* the session's handle */
205   CK_SESSION_INFO_PTR pInfo      /* receives session info */
206 );
207 #endif


210 /* C_GetOperationState obtains the state of the cryptographic operation
211  * in a session. */
212 CK_PKCS11_FUNCTION_INFO(C_GetOperationState)
213 #ifdef CK_NEED_ARG_LIST
214 (
215   CK_SESSION_HANDLE hSession,             /* session's handle */
216   CK_BYTE_PTR       pOperationState,      /* gets state */
217   CK_ULONG_PTR      pulOperationStateLen  /* gets state length */
218 );
219 #endif


222 /* C_SetOperationState restores the state of the cryptographic
223  * operation in a session. */
224 CK_PKCS11_FUNCTION_INFO(C_SetOperationState)
225 #ifdef CK_NEED_ARG_LIST
226 (
227   CK_SESSION_HANDLE hSession,            /* session's handle */
228   CK_BYTE_PTR      pOperationState,      /* holds state */
229   CK_ULONG         ulOperationStateLen,  /* holds state length */
230   CK_OBJECT_HANDLE hEncryptionKey,       /* en/decryption key */
231   CK_OBJECT_HANDLE hAuthenticationKey    /* sign/verify key */
232 );
233 #endif


236 /* C_Login logs a user into a token. */
237 CK_PKCS11_FUNCTION_INFO(C_Login)
238 #ifdef CK_NEED_ARG_LIST
239 (
240   CK_SESSION_HANDLE hSession,  /* the session's handle */
241   CK_USER_TYPE      userType,  /* the user type */
242   CK_UTF8CHAR_PTR   pPin,      /* the user's PIN */
243   CK_ULONG          ulPinLen   /* the length of the PIN */
244 );
245 #endif


248 /* C_Logout logs a user out from a token. */
249 CK_PKCS11_FUNCTION_INFO(C_Logout)
250 #ifdef CK_NEED_ARG_LIST
251 (
252   CK_SESSION_HANDLE hSession  /* the session's handle */
253 );
254 #endif



258 /* Object management */
```

```
260 /* C_CreateObject creates a new object. */
261 CK_PKCS11_FUNCTION_INFO(C_CreateObject)
262 #ifdef CK_NEED_ARG_LIST
263 (
264   CK_SESSION_HANDLE hSession,    /* the session's handle */
265   CK_ATTRIBUTE_PTR  pTemplate,   /* the object's template */
266   CK_ULONG          ulCount,     /* attributes in template */
267   CK_OBJECT_HANDLE_PTR phObject  /* gets new object's handle. */
268 );
269 #endif


272 /* C_CopyObject copies an object, creating a new object for the
273  * copy. */
274 CK_PKCS11_FUNCTION_INFO(C_CopyObject)
275 #ifdef CK_NEED_ARG_LIST
276 (
277   CK_SESSION_HANDLE    hSession,    /* the session's handle */
278   CK_OBJECT_HANDLE     hObject,     /* the object's handle */
279   CK_ATTRIBUTE_PTR     pTemplate,   /* template for new object */
280   CK_ULONG             ulCount,     /* attributes in template */
281   CK_OBJECT_HANDLE_PTR phNewObject  /* receives handle of copy */
282 );
283 #endif


286 /* C_DestroyObject destroys an object. */
287 CK_PKCS11_FUNCTION_INFO(C_DestroyObject)
288 #ifdef CK_NEED_ARG_LIST
289 (
290   CK_SESSION_HANDLE hSession,  /* the session's handle */
291   CK_OBJECT_HANDLE  hObject    /* the object's handle */
292 );
293 #endif


296 /* C_GetObjectSize gets the size of an object in bytes. */
297 CK_PKCS11_FUNCTION_INFO(C_GetObjectSize)
298 #ifdef CK_NEED_ARG_LIST
299 (
300   CK_SESSION_HANDLE hSession,  /* the session's handle */
301   CK_OBJECT_HANDLE  hObject,   /* the object's handle */
302   CK_ULONG_PTR      pulSize    /* receives size of object */
303 );
304 #endif


307 /* C_GetAttributeValue obtains the value of one or more object
308  * attributes. */
309 CK_PKCS11_FUNCTION_INFO(C_GetAttributeValue)
310 #ifdef CK_NEED_ARG_LIST
311 (
312   CK_SESSION_HANDLE hSession,   /* the session's handle */
313   CK_OBJECT_HANDLE  hObject,    /* the object's handle */
314   CK_ATTRIBUTE_PTR  pTemplate,  /* specifies attrs; gets vals */
315   CK_ULONG          ulCount     /* attributes in template */
316 );
317 #endif


320 /* C_SetAttributeValue modifies the value of one or more object
321  * attributes */
322 CK_PKCS11_FUNCTION_INFO(C_SetAttributeValue)
323 #ifdef CK_NEED_ARG_LIST
324 (
325   CK_SESSION_HANDLE hSession,   /* the session's handle */
```



```
326   CK_OBJECT_HANDLE  hObject,    /* the object's handle */
327   CK_ATTRIBUTE_PTR  pTemplate,  /* specifies attrs and values */
328   CK_ULONG          ulCount     /* attributes in template */
329 );
330 #endif


333 /* C_FindObjectsInit initializes a search for token and session
334  * objects that match a template. */
335 CK_PKCS11_FUNCTION_INFO(C_FindObjectsInit)
336 #ifdef CK_NEED_ARG_LIST
337 (
338   CK_SESSION_HANDLE hSession,   /* the session's handle */
339   CK_ATTRIBUTE_PTR  pTemplate,  /* attribute values to match */
340   CK_ULONG          ulCount     /* attrs in search template */
341 );
342 #endif


345 /* C_FindObjects continues a search for token and session
346  * objects that match a template, obtaining additional object
347  * handles. */
348 CK_PKCS11_FUNCTION_INFO(C_FindObjects)
349 #ifdef CK_NEED_ARG_LIST
350 (
351  CK_SESSION_HANDLE    hSession,          /* session's handle */
352  CK_OBJECT_HANDLE_PTR phObject,          /* gets obj. handles */
353  CK_ULONG             ulMaxObjectCount,  /* max handles to get */
354  CK_ULONG_PTR         pulObjectCount     /* actual # returned */
355 );
356 #endif


359 /* C_FindObjectsFinal finishes a search for token and session
360  * objects. */
361 CK_PKCS11_FUNCTION_INFO(C_FindObjectsFinal)
362 #ifdef CK_NEED_ARG_LIST
363 (
364   CK_SESSION_HANDLE hSession  /* the session's handle */
365 );
366 #endif



370 /* Encryption and decryption */

372 /* C_EncryptInit initializes an encryption operation. */
373 CK_PKCS11_FUNCTION_INFO(C_EncryptInit)
374 #ifdef CK_NEED_ARG_LIST
375 (
376   CK_SESSION_HANDLE hSession,    /* the session's handle */
377   CK_MECHANISM_PTR  pMechanism,  /* the encryption mechanism */
378   CK_OBJECT_HANDLE  hKey         /* handle of encryption key */
379 );
380 #endif


383 /* C_Encrypt encrypts single-part data. */
384 CK_PKCS11_FUNCTION_INFO(C_Encrypt)
385 #ifdef CK_NEED_ARG_LIST
386 (
387   CK_SESSION_HANDLE hSession,            /* session's handle */
388   CK_BYTE_PTR       pData,               /* the plaintext data */
389   CK_ULONG          ulDataLen,           /* bytes of plaintext */
390   CK_BYTE_PTR       pEncryptedData,      /* gets ciphertext */
391   CK_ULONG_PTR      pulEncryptedDataLen  /* gets c-text size */
```

```
392 );
393 #endif


396 /* C_EncryptUpdate continues a multiple-part encryption
397  * operation. */
398 CK_PKCS11_FUNCTION_INFO(C_EncryptUpdate)
399 #ifdef CK_NEED_ARG_LIST
400 (
401   CK_SESSION_HANDLE hSession,           /* session's handle */
402   CK_BYTE_PTR       pPart,              /* the plaintext data */
403   CK_ULONG          ulPartLen,          /* plaintext data len */
404   CK_BYTE_PTR       pEncryptedPart,     /* gets ciphertext */
405   CK_ULONG_PTR      pulEncryptedPartLen /* gets c-text size */
406 );
407 #endif


410 /* C_EncryptFinal finishes a multiple-part encryption
411  * operation. */
412 CK_PKCS11_FUNCTION_INFO(C_EncryptFinal)
413 #ifdef CK_NEED_ARG_LIST
414 (
415   CK_SESSION_HANDLE hSession,                /* session handle */
416   CK_BYTE_PTR       pLastEncryptedPart,      /* last c-text */
417   CK_ULONG_PTR      pulLastEncryptedPartLen  /* gets last size */
418 );
419 #endif


422 /* C_DecryptInit initializes a decryption operation. */
423 CK_PKCS11_FUNCTION_INFO(C_DecryptInit)
424 #ifdef CK_NEED_ARG_LIST
425 (
426   CK_SESSION_HANDLE hSession,    /* the session's handle */
427   CK_MECHANISM_PTR  pMechanism,  /* the decryption mechanism */
428   CK_OBJECT_HANDLE  hKey         /* handle of decryption key */
429 );
430 #endif


433 /* C_Decrypt decrypts encrypted data in a single part. */
434 CK_PKCS11_FUNCTION_INFO(C_Decrypt)
435 #ifdef CK_NEED_ARG_LIST
436 (
437   CK_SESSION_HANDLE hSession,           /* session's handle */
438   CK_BYTE_PTR       pEncryptedData,     /* ciphertext */
439   CK_ULONG          ulEncryptedDataLen, /* ciphertext length */
440   CK_BYTE_PTR       pData,              /* gets plaintext */
441   CK_ULONG_PTR      pulDataLen          /* gets p-text size */
442 );
443 #endif


446 /* C_DecryptUpdate continues a multiple-part decryption
447  * operation. */
448 CK_PKCS11_FUNCTION_INFO(C_DecryptUpdate)
449 #ifdef CK_NEED_ARG_LIST
450 (
451   CK_SESSION_HANDLE hSession,            /* session's handle */
452   CK_BYTE_PTR       pEncryptedPart,      /* encrypted data */
453   CK_ULONG          ulEncryptedPartLen,  /* input length */
454   CK_BYTE_PTR       pPart,               /* gets plaintext */
455   CK_ULONG_PTR      pulPartLen           /* p-text size */
456 );
457 #endif
```



```


460 /* C_DecryptFinal finishes a multiple-part decryption
461  * operation. */
462 CK_PKCS11_FUNCTION_INFO(C_DecryptFinal)
463 #ifdef CK_NEED_ARG_LIST
464 (
465   CK_SESSION_HANDLE hSession,       /* the session's handle */
466   CK_BYTE_PTR       pLastPart,      /* gets plaintext */
467   CK_ULONG_PTR      pulLastPartLen  /* p-text size */
468 );
469 #endif



473 /* Message digesting */

475 /* C_DigestInit initializes a message-digesting operation. */
476 CK_PKCS11_FUNCTION_INFO(C_DigestInit)
477 #ifdef CK_NEED_ARG_LIST
478 (
479   CK_SESSION_HANDLE hSession,   /* the session's handle */
480   CK_MECHANISM_PTR  pMechanism  /* the digesting mechanism */
481 );
482 #endif


485 /* C_Digest digests data in a single part. */
486 CK_PKCS11_FUNCTION_INFO(C_Digest)
487 #ifdef CK_NEED_ARG_LIST
488 (
489   CK_SESSION_HANDLE hSession,     /* the session's handle */
490   CK_BYTE_PTR       pData,        /* data to be digested */
491   CK_ULONG          ulDataLen,    /* bytes of data to digest */
492   CK_BYTE_PTR       pDigest,      /* gets the message digest */
493   CK_ULONG_PTR      pulDigestLen  /* gets digest length */
494 );
495 #endif


498 /* C_DigestUpdate continues a multiple-part message-digesting
499  * operation. */
500 CK_PKCS11_FUNCTION_INFO(C_DigestUpdate)
501 #ifdef CK_NEED_ARG_LIST
502 (
503   CK_SESSION_HANDLE hSession,  /* the session's handle */
504   CK_BYTE_PTR       pPart,     /* data to be digested */
505   CK_ULONG          ulPartLen  /* bytes of data to be digested */
506 );
507 #endif


510 /* C_DigestKey continues a multi-part message-digesting
511  * operation, by digesting the value of a secret key as part of
512  * the data already digested. */
513 CK_PKCS11_FUNCTION_INFO(C_DigestKey)
514 #ifdef CK_NEED_ARG_LIST
515 (
516   CK_SESSION_HANDLE hSession,  /* the session's handle */
517   CK_OBJECT_HANDLE  hKey       /* secret key to digest */
518 );
519 #endif


522 /* C_DigestFinal finishes a multiple-part message-digesting
523  * operation. */
```

```
524 CK_PKCS11_FUNCTION_INFO(C_DigestFinal)
525 #ifdef CK_NEED_ARG_LIST
526 (
527   CK_SESSION_HANDLE hSession,     /* the session's handle */
528   CK_BYTE_PTR       pDigest,      /* gets the message digest */
529   CK_ULONG_PTR      pulDigestLen  /* gets byte count of digest */
530 );
531 #endif



535 /* Signing and MACing */

537 /* C_SignInit initializes a signature (private key encryption)
538  * operation, where the signature is (will be) an appendix to
539  * the data, and plaintext cannot be recovered from the
540  *signature. */
541 CK_PKCS11_FUNCTION_INFO(C_SignInit)
542 #ifdef CK_NEED_ARG_LIST
543 (
544   CK_SESSION_HANDLE hSession,    /* the session's handle */
545   CK_MECHANISM_PTR  pMechanism,  /* the signature mechanism */
546   CK_OBJECT_HANDLE  hKey         /* handle of signature key */
547 );
548 #endif


551 /* C_Sign signs (encrypts with private key) data in a single
552  * part, where the signature is (will be) an appendix to the
553  * data, and plaintext cannot be recovered from the signature. */
554 CK_PKCS11_FUNCTION_INFO(C_Sign)
555 #ifdef CK_NEED_ARG_LIST
556 (
557   CK_SESSION_HANDLE hSession,        /* the session's handle */
558   CK_BYTE_PTR       pData,           /* the data to sign */
559   CK_ULONG          ulDataLen,       /* count of bytes to sign */
560   CK_BYTE_PTR       pSignature,      /* gets the signature */
561   CK_ULONG_PTR      pulSignatureLen  /* gets signature length */
562 );
563 #endif


566 /* C_SignUpdate continues a multiple-part signature operation,
567  * where the signature is (will be) an appendix to the data,
568  * and plaintext cannot be recovered from the signature. */
569 CK_PKCS11_FUNCTION_INFO(C_SignUpdate)
570 #ifdef CK_NEED_ARG_LIST
571 (
572   CK_SESSION_HANDLE hSession,  /* the session's handle */
573   CK_BYTE_PTR       pPart,     /* the data to sign */
574   CK_ULONG          ulPartLen  /* count of bytes to sign */
575 );
576 #endif


579 /* C_SignFinal finishes a multiple-part signature operation,
580  * returning the signature. */
581 CK_PKCS11_FUNCTION_INFO(C_SignFinal)
582 #ifdef CK_NEED_ARG_LIST
583 (
584   CK_SESSION_HANDLE hSession,        /* the session's handle */
585   CK_BYTE_PTR       pSignature,      /* gets the signature */
586   CK_ULONG_PTR      pulSignatureLen  /* gets signature length */
587 );
588 #endif
```



```
591 /* C_SignRecoverInit initializes a signature operation, where
592  * the data can be recovered from the signature. */
593 CK_PKCS11_FUNCTION_INFO(C_SignRecoverInit)
594 #ifdef CK_NEED_ARG_LIST
595 (
596   CK_SESSION_HANDLE hSession,   /* the session's handle */
597   CK_MECHANISM_PTR  pMechanism, /* the signature mechanism */
598   CK_OBJECT_HANDLE  hKey        /* handle of the signature key */
599 );
600 #endif


603 /* C_SignRecover signs data in a single operation, where the
604  * data can be recovered from the signature. */
605 CK_PKCS11_FUNCTION_INFO(C_SignRecover)
606 #ifdef CK_NEED_ARG_LIST
607 (
608   CK_SESSION_HANDLE hSession,        /* the session's handle */
609   CK_BYTE_PTR       pData,           /* the data to sign */
610   CK_ULONG          ulDataLen,       /* count of bytes to sign */
611   CK_BYTE_PTR       pSignature,      /* gets the signature */
612   CK_ULONG_PTR      pulSignatureLen  /* gets signature length */
613 );
614 #endif



618 /* Verifying signatures and MACs */

620 /* C_VerifyInit initializes a verification operation, where the
621  * signature is an appendix to the data, and plaintext cannot
622  *  cannot be recovered from the signature (e.g. DSA). */
623 CK_PKCS11_FUNCTION_INFO(C_VerifyInit)
624 #ifdef CK_NEED_ARG_LIST
625 (
626   CK_SESSION_HANDLE hSession,    /* the session's handle */
627   CK_MECHANISM_PTR  pMechanism,  /* the verification mechanism */
628   CK_OBJECT_HANDLE  hKey         /* verification key */
629 );
630 #endif


633 /* C_Verify verifies a signature in a single-part operation,
634  * where the signature is an appendix to the data, and plaintext
635  * cannot be recovered from the signature. */
636 CK_PKCS11_FUNCTION_INFO(C_Verify)
637 #ifdef CK_NEED_ARG_LIST
638 (
639   CK_SESSION_HANDLE hSession,       /* the session's handle */
640   CK_BYTE_PTR       pData,          /* signed data */
641   CK_ULONG          ulDataLen,      /* length of signed data */
642   CK_BYTE_PTR       pSignature,     /* signature */
643   CK_ULONG          ulSignatureLen  /* signature length*/
644 );
645 #endif


648 /* C_VerifyUpdate continues a multiple-part verification
649  * operation, where the signature is an appendix to the data,
650  * and plaintext cannot be recovered from the signature. */
651 CK_PKCS11_FUNCTION_INFO(C_VerifyUpdate)
652 #ifdef CK_NEED_ARG_LIST
653 (
654   CK_SESSION_HANDLE hSession,  /* the session's handle */
655   CK_BYTE_PTR       pPart,     /* signed data */
```

```
656   CK_ULONG          ulPartLen  /* length of signed data */
657 );
658 #endif


661 /* C_VerifyFinal finishes a multiple-part verification
662  * operation, checking the signature. */
663 CK_PKCS11_FUNCTION_INFO(C_VerifyFinal)
664 #ifdef CK_NEED_ARG_LIST
665 (
666   CK_SESSION_HANDLE hSession,       /* the session's handle */
667   CK_BYTE_PTR       pSignature,     /* signature to verify */
668   CK_ULONG          ulSignatureLen  /* signature length */
669 );
670 #endif


673 /* C_VerifyRecoverInit initializes a signature verification
674  * operation, where the data is recovered from the signature. */
675 CK_PKCS11_FUNCTION_INFO(C_VerifyRecoverInit)
676 #ifdef CK_NEED_ARG_LIST
677 (
678   CK_SESSION_HANDLE hSession,    /* the session's handle */
679   CK_MECHANISM_PTR  pMechanism,  /* the verification mechanism */
680   CK_OBJECT_HANDLE  hKey         /* verification key */
681 );
682 #endif


685 /* C_VerifyRecover verifies a signature in a single-part
686  * operation, where the data is recovered from the signature. */
687 CK_PKCS11_FUNCTION_INFO(C_VerifyRecover)
688 #ifdef CK_NEED_ARG_LIST
689 (
690   CK_SESSION_HANDLE hSession,        /* the session's handle */
691   CK_BYTE_PTR       pSignature,      /* signature to verify */
692   CK_ULONG          ulSignatureLen,  /* signature length */
693   CK_BYTE_PTR       pData,           /* gets signed data */
694   CK_ULONG_PTR      pulDataLen       /* gets signed data len */
695 );
696 #endif



700 /* Dual-function cryptographic operations */

702 /* C_DigestEncryptUpdate continues a multiple-part digesting
703  * and encryption operation. */
704 CK_PKCS11_FUNCTION_INFO(C_DigestEncryptUpdate)
705 #ifdef CK_NEED_ARG_LIST
706 (
707   CK_SESSION_HANDLE hSession,            /* session's handle */
708   CK_BYTE_PTR       pPart,               /* the plaintext data */
709   CK_ULONG          ulPartLen,           /* plaintext length */
710   CK_BYTE_PTR       pEncryptedPart,      /* gets ciphertext */
711   CK_ULONG_PTR      pulEncryptedPartLen  /* gets c-text length */
712 );
713 #endif


716 /* C_DecryptDigestUpdate continues a multiple-part decryption and
717  * digesting operation. */
718 CK_PKCS11_FUNCTION_INFO(C_DecryptDigestUpdate)
719 #ifdef CK_NEED_ARG_LIST
720 (
721   CK_SESSION_HANDLE hSession,            /* session's handle */
```



```
722   CK_BYTE_PTR       pEncryptedPart,      /* ciphertext */
723   CK_ULONG          ulEncryptedPartLen,  /* ciphertext length */
724   CK_BYTE_PTR       pPart,               /* gets plaintext */
725   CK_ULONG_PTR      pulPartLen           /* gets plaintext len */
726 );
727 #endif


730 /* C_SignEncryptUpdate continues a multiple-part signing and
731  * encryption operation. */
732 CK_PKCS11_FUNCTION_INFO(C_SignEncryptUpdate)
733 #ifdef CK_NEED_ARG_LIST
734 (
735   CK_SESSION_HANDLE hSession,            /* session's handle */
736   CK_BYTE_PTR       pPart,               /* the plaintext data */
737   CK_ULONG          ulPartLen,           /* plaintext length */
738   CK_BYTE_PTR       pEncryptedPart,      /* gets ciphertext */
739   CK_ULONG_PTR      pulEncryptedPartLen  /* gets c-text length */
740 );
741 #endif


744 /* C_DecryptVerifyUpdate continues a multiple-part decryption and
745  * verify operation. */
746 CK_PKCS11_FUNCTION_INFO(C_DecryptVerifyUpdate)
747 #ifdef CK_NEED_ARG_LIST
748 (
749   CK_SESSION_HANDLE hSession,            /* session's handle */
750   CK_BYTE_PTR       pEncryptedPart,      /* ciphertext */
751   CK_ULONG          ulEncryptedPartLen,  /* ciphertext length */
752   CK_BYTE_PTR       pPart,               /* gets plaintext */
753   CK_ULONG_PTR      pulPartLen           /* gets p-text length */
754 );
755 #endif



759 /* Key management */

761 /* C_GenerateKey generates a secret key, creating a new key
762  * object. */
763 CK_PKCS11_FUNCTION_INFO(C_GenerateKey)
764 #ifdef CK_NEED_ARG_LIST
765 (
766   CK_SESSION_HANDLE    hSession,    /* the session's handle */
767   CK_MECHANISM_PTR     pMechanism,  /* key generation mech. */
768   CK_ATTRIBUTE_PTR     pTemplate,   /* template for new key */
769   CK_ULONG             ulCount,     /* # of attrs in template */
770   CK_OBJECT_HANDLE_PTR phKey        /* gets handle of new key */
771 );
772 #endif


775 /* C_GenerateKeyPair generates a public-key/private-key pair,
776  * creating new key objects. */
777 CK_PKCS11_FUNCTION_INFO(C_GenerateKeyPair)
778 #ifdef CK_NEED_ARG_LIST
779 (
780   CK_SESSION_HANDLE    hSession,                    /* session
781                                                     * handle */
782   CK_MECHANISM_PTR     pMechanism,                  /* key-gen
783                                                     * mech. */
784   CK_ATTRIBUTE_PTR     pPublicKeyTemplate,          /* template
785                                                     * for pub.
786                                                     * key */
787   CK_ULONG             ulPublicKeyAttributeCount,   /* # pub.
```

```
788                                                 * attrs. */
789   CK_ATTRIBUTE_PTR     pPrivateKeyTemplate,     /* template
790                                                 * for priv.
791                                                 * key */
792   CK_ULONG             ulPrivateKeyAttributeCount,  /* # priv.
793                                                 * attrs. */
794   CK_OBJECT_HANDLE_PTR phPublicKey,             /* gets pub.
795                                                 * key
796                                                 * handle */
797   CK_OBJECT_HANDLE_PTR phPrivateKey             /* gets
798                                                 * priv. key
799                                                 * handle */
800 );
801 #endif


804 /* C_WrapKey wraps (i.e., encrypts) a key. */
805 CK_PKCS11_FUNCTION_INFO(C_WrapKey)
806 #ifdef CK_NEED_ARG_LIST
807 (
808   CK_SESSION_HANDLE hSession,        /* the session's handle */
809   CK_MECHANISM_PTR  pMechanism,      /* the wrapping mechanism */
810   CK_OBJECT_HANDLE  hWrappingKey,    /* wrapping key */
811   CK_OBJECT_HANDLE  hKey,            /* key to be wrapped */
812   CK_BYTE_PTR       pWrappedKey,     /* gets wrapped key */
813   CK_ULONG_PTR      pulWrappedKeyLen /* gets wrapped key size */
814 );
815 #endif


818 /* C_UnwrapKey unwraps (decrypts) a wrapped key, creating a new
819  * key object. */
820 CK_PKCS11_FUNCTION_INFO(C_UnwrapKey)
821 #ifdef CK_NEED_ARG_LIST
822 (
823   CK_SESSION_HANDLE    hSession,          /* session's handle */
824   CK_MECHANISM_PTR     pMechanism,        /* unwrapping mech. */
825   CK_OBJECT_HANDLE     hUnwrappingKey,    /* unwrapping key */
826   CK_BYTE_PTR          pWrappedKey,       /* the wrapped key */
827   CK_ULONG             ulWrappedKeyLen,   /* wrapped key len */
828   CK_ATTRIBUTE_PTR     pTemplate,         /* new key template */
829   CK_ULONG             ulAttributeCount,  /* template length */
830   CK_OBJECT_HANDLE_PTR phKey              /* gets new handle */
831 );
832 #endif


835 /* C_DeriveKey derives a key from a base key, creating a new key
836  * object. */
837 CK_PKCS11_FUNCTION_INFO(C_DeriveKey)
838 #ifdef CK_NEED_ARG_LIST
839 (
840   CK_SESSION_HANDLE    hSession,          /* session's handle */
841   CK_MECHANISM_PTR     pMechanism,        /* key deriv. mech. */
842   CK_OBJECT_HANDLE     hBaseKey,          /* base key */
843   CK_ATTRIBUTE_PTR     pTemplate,         /* new key template */
844   CK_ULONG             ulAttributeCount,  /* template length */
845   CK_OBJECT_HANDLE_PTR phKey              /* gets new handle */
846 );
847 #endif



851 /* Random number generation */

853 /* C_SeedRandom mixes additional seed material into the token's
```



```
854  * random number generator. */
855 CK_PKCS11_FUNCTION_INFO(C_SeedRandom)
856 #ifdef CK_NEED_ARG_LIST
857 (
858   CK_SESSION_HANDLE hSession,  /* the session's handle */
859   CK_BYTE_PTR       pSeed,     /* the seed material */
860   CK_ULONG          ulSeedLen  /* length of seed material */
861 );
862 #endif


865 /* C_GenerateRandom generates random data. */
866 CK_PKCS11_FUNCTION_INFO(C_GenerateRandom)
867 #ifdef CK_NEED_ARG_LIST
868 (
869   CK_SESSION_HANDLE hSession,    /* the session's handle */
870   CK_BYTE_PTR       RandomData,  /* receives the random data */
871   CK_ULONG          ulRandomLen  /* # of bytes to generate */
872 );
873 #endif



877 /* Parallel function management */

879 /* C_GetFunctionStatus is a legacy function; it obtains an
880  * updated status of a function running in parallel with an
881  * application. */
882 CK_PKCS11_FUNCTION_INFO(C_GetFunctionStatus)
883 #ifdef CK_NEED_ARG_LIST
884 (
885   CK_SESSION_HANDLE hSession  /* the session's handle */
886 );
887 #endif


890 /* C_CancelFunction is a legacy function; it cancels a function
891  * running in parallel. */
892 CK_PKCS11_FUNCTION_INFO(C_CancelFunction)
893 #ifdef CK_NEED_ARG_LIST
894 (
895   CK_SESSION_HANDLE hSession  /* the session's handle */
896 );
897 #endif



901 /* Functions added in for Cryptoki Version 2.01 or later */

903 /* C_WaitForSlotEvent waits for a slot event (token insertion,
904  * removal, etc.) to occur. */
905 CK_PKCS11_FUNCTION_INFO(C_WaitForSlotEvent)
906 #ifdef CK_NEED_ARG_LIST
907 (
908   CK_FLAGS flags,        /* blocking/nonblocking flag */
909   CK_SLOT_ID_PTR pSlot,  /* location that receives the slot ID */
910   CK_VOID_PTR pRserved   /* reserved.  Should be NULL_PTR */
911 );
912 #endif
913 #endif /* ! codereview */
```

```
**********************************************************
   69057 Wed Aug 13 19:32:54 2014
new/usr/src/lib/openssl/include/pkcs11t.h
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* pkcs11t.h include file for PKCS #11. */
/* $Revision: 1.10 $ */

/* License to copy and use this software is granted provided that it is
 * identified as "RSA Security Inc. PKCS #11 Cryptographic Token Interface
 * (Cryptoki)" in all material mentioning or referencing this software.

 * License is also granted to make and use derivative works provided that
 * such works are identified as "derived from the RSA Security Inc. PKCS #11
 * Cryptographic Token Interface (Cryptoki)" in all material mentioning or
 * referencing the derived work.

 * RSA Security Inc. makes no representations concerning either the
 * merchantability of this software or the suitability of this software for
 * any particular purpose. It is provided "as is" without express or implied
 * warranty of any kind.
 */

/* See top of pkcs11.h for information about the macros that
 * must be defined and the structure-packing conventions that
 * must be set before including this file. */

#ifndef _PKCS11T_H_
#define _PKCS11T_H_ 1

#define CRYPTOKI_VERSION_MAJOR 2
#define CRYPTOKI_VERSION_MINOR 20
#define CRYPTOKI_VERSION_AMENDMENT 3

#define CK_TRUE 1
#define CK_FALSE 0

#ifndef CK_DISABLE_TRUE_FALSE
#ifndef FALSE
#define FALSE CK_FALSE
#endif

#ifndef TRUE
#define TRUE CK_TRUE
#endif
#endif

/* an unsigned 8-bit value */
typedef unsigned char     CK_BYTE;

/* an unsigned 8-bit character */
typedef CK_BYTE           CK_CHAR;

/* an 8-bit UTF-8 character */
typedef CK_BYTE           CK_UTF8CHAR;

/* a BYTE-sized Boolean flag */
typedef CK_BYTE           CK_BBOOL;

/* an unsigned value, at least 32 bits long */
typedef unsigned long int CK_ULONG;

/* a signed value, the same size as a CK_ULONG */
/* CK_LONG is new for v2.0 */
typedef long int          CK_LONG;
```



```
/* at least 32 bits; each bit is a Boolean flag */
typedef CK_ULONG          CK_FLAGS;


/* some special values for certain CK_ULONG variables */
#define CK_UNAVAILABLE_INFORMATION (~0UL)
#define CK_EFFECTIVELY_INFINITE    0


typedef CK_BYTE     CK_PTR   CK_BYTE_PTR;
typedef CK_CHAR     CK_PTR   CK_CHAR_PTR;
typedef CK_UTF8CHAR CK_PTR   CK_UTF8CHAR_PTR;
typedef CK_ULONG    CK_PTR   CK_ULONG_PTR;
typedef void        CK_PTR   CK_VOID_PTR;

/* Pointer to a CK_VOID_PTR-- i.e., pointer to pointer to void */
typedef CK_VOID_PTR CK_PTR CK_VOID_PTR_PTR;


/* The following value is always invalid if used as a session */
/* handle or object handle */
#define CK_INVALID_HANDLE 0


typedef struct CK_VERSION {
  CK_BYTE       major;  /* integer portion of version number */
  CK_BYTE       minor;  /* 1/100ths portion of version number */
} CK_VERSION;

typedef CK_VERSION CK_PTR CK_VERSION_PTR;


typedef struct CK_INFO {
  /* manufacturerID and libraryDecription have been changed from
   * CK_CHAR to CK_UTF8CHAR for v2.10 */
  CK_VERSION    cryptokiVersion;     /* Cryptoki interface ver */
  CK_UTF8CHAR   manufacturerID[32];  /* blank padded */
  CK_FLAGS      flags;               /* must be zero */

  /* libraryDescription and libraryVersion are new for v2.0 */
  CK_UTF8CHAR   libraryDescription[32];  /* blank padded */
  CK_VERSION    libraryVersion;          /* version of library */
} CK_INFO;

typedef CK_INFO CK_PTR    CK_INFO_PTR;


/* CK_NOTIFICATION enumerates the types of notifications that
 * Cryptoki provides to an application */
/* CK_NOTIFICATION has been changed from an enum to a CK_ULONG
 * for v2.0 */
typedef CK_ULONG CK_NOTIFICATION;
#define CKN_SURRENDER       0

/* The following notification is new for PKCS #11 v2.20 amendment 3 */
#define CKN_OTP_CHANGED     1


typedef CK_ULONG          CK_SLOT_ID;

typedef CK_SLOT_ID CK_PTR CK_SLOT_ID_PTR;


/* CK_SLOT_INFO provides information about a slot */
typedef struct CK_SLOT_INFO {
  /* slotDescription and manufacturerID have been changed from
```

```
128    * CK_CHAR to CK_UTF8CHAR for v2.10 */
129   CK_UTF8CHAR   slotDescription[64];  /* blank padded */
130   CK_UTF8CHAR   manufacturerID[32];   /* blank padded */
131   CK_FLAGS      flags;

133   /* hardwareVersion and firmwareVersion are new for v2.0 */
134   CK_VERSION    hardwareVersion;  /* version of hardware */
135   CK_VERSION    firmwareVersion;  /* version of firmware */
136 } CK_SLOT_INFO;

138 /* flags: bit flags that provide capabilities of the slot
139  *      Bit Flag              Mask        Meaning
140  */
141 #define CKF_TOKEN_PRESENT     0x00000001  /* a token is there */
142 #define CKF_REMOVABLE_DEVICE  0x00000002  /* removable devices*/
143 #define CKF_HW_SLOT           0x00000004  /* hardware slot */

145 typedef CK_SLOT_INFO CK_PTR CK_SLOT_INFO_PTR;


148 /* CK_TOKEN_INFO provides information about a token */
149 typedef struct CK_TOKEN_INFO {
150   /* label, manufacturerID, and model have been changed from
151    * CK_CHAR to CK_UTF8CHAR for v2.10 */
152   CK_UTF8CHAR   label[32];           /* blank padded */
153   CK_UTF8CHAR   manufacturerID[32];  /* blank padded */
154   CK_UTF8CHAR   model[16];           /* blank padded */
155   CK_CHAR       serialNumber[16];    /* blank padded */
156   CK_FLAGS      flags;               /* see below */

158   /* ulMaxSessionCount, ulSessionCount, ulMaxRwSessionCount,
159    * ulRwSessionCount, ulMaxPinLen, and ulMinPinLen have all been
160    * changed from CK_USHORT to CK_ULONG for v2.0 */
161   CK_ULONG      ulMaxSessionCount;     /* max open sessions */
162   CK_ULONG      ulSessionCount;        /* sess. now open */
163   CK_ULONG      ulMaxRwSessionCount;   /* max R/W sessions */
164   CK_ULONG      ulRwSessionCount;      /* R/W sess. now open */
165   CK_ULONG      ulMaxPinLen;           /* in bytes */
166   CK_ULONG      ulMinPinLen;           /* in bytes */
167   CK_ULONG      ulTotalPublicMemory;   /* in bytes */
168   CK_ULONG      ulFreePublicMemory;    /* in bytes */
169   CK_ULONG      ulTotalPrivateMemory;  /* in bytes */
170   CK_ULONG      ulFreePrivateMemory;   /* in bytes */

172   /* hardwareVersion, firmwareVersion, and time are new for
173    * v2.0 */
174   CK_VERSION    hardwareVersion;       /* version of hardware */
175   CK_VERSION    firmwareVersion;       /* version of firmware */
176   CK_CHAR       utcTime[16];           /* time */
177 } CK_TOKEN_INFO;

179 /* The flags parameter is defined as follows:
180  *      Bit Flag                    Mask        Meaning
181  */
182 #define CKF_RNG                     0x00000001  /* has random #
183                                                  * generator */
184 #define CKF_WRITE_PROTECTED         0x00000002  /* token is
185                                                  * write-
186                                                  * protected */
187 #define CKF_LOGIN_REQUIRED          0x00000004  /* user must
188                                                  * login */
189 #define CKF_USER_PIN_INITIALIZED    0x00000008  /* normal user's
190                                                  * PIN is set */

192 /* CKF_RESTORE_KEY_NOT_NEEDED is new for v2.0.  If it is set,
193  * that means that *every* time the state of cryptographic
```



```
194  * operations of a session is successfully saved, all keys
195  * needed to continue those operations are stored in the state */
196 #define CKF_RESTORE_KEY_NOT_NEEDED  0x00000020

198 /* CKF_CLOCK_ON_TOKEN is new for v2.0.  If it is set, that means
199  * that the token has some sort of clock.  The time on that
200  * clock is returned in the token info structure */
201 #define CKF_CLOCK_ON_TOKEN          0x00000040

203 /* CKF_PROTECTED_AUTHENTICATION_PATH is new for v2.0.  If it is
204  * set, that means that there is some way for the user to login
205  * without sending a PIN through the Cryptoki library itself */
206 #define CKF_PROTECTED_AUTHENTICATION_PATH 0x00000100

208 /* CKF_DUAL_CRYPTO_OPERATIONS is new for v2.0.  If it is true,
209  * that means that a single session with the token can perform
210  * dual simultaneous cryptographic operations (digest and
211  * encrypt; decrypt and digest; sign and encrypt; and decrypt
212  * and sign) */
213 #define CKF_DUAL_CRYPTO_OPERATIONS  0x00000200

215 /* CKF_TOKEN_INITIALIZED if new for v2.10. If it is true, the
216  * token has been initialized using C_InitializeToken or an
217  * equivalent mechanism outside the scope of PKCS #11.
218  * Calling C_InitializeToken when this flag is set will cause
219  * the token to be reinitialized. */
220 #define CKF_TOKEN_INITIALIZED       0x00000400

222 /* CKF_SECONDARY_AUTHENTICATION if new for v2.10. If it is
223  * true, the token supports secondary authentication for
224  * private key objects. This flag is deprecated in v2.11 and
225    onwards. */
226 #define CKF_SECONDARY_AUTHENTICATION  0x00000800

228 /* CKF_USER_PIN_COUNT_LOW if new for v2.10. If it is true, an
229  * incorrect user login PIN has been entered at least once
230  * since the last successful authentication. */
231 #define CKF_USER_PIN_COUNT_LOW       0x00010000

233 /* CKF_USER_PIN_FINAL_TRY if new for v2.10. If it is true,
234  * supplying an incorrect user PIN will it to become locked. */
235 #define CKF_USER_PIN_FINAL_TRY       0x00020000

237 /* CKF_USER_PIN_LOCKED if new for v2.10. If it is true, the
238  * user PIN has been locked. User login to the token is not
239  * possible. */
240 #define CKF_USER_PIN_LOCKED          0x00040000

242 /* CKF_USER_PIN_TO_BE_CHANGED if new for v2.10. If it is true,
243  * the user PIN value is the default value set by token
244  * initialization or manufacturing, or the PIN has been
245  * expired by the card. */
246 #define CKF_USER_PIN_TO_BE_CHANGED   0x00080000

248 /* CKF_SO_PIN_COUNT_LOW if new for v2.10. If it is true, an
249  * incorrect SO login PIN has been entered at least once since
250  * the last successful authentication. */
251 #define CKF_SO_PIN_COUNT_LOW         0x00100000

253 /* CKF_SO_PIN_FINAL_TRY if new for v2.10. If it is true,
254  * supplying an incorrect SO PIN will it to become locked. */
255 #define CKF_SO_PIN_FINAL_TRY         0x00200000

257 /* CKF_SO_PIN_LOCKED if new for v2.10. If it is true, the SO
258  * PIN has been locked. SO login to the token is not possible.
259  */
```

```
260 #define CKF_SO_PIN_LOCKED            0x00400000

262 /* CKF_SO_PIN_TO_BE_CHANGED if new for v2.10. If it is true,
263  * the SO PIN value is the default value set by token
264  * initialization or manufacturing, or the PIN has been
265  * expired by the card. */
266 #define CKF_SO_PIN_TO_BE_CHANGED     0x00800000

268 typedef CK_TOKEN_INFO CK_PTR CK_TOKEN_INFO_PTR;


271 /* CK_SESSION_HANDLE is a Cryptoki-assigned value that
272  * identifies a session */
273 typedef CK_ULONG          CK_SESSION_HANDLE;

275 typedef CK_SESSION_HANDLE CK_PTR CK_SESSION_HANDLE_PTR;


278 /* CK_USER_TYPE enumerates the types of Cryptoki users */
279 /* CK_USER_TYPE has been changed from an enum to a CK_ULONG for
280  * v2.0 */
281 typedef CK_ULONG          CK_USER_TYPE;
282 /* Security Officer */
283 #define CKU_SO    0
284 /* Normal user */
285 #define CKU_USER  1
286 /* Context specific (added in v2.20) */
287 #define CKU_CONTEXT_SPECIFIC   2

289 /* CK_STATE enumerates the session states */
290 /* CK_STATE has been changed from an enum to a CK_ULONG for
291  * v2.0 */
292 typedef CK_ULONG          CK_STATE;
293 #define CKS_RO_PUBLIC_SESSION  0
294 #define CKS_RO_USER_FUNCTIONS  1
295 #define CKS_RW_PUBLIC_SESSION  2
296 #define CKS_RW_USER_FUNCTIONS  3
297 #define CKS_RW_SO_FUNCTIONS    4


300 /* CK_SESSION_INFO provides information about a session */
301 typedef struct CK_SESSION_INFO {
302   CK_SLOT_ID    slotID;
303   CK_STATE      state;
304   CK_FLAGS      flags;          /* see below */

306   /* ulDeviceError was changed from CK_USHORT to CK_ULONG for
307    * v2.0 */
308   CK_ULONG      ulDeviceError;  /* device-dependent error code */
309 } CK_SESSION_INFO;

311 /* The flags are defined in the following table:
312  *      Bit Flag                Mask        Meaning
313  */
314 #define CKF_RW_SESSION          0x00000002  /* session is r/w */
315 #define CKF_SERIAL_SESSION      0x00000004  /* no parallel */

317 typedef CK_SESSION_INFO CK_PTR CK_SESSION_INFO_PTR;


320 /* CK_OBJECT_HANDLE is a token-specific identifier for an
321  * object  */
322 typedef CK_ULONG          CK_OBJECT_HANDLE;

324 typedef CK_OBJECT_HANDLE CK_PTR CK_OBJECT_HANDLE_PTR;
```



```
327 /* CK_OBJECT_CLASS is a value that identifies the classes (or
328  * types) of objects that Cryptoki recognizes.  It is defined
329  * as follows: */
330 /* CK_OBJECT_CLASS was changed from CK_USHORT to CK_ULONG for
331  * v2.0 */
332 typedef CK_ULONG          CK_OBJECT_CLASS;

334 /* The following classes of objects are defined: */
335 /* CKO_HW_FEATURE is new for v2.10 */
336 /* CKO_DOMAIN_PARAMETERS is new for v2.11 */
337 /* CKO_MECHANISM is new for v2.20 */
338 #define CKO_DATA                0x00000000
339 #define CKO_CERTIFICATE         0x00000001
340 #define CKO_PUBLIC_KEY          0x00000002
341 #define CKO_PRIVATE_KEY         0x00000003
342 #define CKO_SECRET_KEY          0x00000004
343 #define CKO_HW_FEATURE          0x00000005
344 #define CKO_DOMAIN_PARAMETERS 0x00000006
345 #define CKO_MECHANISM           0x00000007

347 /* CKO_OTP_KEY is new for PKCS #11 v2.20 amendment 1 */
348 #define CKO_OTP_KEY             0x00000008

350 #define CKO_VENDOR_DEFINED    0x80000000

352 typedef CK_OBJECT_CLASS CK_PTR CK_OBJECT_CLASS_PTR;

354 /* CK_HW_FEATURE_TYPE is new for v2.10. CK_HW_FEATURE_TYPE is a
355  * value that identifies the hardware feature type of an object
356  * with CK_OBJECT_CLASS equal to CKO_HW_FEATURE. */
357 typedef CK_ULONG          CK_HW_FEATURE_TYPE;

359 /* The following hardware feature types are defined */
360 /* CKH_USER_INTERFACE is new for v2.20 */
361 #define CKH_MONOTONIC_COUNTER  0x00000001
362 #define CKH_CLOCK              0x00000002
363 #define CKH_USER_INTERFACE     0x00000003
364 #define CKH_VENDOR_DEFINED     0x80000000

366 /* CK_KEY_TYPE is a value that identifies a key type */
367 /* CK_KEY_TYPE was changed from CK_USHORT to CK_ULONG for v2.0 */
368 typedef CK_ULONG          CK_KEY_TYPE;

370 /* the following key types are defined: */
371 #define CKK_RSA                 0x00000000
372 #define CKK_DSA                 0x00000001
373 #define CKK_DH                  0x00000002

375 /* CKK_ECDSA and CKK_KEA are new for v2.0 */
376 /* CKK_ECDSA is deprecated in v2.11, CKK_EC is preferred. */
377 #define CKK_ECDSA               0x00000003
378 #define CKK_EC                  0x00000003
379 #define CKK_X9_42_DH            0x00000004
380 #define CKK_KEA                 0x00000005

382 #define CKK_GENERIC_SECRET  0x00000010
383 #define CKK_RC2                 0x00000011
384 #define CKK_RC4                 0x00000012
385 #define CKK_DES                 0x00000013
386 #define CKK_DES2                0x00000014
387 #define CKK_DES3                0x00000015

389 /* all these key types are new for v2.0 */
390 #define CKK_CAST                0x00000016
391 #define CKK_CAST3               0x00000017
```

```
/* CKK_CAST5 is deprecated in v2.11, CKK_CAST128 is preferred. */
#define CKK_CAST5           0x00000018
#define CKK_CAST128         0x00000018
#define CKK_RC5             0x00000019
#define CKK_IDEA            0x0000001A
#define CKK_SKIPJACK        0x0000001B
#define CKK_BATON           0x0000001C
#define CKK_JUNIPER         0x0000001D
#define CKK_CDMF            0x0000001E
#define CKK_AES             0x0000001F

/* BlowFish and TwoFish are new for v2.20 */
#define CKK_BLOWFISH        0x00000020
#define CKK_TWOFISH         0x00000021

/* SecurID, HOTP, and ACTI are new for PKCS #11 v2.20 amendment 1 */
#define CKK_SECURID         0x00000022
#define CKK_HOTP            0x00000023
#define CKK_ACTI            0x00000024

/* Camellia is new for PKCS #11 v2.20 amendment 3 */
#define CKK_CAMELLIA                    0x00000025
/* ARIA is new for PKCS #11 v2.20 amendment 3 */
#define CKK_ARIA                        0x00000026


#define CKK_VENDOR_DEFINED  0x80000000


/* CK_CERTIFICATE_TYPE is a value that identifies a certificate
 * type */
/* CK_CERTIFICATE_TYPE was changed from CK_USHORT to CK_ULONG
 * for v2.0 */
typedef CK_ULONG          CK_CERTIFICATE_TYPE;

/* The following certificate types are defined: */
/* CKC_X_509_ATTR_CERT is new for v2.10 */
/* CKC_WTLS is new for v2.20 */
#define CKC_X_509           0x00000000
#define CKC_X_509_ATTR_CERT 0x00000001
#define CKC_WTLS            0x00000002
#define CKC_VENDOR_DEFINED  0x80000000


/* CK_ATTRIBUTE_TYPE is a value that identifies an attribute
 * type */
/* CK_ATTRIBUTE_TYPE was changed from CK_USHORT to CK_ULONG for
 * v2.0 */
typedef CK_ULONG          CK_ATTRIBUTE_TYPE;

/* The CKF_ARRAY_ATTRIBUTE flag identifies an attribute which
   consists of an array of values. */
#define CKF_ARRAY_ATTRIBUTE    0x40000000

/* The following OTP-related defines are new for PKCS #11 v2.20 amendment 1
   and relates to the CKA_OTP_FORMAT attribute */
#define CK_OTP_FORMAT_DECIMAL      0
#define CK_OTP_FORMAT_HEXADECIMAL  1
#define CK_OTP_FORMAT_ALPHANUMERIC 2
#define CK_OTP_FORMAT_BINARY       3

/* The following OTP-related defines are new for PKCS #11 v2.20 amendment 1
   and relates to the CKA_OTP_..._REQUIREMENT attributes */
#define CK_OTP_PARAM_IGNORED       0
#define CK_OTP_PARAM_OPTIONAL      1
#define CK_OTP_PARAM_MANDATORY     2
```



```
/* The following attribute types are defined: */
#define CKA_CLASS               0x00000000
#define CKA_TOKEN               0x00000001
#define CKA_PRIVATE             0x00000002
#define CKA_LABEL               0x00000003
#define CKA_APPLICATION         0x00000010
#define CKA_VALUE               0x00000011

/* CKA_OBJECT_ID is new for v2.10 */
#define CKA_OBJECT_ID           0x00000012

#define CKA_CERTIFICATE_TYPE    0x00000080
#define CKA_ISSUER              0x00000081
#define CKA_SERIAL_NUMBER       0x00000082

/* CKA_AC_ISSUER, CKA_OWNER, and CKA_ATTR_TYPES are new
 * for v2.10 */
#define CKA_AC_ISSUER           0x00000083
#define CKA_OWNER               0x00000084
#define CKA_ATTR_TYPES          0x00000085

/* CKA_TRUSTED is new for v2.11 */
#define CKA_TRUSTED             0x00000086

/* CKA_CERTIFICATE_CATEGORY ...
 * CKA_CHECK_VALUE are new for v2.20 */
#define CKA_CERTIFICATE_CATEGORY        0x00000087
#define CKA_JAVA_MIDP_SECURITY_DOMAIN   0x00000088
#define CKA_URL                         0x00000089
#define CKA_HASH_OF_SUBJECT_PUBLIC_KEY  0x0000008A
#define CKA_HASH_OF_ISSUER_PUBLIC_KEY   0x0000008B
#define CKA_CHECK_VALUE                 0x00000090

#define CKA_KEY_TYPE            0x00000100
#define CKA_SUBJECT             0x00000101
#define CKA_ID                  0x00000102
#define CKA_SENSITIVE           0x00000103
#define CKA_ENCRYPT             0x00000104
#define CKA_DECRYPT             0x00000105
#define CKA_WRAP                0x00000106
#define CKA_UNWRAP              0x00000107
#define CKA_SIGN                0x00000108
#define CKA_SIGN_RECOVER        0x00000109
#define CKA_VERIFY              0x0000010A
#define CKA_VERIFY_RECOVER      0x0000010B
#define CKA_DERIVE              0x0000010C
#define CKA_START_DATE          0x00000110
#define CKA_END_DATE            0x00000111
#define CKA_MODULUS             0x00000120
#define CKA_MODULUS_BITS        0x00000121
#define CKA_PUBLIC_EXPONENT     0x00000122
#define CKA_PRIVATE_EXPONENT    0x00000123
#define CKA_PRIME_1             0x00000124
#define CKA_PRIME_2             0x00000125
#define CKA_EXPONENT_1          0x00000126
#define CKA_EXPONENT_2          0x00000127
#define CKA_COEFFICIENT         0x00000128
#define CKA_PRIME               0x00000130
#define CKA_SUBPRIME            0x00000131
#define CKA_BASE                0x00000132

/* CKA_PRIME_BITS and CKA_SUB_PRIME_BITS are new for v2.11 */
#define CKA_PRIME_BITS          0x00000133
#define CKA_SUBPRIME_BITS       0x00000134
#define CKA_SUB_PRIME_BITS      CKA_SUBPRIME_BITS
```

```
524 /* (To retain backwards-compatibility) */

526 #define CKA_VALUE_BITS          0x00000160
527 #define CKA_VALUE_LEN           0x00000161

529 /* CKA_EXTRACTABLE, CKA_LOCAL, CKA_NEVER_EXTRACTABLE,
530  * CKA_ALWAYS_SENSITIVE, CKA_MODIFIABLE, CKA_ECDSA_PARAMS,
531  * and CKA_EC_POINT are new for v2.0 */
532 #define CKA_EXTRACTABLE         0x00000162
533 #define CKA_LOCAL               0x00000163
534 #define CKA_NEVER_EXTRACTABLE   0x00000164
535 #define CKA_ALWAYS_SENSITIVE    0x00000165

537 /* CKA_KEY_GEN_MECHANISM is new for v2.11 */
538 #define CKA_KEY_GEN_MECHANISM   0x00000166

540 #define CKA_MODIFIABLE          0x00000170

542 /* CKA_ECDSA_PARAMS is deprecated in v2.11,
543  * CKA_EC_PARAMS is preferred. */
544 #define CKA_ECDSA_PARAMS        0x00000180
545 #define CKA_EC_PARAMS           0x00000180

547 #define CKA_EC_POINT            0x00000181

549 /* CKA_SECONDARY_AUTH, CKA_AUTH_PIN_FLAGS,
550  * are new for v2.10. Deprecated in v2.11 and onwards. */
551 #define CKA_SECONDARY_AUTH      0x00000200
552 #define CKA_AUTH_PIN_FLAGS      0x00000201

554 /* CKA_ALWAYS_AUTHENTICATE ...
555  * CKA_UNWRAP_TEMPLATE are new for v2.20 */
556 #define CKA_ALWAYS_AUTHENTICATE  0x00000202

558 #define CKA_WRAP_WITH_TRUSTED    0x00000210
559 #define CKA_WRAP_TEMPLATE        (CKF_ARRAY_ATTRIBUTE|0x00000211)
560 #define CKA_UNWRAP_TEMPLATE      (CKF_ARRAY_ATTRIBUTE|0x00000212)

562 /* CKA_OTP... atttributes are new for PKCS #11 v2.20 amendment 3. */
563 #define CKA_OTP_FORMAT                0x00000220
564 #define CKA_OTP_LENGTH                0x00000221
565 #define CKA_OTP_TIME_INTERVAL         0x00000222
566 #define CKA_OTP_USER_FRIENDLY_MODE    0x00000223
567 #define CKA_OTP_CHALLENGE_REQUIREMENT 0x00000224
568 #define CKA_OTP_TIME_REQUIREMENT      0x00000225
569 #define CKA_OTP_COUNTER_REQUIREMENT   0x00000226
570 #define CKA_OTP_PIN_REQUIREMENT       0x00000227
571 #define CKA_OTP_COUNTER               0x0000022E
572 #define CKA_OTP_TIME                  0x0000022F
573 #define CKA_OTP_USER_IDENTIFIER       0x0000022A
574 #define CKA_OTP_SERVICE_IDENTIFIER    0x0000022B
575 #define CKA_OTP_SERVICE_LOGO          0x0000022C
576 #define CKA_OTP_SERVICE_LOGO_TYPE     0x0000022D


579 /* CKA_HW_FEATURE_TYPE, CKA_RESET_ON_INIT, and CKA_HAS_RESET
580  * are new for v2.10 */
581 #define CKA_HW_FEATURE_TYPE    0x00000300
582 #define CKA_RESET_ON_INIT      0x00000301
583 #define CKA_HAS_RESET          0x00000302

585 /* The following attributes are new for v2.20 */
586 #define CKA_PIXEL_X                    0x00000400
587 #define CKA_PIXEL_Y                    0x00000401
588 #define CKA_RESOLUTION                 0x00000402
589 #define CKA_CHAR_ROWS                  0x00000403
```



```
590 #define CKA_CHAR_COLUMNS               0x00000404
591 #define CKA_COLOR                      0x00000405
592 #define CKA_BITS_PER_PIXEL             0x00000406
593 #define CKA_CHAR_SETS                  0x00000480
594 #define CKA_ENCODING_METHODS           0x00000481
595 #define CKA_MIME_TYPES                 0x00000482
596 #define CKA_MECHANISM_TYPE             0x00000500
597 #define CKA_REQUIRED_CMS_ATTRIBUTES    0x00000501
598 #define CKA_DEFAULT_CMS_ATTRIBUTES     0x00000502
599 #define CKA_SUPPORTED_CMS_ATTRIBUTES   0x00000503
600 #define CKA_ALLOWED_MECHANISMS         (CKF_ARRAY_ATTRIBUTE|0x00000600)

602 #define CKA_VENDOR_DEFINED     0x80000000

604 /* CK_ATTRIBUTE is a structure that includes the type, length
605  * and value of an attribute */
606 typedef struct CK_ATTRIBUTE {
607   CK_ATTRIBUTE_TYPE type;
608   CK_VOID_PTR       pValue;

610   /* ulValueLen went from CK_USHORT to CK_ULONG for v2.0 */
611   CK_ULONG          ulValueLen;  /* in bytes */
612 } CK_ATTRIBUTE;

614 typedef CK_ATTRIBUTE CK_PTR CK_ATTRIBUTE_PTR;


617 /* CK_DATE is a structure that defines a date */
618 typedef struct CK_DATE{
619   CK_CHAR       year[4];   /* the year ("1900" - "9999") */
620   CK_CHAR       month[2];  /* the month ("01" - "12") */
621   CK_CHAR       day[2];    /* the day   ("01" - "31") */
622 } CK_DATE;


625 /* CK_MECHANISM_TYPE is a value that identifies a mechanism
626  * type */
627 /* CK_MECHANISM_TYPE was changed from CK_USHORT to CK_ULONG for
628  * v2.0 */
629 typedef CK_ULONG          CK_MECHANISM_TYPE;

631 /* the following mechanism types are defined: */
632 #define CKM_RSA_PKCS_KEY_PAIR_GEN      0x00000000
633 #define CKM_RSA_PKCS                   0x00000001
634 #define CKM_RSA_9796                   0x00000002
635 #define CKM_RSA_X_509                  0x00000003

637 /* CKM_MD2_RSA_PKCS, CKM_MD5_RSA_PKCS, and CKM_SHA1_RSA_PKCS
638  * are new for v2.0.  They are mechanisms which hash and sign */
639 #define CKM_MD2_RSA_PKCS               0x00000004
640 #define CKM_MD5_RSA_PKCS               0x00000005
641 #define CKM_SHA1_RSA_PKCS              0x00000006

643 /* CKM_RIPEMD128_RSA_PKCS, CKM_RIPEMD160_RSA_PKCS, and
644  * CKM_RSA_PKCS_OAEP are new for v2.10 */
645 #define CKM_RIPEMD128_RSA_PKCS         0x00000007
646 #define CKM_RIPEMD160_RSA_PKCS         0x00000008
647 #define CKM_RSA_PKCS_OAEP              0x00000009

649 /* CKM_RSA_X9_31_KEY_PAIR_GEN, CKM_RSA_X9_31, CKM_SHA1_RSA_X9_31,
650  * CKM_RSA_PKCS_PSS, and CKM_SHA1_RSA_PKCS_PSS are new for v2.11 */
651 #define CKM_RSA_X9_31_KEY_PAIR_GEN     0x0000000A
652 #define CKM_RSA_X9_31                  0x0000000B
653 #define CKM_SHA1_RSA_X9_31             0x0000000C
654 #define CKM_RSA_PKCS_PSS               0x0000000D
655 #define CKM_SHA1_RSA_PKCS_PSS          0x0000000E
```

```
#define CKM_DSA_KEY_PAIR_GEN           0x00000010
#define CKM_DSA                        0x00000011
#define CKM_DSA_SHA1                   0x00000012
#define CKM_DH_PKCS_KEY_PAIR_GEN       0x00000020
#define CKM_DH_PKCS_DERIVE             0x00000021

/* CKM_X9_42_DH_KEY_PAIR_GEN, CKM_X9_42_DH_DERIVE,
 * CKM_X9_42_DH_HYBRID_DERIVE, and CKM_X9_42_MQV_DERIVE are new for
 * v2.11 */
#define CKM_X9_42_DH_KEY_PAIR_GEN      0x00000030
#define CKM_X9_42_DH_DERIVE            0x00000031
#define CKM_X9_42_DH_HYBRID_DERIVE     0x00000032
#define CKM_X9_42_MQV_DERIVE           0x00000033

/* CKM_SHA256/384/512 are new for v2.20 */
#define CKM_SHA256_RSA_PKCS            0x00000040
#define CKM_SHA384_RSA_PKCS            0x00000041
#define CKM_SHA512_RSA_PKCS            0x00000042
#define CKM_SHA256_RSA_PKCS_PSS        0x00000043
#define CKM_SHA384_RSA_PKCS_PSS        0x00000044
#define CKM_SHA512_RSA_PKCS_PSS        0x00000045

/* SHA-224 RSA mechanisms are new for PKCS #11 v2.20 amendment 3 */
#define CKM_SHA224_RSA_PKCS            0x00000046
#define CKM_SHA224_RSA_PKCS_PSS        0x00000047

#define CKM_RC2_KEY_GEN                0x00000100
#define CKM_RC2_ECB                    0x00000101
#define CKM_RC2_CBC                    0x00000102
#define CKM_RC2_MAC                    0x00000103

/* CKM_RC2_MAC_GENERAL and CKM_RC2_CBC_PAD are new for v2.0 */
#define CKM_RC2_MAC_GENERAL            0x00000104
#define CKM_RC2_CBC_PAD                0x00000105

#define CKM_RC4_KEY_GEN                0x00000110
#define CKM_RC4                        0x00000111
#define CKM_DES_KEY_GEN                0x00000120
#define CKM_DES_ECB                    0x00000121
#define CKM_DES_CBC                    0x00000122
#define CKM_DES_MAC                    0x00000123

/* CKM_DES_MAC_GENERAL and CKM_DES_CBC_PAD are new for v2.0 */
#define CKM_DES_MAC_GENERAL            0x00000124
#define CKM_DES_CBC_PAD                0x00000125

#define CKM_DES2_KEY_GEN               0x00000130
#define CKM_DES3_KEY_GEN               0x00000131
#define CKM_DES3_ECB                   0x00000132
#define CKM_DES3_CBC                   0x00000133
#define CKM_DES3_MAC                   0x00000134

/* CKM_DES3_MAC_GENERAL, CKM_DES3_CBC_PAD, CKM_CDMF_KEY_GEN,
 * CKM_CDMF_ECB, CKM_CDMF_CBC, CKM_CDMF_MAC,
 * CKM_CDMF_MAC_GENERAL, and CKM_CDMF_CBC_PAD are new for v2.0 */
#define CKM_DES3_MAC_GENERAL           0x00000135
#define CKM_DES3_CBC_PAD               0x00000136
#define CKM_CDMF_KEY_GEN               0x00000140
#define CKM_CDMF_ECB                   0x00000141
#define CKM_CDMF_CBC                   0x00000142
#define CKM_CDMF_MAC                   0x00000143
#define CKM_CDMF_MAC_GENERAL           0x00000144
#define CKM_CDMF_CBC_PAD               0x00000145

/* the following four DES mechanisms are new for v2.20 */
```



```
#define CKM_DES_OFB64                  0x00000150
#define CKM_DES_OFB8                   0x00000151
#define CKM_DES_CFB64                  0x00000152
#define CKM_DES_CFB8                   0x00000153

#define CKM_MD2                        0x00000200

/* CKM_MD2_HMAC and CKM_MD2_HMAC_GENERAL are new for v2.0 */
#define CKM_MD2_HMAC                   0x00000201
#define CKM_MD2_HMAC_GENERAL           0x00000202

#define CKM_MD5                        0x00000210

/* CKM_MD5_HMAC and CKM_MD5_HMAC_GENERAL are new for v2.0 */
#define CKM_MD5_HMAC                   0x00000211
#define CKM_MD5_HMAC_GENERAL           0x00000212

#define CKM_SHA_1                      0x00000220

/* CKM_SHA_1_HMAC and CKM_SHA_1_HMAC_GENERAL are new for v2.0 */
#define CKM_SHA_1_HMAC                 0x00000221
#define CKM_SHA_1_HMAC_GENERAL         0x00000222

/* CKM_RIPEMD128, CKM_RIPEMD128_HMAC,
 * CKM_RIPEMD128_HMAC_GENERAL, CKM_RIPEMD160, CKM_RIPEMD160_HMAC,
 * and CKM_RIPEMD160_HMAC_GENERAL are new for v2.10 */
#define CKM_RIPEMD128                  0x00000230
#define CKM_RIPEMD128_HMAC             0x00000231
#define CKM_RIPEMD128_HMAC_GENERAL     0x00000232
#define CKM_RIPEMD160                  0x00000240
#define CKM_RIPEMD160_HMAC             0x00000241
#define CKM_RIPEMD160_HMAC_GENERAL     0x00000242

/* CKM_SHA256/384/512 are new for v2.20 */
#define CKM_SHA256                     0x00000250
#define CKM_SHA256_HMAC                0x00000251
#define CKM_SHA256_HMAC_GENERAL        0x00000252

/* SHA-224 is new for PKCS #11 v2.20 amendment 3 */
#define CKM_SHA224                     0x00000255
#define CKM_SHA224_HMAC                0x00000256
#define CKM_SHA224_HMAC_GENERAL        0x00000257

#define CKM_SHA384                     0x00000260
#define CKM_SHA384_HMAC                0x00000261
#define CKM_SHA384_HMAC_GENERAL        0x00000262
#define CKM_SHA512                     0x00000270
#define CKM_SHA512_HMAC                0x00000271
#define CKM_SHA512_HMAC_GENERAL        0x00000272

/* SecurID is new for PKCS #11 v2.20 amendment 1 */
#define CKM_SECURID_KEY_GEN            0x00000280
#define CKM_SECURID                    0x00000282

/* HOTP is new for PKCS #11 v2.20 amendment 1 */
#define CKM_HOTP_KEY_GEN    0x00000290
#define CKM_HOTP            0x00000291

/* ACTI is new for PKCS #11 v2.20 amendment 1 */
#define CKM_ACTI            0x000002A0
#define CKM_ACTI_KEY_GEN    0x000002A1

/* All of the following mechanisms are new for v2.0 */
/* Note that CAST128 and CAST5 are the same algorithm */
#define CKM_CAST_KEY_GEN               0x00000300
#define CKM_CAST_ECB                   0x00000301
```

```
788 #define CKM_CAST_CBC                    0x00000302
789 #define CKM_CAST_MAC                    0x00000303
790 #define CKM_CAST_MAC_GENERAL            0x00000304
791 #define CKM_CAST_CBC_PAD                0x00000305
792 #define CKM_CAST3_KEY_GEN               0x00000310
793 #define CKM_CAST3_ECB                   0x00000311
794 #define CKM_CAST3_CBC                   0x00000312
795 #define CKM_CAST3_MAC                   0x00000313
796 #define CKM_CAST3_MAC_GENERAL           0x00000314
797 #define CKM_CAST3_CBC_PAD               0x00000315
798 #define CKM_CAST5_KEY_GEN               0x00000320
799 #define CKM_CAST128_KEY_GEN             0x00000320
800 #define CKM_CAST5_ECB                   0x00000321
801 #define CKM_CAST128_ECB                 0x00000321
802 #define CKM_CAST5_CBC                   0x00000322
803 #define CKM_CAST128_CBC                 0x00000322
804 #define CKM_CAST5_MAC                   0x00000323
805 #define CKM_CAST128_MAC                 0x00000323
806 #define CKM_CAST5_MAC_GENERAL           0x00000324
807 #define CKM_CAST128_MAC_GENERAL         0x00000324
808 #define CKM_CAST5_CBC_PAD               0x00000325
809 #define CKM_CAST128_CBC_PAD             0x00000325
810 #define CKM_RC5_KEY_GEN                 0x00000330
811 #define CKM_RC5_ECB                     0x00000331
812 #define CKM_RC5_CBC                     0x00000332
813 #define CKM_RC5_MAC                     0x00000333
814 #define CKM_RC5_MAC_GENERAL             0x00000334
815 #define CKM_RC5_CBC_PAD                 0x00000335
816 #define CKM_IDEA_KEY_GEN                0x00000340
817 #define CKM_IDEA_ECB                    0x00000341
818 #define CKM_IDEA_CBC                    0x00000342
819 #define CKM_IDEA_MAC                    0x00000343
820 #define CKM_IDEA_MAC_GENERAL            0x00000344
821 #define CKM_IDEA_CBC_PAD                0x00000345
822 #define CKM_GENERIC_SECRET_KEY_GEN      0x00000350
823 #define CKM_CONCATENATE_BASE_AND_KEY    0x00000360
824 #define CKM_CONCATENATE_BASE_AND_DATA   0x00000362
825 #define CKM_CONCATENATE_DATA_AND_BASE   0x00000363
826 #define CKM_XOR_BASE_AND_DATA           0x00000364
827 #define CKM_EXTRACT_KEY_FROM_KEY        0x00000365
828 #define CKM_SSL3_PRE_MASTER_KEY_GEN     0x00000370
829 #define CKM_SSL3_MASTER_KEY_DERIVE      0x00000371
830 #define CKM_SSL3_KEY_AND_MAC_DERIVE     0x00000372

832 /* CKM_SSL3_MASTER_KEY_DERIVE_DH, CKM_TLS_PRE_MASTER_KEY_GEN,
833  * CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_KEY_AND_MAC_DERIVE, and
834  * CKM_TLS_MASTER_KEY_DERIVE_DH are new for v2.11 */
835 #define CKM_SSL3_MASTER_KEY_DERIVE_DH   0x00000373
836 #define CKM_TLS_PRE_MASTER_KEY_GEN      0x00000374
837 #define CKM_TLS_MASTER_KEY_DERIVE       0x00000375
838 #define CKM_TLS_KEY_AND_MAC_DERIVE      0x00000376
839 #define CKM_TLS_MASTER_KEY_DERIVE_DH    0x00000377

841 /* CKM_TLS_PRF is new for v2.20 */
842 #define CKM_TLS_PRF                     0x00000378

844 #define CKM_SSL3_MD5_MAC                0x00000380
845 #define CKM_SSL3_SHA1_MAC               0x00000381
846 #define CKM_MD5_KEY_DERIVATION          0x00000390
847 #define CKM_MD2_KEY_DERIVATION          0x00000391
848 #define CKM_SHA1_KEY_DERIVATION         0x00000392

850 /* CKM_SHA256/384/512 are new for v2.20 */
851 #define CKM_SHA256_KEY_DERIVATION       0x00000393
852 #define CKM_SHA384_KEY_DERIVATION       0x00000394
853 #define CKM_SHA512_KEY_DERIVATION       0x00000395
```



```
855 /* SHA-224 key derivation is new for PKCS #11 v2.20 amendment 3 */
856 #define CKM_SHA224_KEY_DERIVATION       0x00000396

858 #define CKM_PBE_MD2_DES_CBC             0x000003A0
859 #define CKM_PBE_MD5_DES_CBC             0x000003A1
860 #define CKM_PBE_MD5_CAST_CBC            0x000003A2
861 #define CKM_PBE_MD5_CAST3_CBC           0x000003A3
862 #define CKM_PBE_MD5_CAST5_CBC           0x000003A4
863 #define CKM_PBE_MD5_CAST128_CBC         0x000003A4
864 #define CKM_PBE_SHA1_CAST5_CBC          0x000003A5
865 #define CKM_PBE_SHA1_CAST128_CBC        0x000003A5
866 #define CKM_PBE_SHA1_RC4_128            0x000003A6
867 #define CKM_PBE_SHA1_RC4_40             0x000003A7
868 #define CKM_PBE_SHA1_DES3_EDE_CBC       0x000003A8
869 #define CKM_PBE_SHA1_DES2_EDE_CBC       0x000003A9
870 #define CKM_PBE_SHA1_RC2_128_CBC        0x000003AA
871 #define CKM_PBE_SHA1_RC2_40_CBC         0x000003AB

873 /* CKM_PKCS5_PBKD2 is new for v2.10 */
874 #define CKM_PKCS5_PBKD2                 0x000003B0

876 #define CKM_PBA_SHA1_WITH_SHA1_HMAC     0x000003C0

878 /* WTLS mechanisms are new for v2.20 */
879 #define CKM_WTLS_PRE_MASTER_KEY_GEN         0x000003D0
880 #define CKM_WTLS_MASTER_KEY_DERIVE          0x000003D1
881 #define CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC   0x000003D2
882 #define CKM_WTLS_PRF                        0x000003D3
883 #define CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE  0x000003D4
884 #define CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE  0x000003D5

886 #define CKM_KEY_WRAP_LYNKS              0x00000400
887 #define CKM_KEY_WRAP_SET_OAEP           0x00000401

889 /* CKM_CMS_SIG is new for v2.20 */
890 #define CKM_CMS_SIG                     0x00000500

892 /* CKM_KIP mechanisms are new for PKCS #11 v2.20 amendment 2 */
893 #define CKM_KIP_DERIVE                  0x00000510
894 #define CKM_KIP_WRAP                    0x00000511
895 #define CKM_KIP_MAC                     0x00000512

897 /* Camellia is new for PKCS #11 v2.20 amendment 3 */
898 #define CKM_CAMELLIA_KEY_GEN            0x00000550
899 #define CKM_CAMELLIA_ECB                0x00000551
900 #define CKM_CAMELLIA_CBC                0x00000552
901 #define CKM_CAMELLIA_MAC                0x00000553
902 #define CKM_CAMELLIA_MAC_GENERAL        0x00000554
903 #define CKM_CAMELLIA_CBC_PAD            0x00000555
904 #define CKM_CAMELLIA_ECB_ENCRYPT_DATA   0x00000556
905 #define CKM_CAMELLIA_CBC_ENCRYPT_DATA   0x00000557
906 #define CKM_CAMELLIA_CTR                0x00000558

908 /* ARIA is new for PKCS #11 v2.20 amendment 3 */
909 #define CKM_ARIA_KEY_GEN                0x00000560
910 #define CKM_ARIA_ECB                    0x00000561
911 #define CKM_ARIA_CBC                    0x00000562
912 #define CKM_ARIA_MAC                    0x00000563
913 #define CKM_ARIA_MAC_GENERAL            0x00000564
914 #define CKM_ARIA_CBC_PAD                0x00000565
915 #define CKM_ARIA_ECB_ENCRYPT_DATA       0x00000566
916 #define CKM_ARIA_CBC_ENCRYPT_DATA       0x00000567

918 /* Fortezza mechanisms */
919 #define CKM_SKIPJACK_KEY_GEN            0x00001000
```

```
#define CKM_SKIPJACK_ECB64              0x00001001
#define CKM_SKIPJACK_CBC64              0x00001002
#define CKM_SKIPJACK_OFB64              0x00001003
#define CKM_SKIPJACK_CFB64              0x00001004
#define CKM_SKIPJACK_CFB32              0x00001005
#define CKM_SKIPJACK_CFB16              0x00001006
#define CKM_SKIPJACK_CFB8               0x00001007
#define CKM_SKIPJACK_WRAP               0x00001008
#define CKM_SKIPJACK_PRIVATE_WRAP       0x00001009
#define CKM_SKIPJACK_RELAYX             0x0000100a
#define CKM_KEA_KEY_PAIR_GEN            0x00001010
#define CKM_KEA_KEY_DERIVE              0x00001011
#define CKM_FORTEZZA_TIMESTAMP          0x00001020
#define CKM_BATON_KEY_GEN               0x00001030
#define CKM_BATON_ECB128                0x00001031
#define CKM_BATON_ECB96                 0x00001032
#define CKM_BATON_CBC128                0x00001033
#define CKM_BATON_COUNTER               0x00001034
#define CKM_BATON_SHUFFLE               0x00001035
#define CKM_BATON_WRAP                  0x00001036

/* CKM_ECDSA_KEY_PAIR_GEN is deprecated in v2.11,
 * CKM_EC_KEY_PAIR_GEN is preferred */
#define CKM_ECDSA_KEY_PAIR_GEN          0x00001040
#define CKM_EC_KEY_PAIR_GEN             0x00001040

#define CKM_ECDSA                       0x00001041
#define CKM_ECDSA_SHA1                  0x00001042

/* CKM_ECDH1_DERIVE, CKM_ECDH1_COFACTOR_DERIVE, and CKM_ECMQV_DERIVE
 * are new for v2.11 */
#define CKM_ECDH1_DERIVE                0x00001050
#define CKM_ECDH1_COFACTOR_DERIVE       0x00001051
#define CKM_ECMQV_DERIVE                0x00001052

#define CKM_JUNIPER_KEY_GEN             0x00001060
#define CKM_JUNIPER_ECB128              0x00001061
#define CKM_JUNIPER_CBC128              0x00001062
#define CKM_JUNIPER_COUNTER             0x00001063
#define CKM_JUNIPER_SHUFFLE             0x00001064
#define CKM_JUNIPER_WRAP                0x00001065
#define CKM_FASTHASH                    0x00001070

/* CKM_AES_KEY_GEN, CKM_AES_ECB, CKM_AES_CBC, CKM_AES_MAC,
 * CKM_AES_MAC_GENERAL, CKM_AES_CBC_PAD, CKM_DSA_PARAMETER_GEN,
 * CKM_DH_PKCS_PARAMETER_GEN, and CKM_X9_42_DH_PARAMETER_GEN are
 * new for v2.11 */
#define CKM_AES_KEY_GEN                 0x00001080
#define CKM_AES_ECB                     0x00001081
#define CKM_AES_CBC                     0x00001082
#define CKM_AES_MAC                     0x00001083
#define CKM_AES_MAC_GENERAL             0x00001084
#define CKM_AES_CBC_PAD                 0x00001085

/* AES counter mode is new for PKCS #11 v2.20 amendment 3 */
#define CKM_AES_CTR                     0x00001086

/* BlowFish and TwoFish are new for v2.20 */
#define CKM_BLOWFISH_KEY_GEN            0x00001090
#define CKM_BLOWFISH_CBC                0x00001091
#define CKM_TWOFISH_KEY_GEN             0x00001092
#define CKM_TWOFISH_CBC                 0x00001093


/* CKM_xxx_ENCRYPT_DATA mechanisms are new for v2.20 */
#define CKM_DES_ECB_ENCRYPT_DATA        0x00001100
```



```
#define CKM_DES_CBC_ENCRYPT_DATA        0x00001101
#define CKM_DES3_ECB_ENCRYPT_DATA       0x00001102
#define CKM_DES3_CBC_ENCRYPT_DATA       0x00001103
#define CKM_AES_ECB_ENCRYPT_DATA        0x00001104
#define CKM_AES_CBC_ENCRYPT_DATA        0x00001105

#define CKM_DSA_PARAMETER_GEN           0x00002000
#define CKM_DH_PKCS_PARAMETER_GEN       0x00002001
#define CKM_X9_42_DH_PARAMETER_GEN      0x00002002

#define CKM_VENDOR_DEFINED              0x80000000

typedef CK_MECHANISM_TYPE CK_PTR CK_MECHANISM_TYPE_PTR;


/* CK_MECHANISM is a structure that specifies a particular
 * mechanism  */
typedef struct CK_MECHANISM {
  CK_MECHANISM_TYPE mechanism;
  CK_VOID_PTR       pParameter;

  /* ulParameterLen was changed from CK_USHORT to CK_ULONG for
   * v2.0 */
  CK_ULONG          ulParameterLen;  /* in bytes */
} CK_MECHANISM;

typedef CK_MECHANISM CK_PTR CK_MECHANISM_PTR;


/* CK_MECHANISM_INFO provides information about a particular
 * mechanism */
typedef struct CK_MECHANISM_INFO {
    CK_ULONG    ulMinKeySize;
    CK_ULONG    ulMaxKeySize;
    CK_FLAGS    flags;
} CK_MECHANISM_INFO;

/* The flags are defined as follows:
 *      Bit Flag               Mask        Meaning */
#define CKF_HW                  0x00000001  /* performed by HW */

/* The flags CKF_ENCRYPT, CKF_DECRYPT, CKF_DIGEST, CKF_SIGN,
 * CKG_SIGN_RECOVER, CKF_VERIFY, CKF_VERIFY_RECOVER,
 * CKF_GENERATE, CKF_GENERATE_KEY_PAIR, CKF_WRAP, CKF_UNWRAP,
 * and CKF_DERIVE are new for v2.0.  They specify whether or not
 * a mechanism can be used for a particular task */
#define CKF_ENCRYPT             0x00000100
#define CKF_DECRYPT             0x00000200
#define CKF_DIGEST              0x00000400
#define CKF_SIGN                0x00000800
#define CKF_SIGN_RECOVER        0x00001000
#define CKF_VERIFY              0x00002000
#define CKF_VERIFY_RECOVER      0x00004000
#define CKF_GENERATE            0x00008000
#define CKF_GENERATE_KEY_PAIR   0x00010000
#define CKF_WRAP                0x00020000
#define CKF_UNWRAP              0x00040000
#define CKF_DERIVE              0x00080000

/* CKF_EC_F_P, CKF_EC_F_2M, CKF_EC_ECPARAMETERS, CKF_EC_NAMEDCURVE,
 * CKF_EC_UNCOMPRESS, and CKF_EC_COMPRESS are new for v2.11. They
 * describe a token's EC capabilities not available in mechanism
 * information. */
#define CKF_EC_F_P              0x00100000
#define CKF_EC_F_2M             0x00200000
#define CKF_EC_ECPARAMETERS     0x00400000
```

```
#define CKF_EC_NAMEDCURVE      0x00800000
#define CKF_EC_UNCOMPRESS      0x01000000
#define CKF_EC_COMPRESS        0x02000000

#define CKF_EXTENSION          0x80000000 /* FALSE for this version */

typedef CK_MECHANISM_INFO CK_PTR CK_MECHANISM_INFO_PTR;


/* CK_RV is a value that identifies the return value of a
 * Cryptoki function */
/* CK_RV was changed from CK_USHORT to CK_ULONG for v2.0 */
typedef CK_ULONG          CK_RV;

#define CKR_OK                                 0x00000000
#define CKR_CANCEL                             0x00000001
#define CKR_HOST_MEMORY                        0x00000002
#define CKR_SLOT_ID_INVALID                    0x00000003

/* CKR_FLAGS_INVALID was removed for v2.0 */

/* CKR_GENERAL_ERROR and CKR_FUNCTION_FAILED are new for v2.0 */
#define CKR_GENERAL_ERROR                      0x00000005
#define CKR_FUNCTION_FAILED                    0x00000006

/* CKR_ARGUMENTS_BAD, CKR_NO_EVENT, CKR_NEED_TO_CREATE_THREADS,
 * and CKR_CANT_LOCK are new for v2.01 */
#define CKR_ARGUMENTS_BAD                      0x00000007
#define CKR_NO_EVENT                           0x00000008
#define CKR_NEED_TO_CREATE_THREADS             0x00000009
#define CKR_CANT_LOCK                          0x0000000A

#define CKR_ATTRIBUTE_READ_ONLY                0x00000010
#define CKR_ATTRIBUTE_SENSITIVE                0x00000011
#define CKR_ATTRIBUTE_TYPE_INVALID             0x00000012
#define CKR_ATTRIBUTE_VALUE_INVALID            0x00000013
#define CKR_DATA_INVALID                       0x00000020
#define CKR_DATA_LEN_RANGE                     0x00000021
#define CKR_DEVICE_ERROR                       0x00000030
#define CKR_DEVICE_MEMORY                      0x00000031
#define CKR_DEVICE_REMOVED                     0x00000032
#define CKR_ENCRYPTED_DATA_INVALID             0x00000040
#define CKR_ENCRYPTED_DATA_LEN_RANGE           0x00000041
#define CKR_FUNCTION_CANCELED                  0x00000050
#define CKR_FUNCTION_NOT_PARALLEL              0x00000051

/* CKR_FUNCTION_NOT_SUPPORTED is new for v2.0 */
#define CKR_FUNCTION_NOT_SUPPORTED             0x00000054

#define CKR_KEY_HANDLE_INVALID                 0x00000060

/* CKR_KEY_SENSITIVE was removed for v2.0 */

#define CKR_KEY_SIZE_RANGE                     0x00000062
#define CKR_KEY_TYPE_INCONSISTENT              0x00000063

/* CKR_KEY_NOT_NEEDED, CKR_KEY_CHANGED, CKR_KEY_NEEDED,
 * CKR_KEY_INDIGESTIBLE, CKR_KEY_FUNCTION_NOT_PERMITTED,
 * CKR_KEY_NOT_WRAPPABLE, and CKR_KEY_UNEXTRACTABLE are new for
 * v2.0 */
#define CKR_KEY_NOT_NEEDED                     0x00000064
#define CKR_KEY_CHANGED                        0x00000065
#define CKR_KEY_NEEDED                         0x00000066
#define CKR_KEY_INDIGESTIBLE                   0x00000067
#define CKR_KEY_FUNCTION_NOT_PERMITTED         0x00000068
#define CKR_KEY_NOT_WRAPPABLE                  0x00000069
```



```
#define CKR_KEY_UNEXTRACTABLE                  0x0000006A

#define CKR_MECHANISM_INVALID                  0x00000070
#define CKR_MECHANISM_PARAM_INVALID            0x00000071

/* CKR_OBJECT_CLASS_INCONSISTENT and CKR_OBJECT_CLASS_INVALID
 * were removed for v2.0 */
#define CKR_OBJECT_HANDLE_INVALID              0x00000082
#define CKR_OPERATION_ACTIVE                   0x00000090
#define CKR_OPERATION_NOT_INITIALIZED          0x00000091
#define CKR_PIN_INCORRECT                      0x000000A0
#define CKR_PIN_INVALID                        0x000000A1
#define CKR_PIN_LEN_RANGE                      0x000000A2

/* CKR_PIN_EXPIRED and CKR_PIN_LOCKED are new for v2.0 */
#define CKR_PIN_EXPIRED                        0x000000A3
#define CKR_PIN_LOCKED                         0x000000A4

#define CKR_SESSION_CLOSED                     0x000000B0
#define CKR_SESSION_COUNT                      0x000000B1
#define CKR_SESSION_HANDLE_INVALID             0x000000B3
#define CKR_SESSION_PARALLEL_NOT_SUPPORTED     0x000000B4
#define CKR_SESSION_READ_ONLY                  0x000000B5
#define CKR_SESSION_EXISTS                     0x000000B6

/* CKR_SESSION_READ_ONLY_EXISTS and
 * CKR_SESSION_READ_WRITE_SO_EXISTS are new for v2.0 */
#define CKR_SESSION_READ_ONLY_EXISTS           0x000000B7
#define CKR_SESSION_READ_WRITE_SO_EXISTS       0x000000B8

#define CKR_SIGNATURE_INVALID                  0x000000C0
#define CKR_SIGNATURE_LEN_RANGE                0x000000C1
#define CKR_TEMPLATE_INCOMPLETE                0x000000D0
#define CKR_TEMPLATE_INCONSISTENT              0x000000D1
#define CKR_TOKEN_NOT_PRESENT                  0x000000E0
#define CKR_TOKEN_NOT_RECOGNIZED               0x000000E1
#define CKR_TOKEN_WRITE_PROTECTED              0x000000E2
#define CKR_UNWRAPPING_KEY_HANDLE_INVALID      0x000000F0
#define CKR_UNWRAPPING_KEY_SIZE_RANGE          0x000000F1
#define CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT   0x000000F2
#define CKR_USER_ALREADY_LOGGED_IN             0x00000100
#define CKR_USER_NOT_LOGGED_IN                 0x00000101
#define CKR_USER_PIN_NOT_INITIALIZED           0x00000102
#define CKR_USER_TYPE_INVALID                  0x00000103

/* CKR_USER_ANOTHER_ALREADY_LOGGED_IN and CKR_USER_TOO_MANY_TYPES
 * are new to v2.01 */
#define CKR_USER_ANOTHER_ALREADY_LOGGED_IN     0x00000104
#define CKR_USER_TOO_MANY_TYPES                0x00000105

#define CKR_WRAPPED_KEY_INVALID                0x00000110
#define CKR_WRAPPED_KEY_LEN_RANGE              0x00000112
#define CKR_WRAPPING_KEY_HANDLE_INVALID        0x00000113
#define CKR_WRAPPING_KEY_SIZE_RANGE            0x00000114
#define CKR_WRAPPING_KEY_TYPE_INCONSISTENT     0x00000115
#define CKR_RANDOM_SEED_NOT_SUPPORTED          0x00000120

/* These are new to v2.0 */
#define CKR_RANDOM_NO_RNG                      0x00000121

/* These are new to v2.11 */
#define CKR_DOMAIN_PARAMS_INVALID              0x00000130

/* These are new to v2.0 */
#define CKR_BUFFER_TOO_SMALL                   0x00000150
#define CKR_SAVED_STATE_INVALID                0x00000160
```

```
#define CKR_INFORMATION_SENSITIVE              0x00000170
#define CKR_STATE_UNSAVEABLE                   0x00000180

/* These are new to v2.01 */
#define CKR_CRYPTOKI_NOT_INITIALIZED           0x00000190
#define CKR_CRYPTOKI_ALREADY_INITIALIZED       0x00000191
#define CKR_MUTEX_BAD                          0x000001A0
#define CKR_MUTEX_NOT_LOCKED                   0x000001A1

/* The following return values are new for PKCS #11 v2.20 amendment 3 */
#define CKR_NEW_PIN_MODE                       0x000001B0
#define CKR_NEXT_OTP                           0x000001B1

/* This is new to v2.20 */
#define CKR_FUNCTION_REJECTED                  0x00000200

#define CKR_VENDOR_DEFINED                     0x80000000


/* CK_NOTIFY is an application callback that processes events */
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_NOTIFY)(
  CK_SESSION_HANDLE hSession,     /* the session's handle */
  CK_NOTIFICATION   event,
  CK_VOID_PTR       pApplication  /* passed to C_OpenSession */
);


/* CK_FUNCTION_LIST is a structure holding a Cryptoki spec
 * version and pointers of appropriate types to all the
 * Cryptoki functions */
/* CK_FUNCTION_LIST is new for v2.0 */
typedef struct CK_FUNCTION_LIST CK_FUNCTION_LIST;

typedef CK_FUNCTION_LIST CK_PTR CK_FUNCTION_LIST_PTR;

typedef CK_FUNCTION_LIST_PTR CK_PTR CK_FUNCTION_LIST_PTR_PTR;


/* CK_CREATEMUTEX is an application callback for creating a
 * mutex object */
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_CREATEMUTEX)(
  CK_VOID_PTR_PTR ppMutex  /* location to receive ptr to mutex */
);


/* CK_DESTROYMUTEX is an application callback for destroying a
 * mutex object */
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_DESTROYMUTEX)(
  CK_VOID_PTR pMutex  /* pointer to mutex */
);


/* CK_LOCKMUTEX is an application callback for locking a mutex */
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_LOCKMUTEX)(
  CK_VOID_PTR pMutex  /* pointer to mutex */
);


/* CK_UNLOCKMUTEX is an application callback for unlocking a
 * mutex */
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_UNLOCKMUTEX)(
  CK_VOID_PTR pMutex  /* pointer to mutex */
);


/* CK_C_INITIALIZE_ARGS provides the optional arguments to
```



```
 * C_Initialize */
typedef struct CK_C_INITIALIZE_ARGS {
  CK_CREATEMUTEX CreateMutex;
  CK_DESTROYMUTEX DestroyMutex;
  CK_LOCKMUTEX LockMutex;
  CK_UNLOCKMUTEX UnlockMutex;
  CK_FLAGS flags;
  CK_VOID_PTR pReserved;
} CK_C_INITIALIZE_ARGS;

/* flags: bit flags that provide capabilities of the slot
 *      Bit Flag                          Mask       Meaning
 */
#define CKF_LIBRARY_CANT_CREATE_OS_THREADS 0x00000001
#define CKF_OS_LOCKING_OK                  0x00000002

typedef CK_C_INITIALIZE_ARGS CK_PTR CK_C_INITIALIZE_ARGS_PTR;


/* additional flags for parameters to functions */

/* CKF_DONT_BLOCK is for the function C_WaitForSlotEvent */
#define CKF_DONT_BLOCK     1

/* CK_RSA_PKCS_OAEP_MGF_TYPE is new for v2.10.
 * CK_RSA_PKCS_OAEP_MGF_TYPE  is used to indicate the Message
 * Generation Function (MGF) applied to a message block when
 * formatting a message block for the PKCS #1 OAEP encryption
 * scheme. */
typedef CK_ULONG CK_RSA_PKCS_MGF_TYPE;

typedef CK_RSA_PKCS_MGF_TYPE CK_PTR CK_RSA_PKCS_MGF_TYPE_PTR;

/* The following MGFs are defined */
/* CKG_MGF1_SHA256, CKG_MGF1_SHA384, and CKG_MGF1_SHA512
 * are new for v2.20 */
#define CKG_MGF1_SHA1         0x00000001
#define CKG_MGF1_SHA256       0x00000002
#define CKG_MGF1_SHA384       0x00000003
#define CKG_MGF1_SHA512       0x00000004
/* SHA-224 is new for PKCS #11 v2.20 amendment 3 */
#define CKG_MGF1_SHA224       0x00000005

/* CK_RSA_PKCS_OAEP_SOURCE_TYPE is new for v2.10.
 * CK_RSA_PKCS_OAEP_SOURCE_TYPE  is used to indicate the source
 * of the encoding parameter when formatting a message block
 * for the PKCS #1 OAEP encryption scheme. */
typedef CK_ULONG CK_RSA_PKCS_OAEP_SOURCE_TYPE;

typedef CK_RSA_PKCS_OAEP_SOURCE_TYPE CK_PTR CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR;

/* The following encoding parameter sources are defined */
#define CKZ_DATA_SPECIFIED    0x00000001

/* CK_RSA_PKCS_OAEP_PARAMS is new for v2.10.
 * CK_RSA_PKCS_OAEP_PARAMS provides the parameters to the
 * CKM_RSA_PKCS_OAEP mechanism. */
typedef struct CK_RSA_PKCS_OAEP_PARAMS {
        CK_MECHANISM_TYPE hashAlg;
        CK_RSA_PKCS_MGF_TYPE mgf;
        CK_RSA_PKCS_OAEP_SOURCE_TYPE source;
        CK_VOID_PTR pSourceData;
        CK_ULONG ulSourceDataLen;
} CK_RSA_PKCS_OAEP_PARAMS;

typedef CK_RSA_PKCS_OAEP_PARAMS CK_PTR CK_RSA_PKCS_OAEP_PARAMS_PTR;
```

```
/* CK_RSA_PKCS_PSS_PARAMS is new for v2.11.
 * CK_RSA_PKCS_PSS_PARAMS provides the parameters to the
 * CKM_RSA_PKCS_PSS mechanism(s). */
typedef struct CK_RSA_PKCS_PSS_PARAMS {
        CK_MECHANISM_TYPE    hashAlg;
        CK_RSA_PKCS_MGF_TYPE mgf;
        CK_ULONG             sLen;
} CK_RSA_PKCS_PSS_PARAMS;

typedef CK_RSA_PKCS_PSS_PARAMS CK_PTR CK_RSA_PKCS_PSS_PARAMS_PTR;

/* CK_EC_KDF_TYPE is new for v2.11. */
typedef CK_ULONG CK_EC_KDF_TYPE;

/* The following EC Key Derivation Functions are defined */
#define CKD_NULL                0x00000001
#define CKD_SHA1_KDF            0x00000002

/* CK_ECDH1_DERIVE_PARAMS is new for v2.11.
 * CK_ECDH1_DERIVE_PARAMS provides the parameters to the
 * CKM_ECDH1_DERIVE and CKM_ECDH1_COFACTOR_DERIVE mechanisms,
 * where each party contributes one key pair.
 */
typedef struct CK_ECDH1_DERIVE_PARAMS {
  CK_EC_KDF_TYPE kdf;
  CK_ULONG ulSharedDataLen;
  CK_BYTE_PTR pSharedData;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
} CK_ECDH1_DERIVE_PARAMS;

typedef CK_ECDH1_DERIVE_PARAMS CK_PTR CK_ECDH1_DERIVE_PARAMS_PTR;


/* CK_ECDH2_DERIVE_PARAMS is new for v2.11.
 * CK_ECDH2_DERIVE_PARAMS provides the parameters to the
 * CKM_ECMQV_DERIVE mechanism, where each party contributes two key pairs. */
typedef struct CK_ECDH2_DERIVE_PARAMS {
  CK_EC_KDF_TYPE kdf;
  CK_ULONG ulSharedDataLen;
  CK_BYTE_PTR pSharedData;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
  CK_ULONG ulPrivateDataLen;
  CK_OBJECT_HANDLE hPrivateData;
  CK_ULONG ulPublicDataLen2;
  CK_BYTE_PTR pPublicData2;
} CK_ECDH2_DERIVE_PARAMS;

typedef CK_ECDH2_DERIVE_PARAMS CK_PTR CK_ECDH2_DERIVE_PARAMS_PTR;

typedef struct CK_ECMQV_DERIVE_PARAMS {
  CK_EC_KDF_TYPE kdf;
  CK_ULONG ulSharedDataLen;
  CK_BYTE_PTR pSharedData;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
  CK_ULONG ulPrivateDataLen;
  CK_OBJECT_HANDLE hPrivateData;
  CK_ULONG ulPublicDataLen2;
  CK_BYTE_PTR pPublicData2;
  CK_OBJECT_HANDLE publicKey;
} CK_ECMQV_DERIVE_PARAMS;

typedef CK_ECMQV_DERIVE_PARAMS CK_PTR CK_ECMQV_DERIVE_PARAMS_PTR;
```



```
/* Typedefs and defines for the CKM_X9_42_DH_KEY_PAIR_GEN and the
 * CKM_X9_42_DH_PARAMETER_GEN mechanisms (new for PKCS #11 v2.11) */
typedef CK_ULONG CK_X9_42_DH_KDF_TYPE;
typedef CK_X9_42_DH_KDF_TYPE CK_PTR CK_X9_42_DH_KDF_TYPE_PTR;

/* The following X9.42 DH key derivation functions are defined
   (besides CKD_NULL already defined : */
#define CKD_SHA1_KDF_ASN1       0x00000003
#define CKD_SHA1_KDF_CONCATENATE 0x00000004

/* CK_X9_42_DH1_DERIVE_PARAMS is new for v2.11.
 * CK_X9_42_DH1_DERIVE_PARAMS provides the parameters to the
 * CKM_X9_42_DH_DERIVE key derivation mechanism, where each party
 * contributes one key pair */
typedef struct CK_X9_42_DH1_DERIVE_PARAMS {
  CK_X9_42_DH_KDF_TYPE kdf;
  CK_ULONG ulOtherInfoLen;
  CK_BYTE_PTR pOtherInfo;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
} CK_X9_42_DH1_DERIVE_PARAMS;

typedef struct CK_X9_42_DH1_DERIVE_PARAMS CK_PTR CK_X9_42_DH1_DERIVE_PARAMS_PTR;

/* CK_X9_42_DH2_DERIVE_PARAMS is new for v2.11.
 * CK_X9_42_DH2_DERIVE_PARAMS provides the parameters to the
 * CKM_X9_42_DH_HYBRID_DERIVE and CKM_X9_42_MQV_DERIVE key derivation
 * mechanisms, where each party contributes two key pairs */
typedef struct CK_X9_42_DH2_DERIVE_PARAMS {
  CK_X9_42_DH_KDF_TYPE kdf;
  CK_ULONG ulOtherInfoLen;
  CK_BYTE_PTR pOtherInfo;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
  CK_ULONG ulPrivateDataLen;
  CK_OBJECT_HANDLE hPrivateData;
  CK_ULONG ulPublicDataLen2;
  CK_BYTE_PTR pPublicData2;
} CK_X9_42_DH2_DERIVE_PARAMS;

typedef CK_X9_42_DH2_DERIVE_PARAMS CK_PTR CK_X9_42_DH2_DERIVE_PARAMS_PTR;

typedef struct CK_X9_42_MQV_DERIVE_PARAMS {
  CK_X9_42_DH_KDF_TYPE kdf;
  CK_ULONG ulOtherInfoLen;
  CK_BYTE_PTR pOtherInfo;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
  CK_ULONG ulPrivateDataLen;
  CK_OBJECT_HANDLE hPrivateData;
  CK_ULONG ulPublicDataLen2;
  CK_BYTE_PTR pPublicData2;
  CK_OBJECT_HANDLE publicKey;
} CK_X9_42_MQV_DERIVE_PARAMS;

typedef CK_X9_42_MQV_DERIVE_PARAMS CK_PTR CK_X9_42_MQV_DERIVE_PARAMS_PTR;

/* CK_KEA_DERIVE_PARAMS provides the parameters to the
 * CKM_KEA_DERIVE mechanism */
/* CK_KEA_DERIVE_PARAMS is new for v2.0 */
typedef struct CK_KEA_DERIVE_PARAMS {
  CK_BBOOL      isSender;
  CK_ULONG      ulRandomLen;
  CK_BYTE_PTR   pRandomA;
  CK_BYTE_PTR   pRandomB;
```

```
  CK_ULONG      ulPublicDataLen;
  CK_BYTE_PTR   pPublicData;
} CK_KEA_DERIVE_PARAMS;

typedef CK_KEA_DERIVE_PARAMS CK_PTR CK_KEA_DERIVE_PARAMS_PTR;


/* CK_RC2_PARAMS provides the parameters to the CKM_RC2_ECB and
 * CKM_RC2_MAC mechanisms.  An instance of CK_RC2_PARAMS just
 * holds the effective keysize */
typedef CK_ULONG          CK_RC2_PARAMS;

typedef CK_RC2_PARAMS CK_PTR CK_RC2_PARAMS_PTR;


/* CK_RC2_CBC_PARAMS provides the parameters to the CKM_RC2_CBC
 * mechanism */
typedef struct CK_RC2_CBC_PARAMS {
  /* ulEffectiveBits was changed from CK_USHORT to CK_ULONG for
   * v2.0 */
  CK_ULONG      ulEffectiveBits;  /* effective bits (1-1024) */

  CK_BYTE       iv[8];            /* IV for CBC mode */
} CK_RC2_CBC_PARAMS;

typedef CK_RC2_CBC_PARAMS CK_PTR CK_RC2_CBC_PARAMS_PTR;


/* CK_RC2_MAC_GENERAL_PARAMS provides the parameters for the
 * CKM_RC2_MAC_GENERAL mechanism */
/* CK_RC2_MAC_GENERAL_PARAMS is new for v2.0 */
typedef struct CK_RC2_MAC_GENERAL_PARAMS {
  CK_ULONG      ulEffectiveBits;  /* effective bits (1-1024) */
  CK_ULONG      ulMacLength;      /* Length of MAC in bytes */
} CK_RC2_MAC_GENERAL_PARAMS;

typedef CK_RC2_MAC_GENERAL_PARAMS CK_PTR \
  CK_RC2_MAC_GENERAL_PARAMS_PTR;


/* CK_RC5_PARAMS provides the parameters to the CKM_RC5_ECB and
 * CKM_RC5_MAC mechanisms */
/* CK_RC5_PARAMS is new for v2.0 */
typedef struct CK_RC5_PARAMS {
  CK_ULONG      ulWordsize;  /* wordsize in bits */
  CK_ULONG      ulRounds;    /* number of rounds */
} CK_RC5_PARAMS;

typedef CK_RC5_PARAMS CK_PTR CK_RC5_PARAMS_PTR;


/* CK_RC5_CBC_PARAMS provides the parameters to the CKM_RC5_CBC
 * mechanism */
/* CK_RC5_CBC_PARAMS is new for v2.0 */
typedef struct CK_RC5_CBC_PARAMS {
  CK_ULONG      ulWordsize;  /* wordsize in bits */
  CK_ULONG      ulRounds;    /* number of rounds */
  CK_BYTE_PTR   pIv;         /* pointer to IV */
  CK_ULONG      ulIvLen;     /* length of IV in bytes */
} CK_RC5_CBC_PARAMS;

typedef CK_RC5_CBC_PARAMS CK_PTR CK_RC5_CBC_PARAMS_PTR;


/* CK_RC5_MAC_GENERAL_PARAMS provides the parameters for the
 * CKM_RC5_MAC_GENERAL mechanism */
```



```
/* CK_RC5_MAC_GENERAL_PARAMS is new for v2.0 */
typedef struct CK_RC5_MAC_GENERAL_PARAMS {
  CK_ULONG      ulWordsize;   /* wordsize in bits */
  CK_ULONG      ulRounds;     /* number of rounds */
  CK_ULONG      ulMacLength;  /* Length of MAC in bytes */
} CK_RC5_MAC_GENERAL_PARAMS;

typedef CK_RC5_MAC_GENERAL_PARAMS CK_PTR \
  CK_RC5_MAC_GENERAL_PARAMS_PTR;


/* CK_MAC_GENERAL_PARAMS provides the parameters to most block
 * ciphers' MAC_GENERAL mechanisms.  Its value is the length of
 * the MAC */
/* CK_MAC_GENERAL_PARAMS is new for v2.0 */
typedef CK_ULONG          CK_MAC_GENERAL_PARAMS;

typedef CK_MAC_GENERAL_PARAMS CK_PTR CK_MAC_GENERAL_PARAMS_PTR;

/* CK_DES/AES_ECB/CBC_ENCRYPT_DATA_PARAMS are new for v2.20 */
typedef struct CK_DES_CBC_ENCRYPT_DATA_PARAMS {
  CK_BYTE      iv[8];
  CK_BYTE_PTR  pData;
  CK_ULONG     length;
} CK_DES_CBC_ENCRYPT_DATA_PARAMS;

typedef CK_DES_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_DES_CBC_ENCRYPT_DATA_PARAMS_PTR

typedef struct CK_AES_CBC_ENCRYPT_DATA_PARAMS {
  CK_BYTE      iv[16];
  CK_BYTE_PTR  pData;
  CK_ULONG     length;
} CK_AES_CBC_ENCRYPT_DATA_PARAMS;

typedef CK_AES_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_AES_CBC_ENCRYPT_DATA_PARAMS_PTR

/* CK_SKIPJACK_PRIVATE_WRAP_PARAMS provides the parameters to the
 * CKM_SKIPJACK_PRIVATE_WRAP mechanism */
/* CK_SKIPJACK_PRIVATE_WRAP_PARAMS is new for v2.0 */
typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
  CK_ULONG      ulPasswordLen;
  CK_BYTE_PTR   pPassword;
  CK_ULONG      ulPublicDataLen;
  CK_BYTE_PTR   pPublicData;
  CK_ULONG      ulPAndGLen;
  CK_ULONG      ulQLen;
  CK_ULONG      ulRandomLen;
  CK_BYTE_PTR   pRandomA;
  CK_BYTE_PTR   pPrimeP;
  CK_BYTE_PTR   pBaseG;
  CK_BYTE_PTR   pSubprimeQ;
} CK_SKIPJACK_PRIVATE_WRAP_PARAMS;

typedef CK_SKIPJACK_PRIVATE_WRAP_PARAMS CK_PTR \
  CK_SKIPJACK_PRIVATE_WRAP_PTR;


/* CK_SKIPJACK_RELAYX_PARAMS provides the parameters to the
 * CKM_SKIPJACK_RELAYX mechanism */
/* CK_SKIPJACK_RELAYX_PARAMS is new for v2.0 */
typedef struct CK_SKIPJACK_RELAYX_PARAMS {
  CK_ULONG      ulOldWrappedXLen;
  CK_BYTE_PTR   pOldWrappedX;
  CK_ULONG      ulOldPasswordLen;
  CK_BYTE_PTR   pOldPassword;
  CK_ULONG      ulOldPublicDataLen;
```

```
1580   CK_BYTE_PTR   pOldPublicData;
1581   CK_ULONG      ulOldRandomLen;
1582   CK_BYTE_PTR   pOldRandomA;
1583   CK_ULONG      ulNewPasswordLen;
1584   CK_BYTE_PTR   pNewPassword;
1585   CK_ULONG      ulNewPublicDataLen;
1586   CK_BYTE_PTR   pNewPublicData;
1587   CK_ULONG      ulNewRandomLen;
1588   CK_BYTE_PTR   pNewRandomA;
1589 } CK_SKIPJACK_RELAYX_PARAMS;

1591 typedef CK_SKIPJACK_RELAYX_PARAMS CK_PTR \
1592   CK_SKIPJACK_RELAYX_PARAMS_PTR;


1595 typedef struct CK_PBE_PARAMS {
1596   CK_BYTE_PTR      pInitVector;
1597   CK_UTF8CHAR_PTR  pPassword;
1598   CK_ULONG         ulPasswordLen;
1599   CK_BYTE_PTR      pSalt;
1600   CK_ULONG         ulSaltLen;
1601   CK_ULONG         ulIteration;
1602 } CK_PBE_PARAMS;

1604 typedef CK_PBE_PARAMS CK_PTR CK_PBE_PARAMS_PTR;


1607 /* CK_KEY_WRAP_SET_OAEP_PARAMS provides the parameters to the
1608  * CKM_KEY_WRAP_SET_OAEP mechanism */
1609 /* CK_KEY_WRAP_SET_OAEP_PARAMS is new for v2.0 */
1610 typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {
1611   CK_BYTE       bBC;     /* block contents byte */
1612   CK_BYTE_PTR   pX;      /* extra data */
1613   CK_ULONG      ulXLen;  /* length of extra data in bytes */
1614 } CK_KEY_WRAP_SET_OAEP_PARAMS;

1616 typedef CK_KEY_WRAP_SET_OAEP_PARAMS CK_PTR \
1617   CK_KEY_WRAP_SET_OAEP_PARAMS_PTR;


1620 typedef struct CK_SSL3_RANDOM_DATA {
1621   CK_BYTE_PTR  pClientRandom;
1622   CK_ULONG     ulClientRandomLen;
1623   CK_BYTE_PTR  pServerRandom;
1624   CK_ULONG     ulServerRandomLen;
1625 } CK_SSL3_RANDOM_DATA;


1628 typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS {
1629   CK_SSL3_RANDOM_DATA RandomInfo;
1630   CK_VERSION_PTR pVersion;
1631 } CK_SSL3_MASTER_KEY_DERIVE_PARAMS;

1633 typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS CK_PTR \
1634   CK_SSL3_MASTER_KEY_DERIVE_PARAMS_PTR;


1637 typedef struct CK_SSL3_KEY_MAT_OUT {
1638   CK_OBJECT_HANDLE hClientMacSecret;
1639   CK_OBJECT_HANDLE hServerMacSecret;
1640   CK_OBJECT_HANDLE hClientKey;
1641   CK_OBJECT_HANDLE hServerKey;
1642   CK_BYTE_PTR      pIVClient;
1643   CK_BYTE_PTR      pIVServer;
1644 } CK_SSL3_KEY_MAT_OUT;
```



```
1646 typedef CK_SSL3_KEY_MAT_OUT CK_PTR CK_SSL3_KEY_MAT_OUT_PTR;


1649 typedef struct CK_SSL3_KEY_MAT_PARAMS {
1650   CK_ULONG               ulMacSizeInBits;
1651   CK_ULONG               ulKeySizeInBits;
1652   CK_ULONG               ulIVSizeInBits;
1653   CK_BBOOL               bIsExport;
1654   CK_SSL3_RANDOM_DATA    RandomInfo;
1655   CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial;
1656 } CK_SSL3_KEY_MAT_PARAMS;

1658 typedef CK_SSL3_KEY_MAT_PARAMS CK_PTR CK_SSL3_KEY_MAT_PARAMS_PTR;

1660 /* CK_TLS_PRF_PARAMS is new for version 2.20 */
1661 typedef struct CK_TLS_PRF_PARAMS {
1662   CK_BYTE_PTR  pSeed;
1663   CK_ULONG     ulSeedLen;
1664   CK_BYTE_PTR  pLabel;
1665   CK_ULONG     ulLabelLen;
1666   CK_BYTE_PTR  pOutput;
1667   CK_ULONG_PTR pulOutputLen;
1668 } CK_TLS_PRF_PARAMS;

1670 typedef CK_TLS_PRF_PARAMS CK_PTR CK_TLS_PRF_PARAMS_PTR;

1672 /* WTLS is new for version 2.20 */
1673 typedef struct CK_WTLS_RANDOM_DATA {
1674   CK_BYTE_PTR pClientRandom;
1675   CK_ULONG    ulClientRandomLen;
1676   CK_BYTE_PTR pServerRandom;
1677   CK_ULONG    ulServerRandomLen;
1678 } CK_WTLS_RANDOM_DATA;

1680 typedef CK_WTLS_RANDOM_DATA CK_PTR CK_WTLS_RANDOM_DATA_PTR;

1682 typedef struct CK_WTLS_MASTER_KEY_DERIVE_PARAMS {
1683   CK_MECHANISM_TYPE   DigestMechanism;
1684   CK_WTLS_RANDOM_DATA RandomInfo;
1685   CK_BYTE_PTR         pVersion;
1686 } CK_WTLS_MASTER_KEY_DERIVE_PARAMS;

1688 typedef CK_WTLS_MASTER_KEY_DERIVE_PARAMS CK_PTR \
1689   CK_WTLS_MASTER_KEY_DERIVE_PARAMS_PTR;

1691 typedef struct CK_WTLS_PRF_PARAMS {
1692   CK_MECHANISM_TYPE DigestMechanism;
1693   CK_BYTE_PTR       pSeed;
1694   CK_ULONG          ulSeedLen;
1695   CK_BYTE_PTR       pLabel;
1696   CK_ULONG          ulLabelLen;
1697   CK_BYTE_PTR       pOutput;
1698   CK_ULONG_PTR      pulOutputLen;
1699 } CK_WTLS_PRF_PARAMS;

1701 typedef CK_WTLS_PRF_PARAMS CK_PTR CK_WTLS_PRF_PARAMS_PTR;

1703 typedef struct CK_WTLS_KEY_MAT_OUT {
1704   CK_OBJECT_HANDLE hMacSecret;
1705   CK_OBJECT_HANDLE hKey;
1706   CK_BYTE_PTR      pIV;
1707 } CK_WTLS_KEY_MAT_OUT;

1709 typedef CK_WTLS_KEY_MAT_OUT CK_PTR CK_WTLS_KEY_MAT_OUT_PTR;

1711 typedef struct CK_WTLS_KEY_MAT_PARAMS {
```

```
  CK_MECHANISM_TYPE       DigestMechanism;
  CK_ULONG                ulMacSizeInBits;
  CK_ULONG                ulKeySizeInBits;
  CK_ULONG                ulIVSizeInBits;
  CK_ULONG                ulSequenceNumber;
  CK_BBOOL                bIsExport;
  CK_WTLS_RANDOM_DATA     RandomInfo;
  CK_WTLS_KEY_MAT_OUT_PTR pReturnedKeyMaterial;
} CK_WTLS_KEY_MAT_PARAMS;

typedef CK_WTLS_KEY_MAT_PARAMS CK_PTR CK_WTLS_KEY_MAT_PARAMS_PTR;

/* CMS is new for version 2.20 */
typedef struct CK_CMS_SIG_PARAMS {
  CK_OBJECT_HANDLE      certificateHandle;
  CK_MECHANISM_PTR      pSigningMechanism;
  CK_MECHANISM_PTR      pDigestMechanism;
  CK_UTF8CHAR_PTR       pContentType;
  CK_BYTE_PTR           pRequestedAttributes;
  CK_ULONG              ulRequestedAttributesLen;
  CK_BYTE_PTR           pRequiredAttributes;
  CK_ULONG              ulRequiredAttributesLen;
} CK_CMS_SIG_PARAMS;

typedef CK_CMS_SIG_PARAMS CK_PTR CK_CMS_SIG_PARAMS_PTR;

typedef struct CK_KEY_DERIVATION_STRING_DATA {
  CK_BYTE_PTR pData;
  CK_ULONG    ulLen;
} CK_KEY_DERIVATION_STRING_DATA;

typedef CK_KEY_DERIVATION_STRING_DATA CK_PTR \
  CK_KEY_DERIVATION_STRING_DATA_PTR;


/* The CK_EXTRACT_PARAMS is used for the
 * CKM_EXTRACT_KEY_FROM_KEY mechanism.  It specifies which bit
 * of the base key should be used as the first bit of the
 * derived key */
/* CK_EXTRACT_PARAMS is new for v2.0 */
typedef CK_ULONG CK_EXTRACT_PARAMS;

typedef CK_EXTRACT_PARAMS CK_PTR CK_EXTRACT_PARAMS_PTR;

/* CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE is new for v2.10.
 * CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE is used to
 * indicate the Pseudo-Random Function (PRF) used to generate
 * key bits using PKCS #5 PBKDF2. */
typedef CK_ULONG CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE;

typedef CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE CK_PTR CK_PKCS5_PBKD2_PSEUDO_

/* The following PRFs are defined in PKCS #5 v2.0. */
#define CKP_PKCS5_PBKD2_HMAC_SHA1 0x00000001


/* CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE is new for v2.10.
 * CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE is used to indicate the
 * source of the salt value when deriving a key using PKCS #5
 * PBKDF2. */
typedef CK_ULONG CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE;

typedef CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE CK_PTR CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE

/* The following salt value sources are defined in PKCS #5 v2.0. */
#define CKZ_SALT_SPECIFIED        0x00000001
```



```
/* CK_PKCS5_PBKD2_PARAMS is new for v2.10.
 * CK_PKCS5_PBKD2_PARAMS is a structure that provides the
 * parameters to the CKM_PKCS5_PBKD2 mechanism. */
typedef struct CK_PKCS5_PBKD2_PARAMS {
        CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE           saltSource;
        CK_VOID_PTR                                pSaltSourceData;
        CK_ULONG                                   ulSaltSourceDataLen;
        CK_ULONG                                   iterations;
        CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf;
        CK_VOID_PTR                                pPrfData;
        CK_ULONG                                   ulPrfDataLen;
        CK_UTF8CHAR_PTR                            pPassword;
        CK_ULONG_PTR                               ulPasswordLen;
} CK_PKCS5_PBKD2_PARAMS;

typedef CK_PKCS5_PBKD2_PARAMS CK_PTR CK_PKCS5_PBKD2_PARAMS_PTR;

/* All CK_OTP structs are new for PKCS #11 v2.20 amendment 3 */

typedef CK_ULONG CK_OTP_PARAM_TYPE;
typedef CK_OTP_PARAM_TYPE CK_PARAM_TYPE; /* B/w compatibility */

typedef struct CK_OTP_PARAM {
    CK_OTP_PARAM_TYPE type;
    CK_VOID_PTR pValue;
    CK_ULONG ulValueLen;
} CK_OTP_PARAM;

typedef CK_OTP_PARAM CK_PTR CK_OTP_PARAM_PTR;

typedef struct CK_OTP_PARAMS {
    CK_OTP_PARAM_PTR pParams;
    CK_ULONG ulCount;
} CK_OTP_PARAMS;

typedef CK_OTP_PARAMS CK_PTR CK_OTP_PARAMS_PTR;

typedef struct CK_OTP_SIGNATURE_INFO {
    CK_OTP_PARAM_PTR pParams;
    CK_ULONG ulCount;
} CK_OTP_SIGNATURE_INFO;

typedef CK_OTP_SIGNATURE_INFO CK_PTR CK_OTP_SIGNATURE_INFO_PTR;

/* The following OTP-related defines are new for PKCS #11 v2.20 amendment 1 */
#define CK_OTP_VALUE         0
#define CK_OTP_PIN           1
#define CK_OTP_CHALLENGE     2
#define CK_OTP_TIME          3
#define CK_OTP_COUNTER       4
#define CK_OTP_FLAGS         5
#define CK_OTP_OUTPUT_LENGTH 6
#define CK_OTP_OUTPUT_FORMAT 7

/* The following OTP-related defines are new for PKCS #11 v2.20 amendment 1 */
#define CKF_NEXT_OTP         0x00000001
#define CKF_EXCLUDE_TIME     0x00000002
#define CKF_EXCLUDE_COUNTER  0x00000004
#define CKF_EXCLUDE_CHALLENGE 0x00000008
#define CKF_EXCLUDE_PIN      0x00000010
#define CKF_USER_FRIENDLY_OTP 0x00000020

/* CK_KIP_PARAMS is new for PKCS #11 v2.20 amendment 2 */
typedef struct CK_KIP_PARAMS {
    CK_MECHANISM_PTR  pMechanism;
```

```
1844     CK_OBJECT_HANDLE  hKey;
1845     CK_BYTE_PTR       pSeed;
1846     CK_ULONG          ulSeedLen;
1847 } CK_KIP_PARAMS;

1849 typedef CK_KIP_PARAMS CK_PTR CK_KIP_PARAMS_PTR;

1851 /* CK_AES_CTR_PARAMS is new for PKCS #11 v2.20 amendment 3 */
1852 typedef struct CK_AES_CTR_PARAMS {
1853     CK_ULONG ulCounterBits;
1854     CK_BYTE cb[16];
1855 } CK_AES_CTR_PARAMS;

1857 typedef CK_AES_CTR_PARAMS CK_PTR CK_AES_CTR_PARAMS_PTR;

1859 /* CK_CAMELLIA_CTR_PARAMS is new for PKCS #11 v2.20 amendment 3 */
1860 typedef struct CK_CAMELLIA_CTR_PARAMS {
1861     CK_ULONG ulCounterBits;
1862     CK_BYTE cb[16];
1863 } CK_CAMELLIA_CTR_PARAMS;

1865 typedef CK_CAMELLIA_CTR_PARAMS CK_PTR CK_CAMELLIA_CTR_PARAMS_PTR;

1867 /* CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS is new for PKCS #11 v2.20 amendment 3 */
1868 typedef struct CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS {
1869     CK_BYTE      iv[16];
1870     CK_BYTE_PTR  pData;
1871     CK_ULONG     length;
1872 } CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS;

1874 typedef CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_CAMELLIA_CBC_ENCRYPT_DATA_

1876 /* CK_ARIA_CBC_ENCRYPT_DATA_PARAMS is new for PKCS #11 v2.20 amendment 3 */
1877 typedef struct CK_ARIA_CBC_ENCRYPT_DATA_PARAMS {
1878     CK_BYTE      iv[16];
1879     CK_BYTE_PTR  pData;
1880     CK_ULONG     length;
1881 } CK_ARIA_CBC_ENCRYPT_DATA_PARAMS;

1883 typedef CK_ARIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_ARIA_CBC_ENCRYPT_DATA_PARAMS_P

1885 #endif
1886 #endif /* ! codereview */
```

```
**********************************************************
   20631 Wed Aug 13 19:32:54 2014
new/usr/src/lib/openssl/libsunw_crypto/bn/bn_const.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/bn/knownprimes.c */
   2 /* Insert boilerplate */

   4 #include <openssl/bn.h>
   4 #include "bn.h"

   6 /* "First Oakley Default Group" from RFC2409, section 6.1.
   7  *
   8  * The prime is: 2^768 - 2 ^704 - 1 + 2^64 * { [2^638 pi] + 149686 }
   9  *
  10  * RFC2409 specifies a generator of 2.
  11  * RFC2412 specifies a generator of of 22.
  12  */

  14 BIGNUM *get_rfc2409_prime_768(BIGNUM *bn)
  15         {
  16         static const unsigned char RFC2409_PRIME_768[]={
  17                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
  18                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
  19                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
  20                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
  21                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
  22                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
  23                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
  24                 0xA6,0x3A,0x36,0x20,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
  25                 };
  26         return BN_bin2bn(RFC2409_PRIME_768,sizeof(RFC2409_PRIME_768),bn);
  27         }

  29 /* "Second Oakley Default Group" from RFC2409, section 6.2.
  30  *
  31  * The prime is: 2^1024 - 2^960 - 1 + 2^64 * { [2^894 pi] + 129093 }.
  32  *
  33  * RFC2409 specifies a generator of 2.
  34  * RFC2412 specifies a generator of 22.
  35  */

  37 BIGNUM *get_rfc2409_prime_1024(BIGNUM *bn)
  38         {
  39         static const unsigned char RFC2409_PRIME_1024[]={
  40                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
  41                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
  42                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
  43                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
  44                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
  45                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
  46                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
  47                 0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
  48                 0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
  49                 0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE6,0x53,0x81,
  50                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
  51                 };
  52         return BN_bin2bn(RFC2409_PRIME_1024,sizeof(RFC2409_PRIME_1024),bn);
  53         }

  55 /* "1536-bit MODP Group" from RFC3526, Section 2.
  56  *
  57  * The prime is: 2^1536 - 2^1472 - 1 + 2^64 * { [2^1406 pi] + 741804 }
  58  *
  59  * RFC3526 specifies a generator of 2.
  60  * RFC2312 specifies a generator of 22.
```



```
  61  */

  63 BIGNUM *get_rfc3526_prime_1536(BIGNUM *bn)
  64         {
  65         static const unsigned char RFC3526_PRIME_1536[]={
  66                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
  67                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
  68                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
  69                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
  70                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
  71                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
  72                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
  73                 0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
  74                 0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
  75                 0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
  76                 0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
  77                 0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
  78                 0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
  79                 0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
  80                 0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
  81                 0xCA,0x23,0x73,0x27,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
  82                 };
  83         return BN_bin2bn(RFC3526_PRIME_1536,sizeof(RFC3526_PRIME_1536),bn);
  84         }

  86 /* "2048-bit MODP Group" from RFC3526, Section 3.
  87  *
  88  * The prime is: 2^2048 - 2^1984 - 1 + 2^64 * { [2^1918 pi] + 124476 }
  89  *
  90  * RFC3526 specifies a generator of 2.
  91  */

  93 BIGNUM *get_rfc3526_prime_2048(BIGNUM *bn)
  94         {
  95         static const unsigned char RFC3526_PRIME_2048[]={
  96                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
  97                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
  98                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
  99                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
 100                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
 101                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
 102                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
 103                 0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
 104                 0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
 105                 0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
 106                 0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
 107                 0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
 108                 0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
 109                 0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
 110                 0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
 111                 0xCA,0x18,0x21,0x7C,0x32,0x90,0x5E,0x46,0x2E,0x36,0xCE,0x3B,
 112                 0xE3,0x9E,0x77,0x2C,0x18,0x0E,0x86,0x03,0x9B,0x27,0x83,0xA2,
 113                 0xEC,0x07,0xA2,0x8F,0xB5,0xC5,0x5D,0xF0,0x6F,0x4C,0x52,0xC9,
 114                 0xDE,0x2B,0xCB,0xF6,0x95,0x58,0x17,0x18,0x39,0x95,0x49,0x7C,
 115                 0xEA,0x95,0x6A,0xE5,0x15,0xD2,0x26,0x18,0x98,0xFA,0x05,0x10,
 116                 0x15,0x72,0x8E,0x5A,0x8A,0xAC,0xAA,0x68,0xFF,0xFF,0xFF,0xFF,
 117                 0xFF,0xFF,0xFF,0xFF,
 118                 };
 119         return BN_bin2bn(RFC3526_PRIME_2048,sizeof(RFC3526_PRIME_2048),bn);
 120         }

 122 /* "3072-bit MODP Group" from RFC3526, Section 4.
 123  *
 124  * The prime is: 2^3072 - 2^3008 - 1 + 2^64 * { [2^2942 pi] + 1690314 }
 125  *
 126  * RFC3526 specifies a generator of 2.
```

```
 127  */

 129 BIGNUM *get_rfc3526_prime_3072(BIGNUM *bn)
 130         {
 131         static const unsigned char RFC3526_PRIME_3072[]={
 132                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
 133                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
 134                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
 135                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
 136                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
 137                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
 138                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
 139                 0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
 140                 0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
 141                 0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
 142                 0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
 143                 0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
 144                 0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
 145                 0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
 146                 0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
 147                 0xCA,0x18,0x21,0x7C,0x32,0x90,0x5E,0x46,0x2E,0x36,0xCE,0x3B,
 148                 0xE3,0x9E,0x77,0x2C,0x18,0x0E,0x86,0x03,0x9B,0x27,0x83,0xA2,
 149                 0xEC,0x07,0xA2,0x8F,0xB5,0xC5,0x5D,0xF0,0x6F,0x4C,0x52,0xC9,
 150                 0xDE,0x2B,0xCB,0xF6,0x95,0x58,0x17,0x18,0x39,0x95,0x49,0x7C,
 151                 0xEA,0x95,0x6A,0xE5,0x15,0xD2,0x26,0x18,0x98,0xFA,0x05,0x10,
 152                 0x15,0x72,0x8E,0x5A,0x8A,0xAA,0xC4,0x2D,0xAD,0x33,0x17,0x0D,
 153                 0x04,0x50,0x7A,0x33,0xA8,0x55,0x21,0xAB,0xDF,0x1C,0xBA,0x64,
 154                 0xEC,0xFB,0x85,0x04,0x58,0xDB,0xEF,0x0A,0x8A,0xEA,0x71,0x57,
 155                 0x5D,0x06,0x0C,0x7D,0xB3,0x97,0x0F,0x85,0xA6,0xE1,0xE4,0xC7,
 156                 0xAB,0xF5,0xAE,0x8C,0xDB,0x09,0x33,0xD7,0x1E,0x8C,0x94,0xE0,
 157                 0x4A,0x25,0x61,0x9D,0xCE,0xE3,0xD2,0x26,0x1A,0xD2,0xEE,0x6B,
 158                 0xF1,0x2F,0xFA,0x06,0xD9,0x8A,0x08,0x64,0xD8,0x76,0x02,0x73,
 159                 0x3E,0xC8,0x6A,0x64,0x52,0x1F,0x2B,0x18,0x17,0x7B,0x20,0x0C,
 160                 0xBB,0xE1,0x17,0x57,0x7A,0x61,0x5D,0x6C,0x77,0x09,0x88,0xC0,
 161                 0xBA,0xD9,0x46,0xE2,0x08,0xE2,0x4F,0xA0,0x74,0xE5,0xAB,0x31,
 162                 0x43,0xDB,0x5B,0xFC,0xE0,0xFD,0x10,0x8E,0x4B,0x82,0xD1,0x20,
 163                 0xA9,0x3A,0xD2,0xCA,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
 164                 };
 165         return BN_bin2bn(RFC3526_PRIME_3072,sizeof(RFC3526_PRIME_3072),bn);
 166         }

 168 /* "4096-bit MODP Group" from RFC3526, Section 5.
 169  *
 170  * The prime is: 2^4096 - 2^4032 - 1 + 2^64 * { [2^3966 pi] + 240904 }
 171  *
 172  * RFC3526 specifies a generator of 2.
 173  */

 175 BIGNUM *get_rfc3526_prime_4096(BIGNUM *bn)
 176         {
 177         static const unsigned char RFC3526_PRIME_4096[]={
 178                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
 179                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
 180                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
 181                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
 182                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
 183                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
 184                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
 185                 0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
 186                 0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
 187                 0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
 188                 0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
 189                 0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
 190                 0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
 191                 0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
 192                 0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
```



```
 193                 0xCA,0x18,0x21,0x7C,0x32,0x90,0x5E,0x46,0x2E,0x36,0xCE,0x3B,
 194                 0xE3,0x9E,0x77,0x2C,0x18,0x0E,0x86,0x03,0x9B,0x27,0x83,0xA2,
 195                 0xEC,0x07,0xA2,0x8F,0xB5,0xC5,0x5D,0xF0,0x6F,0x4C,0x52,0xC9,
 196                 0xDE,0x2B,0xCB,0xF6,0x95,0x58,0x17,0x18,0x39,0x95,0x49,0x7C,
 197                 0xEA,0x95,0x6A,0xE5,0x15,0xD2,0x26,0x18,0x98,0xFA,0x05,0x10,
 198                 0x15,0x72,0x8E,0x5A,0x8A,0xAA,0xC4,0x2D,0xAD,0x33,0x17,0x0D,
 199                 0x04,0x50,0x7A,0x33,0xA8,0x55,0x21,0xAB,0xDF,0x1C,0xBA,0x64,
 200                 0xEC,0xFB,0x85,0x04,0x58,0xDB,0xEF,0x0A,0x8A,0xEA,0x71,0x57,
 201                 0x5D,0x06,0x0C,0x7D,0xB3,0x97,0x0F,0x85,0xA6,0xE1,0xE4,0xC7,
 202                 0xAB,0xF5,0xAE,0x8C,0xDB,0x09,0x33,0xD7,0x1E,0x8C,0x94,0xE0,
 203                 0x4A,0x25,0x61,0x9D,0xCE,0xE3,0xD2,0x26,0x1A,0xD2,0xEE,0x6B,
 204                 0xF1,0x2F,0xFA,0x06,0xD9,0x8A,0x08,0x64,0xD8,0x76,0x02,0x73,
 205                 0x3E,0xC8,0x6A,0x64,0x52,0x1F,0x2B,0x18,0x17,0x7B,0x20,0x0C,
 206                 0xBB,0xE1,0x17,0x57,0x7A,0x61,0x5D,0x6C,0x77,0x09,0x88,0xC0,
 207                 0xBA,0xD9,0x46,0xE2,0x08,0xE2,0x4F,0xA0,0x74,0xE5,0xAB,0x31,
 208                 0x43,0xDB,0x5B,0xFC,0xE0,0xFD,0x10,0x8E,0x4B,0x82,0xD1,0x20,
 209                 0xA9,0x21,0x08,0x01,0x1A,0x72,0x3C,0x12,0xA7,0x87,0xE6,0xD7,
 210                 0x88,0x71,0x9A,0x10,0xBD,0xBA,0x5B,0x26,0x99,0xC3,0x27,0x18,
 211                 0x6A,0xF4,0xE2,0x3C,0x1A,0x94,0x68,0x34,0xB6,0x15,0x0B,0xDA,
 212                 0x25,0x83,0xE9,0xCA,0x2A,0xD4,0x4C,0xE8,0xDB,0xBB,0xC2,0xDB,
 213                 0x04,0xDE,0x8E,0xF9,0x2E,0x8E,0xFC,0x14,0x1F,0xBE,0xCA,0xA6,
 214                 0x28,0x7C,0x59,0x47,0x4E,0x6B,0xC0,0x5D,0x99,0xB2,0x96,0x4F,
 215                 0xA0,0x90,0xC3,0xA2,0x23,0x3B,0xA1,0x86,0x51,0x5B,0xE7,0xED,
 216                 0x1F,0x61,0x29,0x70,0xCE,0xE2,0xD7,0xAF,0xB8,0x1B,0xDD,0x76,
 217                 0x21,0x70,0x48,0x1C,0xD0,0x06,0x91,0x27,0xD5,0xB0,0x5A,0xA9,
 218                 0x93,0xB4,0xEA,0x98,0x8D,0x8F,0xDD,0xC1,0x86,0xFF,0xB7,0xDC,
 219                 0x90,0xA6,0xC0,0x8F,0x4D,0xF4,0x35,0xC9,0x34,0x06,0x31,0x99,
 220                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
 221                 };
 222         return BN_bin2bn(RFC3526_PRIME_4096,sizeof(RFC3526_PRIME_4096),bn);
 223         }

 225 /* "6144-bit MODP Group" from RFC3526, Section 6.
 226  *
 227  * The prime is: 2^6144 - 2^6080 - 1 + 2^64 * { [2^6014 pi] + 929484 }
 228  *
 229  * RFC3526 specifies a generator of 2.
 230  */

 232 BIGNUM *get_rfc3526_prime_6144(BIGNUM *bn)
 233         {
 234         static const unsigned char RFC3526_PRIME_6144[]={
 235                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
 236                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
 237                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
 238                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
 239                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
 240                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
 241                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
 242                 0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
 243                 0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
 244                 0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
 245                 0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
 246                 0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
 247                 0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
 248                 0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
 249                 0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
 250                 0xCA,0x18,0x21,0x7C,0x32,0x90,0x5E,0x46,0x2E,0x36,0xCE,0x3B,
 251                 0xE3,0x9E,0x77,0x2C,0x18,0x0E,0x86,0x03,0x9B,0x27,0x83,0xA2,
 252                 0xEC,0x07,0xA2,0x8F,0xB5,0xC5,0x5D,0xF0,0x6F,0x4C,0x52,0xC9,
 253                 0xDE,0x2B,0xCB,0xF6,0x95,0x58,0x17,0x18,0x39,0x95,0x49,0x7C,
 254                 0xEA,0x95,0x6A,0xE5,0x15,0xD2,0x26,0x18,0x98,0xFA,0x05,0x10,
 255                 0x15,0x72,0x8E,0x5A,0x8A,0xAA,0xC4,0x2D,0xAD,0x33,0x17,0x0D,
 256                 0x04,0x50,0x7A,0x33,0xA8,0x55,0x21,0xAB,0xDF,0x1C,0xBA,0x64,
 257                 0xEC,0xFB,0x85,0x04,0x58,0xDB,0xEF,0x0A,0x8A,0xEA,0x71,0x57,
 258                 0x5D,0x06,0x0C,0x7D,0xB3,0x97,0x0F,0x85,0xA6,0xE1,0xE4,0xC7,
```

```
259                 0xAB,0xF5,0xAE,0x8C,0xDB,0x09,0x33,0xD7,0x1E,0x8C,0x94,0xE0,
260                 0x4A,0x25,0x61,0x9D,0xCE,0xE3,0xD2,0x26,0x1A,0xD2,0xEE,0x6B,
261                 0xF1,0x2F,0xFA,0x06,0xD9,0x8A,0x08,0x64,0xD8,0x76,0x02,0x73,
262                 0x3E,0xC8,0x6A,0x64,0x52,0x1F,0x2B,0x18,0x17,0x7B,0x20,0x0C,
263                 0xBB,0xE1,0x17,0x57,0x7A,0x61,0x5D,0x6C,0x77,0x09,0x88,0xC0,
264                 0xBA,0xD9,0x46,0xE2,0x08,0xE2,0x4F,0xA0,0x74,0xE5,0xAB,0x31,
265                 0x43,0xDB,0x5B,0xFC,0xE0,0xFD,0x10,0x8E,0x4B,0x82,0xD1,0x20,
266                 0xA9,0x21,0x08,0x01,0x1A,0x72,0x3C,0x12,0xA7,0x87,0xE6,0xD7,
267                 0x88,0x71,0x9A,0x10,0xBD,0xBA,0x5B,0x26,0x99,0xC3,0x27,0x18,
268                 0x6A,0xF4,0xE2,0x3C,0x1A,0x94,0x68,0x34,0xB6,0x15,0x0B,0xDA,
269                 0x25,0x83,0xE9,0xCA,0x2A,0xD4,0x4C,0xE8,0xDB,0xBB,0xC2,0xDB,
270                 0x04,0xDE,0x8E,0xF9,0x2E,0x8E,0xFC,0x14,0x1F,0xBE,0xCA,0xA6,
271                 0x28,0x7C,0x59,0x47,0x4E,0x6B,0xC0,0x5D,0x99,0xB2,0x96,0x4F,
272                 0xA0,0x90,0xC3,0xA2,0x23,0x3B,0xA1,0x86,0x51,0x5B,0xE7,0xED,
273                 0x1F,0x61,0x29,0x70,0xCE,0xE2,0xD7,0xAF,0xB8,0x1B,0xDD,0x76,
274                 0x21,0x70,0x48,0x1C,0xD0,0x06,0x91,0x27,0xD5,0xB0,0x5A,0xA9,
275                 0x93,0xB4,0xEA,0x98,0x8D,0x8F,0xDD,0xC1,0x86,0xFF,0xB7,0xDC,
276                 0x90,0xA6,0xC0,0x8F,0x4D,0xF4,0x35,0xC9,0x34,0x02,0x84,0x92,
277                 0x36,0xC3,0xFA,0xB4,0xD2,0x7C,0x70,0x26,0xC1,0xD4,0xDC,0xB2,
278                 0x60,0x26,0x46,0xDE,0xC9,0x75,0x1E,0x76,0x3D,0xBA,0x37,0xBD,
279                 0xF8,0xFF,0x94,0x06,0xAD,0x9E,0x53,0x0E,0xE5,0xDB,0x38,0x2F,
280                 0x41,0x30,0x01,0xAE,0xB0,0x6A,0x53,0xED,0x90,0x27,0xD8,0x31,
281                 0x17,0x97,0x27,0xB0,0x86,0x5A,0x89,0x18,0xDA,0x3E,0xDB,0xEB,
282                 0xCF,0x9B,0x14,0xED,0x44,0xCE,0x6C,0xBA,0xCE,0xD4,0xBB,0x1B,
283                 0xDB,0x7F,0x14,0x47,0xE6,0xCC,0x25,0x4B,0x33,0x20,0x51,0x51,
284                 0x2B,0xD7,0xAF,0x42,0x6F,0xB8,0xF4,0x01,0x37,0x8C,0xD2,0xBF,
285                 0x59,0x83,0xCA,0x01,0xC6,0x4B,0x92,0xEC,0xF0,0x32,0xEA,0x15,
286                 0xD1,0x72,0x1D,0x03,0xF4,0x82,0xD7,0xCE,0x6E,0x74,0xFE,0xF6,
287                 0xD5,0x5E,0x70,0x2F,0x46,0x98,0x0C,0x82,0xB5,0xA8,0x40,0x31,
288                 0x90,0x0B,0x1C,0x9E,0x59,0xE7,0xC9,0x7F,0xBE,0xC7,0xE8,0xF3,
289                 0x23,0xA9,0x7A,0x7E,0x36,0xCC,0x88,0xBE,0x0F,0x1D,0x45,0xB7,
290                 0xFF,0x58,0x5A,0xC5,0x4B,0xD4,0x07,0xB2,0x2B,0x41,0x54,0xAA,
291                 0xCC,0x8F,0x6D,0x7E,0xBF,0x48,0xE1,0xD8,0x14,0xCC,0x5E,0xD2,
292                 0x0F,0x80,0x37,0xE0,0xA7,0x97,0x15,0xEE,0xF2,0x9B,0xE3,0x28,
293                 0x06,0xA1,0xD5,0x8B,0xB7,0xC5,0xDA,0x76,0xF5,0x50,0xAA,0x3D,
294                 0x8A,0x1F,0xBF,0xF0,0xEB,0x19,0xCC,0xB1,0xA3,0x13,0xD5,0x5C,
295                 0xDA,0x56,0xC9,0xEC,0x2E,0xF2,0x96,0x32,0x38,0x7F,0xE8,0xD7,
296                 0x6E,0x3C,0x04,0x68,0x04,0x3E,0x8F,0x66,0x3F,0x48,0x60,0xEE,
297                 0x12,0xBF,0x2D,0x5B,0x0B,0x74,0x74,0xD6,0xE6,0x94,0xF9,0x1E,
298                 0x6D,0xCC,0x40,0x24,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
299                 };
300         return BN_bin2bn(RFC3526_PRIME_6144,sizeof(RFC3526_PRIME_6144),bn);
301         }

303 /* "8192-bit MODP Group" from RFC3526, Section 7.
304  *
305  * The prime is: 2^8192 - 2^8128 - 1 + 2^64 * { [2^8062 pi] + 4743158 }
306  *
307  * RFC3526 specifies a generator of 2.
308  */

310 BIGNUM *get_rfc3526_prime_8192(BIGNUM *bn)
311         {
312         static const unsigned char RFC3526_PRIME_8192[]={
313                 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
314                 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
315                 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
316                 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
317                 0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
318                 0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
319                 0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
320                 0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
321                 0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
322                 0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
323                 0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
324                 0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
```



```
325                 0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
326                 0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
327                 0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
328                 0xCA,0x18,0x21,0x7C,0x32,0x90,0x5E,0x46,0x2E,0x36,0xCE,0x3B,
329                 0xE3,0x9E,0x77,0x2C,0x18,0x0E,0x86,0x03,0x9B,0x27,0x83,0xA2,
330                 0xEC,0x07,0xA2,0x8F,0xB5,0xC5,0x5D,0xF0,0x6F,0x4C,0x52,0xC9,
331                 0xDE,0x2B,0xCB,0xF6,0x95,0x58,0x17,0x18,0x39,0x95,0x49,0x7C,
332                 0xEA,0x95,0x6A,0xE5,0x15,0xD2,0x26,0x18,0x98,0xFA,0x05,0x10,
333                 0x15,0x72,0x8E,0x5A,0x8A,0xAA,0xC4,0x2D,0xAD,0x33,0x17,0x0D,
334                 0x04,0x50,0x7A,0x33,0xA8,0x55,0x21,0xAB,0xDF,0x1C,0xBA,0x64,
335                 0xEC,0xFB,0x85,0x04,0x58,0xDB,0xEF,0x0A,0x8A,0xEA,0x71,0x57,
336                 0x5D,0x06,0x0C,0x7D,0xB3,0x97,0x0F,0x85,0xA6,0xE1,0xE4,0xC7,
337                 0xAB,0xF5,0xAE,0x8C,0xDB,0x09,0x33,0xD7,0x1E,0x8C,0x94,0xE0,
338                 0x4A,0x25,0x61,0x9D,0xCE,0xE3,0xD2,0x26,0x1A,0xD2,0xEE,0x6B,
339                 0xF1,0x2F,0xFA,0x06,0xD9,0x8A,0x08,0x64,0xD8,0x76,0x02,0x73,
340                 0x3E,0xC8,0x6A,0x64,0x52,0x1F,0x2B,0x18,0x17,0x7B,0x20,0x0C,
341                 0xBB,0xE1,0x17,0x57,0x7A,0x61,0x5D,0x6C,0x77,0x09,0x88,0xC0,
342                 0xBA,0xD9,0x46,0xE2,0x08,0xE2,0x4F,0xA0,0x74,0xE5,0xAB,0x31,
343                 0x43,0xDB,0x5B,0xFC,0xE0,0xFD,0x10,0x8E,0x4B,0x82,0xD1,0x20,
344                 0xA9,0x21,0x08,0x01,0x1A,0x72,0x3C,0x12,0xA7,0x87,0xE6,0xD7,
345                 0x88,0x71,0x9A,0x10,0xBD,0xBA,0x5B,0x26,0x99,0xC3,0x27,0x18,
346                 0x6A,0xF4,0xE2,0x3C,0x1A,0x94,0x68,0x34,0xB6,0x15,0x0B,0xDA,
347                 0x25,0x83,0xE9,0xCA,0x2A,0xD4,0x4C,0xE8,0xDB,0xBB,0xC2,0xDB,
348                 0x04,0xDE,0x8E,0xF9,0x2E,0x8E,0xFC,0x14,0x1F,0xBE,0xCA,0xA6,
349                 0x28,0x7C,0x59,0x47,0x4E,0x6B,0xC0,0x5D,0x99,0xB2,0x96,0x4F,
350                 0xA0,0x90,0xC3,0xA2,0x23,0x3B,0xA1,0x86,0x51,0x5B,0xE7,0xED,
351                 0x1F,0x61,0x29,0x70,0xCE,0xE2,0xD7,0xAF,0xB8,0x1B,0xDD,0x76,
352                 0x21,0x70,0x48,0x1C,0xD0,0x06,0x91,0x27,0xD5,0xB0,0x5A,0xA9,
353                 0x93,0xB4,0xEA,0x98,0x8D,0x8F,0xDD,0xC1,0x86,0xFF,0xB7,0xDC,
354                 0x90,0xA6,0xC0,0x8F,0x4D,0xF4,0x35,0xC9,0x34,0x02,0x84,0x92,
355                 0x36,0xC3,0xFA,0xB4,0xD2,0x7C,0x70,0x26,0xC1,0xD4,0xDC,0xB2,
356                 0x60,0x26,0x46,0xDE,0xC9,0x75,0x1E,0x76,0x3D,0xBA,0x37,0xBD,
357                 0xF8,0xFF,0x94,0x06,0xAD,0x9E,0x53,0x0E,0xE5,0xDB,0x38,0x2F,
358                 0x41,0x30,0x01,0xAE,0xB0,0x6A,0x53,0xED,0x90,0x27,0xD8,0x31,
359                 0x17,0x97,0x27,0xB0,0x86,0x5A,0x89,0x18,0xDA,0x3E,0xDB,0xEB,
360                 0xCF,0x9B,0x14,0xED,0x44,0xCE,0x6C,0xBA,0xCE,0xD4,0xBB,0x1B,
361                 0xDB,0x7F,0x14,0x47,0xE6,0xCC,0x25,0x4B,0x33,0x20,0x51,0x51,
362                 0x2B,0xD7,0xAF,0x42,0x6F,0xB8,0xF4,0x01,0x37,0x8C,0xD2,0xBF,
363                 0x59,0x83,0xCA,0x01,0xC6,0x4B,0x92,0xEC,0xF0,0x32,0xEA,0x15,
364                 0xD1,0x72,0x1D,0x03,0xF4,0x82,0xD7,0xCE,0x6E,0x74,0xFE,0xF6,
365                 0xD5,0x5E,0x70,0x2F,0x46,0x98,0x0C,0x82,0xB5,0xA8,0x40,0x31,
366                 0x90,0x0B,0x1C,0x9E,0x59,0xE7,0xC9,0x7F,0xBE,0xC7,0xE8,0xF3,
367                 0x23,0xA9,0x7A,0x7E,0x36,0xCC,0x88,0xBE,0x0F,0x1D,0x45,0xB7,
368                 0xFF,0x58,0x5A,0xC5,0x4B,0xD4,0x07,0xB2,0x2B,0x41,0x54,0xAA,
369                 0xCC,0x8F,0x6D,0x7E,0xBF,0x48,0xE1,0xD8,0x14,0xCC,0x5E,0xD2,
370                 0x0F,0x80,0x37,0xE0,0xA7,0x97,0x15,0xEE,0xF2,0x9B,0xE3,0x28,
371                 0x06,0xA1,0xD5,0x8B,0xB7,0xC5,0xDA,0x76,0xF5,0x50,0xAA,0x3D,
372                 0x8A,0x1F,0xBF,0xF0,0xEB,0x19,0xCC,0xB1,0xA3,0x13,0xD5,0x5C,
373                 0xDA,0x56,0xC9,0xEC,0x2E,0xF2,0x96,0x32,0x38,0x7F,0xE8,0xD7,
374                 0x6E,0x3C,0x04,0x68,0x04,0x3E,0x8F,0x66,0x3F,0x48,0x60,0xEE,
375                 0x12,0xBF,0x2D,0x5B,0x0B,0x74,0x74,0xD6,0xE6,0x94,0xF9,0x1E,
376                 0x6D,0xBE,0x11,0x59,0x74,0xA3,0x92,0x6F,0x12,0xFE,0xE5,0xE4,
377                 0x38,0x77,0x7C,0xB6,0xA9,0x32,0xDF,0x8C,0xD8,0xBE,0xC4,0xD0,
378                 0x73,0xB9,0x31,0xBA,0x3B,0xC8,0x32,0xB6,0x8D,0x9D,0xD3,0x00,
379                 0x74,0x1F,0xA7,0xBF,0x8A,0xFC,0x47,0xED,0x25,0x76,0xF6,0x93,
380                 0x6B,0xA4,0x24,0x66,0x3A,0xAB,0x63,0x9C,0x5A,0xE4,0xF5,0x68,
381                 0x34,0x23,0xB4,0x74,0x2B,0xF1,0xC9,0x78,0x23,0x8F,0x16,0xCB,
382                 0xE3,0x9D,0x65,0x2D,0xE3,0xFD,0xB8,0xBE,0xFC,0x84,0x8A,0xD9,
383                 0x22,0x22,0x2E,0x04,0xA4,0x03,0x7C,0x07,0x13,0xEB,0x57,0xA8,
384                 0x1A,0x23,0xF0,0xC7,0x34,0x73,0xFC,0x64,0x6C,0xEA,0x30,0x6B,
385                 0x4B,0xCB,0xC8,0x86,0x2F,0x83,0x85,0xDD,0xFA,0x9D,0x4B,0x7F,
386                 0xA2,0xC0,0x87,0xE8,0x79,0x68,0x33,0x03,0xED,0x5B,0xDD,0x3A,
387                 0x06,0x2B,0x3C,0xF5,0xB3,0xA2,0x78,0xA6,0x6D,0x2A,0x13,0xF8,
388                 0x3F,0x44,0xF8,0x2D,0xDF,0x31,0x0E,0xE0,0x74,0xAB,0x6A,0x36,
389                 0x45,0x97,0xE8,0x99,0xA0,0x25,0x5D,0xC1,0x64,0xF3,0x1C,0xC5,
390                 0x08,0x46,0x85,0x1D,0xF9,0xAB,0x48,0x19,0x5D,0xED,0x7E,0xA1,
```

```
391                 0xB1,0xD5,0x10,0xBD,0x7E,0xE7,0x4D,0x73,0xFA,0xF3,0x6B,0xC3,
392                 0x1E,0xCF,0xA2,0x68,0x35,0x90,0x46,0xF4,0xEB,0x87,0x9F,0x92,
393                 0x40,0x09,0x43,0x8B,0x48,0x1C,0x6C,0xD7,0x88,0x9A,0x00,0x2E,
394                 0xD5,0xEE,0x38,0x2B,0xC9,0x19,0x0D,0xA6,0xFC,0x02,0x6E,0x47,
395                 0x95,0x58,0xE4,0x47,0x56,0x77,0xE9,0xAA,0x9E,0x30,0x50,0xE2,
396                 0x76,0x56,0x94,0xDF,0xC8,0x1F,0x56,0xE8,0x80,0xB9,0x6E,0x71,
397                 0x60,0xC9,0x80,0xDD,0x98,0xED,0xD3,0xDF,0xFF,0xFF,0xFF,0xFF,
398                 0xFF,0xFF,0xFF,0xFF,
399                 };
400         return BN_bin2bn(RFC3526_PRIME_8192,sizeof(RFC3526_PRIME_8192),bn);
401         }
```

```
**********************************************************
   12601 Wed Aug 13 19:32:54 2014
new/usr/src/lib/openssl/libsunw_crypto/bn/bn_div.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/bn/bn_div.c */
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */

#include <stdio.h>
#include <openssl/bn.h>
#include <cryptlib.h>
```



```
#include <bn_lcl.h>
#include "cryptlib.h"
#include "bn_lcl.h"


/* The old slow way */
#if 0
int BN_div(BIGNUM *dv, BIGNUM *rem, const BIGNUM *m, const BIGNUM *d,
           BN_CTX *ctx)
        {
        int i,nm,nd;
        int ret = 0;
        BIGNUM *D;

        bn_check_top(m);
        bn_check_top(d);
        if (BN_is_zero(d))
                {
                BNerr(BN_F_BN_DIV,BN_R_DIV_BY_ZERO);
                return(0);
                }

        if (BN_ucmp(m,d) < 0)
                {
                if (rem != NULL)
                        { if (BN_copy(rem,m) == NULL) return(0); }
                if (dv != NULL) BN_zero(dv);
                return(1);
                }

        BN_CTX_start(ctx);
        D = BN_CTX_get(ctx);
        if (dv == NULL) dv = BN_CTX_get(ctx);
        if (rem == NULL) rem = BN_CTX_get(ctx);
        if (D == NULL || dv == NULL || rem == NULL)
                goto end;

        nd=BN_num_bits(d);
        nm=BN_num_bits(m);
        if (BN_copy(D,d) == NULL) goto end;
        if (BN_copy(rem,m) == NULL) goto end;

        /* The next 2 are needed so we can do a dv->d[0]|=1 later
         * since BN_lshift1 will only work once there is a value :-) */
        BN_zero(dv);
        if(bn_wexpand(dv,1) == NULL) goto end;
        dv->top=1;

        if (!BN_lshift(D,D,nm-nd)) goto end;
        for (i=nm-nd; i>=0; i--)
                {
                if (!BN_lshift1(dv,dv)) goto end;
                if (BN_ucmp(rem,D) >= 0)
                        {
                        dv->d[0]|=1;
                        if (!BN_usub(rem,rem,D)) goto end;
                        }
/* CAN IMPROVE (and have now :=) */
                if (!BN_rshift1(D,D)) goto end;
                }
        rem->neg=BN_is_zero(rem)?0:m->neg;
        dv->neg=m->neg^d->neg;
        ret = 1;
 end:
        BN_CTX_end(ctx);
        return(ret);
```

```
126         }

128 #else

130 #if !defined(OPENSSL_NO_ASM) && !defined(OPENSSL_NO_INLINE_ASM) \
131     && !defined(PEDANTIC) && !defined(BN_DIV3W)
132 # if defined(__GNUC__) && __GNUC__>=2
133 #  if defined(__i386) || defined (__i386__)
134    /*
135     * There were two reasons for implementing this template:
136     * - GNU C generates a call to a function (__udivdi3 to be exact)
137     *   in reply to ((((BN_ULLONG)n0)<<BN_BITS2)|n1)/d0 (I fail to
138     *   understand why...);
139     * - divl doesn't only calculate quotient, but also leaves
140     *   remainder in %edx which we can definitely use here:-)
141     *
142     *                                      <appro@fy.chalmers.se>
143     */
144 #undef bn_div_words
145 #  define bn_div_words(n0,n1,d0)                \
146         ({  __asm volatile (                    \
146         ({  asm volatile (                      \
147                 "divl   %4"                     \
148                 : "=a"(q), "=d"(rem)            \
149                 : "a"(n1), "d"(n0), "g"(d0)     \
150                 : "cc");                        \
151             q;                                  \
152         })
153 #  define REMAINDER_IS_ALREADY_CALCULATED
154 #  elif defined(__x86_64) && defined(SIXTY_FOUR_BIT_LONG)
155    /*
156     * Same story here, but it's 128-bit by 64-bit division. Wow!
157     *                                      <appro@fy.chalmers.se>
158     */
159 #  undef bn_div_words
160 #  define bn_div_words(n0,n1,d0)                \
161         ({  __asm volatile (                    \
161         ({  asm volatile (                      \
162                 "divq   %4"                     \
163                 : "=a"(q), "=d"(rem)            \
164                 : "a"(n1), "d"(n0), "g"(d0)     \
165                 : "cc");                        \
166             q;                                  \
167         })
168 #  define REMAINDER_IS_ALREADY_CALCULATED
169 #  endif /* __<cpu> */
170 # endif /* __GNUC__ */
171 #endif /* OPENSSL_NO_ASM */


174 /* BN_div computes  dv := num / divisor,  rounding towards
175  * zero, and sets up rm  such that  dv*divisor + rm = num  holds.
176  * Thus:
177  *     dv->neg == num->neg ^ divisor->neg  (unless the result is zero)
178  *     rm->neg == num->neg                 (unless the remainder is zero)
179  * If 'dv' or 'rm' is NULL, the respective value is not returned.
180  */
181 int BN_div(BIGNUM *dv, BIGNUM *rm, const BIGNUM *num, const BIGNUM *divisor,
182            BN_CTX *ctx)
183         {
184         int norm_shift,i,loop;
185         BIGNUM *tmp,wnum,*snum,*sdiv,*res;
186         BN_ULONG *resp,*wnump;
187         BN_ULONG d0,d1;
188         int num_n,div_n;
189         int no_branch=0;
```



```
191         /* Invalid zero-padding would have particularly bad consequences
192          * in the case of 'num', so don't just rely on bn_check_top() for this o
193          * (bn_check_top() works only for BN_DEBUG builds) */
194         if (num->top > 0 && num->d[num->top - 1] == 0)
195                 {
196                 BNerr(BN_F_BN_DIV,BN_R_NOT_INITIALIZED);
197                 return 0;
198                 }

200         bn_check_top(num);

202         if ((BN_get_flags(num, BN_FLG_CONSTTIME) != 0) || (BN_get_flags(divisor,
203                 {
204                 no_branch=1;
205                 }

207         bn_check_top(dv);
208         bn_check_top(rm);
209         /* bn_check_top(num); */ /* 'num' has been checked already */
210         bn_check_top(divisor);

212         if (BN_is_zero(divisor))
213                 {
214                 BNerr(BN_F_BN_DIV,BN_R_DIV_BY_ZERO);
215                 return(0);
216                 }

218         if (!no_branch && BN_ucmp(num,divisor) < 0)
219                 {
220                 if (rm != NULL)
221                         { if (BN_copy(rm,num) == NULL) return(0); }
222                 if (dv != NULL) BN_zero(dv);
223                 return(1);
224                 }

226         BN_CTX_start(ctx);
227         tmp=BN_CTX_get(ctx);
228         snum=BN_CTX_get(ctx);
229         sdiv=BN_CTX_get(ctx);
230         if (dv == NULL)
231                 res=BN_CTX_get(ctx);
232         else    res=dv;
233         if (sdiv == NULL || res == NULL || tmp == NULL || snum == NULL)
234                 goto err;

236         /* First we normalise the numbers */
237         norm_shift=BN_BITS2-((BN_num_bits(divisor))%BN_BITS2);
238         if (!(BN_lshift(sdiv,divisor,norm_shift))) goto err;
239         sdiv->neg=0;
240         norm_shift+=BN_BITS2;
241         if (!(BN_lshift(snum,num,norm_shift))) goto err;
242         snum->neg=0;

244         if (no_branch)
245                 {
246                 /* Since we don't know whether snum is larger than sdiv,
247                  * we pad snum with enough zeroes without changing its
248                  * value.
249                  */
250                 if (snum->top <= sdiv->top+1)
251                         {
252                         if (bn_wexpand(snum, sdiv->top + 2) == NULL) goto err;
253                         for (i = snum->top; i < sdiv->top + 2; i++) snum->d[i] =
254                         snum->top = sdiv->top + 2;
255                         }
```

```
256                 else
257                         {
258                         if (bn_wexpand(snum, snum->top + 1) == NULL) goto err;
259                         snum->d[snum->top] = 0;
260                         snum->top ++;
261                         }
262                 }
263
264         div_n=sdiv->top;
265         num_n=snum->top;
266         loop=num_n-div_n;
267         /* Lets setup a 'window' into snum
268          * This is the part that corresponds to the current
269          * 'area' being divided */
270         wnum.neg   = 0;
271         wnum.d     = &(snum->d[loop]);
272         wnum.top   = div_n;
273         /* only needed when BN_ucmp messes up the values between top and max */
274         wnum.dmax  = snum->dmax - loop; /* so we don't step out of bounds */
275
276         /* Get the top 2 words of sdiv */
277         /* div_n=sdiv->top; */
278         d0=sdiv->d[div_n-1];
279         d1=(div_n == 1)?0:sdiv->d[div_n-2];
280
281         /* pointer to the 'top' of snum */
282         wnump= &(snum->d[num_n-1]);
283
284         /* Setup to 'res' */
285         res->neg= (num->neg^divisor->neg);
286         if (!bn_wexpand(res,(loop+1))) goto err;
287         res->top=loop-no_branch;
288         resp= &(res->d[loop-1]);
289
290         /* space for temp */
291         if (!bn_wexpand(tmp,(div_n+1))) goto err;
292
293         if (!no_branch)
294                 {
295                 if (BN_ucmp(&wnum,sdiv) >= 0)
296                         {
297                         /* If BN_DEBUG_RAND is defined BN_ucmp changes (via
298                          * bn_pollute) the const bignum arguments =>
299                          * clean the values between top and max again */
300                         bn_clear_top2max(&wnum);
301                         bn_sub_words(wnum.d, wnum.d, sdiv->d, div_n);
302                         *resp=1;
303                         }
304                 else
305                         res->top--;
306                 }
307
308         /* if res->top == 0 then clear the neg value otherwise decrease
309          * the resp pointer */
310         if (res->top == 0)
311                 res->neg = 0;
312         else
313                 resp--;
314
315         for (i=0; i<loop-1; i++, wnump--, resp--)
316                 {
317                 BN_ULONG q,l0;
318                 /* the first part of the loop uses the top two words of
319                  * snum and sdiv to calculate a BN_ULONG q such that
320                  * | wnum - sdiv * q | < sdiv */
321 #if defined(BN_DIV3W) && !defined(OPENSSL_NO_ASM)
```



```
322                 BN_ULONG bn_div_3_words(BN_ULONG*,BN_ULONG,BN_ULONG);
323                 q=bn_div_3_words(wnump,d1,d0);
324 #else
325                 BN_ULONG n0,n1,rem=0;
326
327                 n0=wnump[0];
328                 n1=wnump[-1];
329                 if (n0 == d0)
330                         q=BN_MASK2;
331                 else                    /* n0 < d0 */
332                         {
333 #ifdef BN_LLONG
334                         BN_ULLONG t2;
335
336 #if defined(BN_LLONG) && defined(BN_DIV2W) && !defined(bn_div_words)
337                         q=(BN_ULONG)(((((BN_ULLONG)n0)<<BN_BITS2)|n1)/d0);
338 #else
339                         q=bn_div_words(n0,n1,d0);
340 #ifdef BN_DEBUG_LEVITTE
341                         fprintf(stderr,"DEBUG: bn_div_words(0x%08X,0x%08X,0x%08\
342 X) -> 0x%08X\n",
343                                 n0, n1, d0, q);
344 #endif
345 #endif
346
347 #ifndef REMAINDER_IS_ALREADY_CALCULATED
348                         /*
349                          * rem doesn't have to be BN_ULLONG. The least we
350                          * know it's less that d0, isn't it?
351                          */
352                         rem=(n1-q*d0)&BN_MASK2;
353 #endif
354                         t2=(BN_ULLONG)d1*q;
355
356                         for (;;)
357                                 {
358                                 if (t2 <= ((((BN_ULLONG)rem)<<BN_BITS2)|wnump[-2
359                                         break;
360                                 q--;
361                                 rem += d0;
362                                 if (rem < d0) break; /* don't let rem overflow *
363                                 t2 -= d1;
364                                 }
365 #else /* !BN_LLONG */
366                         BN_ULONG t2l,t2h;
367
368                         q=bn_div_words(n0,n1,d0);
369 #ifdef BN_DEBUG_LEVITTE
370                         fprintf(stderr,"DEBUG: bn_div_words(0x%08X,0x%08X,0x%08\
371 X) -> 0x%08X\n",
372                                 n0, n1, d0, q);
373 #endif
374 #ifndef REMAINDER_IS_ALREADY_CALCULATED
375                         rem=(n1-q*d0)&BN_MASK2;
376 #endif
377
378 #if defined(BN_UMULT_LOHI)
379                         BN_UMULT_LOHI(t2l,t2h,d1,q);
380 #elif defined(BN_UMULT_HIGH)
381                         t2l = d1 * q;
382                         t2h = BN_UMULT_HIGH(d1,q);
383 #else
384                         {
385                         BN_ULONG ql, qh;
386                         t2l=LBITS(d1); t2h=HBITS(d1);
387                         ql =LBITS(q);  qh =HBITS(q);
```

```
388                         mul64(t2l,t2h,ql,qh); /* t2=(BN_ULLONG)d1*q; */
389                         }
390 #endif

392                         for (;;)
393                                 {
394                                 if ((t2h < rem) ||
395                                         ((t2h == rem) && (t2l <= wnump[-2])))
396                                         break;
397                                 q--;
398                                 rem += d0;
399                                 if (rem < d0) break; /* don't let rem overflow *
400                                 if (t2l < d1) t2h--; t2l -= d1;
401                                 }
402 #endif /* !BN_LLONG */
403                         }
404 #endif /* !BN_DIV3W */

406                 l0=bn_mul_words(tmp->d,sdiv->d,div_n,q);
407                 tmp->d[div_n]=l0;
408                 wnum.d--;
409                 /* ingore top values of the bignums just sub the two
410                  * BN_ULONG arrays with bn_sub_words */
411                 if (bn_sub_words(wnum.d, wnum.d, tmp->d, div_n+1))
412                         {
413                         /* Note: As we have considered only the leading
414                          * two BN_ULONGs in the calculation of q, sdiv * q
415                          * might be greater than wnum (but then (q-1) * sdiv
416                          * is less or equal than wnum)
417                          */
418                         q--;
419                         if (bn_add_words(wnum.d, wnum.d, sdiv->d, div_n))
420                                 /* we can't have an overflow here (assuming
421                                  * that q != 0, but if q == 0 then tmp is
422                                  * zero anyway) */
423                                 (*wnump)++;
424                         }
425                 /* store part of the result */
426                 *resp = q;
427                 }
428         bn_correct_top(snum);
429         if (rm != NULL)
430                 {
431                 /* Keep a copy of the neg flag in num because if rm==num
432                  * BN_rshift() will overwrite it.
433                  */
434                 int neg = num->neg;
435                 BN_rshift(rm,snum,norm_shift);
436                 if (!BN_is_zero(rm))
437                         rm->neg = neg;
438                 bn_check_top(rm);
439                 }
440         if (no_branch)  bn_correct_top(res);
441         BN_CTX_end(ctx);
442         return(1);
443 err:
444         bn_check_top(rm);
445         BN_CTX_end(ctx);
446         return(0);
447         }
448 #endif
```

```
**********************************************************
   13606 Wed Aug 13 19:32:55 2014
new/usr/src/lib/openssl/libsunw_crypto/bn/x86_64-gcc.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 #include <bn_lcl.h>
   1 #include "../bn_lcl.h"
   2 #if !(defined(__GNUC__) && __GNUC__>=2)
   3 # include "bn_asm.c"    /* kind of dirty hack for Sun Studio */
   3 # include "../bn_asm.c" /* kind of dirty hack for Sun Studio */
   4 #else
   5 /*
   6  * x86_64 BIGNUM accelerator version 0.1, December 2002.
   7  *
   8  * Implemented by Andy Polyakov <appro@fy.chalmers.se> for the OpenSSL
   9  * project.
  10  *
  11  * Rights for redistribution and usage in source and binary forms are
  12  * granted according to the OpenSSL license. Warranty of any kind is
  13  * disclaimed.
  14  *
  15  * Q. Version 0.1? It doesn't sound like Andy, he used to assign real
  16  *    versions, like 1.0...
  17  * A. Well, that's because this code is basically a quick-n-dirty
  18  *    proof-of-concept hack. As you can see it's implemented with
  19  *    inline assembler, which means that you're bound to GCC and that
  20  *    there might be enough room for further improvement.
  21  *
  22  * Q. Why inline assembler?
  23  * A. x86_64 features own ABI which I'm not familiar with. This is
  24  *    why I decided to let the compiler take care of subroutine
  25  *    prologue/epilogue as well as register allocation. For reference.
  26  *    Win64 implements different ABI for AMD64, different from Linux.
  27  *
  28  * Q. How much faster does it get?
  29  * A. 'apps/openssl speed rsa dsa' output with no-asm:
  30  *
  31  *                         sign    verify    sign/s verify/s
  32  *      rsa  512 bits   0.0006s   0.0001s   1683.8  18456.2
  33  *      rsa 1024 bits   0.0028s   0.0002s    356.0   6407.0
  34  *      rsa 2048 bits   0.0172s   0.0005s     58.0   1957.8
  35  *      rsa 4096 bits   0.1155s   0.0018s      8.7    555.6
  36  *                         sign    verify    sign/s verify/s
  37  *      dsa  512 bits   0.0005s   0.0006s   2100.8   1768.3
  38  *      dsa 1024 bits   0.0014s   0.0018s    692.3    559.2
  39  *      dsa 2048 bits   0.0049s   0.0061s    204.7    165.0
  40  *
  41  *    'apps/openssl speed rsa dsa' output with this module:
  42  *
  43  *                         sign    verify    sign/s verify/s
  44  *      rsa  512 bits   0.0004s   0.0000s   2767.1  33297.9
  45  *      rsa 1024 bits   0.0012s   0.0001s    867.4  14674.7
  46  *      rsa 2048 bits   0.0061s   0.0002s    164.0   5270.0
  47  *      rsa 4096 bits   0.0384s   0.0006s     26.1   1650.8
  48  *                         sign    verify    sign/s verify/s
  49  *      dsa  512 bits   0.0002s   0.0003s   4442.2   3786.3
  50  *      dsa 1024 bits   0.0005s   0.0007s   1835.1   1497.4
  51  *      dsa 2048 bits   0.0016s   0.0020s    620.4    504.6
  52  *
  53  *    For the reference. IA-32 assembler implementation performs
  54  *    very much like 64-bit code compiled with no-asm on the same
  55  *    machine.
  56  */

  58 #ifdef _WIN64
  59 #define BN_ULONG unsigned long long
```



```
  60 #else
  61 #define BN_ULONG unsigned long
  62 #endif

  64 #undef mul
  65 #undef mul_add
  66 #undef sqr

  68 /*
  69  * "m"(a), "+m"(r)      is the way to favor DirectPath µ-code;
  70  * "g"(0)               let the compiler to decide where does it
  71  *                      want to keep the value of zero;
  72  */
  73 #define mul_add(r,a,word,carry) do {    \
  74         register BN_ULONG high,low;     \
  75         __asm__ ("mulq %3"                      \
  75         asm ("mulq %3"                  \
  76                 : "=a"(low),"=d"(high)  \
  77                 : "a"(word),"m"(a)      \
  78                 : "cc");                \
  79         __asm__ ("addq %2,%0; adcq %3,%1"       \
  79         asm ("addq %2,%0; adcq %3,%1"   \
  80                 : "+r"(carry),"+d"(high)\
  81                 : "a"(low),"g"(0)       \
  82                 : "cc");                \
  83         __asm__ ("addq %2,%0; adcq %3,%1"       \
  83         asm ("addq %2,%0; adcq %3,%1"   \
  84                 : "+m"(r),"+d"(high)    \
  85                 : "r"(carry),"g"(0)     \
  86                 : "cc");                \
  87         carry=high;                     \
  88         } while (0)

  90 #define mul(r,a,word,carry) do {        \
  91         register BN_ULONG high,low;     \
  92         __asm__ ("mulq %3"                      \
  92         asm ("mulq %3"                  \
  93                 : "=a"(low),"=d"(high)  \
  94                 : "a"(word),"g"(a)      \
  95                 : "cc");                \
  96         __asm__ ("addq %2,%0; adcq %3,%1"       \
  96         asm ("addq %2,%0; adcq %3,%1"   \
  97                 : "+r"(carry),"+d"(high)\
  98                 : "a"(low),"g"(0)       \
  99                 : "cc");                \
 100         (r)=carry, carry=high;          \
 101         } while (0)

 103 #define sqr(r0,r1,a)                    \
 104         __asm__ ("mulq %2"                      \
 104         asm ("mulq %2"                  \
 105                 : "=a"(r0),"=d"(r1)     \
 106                 : "a"(a)                \
 107                 : "cc");

 109 BN_ULONG bn_mul_add_words(BN_ULONG *rp, const BN_ULONG *ap, int num, BN_ULONG w)
 110         {
 111         BN_ULONG c1=0;

 113         if (num <= 0) return(c1);

 115         while (num&~3)
 116                 {
 117                 mul_add(rp[0],ap[0],w,c1);
 118                 mul_add(rp[1],ap[1],w,c1);
 119                 mul_add(rp[2],ap[2],w,c1);
```

```
120                 mul_add(rp[3],ap[3],w,c1);
121                 ap+=4; rp+=4; num-=4;
122                 }
123         if (num)
124                 {
125                 mul_add(rp[0],ap[0],w,c1); if (--num==0) return c1;
126                 mul_add(rp[1],ap[1],w,c1); if (--num==0) return c1;
127                 mul_add(rp[2],ap[2],w,c1); return c1;
128                 }

130         return(c1);
131         }

133 BN_ULONG bn_mul_words(BN_ULONG *rp, const BN_ULONG *ap, int num, BN_ULONG w)
134         {
135         BN_ULONG c1=0;

137         if (num <= 0) return(c1);

139         while (num&~3)
140                 {
141                 mul(rp[0],ap[0],w,c1);
142                 mul(rp[1],ap[1],w,c1);
143                 mul(rp[2],ap[2],w,c1);
144                 mul(rp[3],ap[3],w,c1);
145                 ap+=4; rp+=4; num-=4;
146                 }
147         if (num)
148                 {
149                 mul(rp[0],ap[0],w,c1); if (--num == 0) return c1;
150                 mul(rp[1],ap[1],w,c1); if (--num == 0) return c1;
151                 mul(rp[2],ap[2],w,c1);
152                 }
153         return(c1);
154         }

156 void bn_sqr_words(BN_ULONG *r, const BN_ULONG *a, int n)
157         {
158         if (n <= 0) return;

160         while (n&~3)
161                 {
162                 sqr(r[0],r[1],a[0]);
163                 sqr(r[2],r[3],a[1]);
164                 sqr(r[4],r[5],a[2]);
165                 sqr(r[6],r[7],a[3]);
166                 a+=4; r+=8; n-=4;
167                 }
168         if (n)
169                 {
170                 sqr(r[0],r[1],a[0]); if (--n == 0) return;
171                 sqr(r[2],r[3],a[1]); if (--n == 0) return;
172                 sqr(r[4],r[5],a[2]);
173                 }
174         }

176 BN_ULONG bn_div_words(BN_ULONG h, BN_ULONG l, BN_ULONG d)
177 {       BN_ULONG ret,waste;

179         __asm__ ("divq  %4"
179         asm ("divq      %4"
180                 : "=a"(ret),"=d"(waste)
181                 : "a"(l),"d"(h),"g"(d)
182                 : "cc");

184         return ret;
```



```
185 }

187 BN_ULONG bn_add_words (BN_ULONG *rp, const BN_ULONG *ap, const BN_ULONG *bp,int
188 { BN_ULONG ret=0,i=0;

190         if (n <= 0) return 0;

192         __asm__ (
192         asm (
193         "       subq    %2,%2           \n"
194         ".p2align 4                     \n"
195         "1:     movq    (%4,%2,8),%0    \n"
196         "       adcq    (%5,%2,8),%0    \n"
197         "       movq    %0,(%3,%2,8)    \n"
198         "       leaq    1(%2),%2        \n"
199         "       loop    1b              \n"
200         "       sbbq    %0,%0           \n"
201                 : "=&a"(ret),"+c"(n),"=&r"(i)
202                 : "r"(rp),"r"(ap),"r"(bp)
203                 : "cc"
204         );

206   return ret&1;
207 }

209 #ifndef SIMICS
210 BN_ULONG bn_sub_words (BN_ULONG *rp, const BN_ULONG *ap, const BN_ULONG *bp,int
211 { BN_ULONG ret=0,i=0;

213         if (n <= 0) return 0;

215         __asm__ (
215         asm (
216         "       subq    %2,%2           \n"
217         ".p2align 4                     \n"
218         "1:     movq    (%4,%2,8),%0    \n"
219         "       sbbq    (%5,%2,8),%0    \n"
220         "       movq    %0,(%3,%2,8)    \n"
221         "       leaq    1(%2),%2        \n"
222         "       loop    1b              \n"
223         "       sbbq    %0,%0           \n"
224                 : "=&a"(ret),"+c"(n),"=&r"(i)
225                 : "r"(rp),"r"(ap),"r"(bp)
226                 : "cc"
227         );

229   return ret&1;
230 }
231 #else
232 /* Simics 1.4<7 has buggy sbbq:-( */
233 #define BN_MASK2 0xffffffffffffffffL
234 BN_ULONG bn_sub_words(BN_ULONG *r, BN_ULONG *a, BN_ULONG *b, int n)
235         {
236         BN_ULONG t1,t2;
237         int c=0;

239         if (n <= 0) return((BN_ULONG)0);

241         for (;;)
242                 {
243                 t1=a[0]; t2=b[0];
244                 r[0]=(t1-t2-c)&BN_MASK2;
245                 if (t1 != t2) c=(t1 < t2);
246                 if (--n <= 0) break;

248                 t1=a[1]; t2=b[1];
```

```
249                 r[1]=(t1-t2-c)&BN_MASK2;
250                 if (t1 != t2) c=(t1 < t2);
251                 if (--n <= 0) break;

253                 t1=a[2]; t2=b[2];
254                 r[2]=(t1-t2-c)&BN_MASK2;
255                 if (t1 != t2) c=(t1 < t2);
256                 if (--n <= 0) break;

258                 t1=a[3]; t2=b[3];
259                 r[3]=(t1-t2-c)&BN_MASK2;
260                 if (t1 != t2) c=(t1 < t2);
261                 if (--n <= 0) break;

263                 a+=4;
264                 b+=4;
265                 r+=4;
266                 }
267         return(c);
268         }
269 #endif

271 /* mul_add_c(a,b,c0,c1,c2)  -- c+=a*b for three word number c=(c2,c1,c0) */
272 /* mul_add_c2(a,b,c0,c1,c2) -- c+=2*a*b for three word number c=(c2,c1,c0) */
273 /* sqr_add_c(a,i,c0,c1,c2)  -- c+=a[i]^2 for three word number c=(c2,c1,c0) */
274 /* sqr_add_c2(a,i,c0,c1,c2) -- c+=2*a[i]*a[j] for three word number c=(c2,c1,c0)

276 #if 0
277 /* original macros are kept for reference purposes */
278 #define mul_add_c(a,b,c0,c1,c2) {       \
279         BN_ULONG ta=(a),tb=(b);         \
280         t1 = ta * tb;                   \
281         t2 = BN_UMULT_HIGH(ta,tb);      \
282         c0 += t1; t2 += (c0<t1)?1:0;    \
283         c1 += t2; c2 += (c1<t2)?1:0;    \
284         }

286 #define mul_add_c2(a,b,c0,c1,c2) {      \
287         BN_ULONG ta=(a),tb=(b),t0;      \
288         t1 = BN_UMULT_HIGH(ta,tb);      \
289         t0 = ta * tb;                   \
290         t2 = t1+t1; c2 += (t2<t1)?1:0;  \
291         t1 = t0+t0; t2 += (t1<t0)?1:0;  \
292         c0 += t1; t2 += (c0<t1)?1:0;    \
293         c1 += t2; c2 += (c1<t2)?1:0;    \
294         }
295 #else
296 #define mul_add_c(a,b,c0,c1,c2) do {    \
297         __asm__ ("mulq %3"                      \
297         asm ("mulq %3"                  \
298                 : "=a"(t1),"=d"(t2)     \
299                 : "a"(a),"m"(b)         \
300                 : "cc");                \
301         __asm__ ("addq %2,%0; adcq %3,%1"       \
301         asm ("addq %2,%0; adcq %3,%1"   \
302                 : "+r"(c0),"+d"(t2)     \
303                 : "a"(t1),"g"(0)        \
304                 : "cc");                \
305         __asm__ ("addq %2,%0; adcq %3,%1"       \
305         asm ("addq %2,%0; adcq %3,%1"   \
306                 : "+r"(c1),"+r"(c2)     \
307                 : "d"(t2),"g"(0)        \
308                 : "cc");                \
309         } while (0)

311 #define sqr_add_c(a,i,c0,c1,c2) do {    \
```



```
312         __asm__ ("mulq %2"                      \
312         asm ("mulq %2"                  \
313                 : "=a"(t1),"=d"(t2)     \
314                 : "a"(a[i])             \
315                 : "cc");                \
316         __asm__ ("addq %2,%0; adcq %3,%1"       \
316         asm ("addq %2,%0; adcq %3,%1"   \
317                 : "+r"(c0),"+d"(t2)     \
318                 : "a"(t1),"g"(0)        \
319                 : "cc");                \
320         __asm__ ("addq %2,%0; adcq %3,%1"       \
320         asm ("addq %2,%0; adcq %3,%1"   \
321                 : "+r"(c1),"+r"(c2)     \
322                 : "d"(t2),"g"(0)        \
323                 : "cc");                \
324         } while (0)

326 #define mul_add_c2(a,b,c0,c1,c2) do {   \
327         __asm__ ("mulq %3"                      \
327         asm ("mulq %3"                  \
328                 : "=a"(t1),"=d"(t2)     \
329                 : "a"(a),"m"(b)         \
330                 : "cc");                \
331         __asm__ ("addq %0,%0; adcq %2,%1"       \
331         asm ("addq %0,%0; adcq %2,%1"   \
332                 : "+d"(t2),"+r"(c2)     \
333                 : "g"(0)                \
334                 : "cc");                \
335         __asm__ ("addq %0,%0; adcq %2,%1"       \
335         asm ("addq %0,%0; adcq %2,%1"   \
336                 : "+a"(t1),"+d"(t2)     \
337                 : "g"(0)                \
338                 : "cc");                \
339         __asm__ ("addq %2,%0; adcq %3,%1"       \
339         asm ("addq %2,%0; adcq %3,%1"   \
340                 : "+r"(c0),"+d"(t2)     \
341                 : "a"(t1),"g"(0)        \
342                 : "cc");                \
343         __asm__ ("addq %2,%0; adcq %3,%1"       \
343         asm ("addq %2,%0; adcq %3,%1"   \
344                 : "+r"(c1),"+r"(c2)     \
345                 : "d"(t2),"g"(0)        \
346                 : "cc");                \
347         } while (0)
348 #endif

350 #define sqr_add_c2(a,i,j,c0,c1,c2)      \
351         mul_add_c2((a)[i],(a)[j],c0,c1,c2)

353 void bn_mul_comba8(BN_ULONG *r, BN_ULONG *a, BN_ULONG *b)
354         {
355         BN_ULONG t1,t2;
356         BN_ULONG c1,c2,c3;

358         c1=0;
359         c2=0;
360         c3=0;
361         mul_add_c(a[0],b[0],c1,c2,c3);
362         r[0]=c1;
363         c1=0;
364         mul_add_c(a[0],b[1],c2,c3,c1);
365         mul_add_c(a[1],b[0],c2,c3,c1);
366         r[1]=c2;
367         c2=0;
368         mul_add_c(a[2],b[0],c3,c1,c2);
369         mul_add_c(a[1],b[1],c3,c1,c2);
```

```
370         mul_add_c(a[0],b[2],c3,c1,c2);
371         r[2]=c3;
372         c3=0;
373         mul_add_c(a[0],b[3],c1,c2,c3);
374         mul_add_c(a[1],b[2],c1,c2,c3);
375         mul_add_c(a[2],b[1],c1,c2,c3);
376         mul_add_c(a[3],b[0],c1,c2,c3);
377         r[3]=c1;
378         c1=0;
379         mul_add_c(a[4],b[0],c2,c3,c1);
380         mul_add_c(a[3],b[1],c2,c3,c1);
381         mul_add_c(a[2],b[2],c2,c3,c1);
382         mul_add_c(a[1],b[3],c2,c3,c1);
383         mul_add_c(a[0],b[4],c2,c3,c1);
384         r[4]=c2;
385         c2=0;
386         mul_add_c(a[0],b[5],c3,c1,c2);
387         mul_add_c(a[1],b[4],c3,c1,c2);
388         mul_add_c(a[2],b[3],c3,c1,c2);
389         mul_add_c(a[3],b[2],c3,c1,c2);
390         mul_add_c(a[4],b[1],c3,c1,c2);
391         mul_add_c(a[5],b[0],c3,c1,c2);
392         r[5]=c3;
393         c3=0;
394         mul_add_c(a[6],b[0],c1,c2,c3);
395         mul_add_c(a[5],b[1],c1,c2,c3);
396         mul_add_c(a[4],b[2],c1,c2,c3);
397         mul_add_c(a[3],b[3],c1,c2,c3);
398         mul_add_c(a[2],b[4],c1,c2,c3);
399         mul_add_c(a[1],b[5],c1,c2,c3);
400         mul_add_c(a[0],b[6],c1,c2,c3);
401         r[6]=c1;
402         c1=0;
403         mul_add_c(a[0],b[7],c2,c3,c1);
404         mul_add_c(a[1],b[6],c2,c3,c1);
405         mul_add_c(a[2],b[5],c2,c3,c1);
406         mul_add_c(a[3],b[4],c2,c3,c1);
407         mul_add_c(a[4],b[3],c2,c3,c1);
408         mul_add_c(a[5],b[2],c2,c3,c1);
409         mul_add_c(a[6],b[1],c2,c3,c1);
410         mul_add_c(a[7],b[0],c2,c3,c1);
411         r[7]=c2;
412         c2=0;
413         mul_add_c(a[7],b[1],c3,c1,c2);
414         mul_add_c(a[6],b[2],c3,c1,c2);
415         mul_add_c(a[5],b[3],c3,c1,c2);
416         mul_add_c(a[4],b[4],c3,c1,c2);
417         mul_add_c(a[3],b[5],c3,c1,c2);
418         mul_add_c(a[2],b[6],c3,c1,c2);
419         mul_add_c(a[1],b[7],c3,c1,c2);
420         r[8]=c3;
421         c3=0;
422         mul_add_c(a[2],b[7],c1,c2,c3);
423         mul_add_c(a[3],b[6],c1,c2,c3);
424         mul_add_c(a[4],b[5],c1,c2,c3);
425         mul_add_c(a[5],b[4],c1,c2,c3);
426         mul_add_c(a[6],b[3],c1,c2,c3);
427         mul_add_c(a[7],b[2],c1,c2,c3);
428         r[9]=c1;
429         c1=0;
430         mul_add_c(a[7],b[3],c2,c3,c1);
431         mul_add_c(a[6],b[4],c2,c3,c1);
432         mul_add_c(a[5],b[5],c2,c3,c1);
433         mul_add_c(a[4],b[6],c2,c3,c1);
434         mul_add_c(a[3],b[7],c2,c3,c1);
435         r[10]=c2;
```



```
436         c2=0;
437         mul_add_c(a[4],b[7],c3,c1,c2);
438         mul_add_c(a[5],b[6],c3,c1,c2);
439         mul_add_c(a[6],b[5],c3,c1,c2);
440         mul_add_c(a[7],b[4],c3,c1,c2);
441         r[11]=c3;
442         c3=0;
443         mul_add_c(a[7],b[5],c1,c2,c3);
444         mul_add_c(a[6],b[6],c1,c2,c3);
445         mul_add_c(a[5],b[7],c1,c2,c3);
446         r[12]=c1;
447         c1=0;
448         mul_add_c(a[6],b[7],c2,c3,c1);
449         mul_add_c(a[7],b[6],c2,c3,c1);
450         r[13]=c2;
451         c2=0;
452         mul_add_c(a[7],b[7],c3,c1,c2);
453         r[14]=c3;
454         r[15]=c1;
455         }

457 void bn_mul_comba4(BN_ULONG *r, BN_ULONG *a, BN_ULONG *b)
458         {
459         BN_ULONG t1,t2;
460         BN_ULONG c1,c2,c3;

462         c1=0;
463         c2=0;
464         c3=0;
465         mul_add_c(a[0],b[0],c1,c2,c3);
466         r[0]=c1;
467         c1=0;
468         mul_add_c(a[0],b[1],c2,c3,c1);
469         mul_add_c(a[1],b[0],c2,c3,c1);
470         r[1]=c2;
471         c2=0;
472         mul_add_c(a[2],b[0],c3,c1,c2);
473         mul_add_c(a[1],b[1],c3,c1,c2);
474         mul_add_c(a[0],b[2],c3,c1,c2);
475         r[2]=c3;
476         c3=0;
477         mul_add_c(a[0],b[3],c1,c2,c3);
478         mul_add_c(a[1],b[2],c1,c2,c3);
479         mul_add_c(a[2],b[1],c1,c2,c3);
480         mul_add_c(a[3],b[0],c1,c2,c3);
481         r[3]=c1;
482         c1=0;
483         mul_add_c(a[3],b[1],c2,c3,c1);
484         mul_add_c(a[2],b[2],c2,c3,c1);
485         mul_add_c(a[1],b[3],c2,c3,c1);
486         r[4]=c2;
487         c2=0;
488         mul_add_c(a[2],b[3],c3,c1,c2);
489         mul_add_c(a[3],b[2],c3,c1,c2);
490         r[5]=c3;
491         c3=0;
492         mul_add_c(a[3],b[3],c1,c2,c3);
493         r[6]=c1;
494         r[7]=c2;
495         }

497 void bn_sqr_comba8(BN_ULONG *r, const BN_ULONG *a)
498         {
499         BN_ULONG t1,t2;
500         BN_ULONG c1,c2,c3;
```

```
502         c1=0;
503         c2=0;
504         c3=0;
505         sqr_add_c(a,0,c1,c2,c3);
506         r[0]=c1;
507         c1=0;
508         sqr_add_c2(a,1,0,c2,c3,c1);
509         r[1]=c2;
510         c2=0;
511         sqr_add_c(a,1,c3,c1,c2);
512         sqr_add_c2(a,2,0,c3,c1,c2);
513         r[2]=c3;
514         c3=0;
515         sqr_add_c2(a,3,0,c1,c2,c3);
516         sqr_add_c2(a,2,1,c1,c2,c3);
517         r[3]=c1;
518         c1=0;
519         sqr_add_c(a,2,c2,c3,c1);
520         sqr_add_c2(a,3,1,c2,c3,c1);
521         sqr_add_c2(a,4,0,c2,c3,c1);
522         r[4]=c2;
523         c2=0;
524         sqr_add_c2(a,5,0,c3,c1,c2);
525         sqr_add_c2(a,4,1,c3,c1,c2);
526         sqr_add_c2(a,3,2,c3,c1,c2);
527         r[5]=c3;
528         c3=0;
529         sqr_add_c(a,3,c1,c2,c3);
530         sqr_add_c2(a,4,2,c1,c2,c3);
531         sqr_add_c2(a,5,1,c1,c2,c3);
532         sqr_add_c2(a,6,0,c1,c2,c3);
533         r[6]=c1;
534         c1=0;
535         sqr_add_c2(a,7,0,c2,c3,c1);
536         sqr_add_c2(a,6,1,c2,c3,c1);
537         sqr_add_c2(a,5,2,c2,c3,c1);
538         sqr_add_c2(a,4,3,c2,c3,c1);
539         r[7]=c2;
540         c2=0;
541         sqr_add_c(a,4,c3,c1,c2);
542         sqr_add_c2(a,5,3,c3,c1,c2);
543         sqr_add_c2(a,6,2,c3,c1,c2);
544         sqr_add_c2(a,7,1,c3,c1,c2);
545         r[8]=c3;
546         c3=0;
547         sqr_add_c2(a,7,2,c1,c2,c3);
548         sqr_add_c2(a,6,3,c1,c2,c3);
549         sqr_add_c2(a,5,4,c1,c2,c3);
550         r[9]=c1;
551         c1=0;
552         sqr_add_c(a,5,c2,c3,c1);
553         sqr_add_c2(a,6,4,c2,c3,c1);
554         sqr_add_c2(a,7,3,c2,c3,c1);
555         r[10]=c2;
556         c2=0;
557         sqr_add_c2(a,7,4,c3,c1,c2);
558         sqr_add_c2(a,6,5,c3,c1,c2);
559         r[11]=c3;
560         c3=0;
561         sqr_add_c(a,6,c1,c2,c3);
562         sqr_add_c2(a,7,5,c1,c2,c3);
563         r[12]=c1;
564         c1=0;
565         sqr_add_c2(a,7,6,c2,c3,c1);
566         r[13]=c2;
567         c2=0;
```



```
568         sqr_add_c(a,7,c3,c1,c2);
569         r[14]=c3;
570         r[15]=c1;
571         }

573 void bn_sqr_comba4(BN_ULONG *r, const BN_ULONG *a)
574         {
575         BN_ULONG t1,t2;
576         BN_ULONG c1,c2,c3;

578         c1=0;
579         c2=0;
580         c3=0;
581         sqr_add_c(a,0,c1,c2,c3);
582         r[0]=c1;
583         c1=0;
584         sqr_add_c2(a,1,0,c2,c3,c1);
585         r[1]=c2;
586         c2=0;
587         sqr_add_c(a,1,c3,c1,c2);
588         sqr_add_c2(a,2,0,c3,c1,c2);
589         r[2]=c3;
590         c3=0;
591         sqr_add_c2(a,3,0,c1,c2,c3);
592         sqr_add_c2(a,2,1,c1,c2,c3);
593         r[3]=c1;
594         c1=0;
595         sqr_add_c(a,2,c2,c3,c1);
596         sqr_add_c2(a,3,1,c2,c3,c1);
597         r[4]=c2;
598         c2=0;
599         sqr_add_c2(a,3,2,c3,c1,c2);
600         r[5]=c3;
601         c3=0;
602         sqr_add_c(a,3,c1,c2,c3);
603         r[6]=c1;
604         r[7]=c2;
605         }
606 #endif
```

```
**********************************************************
    6794 Wed Aug 13 19:32:55 2014
new/usr/src/lib/openssl/libsunw_crypto/camellia/cmll_cfb.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/camellia/camellia_cfb.c -*- mode:C; c-file-style: "eay" -*- */
   2 /* ====================================================================
   3  * Copyright (c) 2006 The OpenSSL Project.  All rights reserved.
   4  *
   5  * Redistribution and use in source and binary forms, with or without
   6  * modification, are permitted provided that the following conditions
   7  * are met:
   8  *
   9  * 1. Redistributions of source code must retain the above copyright
  10  *    notice, this list of conditions and the following disclaimer.
  11  *
  12  * 2. Redistributions in binary form must reproduce the above copyright
  13  *    notice, this list of conditions and the following disclaimer in
  14  *    the documentation and/or other materials provided with the
  15  *    distribution.
  16  *
  17  * 3. All advertising materials mentioning features or use of this
  18  *    software must display the following acknowledgment:
  19  *    "This product includes software developed by the OpenSSL Project
  20  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
  21  *
  22  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  23  *    endorse or promote products derived from this software without
  24  *    prior written permission. For written permission, please contact
  25  *    openssl-core@openssl.org.
  26  *
  27  * 5. Products derived from this software may not be called "OpenSSL"
  28  *    nor may "OpenSSL" appear in their names without prior written
  29  *    permission of the OpenSSL Project.
  30  *
  31  * 6. Redistributions of any form whatsoever must retain the following
  32  *    acknowledgment:
  33  *    "This product includes software developed by the OpenSSL Project
  34  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
  35  *
  36  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
  37  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  38  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  39  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  40  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  41  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  42  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  43  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  44  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  45  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  46  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  47  * OF THE POSSIBILITY OF SUCH DAMAGE.
  48  * ====================================================================
  49  *
  50  */
  51 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
  52  * All rights reserved.
  53  *
  54  * This package is an SSL implementation written
  55  * by Eric Young (eay@cryptsoft.com).
  56  * The implementation was written so as to conform with Netscapes SSL.
  57  *
  58  * This library is free for commercial and non-commercial use as long as
  59  * the following conditions are aheared to.  The following conditions
  60  * apply to all code found in this distribution, be it the RC4, RSA,
  61  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
```



```
  62  * included with this distribution is covered by the same copyright terms
  63  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  64  *
  65  * Copyright remains Eric Young's, and as such any Copyright notices in
  66  * the code are not to be removed.
  67  * If this package is used in a product, Eric Young should be given attribution
  68  * as the author of the parts of the library used.
  69  * This can be in the form of a textual message at program startup or
  70  * in documentation (online or textual) provided with the package.
  71  *
  72  * Redistribution and use in source and binary forms, with or without
  73  * modification, are permitted provided that the following conditions
  74  * are met:
  75  * 1. Redistributions of source code must retain the copyright
  76  *    notice, this list of conditions and the following disclaimer.
  77  * 2. Redistributions in binary form must reproduce the above copyright
  78  *    notice, this list of conditions and the following disclaimer in the
  79  *    documentation and/or other materials provided with the distribution.
  80  * 3. All advertising materials mentioning features or use of this software
  81  *    must display the following acknowledgement:
  82  *    "This product includes cryptographic software written by
  83  *     Eric Young (eay@cryptsoft.com)"
  84  *    The word 'cryptographic' can be left out if the rouines from the library
  85  *    being used are not cryptographic related :-).
  86  * 4. If you include any Windows specific code (or a derivative thereof) from
  87  *    the apps directory (application code) you must include an acknowledgement:
  88  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  89  *
  90  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
  91  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  92  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  93  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
  94  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  95  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  96  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  97  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  98  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  99  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 100  * SUCH DAMAGE.
 101  *
 102  * The licence and distribution terms for any publically available version or
 103  * derivative of this code cannot be changed.  i.e. this code cannot simply be
 104  * copied and put under another distribution licence
 105  * [including the GNU Public Licence.]
 106  */

 108 #include <openssl/opensslconf.h>
 109 #endif /* ! codereview */
 110 #include <openssl/camellia.h>
 111 #include <openssl/modes.h>


 114 /* The input and output encrypted as though 128bit cfb mode is being
 115  * used.  The extra state information to record how much of the
 116  * 128bit block we have used is contained in *num;
 117  */

 119 void Camellia_cfb128_encrypt(const unsigned char *in, unsigned char *out,
 120         size_t length, const CAMELLIA_KEY *key,
 121         unsigned char *ivec, int *num, const int enc)
 122         {

 124         CRYPTO_cfb128_encrypt(in,out,length,key,ivec,num,enc,(block128_f)Camelli
 125         }

 127 /* N.B. This expects the input to be packed, MS bit first */
```

```
128 void Camellia_cfb1_encrypt(const unsigned char *in, unsigned char *out,
129         size_t length, const CAMELLIA_KEY *key,
130         unsigned char *ivec, int *num, const int enc)
131         {
132         CRYPTO_cfb128_1_encrypt(in,out,length,key,ivec,num,enc,(block128_f)Camel
133         }

135 void Camellia_cfb8_encrypt(const unsigned char *in, unsigned char *out,
136         size_t length, const CAMELLIA_KEY *key,
137         unsigned char *ivec, int *num, const int enc)
138         {
139         CRYPTO_cfb128_8_encrypt(in,out,length,key,ivec,num,enc,(block128_f)Camel
140         }
```

```
**********************************************************
    5965 Wed Aug 13 19:32:55 2014
new/usr/src/lib/openssl/libsunw_crypto/cast/c_enc.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/cast/c_enc.c */
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */

#include <openssl/cast.h>
#include <cast_lcl.h>
#include "cast_lcl.h"
```



```
void CAST_encrypt(CAST_LONG *data, const CAST_KEY *key)
        {
        register CAST_LONG l,r,t;
        register const CAST_LONG *k;
        const register CAST_LONG *k;

        k= &(key->data[0]);
        l=data[0];
        r=data[1];

        E_CAST( 0,k,l,r,+,^,-);
        E_CAST( 1,k,r,l,^,-,+);
        E_CAST( 2,k,l,r,-,+,^);
        E_CAST( 3,k,r,l,+,^,-);
        E_CAST( 4,k,l,r,^,-,+);
        E_CAST( 5,k,r,l,-,+,^);
        E_CAST( 6,k,l,r,+,^,-);
        E_CAST( 7,k,r,l,^,-,+);
        E_CAST( 8,k,l,r,-,+,^);
        E_CAST( 9,k,r,l,+,^,-);
        E_CAST(10,k,l,r,^,-,+);
        E_CAST(11,k,r,l,-,+,^);
        if(!key->short_key)
            {
            E_CAST(12,k,l,r,+,^,-);
            E_CAST(13,k,r,l,^,-,+);
            E_CAST(14,k,l,r,-,+,^);
            E_CAST(15,k,r,l,+,^,-);
            }

        data[1]=l&0xffffffffL;
        data[0]=r&0xffffffffL;
        }

void CAST_decrypt(CAST_LONG *data, const CAST_KEY *key)
        {
        register CAST_LONG l,r,t;
        register const CAST_LONG *k;
        const register CAST_LONG *k;

        k= &(key->data[0]);
        l=data[0];
        r=data[1];

        if(!key->short_key)
            {
            E_CAST(15,k,l,r,+,^,-);
            E_CAST(14,k,r,l,-,+,^);
            E_CAST(13,k,l,r,^,-,+);
            E_CAST(12,k,r,l,+,^,-);
            }
        E_CAST(11,k,l,r,-,+,^);
        E_CAST(10,k,r,l,^,-,+);
        E_CAST( 9,k,l,r,+,^,-);
        E_CAST( 8,k,r,l,-,+,^);
        E_CAST( 7,k,l,r,^,-,+);
        E_CAST( 6,k,r,l,+,^,-);
        E_CAST( 5,k,l,r,-,+,^);
        E_CAST( 4,k,r,l,^,-,+);
        E_CAST( 3,k,l,r,+,^,-);
        E_CAST( 2,k,r,l,-,+,^);
        E_CAST( 1,k,l,r,^,-,+);
        E_CAST( 0,k,r,l,+,^,-);

        data[1]=l&0xffffffffL;
```

```
125         data[0]=r&0xffffffffL;
126         }

128 void CAST_cbc_encrypt(const unsigned char *in, unsigned char *out, long length,
129              const CAST_KEY *ks, unsigned char *iv, int enc)
130         {
131         register CAST_LONG tin0,tin1;
132         register CAST_LONG tout0,tout1,xor0,xor1;
133         register long l=length;
134         CAST_LONG tin[2];

136         if (enc)
137                 {
138                 n2l(iv,tout0);
139                 n2l(iv,tout1);
140                 iv-=8;
141                 for (l-=8; l>=0; l-=8)
142                         {
143                         n2l(in,tin0);
144                         n2l(in,tin1);
145                         tin0^=tout0;
146                         tin1^=tout1;
147                         tin[0]=tin0;
148                         tin[1]=tin1;
149                         CAST_encrypt(tin,ks);
150                         tout0=tin[0];
151                         tout1=tin[1];
152                         l2n(tout0,out);
153                         l2n(tout1,out);
154                         }
155                 if (l != -8)
156                         {
157                         n2ln(in,tin0,tin1,l+8);
158                         tin0^=tout0;
159                         tin1^=tout1;
160                         tin[0]=tin0;
161                         tin[1]=tin1;
162                         CAST_encrypt(tin,ks);
163                         tout0=tin[0];
164                         tout1=tin[1];
165                         l2n(tout0,out);
166                         l2n(tout1,out);
167                         }
168                 l2n(tout0,iv);
169                 l2n(tout1,iv);
170                 }
171         else
172                 {
173                 n2l(iv,xor0);
174                 n2l(iv,xor1);
175                 iv-=8;
176                 for (l-=8; l>=0; l-=8)
177                         {
178                         n2l(in,tin0);
179                         n2l(in,tin1);
180                         tin[0]=tin0;
181                         tin[1]=tin1;
182                         CAST_decrypt(tin,ks);
183                         tout0=tin[0]^xor0;
184                         tout1=tin[1]^xor1;
185                         l2n(tout0,out);
186                         l2n(tout1,out);
187                         xor0=tin0;
188                         xor1=tin1;
189                         }
190                 if (l != -8)
```



```
191                         {
192                         n2l(in,tin0);
193                         n2l(in,tin1);
194                         tin[0]=tin0;
195                         tin[1]=tin1;
196                         CAST_decrypt(tin,ks);
197                         tout0=tin[0]^xor0;
198                         tout1=tin[1]^xor1;
199                         l2nn(tout0,tout1,out,l+8);
200                         xor0=tin0;
201                         xor1=tin1;
202                         }
203                 l2n(xor0,iv);
204                 l2n(xor1,iv);
205                 }
206         tin0=tin1=tout0=tout1=xor0=xor1=0;
207         tin[0]=tin[1]=0;
208         }
```

```
**********************************************************
    6668 Wed Aug 13 19:32:56 2014
new/usr/src/lib/openssl/libsunw_crypto/cast/c_skey.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/cast/c_skey.c */
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */

#include <openssl/crypto.h>
#include <openssl/cast.h>
#include <cast_lcl.h>
```



```
#include <cast_s.h>
#include "cast_lcl.h"
#include "cast_s.h"

#define CAST_exp(l,A,a,n) \
        A[n/4]=l; \
        a[n+3]=(l    )&0xff; \
        a[n+2]=(l>> 8)&0xff; \
        a[n+1]=(l>>16)&0xff; \
        a[n+0]=(l>>24)&0xff;

#define S4 CAST_S_table4
#define S5 CAST_S_table5
#define S6 CAST_S_table6
#define S7 CAST_S_table7
void CAST_set_key(CAST_KEY *key, int len, const unsigned char *data)
#ifdef OPENSSL_FIPS
        {
        fips_cipher_abort(CAST);
        private_CAST_set_key(key, len, data);
        }
void private_CAST_set_key(CAST_KEY *key, int len, const unsigned char *data)
#endif
        {
        CAST_LONG x[16];
        CAST_LONG z[16];
        CAST_LONG k[32];
        CAST_LONG X[4],Z[4];
        CAST_LONG l,*K;
        int i;

        for (i=0; i<16; i++) x[i]=0;
        if (len > 16) len=16;
        for (i=0; i<len; i++)
                x[i]=data[i];
        if(len <= 10)
            key->short_key=1;
        else
            key->short_key=0;

        K= &k[0];
        X[0]=((x[ 0]<<24)|(x[ 1]<<16)|(x[ 2]<<8)|x[ 3])&0xffffffffL;
        X[1]=((x[ 4]<<24)|(x[ 5]<<16)|(x[ 6]<<8)|x[ 7])&0xffffffffL;
        X[2]=((x[ 8]<<24)|(x[ 9]<<16)|(x[10]<<8)|x[11])&0xffffffffL;
        X[3]=((x[12]<<24)|(x[13]<<16)|(x[14]<<8)|x[15])&0xffffffffL;

        for (;;)
                {
        l=X[0]^S4[x[13]]^S5[x[15]]^S6[x[12]]^S7[x[14]]^S6[x[ 8]];
        CAST_exp(l,Z,z, 0);
        l=X[2]^S4[z[ 0]]^S5[z[ 2]]^S6[z[ 1]]^S7[z[ 3]]^S7[x[10]];
        CAST_exp(l,Z,z, 4);
        l=X[3]^S4[z[ 7]]^S5[z[ 6]]^S6[z[ 5]]^S7[z[ 4]]^S4[x[ 9]];
        CAST_exp(l,Z,z, 8);
        l=X[1]^S4[z[10]]^S5[z[ 9]]^S6[z[11]]^S7[z[ 8]]^S5[x[11]];
        CAST_exp(l,Z,z,12);

        K[ 0]= S4[z[ 8]]^S5[z[ 9]]^S6[z[ 7]]^S7[z[ 6]]^S4[z[ 2]];
        K[ 1]= S4[z[10]]^S5[z[11]]^S6[z[ 5]]^S7[z[ 4]]^S5[z[ 6]];
        K[ 2]= S4[z[12]]^S5[z[13]]^S6[z[ 3]]^S7[z[ 2]]^S6[z[ 9]];
        K[ 3]= S4[z[14]]^S5[z[15]]^S6[z[ 1]]^S7[z[ 0]]^S7[z[12]];

        l=Z[2]^S4[z[ 5]]^S5[z[ 7]]^S6[z[ 4]]^S7[z[ 6]]^S6[z[ 0]];
        CAST_exp(l,X,x, 0);
        l=Z[0]^S4[x[ 0]]^S5[x[ 2]]^S6[x[ 1]]^S7[x[ 3]]^S7[z[ 2]];
        CAST_exp(l,X,x, 4);
```

```
126         l=Z[1]^S4[x[ 7]]^S5[x[ 6]]^S6[x[ 5]]^S7[x[ 4]]^S4[z[ 1]];
127         CAST_exp(l,X,x, 8);
128         l=Z[3]^S4[x[10]]^S5[x[ 9]]^S6[x[11]]^S7[x[ 8]]^S5[z[ 3]];
129         CAST_exp(l,X,x,12);

131         K[ 4]= S4[x[ 3]]^S5[x[ 2]]^S6[x[12]]^S7[x[13]]^S4[x[ 8]];
132         K[ 5]= S4[x[ 1]]^S5[x[ 0]]^S6[x[14]]^S7[x[15]]^S5[x[13]];
133         K[ 6]= S4[x[ 7]]^S5[x[ 6]]^S6[x[ 8]]^S7[x[ 9]]^S6[x[ 3]];
134         K[ 7]= S4[x[ 5]]^S5[x[ 4]]^S6[x[10]]^S7[x[11]]^S7[x[ 7]];

136         l=X[0]^S4[x[13]]^S5[x[15]]^S6[x[12]]^S7[x[14]]^S6[x[ 8]];
137         CAST_exp(l,Z,z, 0);
138         l=X[2]^S4[z[ 0]]^S5[z[ 2]]^S6[z[ 1]]^S7[z[ 3]]^S7[x[10]];
139         CAST_exp(l,Z,z, 4);
140         l=X[3]^S4[z[ 7]]^S5[z[ 6]]^S6[z[ 5]]^S7[z[ 4]]^S4[x[ 9]];
141         CAST_exp(l,Z,z, 8);
142         l=X[1]^S4[z[10]]^S5[z[ 9]]^S6[z[11]]^S7[z[ 8]]^S5[x[11]];
143         CAST_exp(l,Z,z,12);

145         K[ 8]= S4[z[ 3]]^S5[z[ 2]]^S6[z[12]]^S7[z[13]]^S4[z[ 9]];
146         K[ 9]= S4[z[ 1]]^S5[z[ 0]]^S6[z[14]]^S7[z[15]]^S5[z[12]];
147         K[10]= S4[z[ 7]]^S5[z[ 6]]^S6[z[ 8]]^S7[z[ 9]]^S6[z[ 2]];
148         K[11]= S4[z[ 5]]^S5[z[ 4]]^S6[z[10]]^S7[z[11]]^S7[z[ 6]];

150         l=Z[2]^S4[z[ 5]]^S5[z[ 7]]^S6[z[ 4]]^S7[z[ 6]]^S6[z[ 0]];
151         CAST_exp(l,X,x, 0);
152         l=Z[0]^S4[x[ 0]]^S5[x[ 2]]^S6[x[ 1]]^S7[x[ 3]]^S7[z[ 2]];
153         CAST_exp(l,X,x, 4);
154         l=Z[1]^S4[x[ 7]]^S5[x[ 6]]^S6[x[ 5]]^S7[x[ 4]]^S4[z[ 1]];
155         CAST_exp(l,X,x, 8);
156         l=Z[3]^S4[x[10]]^S5[x[ 9]]^S6[x[11]]^S7[x[ 8]]^S5[z[ 3]];
157         CAST_exp(l,X,x,12);

159         K[12]= S4[x[ 8]]^S5[x[ 9]]^S6[x[ 7]]^S7[x[ 6]]^S4[x[ 3]];
160         K[13]= S4[x[10]]^S5[x[11]]^S6[x[ 5]]^S7[x[ 4]]^S5[x[ 7]];
161         K[14]= S4[x[12]]^S5[x[13]]^S6[x[ 3]]^S7[x[ 2]]^S6[x[ 8]];
162         K[15]= S4[x[14]]^S5[x[15]]^S6[x[ 1]]^S7[x[ 0]]^S7[x[13]];
163         if (K != k)  break;
164         K+=16;
165                 }

167         for (i=0; i<16; i++)
168                 {
169                 key->data[i*2]=k[i];
170                 key->data[i*2+1]=((k[i+16])+16)&0x1f;
171                 }
172         }
```

```
**********************************************************
   16753 Wed Aug 13 19:32:56 2014
new/usr/src/lib/openssl/libsunw_crypto/cms/cms_asn1.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/cms/cms_asn1.c */
   2 /* Written by Dr Stephen N Henson (steve@openssl.org) for the OpenSSL
   3  * project.
   4  */
   5 /* ====================================================================
   6  * Copyright (c) 2008 The OpenSSL Project.  All rights reserved.
   7  *
   8  * Redistribution and use in source and binary forms, with or without
   9  * modification, are permitted provided that the following conditions
  10  * are met:
  11  *
  12  * 1. Redistributions of source code must retain the above copyright
  13  *    notice, this list of conditions and the following disclaimer.
  14  *
  15  * 2. Redistributions in binary form must reproduce the above copyright
  16  *    notice, this list of conditions and the following disclaimer in
  17  *    the documentation and/or other materials provided with the
  18  *    distribution.
  19  *
  20  * 3. All advertising materials mentioning features or use of this
  21  *    software must display the following acknowledgment:
  22  *    "This product includes software developed by the OpenSSL Project
  23  *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
  24  *
  25  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  26  *    endorse or promote products derived from this software without
  27  *    prior written permission. For written permission, please contact
  28  *    licensing@OpenSSL.org.
  29  *
  30  * 5. Products derived from this software may not be called "OpenSSL"
  31  *    nor may "OpenSSL" appear in their names without prior written
  32  *    permission of the OpenSSL Project.
  33  *
  34  * 6. Redistributions of any form whatsoever must retain the following
  35  *    acknowledgment:
  36  *    "This product includes software developed by the OpenSSL Project
  37  *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
  38  *
  39  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
  40  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  41  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  42  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  43  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  44  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  45  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  46  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  47  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  48  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  49  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  50  * OF THE POSSIBILITY OF SUCH DAMAGE.
  51  * ====================================================================
  52  */

  54 #include <openssl/asn1t.h>
  55 #include <openssl/pem.h>
  56 #include <openssl/x509v3.h>
  57 #include <openssl/cms.h>
  58 #include <cms_lcl.h>
  57 #include "cms.h"
  58 #include "cms_lcl.h"
```



```
  61 ASN1_SEQUENCE(CMS_IssuerAndSerialNumber) = {
  62         ASN1_SIMPLE(CMS_IssuerAndSerialNumber, issuer, X509_NAME),
  63         ASN1_SIMPLE(CMS_IssuerAndSerialNumber, serialNumber, ASN1_INTEGER)
  64 } ASN1_SEQUENCE_END(CMS_IssuerAndSerialNumber)
______unchanged_portion_omitted_
```

```
**********************************************************
    6086 Wed Aug 13 19:32:56 2014
new/usr/src/lib/openssl/libsunw_crypto/cms/cms_att.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/cms/cms_att.c */
/* Written by Dr Stephen N Henson (steve@openssl.org) for the OpenSSL
 * project.
 */
/* ====================================================================
 * Copyright (c) 2008 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 */

#include <openssl/asn1t.h>
#include <openssl/pem.h>
#include <openssl/x509v3.h>
#include <openssl/err.h>
#include <openssl/cms.h>
#include <cms_lcl.h>
#include "cms.h"
#include "cms_lcl.h"
```



```
/* CMS SignedData Attribute utilities */

int CMS_signed_get_attr_count(const CMS_SignerInfo *si)
{
        return X509at_get_attr_count(si->signedAttrs);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    4662 Wed Aug 13 19:32:57 2014
new/usr/src/lib/openssl/libsunw_crypto/cms/cms_io.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/cms/cms_io.c */
/* Written by Dr Stephen N Henson (steve@openssl.org) for the OpenSSL
 * project.
 */
/* ====================================================================
 * Copyright (c) 2008 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 */

#include <openssl/asn1t.h>
#include <openssl/x509.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/cms.h>
#include <cms_lcl.h>
#include "cms.h"
#include "cms_lcl.h"
```



```
int CMS_stream(unsigned char ***boundary, CMS_ContentInfo *cms)
        {
        ASN1_OCTET_STRING **pos;
        pos = CMS_get0_content(cms);
        if (!pos)
                return 0;
        if (!*pos)
                *pos = ASN1_OCTET_STRING_new();
        if (*pos)
                {
                (*pos)->flags |= ASN1_STRING_FLAG_NDEF;
                (*pos)->flags &= ~ASN1_STRING_FLAG_CONT;
                *boundary = &(*pos)->data;
                return 1;
                }
        CMSerr(CMS_F_CMS_STREAM, ERR_R_MALLOC_FAILURE);
        return 0;
        }

CMS_ContentInfo *d2i_CMS_bio(BIO *bp, CMS_ContentInfo **cms)
        {
        return ASN1_item_d2i_bio(ASN1_ITEM_rptr(CMS_ContentInfo), bp, cms);
        }

int i2d_CMS_bio(BIO *bp, CMS_ContentInfo *cms)
        {
        return ASN1_item_i2d_bio(ASN1_ITEM_rptr(CMS_ContentInfo), bp, cms);
        }

IMPLEMENT_PEM_rw_const(CMS, CMS_ContentInfo, PEM_STRING_CMS, CMS_ContentInfo)

BIO *BIO_new_CMS(BIO *out, CMS_ContentInfo *cms)
        {
        return BIO_new_NDEF(out, (ASN1_VALUE *)cms,
                                ASN1_ITEM_rptr(CMS_ContentInfo));
        }

/* CMS wrappers round generalised stream and MIME routines */

int i2d_CMS_bio_stream(BIO *out, CMS_ContentInfo *cms, BIO *in, int flags)
        {
        return i2d_ASN1_bio_stream(out, (ASN1_VALUE *)cms, in, flags,
                                        ASN1_ITEM_rptr(CMS_ContentInfo));
        }

int PEM_write_bio_CMS_stream(BIO *out, CMS_ContentInfo *cms, BIO *in, int flags)
        {
        return PEM_write_bio_ASN1_stream(out, (ASN1_VALUE *) cms, in, flags,
                                        "CMS",
                                        ASN1_ITEM_rptr(CMS_ContentInfo));
        }

int SMIME_write_CMS(BIO *bio, CMS_ContentInfo *cms, BIO *data, int flags)
        {
        STACK_OF(X509_ALGOR) *mdalgs;
        int ctype_nid = OBJ_obj2nid(cms->contentType);
        int econt_nid = OBJ_obj2nid(CMS_get0_eContentType(cms));
        if (ctype_nid == NID_pkcs7_signed)
                mdalgs = cms->d.signedData->digestAlgorithms;
        else
                mdalgs = NULL;

        return SMIME_write_ASN1(bio, (ASN1_VALUE *)cms, data, flags,
                                        ctype_nid, econt_nid, mdalgs,
                                        ASN1_ITEM_rptr(CMS_ContentInfo));
```

```
126         }

128 CMS_ContentInfo *SMIME_read_CMS(BIO *bio, BIO **bcont)
129         {
130         return (CMS_ContentInfo *)SMIME_read_ASN1(bio, bcont,
131                                         ASN1_ITEM_rptr(CMS_ContentInfo));
132         }
```

```
**********************************************************
   15325 Wed Aug 13 19:32:57 2014
new/usr/src/lib/openssl/libsunw_crypto/cms/cms_lib.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/cms/cms_lib.c */
/* Written by Dr Stephen N Henson (steve@openssl.org) for the OpenSSL
 * project.
 */
/* ====================================================================
 * Copyright (c) 2008 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 */

#include <openssl/asn1t.h>
#include <openssl/x509.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/bio.h>
#include <openssl/asn1.h>
#include <openssl/cms.h>
#include <cms_lcl.h>
```



```
#include "cms.h"
#include "cms_lcl.h"

IMPLEMENT_ASN1_FUNCTIONS(CMS_ContentInfo)
IMPLEMENT_ASN1_PRINT_FUNCTION(CMS_ContentInfo)

DECLARE_ASN1_ITEM(CMS_CertificateChoices)
DECLARE_ASN1_ITEM(CMS_RevocationInfoChoice)
DECLARE_STACK_OF(CMS_CertificateChoices)
DECLARE_STACK_OF(CMS_RevocationInfoChoice)

const ASN1_OBJECT *CMS_get0_type(CMS_ContentInfo *cms)
        {
        return cms->contentType;
        }

CMS_ContentInfo *cms_Data_create(void)
        {
        CMS_ContentInfo *cms;
        cms = CMS_ContentInfo_new();
        if (cms)
                {
                cms->contentType = OBJ_nid2obj(NID_pkcs7_data);
                /* Never detached */
                CMS_set_detached(cms, 0);
                }
        return cms;
        }

BIO *cms_content_bio(CMS_ContentInfo *cms)
        {
        ASN1_OCTET_STRING **pos = CMS_get0_content(cms);
        if (!pos)
                return NULL;
        /* If content detached data goes nowhere: create NULL BIO */
        if (!*pos)
                return BIO_new(BIO_s_null());
        /* If content not detached and created return memory BIO
         */
        if (!*pos || ((*pos)->flags == ASN1_STRING_FLAG_CONT))
                return BIO_new(BIO_s_mem());
        /* Else content was read in: return read only BIO for it */
        return BIO_new_mem_buf((*pos)->data, (*pos)->length);
        }

BIO *CMS_dataInit(CMS_ContentInfo *cms, BIO *icont)
        {
        BIO *cmsbio, *cont;
        if (icont)
                cont = icont;
        else
                cont = cms_content_bio(cms);
        if (!cont)
                {
                CMSerr(CMS_F_CMS_DATAINIT, CMS_R_NO_CONTENT);
                return NULL;
                }
        switch (OBJ_obj2nid(cms->contentType))
                {

                case NID_pkcs7_data:
                return cont;

                case NID_pkcs7_signed:
                cmsbio = cms_SignedData_init_bio(cms);
                break;
```

```
127                 case NID_pkcs7_digest:
128                 cmsbio = cms_DigestedData_init_bio(cms);
129                 break;
130 #ifdef ZLIB
131                 case NID_id_smime_ct_compressedData:
132                 cmsbio = cms_CompressedData_init_bio(cms);
133                 break;
134 #endif

136                 case NID_pkcs7_encrypted:
137                 cmsbio = cms_EncryptedData_init_bio(cms);
138                 break;

140                 case NID_pkcs7_enveloped:
141                 cmsbio = cms_EnvelopedData_init_bio(cms);
142                 break;

144                 default:
145                 CMSerr(CMS_F_CMS_DATAINIT, CMS_R_UNSUPPORTED_TYPE);
146                 return NULL;
147                 }

149         if (cmsbio)
150                 return BIO_push(cmsbio, cont);

152         if (!icont)
153                 BIO_free(cont);
154         return NULL;

156         }

158 int CMS_dataFinal(CMS_ContentInfo *cms, BIO *cmsbio)
159         {
160         ASN1_OCTET_STRING **pos = CMS_get0_content(cms);
161         if (!pos)
162                 return 0;
163         /* If ebmedded content find memory BIO and set content */
164         if (*pos && ((*pos)->flags & ASN1_STRING_FLAG_CONT))
165                 {
166                 BIO *mbio;
167                 unsigned char *cont;
168                 long contlen;
169                 mbio = BIO_find_type(cmsbio, BIO_TYPE_MEM);
170                 if (!mbio)
171                         {
172                         CMSerr(CMS_F_CMS_DATAFINAL, CMS_R_CONTENT_NOT_FOUND);
173                         return 0;
174                         }
175                 contlen = BIO_get_mem_data(mbio, &cont);
176                 /* Set bio as read only so its content can't be clobbered */
177                 BIO_set_flags(mbio, BIO_FLAGS_MEM_RDONLY);
178                 BIO_set_mem_eof_return(mbio, 0);
179                 ASN1_STRING_set0(*pos, cont, contlen);
180                 (*pos)->flags &= ~ASN1_STRING_FLAG_CONT;
181                 }

183         switch (OBJ_obj2nid(cms->contentType))
184                 {

186                 case NID_pkcs7_data:
187                 case NID_pkcs7_enveloped:
188                 case NID_pkcs7_encrypted:
189                 case NID_id_smime_ct_compressedData:
190                 /* Nothing to do */
191                 return 1;
```



```
193                 case NID_pkcs7_signed:
194                 return cms_SignedData_final(cms, cmsbio);

196                 case NID_pkcs7_digest:
197                 return cms_DigestedData_do_final(cms, cmsbio, 0);

199                 default:
200                 CMSerr(CMS_F_CMS_DATAFINAL, CMS_R_UNSUPPORTED_TYPE);
201                 return 0;
202                 }
203         }

205 /* Return an OCTET STRING pointer to content. This allows it to
206  * be accessed or set later.
207  */

209 ASN1_OCTET_STRING **CMS_get0_content(CMS_ContentInfo *cms)
210         {
211         switch (OBJ_obj2nid(cms->contentType))
212                 {

214                 case NID_pkcs7_data:
215                 return &cms->d.data;

217                 case NID_pkcs7_signed:
218                 return &cms->d.signedData->encapContentInfo->eContent;

220                 case NID_pkcs7_enveloped:
221                 return &cms->d.envelopedData->encryptedContentInfo->encryptedCon

223                 case NID_pkcs7_digest:
224                 return &cms->d.digestedData->encapContentInfo->eContent;

226                 case NID_pkcs7_encrypted:
227                 return &cms->d.encryptedData->encryptedContentInfo->encryptedCon

229                 case NID_id_smime_ct_authData:
230                 return &cms->d.authenticatedData->encapContentInfo->eContent;

232                 case NID_id_smime_ct_compressedData:
233                 return &cms->d.compressedData->encapContentInfo->eContent;

235                 default:
236                 if (cms->d.other->type == V_ASN1_OCTET_STRING)
237                         return &cms->d.other->value.octet_string;
238                 CMSerr(CMS_F_CMS_GET0_CONTENT, CMS_R_UNSUPPORTED_CONTENT_TYPE);
239                 return NULL;

241                 }
242         }

244 /* Return an ASN1_OBJECT pointer to content type. This allows it to
245  * be accessed or set later.
246  */

248 static ASN1_OBJECT **cms_get0_econtent_type(CMS_ContentInfo *cms)
249         {
250         switch (OBJ_obj2nid(cms->contentType))
251                 {

253                 case NID_pkcs7_signed:
254                 return &cms->d.signedData->encapContentInfo->eContentType;

256                 case NID_pkcs7_enveloped:
257                 return &cms->d.envelopedData->encryptedContentInfo->contentType;
```

```
259                 case NID_pkcs7_digest:
260                 return &cms->d.digestedData->encapContentInfo->eContentType;

262                 case NID_pkcs7_encrypted:
263                 return &cms->d.encryptedData->encryptedContentInfo->contentType;

265                 case NID_id_smime_ct_authData:
266                 return &cms->d.authenticatedData->encapContentInfo->eContentType

268                 case NID_id_smime_ct_compressedData:
269                 return &cms->d.compressedData->encapContentInfo->eContentType;

271                 default:
272                 CMSerr(CMS_F_CMS_GET0_ECONTENT_TYPE,
273                                         CMS_R_UNSUPPORTED_CONTENT_TYPE);
274                 return NULL;

276                 }
277         }

279 const ASN1_OBJECT *CMS_get0_eContentType(CMS_ContentInfo *cms)
280         {
281         ASN1_OBJECT **petype;
282         petype = cms_get0_econtent_type(cms);
283         if (petype)
284                 return *petype;
285         return NULL;
286         }

288 int CMS_set1_eContentType(CMS_ContentInfo *cms, const ASN1_OBJECT *oid)
289         {
290         ASN1_OBJECT **petype, *etype;
291         petype = cms_get0_econtent_type(cms);
292         if (!petype)
293                 return 0;
294         if (!oid)
295                 return 1;
296         etype = OBJ_dup(oid);
297         if (!etype)
298                 return 0;
299         ASN1_OBJECT_free(*petype);
300         *petype = etype;
301         return 1;
302         }

304 int CMS_is_detached(CMS_ContentInfo *cms)
305         {
306         ASN1_OCTET_STRING **pos;
307         pos = CMS_get0_content(cms);
308         if (!pos)
309                 return -1;
310         if (*pos)
311                 return 0;
312         return 1;
313         }

315 int CMS_set_detached(CMS_ContentInfo *cms, int detached)
316         {
317         ASN1_OCTET_STRING **pos;
318         pos = CMS_get0_content(cms);
319         if (!pos)
320                 return 0;
321         if (detached)
322                 {
323                 if (*pos)
```



```
324                         {
325                         ASN1_OCTET_STRING_free(*pos);
326                         *pos = NULL;
327                         }
328                 return 1;
329                 }
330         if (!*pos)
331                 *pos = ASN1_OCTET_STRING_new();
332         if (*pos)
333                 {
334                 /* NB: special flag to show content is created and not
335                  * read in.
336                  */
337                 (*pos)->flags |= ASN1_STRING_FLAG_CONT;
338                 return 1;
339                 }
340         CMSerr(CMS_F_CMS_SET_DETACHED, ERR_R_MALLOC_FAILURE);
341         return 0;
342         }

344 /* Set up an X509_ALGOR DigestAlgorithmIdentifier from an EVP_MD */

346 void cms_DigestAlgorithm_set(X509_ALGOR *alg, const EVP_MD *md)
347         {
348         int param_type;

350         if (md->flags & EVP_MD_FLAG_DIGALGID_ABSENT)
351                 param_type = V_ASN1_UNDEF;
352         else
353                 param_type = V_ASN1_NULL;

355         X509_ALGOR_set0(alg, OBJ_nid2obj(EVP_MD_type(md)), param_type, NULL);

357         }

359 /* Create a digest BIO from an X509_ALGOR structure */

361 BIO *cms_DigestAlgorithm_init_bio(X509_ALGOR *digestAlgorithm)
362         {
363         BIO *mdbio = NULL;
364         ASN1_OBJECT *digestoid;
365         const EVP_MD *digest;
366         X509_ALGOR_get0(&digestoid, NULL, NULL, digestAlgorithm);
367         digest = EVP_get_digestbyobj(digestoid);
368         if (!digest)
369                 {
370                 CMSerr(CMS_F_CMS_DIGESTALGORITHM_INIT_BIO,
371                                 CMS_R_UNKNOWN_DIGEST_ALGORIHM);
372                 goto err;
373                 }
374         mdbio = BIO_new(BIO_f_md());
375         if (!mdbio || !BIO_set_md(mdbio, digest))
376                 {
377                 CMSerr(CMS_F_CMS_DIGESTALGORITHM_INIT_BIO,
378                                 CMS_R_MD_BIO_INIT_ERROR);
379                 goto err;
380                 }
381         return mdbio;
382         err:
383         if (mdbio)
384                 BIO_free(mdbio);
385         return NULL;
386         }

388 /* Locate a message digest content from a BIO chain based on SignerInfo */
```

```
390 int cms_DigestAlgorithm_find_ctx(EVP_MD_CTX *mctx, BIO *chain,
391                                   X509_ALGOR *mdalg)
392         {
393         int nid;
394         ASN1_OBJECT *mdoid;
395         X509_ALGOR_get0(&mdoid, NULL, NULL, mdalg);
396         nid = OBJ_obj2nid(mdoid);
397         /* Look for digest type to match signature */
398         for (;;)
399                 {
400                 EVP_MD_CTX *mtmp;
401                 chain = BIO_find_type(chain, BIO_TYPE_MD);
402                 if (chain == NULL)
403                         {
404                         CMSerr(CMS_F_CMS_DIGESTALGORITHM_FIND_CTX,
405                                         CMS_R_NO_MATCHING_DIGEST);
406                         return 0;
407                         }
408                 BIO_get_md_ctx(chain, &mtmp);
409                 if (EVP_MD_CTX_type(mtmp) == nid
410                 /* Workaround for broken implementations that use signature
411                  * algorithm  OID instead of digest.
412                  */
413                         || EVP_MD_pkey_type(EVP_MD_CTX_md(mtmp)) == nid)
414                         return EVP_MD_CTX_copy_ex(mctx, mtmp);
415                 chain = BIO_next(chain);
416                 }
417         }

419 static STACK_OF(CMS_CertificateChoices) **cms_get0_certificate_choices(CMS_Conte
420         {
421         switch (OBJ_obj2nid(cms->contentType))
422                 {

424                 case NID_pkcs7_signed:
425                 return &cms->d.signedData->certificates;

427                 case NID_pkcs7_enveloped:
428                 return &cms->d.envelopedData->originatorInfo->certificates;

430                 default:
431                 CMSerr(CMS_F_CMS_GET0_CERTIFICATE_CHOICES,
432                                         CMS_R_UNSUPPORTED_CONTENT_TYPE);
433                 return NULL;

435                 }
436         }

438 CMS_CertificateChoices *CMS_add0_CertificateChoices(CMS_ContentInfo *cms)
439         {
440         STACK_OF(CMS_CertificateChoices) **pcerts;
441         CMS_CertificateChoices *cch;
442         pcerts = cms_get0_certificate_choices(cms);
443         if (!pcerts)
444                 return NULL;
445         if (!*pcerts)
446                 *pcerts = sk_CMS_CertificateChoices_new_null();
447         if (!*pcerts)
448                 return NULL;
449         cch = M_ASN1_new_of(CMS_CertificateChoices);
450         if (!cch)
451                 return NULL;
452         if (!sk_CMS_CertificateChoices_push(*pcerts, cch))
453                 {
454                 M_ASN1_free_of(cch, CMS_CertificateChoices);
455                 return NULL;
```



```
456                 }
457         return cch;
458         }

460 int CMS_add0_cert(CMS_ContentInfo *cms, X509 *cert)
461         {
462         CMS_CertificateChoices *cch;
463         STACK_OF(CMS_CertificateChoices) **pcerts;
464         int i;
465         pcerts = cms_get0_certificate_choices(cms);
466         if (!pcerts)
467                 return 0;
468         for (i = 0; i < sk_CMS_CertificateChoices_num(*pcerts); i++)
469                 {
470                 cch = sk_CMS_CertificateChoices_value(*pcerts, i);
471                 if (cch->type == CMS_CERTCHOICE_CERT)
472                         {
473                         if (!X509_cmp(cch->d.certificate, cert))
474                                 {
475                                 CMSerr(CMS_F_CMS_ADD0_CERT,
476                                         CMS_R_CERTIFICATE_ALREADY_PRESENT);
477                                 return 0;
478                                 }
479                         }
480                 }
481         cch = CMS_add0_CertificateChoices(cms);
482         if (!cch)
483                 return 0;
484         cch->type = CMS_CERTCHOICE_CERT;
485         cch->d.certificate = cert;
486         return 1;
487         }

489 int CMS_add1_cert(CMS_ContentInfo *cms, X509 *cert)
490         {
491         int r;
492         r = CMS_add0_cert(cms, cert);
493         if (r > 0)
494                 CRYPTO_add(&cert->references, 1, CRYPTO_LOCK_X509);
495         return r;
496         }

498 static STACK_OF(CMS_RevocationInfoChoice) **cms_get0_revocation_choices(CMS_Cont
499         {
500         switch (OBJ_obj2nid(cms->contentType))
501                 {

503                 case NID_pkcs7_signed:
504                 return &cms->d.signedData->crls;

506                 case NID_pkcs7_enveloped:
507                 return &cms->d.envelopedData->originatorInfo->crls;

509                 default:
510                 CMSerr(CMS_F_CMS_GET0_REVOCATION_CHOICES,
511                                         CMS_R_UNSUPPORTED_CONTENT_TYPE);
512                 return NULL;

514                 }
515         }

517 CMS_RevocationInfoChoice *CMS_add0_RevocationInfoChoice(CMS_ContentInfo *cms)
518         {
519         STACK_OF(CMS_RevocationInfoChoice) **pcrls;
520         CMS_RevocationInfoChoice *rch;
521         pcrls = cms_get0_revocation_choices(cms);
```

```
522         if (!pcrls)
523                 return NULL;
524         if (!*pcrls)
525                 *pcrls = sk_CMS_RevocationInfoChoice_new_null();
526         if (!*pcrls)
527                 return NULL;
528         rch = M_ASN1_new_of(CMS_RevocationInfoChoice);
529         if (!rch)
530                 return NULL;
531         if (!sk_CMS_RevocationInfoChoice_push(*pcrls, rch))
532                 {
533                 M_ASN1_free_of(rch, CMS_RevocationInfoChoice);
534                 return NULL;
535                 }
536         return rch;
537         }

539 int CMS_add0_crl(CMS_ContentInfo *cms, X509_CRL *crl)
540         {
541         CMS_RevocationInfoChoice *rch;
542         rch = CMS_add0_RevocationInfoChoice(cms);
543         if (!rch)
544                 return 0;
545         rch->type = CMS_REVCHOICE_CRL;
546         rch->d.crl = crl;
547         return 1;
548         }

550 int CMS_add1_crl(CMS_ContentInfo *cms, X509_CRL *crl)
551         {
552         int r;
553         r = CMS_add0_crl(cms, crl);
554         if (r > 0)
555                 CRYPTO_add(&crl->references, 1, CRYPTO_LOCK_X509_CRL);
556         return r;
557         }

559 STACK_OF(X509) *CMS_get1_certs(CMS_ContentInfo *cms)
560         {
561         STACK_OF(X509) *certs = NULL;
562         CMS_CertificateChoices *cch;
563         STACK_OF(CMS_CertificateChoices) **pcerts;
564         int i;
565         pcerts = cms_get0_certificate_choices(cms);
566         if (!pcerts)
567                 return NULL;
568         for (i = 0; i < sk_CMS_CertificateChoices_num(*pcerts); i++)
569                 {
570                 cch = sk_CMS_CertificateChoices_value(*pcerts, i);
571                 if (cch->type == 0)
572                         {
573                         if (!certs)
574                                 {
575                                 certs = sk_X509_new_null();
576                                 if (!certs)
577                                         return NULL;
578                                 }
579                         if (!sk_X509_push(certs, cch->d.certificate))
580                                 {
581                                 sk_X509_pop_free(certs, X509_free);
582                                 return NULL;
583                                 }
584                         CRYPTO_add(&cch->d.certificate->references,
585                                                         1, CRYPTO_LOCK_X509);
586                         }
587                 }
```



```
588         return certs;

590         }

592 STACK_OF(X509_CRL) *CMS_get1_crls(CMS_ContentInfo *cms)
593         {
594         STACK_OF(X509_CRL) *crls = NULL;
595         STACK_OF(CMS_RevocationInfoChoice) **pcrls;
596         CMS_RevocationInfoChoice *rch;
597         int i;
598         pcrls = cms_get0_revocation_choices(cms);
599         if (!pcrls)
600                 return NULL;
601         for (i = 0; i < sk_CMS_RevocationInfoChoice_num(*pcrls); i++)
602                 {
603                 rch = sk_CMS_RevocationInfoChoice_value(*pcrls, i);
604                 if (rch->type == 0)
605                         {
606                         if (!crls)
607                                 {
608                                 crls = sk_X509_CRL_new_null();
609                                 if (!crls)
610                                         return NULL;
611                                 }
612                         if (!sk_X509_CRL_push(crls, rch->d.crl))
613                                 {
614                                 sk_X509_CRL_pop_free(crls, X509_CRL_free);
615                                 return NULL;
616                                 }
617                         CRYPTO_add(&rch->d.crl->references,
618                                                         1, CRYPTO_LOCK_X509_CRL);
619                         }
620                 }
621         return crls;
622         }
```

```
**********************************************************
   28436 Wed Aug 13 19:32:57 2014
new/usr/src/lib/openssl/libsunw_crypto/cryptlib.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/cryptlib.c */
   2 /* ====================================================================
   3  * Copyright (c) 1998-2006 The OpenSSL Project.  All rights reserved.
   4  *
   5  * Redistribution and use in source and binary forms, with or without
   6  * modification, are permitted provided that the following conditions
   7  * are met:
   8  *
   9  * 1. Redistributions of source code must retain the above copyright
  10  *    notice, this list of conditions and the following disclaimer.
  11  *
  12  * 2. Redistributions in binary form must reproduce the above copyright
  13  *    notice, this list of conditions and the following disclaimer in
  14  *    the documentation and/or other materials provided with the
  15  *    distribution.
  16  *
  17  * 3. All advertising materials mentioning features or use of this
  18  *    software must display the following acknowledgment:
  19  *    "This product includes software developed by the OpenSSL Project
  20  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
  21  *
  22  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  23  *    endorse or promote products derived from this software without
  24  *    prior written permission. For written permission, please contact
  25  *    openssl-core@openssl.org.
  26  *
  27  * 5. Products derived from this software may not be called "OpenSSL"
  28  *    nor may "OpenSSL" appear in their names without prior written
  29  *    permission of the OpenSSL Project.
  30  *
  31  * 6. Redistributions of any form whatsoever must retain the following
  32  *    acknowledgment:
  33  *    "This product includes software developed by the OpenSSL Project
  34  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
  35  *
  36  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
  37  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  38  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  39  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  40  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  41  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  42  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  43  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  44  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  45  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  46  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  47  * OF THE POSSIBILITY OF SUCH DAMAGE.
  48  * ====================================================================
  49  *
  50  * This product includes cryptographic software written by Eric Young
  51  * (eay@cryptsoft.com).  This product includes software written by Tim
  52  * Hudson (tjh@cryptsoft.com).
  53  *
  54  */
  55 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
  56  * All rights reserved.
  57  *
  58  * This package is an SSL implementation written
  59  * by Eric Young (eay@cryptsoft.com).
  60  * The implementation was written so as to conform with Netscapes SSL.
  61  *
```



```
  62  * This library is free for commercial and non-commercial use as long as
  63  * the following conditions are aheared to.  The following conditions
  64  * apply to all code found in this distribution, be it the RC4, RSA,
  65  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
  66  * included with this distribution is covered by the same copyright terms
  67  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  68  *
  69  * Copyright remains Eric Young's, and as such any Copyright notices in
  70  * the code are not to be removed.
  71  * If this package is used in a product, Eric Young should be given attribution
  72  * as the author of the parts of the library used.
  73  * This can be in the form of a textual message at program startup or
  74  * in documentation (online or textual) provided with the package.
  75  *
  76  * Redistribution and use in source and binary forms, with or without
  77  * modification, are permitted provided that the following conditions
  78  * are met:
  79  * 1. Redistributions of source code must retain the copyright
  80  *    notice, this list of conditions and the following disclaimer.
  81  * 2. Redistributions in binary form must reproduce the above copyright
  82  *    notice, this list of conditions and the following disclaimer in the
  83  *    documentation and/or other materials provided with the distribution.
  84  * 3. All advertising materials mentioning features or use of this software
  85  *    must display the following acknowledgement:
  86  *    "This product includes cryptographic software written by
  87  *     Eric Young (eay@cryptsoft.com)"
  88  *    The word 'cryptographic' can be left out if the rouines from the library
  89  *    being used are not cryptographic related :-).
  90  * 4. If you include any Windows specific code (or a derivative thereof) from
  91  *    the apps directory (application code) you must include an acknowledgement:
  92  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  93  *
  94  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
  95  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  96  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  97  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
  98  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  99  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 100  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 101  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 102  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 103  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 104  * SUCH DAMAGE.
 105  *
 106  * The licence and distribution terms for any publically available version or
 107  * derivative of this code cannot be changed.  i.e. this code cannot simply be
 108  * copied and put under another distribution licence
 109  * [including the GNU Public Licence.]
 110  */
 111 /* ====================================================================
 112  * Copyright 2002 Sun Microsystems, Inc. ALL RIGHTS RESERVED.
 113  * ECDH support in OpenSSL originally developed by
 114  * SUN MICROSYSTEMS, INC., and contributed to the OpenSSL project.
 115  */

 117 #include "cryptlib.h"
 118 #include <openssl/safestack.h>
 119 #include <pthread.h>
 120 #endif /* ! codereview */

 122 #if defined(OPENSSL_SYS_WIN32) || defined(OPENSSL_SYS_WIN16)
 123 static double SSLeay_MSVC5_hack=0.0; /* and for VC1.5 */
 124 #endif

 126 DECLARE_STACK_OF(CRYPTO_dynlock)
```

```
/* real #defines in crypto.h, keep these upto date */
static const char* const lock_names[CRYPTO_NUM_LOCKS] =
        {
        "<<ERROR>>",
        "err",
        "ex_data",
        "x509",
        "x509_info",
        "x509_pkey",
        "x509_crl",
        "x509_req",
        "dsa",
        "rsa",
        "evp_pkey",
        "x509_store",
        "ssl_ctx",
        "ssl_cert",
        "ssl_session",
        "ssl_sess_cert",
        "ssl",
        "ssl_method",
        "rand",
        "rand2",
        "debug_malloc",
        "BIO",
        "gethostbyname",
        "getservbyname",
        "readdir",
        "RSA_blinding",
        "dh",
        "debug_malloc2",
        "dso",
        "dynlock",
        "engine",
        "ui",
        "ecdsa",
        "ec",
        "ecdh",
        "bn",
        "ec_pre_comp",
        "store",
        "comp",
        "fips",
        "fips2",
#if CRYPTO_NUM_LOCKS != 41
# error "Inconsistency between crypto.h and cryptlib.c"
#endif
        };

/* This is for applications to allocate new type names in the non-dynamic
   array of lock names.  These are numbered with positive numbers.  */
static STACK_OF(OPENSSL_STRING) *app_locks=NULL;

/* For applications that want a more dynamic way of handling threads, the
   following stack is used.  These are externally numbered with negative
   numbers.  */
static STACK_OF(CRYPTO_dynlock) *dyn_locks=NULL;

static pthread_mutex_t *illumos_openssl_locks;
#endif /* ! codereview */

static void (MS_FAR *locking_callback)(int mode,int type,
        const char *file,int line)=0;
static int (MS_FAR *add_lock_callback)(int *pointer,int amount,
        int type,const char *file,int line)=0;
#ifndef OPENSSL_NO_DEPRECATED
```



```
static unsigned long (MS_FAR *id_callback)(void)=0;
#endif
static void (MS_FAR *threadid_callback)(CRYPTO_THREADID *)=0;
static struct CRYPTO_dynlock_value *(MS_FAR *dynlock_create_callback)
        (const char *file,int line)=0;
static void (MS_FAR *dynlock_lock_callback)(int mode,
        struct CRYPTO_dynlock_value *l, const char *file,int line)=0;
static void (MS_FAR *dynlock_destroy_callback)(struct CRYPTO_dynlock_value *l,
        const char *file,int line)=0;

int CRYPTO_get_new_lockid(char *name)
        {
        char *str;
        int i;

#if defined(OPENSSL_SYS_WIN32) || defined(OPENSSL_SYS_WIN16)
        /* A hack to make Visual C++ 5.0 work correctly when linking as
         * a DLL using /MT. Without this, the application cannot use
         * any floating point printf's.
         * It also seems to be needed for Visual C 1.5 (win16) */
        SSLeay_MSVC5_hack=(double)name[0]*(double)name[1];
#endif

        if ((app_locks == NULL) && ((app_locks=sk_OPENSSL_STRING_new_null()) ==
                {
                CRYPTOerr(CRYPTO_F_CRYPTO_GET_NEW_LOCKID,ERR_R_MALLOC_FAILURE);
                return(0);
                }
        if ((str=BUF_strdup(name)) == NULL)
                {
                CRYPTOerr(CRYPTO_F_CRYPTO_GET_NEW_LOCKID,ERR_R_MALLOC_FAILURE);
                return(0);
                }
        i=sk_OPENSSL_STRING_push(app_locks,str);
        if (!i)
                OPENSSL_free(str);
        else
                i+=CRYPTO_NUM_LOCKS; /* gap of one :-) */
        return(i);
        }

int CRYPTO_num_locks(void)
        {
        return CRYPTO_NUM_LOCKS;
        }

int CRYPTO_get_new_dynlockid(void)
        {
        int i = 0;
        CRYPTO_dynlock *pointer = NULL;

        if (dynlock_create_callback == NULL)
                {
                CRYPTOerr(CRYPTO_F_CRYPTO_GET_NEW_DYNLOCKID,CRYPTO_R_NO_DYNLOCK_
                return(0);
                }
        CRYPTO_w_lock(CRYPTO_LOCK_DYNLOCK);
        if ((dyn_locks == NULL)
                && ((dyn_locks=sk_CRYPTO_dynlock_new_null()) == NULL))
                {
                CRYPTO_w_unlock(CRYPTO_LOCK_DYNLOCK);
                CRYPTOerr(CRYPTO_F_CRYPTO_GET_NEW_DYNLOCKID,ERR_R_MALLOC_FAILURE
                return(0);
                }
        CRYPTO_w_unlock(CRYPTO_LOCK_DYNLOCK);
```

```
260         pointer = (CRYPTO_dynlock *)OPENSSL_malloc(sizeof(CRYPTO_dynlock));
261         if (pointer == NULL)
262                 {
263                 CRYPTOerr(CRYPTO_F_CRYPTO_GET_NEW_DYNLOCKID,ERR_R_MALLOC_FAILURE
264                 return(0);
265                 }
266         pointer->references = 1;
267         pointer->data = dynlock_create_callback(__FILE__,__LINE__);
268         if (pointer->data == NULL)
269                 {
270                 OPENSSL_free(pointer);
271                 CRYPTOerr(CRYPTO_F_CRYPTO_GET_NEW_DYNLOCKID,ERR_R_MALLOC_FAILURE
272                 return(0);
273                 }

275         CRYPTO_w_lock(CRYPTO_LOCK_DYNLOCK);
276         /* First, try to find an existing empty slot */
277         i=sk_CRYPTO_dynlock_find(dyn_locks,NULL);
278         /* If there was none, push, thereby creating a new one */
279         if (i == -1)
280                 /* Since sk_push() returns the number of items on the
281                    stack, not the location of the pushed item, we need
282                    to transform the returned number into a position,
283                    by decreasing it.  */
284                 i=sk_CRYPTO_dynlock_push(dyn_locks,pointer) - 1;
285         else
286                 /* If we found a place with a NULL pointer, put our pointer
287                    in it.  */
288                 (void)sk_CRYPTO_dynlock_set(dyn_locks,i,pointer);
289         CRYPTO_w_unlock(CRYPTO_LOCK_DYNLOCK);

291         if (i == -1)
292                 {
293                 dynlock_destroy_callback(pointer->data,__FILE__,__LINE__);
294                 OPENSSL_free(pointer);
295                 }
296         else
297                 i += 1; /* to avoid 0 */
298         return -i;
299         }

301 void CRYPTO_destroy_dynlockid(int i)
302         {
303         CRYPTO_dynlock *pointer = NULL;
304         if (i)
305                 i = -i-1;
306         if (dynlock_destroy_callback == NULL)
307                 return;

309         CRYPTO_w_lock(CRYPTO_LOCK_DYNLOCK);

311         if (dyn_locks == NULL || i >= sk_CRYPTO_dynlock_num(dyn_locks))
312                 {
313                 CRYPTO_w_unlock(CRYPTO_LOCK_DYNLOCK);
314                 return;
315                 }
316         pointer = sk_CRYPTO_dynlock_value(dyn_locks, i);
317         if (pointer != NULL)
318                 {
319                 --pointer->references;
320 #ifdef REF_CHECK
321                 if (pointer->references < 0)
322                         {
323                         fprintf(stderr,"CRYPTO_destroy_dynlockid, bad reference
324                         abort();
325                         }
```



```
326                 else
327 #endif
328                         if (pointer->references <= 0)
329                                 {
330                                 (void)sk_CRYPTO_dynlock_set(dyn_locks, i, NULL);
331                                 }
332                         else
333                                 pointer = NULL;
334                 }
335         CRYPTO_w_unlock(CRYPTO_LOCK_DYNLOCK);

337         if (pointer)
338                 {
339                 dynlock_destroy_callback(pointer->data,__FILE__,__LINE__);
340                 OPENSSL_free(pointer);
341                 }
342         }

344 struct CRYPTO_dynlock_value *CRYPTO_get_dynlock_value(int i)
345         {
346         CRYPTO_dynlock *pointer = NULL;
347         if (i)
348                 i = -i-1;

350         CRYPTO_w_lock(CRYPTO_LOCK_DYNLOCK);

352         if (dyn_locks != NULL && i < sk_CRYPTO_dynlock_num(dyn_locks))
353                 pointer = sk_CRYPTO_dynlock_value(dyn_locks, i);
354         if (pointer)
355                 pointer->references++;

357         CRYPTO_w_unlock(CRYPTO_LOCK_DYNLOCK);

359         if (pointer)
360                 return pointer->data;
361         return NULL;
362         }

364 struct CRYPTO_dynlock_value *(*CRYPTO_get_dynlock_create_callback(void))
365         (const char *file,int line)
366         {
367         return(dynlock_create_callback);
368         }

370 void (*CRYPTO_get_dynlock_lock_callback(void))(int mode,
371         struct CRYPTO_dynlock_value *l, const char *file,int line)
372         {
373         return(dynlock_lock_callback);
374         }

376 void (*CRYPTO_get_dynlock_destroy_callback(void))
377         (struct CRYPTO_dynlock_value *l, const char *file,int line)
378         {
379         return(dynlock_destroy_callback);
380         }

382 void CRYPTO_set_dynlock_create_callback(struct CRYPTO_dynlock_value *(*func)
383         (const char *file, int line))
384         {
385         dynlock_create_callback=func;
386         }

388 void CRYPTO_set_dynlock_lock_callback(void (*func)(int mode,
389         struct CRYPTO_dynlock_value *l, const char *file, int line))
390         {
391         dynlock_lock_callback=func;
```

```
392         }

394 void CRYPTO_set_dynlock_destroy_callback(void (*func)
395         (struct CRYPTO_dynlock_value *l, const char *file, int line))
396         {
397         dynlock_destroy_callback=func;
398         }


401 void (*CRYPTO_get_locking_callback(void))(int mode,int type,const char *file,
402                 int line)
403         {
404         return(locking_callback);
405         }

407 int (*CRYPTO_get_add_lock_callback(void))(int *num,int mount,int type,
408                                           const char *file,int line)
409         {
410         return(add_lock_callback);
411         }

413 /*
414  * This is the locking callback function which all applications will be
415  * using when CRYPTO_lock() is called.
416  */
417 static void illumos_locking_callback(int mode, int type, const char *file,
418     int line)
419         {
420         if (mode & CRYPTO_LOCK)
421                 {
422                 pthread_mutex_lock(&illumos_openssl_locks[type]);
423                 }
424         else
425                 {
426                 pthread_mutex_unlock(&illumos_openssl_locks[type]);
427                 }
428         }


431 /*
432  * This function is called when a child process is forked to setup its own
433  * global locking callback function ptr and mutexes.
434  */
435 static void illumos_fork_child(void)
436         {
437                 /*
438                  * clear locking_callback to indicate that locks should
439                  * be reinitialized.
440                  */
441                 locking_callback = NULL;
442                 illumos_locking_setup();
443         }

445 /*
446  * This function allocates and initializes the global mutex array, and
447  * sets the locking callback.
448  */
449 void illumos_locking_setup()
450         {
451         int i;
452         int num_locks;

454         /* locking callback is already setup. Nothing to do */
455         if (locking_callback != NULL)
456                 {
457                 return;
```



```
458                 }

460         /*
461          * Set atfork handler so that child can setup its own mutexes and
462          * locking callbacks when it is forked
463          */
464         (void) pthread_atfork(NULL, NULL, illumos_fork_child);

466         /* allocate locks needed by OpenSSL  */
467         num_locks = CRYPTO_num_locks();
468         illumos_openssl_locks =
469             OPENSSL_malloc(sizeof (pthread_mutex_t) * num_locks);
470         if (illumos_openssl_locks == NULL)
471                 {
472                 fprintf(stderr,
473                         "illumos_locking_setup: memory allocation failure.\n");
474                 abort();
475                 }

477         /* initialize openssl mutexes */
478         for (i = 0; i < num_locks; i++)
479                 {
480                 pthread_mutex_init(&illumos_openssl_locks[i], NULL);
481                 }
482         locking_callback = illumos_locking_callback;

484         }

486 #endif /* ! codereview */
487 void CRYPTO_set_locking_callback(void (*func)(int mode,int type,
488                                               const char *file,int line))
489         {
490         /* Calling this here ensures initialisation before any threads
491          * are started.
492          */
493         OPENSSL_init();

495         /*
496          * we now setup our own locking callback and mutexes, and disallow
497          * setting of another locking callback.
498          */
119         locking_callback=func;
499         }

501 void CRYPTO_set_add_lock_callback(int (*func)(int *num,int mount,int type,
502                                               const char *file,int line))
503         {
504         add_lock_callback=func;
505         }

507 /* the memset() here and in set_pointer() seem overkill, but for the sake of
508  * CRYPTO_THREADID_cmp() this avoids any platform silliness that might cause two
509  * "equal" THREADID structs to not be memcmp()-identical. */
510 void CRYPTO_THREADID_set_numeric(CRYPTO_THREADID *id, unsigned long val)
511         {
512         memset(id, 0, sizeof(*id));
513         id->val = val;
514         }

516 static const unsigned char hash_coeffs[] = { 3, 5, 7, 11, 13, 17, 19, 23 };
517 void CRYPTO_THREADID_set_pointer(CRYPTO_THREADID *id, void *ptr)
518         {
519         unsigned char *dest = (void *)&id->val;
520         unsigned int accum = 0;
521         unsigned char dnum = sizeof(id->val);
```

```
523         memset(id, 0, sizeof(*id));
524         id->ptr = ptr;
525         if (sizeof(id->val) >= sizeof(id->ptr))
526                 {
527                 /* 'ptr' can be embedded in 'val' without loss of uniqueness */
528                 id->val = (unsigned long)id->ptr;
529                 return;
530                 }
531         /* hash ptr ==> val. Each byte of 'val' gets the mod-256 total of a
532          * linear function over the bytes in 'ptr', the co-efficients of which
533          * are a sequence of low-primes (hash_coeffs is an 8-element cycle) -
534          * the starting prime for the sequence varies for each byte of 'val'
535          * (unique polynomials unless pointers are >64-bit). For added spice,
536          * the totals accumulate rather than restarting from zero, and the index
537          * of the 'val' byte is added each time (position dependence). If I was
538          * a black-belt, I'd scan big-endian pointers in reverse to give
539          * low-order bits more play, but this isn't crypto and I'd prefer nobody
540          * mistake it as such. Plus I'm lazy. */
541         while (dnum--)
542                 {
543                 const unsigned char *src = (void *)&id->ptr;
544                 unsigned char snum = sizeof(id->ptr);
545                 while (snum--)
546                         accum += *(src++) * hash_coeffs[(snum + dnum) & 7];
547                 accum += dnum;
548                 *(dest++) = accum & 255;
549                 }
550         }

552 int CRYPTO_THREADID_set_callback(void (*func)(CRYPTO_THREADID *))
553         {
554         if (threadid_callback)
555                 return 0;
556         threadid_callback = func;
557         return 1;
558         }

560 void (*CRYPTO_THREADID_get_callback(void))(CRYPTO_THREADID *)
561         {
562         return threadid_callback;
563         }

565 void CRYPTO_THREADID_current(CRYPTO_THREADID *id)
566         {
567         if (threadid_callback)
568                 {
569                 threadid_callback(id);
570                 return;
571                 }
572 #ifndef OPENSSL_NO_DEPRECATED
573         /* If the deprecated callback was set, fall back to that */
574         if (id_callback)
575                 {
576                 CRYPTO_THREADID_set_numeric(id, id_callback());
577                 return;
578                 }
579 #endif
580         /* Else pick a backup */
581 #ifdef OPENSSL_SYS_WIN16
582         CRYPTO_THREADID_set_numeric(id, (unsigned long)GetCurrentTask());
583 #elif defined(OPENSSL_SYS_WIN32)
584         CRYPTO_THREADID_set_numeric(id, (unsigned long)GetCurrentThreadId());
585 #elif defined(OPENSSL_SYS_BEOS)
586         CRYPTO_THREADID_set_numeric(id, (unsigned long)find_thread(NULL));
587 #else
588         /* For everything else, default to using the address of 'errno' */
```



```
589         CRYPTO_THREADID_set_pointer(id, (void*)&errno);
590 #endif
591         }

593 int CRYPTO_THREADID_cmp(const CRYPTO_THREADID *a, const CRYPTO_THREADID *b)
594         {
595         return memcmp(a, b, sizeof(*a));
596         }

598 void CRYPTO_THREADID_cpy(CRYPTO_THREADID *dest, const CRYPTO_THREADID *src)
599         {
600         memcpy(dest, src, sizeof(*src));
601         }

603 unsigned long CRYPTO_THREADID_hash(const CRYPTO_THREADID *id)
604         {
605         return id->val;
606         }

608 #ifndef OPENSSL_NO_DEPRECATED
609 unsigned long (*CRYPTO_get_id_callback(void))(void)
610         {
611         return(id_callback);
612         }

614 void CRYPTO_set_id_callback(unsigned long (*func)(void))
615         {
616         id_callback=func;
617         }

619 unsigned long CRYPTO_thread_id(void)
620         {
621         unsigned long ret=0;

623         if (id_callback == NULL)
624                 {
625 #ifdef OPENSSL_SYS_WIN16
626                 ret=(unsigned long)GetCurrentTask();
627 #elif defined(OPENSSL_SYS_WIN32)
628                 ret=(unsigned long)GetCurrentThreadId();
629 #elif defined(GETPID_IS_MEANINGLESS)
630                 ret=1L;
631 #elif defined(OPENSSL_SYS_BEOS)
632                 ret=(unsigned long)find_thread(NULL);
633 #else
634                 ret=(unsigned long)getpid();
635 #endif
636                 }
637         else
638                 ret=id_callback();
639         return(ret);
640         }
641 #endif

643 void CRYPTO_lock(int mode, int type, const char *file, int line)
644         {
645 #ifdef LOCK_DEBUG
646                 {
647                 CRYPTO_THREADID id;
648                 char *rw_text,*operation_text;

650                 if (mode & CRYPTO_LOCK)
651                         operation_text="lock  ";
652                 else if (mode & CRYPTO_UNLOCK)
653                         operation_text="unlock";
654                 else
```

```
655                         operation_text="ERROR ";

657                 if (mode & CRYPTO_READ)
658                         rw_text="r";
659                 else if (mode & CRYPTO_WRITE)
660                         rw_text="w";
661                 else
662                         rw_text="ERROR";

664                 CRYPTO_THREADID_current(&id);
665                 fprintf(stderr,"lock:%08lx:(%s)%s %-18s %s:%d\n",
666                         CRYPTO_THREADID_hash(&id), rw_text, operation_text,
667                         CRYPTO_get_lock_name(type), file, line);
668                 }
669 #endif
670         if (type < 0)
671                 {
672                 if (dynlock_lock_callback != NULL)
673                         {
674                         struct CRYPTO_dynlock_value *pointer
675                                 = CRYPTO_get_dynlock_value(type);

677                         OPENSSL_assert(pointer != NULL);

679                         dynlock_lock_callback(mode, pointer, file, line);

681                         CRYPTO_destroy_dynlockid(type);
682                         }
683                 }
684         else
685                 if (locking_callback != NULL)
686                         locking_callback(mode,type,file,line);
687         }

689 int CRYPTO_add_lock(int *pointer, int amount, int type, const char *file,
690              int line)
691         {
692         int ret = 0;

694         if (add_lock_callback != NULL)
695                 {
696 #ifdef LOCK_DEBUG
697                 int before= *pointer;
698 #endif

700                 ret=add_lock_callback(pointer,amount,type,file,line);
701 #ifdef LOCK_DEBUG
702                 {
703                 CRYPTO_THREADID id;
704                 CRYPTO_THREADID_current(&id);
705                 fprintf(stderr,"ladd:%08lx:%2d+%2d->%2d %-18s %s:%d\n",
706                         CRYPTO_THREADID_hash(&id), before,amount,ret,
707                         CRYPTO_get_lock_name(type),
708                         file,line);
709                 }
710 #endif
711                 }
712         else
713                 {
714                 CRYPTO_lock(CRYPTO_LOCK|CRYPTO_WRITE,type,file,line);

716                 ret= *pointer+amount;
717 #ifdef LOCK_DEBUG
718                 {
719                 CRYPTO_THREADID id;
720                 CRYPTO_THREADID_current(&id);
```



```
721                 fprintf(stderr,"ladd:%08lx:%2d+%2d->%2d %-18s %s:%d\n",
722                         CRYPTO_THREADID_hash(&id),
723                         *pointer,amount,ret,
724                         CRYPTO_get_lock_name(type),
725                         file,line);
726                 }
727 #endif
728                 *pointer=ret;
729                 CRYPTO_lock(CRYPTO_UNLOCK|CRYPTO_WRITE,type,file,line);
730                 }
731         return(ret);
732         }

734 const char *CRYPTO_get_lock_name(int type)
735         {
736         if (type < 0)
737                 return("dynamic");
738         else if (type < CRYPTO_NUM_LOCKS)
739                 return(lock_names[type]);
740         else if (type-CRYPTO_NUM_LOCKS > sk_OPENSSL_STRING_num(app_locks))
741                 return("ERROR");
742         else
743                 return(sk_OPENSSL_STRING_value(app_locks,type-CRYPTO_NUM_LOCKS))
744         }

746 #if     defined(__i386)   || defined(__i386__)   || defined(_M_IX86) || \
747         defined(__INTEL__) || \
748         defined(__x86_64) || defined(__x86_64__) || defined(_M_AMD64) || defined

750 unsigned int  OPENSSL_ia32cap_P[2];
751 unsigned long *OPENSSL_ia32cap_loc(void)
752 {   if (sizeof(long)==4)
753         /*
754          * If 32-bit application pulls address of OPENSSL_ia32cap_P[0]
755          * clear second element to maintain the illusion that vector
756          * is 32-bit.
757          */
758         OPENSSL_ia32cap_P[1]=0;
759     return (unsigned long *)OPENSSL_ia32cap_P;
760 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   10517 Wed Aug 13 19:32:58 2014
new/usr/src/lib/openssl/libsunw_crypto/des/des_enc.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/des/des_enc.c */
   2 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
   3  * All rights reserved.
   4  *
   5  * This package is an SSL implementation written
   6  * by Eric Young (eay@cryptsoft.com).
   7  * The implementation was written so as to conform with Netscapes SSL.
   8  *
   9  * This library is free for commercial and non-commercial use as long as
  10  * the following conditions are aheared to.  The following conditions
  11  * apply to all code found in this distribution, be it the RC4, RSA,
  12  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
  13  * included with this distribution is covered by the same copyright terms
  14  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  15  *
  16  * Copyright remains Eric Young's, and as such any Copyright notices in
  17  * the code are not to be removed.
  18  * If this package is used in a product, Eric Young should be given attribution
  19  * as the author of the parts of the library used.
  20  * This can be in the form of a textual message at program startup or
  21  * in documentation (online or textual) provided with the package.
  22  *
  23  * Redistribution and use in source and binary forms, with or without
  24  * modification, are permitted provided that the following conditions
  25  * are met:
  26  * 1. Redistributions of source code must retain the copyright
  27  *    notice, this list of conditions and the following disclaimer.
  28  * 2. Redistributions in binary form must reproduce the above copyright
  29  *    notice, this list of conditions and the following disclaimer in the
  30  *    documentation and/or other materials provided with the distribution.
  31  * 3. All advertising materials mentioning features or use of this software
  32  *    must display the following acknowledgement:
  33  *    "This product includes cryptographic software written by
  34  *     Eric Young (eay@cryptsoft.com)"
  35  *    The word 'cryptographic' can be left out if the rouines from the library
  36  *    being used are not cryptographic related :-).
  37  * 4. If you include any Windows specific code (or a derivative thereof) from
  38  *    the apps directory (application code) you must include an acknowledgement:
  39  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  40  *
  41  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
  42  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  43  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  44  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
  45  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  46  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  47  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  48  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  49  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  50  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  51  * SUCH DAMAGE.
  52  *
  53  * The licence and distribution terms for any publically available version or
  54  * derivative of this code cannot be changed.  i.e. this code cannot simply be
  55  * copied and put under another distribution licence
  56  * [including the GNU Public Licence.]
  57  */

  59 #include <des_locl.h>
  60 #include <spr.h>
  59 #include "des_locl.h"
```



```
  60 #include "spr.h"

  62 void DES_encrypt1(DES_LONG *data, DES_key_schedule *ks, int enc)
  63         {
  64         register DES_LONG l,r,t,u;
  65 #ifdef DES_PTR
  66         register const unsigned char *des_SP=(const unsigned char *)DES_SPtrans;
  67 #endif
  68 #ifndef DES_UNROLL
  69         register int i;
  70 #endif
  71         register DES_LONG *s;

  73         r=data[0];
  74         l=data[1];

  76         IP(r,l);
  77         /* Things have been modified so that the initial rotate is
  78          * done outside the loop.  This required the
  79          * DES_SPtrans values in sp.h to be rotated 1 bit to the right.
  80          * One perl script later and things have a 5% speed up on a sparc2.
  81          * Thanks to Richard Outerbridge <71755.204@CompuServe.COM>
  82          * for pointing this out. */
  83         /* clear the top bits on machines with 8byte longs */
  84         /* shift left by 2 */
  85         r=ROTATE(r,29)&0xffffffffL;
  86         l=ROTATE(l,29)&0xffffffffL;

  88         s=ks->ks->deslong;
  89         /* I don't know if it is worth the effort of loop unrolling the
  90          * inner loop */
  91         if (enc)
  92                 {
  93 #ifdef DES_UNROLL
  94                 D_ENCRYPT(l,r, 0); /*  1 */
  95                 D_ENCRYPT(r,l, 2); /*  2 */
  96                 D_ENCRYPT(l,r, 4); /*  3 */
  97                 D_ENCRYPT(r,l, 6); /*  4 */
  98                 D_ENCRYPT(l,r, 8); /*  5 */
  99                 D_ENCRYPT(r,l,10); /*  6 */
 100                 D_ENCRYPT(l,r,12); /*  7 */
 101                 D_ENCRYPT(r,l,14); /*  8 */
 102                 D_ENCRYPT(l,r,16); /*  9 */
 103                 D_ENCRYPT(r,l,18); /*  10 */
 104                 D_ENCRYPT(l,r,20); /*  11 */
 105                 D_ENCRYPT(r,l,22); /*  12 */
 106                 D_ENCRYPT(l,r,24); /*  13 */
 107                 D_ENCRYPT(r,l,26); /*  14 */
 108                 D_ENCRYPT(l,r,28); /*  15 */
 109                 D_ENCRYPT(r,l,30); /*  16 */
 110 #else
 111                 for (i=0; i<32; i+=4)
 112                         {
 113                         D_ENCRYPT(l,r,i+0); /*  1 */
 114                         D_ENCRYPT(r,l,i+2); /*  2 */
 115                         }
 116 #endif
 117                 }
 118         else
 119                 {
 120 #ifdef DES_UNROLL
 121                 D_ENCRYPT(l,r,30); /* 16 */
 122                 D_ENCRYPT(r,l,28); /* 15 */
 123                 D_ENCRYPT(l,r,26); /* 14 */
 124                 D_ENCRYPT(r,l,24); /* 13 */
 125                 D_ENCRYPT(l,r,22); /* 12 */
```

```
126                 D_ENCRYPT(r,l,20); /* 11 */
127                 D_ENCRYPT(l,r,18); /* 10 */
128                 D_ENCRYPT(r,l,16); /*  9 */
129                 D_ENCRYPT(l,r,14); /*  8 */
130                 D_ENCRYPT(r,l,12); /*  7 */
131                 D_ENCRYPT(l,r,10); /*  6 */
132                 D_ENCRYPT(r,l, 8); /*  5 */
133                 D_ENCRYPT(l,r, 6); /*  4 */
134                 D_ENCRYPT(r,l, 4); /*  3 */
135                 D_ENCRYPT(l,r, 2); /*  2 */
136                 D_ENCRYPT(r,l, 0); /*  1 */
137 #else
138                 for (i=30; i>0; i-=4)
139                         {
140                         D_ENCRYPT(l,r,i-0); /* 16 */
141                         D_ENCRYPT(r,l,i-2); /* 15 */
142                         }
143 #endif
144                 }

146         /* rotate and clear the top bits on machines with 8byte longs */
147         l=ROTATE(l,3)&0xffffffffL;
148         r=ROTATE(r,3)&0xffffffffL;

150         FP(r,l);
151         data[0]=l;
152         data[1]=r;
153         l=r=t=u=0;
154         }

156 void DES_encrypt2(DES_LONG *data, DES_key_schedule *ks, int enc)
157         {
158         register DES_LONG l,r,t,u;
159 #ifdef DES_PTR
160         register const unsigned char *des_SP=(const unsigned char *)DES_SPtrans;
161 #endif
162 #ifndef DES_UNROLL
163         register int i;
164 #endif
165         register DES_LONG *s;

167         r=data[0];
168         l=data[1];

170         /* Things have been modified so that the initial rotate is
171          * done outside the loop.  This required the
172          * DES_SPtrans values in sp.h to be rotated 1 bit to the right.
173          * One perl script later and things have a 5% speed up on a sparc2.
174          * Thanks to Richard Outerbridge <71755.204@CompuServe.COM>
175          * for pointing this out. */
176         /* clear the top bits on machines with 8byte longs */
177         r=ROTATE(r,29)&0xffffffffL;
178         l=ROTATE(l,29)&0xffffffffL;

180         s=ks->ks->deslong;
181         /* I don't know if it is worth the effort of loop unrolling the
182          * inner loop */
183         if (enc)
184                 {
185 #ifdef DES_UNROLL
186                 D_ENCRYPT(l,r, 0); /*  1 */
187                 D_ENCRYPT(r,l, 2); /*  2 */
188                 D_ENCRYPT(l,r, 4); /*  3 */
189                 D_ENCRYPT(r,l, 6); /*  4 */
190                 D_ENCRYPT(l,r, 8); /*  5 */
191                 D_ENCRYPT(r,l,10); /*  6 */
```



```
192                 D_ENCRYPT(l,r,12); /*  7 */
193                 D_ENCRYPT(r,l,14); /*  8 */
194                 D_ENCRYPT(l,r,16); /*  9 */
195                 D_ENCRYPT(r,l,18); /*  10 */
196                 D_ENCRYPT(l,r,20); /*  11 */
197                 D_ENCRYPT(r,l,22); /*  12 */
198                 D_ENCRYPT(l,r,24); /*  13 */
199                 D_ENCRYPT(r,l,26); /*  14 */
200                 D_ENCRYPT(l,r,28); /*  15 */
201                 D_ENCRYPT(r,l,30); /*  16 */
202 #else
203                 for (i=0; i<32; i+=4)
204                         {
205                         D_ENCRYPT(l,r,i+0); /*  1 */
206                         D_ENCRYPT(r,l,i+2); /*  2 */
207                         }
208 #endif
209                 }
210         else
211                 {
212 #ifdef DES_UNROLL
213                 D_ENCRYPT(l,r,30); /* 16 */
214                 D_ENCRYPT(r,l,28); /* 15 */
215                 D_ENCRYPT(l,r,26); /* 14 */
216                 D_ENCRYPT(r,l,24); /* 13 */
217                 D_ENCRYPT(l,r,22); /* 12 */
218                 D_ENCRYPT(r,l,20); /* 11 */
219                 D_ENCRYPT(l,r,18); /* 10 */
220                 D_ENCRYPT(r,l,16); /*  9 */
221                 D_ENCRYPT(l,r,14); /*  8 */
222                 D_ENCRYPT(r,l,12); /*  7 */
223                 D_ENCRYPT(l,r,10); /*  6 */
224                 D_ENCRYPT(r,l, 8); /*  5 */
225                 D_ENCRYPT(l,r, 6); /*  4 */
226                 D_ENCRYPT(r,l, 4); /*  3 */
227                 D_ENCRYPT(l,r, 2); /*  2 */
228                 D_ENCRYPT(r,l, 0); /*  1 */
229 #else
230                 for (i=30; i>0; i-=4)
231                         {
232                         D_ENCRYPT(l,r,i-0); /* 16 */
233                         D_ENCRYPT(r,l,i-2); /* 15 */
234                         }
235 #endif
236                 }
237         /* rotate and clear the top bits on machines with 8byte longs */
238         data[0]=ROTATE(l,3)&0xffffffffL;
239         data[1]=ROTATE(r,3)&0xffffffffL;
240         l=r=t=u=0;
241         }

243 void DES_encrypt3(DES_LONG *data, DES_key_schedule *ks1,
244                   DES_key_schedule *ks2, DES_key_schedule *ks3)
245         {
246         register DES_LONG l,r;

248         l=data[0];
249         r=data[1];
250         IP(l,r);
251         data[0]=l;
252         data[1]=r;
253         DES_encrypt2((DES_LONG *)data,ks1,DES_ENCRYPT);
254         DES_encrypt2((DES_LONG *)data,ks2,DES_DECRYPT);
255         DES_encrypt2((DES_LONG *)data,ks3,DES_ENCRYPT);
256         l=data[0];
257         r=data[1];
```

```
258         FP(r,l);
259         data[0]=l;
260         data[1]=r;
261         }

263 void DES_decrypt3(DES_LONG *data, DES_key_schedule *ks1,
264                    DES_key_schedule *ks2, DES_key_schedule *ks3)
265         {
266         register DES_LONG l,r;

268         l=data[0];
269         r=data[1];
270         IP(l,r);
271         data[0]=l;
272         data[1]=r;
273         DES_encrypt2((DES_LONG *)data,ks3,DES_DECRYPT);
274         DES_encrypt2((DES_LONG *)data,ks2,DES_ENCRYPT);
275         DES_encrypt2((DES_LONG *)data,ks1,DES_DECRYPT);
276         l=data[0];
277         r=data[1];
278         FP(r,l);
279         data[0]=l;
280         data[1]=r;
281         }

283 #ifndef DES_DEFAULT_OPTIONS

285 #undef CBC_ENC_C__DONT_UPDATE_IV
286 #include "ncbc_enc.c" /* DES_ncbc_encrypt */

288 void DES_ede3_cbc_encrypt(const unsigned char *input, unsigned char *output,
289                           long length, DES_key_schedule *ks1,
290                           DES_key_schedule *ks2, DES_key_schedule *ks3,
291                           DES_cblock *ivec, int enc)
292         {
293         register DES_LONG tin0,tin1;
294         register DES_LONG tout0,tout1,xor0,xor1;
295         register const unsigned char *in;
296         unsigned char *out;
297         register long l=length;
298         DES_LONG tin[2];
299         unsigned char *iv;

301         in=input;
302         out=output;
303         iv = &(*ivec)[0];

305         if (enc)
306                 {
307                 c2l(iv,tout0);
308                 c2l(iv,tout1);
309                 for (l-=8; l>=0; l-=8)
310                         {
311                         c2l(in,tin0);
312                         c2l(in,tin1);
313                         tin0^=tout0;
314                         tin1^=tout1;

316                         tin[0]=tin0;
317                         tin[1]=tin1;
318                         DES_encrypt3((DES_LONG *)tin,ks1,ks2,ks3);
319                         tout0=tin[0];
320                         tout1=tin[1];

322                         l2c(tout0,out);
323                         l2c(tout1,out);
```



```
324                         }
325                 if (l != -8)
326                         {
327                         c2ln(in,tin0,tin1,l+8);
328                         tin0^=tout0;
329                         tin1^=tout1;

331                         tin[0]=tin0;
332                         tin[1]=tin1;
333                         DES_encrypt3((DES_LONG *)tin,ks1,ks2,ks3);
334                         tout0=tin[0];
335                         tout1=tin[1];

337                         l2c(tout0,out);
338                         l2c(tout1,out);
339                         }
340                 iv = &(*ivec)[0];
341                 l2c(tout0,iv);
342                 l2c(tout1,iv);
343                 }
344         else
345                 {
346                 register DES_LONG t0,t1;

348                 c2l(iv,xor0);
349                 c2l(iv,xor1);
350                 for (l-=8; l>=0; l-=8)
351                         {
352                         c2l(in,tin0);
353                         c2l(in,tin1);

355                         t0=tin0;
356                         t1=tin1;

358                         tin[0]=tin0;
359                         tin[1]=tin1;
360                         DES_decrypt3((DES_LONG *)tin,ks1,ks2,ks3);
361                         tout0=tin[0];
362                         tout1=tin[1];

364                         tout0^=xor0;
365                         tout1^=xor1;
366                         l2c(tout0,out);
367                         l2c(tout1,out);
368                         xor0=t0;
369                         xor1=t1;
370                         }
371                 if (l != -8)
372                         {
373                         c2l(in,tin0);
374                         c2l(in,tin1);

376                         t0=tin0;
377                         t1=tin1;

379                         tin[0]=tin0;
380                         tin[1]=tin1;
381                         DES_decrypt3((DES_LONG *)tin,ks1,ks2,ks3);
382                         tout0=tin[0];
383                         tout1=tin[1];

385                         tout0^=xor0;
386                         tout1^=xor1;
387                         l2cn(tout0,tout1,out,l+8);
388                         xor0=t0;
389                         xor1=t1;
```

```
390                         }

392                 iv = &(*ivec)[0];
393                 l2c(xor0,iv);
394                 l2c(xor1,iv);
395                 }
396         tin0=tin1=tout0=tout1=xor0=xor1=0;
397         tin[0]=tin[1]=0;
398         }

400 #endif /* DES_DEFAULT_OPTIONS */
```

```
**********************************************************
    4678 Wed Aug 13 19:32:58 2014
new/usr/src/lib/openssl/libsunw_crypto/engine/eng_all.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/engine/eng_all.c -*- mode: C; c-file-style: "eay" -*- */
   2 /* Written by Richard Levitte <richard@levitte.org> for the OpenSSL
   3  * project 2000.
   4  */
   5 /* ====================================================================
   6  * Copyright (c) 2000-2001 The OpenSSL Project.  All rights reserved.
   7  *
   8  * Redistribution and use in source and binary forms, with or without
   9  * modification, are permitted provided that the following conditions
  10  * are met:
  11  *
  12  * 1. Redistributions of source code must retain the above copyright
  13  *    notice, this list of conditions and the following disclaimer.
  14  *
  15  * 2. Redistributions in binary form must reproduce the above copyright
  16  *    notice, this list of conditions and the following disclaimer in
  17  *    the documentation and/or other materials provided with the
  18  *    distribution.
  19  *
  20  * 3. All advertising materials mentioning features or use of this
  21  *    software must display the following acknowledgment:
  22  *    "This product includes software developed by the OpenSSL Project
  23  *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
  24  *
  25  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  26  *    endorse or promote products derived from this software without
  27  *    prior written permission. For written permission, please contact
  28  *    licensing@OpenSSL.org.
  29  *
  30  * 5. Products derived from this software may not be called "OpenSSL"
  31  *    nor may "OpenSSL" appear in their names without prior written
  32  *    permission of the OpenSSL Project.
  33  *
  34  * 6. Redistributions of any form whatsoever must retain the following
  35  *    acknowledgment:
  36  *    "This product includes software developed by the OpenSSL Project
  37  *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
  38  *
  39  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ''AS IS'' AND ANY
  40  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  41  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  42  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  43  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  44  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  45  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  46  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  47  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  48  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  49  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  50  * OF THE POSSIBILITY OF SUCH DAMAGE.
  51  * ====================================================================
  52  *
  53  * This product includes cryptographic software written by Eric Young
  54  * (eay@cryptsoft.com).  This product includes software written by Tim
  55  * Hudson (tjh@cryptsoft.com).
  56  *
  57  */

  59 #include "cryptlib.h"
  60 #include "eng_int.h"
```



```
  62 void ENGINE_load_builtin_engines(void)
  63         {
  64         /* Some ENGINEs need this */
  65         OPENSSL_cpuid_setup();
  66 #if 0
  67         /* There's no longer any need for an "openssl" ENGINE unless, one day,
  68          * it is the *only* way for standard builtin implementations to be be
  69          * accessed (ie. it would be possible to statically link binaries with
  70          * *no* builtin implementations). */
  71         ENGINE_load_openssl();
  72 #endif
  73 #if !defined(OPENSSL_NO_HW) && (defined(__OpenBSD__) || defined(__FreeBSD__) ||
  74         ENGINE_load_cryptodev();
  75 #endif
  76 #ifndef OPENSSL_NO_RSAX
  77         ENGINE_load_rsax();
  78 #endif
  79 #ifndef OPENSSL_NO_RDRAND
  80         ENGINE_load_rdrand();
  81 #endif
  82         ENGINE_load_dynamic();
  83 #ifndef OPENSSL_NO_STATIC_ENGINE
  84 #ifndef OPENSSL_NO_HW
  85 #ifndef OPENSSL_NO_HW_4758_CCA
  86         ENGINE_load_4758cca();
  87 #endif
  88 #ifndef OPENSSL_NO_HW_AEP
  89         ENGINE_load_aep();
  90 #endif
  91 #ifndef OPENSSL_NO_HW_ATALLA
  92         ENGINE_load_atalla();
  93 #endif
  94 #ifndef OPENSSL_NO_HW_CSWIFT
  95         ENGINE_load_cswift();
  96 #endif
  97 #ifndef OPENSSL_NO_HW_NCIPHER
  98         ENGINE_load_chil();
  99 #endif
 100 #ifndef OPENSSL_NO_HW_NURON
 101         ENGINE_load_nuron();
 102 #endif
 103 #ifndef OPENSSL_NO_HW_SUREWARE
 104         ENGINE_load_sureware();
 105 #endif
 106 #ifndef OPENSSL_NO_HW_UBSEC
 107         ENGINE_load_ubsec();
 108 #endif
 109 #ifndef OPENSSL_NO_HW_PADLOCK
 110         ENGINE_load_padlock();
 111 #endif
 112 #ifndef OPENSSL_NO_HW_PKCS11
 113         ENGINE_load_pk11();
 114 #endif
 115 #endif /* ! codereview */
 116 #endif
 117 #ifndef OPENSSL_NO_GOST
 118         ENGINE_load_gost();
 119 #endif
 120 #ifndef OPENSSL_NO_GMP
 121         ENGINE_load_gmp();
 122 #endif
 123 #if defined(OPENSSL_SYS_WIN32) && !defined(OPENSSL_NO_CAPIENG)
 124         ENGINE_load_capi();
 125 #endif
 126 #endif
 127         ENGINE_register_all_complete();
```

```
128         }

130 #if defined(__OpenBSD__) || defined(__FreeBSD__) || defined(HAVE_CRYPTODEV)
131 void ENGINE_setup_bsd_cryptodev(void) {
132         static int bsd_cryptodev_default_loaded = 0;
133         if (!bsd_cryptodev_default_loaded) {
134                 ENGINE_load_cryptodev();
135                 ENGINE_register_all_complete();
136         }
137         bsd_cryptodev_default_loaded=1;
138 }
139 #endif
```

```
**********************************************************
   92357 Wed Aug 13 19:32:58 2014
new/usr/src/lib/openssl/libsunw_crypto/engine/hw_pk11.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* crypto/engine/hw_pk11.c */
/*
 * This product includes software developed by the OpenSSL Project for
 * use in the OpenSSL Toolkit (http://www.openssl.org/).
 *
 * This project also referenced hw_pkcs11-0.9.7b.patch written by
 * Afchine Madjlessi.
 */
/*
 * ====================================================================
 * Copyright (c) 2000-2001 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
```



```
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

#include <openssl/opensslconf.h>
#include <openssl/e_os2.h>
#include <openssl/crypto.h>
#include <openssl/engine.h>
#include <openssl/dso.h>
#include <openssl/err.h>
#include <openssl/bn.h>
#include <openssl/md5.h>
#include <openssl/pem.h>
#ifndef OPENSSL_NO_RSA
#include <openssl/rsa.h>
#endif
#ifndef OPENSSL_NO_DSA
#include <openssl/dsa.h>
#endif
#ifndef OPENSSL_NO_DH
#include <openssl/dh.h>
#endif
#include <openssl/rand.h>
#include <openssl/objects.h>
#include <openssl/x509.h>
#include <openssl/aes.h>
#include <cryptlib.h>
#include <dlfcn.h>
#include <pthread.h>

#ifndef OPENSSL_NO_HW
#ifndef OPENSSL_NO_HW_PK11

/* label for debug messages printed on stderr */
#define PK11_DBG        "PKCS#11 ENGINE DEBUG"
/* prints a lot of debug messages on stderr about slot selection process */
#undef  DEBUG_SLOT_SELECTION
/*
 * Solaris specific code. See comment at check_hw_mechanisms() for more
 * information.
 */
#if defined (__SVR4) && defined (__sun)
#define SOLARIS_HW_SLOT_SELECTION
#endif

/*
 * AES counter mode is not supported in the OpenSSL EVP API yet and neither
 * there are official OIDs for mechanisms based on this mode. With our changes,
 * an application can define its own EVP calls for AES counter mode and then
 * it can make use of hardware acceleration through this engine. However, it's
 * better if we keep AES CTR support code under ifdef's.
 */
#define SOLARIS_AES_CTR

#include "cryptoki.h"
#include "pkcs11.h"
#include "hw_pk11_err.c"
```

```
#ifdef  SOLARIS_HW_SLOT_SELECTION
/*
 * Tables for symmetric ciphers and digest mechs found in the pkcs11_kernel
 * library. See comment at check_hw_mechanisms() for more information.
 */
int *hw_cnids;
int *hw_dnids;
#endif  /* SOLARIS_HW_SLOT_SELECTION */

/* PKCS#11 session caches and their locks for all operation types */
static PK11_CACHE session_cache[OP_MAX];

/*
 * As stated in v2.20, 11.7 Object Management Function, in section for
 * C_FindObjectsInit(), at most one search operation may be active at a given
 * time in a given session. Therefore, C_Find{,Init,Final}Objects() should be
 * grouped together to form one atomic search operation. This is already
 * ensured by the property of unique PKCS#11 session handle used for each
 * PK11_SESSION object.
 *
 * This is however not the biggest concern - maintaining consistency of the
 * underlying object store is more important. The same section of the spec also
 * says that one thread can be in the middle of a search operation while another
 * thread destroys the object matching the search template which would result in
 * invalid handle returned from the search operation.
 *
 * Hence, the following locks are used for both protection of the object stores.
 * They are also used for active list protection.
 */
pthread_mutex_t *find_lock[OP_MAX] = { NULL };

/*
 * lists of asymmetric key handles which are active (referenced by at least one
 * PK11_SESSION structure, either held by a thread or present in free_session
 * list) for given algorithm type
 */
PK11_active *active_list[OP_MAX] = { NULL };

/*
 * Create all secret key objects in a global session so that they are available
 * to use for other sessions. These other sessions may be opened or closed
 * without losing the secret key objects.
 */
static CK_SESSION_HANDLE        global_session = CK_INVALID_HANDLE;

/* ENGINE level stuff */
static int pk11_init(ENGINE *e);
static int pk11_library_init(ENGINE *e);
static int pk11_finish(ENGINE *e);
static int pk11_ctrl(ENGINE *e, int cmd, long i, void *p, void (*f)());
static int pk11_destroy(ENGINE *e);

/* RAND stuff */
static void pk11_rand_seed(const void *buf, int num);
static void pk11_rand_add(const void *buf, int num, double add_entropy);
static void pk11_rand_cleanup(void);
static int pk11_rand_bytes(unsigned char *buf, int num);
static int pk11_rand_status(void);

/* These functions are also used in other files */
PK11_SESSION *pk11_get_session(PK11_OPTYPE optype);
void pk11_return_session(PK11_SESSION *sp, PK11_OPTYPE optype);

/* active list manipulation functions used in this file */
extern int pk11_active_delete(CK_OBJECT_HANDLE h, PK11_OPTYPE type);
```



```
extern void pk11_free_active_list(PK11_OPTYPE type);

#ifndef OPENSSL_NO_RSA
int pk11_destroy_rsa_key_objects(PK11_SESSION *session);
int pk11_destroy_rsa_object_pub(PK11_SESSION *sp, CK_BBOOL uselock);
int pk11_destroy_rsa_object_priv(PK11_SESSION *sp, CK_BBOOL uselock);
#endif
#ifndef OPENSSL_NO_DSA
int pk11_destroy_dsa_key_objects(PK11_SESSION *session);
int pk11_destroy_dsa_object_pub(PK11_SESSION *sp, CK_BBOOL uselock);
int pk11_destroy_dsa_object_priv(PK11_SESSION *sp, CK_BBOOL uselock);
#endif
#ifndef OPENSSL_NO_DH
int pk11_destroy_dh_key_objects(PK11_SESSION *session);
int pk11_destroy_dh_object(PK11_SESSION *session, CK_BBOOL uselock);
#endif

/* Local helper functions */
static int pk11_free_all_sessions(void);
static int pk11_free_session_list(PK11_OPTYPE optype);
static int pk11_setup_session(PK11_SESSION *sp, PK11_OPTYPE optype);
static int pk11_destroy_cipher_key_objects(PK11_SESSION *session);
static int pk11_destroy_object(CK_SESSION_HANDLE session,
        CK_OBJECT_HANDLE oh);
static const char *get_PK11_LIBNAME(void);
static void free_PK11_LIBNAME(void);
static long set_PK11_LIBNAME(const char *name);

/* Symmetric cipher and digest support functions */
static int cipher_nid_to_pk11(int nid);
#ifdef  SOLARIS_AES_CTR
static int pk11_add_aes_ctr_NIDs(void);
#endif  /* SOLARIS_AES_CTR */
static int pk11_usable_ciphers(const int **nids);
static int pk11_usable_digests(const int **nids);
static int pk11_cipher_init(EVP_CIPHER_CTX *ctx, const unsigned char *key,
        const unsigned char *iv, int enc);
static int pk11_cipher_final(PK11_SESSION *sp);
static int pk11_cipher_do_cipher(EVP_CIPHER_CTX *ctx, unsigned char *out,
        const unsigned char *in, size_t inl);
static int pk11_cipher_cleanup(EVP_CIPHER_CTX *ctx);
static int pk11_engine_ciphers(ENGINE *e, const EVP_CIPHER **cipher,
        const int **nids, int nid);
static int pk11_engine_digests(ENGINE *e, const EVP_MD **digest,
        const int **nids, int nid);
static CK_OBJECT_HANDLE pk11_get_cipher_key(EVP_CIPHER_CTX *ctx,
        const unsigned char *key, CK_KEY_TYPE key_type, PK11_SESSION *sp);
static int check_new_cipher_key(PK11_SESSION *sp, const unsigned char *key,
        int key_len);
static int md_nid_to_pk11(int nid);
static int pk11_digest_init(EVP_MD_CTX *ctx);
static int pk11_digest_update(EVP_MD_CTX *ctx, const void *data,
        size_t count);
static int pk11_digest_final(EVP_MD_CTX *ctx, unsigned char *md);
static int pk11_digest_copy(EVP_MD_CTX *to, const EVP_MD_CTX *from);
static int pk11_digest_cleanup(EVP_MD_CTX *ctx);

static int pk11_choose_slots(int *any_slot_found);
static void pk11_find_symmetric_ciphers(CK_FUNCTION_LIST_PTR pflist,
    CK_SLOT_ID current_slot, int *current_slot_n_cipher,
    int *local_cipher_nids);
static void pk11_find_digests(CK_FUNCTION_LIST_PTR pflist,
    CK_SLOT_ID current_slot, int *current_slot_n_digest,
    int *local_digest_nids);
static void pk11_get_symmetric_cipher(CK_FUNCTION_LIST_PTR, int slot_id,
    CK_MECHANISM_TYPE mech, int *current_slot_n_cipher, int *local_cipher_nids,
```

```
260     int id);
261 static void pk11_get_digest(CK_FUNCTION_LIST_PTR pflist, int slot_id,
262     CK_MECHANISM_TYPE mech, int *current_slot_n_digest, int *local_digest_nids,
263     int id);

265 static int pk11_init_all_locks(void);
266 static void pk11_free_all_locks(void);

268 #ifdef  SOLARIS_HW_SLOT_SELECTION
269 static int check_hw_mechanisms(void);
270 static int nid_in_table(int nid, int *nid_table);
271 #endif  /* SOLARIS_HW_SLOT_SELECTION */

273 /* Index for the supported ciphers */
274 enum pk11_cipher_id {
275         PK11_DES_CBC,
276         PK11_DES3_CBC,
277         PK11_DES_ECB,
278         PK11_DES3_ECB,
279         PK11_RC4,
280         PK11_AES_128_CBC,
281         PK11_AES_192_CBC,
282         PK11_AES_256_CBC,
283         PK11_AES_128_ECB,
284         PK11_AES_192_ECB,
285         PK11_AES_256_ECB,
286         PK11_BLOWFISH_CBC,
287 #ifdef  SOLARIS_AES_CTR
288         PK11_AES_128_CTR,
289         PK11_AES_192_CTR,
290         PK11_AES_256_CTR,
291 #endif  /* SOLARIS_AES_CTR */
292         PK11_CIPHER_MAX
293 };

295 /* Index for the supported digests */
296 enum pk11_digest_id {
297         PK11_MD5,
298         PK11_SHA1,
299         PK11_SHA224,
300         PK11_SHA256,
301         PK11_SHA384,
302         PK11_SHA512,
303         PK11_DIGEST_MAX
304 };

306 #define TRY_OBJ_DESTROY(sess_hdl, obj_hdl, retval, uselock, alg_type)   \
307         {                                                               \
308         if (uselock)                                                    \
309                 LOCK_OBJSTORE(alg_type);                                \
310         if (pk11_active_delete(obj_hdl, alg_type) == 1)                 \
311                 {                                                       \
312                 retval = pk11_destroy_object(sess_hdl, obj_hdl);        \
313                 }                                                       \
314         if (uselock)                                                    \
315                 UNLOCK_OBJSTORE(alg_type);                              \
316         }

318 static int cipher_nids[PK11_CIPHER_MAX];
319 static int digest_nids[PK11_DIGEST_MAX];
320 static int cipher_count         = 0;
321 static int digest_count         = 0;
322 static CK_BBOOL pk11_have_rsa   = CK_FALSE;
323 static CK_BBOOL pk11_have_dsa   = CK_FALSE;
324 static CK_BBOOL pk11_have_dh    = CK_FALSE;
325 static CK_BBOOL pk11_have_random = CK_FALSE;
```



```
327 typedef struct PK11_CIPHER_st
328         {
329         enum pk11_cipher_id     id;
330         int                     nid;
331         int                     iv_len;
332         int                     min_key_len;
333         int                     max_key_len;
334         CK_KEY_TYPE             key_type;
335         CK_MECHANISM_TYPE       mech_type;
336         } PK11_CIPHER;

338 static PK11_CIPHER ciphers[] =
339         {
340         { PK11_DES_CBC,         NID_des_cbc,            8,       8,   8,
341                 CKK_DES,        CKM_DES_CBC, },
342         { PK11_DES3_CBC,        NID_des_ede3_cbc,       8,      24,  24,
343                 CKK_DES3,       CKM_DES3_CBC, },
344         { PK11_DES_ECB,         NID_des_ecb,            0,       8,   8,
345                 CKK_DES,        CKM_DES_ECB, },
346         { PK11_DES3_ECB,        NID_des_ede3_ecb,       0,      24,  24,
347                 CKK_DES3,       CKM_DES3_ECB, },
348         { PK11_RC4,             NID_rc4,                0,      16, 256,
349                 CKK_RC4,        CKM_RC4, },
350         { PK11_AES_128_CBC,     NID_aes_128_cbc,        16,     16,  16,
351                 CKK_AES,        CKM_AES_CBC, },
352         { PK11_AES_192_CBC,     NID_aes_192_cbc,        16,     24,  24,
353                 CKK_AES,        CKM_AES_CBC, },
354         { PK11_AES_256_CBC,     NID_aes_256_cbc,        16,     32,  32,
355                 CKK_AES,        CKM_AES_CBC, },
356         { PK11_AES_128_ECB,     NID_aes_128_ecb,        0,      16,  16,
357                 CKK_AES,        CKM_AES_ECB, },
358         { PK11_AES_192_ECB,     NID_aes_192_ecb,        0,      24,  24,
359                 CKK_AES,        CKM_AES_ECB, },
360         { PK11_AES_256_ECB,     NID_aes_256_ecb,        0,      32,  32,
361                 CKK_AES,        CKM_AES_ECB, },
362         { PK11_BLOWFISH_CBC,    NID_bf_cbc,             8,      16,  16,
363                 CKK_BLOWFISH,   CKM_BLOWFISH_CBC, },
364 #ifdef  SOLARIS_AES_CTR
365         /* we don't know the correct NIDs until the engine is initialized */
366         { PK11_AES_128_CTR,     NID_undef,              16,     16,  16,
367                 CKK_AES,        CKM_AES_CTR, },
368         { PK11_AES_192_CTR,     NID_undef,              16,     24,  24,
369                 CKK_AES,        CKM_AES_CTR, },
370         { PK11_AES_256_CTR,     NID_undef,              16,     32,  32,
371                 CKK_AES,        CKM_AES_CTR, },
372 #endif  /* SOLARIS_AES_CTR */
373         };

375 typedef struct PK11_DIGEST_st
376         {
377         enum pk11_digest_id     id;
378         int                     nid;
379         CK_MECHANISM_TYPE       mech_type;
380         } PK11_DIGEST;

382 static PK11_DIGEST digests[] =
383         {
384         {PK11_MD5,      NID_md5,        CKM_MD5, },
385         {PK11_SHA1,     NID_sha1,       CKM_SHA_1, },
386         {PK11_SHA224,   NID_sha224,     CKM_SHA224, },
387         {PK11_SHA256,   NID_sha256,     CKM_SHA256, },
388         {PK11_SHA384,   NID_sha384,     CKM_SHA384, },
389         {PK11_SHA512,   NID_sha512,     CKM_SHA512, },
390         {0,             NID_undef,      0xFFFF, },
391         };
```

```
393 /*
394  * Structure to be used for the cipher_data/md_data in
395  * EVP_CIPHER_CTX/EVP_MD_CTX structures in order to use the same pk11
396  * session in multiple cipher_update calls
397  */
398 typedef struct PK11_CIPHER_STATE_st
399         {
400         PK11_SESSION    *sp;
401         } PK11_CIPHER_STATE;


404 /*
405  * libcrypto EVP stuff - this is how we get wired to EVP so the engine gets
406  * called when libcrypto requests a cipher NID.
407  *
408  * Note how the PK11_CIPHER_STATE is used here.
409  */

411 /* DES CBC EVP */
412 static const EVP_CIPHER pk11_des_cbc =
413         {
414         NID_des_cbc,
415         8, 8, 8,
416         EVP_CIPH_CBC_MODE,
417         pk11_cipher_init,
418         pk11_cipher_do_cipher,
419         pk11_cipher_cleanup,
420         sizeof (PK11_CIPHER_STATE),
421         EVP_CIPHER_set_asn1_iv,
422         EVP_CIPHER_get_asn1_iv,
423         NULL
424         };

426 /* 3DES CBC EVP */
427 static const EVP_CIPHER pk11_3des_cbc =
428         {
429         NID_des_ede3_cbc,
430         8, 24, 8,
431         EVP_CIPH_CBC_MODE,
432         pk11_cipher_init,
433         pk11_cipher_do_cipher,
434         pk11_cipher_cleanup,
435         sizeof (PK11_CIPHER_STATE),
436         EVP_CIPHER_set_asn1_iv,
437         EVP_CIPHER_get_asn1_iv,
438         NULL
439         };

441 /*
442  * ECB modes don’t use an Initial Vector so that’s why set_asn1_parameters and
443  * get_asn1_parameters fields are set to NULL.
444  */
445 static const EVP_CIPHER pk11_des_ecb =
446         {
447         NID_des_ecb,
448         8, 8, 8,
449         EVP_CIPH_ECB_MODE,
450         pk11_cipher_init,
451         pk11_cipher_do_cipher,
452         pk11_cipher_cleanup,
453         sizeof (PK11_CIPHER_STATE),
454         NULL,
455         NULL,
456         NULL
457         };
```



```
459 static const EVP_CIPHER pk11_3des_ecb =
460         {
461         NID_des_ede3_ecb,
462         8, 24, 8,
463         EVP_CIPH_ECB_MODE,
464         pk11_cipher_init,
465         pk11_cipher_do_cipher,
466         pk11_cipher_cleanup,
467         sizeof (PK11_CIPHER_STATE),
468         NULL,
469         NULL,
470         NULL
471         };


474 static const EVP_CIPHER pk11_aes_128_cbc =
475         {
476         NID_aes_128_cbc,
477         16, 16, 16,
478         EVP_CIPH_CBC_MODE,
479         pk11_cipher_init,
480         pk11_cipher_do_cipher,
481         pk11_cipher_cleanup,
482         sizeof (PK11_CIPHER_STATE),
483         EVP_CIPHER_set_asn1_iv,
484         EVP_CIPHER_get_asn1_iv,
485         NULL
486         };

488 static const EVP_CIPHER pk11_aes_192_cbc =
489         {
490         NID_aes_192_cbc,
491         16, 24, 16,
492         EVP_CIPH_CBC_MODE,
493         pk11_cipher_init,
494         pk11_cipher_do_cipher,
495         pk11_cipher_cleanup,
496         sizeof (PK11_CIPHER_STATE),
497         EVP_CIPHER_set_asn1_iv,
498         EVP_CIPHER_get_asn1_iv,
499         NULL
500         };

502 static const EVP_CIPHER pk11_aes_256_cbc =
503         {
504         NID_aes_256_cbc,
505         16, 32, 16,
506         EVP_CIPH_CBC_MODE,
507         pk11_cipher_init,
508         pk11_cipher_do_cipher,
509         pk11_cipher_cleanup,
510         sizeof (PK11_CIPHER_STATE),
511         EVP_CIPHER_set_asn1_iv,
512         EVP_CIPHER_get_asn1_iv,
513         NULL
514         };

516 /*
517  * ECB modes don’t use IV so that’s why set_asn1_parameters and
518  * get_asn1_parameters are set to NULL.
519  */
520 static const EVP_CIPHER pk11_aes_128_ecb =
521         {
522         NID_aes_128_ecb,
523         16, 16, 0,
```

```
524         EVP_CIPH_ECB_MODE,
525         pk11_cipher_init,
526         pk11_cipher_do_cipher,
527         pk11_cipher_cleanup,
528         sizeof (PK11_CIPHER_STATE),
529         NULL,
530         NULL,
531         NULL
532         };

534 static const EVP_CIPHER pk11_aes_192_ecb =
535         {
536         NID_aes_192_ecb,
537         16, 24, 0,
538         EVP_CIPH_ECB_MODE,
539         pk11_cipher_init,
540         pk11_cipher_do_cipher,
541         pk11_cipher_cleanup,
542         sizeof (PK11_CIPHER_STATE),
543         NULL,
544         NULL,
545         NULL
546         };

548 static const EVP_CIPHER pk11_aes_256_ecb =
549         {
550         NID_aes_256_ecb,
551         16, 32, 0,
552         EVP_CIPH_ECB_MODE,
553         pk11_cipher_init,
554         pk11_cipher_do_cipher,
555         pk11_cipher_cleanup,
556         sizeof (PK11_CIPHER_STATE),
557         NULL,
558         NULL,
559         NULL
560         };

562 #ifdef  SOLARIS_AES_CTR
563 /*
564  * NID_undef's will be changed to the AES counter mode NIDs as soon they are
565  * created in pk11_library_init(). Note that the need to change these structures
566  * is the reason why we don't define them with the const keyword.
567  */
568 static EVP_CIPHER pk11_aes_128_ctr =
569         {
570         NID_undef,
571         16, 16, 16,
572         EVP_CIPH_CBC_MODE,
573         pk11_cipher_init,
574         pk11_cipher_do_cipher,
575         pk11_cipher_cleanup,
576         sizeof (PK11_CIPHER_STATE),
577         EVP_CIPHER_set_asn1_iv,
578         EVP_CIPHER_get_asn1_iv,
579         NULL
580         };

582 static EVP_CIPHER pk11_aes_192_ctr =
583         {
584         NID_undef,
585         16, 24, 16,
586         EVP_CIPH_CBC_MODE,
587         pk11_cipher_init,
588         pk11_cipher_do_cipher,
589         pk11_cipher_cleanup,
```



```
590         sizeof (PK11_CIPHER_STATE),
591         EVP_CIPHER_set_asn1_iv,
592         EVP_CIPHER_get_asn1_iv,
593         NULL
594         };

596 static EVP_CIPHER pk11_aes_256_ctr =
597         {
598         NID_undef,
599         16, 32, 16,
600         EVP_CIPH_CBC_MODE,
601         pk11_cipher_init,
602         pk11_cipher_do_cipher,
603         pk11_cipher_cleanup,
604         sizeof (PK11_CIPHER_STATE),
605         EVP_CIPHER_set_asn1_iv,
606         EVP_CIPHER_get_asn1_iv,
607         NULL
608         };
609 #endif  /* SOLARIS_AES_CTR */

611 static const EVP_CIPHER pk11_bf_cbc =
612         {
613         NID_bf_cbc,
614         8, 16, 8,
615         EVP_CIPH_VARIABLE_LENGTH,
616         pk11_cipher_init,
617         pk11_cipher_do_cipher,
618         pk11_cipher_cleanup,
619         sizeof (PK11_CIPHER_STATE),
620         EVP_CIPHER_set_asn1_iv,
621         EVP_CIPHER_get_asn1_iv,
622         NULL
623         };

625 static const EVP_CIPHER pk11_rc4 =
626         {
627         NID_rc4,
628         1, 16, 0,
629         EVP_CIPH_VARIABLE_LENGTH,
630         pk11_cipher_init,
631         pk11_cipher_do_cipher,
632         pk11_cipher_cleanup,
633         sizeof (PK11_CIPHER_STATE),
634         NULL,
635         NULL,
636         NULL
637         };

639 static const EVP_MD pk11_md5 =
640         {
641         NID_md5,
642         NID_md5WithRSAEncryption,
643         MD5_DIGEST_LENGTH,
644         0,
645         pk11_digest_init,
646         pk11_digest_update,
647         pk11_digest_final,
648         pk11_digest_copy,
649         pk11_digest_cleanup,
650         EVP_PKEY_RSA_method,
651         MD5_CBLOCK,
652         sizeof (PK11_CIPHER_STATE),
653         };

655 static const EVP_MD pk11_sha1 =
```

```
656         {
657         NID_sha1,
658         NID_sha1WithRSAEncryption,
659         SHA_DIGEST_LENGTH,
660         0,
661         pk11_digest_init,
662         pk11_digest_update,
663         pk11_digest_final,
664         pk11_digest_copy,
665         pk11_digest_cleanup,
666         EVP_PKEY_RSA_method,
667         SHA_CBLOCK,
668         sizeof (PK11_CIPHER_STATE),
669         };

671 static const EVP_MD pk11_sha224 =
672         {
673         NID_sha224,
674         NID_sha224WithRSAEncryption,
675         SHA224_DIGEST_LENGTH,
676         0,
677         pk11_digest_init,
678         pk11_digest_update,
679         pk11_digest_final,
680         pk11_digest_copy,
681         pk11_digest_cleanup,
682         EVP_PKEY_RSA_method,
683         /* SHA-224 uses the same cblock size as SHA-256 */
684         SHA256_CBLOCK,
685         sizeof (PK11_CIPHER_STATE),
686         };

688 static const EVP_MD pk11_sha256 =
689         {
690         NID_sha256,
691         NID_sha256WithRSAEncryption,
692         SHA256_DIGEST_LENGTH,
693         0,
694         pk11_digest_init,
695         pk11_digest_update,
696         pk11_digest_final,
697         pk11_digest_copy,
698         pk11_digest_cleanup,
699         EVP_PKEY_RSA_method,
700         SHA256_CBLOCK,
701         sizeof (PK11_CIPHER_STATE),
702         };

704 static const EVP_MD pk11_sha384 =
705         {
706         NID_sha384,
707         NID_sha384WithRSAEncryption,
708         SHA384_DIGEST_LENGTH,
709         0,
710         pk11_digest_init,
711         pk11_digest_update,
712         pk11_digest_final,
713         pk11_digest_copy,
714         pk11_digest_cleanup,
715         EVP_PKEY_RSA_method,
716         /* SHA-384 uses the same cblock size as SHA-512 */
717         SHA512_CBLOCK,
718         sizeof (PK11_CIPHER_STATE),
719         };

721 static const EVP_MD pk11_sha512 =
```



```
722         {
723         NID_sha512,
724         NID_sha512WithRSAEncryption,
725         SHA512_DIGEST_LENGTH,
726         0,
727         pk11_digest_init,
728         pk11_digest_update,
729         pk11_digest_final,
730         pk11_digest_copy,
731         pk11_digest_cleanup,
732         EVP_PKEY_RSA_method,
733         SHA512_CBLOCK,
734         sizeof (PK11_CIPHER_STATE),
735         };

737 /*
738  * Initialization function. Sets up various PKCS#11 library components.
739  * The definitions for control commands specific to this engine
740  */
741 #define PK11_CMD_SO_PATH                ENGINE_CMD_BASE
742 static const ENGINE_CMD_DEFN pk11_cmd_defns[] =
743         {
744                 {
745                 PK11_CMD_SO_PATH,
746                 "SO_PATH",
747                 "Specifies the path to the 'pkcs#11' shared library",
748                 ENGINE_CMD_FLAG_STRING
749                 },
750                 {0, NULL, NULL, 0}
751         };


754 static RAND_METHOD pk11_random =
755         {
756         pk11_rand_seed,
757         pk11_rand_bytes,
758         pk11_rand_cleanup,
759         pk11_rand_add,
760         pk11_rand_bytes,
761         pk11_rand_status
762         };


765 /* Constants used when creating the ENGINE */
766 static const char *engine_pk11_id = "pkcs11";
767 static const char *engine_pk11_name = "PKCS #11 engine support";

769 CK_FUNCTION_LIST_PTR pFuncList = NULL;
770 static const char PK11_GET_FUNCTION_LIST[] = "C_GetFunctionList";

772 /*
773  * These is the static string constant for the DSO file name and the function
774  * symbol names to bind to.
775  */
776 static const char def_PK11_LIBNAME[] = PK11_LIB_LOCATION;

778 static CK_BBOOL true = TRUE;
779 static CK_BBOOL false = FALSE;
780 static CK_SLOT_ID pubkey_SLOTID = 0;
781 static CK_SLOT_ID rand_SLOTID = 0;
782 static CK_SLOT_ID SLOTID = 0;
783 static CK_BBOOL pk11_library_initialized = FALSE;
784 static CK_BBOOL pk11_atfork_initialized = FALSE;
785 static int pk11_pid = 0;

787 static DSO *pk11_dso = NULL;
```

```
/* allocate and initialize all locks used by the engine itself */
static int pk11_init_all_locks(void)
        {
        int type;

#ifndef OPENSSL_NO_RSA
        find_lock[OP_RSA] = OPENSSL_malloc(sizeof (pthread_mutex_t));
        if (find_lock[OP_RSA] == NULL)
                goto malloc_err;
        (void) pthread_mutex_init(find_lock[OP_RSA], NULL);
#endif /* OPENSSL_NO_RSA */

#ifndef OPENSSL_NO_DSA
        find_lock[OP_DSA] = OPENSSL_malloc(sizeof (pthread_mutex_t));
        if (find_lock[OP_DSA] == NULL)
                goto malloc_err;
        (void) pthread_mutex_init(find_lock[OP_DSA], NULL);
#endif /* OPENSSL_NO_DSA */

#ifndef OPENSSL_NO_DH
        find_lock[OP_DH] = OPENSSL_malloc(sizeof (pthread_mutex_t));
        if (find_lock[OP_DH] == NULL)
                goto malloc_err;
        (void) pthread_mutex_init(find_lock[OP_DH], NULL);
#endif /* OPENSSL_NO_DH */

        for (type = 0; type < OP_MAX; type++)
                {
                session_cache[type].lock =
                    OPENSSL_malloc(sizeof (pthread_mutex_t));
                if (session_cache[type].lock == NULL)
                        goto malloc_err;
                (void) pthread_mutex_init(session_cache[type].lock, NULL);
                }

        return (1);

malloc_err:
        pk11_free_all_locks();
        PK11err(PK11_F_INIT_ALL_LOCKS, PK11_R_MALLOC_FAILURE);
        return (0);
        }

static void pk11_free_all_locks(void)
        {
        int type;

#ifndef OPENSSL_NO_RSA
        if (find_lock[OP_RSA] != NULL)
                {
                (void) pthread_mutex_destroy(find_lock[OP_RSA]);
                OPENSSL_free(find_lock[OP_RSA]);
                find_lock[OP_RSA] = NULL;
                }
#endif /* OPENSSL_NO_RSA */
#ifndef OPENSSL_NO_DSA
        if (find_lock[OP_DSA] != NULL)
                {
                (void) pthread_mutex_destroy(find_lock[OP_DSA]);
                OPENSSL_free(find_lock[OP_DSA]);
                find_lock[OP_DSA] = NULL;
                }
#endif /* OPENSSL_NO_DSA */
#ifndef OPENSSL_NO_DH
        if (find_lock[OP_DH] != NULL)
```



```
                {
                (void) pthread_mutex_destroy(find_lock[OP_DH]);
                OPENSSL_free(find_lock[OP_DH]);
                find_lock[OP_DH] = NULL;
                }
#endif /* OPENSSL_NO_DH */

        for (type = 0; type < OP_MAX; type++)
                {
                if (session_cache[type].lock != NULL)
                        {
                        (void) pthread_mutex_destroy(session_cache[type].lock);
                        OPENSSL_free(session_cache[type].lock);
                        session_cache[type].lock = NULL;
                        }
                }
        }

/*
 * This internal function is used by ENGINE_pk11() and "dynamic" ENGINE support.
 */
static int bind_pk11(ENGINE *e)
        {
#ifndef OPENSSL_NO_RSA
        const RSA_METHOD *rsa = NULL;
        RSA_METHOD *pk11_rsa = PK11_RSA();
#endif  /* OPENSSL_NO_RSA */
        if (!pk11_library_initialized)
                if (!pk11_library_init(e))
                        return (0);

        if (!ENGINE_set_id(e, engine_pk11_id) ||
            !ENGINE_set_name(e, engine_pk11_name) ||
            !ENGINE_set_ciphers(e, pk11_engine_ciphers) ||
            !ENGINE_set_digests(e, pk11_engine_digests))
                return (0);
#ifndef OPENSSL_NO_RSA
        if (pk11_have_rsa == CK_TRUE)
                {
                if (!ENGINE_set_RSA(e, PK11_RSA()) ||
                    !ENGINE_set_load_privkey_function(e, pk11_load_privkey) ||
                    !ENGINE_set_load_pubkey_function(e, pk11_load_pubkey))
                        return (0);
#ifdef  DEBUG_SLOT_SELECTION
                fprintf(stderr, "%s: registered RSA\n", PK11_DBG);
#endif  /* DEBUG_SLOT_SELECTION */
                }
#endif  /* OPENSSL_NO_RSA */
#ifndef OPENSSL_NO_DSA
        if (pk11_have_dsa == CK_TRUE)
                {
                if (!ENGINE_set_DSA(e, PK11_DSA()))
                        return (0);
#ifdef  DEBUG_SLOT_SELECTION
                fprintf(stderr, "%s: registered DSA\n", PK11_DBG);
#endif  /* DEBUG_SLOT_SELECTION */
                }
#endif  /* OPENSSL_NO_DSA */
#ifndef OPENSSL_NO_DH
        if (pk11_have_dh == CK_TRUE)
                {
                if (!ENGINE_set_DH(e, PK11_DH()))
                        return (0);
#ifdef  DEBUG_SLOT_SELECTION
                fprintf(stderr, "%s: registered DH\n", PK11_DBG);
#endif  /* DEBUG_SLOT_SELECTION */
```

```
920                 }
921 #endif  /* OPENSSL_NO_DH */
922         if (pk11_have_random)
923                 {
924                 if (!ENGINE_set_RAND(e, &pk11_random))
925                         return (0);
926 #ifdef  DEBUG_SLOT_SELECTION
927                 fprintf(stderr, "%s: registered random\n", PK11_DBG);
928 #endif  /* DEBUG_SLOT_SELECTION */
929                 }
930         if (!ENGINE_set_init_function(e, pk11_init) ||
931             !ENGINE_set_destroy_function(e, pk11_destroy) ||
932             !ENGINE_set_finish_function(e, pk11_finish) ||
933             !ENGINE_set_ctrl_function(e, pk11_ctrl) ||
934             !ENGINE_set_cmd_defns(e, pk11_cmd_defns))
935                 return (0);

937 /*
938  * Apache calls OpenSSL function RSA_blinding_on() once during startup
939  * which in turn calls bn_mod_exp. Since we do not implement bn_mod_exp
940  * here, we wire it back to the OpenSSL software implementation.
941  * Since it is used only once, performance is not a concern.
942  */
943 #ifndef OPENSSL_NO_RSA
944         rsa = RSA_PKCS1_SSLeay();
945         pk11_rsa->rsa_mod_exp = rsa->rsa_mod_exp;
946         pk11_rsa->bn_mod_exp = rsa->bn_mod_exp;
947 #endif  /* OPENSSL_NO_RSA */

949         /* Ensure the pk11 error handling is set up */
950         ERR_load_pk11_strings();

952         return (1);
953         }

955 /* Dynamic engine support is disabled at a higher level for Solaris */
956 #ifdef  ENGINE_DYNAMIC_SUPPORT
957 static int bind_helper(ENGINE *e, const char *id)
958         {
959         if (id && (strcmp(id, engine_pk11_id) != 0))
960                 return (0);

962         if (!bind_pk11(e))
963                 return (0);

965         return (1);
966         }

968 IMPLEMENT_DYNAMIC_CHECK_FN()
969 IMPLEMENT_DYNAMIC_BIND_FN(bind_helper)

971 #else
972 static ENGINE *engine_pk11(void)
973         {
974         ENGINE *ret = ENGINE_new();

976         if (!ret)
977                 return (NULL);

979         if (!bind_pk11(ret))
980                 {
981                 ENGINE_free(ret);
982                 return (NULL);
983                 }

985         return (ret);
```



```
986         }

988 void
989 ENGINE_load_pk11(void)
990         {
991         ENGINE *e_pk11 = NULL;

993         /*
994          * Do not use dynamic PKCS#11 library on Solaris due to
995          * security reasons. We will link it in statically.
996          */
997         /* Attempt to load PKCS#11 library */
998         if (!pk11_dso)
999                 pk11_dso = DSO_load(NULL, get_PK11_LIBNAME(), NULL, 0);

1001        if (pk11_dso == NULL)
1002                {
1003                PK11err(PK11_F_LOAD, PK11_R_DSO_FAILURE);
1004                return;
1005                }

1007        e_pk11 = engine_pk11();
1008        if (!e_pk11)
1009                {
1010                DSO_free(pk11_dso);
1011                pk11_dso = NULL;
1012                return;
1013                }

1015        /*
1016         * At this point, the pk11 shared library is either dynamically
1017         * loaded or statically linked in. So, initialize the pk11
1018         * library before calling ENGINE_set_default since the latter
1019         * needs cipher and digest algorithm information
1020         */
1021        if (!pk11_library_init(e_pk11))
1022                {
1023                DSO_free(pk11_dso);
1024                pk11_dso = NULL;
1025                ENGINE_free(e_pk11);
1026                return;
1027                }

1029        ENGINE_add(e_pk11);

1031        ENGINE_free(e_pk11);
1032        ERR_clear_error();
1033        }
1034 #endif  /* ENGINE_DYNAMIC_SUPPORT */

1036 /*
1037  * These are the static string constants for the DSO file name and
1038  * the function symbol names to bind to.
1039  */
1040 static const char *PK11_LIBNAME = NULL;

1042 static const char *get_PK11_LIBNAME(void)
1043         {
1044         if (PK11_LIBNAME)
1045                 return (PK11_LIBNAME);

1047         return (def_PK11_LIBNAME);
1048         }

1050 static void free_PK11_LIBNAME(void)
1051         {
```

```
1052         if (PK11_LIBNAME)
1053                 OPENSSL_free((void*)PK11_LIBNAME);
1054
1055         PK11_LIBNAME = NULL;
1056         }
1057
1058 static long set_PK11_LIBNAME(const char *name)
1059         {
1060         free_PK11_LIBNAME();
1061
1062         return ((PK11_LIBNAME = BUF_strdup(name)) != NULL ? 1 : 0);
1063         }
1064
1065 /* acquire all engine specific mutexes before fork */
1066 static void pk11_fork_prepare(void)
1067         {
1068         int i;
1069
1070         if (!pk11_library_initialized)
1071                 return;
1072
1073         LOCK_OBJSTORE(OP_RSA);
1074         LOCK_OBJSTORE(OP_DSA);
1075         LOCK_OBJSTORE(OP_DH);
1076         for (i = 0; i < OP_MAX; i++)
1077                 {
1078                 (void) pthread_mutex_lock(session_cache[i].lock);
1079                 }
1080         }
1081
1082 /* release all engine specific mutexes */
1083 static void pk11_fork_parent(void)
1084         {
1085         int i;
1086
1087         if (!pk11_library_initialized)
1088                 return;
1089
1090         for (i = OP_MAX - 1; i >= 0; i--)
1091                 {
1092                 (void) pthread_mutex_unlock(session_cache[i].lock);
1093                 }
1094         UNLOCK_OBJSTORE(OP_DH);
1095         UNLOCK_OBJSTORE(OP_DSA);
1096         UNLOCK_OBJSTORE(OP_RSA);
1097         }
1098
1099 /*
1100  * same situation as in parent - we need to unlock all locks to make them
1101  * accessible to all threads.
1102  */
1103 static void pk11_fork_child(void)
1104         {
1105         int i;
1106
1107         if (!pk11_library_initialized)
1108                 return;
1109
1110         for (i = OP_MAX - 1; i >= 0; i--)
1111                 {
1112                 (void) pthread_mutex_unlock(session_cache[i].lock);
1113                 }
1114         UNLOCK_OBJSTORE(OP_DH);
1115         UNLOCK_OBJSTORE(OP_DSA);
1116         UNLOCK_OBJSTORE(OP_RSA);
1117         }
```



```
1119 /* Initialization function for the pk11 engine */
1120 static int pk11_init(ENGINE *e)
1121 {
1122         return (pk11_library_init(e));
1123 }
1124
1125 /*
1126  * Initialization function. Sets up various PKCS#11 library components.
1127  * It selects a slot based on predefined critiera. In the process, it also
1128  * count how many ciphers and digests to support. Since the cipher and
1129  * digest information is needed when setting default engine, this function
1130  * needs to be called before calling ENGINE_set_default.
1131  */
1132 /* ARGSUSED */
1133 static int pk11_library_init(ENGINE *e)
1134         {
1135         CK_C_GetFunctionList p;
1136         CK_RV rv = CKR_OK;
1137         CK_INFO info;
1138         CK_ULONG ul_state_len;
1139         int any_slot_found;
1140         int i;
1141
1142         /*
1143          * pk11_library_initialized is set to 0 in pk11_finish() which is called
1144          * from ENGINE_finish(). However, if there is still at least one
1145          * existing functional reference to the engine (see engine(3) for more
1146          * information), pk11_finish() is skipped. For example, this can happen
1147          * if an application forgets to clear one cipher context. In case of a
1148          * fork() when the application is finishing the engine so that it can be
1149          * reinitialized in the child, forgotten functional reference causes
1150          * pk11_library_initialized to stay 1. In that case we need the PID
1151          * check so that we properly initialize the engine again.
1152          */
1153         if (pk11_library_initialized)
1154                 {
1155                 if (pk11_pid == getpid())
1156                         {
1157                         return (1);
1158                         }
1159                 else
1160                         {
1161                         global_session = CK_INVALID_HANDLE;
1162                         /*
1163                          * free the locks first to prevent memory leak in case
1164                          * the application calls fork() without finishing the
1165                          * engine first.
1166                          */
1167                         pk11_free_all_locks();
1168                         }
1169                 }
1170
1171         if (pk11_dso == NULL)
1172                 {
1173                 PK11err(PK11_F_LIBRARY_INIT, PK11_R_DSO_FAILURE);
1174                 goto err;
1175                 }
1176
1177 #ifdef  SOLARIS_AES_CTR
1178         /*
1179          * We must do this before we start working with slots since we need all
1180          * NIDs there.
1181          */
1182         if (pk11_add_aes_ctr_NIDs() == 0)
1183                 goto err;
```

```
#endif  /* SOLARIS_AES_CTR */

#ifdef  SOLARIS_HW_SLOT_SELECTION
        if (check_hw_mechanisms() == 0)
                goto err;
#endif  /* SOLARIS_HW_SLOT_SELECTION */

        /* get the C_GetFunctionList function from the loaded library */
        p = (CK_C_GetFunctionList)DSO_bind_func(pk11_dso,
                PK11_GET_FUNCTION_LIST);
        if (!p)
                {
                PK11err(PK11_F_LIBRARY_INIT, PK11_R_DSO_FAILURE);
                goto err;
                }

        /* get the full function list from the loaded library */
        rv = p(&pFuncList);
        if (rv != CKR_OK)
                {
                PK11err_add_data(PK11_F_LIBRARY_INIT, PK11_R_DSO_FAILURE, rv);
                goto err;
                }

        rv = pFuncList->C_Initialize(NULL_PTR);
        if ((rv != CKR_OK) && (rv != CKR_CRYPTOKI_ALREADY_INITIALIZED))
                {
                PK11err_add_data(PK11_F_LIBRARY_INIT, PK11_R_INITIALIZE, rv);
                goto err;
                }

        rv = pFuncList->C_GetInfo(&info);
        if (rv != CKR_OK)
                {
                PK11err_add_data(PK11_F_LIBRARY_INIT, PK11_R_GETINFO, rv);
                goto err;
                }

        if (pk11_choose_slots(&any_slot_found) == 0)
                goto err;

        /*
         * The library we use, set in def_PK11_LIBNAME, may not offer any
         * slot(s). In that case, we must not proceed but we must not return an
         * error. The reason is that applications that try to set up the PKCS#11
         * engine don't exit on error during the engine initialization just
         * because no slot was present.
         */
        if (any_slot_found == 0)
                return (1);

        if (global_session == CK_INVALID_HANDLE)
                {
                /* Open the global_session for the new process */
                rv = pFuncList->C_OpenSession(SLOTID, CKF_SERIAL_SESSION,
                        NULL_PTR, NULL_PTR, &global_session);
                if (rv != CKR_OK)
                        {
                        PK11err_add_data(PK11_F_LIBRARY_INIT,
                            PK11_R_OPENSESSION, rv);
                        goto err;
                        }
                }

        /*
         * Disable digest if C_GetOperationState is not supported since
```



```
         * this function is required by OpenSSL digest copy function
         */
        if (pFuncList->C_GetOperationState(global_session, NULL, &ul_state_len)
                        == CKR_FUNCTION_NOT_SUPPORTED) {
#ifdef  DEBUG_SLOT_SELECTION
                fprintf(stderr, "%s: C_GetOperationState() not supported, "
                    "setting digest_count to 0\n", PK11_DBG);
#endif  /* DEBUG_SLOT_SELECTION */
                digest_count = 0;
        }

        pk11_library_initialized = TRUE;
        pk11_pid = getpid();
        /*
         * if initialization of the locks fails pk11_init_all_locks()
         * will do the cleanup.
         */
        if (!pk11_init_all_locks())
                goto err;
        for (i = 0; i < OP_MAX; i++)
                session_cache[i].head = NULL;
        /*
         * initialize active lists. We only use active lists
         * for asymmetric ciphers.
         */
        for (i = 0; i < OP_MAX; i++)
                active_list[i] = NULL;

        if (!pk11_atfork_initialized)
                {
                if (pthread_atfork(pk11_fork_prepare, pk11_fork_parent,
                    pk11_fork_child) != 0)
                        {
                        PK11err(PK11_F_LIBRARY_INIT, PK11_R_ATFORK_FAILED);
                        goto err;
                        }
                pk11_atfork_initialized = TRUE;
                }

        return (1);

err:
        return (0);
        }

/* Destructor (complements the "ENGINE_pk11()" constructor) */
/* ARGSUSED */
static int pk11_destroy(ENGINE *e)
        {
        free_PK11_LIBNAME();
        ERR_unload_pk11_strings();
        return (1);
        }

/*
 * Termination function to clean up the session, the token, and the pk11
 * library.
 */
/* ARGSUSED */
static int pk11_finish(ENGINE *e)
        {
        int i;

        if (pk11_dso == NULL)
                {
                PK11err(PK11_F_FINISH, PK11_R_NOT_LOADED);
```

```
1316                goto err;
1317                }

1319        OPENSSL_assert(pFuncList != NULL);

1321        if (pk11_free_all_sessions() == 0)
1322                goto err;

1324        /* free all active lists */
1325        for (i = 0; i < OP_MAX; i++)
1326                pk11_free_active_list(i);

1328        pFuncList->C_CloseSession(global_session);
1329        global_session = CK_INVALID_HANDLE;

1331        /*
1332         * Since we are part of a library (libcrypto.so), calling this function
1333         * may have side-effects.
1334         */
1335 #if 0
1336        pFuncList->C_Finalize(NULL);
1337 #endif

1339        if (!DSO_free(pk11_dso))
1340                {
1341                PK11err(PK11_F_FINISH, PK11_R_DSO_FAILURE);
1342                goto err;
1343                }
1344        pk11_dso = NULL;
1345        pFuncList = NULL;
1346        pk11_library_initialized = FALSE;
1347        pk11_pid = 0;
1348        /*
1349         * There is no way how to unregister atfork handlers (other than
1350         * unloading the library) so we just free the locks. For this reason
1351         * the atfork handlers check if the engine is initialized and bail out
1352         * immediately if not. This is necessary in case a process finishes
1353         * the engine before calling fork().
1354         */
1355        pk11_free_all_locks();

1357        return (1);

1359 err:
1360        return (0);
1361        }

1363 /* Standard engine interface function to set the dynamic library path */
1364 /* ARGSUSED */
1365 static int pk11_ctrl(ENGINE *e, int cmd, long i, void *p, void (*f)())
1366        {
1367        int initialized = ((pk11_dso == NULL) ? 0 : 1);

1369        switch (cmd)
1370                {
1371        case PK11_CMD_SO_PATH:
1372                if (p == NULL)
1373                        {
1374                        PK11err(PK11_F_CTRL, ERR_R_PASSED_NULL_PARAMETER);
1375                        return (0);
1376                        }

1378                if (initialized)
1379                        {
1380                        PK11err(PK11_F_CTRL, PK11_R_ALREADY_LOADED);
1381                        return (0);
```



```
1382                        }

1384                return (set_PK11_LIBNAME((const char *)p));
1385        default:
1386                break;
1387                }

1389        PK11err(PK11_F_CTRL, PK11_R_CTRL_COMMAND_NOT_IMPLEMENTED);

1391        return (0);
1392        }


1395 /* Required function by the engine random interface. It does nothing here */
1396 static void pk11_rand_cleanup(void)
1397        {
1398        return;
1399        }

1401 /* ARGSUSED */
1402 static void pk11_rand_add(const void *buf, int num, double add)
1403        {
1404        PK11_SESSION *sp;

1406        if ((sp = pk11_get_session(OP_RAND)) == NULL)
1407                return;

1409        /*
1410         * Ignore any errors (e.g. CKR_RANDOM_SEED_NOT_SUPPORTED) since
1411         * the calling functions do not care anyway
1412         */
1413        pFuncList->C_SeedRandom(sp->session, (unsigned char *) buf, num);
1414        pk11_return_session(sp, OP_RAND);

1416        return;
1417        }

1419 static void pk11_rand_seed(const void *buf, int num)
1420        {
1421        pk11_rand_add(buf, num, 0);
1422        }

1424 static int pk11_rand_bytes(unsigned char *buf, int num)
1425        {
1426        CK_RV rv;
1427        PK11_SESSION *sp;

1429        if ((sp = pk11_get_session(OP_RAND)) == NULL)
1430                return (0);

1432        rv = pFuncList->C_GenerateRandom(sp->session, buf, num);
1433        if (rv != CKR_OK)
1434                {
1435                PK11err_add_data(PK11_F_RAND_BYTES, PK11_R_GENERATERANDOM, rv);
1436                pk11_return_session(sp, OP_RAND);
1437                return (0);
1438                }

1440        pk11_return_session(sp, OP_RAND);
1441        return (1);
1442        }

1444 /* Required function by the engine random interface. It does nothing here */
1445 static int pk11_rand_status(void)
1446        {
1447        return (1);
```

```
1448         }

1450 /* Free all BIGNUM structures from PK11_SESSION. */
1451 static void pk11_free_nums(PK11_SESSION *sp, PK11_OPTYPE optype)
1452         {
1453         switch (optype)
1454                 {
1455 #ifndef OPENSSL_NO_RSA
1456                 case OP_RSA:
1457                         if (sp->opdata_rsa_n_num != NULL)
1458                                 {
1459                                 BN_free(sp->opdata_rsa_n_num);
1460                                 sp->opdata_rsa_n_num = NULL;
1461                                 }
1462                         if (sp->opdata_rsa_e_num != NULL)
1463                                 {
1464                                 BN_free(sp->opdata_rsa_e_num);
1465                                 sp->opdata_rsa_e_num = NULL;
1466                                 }
1467                         if (sp->opdata_rsa_d_num != NULL)
1468                                 {
1469                                 BN_free(sp->opdata_rsa_d_num);
1470                                 sp->opdata_rsa_d_num = NULL;
1471                                 }
1472                         break;
1473 #endif
1474 #ifndef OPENSSL_NO_DSA
1475                 case OP_DSA:
1476                         if (sp->opdata_dsa_pub_num != NULL)
1477                                 {
1478                                 BN_free(sp->opdata_dsa_pub_num);
1479                                 sp->opdata_dsa_pub_num = NULL;
1480                                 }
1481                         if (sp->opdata_dsa_priv_num != NULL)
1482                                 {
1483                                 BN_free(sp->opdata_dsa_priv_num);
1484                                 sp->opdata_dsa_priv_num = NULL;
1485                                 }
1486                         break;
1487 #endif
1488 #ifndef OPENSSL_NO_DH
1489                 case OP_DH:
1490                         if (sp->opdata_dh_priv_num != NULL)
1491                                 {
1492                                 BN_free(sp->opdata_dh_priv_num);
1493                                 sp->opdata_dh_priv_num = NULL;
1494                                 }
1495                         break;
1496 #endif
1497                 default:
1498                         break;
1499                 }
1500         }

1502 /*
1503  * Get new PK11_SESSION structure ready for use. Every process must have
1504  * its own freelist of PK11_SESSION structures so handle fork() here
1505  * by destroying the old and creating new freelist.
1506  * The returned PK11_SESSION structure is disconnected from the freelist.
1507  */
1508 PK11_SESSION *
1509 pk11_get_session(PK11_OPTYPE optype)
1510         {
1511         PK11_SESSION *sp = NULL, *sp1, *freelist;
1512         pthread_mutex_t *freelist_lock;
1513         CK_RV rv;
```



```
1515         switch (optype)
1516                 {
1517                 case OP_RSA:
1518                 case OP_DSA:
1519                 case OP_DH:
1520                 case OP_RAND:
1521                 case OP_DIGEST:
1522                 case OP_CIPHER:
1523                         freelist_lock = session_cache[optype].lock;
1524                         break;
1525                 default:
1526                         PK11err(PK11_F_GET_SESSION,
1527                                 PK11_R_INVALID_OPERATION_TYPE);
1528                         return (NULL);
1529                 }
1530         (void) pthread_mutex_lock(freelist_lock);
1531         freelist = session_cache[optype].head;
1532         sp = freelist;

1534         /*
1535          * If the free list is empty, allocate new unitialized (filled
1536          * with zeroes) PK11_SESSION structure otherwise return first
1537          * structure from the freelist.
1538          */
1539         if (sp == NULL)
1540                 {
1541                 if ((sp = OPENSSL_malloc(sizeof (PK11_SESSION))) == NULL)
1542                         {
1543                         PK11err(PK11_F_GET_SESSION,
1544                                 PK11_R_MALLOC_FAILURE);
1545                         goto err;
1546                         }
1547                 (void) memset(sp, 0, sizeof (PK11_SESSION));
1548                 }
1549         else
1550                 {
1551                 freelist = sp->next;
1552                 }

1554         if (sp->pid != 0 && sp->pid != getpid())
1555                 {
1556                 /*
1557                  * We are a new process and thus need to free any inherited
1558                  * PK11_SESSION objects.
1559                  */
1560                 while ((sp1 = freelist) != NULL)
1561                         {
1562                         freelist = sp1->next;
1563                         /*
1564                          * NOTE: we do not want to call pk11_free_all_sessions()
1565                          * here because it would close underlying PKCS#11
1566                          * sessions and destroy all objects.
1567                          */
1568                         pk11_free_nums(sp1, optype);
1569                         OPENSSL_free(sp1);
1570                         }

1572                 /* we have to free the active list as well. */
1573                 pk11_free_active_list(optype);

1575                 /* Initialize the process */
1576                 rv = pFuncList->C_Initialize(NULL_PTR);
1577                 if ((rv != CKR_OK) && (rv != CKR_CRYPTOKI_ALREADY_INITIALIZED))
1578                         {
1579                         PK11err_add_data(PK11_F_GET_SESSION, PK11_R_INITIALIZE,
```

```
1580                            rv);
1581                    OPENSSL_free(sp);
1582                    sp = NULL;
1583                    goto err;
1584                    }

1586            /*
1587             * Choose slot here since the slot table is different on this
1588             * process. If we are here then we must have found at least one
1589             * usable slot before so we don't need to check any_slot_found.
1590             * See pk11_library_init()'s usage of this function for more
1591             * information.
1592             */
1593 #ifdef  SOLARIS_HW_SLOT_SELECTION
1594            if (check_hw_mechanisms() == 0)
1595                    goto err;
1596 #endif  /* SOLARIS_HW_SLOT_SELECTION */
1597            if (pk11_choose_slots(NULL) == 0)
1598                    goto err;

1600            /* Open the global_session for the new process */
1601            rv = pFuncList->C_OpenSession(SLOTID, CKF_SERIAL_SESSION,
1602                    NULL_PTR, NULL_PTR, &global_session);
1603            if (rv != CKR_OK)
1604                    {
1605                    PK11err_add_data(PK11_F_GET_SESSION, PK11_R_OPENSESSION,
1606                            rv);
1607                    OPENSSL_free(sp);
1608                    sp = NULL;
1609                    goto err;
1610                    }

1612            /* It is an inherited session and needs re-initialization. */
1613            if (pk11_setup_session(sp, optype) == 0)
1614                    {
1615                    OPENSSL_free(sp);
1616                    sp = NULL;
1617                    }
1618            }
1619        if (sp->pid == 0)
1620                {
1621                /* It is a new session and needs initialization. */
1622                if (pk11_setup_session(sp, optype) == 0)
1623                        {
1624                        OPENSSL_free(sp);
1625                        sp = NULL;
1626                        }
1627                }

1629        /* set new head for the list of PK11_SESSION objects */
1630        session_cache[optype].head = freelist;

1632 err:
1633        if (sp != NULL)
1634                sp->next = NULL;

1636        (void) pthread_mutex_unlock(freelist_lock);

1638        return (sp);
1639        }


1642 void
1643 pk11_return_session(PK11_SESSION *sp, PK11_OPTYPE optype)
1644        {
1645        pthread_mutex_t *freelist_lock;
```



```
1646        PK11_SESSION *freelist;

1648        if (sp == NULL || sp->pid != getpid())
1649                return;

1651        switch (optype)
1652                {
1653                case OP_RSA:
1654                case OP_DSA:
1655                case OP_DH:
1656                case OP_RAND:
1657                case OP_DIGEST:
1658                case OP_CIPHER:
1659                        freelist_lock = session_cache[optype].lock;
1660                        break;
1661                default:
1662                        PK11err(PK11_F_RETURN_SESSION,
1663                                PK11_R_INVALID_OPERATION_TYPE);
1664                        return;
1665                }

1667        (void) pthread_mutex_lock(freelist_lock);
1668        freelist = session_cache[optype].head;
1669        sp->next = freelist;
1670        session_cache[optype].head = sp;
1671        (void) pthread_mutex_unlock(freelist_lock);
1672        }


1675 /* Destroy all objects. This function is called when the engine is finished */
1676 static int pk11_free_all_sessions()
1677        {
1678        int ret = 1;
1679        int type;

1681 #ifndef OPENSSL_NO_RSA
1682        (void) pk11_destroy_rsa_key_objects(NULL);
1683 #endif  /* OPENSSL_NO_RSA */
1684 #ifndef OPENSSL_NO_DSA
1685        (void) pk11_destroy_dsa_key_objects(NULL);
1686 #endif  /* OPENSSL_NO_DSA */
1687 #ifndef OPENSSL_NO_DH
1688        (void) pk11_destroy_dh_key_objects(NULL);
1689 #endif  /* OPENSSL_NO_DH */
1690        (void) pk11_destroy_cipher_key_objects(NULL);

1692        /*
1693         * We try to release as much as we can but any error means that we will
1694         * return 0 on exit.
1695         */
1696        for (type = 0; type < OP_MAX; type++)
1697                {
1698                if (pk11_free_session_list(type) == 0)
1699                        ret = 0;
1700                }

1702        return (ret);
1703        }

1705 /*
1706  * Destroy session structures from the linked list specified. Free as many
1707  * sessions as possible but any failure in C_CloseSession() means that we
1708  * return an error on return.
1709  */
1710 static int pk11_free_session_list(PK11_OPTYPE optype)
1711        {
```

```
1712         CK_RV rv;
1713         PK11_SESSION *sp = NULL;
1714         PK11_SESSION *freelist = NULL;
1715         pid_t mypid = getpid();
1716         pthread_mutex_t *freelist_lock;
1717         int ret = 1;

1719         switch (optype)
1720                 {
1721                 case OP_RSA:
1722                 case OP_DSA:
1723                 case OP_DH:
1724                 case OP_RAND:
1725                 case OP_DIGEST:
1726                 case OP_CIPHER:
1727                         freelist_lock = session_cache[optype].lock;
1728                         break;
1729                 default:
1730                         PK11err(PK11_F_FREE_ALL_SESSIONS,
1731                                 PK11_R_INVALID_OPERATION_TYPE);
1732                         return (0);
1733                 }

1735         (void) pthread_mutex_lock(freelist_lock);
1736         freelist = session_cache[optype].head;
1737         while ((sp = freelist) != NULL)
1738                 {
1739                 if (sp->session != CK_INVALID_HANDLE && sp->pid == mypid)
1740                         {
1741                         rv = pFuncList->C_CloseSession(sp->session);
1742                         if (rv != CKR_OK)
1743                                 {
1744                                 PK11err_add_data(PK11_F_FREE_ALL_SESSIONS,
1745                                         PK11_R_CLOSESESSION, rv);
1746                                 ret = 0;
1747                                 }
1748                         }
1749                 freelist = sp->next;
1750                 pk11_free_nums(sp, optype);
1751                 OPENSSL_free(sp);
1752                 }

1754         (void) pthread_mutex_unlock(freelist_lock);
1755         return (ret);
1756         }


1759 static int pk11_setup_session(PK11_SESSION *sp, PK11_OPTYPE optype)
1760         {
1761         CK_RV rv;
1762         CK_SLOT_ID myslot;

1764         switch (optype)
1765                 {
1766                 case OP_RSA:
1767                 case OP_DSA:
1768                 case OP_DH:
1769                         myslot = pubkey_SLOTID;
1770                         break;
1771                 case OP_RAND:
1772                         myslot = rand_SLOTID;
1773                         break;
1774                 case OP_DIGEST:
1775                 case OP_CIPHER:
1776                         myslot = SLOTID;
1777                         break;
```



```
1778                 default:
1779                         PK11err(PK11_F_SETUP_SESSION,
1780                             PK11_R_INVALID_OPERATION_TYPE);
1781                         return (0);
1782                 }

1784         sp->session = CK_INVALID_HANDLE;
1785 #ifdef  DEBUG_SLOT_SELECTION
1786         fprintf(stderr, "%s: myslot=%d optype=%d\n", PK11_DBG, myslot, optype);
1787 #endif  /* DEBUG_SLOT_SELECTION */
1788         rv = pFuncList->C_OpenSession(myslot, CKF_SERIAL_SESSION,
1789                 NULL_PTR, NULL_PTR, &sp->session);
1790         if (rv == CKR_CRYPTOKI_NOT_INITIALIZED)
1791                 {
1792                 /*
1793                  * We are probably a child process so force the
1794                  * reinitialize of the session
1795                  */
1796                 pk11_library_initialized = FALSE;
1797                 if (!pk11_library_init(NULL))
1798                         return (0);
1799                 rv = pFuncList->C_OpenSession(myslot, CKF_SERIAL_SESSION,
1800                         NULL_PTR, NULL_PTR, &sp->session);
1801                 }
1802         if (rv != CKR_OK)
1803                 {
1804                 PK11err_add_data(PK11_F_SETUP_SESSION, PK11_R_OPENSESSION, rv);
1805                 return (0);
1806                 }

1808         sp->pid = getpid();

1810         switch (optype)
1811                 {
1812 #ifndef OPENSSL_NO_RSA
1813                 case OP_RSA:
1814                         sp->opdata_rsa_pub_key = CK_INVALID_HANDLE;
1815                         sp->opdata_rsa_priv_key = CK_INVALID_HANDLE;
1816                         sp->opdata_rsa_pub = NULL;
1817                         sp->opdata_rsa_n_num = NULL;
1818                         sp->opdata_rsa_e_num = NULL;
1819                         sp->opdata_rsa_priv = NULL;
1820                         sp->opdata_rsa_d_num = NULL;
1821                         break;
1822 #endif  /* OPENSSL_NO_RSA */
1823 #ifndef OPENSSL_NO_DSA
1824                 case OP_DSA:
1825                         sp->opdata_dsa_pub_key = CK_INVALID_HANDLE;
1826                         sp->opdata_dsa_priv_key = CK_INVALID_HANDLE;
1827                         sp->opdata_dsa_pub = NULL;
1828                         sp->opdata_dsa_pub_num = NULL;
1829                         sp->opdata_dsa_priv = NULL;
1830                         sp->opdata_dsa_priv_num = NULL;
1831                         break;
1832 #endif  /* OPENSSL_NO_DSA */
1833 #ifndef OPENSSL_NO_DH
1834                 case OP_DH:
1835                         sp->opdata_dh_key = CK_INVALID_HANDLE;
1836                         sp->opdata_dh = NULL;
1837                         sp->opdata_dh_priv_num = NULL;
1838                         break;
1839 #endif  /* OPENSSL_NO_DH */
1840                 case OP_CIPHER:
1841                         sp->opdata_cipher_key = CK_INVALID_HANDLE;
1842                         sp->opdata_encrypt = -1;
1843                         break;
```

```
1844                }

1846        return (1);
1847        }

1849 #ifndef OPENSSL_NO_RSA
1850 /* Destroy RSA public key from single session. */
1851 int
1852 pk11_destroy_rsa_object_pub(PK11_SESSION *sp, CK_BBOOL uselock)
1853         {
1854         int ret = 0;

1856         if (sp->opdata_rsa_pub_key != CK_INVALID_HANDLE)
1857                 {
1858                 TRY_OBJ_DESTROY(sp->session, sp->opdata_rsa_pub_key,
1859                     ret, uselock, OP_RSA);
1860                 sp->opdata_rsa_pub_key = CK_INVALID_HANDLE;
1861                 sp->opdata_rsa_pub = NULL;
1862                 if (sp->opdata_rsa_n_num != NULL)
1863                         {
1864                         BN_free(sp->opdata_rsa_n_num);
1865                         sp->opdata_rsa_n_num = NULL;
1866                         }
1867                 if (sp->opdata_rsa_e_num != NULL)
1868                         {
1869                         BN_free(sp->opdata_rsa_e_num);
1870                         sp->opdata_rsa_e_num = NULL;
1871                         }
1872                 }

1874         return (ret);
1875         }

1877 /* Destroy RSA private key from single session. */
1878 int
1879 pk11_destroy_rsa_object_priv(PK11_SESSION *sp, CK_BBOOL uselock)
1880         {
1881         int ret = 0;

1883         if (sp->opdata_rsa_priv_key != CK_INVALID_HANDLE)
1884                 {
1885                 TRY_OBJ_DESTROY(sp->session, sp->opdata_rsa_priv_key,
1886                     ret, uselock, OP_RSA);
1887                 sp->opdata_rsa_priv_key = CK_INVALID_HANDLE;
1888                 sp->opdata_rsa_priv = NULL;
1889                 if (sp->opdata_rsa_d_num != NULL)
1890                         {
1891                         BN_free(sp->opdata_rsa_d_num);
1892                         sp->opdata_rsa_d_num = NULL;
1893                         }
1894                 }

1896         return (ret);
1897         }

1899 /*
1900  * Destroy RSA key object wrapper. If session is NULL, try to destroy all
1901  * objects in the free list.
1902  */
1903 int
1904 pk11_destroy_rsa_key_objects(PK11_SESSION *session)
1905         {
1906         int ret = 1;
1907         PK11_SESSION *sp = NULL;
1908         PK11_SESSION *local_free_session;
1909         CK_BBOOL uselock = TRUE;
```



```
1911         if (session != NULL)
1912                 local_free_session = session;
1913         else
1914                 {
1915                 (void) pthread_mutex_lock(session_cache[OP_RSA].lock);
1916                 local_free_session = session_cache[OP_RSA].head;
1917                 uselock = FALSE;
1918                 }

1920         /*
1921          * go through the list of sessions and delete key objects
1922          */
1923         while ((sp = local_free_session) != NULL)
1924                 {
1925                 local_free_session = sp->next;

1927                 /*
1928                  * Do not terminate list traversal if one of the
1929                  * destroy operations fails.
1930                  */
1931                 if (pk11_destroy_rsa_object_pub(sp, uselock) == 0)
1932                         {
1933                         ret = 0;
1934                         continue;
1935                         }
1936                 if (pk11_destroy_rsa_object_priv(sp, uselock) == 0)
1937                         {
1938                         ret = 0;
1939                         continue;
1940                         }
1941                 }

1943         if (session == NULL)
1944                 (void) pthread_mutex_unlock(session_cache[OP_RSA].lock);

1946         return (ret);
1947         }
1948 #endif  /* OPENSSL_NO_RSA */

1950 #ifndef OPENSSL_NO_DSA
1951 /* Destroy DSA public key from single session. */
1952 int
1953 pk11_destroy_dsa_object_pub(PK11_SESSION *sp, CK_BBOOL uselock)
1954         {
1955         int ret = 0;

1957         if (sp->opdata_dsa_pub_key != CK_INVALID_HANDLE)
1958                 {
1959                 TRY_OBJ_DESTROY(sp->session, sp->opdata_dsa_pub_key,
1960                     ret, uselock, OP_DSA);
1961                 sp->opdata_dsa_pub_key = CK_INVALID_HANDLE;
1962                 sp->opdata_dsa_pub = NULL;
1963                 if (sp->opdata_dsa_pub_num != NULL)
1964                         {
1965                         BN_free(sp->opdata_dsa_pub_num);
1966                         sp->opdata_dsa_pub_num = NULL;
1967                         }
1968                 }

1970         return (ret);
1971         }

1973 /* Destroy DSA private key from single session. */
1974 int
1975 pk11_destroy_dsa_object_priv(PK11_SESSION *sp, CK_BBOOL uselock)
```

```
1976         {
1977         int ret = 0;

1979         if (sp->opdata_dsa_priv_key != CK_INVALID_HANDLE)
1980                 {
1981                 TRY_OBJ_DESTROY(sp->session, sp->opdata_dsa_priv_key,
1982                     ret, uselock, OP_DSA);
1983                 sp->opdata_dsa_priv_key = CK_INVALID_HANDLE;
1984                 sp->opdata_dsa_priv = NULL;
1985                 if (sp->opdata_dsa_priv_num != NULL)
1986                         {
1987                         BN_free(sp->opdata_dsa_priv_num);
1988                         sp->opdata_dsa_priv_num = NULL;
1989                         }
1990                 }

1992         return (ret);
1993         }

1995 /*
1996  * Destroy DSA key object wrapper. If session is NULL, try to destroy all
1997  * objects in the free list.
1998  */
1999 int
2000 pk11_destroy_dsa_key_objects(PK11_SESSION *session)
2001         {
2002         int ret = 1;
2003         PK11_SESSION *sp = NULL;
2004         PK11_SESSION *local_free_session;
2005         CK_BBOOL uselock = TRUE;

2007         if (session != NULL)
2008                 local_free_session = session;
2009         else
2010                 {
2011                 (void) pthread_mutex_lock(session_cache[OP_DSA].lock);
2012                 local_free_session = session_cache[OP_DSA].head;
2013                 uselock = FALSE;
2014                 }

2016         /*
2017          * go through the list of sessions and delete key objects
2018          */
2019         while ((sp = local_free_session) != NULL)
2020                 {
2021                 local_free_session = sp->next;

2023                 /*
2024                  * Do not terminate list traversal if one of the
2025                  * destroy operations fails.
2026                  */
2027                 if (pk11_destroy_dsa_object_pub(sp, uselock) == 0)
2028                         {
2029                         ret = 0;
2030                         continue;
2031                         }
2032                 if (pk11_destroy_dsa_object_priv(sp, uselock) == 0)
2033                         {
2034                         ret = 0;
2035                         continue;
2036                         }
2037                 }

2039         if (session == NULL)
2040                 (void) pthread_mutex_unlock(session_cache[OP_DSA].lock);
```



```
2042         return (ret);
2043         }
2044 #endif  /* OPENSSL_NO_DSA */

2046 #ifndef OPENSSL_NO_DH
2047 /* Destroy DH key from single session. */
2048 int
2049 pk11_destroy_dh_object(PK11_SESSION *sp, CK_BBOOL uselock)
2050         {
2051         int ret = 0;

2053         if (sp->opdata_dh_key != CK_INVALID_HANDLE)
2054                 {
2055                 TRY_OBJ_DESTROY(sp->session, sp->opdata_dh_key,
2056                     ret, uselock, OP_DH);
2057                 sp->opdata_dh_key = CK_INVALID_HANDLE;
2058                 sp->opdata_dh = NULL;
2059                 if (sp->opdata_dh_priv_num != NULL)
2060                         {
2061                         BN_free(sp->opdata_dh_priv_num);
2062                         sp->opdata_dh_priv_num = NULL;
2063                         }
2064                 }

2066         return (ret);
2067         }

2069 /*
2070  * Destroy DH key object wrapper.
2071  *
2072  * arg0: pointer to PKCS#11 engine session structure
2073  *       if session is NULL, try to destroy all objects in the free list
2074  */
2075 int
2076 pk11_destroy_dh_key_objects(PK11_SESSION *session)
2077         {
2078         int ret = 1;
2079         PK11_SESSION *sp = NULL;
2080         PK11_SESSION *local_free_session;
2081         CK_BBOOL uselock = TRUE;

2083         if (session != NULL)
2084                 local_free_session = session;
2085         else
2086                 {
2087                 (void) pthread_mutex_lock(session_cache[OP_DH].lock);
2088                 local_free_session = session_cache[OP_DH].head;
2089                 uselock = FALSE;
2090                 }

2092         while ((sp = local_free_session) != NULL)
2093                 {
2094                 local_free_session = sp->next;

2096                 /*
2097                  * Do not terminate list traversal if one of the
2098                  * destroy operations fails.
2099                  */
2100                 if (pk11_destroy_dh_object(sp, uselock) == 0)
2101                         {
2102                         ret = 0;
2103                         continue;
2104                         }
2105                 }
2106         if (session == NULL)
2107                 (void) pthread_mutex_unlock(session_cache[OP_DH].lock);
```

```
2109         return (ret);
2110         }
2111 #endif  /* OPENSSL_NO_DH */

2113 static int pk11_destroy_object(CK_SESSION_HANDLE session, CK_OBJECT_HANDLE oh)
2114         {
2115         CK_RV rv;
2116         rv = pFuncList->C_DestroyObject(session, oh);
2117         if (rv != CKR_OK)
2118                 {
2119                 PK11err_add_data(PK11_F_DESTROY_OBJECT, PK11_R_DESTROYOBJECT,
2120                     rv);
2121                 return (0);
2122                 }

2124         return (1);
2125         }


2128 /* Symmetric ciphers and digests support functions */

2130 static int
2131 cipher_nid_to_pk11(int nid)
2132         {
2133         int i;

2135         for (i = 0; i < PK11_CIPHER_MAX; i++)
2136                 if (ciphers[i].nid == nid)
2137                         return (ciphers[i].id);
2138         return (-1);
2139         }

2141 static int
2142 pk11_usable_ciphers(const int **nids)
2143         {
2144         if (cipher_count > 0)
2145                 *nids = cipher_nids;
2146         else
2147                 *nids = NULL;
2148         return (cipher_count);
2149         }

2151 static int
2152 pk11_usable_digests(const int **nids)
2153         {
2154         if (digest_count > 0)
2155                 *nids = digest_nids;
2156         else
2157                 *nids = NULL;
2158         return (digest_count);
2159         }

2161 /*
2162  * Init context for encryption or decryption using a symmetric key.
2163  */
2164 static int pk11_init_symmetric(EVP_CIPHER_CTX *ctx, PK11_CIPHER *pcipher,
2165         PK11_SESSION *sp, CK_MECHANISM_PTR pmech)
2166         {
2167         CK_RV rv;
2168 #ifdef  SOLARIS_AES_CTR
2169         CK_AES_CTR_PARAMS ctr_params;
2170 #endif  /* SOLARIS_AES_CTR */

2172         /*
2173          * We expect pmech->mechanism to be already set and
```



```
2174          * pParameter/ulParameterLen initialized to NULL/0 before
2175          * pk11_init_symetric() is called.
2176          */
2177         OPENSSL_assert(pmech->mechanism != NULL);
2178         OPENSSL_assert(pmech->pParameter == NULL);
2179         OPENSSL_assert(pmech->ulParameterLen == 0);

2181 #ifdef  SOLARIS_AES_CTR
2182         if (ctx->cipher->nid == NID_aes_128_ctr ||
2183             ctx->cipher->nid == NID_aes_192_ctr ||
2184             ctx->cipher->nid == NID_aes_256_ctr)
2185                 {
2186                 pmech->pParameter = (void *)(&ctr_params);
2187                 pmech->ulParameterLen = sizeof (ctr_params);
2188                 /*
2189                  * For now, we are limited to the fixed length of the counter,
2190                  * it covers the whole counter block. That's what RFC 4344
2191                  * needs. For more information on internal structure of the
2192                  * counter block, see RFC 3686. If needed in the future, we can
2193                  * add code so that the counter length can be set via
2194                  * ENGINE_ctrl() function.
2195                  */
2196                 ctr_params.ulCounterBits = AES_BLOCK_SIZE * 8;
2197                 OPENSSL_assert(pcipher->iv_len == AES_BLOCK_SIZE);
2198                 (void) memcpy(ctr_params.cb, ctx->iv, AES_BLOCK_SIZE);
2199                 }
2200         else
2201 #endif  /* SOLARIS_AES_CTR */
2202                 {
2203                 if (pcipher->iv_len > 0)
2204                         {
2205                         pmech->pParameter = (void *)ctx->iv;
2206                         pmech->ulParameterLen = pcipher->iv_len;
2207                         }
2208                 }

2210         /* if we get here, the encryption needs to be reinitialized */
2211         if (ctx->encrypt)
2212                 rv = pFuncList->C_EncryptInit(sp->session, pmech,
2213                         sp->opdata_cipher_key);
2214         else
2215                 rv = pFuncList->C_DecryptInit(sp->session, pmech,
2216                         sp->opdata_cipher_key);

2218         if (rv != CKR_OK)
2219                 {
2220                 PK11err_add_data(PK11_F_CIPHER_INIT, ctx->encrypt ?
2221                     PK11_R_ENCRYPTINIT : PK11_R_DECRYPTINIT, rv);
2222                 pk11_return_session(sp, OP_CIPHER);
2223                 return (0);
2224                 }

2226         return (1);
2227         }

2229 /* ARGSUSED */
2230 static int
2231 pk11_cipher_init(EVP_CIPHER_CTX *ctx, const unsigned char *key,
2232     const unsigned char *iv, int enc)
2233         {
2234         CK_MECHANISM mech;
2235         int index;
2236         PK11_CIPHER_STATE *state = (PK11_CIPHER_STATE *) ctx->cipher_data;
2237         PK11_SESSION *sp;
2238         PK11_CIPHER *p_ciph_table_row;
```

```
2240         state->sp = NULL;

2242         index = cipher_nid_to_pk11(ctx->cipher->nid);
2243         if (index < 0 || index >= PK11_CIPHER_MAX)
2244                 return (0);

2246         p_ciph_table_row = &ciphers[index];
2247         /*
2248          * iv_len in the ctx->cipher structure is the maximum IV length for the
2249          * current cipher and it must be less or equal to the IV length in our
2250          * ciphers table. The key length must be in the allowed interval. From
2251          * all cipher modes that the PKCS#11 engine supports only RC4 allows a
2252          * key length to be in some range, all other NIDs have a precise key
2253          * length. Every application can define its own EVP functions so this
2254          * code serves as a sanity check.
2255          *
2256          * Note that the reason why the IV length in ctx->cipher might be
2257          * greater than the actual length is that OpenSSL uses BLOCK_CIPHER_defs
2258          * macro to define functions that return EVP structures for all DES
2259          * modes. So, even ECB modes get 8 byte IV.
2260          */
2261         if (ctx->cipher->iv_len < p_ciph_table_row->iv_len ||
2262             ctx->key_len < p_ciph_table_row->min_key_len ||
2263             ctx->key_len > p_ciph_table_row->max_key_len) {
2264                 PK11err(PK11_F_CIPHER_INIT, PK11_R_KEY_OR_IV_LEN_PROBLEM);
2265                 return (0);
2266         }

2268         if ((sp = pk11_get_session(OP_CIPHER)) == NULL)
2269                 return (0);

2271         /* if applicable, the mechanism parameter is used for IV */
2272         mech.mechanism = p_ciph_table_row->mech_type;
2273         mech.pParameter = NULL;
2274         mech.ulParameterLen = 0;

2276         /* The key object is destroyed here if it is not the current key. */
2277         (void) check_new_cipher_key(sp, key, ctx->key_len);

2279         /*
2280          * If the key is the same and the encryption is also the same, then
2281          * just reuse it. However, we must not forget to reinitialize the
2282          * context that was finalized in pk11_cipher_cleanup().
2283          */
2284         if (sp->opdata_cipher_key != CK_INVALID_HANDLE &&
2285             sp->opdata_encrypt == ctx->encrypt)
2286                 {
2287                 state->sp = sp;
2288                 if (pk11_init_symmetric(ctx, p_ciph_table_row, sp, &mech) == 0)
2289                         return (0);

2291                 return (1);
2292                 }

2294         /*
2295          * Check if the key has been invalidated. If so, a new key object
2296          * needs to be created.
2297          */
2298         if (sp->opdata_cipher_key == CK_INVALID_HANDLE)
2299                 {
2300                 sp->opdata_cipher_key = pk11_get_cipher_key(
2301                         ctx, key, p_ciph_table_row->key_type, sp);
2302                 }

2304         if (sp->opdata_encrypt != ctx->encrypt && sp->opdata_encrypt != -1)
2305                 {
```



```
2306                 /*
2307                  * The previous encryption/decryption is different. Need to
2308                  * terminate the previous * active encryption/decryption here.
2309                  */
2310                 if (!pk11_cipher_final(sp))
2311                         {
2312                         pk11_return_session(sp, OP_CIPHER);
2313                         return (0);
2314                         }
2315                 }

2317         if (sp->opdata_cipher_key == CK_INVALID_HANDLE)
2318                 {
2319                 pk11_return_session(sp, OP_CIPHER);
2320                 return (0);
2321                 }

2323         /* now initialize the context with a new key */
2324         if (pk11_init_symmetric(ctx, p_ciph_table_row, sp, &mech) == 0)
2325                 return (0);

2327         sp->opdata_encrypt = ctx->encrypt;
2328         state->sp = sp;

2330         return (1);
2331         }

2333 /*
2334  * When reusing the same key in an encryption/decryption session for a
2335  * decryption/encryption session, we need to close the active session
2336  * and recreate a new one. Note that the key is in the global session so
2337  * that it needs not be recreated.
2338  *
2339  * It is more appropriate to use C_En/DecryptFinish here. At the time of this
2340  * development, these two functions in the PKCS#11 libraries used return
2341  * unexpected errors when passing in 0 length output. It may be a good
2342  * idea to try them again if performance is a problem here and fix
2343  * C_En/DecryptFinial if there are bugs there causing the problem.
2344  */
2345 static int
2346 pk11_cipher_final(PK11_SESSION *sp)
2347         {
2348         CK_RV rv;

2350         rv = pFuncList->C_CloseSession(sp->session);
2351         if (rv != CKR_OK)
2352                 {
2353                 PK11err_add_data(PK11_F_CIPHER_FINAL, PK11_R_CLOSESESSION, rv);
2354                 return (0);
2355                 }

2357         rv = pFuncList->C_OpenSession(SLOTID, CKF_SERIAL_SESSION,
2358                 NULL_PTR, NULL_PTR, &sp->session);
2359         if (rv != CKR_OK)
2360                 {
2361                 PK11err_add_data(PK11_F_CIPHER_FINAL, PK11_R_OPENSESSION, rv);
2362                 return (0);
2363                 }

2365         return (1);
2366         }

2368 /*
2369  * An engine interface function. The calling function allocates sufficient
2370  * memory for the output buffer "out" to hold the results.
2371  */
```

```
2372 static int
2373 pk11_cipher_do_cipher(EVP_CIPHER_CTX *ctx, unsigned char *out,
2374         const unsigned char *in, size_t inl)
2375         {
2376         PK11_CIPHER_STATE *state = (PK11_CIPHER_STATE *) ctx->cipher_data;
2377         PK11_SESSION *sp;
2378         CK_RV rv;
2379         unsigned long outl = inl;

2381         if (state == NULL || state->sp == NULL)
2382                 return (0);

2384         sp = (PK11_SESSION *) state->sp;

2386         if (!inl)
2387                 return (1);

2389         /* RC4 is the only stream cipher we support */
2390         if (ctx->cipher->nid != NID_rc4 && (inl % ctx->cipher->block_size) != 0)
2391                 return (0);

2393         if (ctx->encrypt)
2394                 {
2395                 rv = pFuncList->C_EncryptUpdate(sp->session,
2396                         (unsigned char *)in, inl, out, &outl);

2398                 if (rv != CKR_OK)
2399                         {
2400                         PK11err_add_data(PK11_F_CIPHER_DO_CIPHER,
2401                             PK11_R_ENCRYPTUPDATE, rv);
2402                         return (0);
2403                         }
2404                 }
2405         else
2406                 {
2407                 rv = pFuncList->C_DecryptUpdate(sp->session,
2408                         (unsigned char *)in, inl, out, &outl);

2410                 if (rv != CKR_OK)
2411                         {
2412                         PK11err_add_data(PK11_F_CIPHER_DO_CIPHER,
2413                             PK11_R_DECRYPTUPDATE, rv);
2414                         return (0);
2415                         }
2416                 }

2418         /*
2419          * For DES_CBC, DES3_CBC, AES_CBC, and RC4, the output size is always
2420          * the same size of input.
2421          * The application has guaranteed to call the block ciphers with
2422          * correctly aligned buffers.
2423          */
2424         if (inl != outl)
2425                 return (0);

2427         return (1);
2428         }

2430 /*
2431  * Return the session to the pool. Calling C_EncryptFinal() and C_DecryptFinal()
2432  * here is the right thing because in EVP_DecryptFinal_ex(), engine's
2433  * do_cipher() is not even called, and in EVP_EncryptFinal_ex() it is called but
2434  * the engine can't find out that it's the finalizing call. We wouldn't
2435  * necessarily have to finalize the context here since reinitializing it with
2436  * C_(Encrypt|Decrypt)Init() should be fine but for the sake of correctness,
2437  * let's do it. Some implementations might leak memory if the previously used
```



```
2438  * context is initialized without finalizing it first.
2439  */
2440 static int
2441 pk11_cipher_cleanup(EVP_CIPHER_CTX *ctx)
2442         {
2443         CK_RV rv;
2444         CK_ULONG len = EVP_MAX_BLOCK_LENGTH;
2445         CK_BYTE buf[EVP_MAX_BLOCK_LENGTH];
2446         PK11_CIPHER_STATE *state = ctx->cipher_data;

2448         if (state != NULL && state->sp != NULL)
2449                 {
2450                 /*
2451                  * We are not interested in the data here, we just need to get
2452                  * rid of the context.
2453                  */
2454                 if (ctx->encrypt)
2455                         rv = pFuncList->C_EncryptFinal(
2456                             state->sp->session, buf, &len);
2457                 else
2458                         rv = pFuncList->C_DecryptFinal(
2459                             state->sp->session, buf, &len);

2461                 if (rv != CKR_OK)
2462                         {
2463                         PK11err_add_data(PK11_F_CIPHER_CLEANUP, ctx->encrypt ?
2464                             PK11_R_ENCRYPTFINAL : PK11_R_DECRYPTFINAL, rv);
2465                         pk11_return_session(state->sp, OP_CIPHER);
2466                         return (0);
2467                         }

2469                 pk11_return_session(state->sp, OP_CIPHER);
2470                 state->sp = NULL;
2471                 }

2473         return (1);
2474         }

2476 /*
2477  * Registered by the ENGINE when used to find out how to deal with
2478  * a particular NID in the ENGINE. This says what we'll do at the
2479  * top level - note, that list is restricted by what we answer with
2480  */
2481 /* ARGSUSED */
2482 static int
2483 pk11_engine_ciphers(ENGINE *e, const EVP_CIPHER **cipher,
2484         const int **nids, int nid)
2485         {
2486         if (!cipher)
2487                 return (pk11_usable_ciphers(nids));

2489         switch (nid)
2490                 {
2491                 case NID_des_ede3_cbc:
2492                         *cipher = &pk11_3des_cbc;
2493                         break;
2494                 case NID_des_cbc:
2495                         *cipher = &pk11_des_cbc;
2496                         break;
2497                 case NID_des_ede3_ecb:
2498                         *cipher = &pk11_3des_ecb;
2499                         break;
2500                 case NID_des_ecb:
2501                         *cipher = &pk11_des_ecb;
2502                         break;
2503                 case NID_aes_128_cbc:
```

```
2504                        *cipher = &pk11_aes_128_cbc;
2505                        break;
2506                case NID_aes_192_cbc:
2507                        *cipher = &pk11_aes_192_cbc;
2508                        break;
2509                case NID_aes_256_cbc:
2510                        *cipher = &pk11_aes_256_cbc;
2511                        break;
2512                case NID_aes_128_ecb:
2513                        *cipher = &pk11_aes_128_ecb;
2514                        break;
2515                case NID_aes_192_ecb:
2516                        *cipher = &pk11_aes_192_ecb;
2517                        break;
2518                case NID_aes_256_ecb:
2519                        *cipher = &pk11_aes_256_ecb;
2520                        break;
2521                case NID_bf_cbc:
2522                        *cipher = &pk11_bf_cbc;
2523                        break;
2524                case NID_rc4:
2525                        *cipher = &pk11_rc4;
2526                        break;
2527                default:
2528 #ifdef  SOLARIS_AES_CTR
2529                        /*
2530                         * These can't be in separated cases because the NIDs
2531                         * here are not constants.
2532                         */
2533                        if (nid == NID_aes_128_ctr)
2534                                *cipher = &pk11_aes_128_ctr;
2535                        else if (nid == NID_aes_192_ctr)
2536                                *cipher = &pk11_aes_192_ctr;
2537                        else if (nid == NID_aes_256_ctr)
2538                                *cipher = &pk11_aes_256_ctr;
2539                        else
2540 #endif  /* SOLARIS_AES_CTR */
2541                        *cipher = NULL;
2542                        break;
2543                }
2544        return (*cipher != NULL);
2545        }

2547 /* ARGSUSED */
2548 static int
2549 pk11_engine_digests(ENGINE *e, const EVP_MD **digest,
2550        const int **nids, int nid)
2551        {
2552        if (!digest)
2553                return (pk11_usable_digests(nids));

2555        switch (nid)
2556                {
2557                case NID_md5:
2558                        *digest = &pk11_md5;
2559                        break;
2560                case NID_sha1:
2561                        *digest = &pk11_sha1;
2562                        break;
2563                case NID_sha224:
2564                        *digest = &pk11_sha224;
2565                        break;
2566                case NID_sha256:
2567                        *digest = &pk11_sha256;
2568                        break;
2569                case NID_sha384:
```



```
2570                        *digest = &pk11_sha384;
2571                        break;
2572                case NID_sha512:
2573                        *digest = &pk11_sha512;
2574                        break;
2575                default:
2576                        *digest = NULL;
2577                        break;
2578                }
2579        return (*digest != NULL);
2580        }


2583 /* Create a secret key object in a PKCS#11 session */
2584 static CK_OBJECT_HANDLE pk11_get_cipher_key(EVP_CIPHER_CTX *ctx,
2585        const unsigned char *key, CK_KEY_TYPE key_type, PK11_SESSION *sp)
2586        {
2587        CK_RV rv;
2588        CK_OBJECT_HANDLE h_key = CK_INVALID_HANDLE;
2589        CK_OBJECT_CLASS obj_key = CKO_SECRET_KEY;
2590        CK_ULONG ul_key_attr_count = 6;

2592        CK_ATTRIBUTE  a_key_template[] =
2593                {
2594                {CKA_CLASS, (void*) NULL, sizeof (CK_OBJECT_CLASS)},
2595                {CKA_KEY_TYPE, (void*) NULL, sizeof (CK_KEY_TYPE)},
2596                {CKA_TOKEN, &false, sizeof (false)},
2597                {CKA_ENCRYPT, &true, sizeof (true)},
2598                {CKA_DECRYPT, &true, sizeof (true)},
2599                {CKA_VALUE, (void*) NULL, 0},
2600                };

2602        /*
2603         * Create secret key object in global_session. All other sessions
2604         * can use the key handles. Here is why:
2605         * OpenSSL will call EncryptInit and EncryptUpdate using a secret key.
2606         * It may then call DecryptInit and DecryptUpdate using the same key.
2607         * To use the same key object, we need to call EncryptFinal with
2608         * a 0 length message. Currently, this does not work for 3DES
2609         * mechanism. To get around this problem, we close the session and
2610         * then create a new session to use the same key object. When a session
2611         * is closed, all the object handles will be invalid. Thus, create key
2612         * objects in a global session, an individual session may be closed to
2613         * terminate the active operation.
2614         */
2615        CK_SESSION_HANDLE session = global_session;
2616        a_key_template[0].pValue = &obj_key;
2617        a_key_template[1].pValue = &key_type;
2618        a_key_template[5].pValue = (void *) key;
2619        a_key_template[5].ulValueLen = (unsigned long) ctx->key_len;

2621        rv = pFuncList->C_CreateObject(session,
2622                a_key_template, ul_key_attr_count, &h_key);
2623        if (rv != CKR_OK)
2624                {
2625                PK11err_add_data(PK11_F_GET_CIPHER_KEY, PK11_R_CREATEOBJECT,
2626                    rv);
2627                goto err;
2628                }

2630        /*
2631         * Save the key information used in this session.
2632         * The max can be saved is PK11_KEY_LEN_MAX.
2633         */
2634        sp->opdata_key_len = ctx->key_len > PK11_KEY_LEN_MAX ?
2635                PK11_KEY_LEN_MAX : ctx->key_len;
```

```
2636         (void) memcpy(sp->opdata_key, key, sp->opdata_key_len);
2637 err:

2639         return (h_key);
2640         }

2642 static int
2643 md_nid_to_pk11(int nid)
2644         {
2645         int i;

2647         for (i = 0; i < PK11_DIGEST_MAX; i++)
2648                 if (digests[i].nid == nid)
2649                         return (digests[i].id);
2650         return (-1);
2651         }

2653 static int
2654 pk11_digest_init(EVP_MD_CTX *ctx)
2655         {
2656         CK_RV rv;
2657         CK_MECHANISM mech;
2658         int index;
2659         PK11_SESSION *sp;
2660         PK11_DIGEST *pdp;
2661         PK11_CIPHER_STATE *state = (PK11_CIPHER_STATE *) ctx->md_data;

2663         state->sp = NULL;

2665         index = md_nid_to_pk11(ctx->digest->type);
2666         if (index < 0 || index >= PK11_DIGEST_MAX)
2667                 return (0);

2669         pdp = &digests[index];
2670         if ((sp = pk11_get_session(OP_DIGEST)) == NULL)
2671                 return (0);

2673         /* at present, no parameter is needed for supported digests */
2674         mech.mechanism = pdp->mech_type;
2675         mech.pParameter = NULL;
2676         mech.ulParameterLen = 0;

2678         rv = pFuncList->C_DigestInit(sp->session, &mech);

2680         if (rv != CKR_OK)
2681                 {
2682                 PK11err_add_data(PK11_F_DIGEST_INIT, PK11_R_DIGESTINIT, rv);
2683                 pk11_return_session(sp, OP_DIGEST);
2684                 return (0);
2685                 }

2687         state->sp = sp;

2689         return (1);
2690         }

2692 static int
2693 pk11_digest_update(EVP_MD_CTX *ctx, const void *data, size_t count)
2694         {
2695         CK_RV rv;
2696         PK11_CIPHER_STATE *state = (PK11_CIPHER_STATE *) ctx->md_data;

2698         /* 0 length message will cause a failure in C_DigestFinal */
2699         if (count == 0)
2700                 return (1);
```



```
2702         if (state == NULL || state->sp == NULL)
2703                 return (0);

2705         rv = pFuncList->C_DigestUpdate(state->sp->session, (CK_BYTE *) data,
2706                 count);

2708         if (rv != CKR_OK)
2709                 {
2710                 PK11err_add_data(PK11_F_DIGEST_UPDATE, PK11_R_DIGESTUPDATE, rv);
2711                 pk11_return_session(state->sp, OP_DIGEST);
2712                 state->sp = NULL;
2713                 return (0);
2714                 }

2716         return (1);
2717         }

2719 static int
2720 pk11_digest_final(EVP_MD_CTX *ctx, unsigned char *md)
2721         {
2722         CK_RV rv;
2723         unsigned long len;
2724         PK11_CIPHER_STATE *state = (PK11_CIPHER_STATE *) ctx->md_data;
2725         len = ctx->digest->md_size;

2727         if (state == NULL || state->sp == NULL)
2728                 return (0);

2730         rv = pFuncList->C_DigestFinal(state->sp->session, md, &len);

2732         if (rv != CKR_OK)
2733                 {
2734                 PK11err_add_data(PK11_F_DIGEST_FINAL, PK11_R_DIGESTFINAL, rv);
2735                 pk11_return_session(state->sp, OP_DIGEST);
2736                 state->sp = NULL;
2737                 return (0);
2738                 }

2740         if (ctx->digest->md_size != len)
2741                 return (0);

2743         /*
2744          * Final is called and digest is returned, so return the session
2745          * to the pool
2746          */
2747         pk11_return_session(state->sp, OP_DIGEST);
2748         state->sp = NULL;

2750         return (1);
2751         }

2753 static int
2754 pk11_digest_copy(EVP_MD_CTX *to, const EVP_MD_CTX *from)
2755         {
2756         CK_RV rv;
2757         int ret = 0;
2758         PK11_CIPHER_STATE *state, *state_to;
2759         CK_BYTE_PTR pstate = NULL;
2760         CK_ULONG ul_state_len;

2762         /* The copy-from state */
2763         state = (PK11_CIPHER_STATE *) from->md_data;
2764         if (state == NULL || state->sp == NULL)
2765                 goto err;

2767         /* Initialize the copy-to state */
```

```
2768         if (!pk11_digest_init(to))
2769                 goto err;
2770         state_to = (PK11_CIPHER_STATE *) to->md_data;

2772         /* Get the size of the operation state of the copy-from session */
2773         rv = pFuncList->C_GetOperationState(state->sp->session, NULL,
2774                 &ul_state_len);

2776         if (rv != CKR_OK)
2777                 {
2778                 PK11err_add_data(PK11_F_DIGEST_COPY, PK11_R_GET_OPERATION_STATE,
2779                     rv);
2780                 goto err;
2781                 }
2782         if (ul_state_len == 0)
2783                 {
2784                 goto err;
2785                 }

2787         pstate = OPENSSL_malloc(ul_state_len);
2788         if (pstate == NULL)
2789                 {
2790                 PK11err(PK11_F_DIGEST_COPY, PK11_R_MALLOC_FAILURE);
2791                 goto err;
2792                 }

2794         /* Get the operation state of the copy-from session */
2795         rv = pFuncList->C_GetOperationState(state->sp->session, pstate,
2796                 &ul_state_len);

2798         if (rv != CKR_OK)
2799                 {
2800                 PK11err_add_data(PK11_F_DIGEST_COPY, PK11_R_GET_OPERATION_STATE,
2801                     rv);
2802                 goto err;
2803                 }

2805         /* Set the operation state of the copy-to session */
2806         rv = pFuncList->C_SetOperationState(state_to->sp->session, pstate,
2807                 ul_state_len, 0, 0);

2809         if (rv != CKR_OK)
2810                 {
2811                 PK11err_add_data(PK11_F_DIGEST_COPY,
2812                     PK11_R_SET_OPERATION_STATE, rv);
2813                 goto err;
2814                 }

2816         ret = 1;
2817 err:
2818         if (pstate != NULL)
2819                 OPENSSL_free(pstate);

2821         return (ret);
2822         }

2824 /* Return any pending session state to the pool */
2825 static int
2826 pk11_digest_cleanup(EVP_MD_CTX *ctx)
2827         {
2828         PK11_CIPHER_STATE *state = ctx->md_data;
2829         unsigned char buf[EVP_MAX_MD_SIZE];

2831         if (state != NULL && state->sp != NULL)
2832                 {
2833                 /*
```



```
2834                  * If state->sp is not NULL then pk11_digest_final() has not
2835                  * been called yet. We must call it now to free any memory
2836                  * that might have been allocated in the token when
2837                  * pk11_digest_init() was called. pk11_digest_final()
2838                  * will return the session to the cache.
2839                  */
2840                 if (!pk11_digest_final(ctx, buf))
2841                         return (0);
2842                 }

2844         return (1);
2845         }

2847 /*
2848  * Check if the new key is the same as the key object in the session. If the key
2849  * is the same, no need to create a new key object. Otherwise, the old key
2850  * object needs to be destroyed and a new one will be created. Return 1 for
2851  * cache hit, 0 for cache miss. Note that we must check the key length first
2852  * otherwise we could end up reusing a different, longer key with the same
2853  * prefix.
2854  */
2855 static int check_new_cipher_key(PK11_SESSION *sp, const unsigned char *key,
2856         int key_len)
2857         {
2858         if (sp->opdata_key_len != key_len ||
2859             memcmp(sp->opdata_key, key, key_len) != 0)
2860                 {
2861                 (void) pk11_destroy_cipher_key_objects(sp);
2862                 return (0);
2863                 }
2864         return (1);
2865         }

2867 /* Destroy one or more secret key objects. */
2868 static int pk11_destroy_cipher_key_objects(PK11_SESSION *session)
2869         {
2870         int ret = 0;
2871         PK11_SESSION *sp = NULL;
2872         PK11_SESSION *local_free_session;

2874         if (session != NULL)
2875                 local_free_session = session;
2876         else
2877                 {
2878                 (void) pthread_mutex_lock(session_cache[OP_CIPHER].lock);
2879                 local_free_session = session_cache[OP_CIPHER].head;
2880                 }

2882         while ((sp = local_free_session) != NULL)
2883                 {
2884                 local_free_session = sp->next;

2886                 if (sp->opdata_cipher_key != CK_INVALID_HANDLE)
2887                         {
2888                         /*
2889                          * The secret key object is created in the
2890                          * global_session. See pk11_get_cipher_key
2891                          */
2892                         if (pk11_destroy_object(global_session,
2893                                 sp->opdata_cipher_key) == 0)
2894                                 goto err;
2895                         sp->opdata_cipher_key = CK_INVALID_HANDLE;
2896                         }
2897                 }
2898         ret = 1;
2899 err:
```

```
2901         if (session == NULL)
2902                 (void) pthread_mutex_unlock(session_cache[OP_CIPHER].lock);

2904         return (ret);
2905         }


2908 /*
2909  * Public key mechanisms optionally supported
2910  *
2911  * CKM_RSA_X_509
2912  * CKM_RSA_PKCS
2913  * CKM_DSA
2914  *
2915  * The first slot that supports at least one of those mechanisms is chosen as a
2916  * public key slot.
2917  *
2918  * Symmetric ciphers optionally supported
2919  *
2920  * CKM_DES3_CBC
2921  * CKM_DES_CBC
2922  * CKM_AES_CBC
2923  * CKM_DES3_ECB
2924  * CKM_DES_ECB
2925  * CKM_AES_ECB
2926  * CKM_AES_CTR
2927  * CKM_RC4
2928  * CKM_BLOWFISH_CBC
2929  *
2930  * Digests optionally supported
2931  *
2932  * CKM_MD5
2933  * CKM_SHA_1
2934  * CKM_SHA224
2935  * CKM_SHA256
2936  * CKM_SHA384
2937  * CKM_SHA512
2938  *
2939  * The output of this function is a set of global variables indicating which
2940  * mechanisms from RSA, DSA, DH and RAND are present, and also two arrays of
2941  * mechanisms, one for symmetric ciphers and one for digests. Also, 3 global
2942  * variables carry information about which slot was chosen for (a) public key
2943  * mechanisms, (b) random operations, and (c) symmetric ciphers and digests.
2944  */
2945 static int
2946 pk11_choose_slots(int *any_slot_found)
2947         {
2948         CK_SLOT_ID_PTR pSlotList = NULL_PTR;
2949         CK_ULONG ulSlotCount = 0;
2950         CK_MECHANISM_INFO mech_info;
2951         CK_TOKEN_INFO token_info;
2952         int i;
2953         CK_RV rv;
2954         CK_SLOT_ID best_slot_sofar = 0;
2955         CK_BBOOL found_candidate_slot = CK_FALSE;
2956         int slot_n_cipher = 0;
2957         int slot_n_digest = 0;
2958         CK_SLOT_ID current_slot = 0;
2959         int current_slot_n_cipher = 0;
2960         int current_slot_n_digest = 0;

2962         int local_cipher_nids[PK11_CIPHER_MAX];
2963         int local_digest_nids[PK11_DIGEST_MAX];

2965         /* let's initialize the output parameter */
```



```
2966         if (any_slot_found != NULL)
2967                 *any_slot_found = 0;

2969         /* Get slot list for memory allocation */
2970         rv = pFuncList->C_GetSlotList(0, NULL_PTR, &ulSlotCount);

2972         if (rv != CKR_OK)
2973                 {
2974                 PK11err_add_data(PK11_F_CHOOSE_SLOT, PK11_R_GETSLOTLIST, rv);
2975                 return (0);
2976                 }

2978         /* it's not an error if we didn't find any providers */
2979         if (ulSlotCount == 0)
2980                 {
2981 #ifdef  DEBUG_SLOT_SELECTION
2982                 fprintf(stderr, "%s: no crypto providers found\n", PK11_DBG);
2983 #endif  /* DEBUG_SLOT_SELECTION */
2984                 return (1);
2985                 }

2987         pSlotList = OPENSSL_malloc(ulSlotCount * sizeof (CK_SLOT_ID));

2989         if (pSlotList == NULL)
2990                 {
2991                 PK11err(PK11_F_CHOOSE_SLOT, PK11_R_MALLOC_FAILURE);
2992                 return (0);
2993                 }

2995         /* Get the slot list for processing */
2996         rv = pFuncList->C_GetSlotList(0, pSlotList, &ulSlotCount);
2997         if (rv != CKR_OK)
2998                 {
2999                 PK11err_add_data(PK11_F_CHOOSE_SLOT, PK11_R_GETSLOTLIST, rv);
3000                 OPENSSL_free(pSlotList);
3001                 return (0);
3002                 }

3004 #ifdef  DEBUG_SLOT_SELECTION
3005         fprintf(stderr, "%s: provider: %s\n", PK11_DBG, def_PK11_LIBNAME);
3006         fprintf(stderr, "%s: number of slots: %d\n", PK11_DBG, ulSlotCount);

3008         fprintf(stderr, "%s: == checking rand slots ==\n", PK11_DBG);
3009 #endif  /* DEBUG_SLOT_SELECTION */
3010         for (i = 0; i < ulSlotCount; i++)
3011                 {
3012                 current_slot = pSlotList[i];

3014 #ifdef  DEBUG_SLOT_SELECTION
3015         fprintf(stderr, "%s: checking slot: %d\n", PK11_DBG, i);
3016 #endif  /* DEBUG_SLOT_SELECTION */
3017                 /* Check if slot has random support. */
3018                 rv = pFuncList->C_GetTokenInfo(current_slot, &token_info);
3019                 if (rv != CKR_OK)
3020                         continue;

3022 #ifdef  DEBUG_SLOT_SELECTION
3023         fprintf(stderr, "%s: token label: %.32s\n", PK11_DBG, token_info.label);
3024 #endif  /* DEBUG_SLOT_SELECTION */

3026                 if (token_info.flags & CKF_RNG)
3027                         {
3028 #ifdef  DEBUG_SLOT_SELECTION
3029         fprintf(stderr, "%s: this token has CKF_RNG flag\n", PK11_DBG);
3030 #endif  /* DEBUG_SLOT_SELECTION */
3031                         pk11_have_random = CK_TRUE;
```

```
3032                        rand_SLOTID = current_slot;
3033                        break;
3034                        }
3035                }

3037 #ifdef  DEBUG_SLOT_SELECTION
3038         fprintf(stderr, "%s: == checking pubkey slots ==\n", PK11_DBG);
3039 #endif  /* DEBUG_SLOT_SELECTION */

3041         pubkey_SLOTID = pSlotList[0];
3042         for (i = 0; i < ulSlotCount; i++)
3043                 {
3044                 CK_BBOOL slot_has_rsa = CK_FALSE;
3045                 CK_BBOOL slot_has_dsa = CK_FALSE;
3046                 CK_BBOOL slot_has_dh = CK_FALSE;
3047                 current_slot = pSlotList[i];

3049 #ifdef  DEBUG_SLOT_SELECTION
3050         fprintf(stderr, "%s: checking slot: %d\n", PK11_DBG, i);
3051 #endif  /* DEBUG_SLOT_SELECTION */
3052                 rv = pFuncList->C_GetTokenInfo(current_slot, &token_info);
3053                 if (rv != CKR_OK)
3054                         continue;

3056 #ifdef  DEBUG_SLOT_SELECTION
3057         fprintf(stderr, "%s: token label: %.32s\n", PK11_DBG, token_info.label);
3058 #endif  /* DEBUG_SLOT_SELECTION */

3060 #ifndef OPENSSL_NO_RSA
3061                 /*
3062                  * Check if this slot is capable of signing and
3063                  * verifying with CKM_RSA_PKCS.
3064                  */
3065                 rv = pFuncList->C_GetMechanismInfo(current_slot, CKM_RSA_PKCS,
3066                         &mech_info);

3068                 if (rv == CKR_OK && ((mech_info.flags & CKF_SIGN) &&
3069                                 (mech_info.flags & CKF_VERIFY)))
3070                         {
3071                         /*
3072                          * Check if this slot is capable of encryption,
3073                          * decryption, sign, and verify with CKM_RSA_X_509.
3074                          */
3075                         rv = pFuncList->C_GetMechanismInfo(current_slot,
3076                             CKM_RSA_X_509, &mech_info);

3078                         if (rv == CKR_OK && ((mech_info.flags & CKF_SIGN) &&
3079                             (mech_info.flags & CKF_VERIFY) &&
3080                             (mech_info.flags & CKF_ENCRYPT) &&
3081                             (mech_info.flags & CKF_VERIFY_RECOVER) &&
3082                             (mech_info.flags & CKF_DECRYPT)))
3083                                 {
3084                                 slot_has_rsa = CK_TRUE;
3085                                 }
3086                         }
3087 #endif  /* OPENSSL_NO_RSA */

3089 #ifndef OPENSSL_NO_DSA
3090                 /*
3091                  * Check if this slot is capable of signing and
3092                  * verifying with CKM_DSA.
3093                  */
3094                 rv = pFuncList->C_GetMechanismInfo(current_slot, CKM_DSA,
3095                         &mech_info);
3096                 if (rv == CKR_OK && ((mech_info.flags & CKF_SIGN) &&
3097                     (mech_info.flags & CKF_VERIFY)))
```



```
3098                         {
3099                         slot_has_dsa = CK_TRUE;
3100                         }

3102 #endif  /* OPENSSL_NO_DSA */

3104 #ifndef OPENSSL_NO_DH
3105                 /*
3106                  * Check if this slot is capable of DH key generataion and
3107                  * derivation.
3108                  */
3109                 rv = pFuncList->C_GetMechanismInfo(current_slot,
3110                     CKM_DH_PKCS_KEY_PAIR_GEN, &mech_info);

3112                 if (rv == CKR_OK && (mech_info.flags & CKF_GENERATE_KEY_PAIR))
3113                         {
3114                         rv = pFuncList->C_GetMechanismInfo(current_slot,
3115                                 CKM_DH_PKCS_DERIVE, &mech_info);
3116                         if (rv == CKR_OK && (mech_info.flags & CKF_DERIVE))
3117                                 {
3118                                 slot_has_dh = CK_TRUE;
3119                                 }
3120                         }
3121 #endif  /* OPENSSL_NO_DH */

3123                 if (!found_candidate_slot &&
3124                     (slot_has_rsa || slot_has_dsa || slot_has_dh))
3125                         {
3126 #ifdef  DEBUG_SLOT_SELECTION
3127                         fprintf(stderr,
3128                             "%s: potential slot: %d\n", PK11_DBG, current_slot);
3129 #endif  /* DEBUG_SLOT_SELECTION */
3130                         best_slot_sofar = current_slot;
3131                         pk11_have_rsa = slot_has_rsa;
3132                         pk11_have_dsa = slot_has_dsa;
3133                         pk11_have_dh = slot_has_dh;
3134                         found_candidate_slot = CK_TRUE;
3135 #ifdef  DEBUG_SLOT_SELECTION
3136                         fprintf(stderr,
3137                             "%s: setting found_candidate_slot to CK_TRUE\n",
3138                             PK11_DBG);
3139                         fprintf(stderr,
3140                             "%s: best so far slot: %d\n", PK11_DBG,
3141                             best_slot_sofar);
3142                         }
3143                 else
3144                         {
3145                         fprintf(stderr,
3146                             "%s: no rsa/dsa/dh\n", PK11_DBG);
3147                         }
3148 #else
3149                         } /* if */
3150 #endif  /* DEBUG_SLOT_SELECTION */
3151                 } /* for */

3153         if (found_candidate_slot)
3154                 {
3155                 pubkey_SLOTID = best_slot_sofar;
3156                 }

3158         found_candidate_slot = CK_FALSE;
3159         best_slot_sofar = 0;

3161 #ifdef  DEBUG_SLOT_SELECTION
3162         fprintf(stderr, "%s: == checking cipher/digest ==\n", PK11_DBG);
3163 #endif  /* DEBUG_SLOT_SELECTION */
```

```
3165         SLOTID = pSlotList[0];
3166         for (i = 0; i < ulSlotCount; i++)
3167                 {
3168 #ifdef  DEBUG_SLOT_SELECTION
3169         fprintf(stderr, "%s: checking slot: %d\n", PK11_DBG, i);
3170 #endif  /* DEBUG_SLOT_SELECTION */

3172                 current_slot = pSlotList[i];
3173                 current_slot_n_cipher = 0;
3174                 current_slot_n_digest = 0;
3175                 (void) memset(local_cipher_nids, 0, sizeof (local_cipher_nids));
3176                 (void) memset(local_digest_nids, 0, sizeof (local_digest_nids));

3178                 pk11_find_symmetric_ciphers(pFuncList, current_slot,
3179                     &current_slot_n_cipher, local_cipher_nids);

3181                 pk11_find_digests(pFuncList, current_slot,
3182                     &current_slot_n_digest, local_digest_nids);

3184 #ifdef  DEBUG_SLOT_SELECTION
3185                 fprintf(stderr, "%s: current_slot_n_cipher %d\n", PK11_DBG,
3186                         current_slot_n_cipher);
3187                 fprintf(stderr, "%s: current_slot_n_digest %d\n", PK11_DBG,
3188                         current_slot_n_digest);
3189                 fprintf(stderr, "%s: best so far cipher/digest slot: %d\n",
3190                         PK11_DBG, best_slot_sofar);
3191 #endif  /* DEBUG_SLOT_SELECTION */

3193                 /*
3194                  * If the current slot supports more ciphers/digests than
3195                  * the previous best one we change the current best to this one,
3196                  * otherwise leave it where it is.
3197                  */
3198                 if ((current_slot_n_cipher + current_slot_n_digest) >
3199                     (slot_n_cipher + slot_n_digest))
3200                         {
3201 #ifdef  DEBUG_SLOT_SELECTION
3202                         fprintf(stderr,
3203                                 "%s: changing best so far slot to %d\n",
3204                                 PK11_DBG, current_slot);
3205 #endif  /* DEBUG_SLOT_SELECTION */
3206                         best_slot_sofar = SLOTID = current_slot;
3207                         cipher_count = slot_n_cipher = current_slot_n_cipher;
3208                         digest_count = slot_n_digest = current_slot_n_digest;
3209                         (void) memcpy(cipher_nids, local_cipher_nids,
3210                             sizeof (local_cipher_nids));
3211                         (void) memcpy(digest_nids, local_digest_nids,
3212                             sizeof (local_digest_nids));
3213                         }
3214                 }

3216 #ifdef  DEBUG_SLOT_SELECTION
3217         fprintf(stderr,
3218             "%s: chosen pubkey slot: %d\n", PK11_DBG, pubkey_SLOTID);
3219         fprintf(stderr,
3220             "%s: chosen rand slot: %d\n", PK11_DBG, rand_SLOTID);
3221         fprintf(stderr,
3222             "%s: chosen cipher/digest slot: %d\n", PK11_DBG, SLOTID);
3223         fprintf(stderr,
3224             "%s: pk11_have_rsa %d\n", PK11_DBG, pk11_have_rsa);
3225         fprintf(stderr,
3226             "%s: pk11_have_dsa %d\n", PK11_DBG, pk11_have_dsa);
3227         fprintf(stderr,
3228             "%s: pk11_have_dh %d\n", PK11_DBG, pk11_have_dh);
3229         fprintf(stderr,
```



```
3230             "%s: pk11_have_random %d\n", PK11_DBG, pk11_have_random);
3231         fprintf(stderr,
3232             "%s: cipher_count %d\n", PK11_DBG, cipher_count);
3233         fprintf(stderr,
3234             "%s: digest_count %d\n", PK11_DBG, digest_count);
3235 #endif  /* DEBUG_SLOT_SELECTION */

3237         if (pSlotList != NULL)
3238                 OPENSSL_free(pSlotList);

3240 #ifdef  SOLARIS_HW_SLOT_SELECTION
3241         OPENSSL_free(hw_cnids);
3242         OPENSSL_free(hw_dnids);
3243 #endif  /* SOLARIS_HW_SLOT_SELECTION */

3245         if (any_slot_found != NULL)
3246                 *any_slot_found = 1;
3247         return (1);
3248         }

3250 static void pk11_get_symmetric_cipher(CK_FUNCTION_LIST_PTR pflist,
3251     int slot_id, CK_MECHANISM_TYPE mech, int *current_slot_n_cipher,
3252     int *local_cipher_nids, int id)
3253         {
3254         CK_MECHANISM_INFO mech_info;
3255         CK_RV rv;

3257 #ifdef  DEBUG_SLOT_SELECTION
3258         fprintf(stderr, "%s: checking mech: %x", PK11_DBG, mech);
3259 #endif  /* DEBUG_SLOT_SELECTION */
3260         rv = pflist->C_GetMechanismInfo(slot_id, mech, &mech_info);

3262         if (rv != CKR_OK)
3263                 {
3264 #ifdef  DEBUG_SLOT_SELECTION
3265                 fprintf(stderr, " not found\n");
3266 #endif  /* DEBUG_SLOT_SELECTION */
3267                 return;
3268                 }

3270         if ((mech_info.flags & CKF_ENCRYPT) &&
3271             (mech_info.flags & CKF_DECRYPT))
3272                 {
3273 #ifdef  SOLARIS_HW_SLOT_SELECTION
3274                 if (nid_in_table(ciphers[id].nid, hw_cnids))
3275 #endif  /* SOLARIS_HW_SLOT_SELECTION */
3276                         {
3277 #ifdef  DEBUG_SLOT_SELECTION
3278                         fprintf(stderr, " usable\n");
3279 #endif  /* DEBUG_SLOT_SELECTION */
3280                         local_cipher_nids[(*current_slot_n_cipher)++] =
3281                             ciphers[id].nid;
3282                         }
3283 #ifdef  SOLARIS_HW_SLOT_SELECTION
3284 #ifdef  DEBUG_SLOT_SELECTION
3285                 else
3286                         {
3287                         fprintf(stderr, " rejected, software implementation only\n");
3288                         }
3289 #endif  /* DEBUG_SLOT_SELECTION */
3290 #endif  /* SOLARIS_HW_SLOT_SELECTION */
3291                 }
3292 #ifdef  DEBUG_SLOT_SELECTION
3293         else
3294                 {
3295                 fprintf(stderr, " unusable\n");
```

```
3296                 }
3297 #endif  /* DEBUG_SLOT_SELECTION */

3299         return;
3300         }

3302 static void pk11_get_digest(CK_FUNCTION_LIST_PTR pflist, int slot_id,
3303     CK_MECHANISM_TYPE mech, int *current_slot_n_digest, int *local_digest_nids,
3304     int id)
3305         {
3306         CK_MECHANISM_INFO mech_info;
3307         CK_RV rv;

3309 #ifdef  DEBUG_SLOT_SELECTION
3310         fprintf(stderr, "%s: checking mech: %x", PK11_DBG, mech);
3311 #endif  /* DEBUG_SLOT_SELECTION */
3312         rv = pflist->C_GetMechanismInfo(slot_id, mech, &mech_info);

3314         if (rv != CKR_OK)
3315                 {
3316 #ifdef  DEBUG_SLOT_SELECTION
3317                 fprintf(stderr, " not found\n");
3318 #endif  /* DEBUG_SLOT_SELECTION */
3319                 return;
3320                 }

3322         if (mech_info.flags & CKF_DIGEST)
3323                 {
3324 #ifdef  SOLARIS_HW_SLOT_SELECTION
3325                 if (nid_in_table(digests[id].nid, hw_dnids))
3326 #endif  /* SOLARIS_HW_SLOT_SELECTION */
3327                         {
3328 #ifdef  DEBUG_SLOT_SELECTION
3329                 fprintf(stderr, " usable\n");
3330 #endif  /* DEBUG_SLOT_SELECTION */
3331                         local_digest_nids[(*current_slot_n_digest)++] =
3332                             digests[id].nid;
3333                         }
3334 #ifdef  SOLARIS_HW_SLOT_SELECTION
3335 #ifdef  DEBUG_SLOT_SELECTION
3336                 else
3337                         {
3338                 fprintf(stderr, " rejected, software implementation only\n");
3339                         }
3340 #endif  /* DEBUG_SLOT_SELECTION */
3341 #endif  /* SOLARIS_HW_SLOT_SELECTION */
3342                 }
3343 #ifdef  DEBUG_SLOT_SELECTION
3344         else
3345                 {
3346                 fprintf(stderr, " unusable\n");
3347                 }
3348 #endif  /* DEBUG_SLOT_SELECTION */

3350         return;
3351         }

3353 #ifdef  SOLARIS_AES_CTR
3354 /*
3355  * Create new NIDs for AES counter mode. OpenSSL doesn't support them now so we
3356  * have to help ourselves here.
3357  */
3358 static int pk11_add_aes_ctr_NIDs(void)
3359         {
3360         /* are we already set? */
3361         if (NID_aes_256_ctr != NID_undef)
```



```
3362                 return (1);

3364         /*
3365          * There are no official names for AES counter modes yet so we just
3366          * follow the format of those that exist.
3367          */
3368         ciphers[PK11_AES_128_CTR].nid = pk11_aes_128_ctr.nid = NID_aes_128_ctr;
3369         ciphers[PK11_AES_192_CTR].nid = pk11_aes_192_ctr.nid = NID_aes_192_ctr;
3370         ciphers[PK11_AES_256_CTR].nid = pk11_aes_256_ctr.nid = NID_aes_256_ctr;
3371         return (1);

3373         }
3374 #endif  /* SOLARIS_AES_CTR */

3376 /* Find what symmetric ciphers this slot supports. */
3377 static void pk11_find_symmetric_ciphers(CK_FUNCTION_LIST_PTR pflist,
3378     CK_SLOT_ID current_slot, int *current_slot_n_cipher, int *local_cipher_nids)
3379         {
3380         int i;

3382         for (i = 0; i < PK11_CIPHER_MAX; ++i)
3383                 {
3384                 pk11_get_symmetric_cipher(pflist, current_slot,
3385                     ciphers[i].mech_type, current_slot_n_cipher,
3386                     local_cipher_nids, ciphers[i].id);
3387                 }
3388         }

3390 /* Find what digest algorithms this slot supports. */
3391 static void pk11_find_digests(CK_FUNCTION_LIST_PTR pflist,
3392     CK_SLOT_ID current_slot, int *current_slot_n_digest, int *local_digest_nids)
3393         {
3394         int i;

3396         for (i = 0; i < PK11_DIGEST_MAX; ++i)
3397                 {
3398                 pk11_get_digest(pflist, current_slot, digests[i].mech_type,
3399                     current_slot_n_digest, local_digest_nids, digests[i].id);
3400                 }
3401         }

3403 #ifdef  SOLARIS_HW_SLOT_SELECTION
3404 /*
3405  * It would be great if we could use pkcs11_kernel directly since this library
3406  * offers hardware slots only. That's the easiest way to achieve the situation
3407  * where we use the hardware accelerators when present and OpenSSL native code
3408  * otherwise. That presumes the fact that OpenSSL native code is faster than the
3409  * code in the soft token. It's a logical assumption - Crypto Framework has some
3410  * inherent overhead so going there for the software implementation of a
3411  * mechanism should be logically slower in contrast to the OpenSSL native code,
3412  * presuming that both implementations are of similar speed. For example, the
3413  * soft token for AES is roughly three times slower than OpenSSL for 64 byte
3414  * blocks and still 20% slower for 8KB blocks. So, if we want to ship products
3415  * that use the PKCS#11 engine by default, we must somehow avoid that regression
3416  * on machines without hardware acceleration. That's why switching to the
3417  * pkcs11_kernel library seems like a very good idea.
3418  *
3419  * The problem is that OpenSSL built with SunStudio is roughly 2x slower for
3420  * asymmetric operations (RSA/DSA/DH) than the soft token built with the same
3421  * compiler. That means that if we switched to pkcs11_kernel from the libpkcs11
3422  * library, we would have had a performance regression on machines without
3423  * hardware acceleration for asymmetric operations for all applications that use
3424  * the PKCS#11 engine. There is one such application - Apache web server since
3425  * it's shipped configured to use the PKCS#11 engine by default. Having said
3426  * that, we can't switch to the pkcs11_kernel library now and have to come with
3427  * a solution that, on non-accelerated machines, uses the OpenSSL native code
```

```
3428  * for all symmetric ciphers and digests while it uses the soft token for
3429  * asymmetric operations.
3430  *
3431  * This is the idea: dlopen() pkcs11_kernel directly and find out what
3432  * mechanisms are there. We don't care about duplications (more slots can
3433  * support the same mechanism), we just want to know what mechanisms can be
3434  * possibly supported in hardware on that particular machine. As said before,
3435  * pkcs11_kernel will show you hardware providers only.
3436  *
3437  * Then, we rely on the fact that since we use libpkcs11 library we will find
3438  * the metaslot. When we go through the metaslot's mechanisms for symmetric
3439  * ciphers and digests, we check that any found mechanism is in the table
3440  * created using the pkcs11_kernel library. So, as a result we have two arrays
3441  * of mechanisms that were advertised as supported in hardware which was the
3442  * goal of that whole excercise. Thus, we can use libpkcs11 but avoid soft token
3443  * code for symmetric ciphers and digests. See pk11_choose_slots() for more
3444  * information.
3445  *
3446  * This is Solaris specific code, if SOLARIS_HW_SLOT_SELECTION is not defined
3447  * the code won't be used.
3448  */
3449 #if defined(__sparcv9) || defined(__x86_64) || defined(__amd64)
3450 static const char pkcs11_kernel[] = "/usr/lib/security/64/pkcs11_kernel.so.1";
3451 #else
3452 static const char pkcs11_kernel[] = "/usr/lib/security/pkcs11_kernel.so.1";
3453 #endif

3455 /*
3456  * Check hardware capabilities of the machines. The output are two lists,
3457  * hw_cnids and hw_dnids, that contain hardware mechanisms found in all hardware
3458  * providers together. They are not sorted and may contain duplicate mechanisms.
3459  */
3460 static int check_hw_mechanisms(void)
3461         {
3462         int i;
3463         CK_RV rv;
3464         void *handle;
3465         CK_C_GetFunctionList p;
3466         CK_TOKEN_INFO token_info;
3467         CK_ULONG ulSlotCount = 0;
3468         int n_cipher = 0, n_digest = 0;
3469         CK_FUNCTION_LIST_PTR pflist = NULL;
3470         CK_SLOT_ID_PTR pSlotList = NULL_PTR;
3471         int *tmp_hw_cnids=NULL, *tmp_hw_dnids=NULL;
3472         int hw_ctable_size, hw_dtable_size;

3474 #ifdef  DEBUG_SLOT_SELECTION
3475         fprintf(stderr, "%s: SOLARIS_HW_SLOT_SELECTION code running\n",
3476             PK11_DBG);
3477 #endif
3478         if ((handle = dlopen(pkcs11_kernel, RTLD_LAZY)) == NULL)
3479                 {
3480                 PK11err(PK11_F_CHECK_HW_MECHANISMS, PK11_R_DSO_FAILURE);
3481                 goto err;
3482                 }

3484         if ((p = (CK_C_GetFunctionList)dlsym(handle,
3485             PK11_GET_FUNCTION_LIST)) == NULL)
3486                 {
3487                 PK11err(PK11_F_CHECK_HW_MECHANISMS, PK11_R_DSO_FAILURE);
3488                 goto err;
3489                 }

3491         /* get the full function list from the loaded library */
3492         if (p(&pflist) != CKR_OK)
3493                 {
```



```
3494                 PK11err(PK11_F_CHECK_HW_MECHANISMS, PK11_R_DSO_FAILURE);
3495                 goto err;
3496                 }

3498         rv = pflist->C_Initialize(NULL_PTR);
3499         if ((rv != CKR_OK) && (rv != CKR_CRYPTOKI_ALREADY_INITIALIZED))
3500                 {
3501                 PK11err_add_data(PK11_F_CHECK_HW_MECHANISMS,
3502                     PK11_R_INITIALIZE, rv);
3503                 goto err;
3504                 }

3506         if (pflist->C_GetSlotList(0, NULL_PTR, &ulSlotCount) != CKR_OK)
3507                 {
3508                 PK11err(PK11_F_CHECK_HW_MECHANISMS, PK11_R_GETSLOTLIST);
3509                 goto err;
3510                 }

3512         /* no slots, set the hw mechanism tables as empty */
3513         if (ulSlotCount == 0)
3514                 {
3515 #ifdef  DEBUG_SLOT_SELECTION
3516         fprintf(stderr, "%s: no hardware mechanisms found\n", PK11_DBG);
3517 #endif
3518                 hw_cnids = OPENSSL_malloc(sizeof (int));
3519                 hw_dnids = OPENSSL_malloc(sizeof (int));
3520                 if (hw_cnids == NULL || hw_dnids == NULL)
3521                         {
3522                         PK11err(PK11_F_CHECK_HW_MECHANISMS,
3523                             PK11_R_MALLOC_FAILURE);
3524                         return (0);
3525                         }
3526                 /* this means empty tables */
3527                 hw_cnids[0] = NID_undef;
3528                 hw_dnids[0] = NID_undef;
3529                 return (1);
3530                 }

3532         pSlotList = OPENSSL_malloc(ulSlotCount * sizeof (CK_SLOT_ID));
3533         if (pSlotList == NULL)
3534                 {
3535                 PK11err(PK11_F_CHECK_HW_MECHANISMS, PK11_R_MALLOC_FAILURE);
3536                 goto err;
3537                 }

3539         /* Get the slot list for processing */
3540         if (pflist->C_GetSlotList(0, pSlotList, &ulSlotCount) != CKR_OK)
3541                 {
3542                 PK11err(PK11_F_CHECK_HW_MECHANISMS, PK11_R_GETSLOTLIST);
3543                 goto err;
3544                 }

3546         /*
3547          * We don't care about duplicit mechanisms in multiple slots and also
3548          * reserve one slot for the terminal NID_undef which we use to stop the
3549          * search.
3550          */
3551         hw_ctable_size = ulSlotCount * PK11_CIPHER_MAX + 1;
3552         hw_dtable_size = ulSlotCount * PK11_DIGEST_MAX + 1;
3553         tmp_hw_cnids = OPENSSL_malloc(hw_ctable_size * sizeof (int));
3554         tmp_hw_dnids = OPENSSL_malloc(hw_dtable_size * sizeof (int));
3555         if (tmp_hw_cnids == NULL || tmp_hw_dnids == NULL)
3556                 {
3557                 PK11err(PK11_F_CHECK_HW_MECHANISMS, PK11_R_MALLOC_FAILURE);
3558                 goto err;
3559                 }
```

```
3561         /*
3562          * Do not use memset since we should not rely on the fact that NID_undef
3563          * is zero now.
3564          */
3565         for (i = 0; i < hw_ctable_size; ++i)
3566                 tmp_hw_cnids[i] = NID_undef;
3567         for (i = 0; i < hw_dtable_size; ++i)
3568                 tmp_hw_dnids[i] = NID_undef;

3570 #ifdef  DEBUG_SLOT_SELECTION
3571         fprintf(stderr, "%s: provider: %s\n", PK11_DBG, pkcs11_kernel);
3572         fprintf(stderr, "%s: found %d hardware slots\n", PK11_DBG, ulSlotCount);
3573         fprintf(stderr, "%s: now looking for mechs supported in hw\n",
3574             PK11_DBG);
3575 #endif  /* DEBUG_SLOT_SELECTION */

3577         for (i = 0; i < ulSlotCount; i++)
3578                 {
3579                 if (pflist->C_GetTokenInfo(pSlotList[i], &token_info) != CKR_OK)
3580                         continue;

3582 #ifdef  DEBUG_SLOT_SELECTION
3583         fprintf(stderr, "%s: token label: %.32s\n", PK11_DBG, token_info.label);
3584 #endif  /* DEBUG_SLOT_SELECTION */

3586                 /*
3587                  * We are filling the hw mech tables here. Global tables are
3588                  * still NULL so all mechanisms are put into tmp tables.
3589                  */
3590                 pk11_find_symmetric_ciphers(pflist, pSlotList[i],
3591                     &n_cipher, tmp_hw_cnids);
3592                 pk11_find_digests(pflist, pSlotList[i],
3593                     &n_digest, tmp_hw_dnids);
3594                 }

3596         /*
3597          * Since we are part of a library (libcrypto.so), calling this function
3598          * may have side-effects. Also, C_Finalize() is triggered by
3599          * dlclose(3C).
3600          */
3601 #if 0
3602         pflist->C_Finalize(NULL);
3603 #endif
3604         OPENSSL_free(pSlotList);
3605         (void) dlclose(handle);
3606         hw_cnids = tmp_hw_cnids;
3607         hw_dnids = tmp_hw_dnids;

3609 #ifdef  DEBUG_SLOT_SELECTION
3610         fprintf(stderr, "%s: hw mechs check complete\n", PK11_DBG);
3611 #endif  /* DEBUG_SLOT_SELECTION */
3612         return (1);

3614 err:
3615         if (pSlotList != NULL)
3616                 OPENSSL_free(pSlotList);
3617         if (tmp_hw_cnids != NULL)
3618                 OPENSSL_free(tmp_hw_cnids);
3619         if (tmp_hw_dnids != NULL)
3620                 OPENSSL_free(tmp_hw_dnids);

3622         return (0);
3623         }

3625 /*
```



```
3626  * Check presence of a NID in the table of NIDs. The table may be NULL (i.e.,
3627  * non-existent).
3628  */
3629 static int nid_in_table(int nid, int *nid_table)
3630         {
3631         int i = 0;

3633         /*
3634          * a special case. NULL means that we are initializing a new
3635          * table.
3636          */
3637         if (nid_table == NULL)
3638                 return (1);

3640         /*
3641          * the table is never full, there is always at least one
3642          * NID_undef.
3643          */
3644         while (nid_table[i] != NID_undef)
3645                 {
3646                 if (nid_table[i++] == nid)
3647                         {
3648 #ifdef  DEBUG_SLOT_SELECTION
3649         fprintf(stderr, " (NID %d in hw table, idx %d)", nid, i);
3650 #endif  /* DEBUG_SLOT_SELECTION */
3651                         return (1);
3652                         }
3653                 }

3655         return (0);
3656         }
3657 #endif  /* SOLARIS_HW_SLOT_SELECTION */

3659 #endif  /* OPENSSL_NO_HW_PK11 */
3660 #endif  /* OPENSSL_NO_HW */
3661 #endif /* ! codereview */
```

```
**********************************************************
   11334 Wed Aug 13 19:32:59 2014
new/usr/src/lib/openssl/libsunw_crypto/engine/hw_pk11_err.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* crypto/engine/hw_pk11_err.c */
/*
 * This product includes software developed by the OpenSSL Project for
 * use in the OpenSSL Toolkit (http://www.openssl.org/).
 *
 * This project also referenced hw_pkcs11-0.9.7b.patch written by
 * Afchine Madjlessi.
 */
/*
 * ====================================================================
 * Copyright (c) 2000-2001 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
```



```
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

#include <stdio.h>
#include <openssl/opensslconf.h>
#include <openssl/err.h>
#include "hw_pk11_err.h"

/* BEGIN ERROR CODES */
#ifndef OPENSSL_NO_ERR
static ERR_STRING_DATA pk11_str_functs[]=
{
{ ERR_PACK(0, PK11_F_INIT, 0),                  "PK11_INIT"},
{ ERR_PACK(0, PK11_F_FINISH, 0),                "PK11_FINISH"},
{ ERR_PACK(0, PK11_F_DESTROY, 0),               "PK11_DESTROY"},
{ ERR_PACK(0, PK11_F_CTRL, 0),                  "PK11_CTRL"},
{ ERR_PACK(0, PK11_F_RSA_INIT, 0),              "PK11_RSA_INIT"},
{ ERR_PACK(0, PK11_F_RSA_FINISH, 0),            "PK11_RSA_FINISH"},
{ ERR_PACK(0, PK11_F_GET_PUB_RSA_KEY, 0),       "PK11_GET_PUB_RSA_KEY"},
{ ERR_PACK(0, PK11_F_GET_PRIV_RSA_KEY, 0),      "PK11_GET_PRIV_RSA_KEY"},
{ ERR_PACK(0, PK11_F_RSA_GEN_KEY, 0),           "PK11_RSA_GEN_KEY"},
{ ERR_PACK(0, PK11_F_RSA_PUB_ENC, 0),           "PK11_RSA_PUB_ENC"},
{ ERR_PACK(0, PK11_F_RSA_PRIV_ENC, 0),          "PK11_RSA_PRIV_ENC"},
{ ERR_PACK(0, PK11_F_RSA_PUB_DEC, 0),           "PK11_RSA_PUB_DEC"},
{ ERR_PACK(0, PK11_F_RSA_PRIV_DEC, 0),          "PK11_RSA_PRIV_DEC"},
{ ERR_PACK(0, PK11_F_RSA_SIGN, 0),              "PK11_RSA_SIGN"},
{ ERR_PACK(0, PK11_F_RSA_VERIFY, 0),            "PK11_RSA_VERIFY"},
{ ERR_PACK(0, PK11_F_RAND_ADD, 0),              "PK11_RAND_ADD"},
{ ERR_PACK(0, PK11_F_RAND_BYTES, 0),            "PK11_RAND_BYTES"},
{ ERR_PACK(0, PK11_F_GET_SESSION, 0),           "PK11_GET_SESSION"},
{ ERR_PACK(0, PK11_F_FREE_SESSION, 0),          "PK11_FREE_SESSION"},
{ ERR_PACK(0, PK11_F_LOAD_PUBKEY, 0),           "PK11_LOAD_PUBKEY"},
{ ERR_PACK(0, PK11_F_LOAD_PRIVKEY, 0),          "PK11_LOAD_PRIV_KEY"},
{ ERR_PACK(0, PK11_F_RSA_PUB_ENC_LOW, 0),       "PK11_RSA_PUB_ENC_LOW"},
{ ERR_PACK(0, PK11_F_RSA_PRIV_ENC_LOW, 0),      "PK11_RSA_PRIV_ENC_LOW"},
{ ERR_PACK(0, PK11_F_RSA_PUB_DEC_LOW, 0),       "PK11_RSA_PUB_DEC_LOW"},
{ ERR_PACK(0, PK11_F_RSA_PRIV_DEC_LOW, 0),      "PK11_RSA_PRIV_DEC_LOW"},
{ ERR_PACK(0, PK11_F_DSA_SIGN, 0),              "PK11_DSA_SIGN"},
{ ERR_PACK(0, PK11_F_DSA_VERIFY, 0),            "PK11_DSA_VERIFY"},
{ ERR_PACK(0, PK11_F_DSA_INIT, 0),              "PK11_DSA_INIT"},
{ ERR_PACK(0, PK11_F_DSA_FINISH, 0),            "PK11_DSA_FINISH"},
{ ERR_PACK(0, PK11_F_GET_PUB_DSA_KEY, 0),       "PK11_GET_PUB_DSA_KEY"},
{ ERR_PACK(0, PK11_F_GET_PRIV_DSA_KEY, 0),      "PK11_GET_PRIV_DSA_KEY"},
{ ERR_PACK(0, PK11_F_DH_INIT, 0),               "PK11_DH_INIT"},
{ ERR_PACK(0, PK11_F_DH_FINISH, 0),             "PK11_DH_FINISH"},
{ ERR_PACK(0, PK11_F_MOD_EXP_DH, 0),            "PK11_MOD_EXP_DH"},
{ ERR_PACK(0, PK11_F_GET_DH_KEY, 0),            "PK11_GET_DH_KEY"},
{ ERR_PACK(0, PK11_F_FREE_ALL_SESSIONS, 0),     "PK11_FREE_ALL_SESSIONS"},
{ ERR_PACK(0, PK11_F_SETUP_SESSION, 0),         "PK11_SETUP_SESSION"},
{ ERR_PACK(0, PK11_F_DESTROY_OBJECT, 0),        "PK11_DESTROY_OBJECT"},
{ ERR_PACK(0, PK11_F_CIPHER_INIT, 0),           "PK11_CIPHER_INIT"},
{ ERR_PACK(0, PK11_F_CIPHER_DO_CIPHER, 0),      "PK11_CIPHER_DO_CIPHER"},
{ ERR_PACK(0, PK11_F_GET_CIPHER_KEY, 0),        "PK11_GET_CIPHER_KEY"},
{ ERR_PACK(0, PK11_F_DIGEST_INIT, 0),           "PK11_DIGEST_INIT"},
{ ERR_PACK(0, PK11_F_DIGEST_UPDATE, 0),         "PK11_DIGEST_UPDATE"},
{ ERR_PACK(0, PK11_F_DIGEST_FINAL, 0),          "PK11_DIGEST_FINAL"},
{ ERR_PACK(0, PK11_F_CHOOSE_SLOT, 0),           "PK11_CHOOSE_SLOT"},
{ ERR_PACK(0, PK11_F_CIPHER_FINAL, 0),          "PK11_CIPHER_FINAL"},
{ ERR_PACK(0, PK11_F_LIBRARY_INIT, 0),          "PK11_LIBRARY_INIT"},
{ ERR_PACK(0, PK11_F_LOAD, 0),                  "ENGINE_LOAD_PK11"},
{ ERR_PACK(0, PK11_F_DH_GEN_KEY, 0),            "PK11_DH_GEN_KEY"},
{ ERR_PACK(0, PK11_F_DH_COMP_KEY, 0),           "PK11_DH_COMP_KEY"},
```

```
{ ERR_PACK(0, PK11_F_DIGEST_COPY, 0),          "PK11_DIGEST_COPY"},
{ ERR_PACK(0, PK11_F_CIPHER_CLEANUP, 0),       "PK11_CIPHER_CLEANUP"},
{ ERR_PACK(0, PK11_F_ACTIVE_ADD, 0),           "PK11_ACTIVE_ADD"},
{ ERR_PACK(0, PK11_F_ACTIVE_DELETE, 0),        "PK11_ACTIVE_DELETE"},
{ ERR_PACK(0, PK11_F_CHECK_HW_MECHANISMS, 0),  "PK11_CHECK_HW_MECHANISMS"},
{ ERR_PACK(0, PK11_F_INIT_SYMMETRIC, 0),       "PK11_INIT_SYMMETRIC"},
{ ERR_PACK(0, PK11_F_ADD_AES_CTR_NIDS, 0),     "PK11_ADD_AES_CTR_NIDS"},
{ 0, NULL}
};

static ERR_STRING_DATA pk11_str_reasons[]=
{
{ PK11_R_ALREADY_LOADED,               "PKCS#11 DSO already loaded"},
{ PK11_R_DSO_FAILURE,                  "unable to load PKCS#11 DSO"},
{ PK11_R_NOT_LOADED,                   "PKCS#11 DSO not loaded"},
{ PK11_R_PASSED_NULL_PARAMETER,        "null parameter passed"},
{ PK11_R_COMMAND_NOT_IMPLEMENTED,      "command not implemented"},
{ PK11_R_INITIALIZE,                   "C_Initialize failed"},
{ PK11_R_FINALIZE,                     "C_Finalize failed"},
{ PK11_R_GETINFO,                      "C_GetInfo faile"},
{ PK11_R_GETSLOTLIST,                  "C_GetSlotList failed"},
{ PK11_R_NO_MODULUS_OR_NO_EXPONENT,    "no modulus or no exponent"},
{ PK11_R_ATTRIBUT_SENSITIVE_OR_INVALID, "attr sensitive or invalid"},
{ PK11_R_GETATTRIBUTVALUE,             "C_GetAttributeValue failed"},
{ PK11_R_NO_MODULUS,                   "no modulus"},
{ PK11_R_NO_EXPONENT,                  "no exponent"},
{ PK11_R_FINDOBJECTSINIT,              "C_FindObjectsInit failed"},
{ PK11_R_FINDOBJECTS,                  "C_FindObjects failed"},
{ PK11_R_FINDOBJECTSFINAL,             "C_FindObjectsFinal failed"},
{ PK11_R_CREATEOBJECT,                 "C_CreateObject failed"},
{ PK11_R_DESTROYOBJECT,                "C_DestroyObject failed"},
{ PK11_R_OPENSESSION,                  "C_OpenSession failed"},
{ PK11_R_CLOSESESSION,                 "C_CloseSession failed"},
{ PK11_R_ENCRYPTINIT,                  "C_EncryptInit failed"},
{ PK11_R_ENCRYPT,                      "C_Encrypt failed"},
{ PK11_R_SIGNINIT,                     "C_SignInit failed"},
{ PK11_R_SIGN,                         "C_Sign failed"},
{ PK11_R_DECRYPTINIT,                  "C_DecryptInit failed"},
{ PK11_R_DECRYPT,                      "C_Decrypt failed"},
{ PK11_R_VERIFYINIT,                   "C_VerifyRecover failed"},
{ PK11_R_VERIFY,                       "C_Verify failed"},
{ PK11_R_VERIFYRECOVERINIT,            "C_VerifyRecoverInit failed"},
{ PK11_R_VERIFYRECOVER,                "C_VerifyRecover failed"},
{ PK11_R_GEN_KEY,                      "C_GenerateKeyPair failed"},
{ PK11_R_SEEDRANDOM,                   "C_SeedRandom failed"},
{ PK11_R_GENERATERANDOM,               "C_GenerateRandom failed"},
{ PK11_R_INVALID_MESSAGE_LENGTH,       "invalid message length"},
{ PK11_R_UNKNOWN_ALGORITHM_TYPE,       "unknown algorithm type"},
{ PK11_R_UNKNOWN_ASN1_OBJECT_ID,       "unknown asn1 onject id"},
{ PK11_R_UNKNOWN_PADDING_TYPE,         "unknown padding type"},
{ PK11_R_PADDING_CHECK_FAILED,         "padding check failed"},
{ PK11_R_DIGEST_TOO_BIG,               "digest too big"},
{ PK11_R_MALLOC_FAILURE,               "malloc failure"},
{ PK11_R_CTRL_COMMAND_NOT_IMPLEMENTED, "ctl command not implemented"},
{ PK11_R_DATA_GREATER_THAN_MOD_LEN,    "data is bigger than mod"},
{ PK11_R_DATA_TOO_LARGE_FOR_MODULUS,   "data is too larger for mod"},
{ PK11_R_MISSING_KEY_COMPONENT,        "a dsa component is missing"},
{ PK11_R_INVALID_SIGNATURE_LENGTH,     "invalid signature length"},
{ PK11_R_INVALID_DSA_SIGNATURE_R,      "missing r in dsa verify"},
{ PK11_R_INVALID_DSA_SIGNATURE_S,      "missing s in dsa verify"},
{ PK11_R_INCONSISTENT_KEY,             "inconsistent key type"},
{ PK11_R_ENCRYPTUPDATE,                "C_EncryptUpdate failed"},
{ PK11_R_DECRYPTUPDATE,                "C_DecryptUpdate failed"},
{ PK11_R_DIGESTINIT,                   "C_DigestInit failed"},
{ PK11_R_DIGESTUPDATE,                 "C_DigestUpdate failed"},
{ PK11_R_DIGESTFINAL,                  "C_DigestFinal failed"},
```



```
{ PK11_R_ENCRYPTFINAL,                 "C_EncryptFinal failed"},
{ PK11_R_DECRYPTFINAL,                 "C_DecryptFinal failed"},
{ PK11_R_NO_PRNG_SUPPORT,              "Slot does not support PRNG"},
{ PK11_R_GETTOKENINFO,                 "C_GetTokenInfo failed"},
{ PK11_R_DERIVEKEY,                    "C_DeriveKey failed"},
{ PK11_R_GET_OPERATION_STATE,          "C_GetOperationState failed"},
{ PK11_R_SET_OPERATION_STATE,          "C_SetOperationState failed"},
{ PK11_R_INVALID_HANDLE,               "invalid PKCS#11 object handle"},
{ PK11_R_KEY_OR_IV_LEN_PROBLEM,        "IV or key length incorrect"},
{ PK11_R_INVALID_OPERATION_TYPE,       "invalid operation type"},
{ PK11_R_ADD_NID_FAILED,               "failed to add NID" },
{ 0,    NULL}
};
#endif  /* OPENSSL_NO_ERR */

static int pk11_lib_error_code = 0;
static int pk11_error_init = 1;

static void
ERR_load_pk11_strings(void)
        {
        if (pk11_lib_error_code == 0)
                pk11_lib_error_code = ERR_get_next_error_library();

        if (pk11_error_init)
                {
                pk11_error_init = 0;
#ifndef OPENSSL_NO_ERR
                ERR_load_strings(pk11_lib_error_code, pk11_str_functs);
                ERR_load_strings(pk11_lib_error_code, pk11_str_reasons);
#endif
                }
}

static void
ERR_unload_pk11_strings(void)
        {
        if (pk11_error_init == 0)
                {
#ifndef OPENSSL_NO_ERR
                ERR_unload_strings(pk11_lib_error_code, pk11_str_functs);
                ERR_unload_strings(pk11_lib_error_code, pk11_str_reasons);
#endif
                pk11_error_init = 1;
                }
}

void
ERR_pk11_error(int function, int reason, char *file, int line)
{
        if (pk11_lib_error_code == 0)
                pk11_lib_error_code = ERR_get_next_error_library();
        ERR_PUT_error(pk11_lib_error_code, function, reason, file, line);
}

void
PK11err_add_data(int function, int reason, CK_RV rv)
{
        char tmp_buf[20];

        PK11err(function, reason);
        (void) snprintf(tmp_buf, sizeof (tmp_buf), "%lx", rv);
        ERR_add_error_data(2, "PK11 CK_RV=0X", tmp_buf);
}
#endif /* ! codereview */
```

```
**********************************************************
   70067 Wed Aug 13 19:32:59 2014
new/usr/src/lib/openssl/libsunw_crypto/engine/hw_pk11_pub.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* crypto/engine/hw_pk11_pub.c */
/*
 * This product includes software developed by the OpenSSL Project for
 * use in the OpenSSL Toolkit (http://www.openssl.org/).
 *
 * This project also referenced hw_pkcs11-0.9.7b.patch written by
 * Afchine Madjlessi.
 */
/*
 * ====================================================================
 * Copyright (c) 2000-2001 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
```



```
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

#include <openssl/e_os2.h>
#include <openssl/crypto.h>
#include <openssl/engine.h>
#include <openssl/dso.h>
#include <openssl/err.h>
#include <openssl/bn.h>
#include <openssl/pem.h>
#ifndef OPENSSL_NO_RSA
#include <openssl/rsa.h>
#endif /* OPENSSL_NO_RSA */
#ifndef OPENSSL_NO_DSA
#include <openssl/dsa.h>
#endif /* OPENSSL_NO_DSA */
#ifndef OPENSSL_NO_DH
#include <openssl/dh.h>
#endif /* OPENSSL_NO_DH */
#include <openssl/rand.h>
#include <openssl/objects.h>
#include <openssl/x509.h>
#include <cryptlib.h>
#include <pthread.h>

#ifndef OPENSSL_NO_HW
#ifndef OPENSSL_NO_HW_PK11

#include "cryptoki.h"
#include "pkcs11.h"
#include "hw_pk11_err.h"

#ifndef OPENSSL_NO_RSA
/* RSA stuff */
static int pk11_RSA_public_encrypt(int flen, const unsigned char *from,
        unsigned char *to, RSA *rsa, int padding);
static int pk11_RSA_private_encrypt(int flen, const unsigned char *from,
        unsigned char *to, RSA *rsa, int padding);
static int pk11_RSA_public_decrypt(int flen, const unsigned char *from,
        unsigned char *to, RSA *rsa, int padding);
static int pk11_RSA_private_decrypt(int flen, const unsigned char *from,
        unsigned char *to, RSA *rsa, int padding);
static int pk11_RSA_init(RSA *rsa);
static int pk11_RSA_finish(RSA *rsa);
static int pk11_RSA_sign(int type, const unsigned char *m, unsigned int m_len,
        unsigned char *sigret, unsigned int *siglen, const RSA *rsa);
static int pk11_RSA_verify(int dtype, const unsigned char *m,
        unsigned int m_len, const unsigned char *sigbuf, unsigned int siglen,
        const RSA *rsa);
EVP_PKEY *pk11_load_privkey(ENGINE*, const char *pubkey_file,
        UI_METHOD *ui_method, void *callback_data);
EVP_PKEY *pk11_load_pubkey(ENGINE*, const char *pubkey_file,
        UI_METHOD *ui_method, void *callback_data);

static int pk11_RSA_public_encrypt_low(int flen, const unsigned char *from,
        unsigned char *to, RSA *rsa);
```

```
128 static int pk11_RSA_private_encrypt_low(int flen, const unsigned char *from,
129         unsigned char *to, RSA *rsa);
130 static int pk11_RSA_public_decrypt_low(int flen, const unsigned char *from,
131         unsigned char *to, RSA *rsa);
132 static int pk11_RSA_private_decrypt_low(int flen, const unsigned char *from,
133         unsigned char *to, RSA *rsa);

135 static CK_OBJECT_HANDLE pk11_get_public_rsa_key(RSA* rsa, RSA** key_ptr,
136         BIGNUM **rsa_n_num, BIGNUM **rsa_e_num, CK_SESSION_HANDLE session);
137 static CK_OBJECT_HANDLE pk11_get_private_rsa_key(RSA* rsa, RSA** key_ptr,
138         BIGNUM **rsa_d_num, CK_SESSION_HANDLE session);

140 static int check_new_rsa_key_pub(PK11_SESSION *sp, const RSA *rsa);
141 static int check_new_rsa_key_priv(PK11_SESSION *sp, const RSA *rsa);
142 #endif

144 /* DSA stuff */
145 #ifndef OPENSSL_NO_DSA
146 static int pk11_DSA_init(DSA *dsa);
147 static int pk11_DSA_finish(DSA *dsa);
148 static DSA_SIG *pk11_dsa_do_sign(const unsigned char *dgst, int dlen,
149         DSA *dsa);
150 static int pk11_dsa_do_verify(const unsigned char *dgst, int dgst_len,
151         DSA_SIG *sig, DSA *dsa);

153 static CK_OBJECT_HANDLE pk11_get_public_dsa_key(DSA* dsa, DSA **key_ptr,
154         BIGNUM **dsa_pub_num, CK_SESSION_HANDLE session);
155 static CK_OBJECT_HANDLE pk11_get_private_dsa_key(DSA* dsa, DSA **key_ptr,
156         BIGNUM **dsa_priv_num, CK_SESSION_HANDLE session);

158 static int check_new_dsa_key_pub(PK11_SESSION *sp, DSA *dsa);
159 static int check_new_dsa_key_priv(PK11_SESSION *sp, DSA *dsa);
160 #endif

162 /* DH stuff */
163 #ifndef OPENSSL_NO_DH
164 static int pk11_DH_init(DH *dh);
165 static int pk11_DH_finish(DH *dh);
166 static int pk11_DH_generate_key(DH *dh);
167 static int pk11_DH_compute_key(unsigned char *key,
168         const BIGNUM *pub_key, DH *dh);

170 static CK_OBJECT_HANDLE pk11_get_dh_key(DH* dh, DH **key_ptr,
171         BIGNUM **priv_key, CK_SESSION_HANDLE session);

173 static int check_new_dh_key(PK11_SESSION *sp, DH *dh);
174 #endif

176 static int init_template_value(BIGNUM *bn, CK_VOID_PTR *pValue,
177         CK_ULONG *ulValueLen);

179 /* Read mode string to be used for fopen() */
180 #if SOLARIS_OPENSSL
181 static char *read_mode_flags = "rF";
182 #else
183 static char *read_mode_flags = "r";
184 #endif

186 /*
187  * increment/create reference for an asymmetric key handle via active list
188  * manipulation. If active list operation fails, unlock (if locked), set error
189  * variable and jump to the specified label.
190  */
191 #define KEY_HANDLE_REFHOLD(key_handle, alg_type, unlock, var, label)    \
192         {                                                               \
193         if (sunw_pk11_active_add(key_handle, alg_type) < 0)
```



```
194                 {                                                       \
195                 var = TRUE;                                             \
196                 if (unlock)                                             \
197                         UNLOCK_OBJSTORE(alg_type);                      \
198                 goto label;                                             \
199                 }                                                       \
200         }

202 /*
203  * Find active list entry according to object handle and return pointer to the
204  * entry otherwise return NULL.
205  *
206  * This function presumes it is called with lock protecting the active list
207  * held.
208  */
209 static PK11_active *pk11_active_find(CK_OBJECT_HANDLE h, PK11_OPTYPE type)
210         {
211         PK11_active *entry;

213         for (entry = active_list[type]; entry != NULL; entry = entry->next)
214                 if (entry->h == h)
215                         return (entry);

217         return (NULL);
218         }

220 /*
221  * Search for an entry in the active list using PKCS#11 object handle as a
222  * search key and return refcnt of the found/created entry or -1 in case of
223  * failure.
224  *
225  * This function presumes it is called with lock protecting the active list
226  * held.
227  */
228 int
229 sunw_pk11_active_add(CK_OBJECT_HANDLE h, PK11_OPTYPE type)
230         {
231         PK11_active *entry = NULL;

233         if (h == CK_INVALID_HANDLE)
234                 {
235                 PK11err(PK11_F_ACTIVE_ADD, PK11_R_INVALID_HANDLE);
236                 return (-1);
237                 }

239         /* search for entry in the active list */
240         if ((entry = pk11_active_find(h, type)) != NULL)
241                 entry->refcnt++;
242         else
243                 {
244                 /* not found, create new entry and add it to the list */
245                 entry = OPENSSL_malloc(sizeof (PK11_active));
246                 if (entry == NULL)
247                         {
248                         PK11err(PK11_F_ACTIVE_ADD, PK11_R_MALLOC_FAILURE);
249                         return (-1);
250                         }
251                 entry->h = h;
252                 entry->refcnt = 1;
253                 entry->prev = NULL;
254                 entry->next = NULL;
255                 /* connect the newly created entry to the list */
256                 if (active_list[type] == NULL)
257                         active_list[type] = entry;
258                 else /* make the entry first in the list */
259                         {
```

```
260                         entry->next = active_list[type];
261                         active_list[type]->prev = entry;
262                         active_list[type] = entry;
263                         }
264                 }

266         return (entry->refcnt);
267         }

269 /*
270  * Remove active list entry from the list and free it.
271  *
272  * This function presumes it is called with lock protecting the active list
273  * held.
274  */
275 void
276 sunw_pk11_active_remove(PK11_active *entry, PK11_OPTYPE type)
277         {
278         PK11_active *prev_entry;

280         /* remove the entry from the list and free it */
281         if ((prev_entry = entry->prev) != NULL)
282                 {
283                 prev_entry->next = entry->next;
284                 if (entry->next != NULL)
285                         entry->next->prev = prev_entry;
286                 }
287         else
288                 {
289                 active_list[type] = entry->next;
290                 /* we were the first but not the only one */
291                 if (entry->next != NULL)
292                         entry->next->prev = NULL;
293                 }

295         /* sanitization */
296         entry->h = CK_INVALID_HANDLE;
297         entry->prev = NULL;
298         entry->next = NULL;
299         OPENSSL_free(entry);
300         }

302 /* Free all entries from the active list. */
303 void
304 sunw_pk11_free_active_list(PK11_OPTYPE type)
305         {
306         PK11_active *entry;

308         /* only for asymmetric types since only they have C_Find* locks. */
309         switch (type)
310                 {
311                 case OP_RSA:
312                 case OP_DSA:
313                 case OP_DH:
314                         break;
315                 default:
316                         return;
317                 }

319         /* see find_lock array definition for more info on object locking */
320         LOCK_OBJSTORE(type);
321         while ((entry = active_list[type]) != NULL)
322                 sunw_pk11_active_remove(entry, type);
323         UNLOCK_OBJSTORE(type);
324         }
```



```
326 /*
327  * Search for active list entry associated with given PKCS#11 object handle,
328  * decrement its refcnt and if it drops to 0, disconnect the entry and free it.
329  *
330  * Return 1 if the PKCS#11 object associated with the entry has no references,
331  * return 0 if there is at least one reference, -1 on error.
332  *
333  * This function presumes it is called with lock protecting the active list
334  * held.
335  */
336 int
337 sunw_pk11_active_delete(CK_OBJECT_HANDLE h, PK11_OPTYPE type)
338         {
339         PK11_active *entry = NULL;

341         if ((entry = pk11_active_find(h, type)) == NULL)
342                 {
343                 PK11err(PK11_F_ACTIVE_DELETE, PK11_R_INVALID_HANDLE);
344                 return (-1);
345                 }

347         OPENSSL_assert(entry->refcnt > 0);
348         entry->refcnt--;
349         if (entry->refcnt == 0)
350                 {
351                 sunw_pk11_active_remove(entry, type);
352                 return (1);
353                 }

355         return (0);
356         }

358 #ifndef OPENSSL_NO_RSA
359 /* Our internal RSA_METHOD that we provide pointers to */
360 static RSA_METHOD pk11_rsa =
361         {
362         "PKCS#11 RSA method",
363         pk11_RSA_public_encrypt,                /* rsa_pub_encrypt */
364         pk11_RSA_public_decrypt,                /* rsa_pub_decrypt */
365         pk11_RSA_private_encrypt,               /* rsa_priv_encrypt */
366         pk11_RSA_private_decrypt,               /* rsa_priv_decrypt */
367         NULL,                                   /* rsa_mod_exp */
368         NULL,                                   /* bn_mod_exp */
369         pk11_RSA_init,                          /* init */
370         pk11_RSA_finish,                        /* finish */
371         RSA_FLAG_SIGN_VER,                      /* flags */
372         NULL,                                   /* app_data */
373         pk11_RSA_sign,                          /* rsa_sign */
374         pk11_RSA_verify                         /* rsa_verify */
375         };

377 RSA_METHOD *
378 PK11_RSA(void)
379         {
380         return (&pk11_rsa);
381         }
382 #endif

384 #ifndef OPENSSL_NO_DSA
385 /* Our internal DSA_METHOD that we provide pointers to */
386 static DSA_METHOD pk11_dsa =
387         {
388         "PKCS#11 DSA method",
389         pk11_dsa_do_sign,       /* dsa_do_sign */
390         NULL,                   /* dsa_sign_setup */
391         pk11_dsa_do_verify,     /* dsa_do_verify */
```

```
392         NULL,                   /* dsa_mod_exp */
393         NULL,                   /* bn_mod_exp */
394         pk11_DSA_init,          /* init */
395         pk11_DSA_finish,        /* finish */
396         0,                      /* flags */
397         NULL                    /* app_data */
398         };

400 DSA_METHOD *
401 PK11_DSA(void)
402         {
403         return (&pk11_dsa);
404         }
405 #endif

407 #ifndef OPENSSL_NO_DH
408 /*
409  * PKCS #11 V2.20, section 11.2 specifies that the number of bytes needed for
410  * output buffer may somewhat exceed the precise number of bytes needed, but
411  * should not exceed it by a large amount. That may be caused, for example, by
412  * rounding it up to multiple of X in the underlying bignum library. 8 should be
413  * enough.
414  */
415 #define DH_BUF_RESERVE  8

417 /* Our internal DH_METHOD that we provide pointers to */
418 static DH_METHOD pk11_dh =
419         {
420         "PKCS#11 DH method",
421         pk11_DH_generate_key,   /* generate_key */
422         pk11_DH_compute_key,    /* compute_key */
423         NULL,                   /* bn_mod_exp */
424         pk11_DH_init,           /* init */
425         pk11_DH_finish,         /* finish */
426         0,                      /* flags */
427         NULL,                   /* app_data */
428         NULL                    /* generate_params */
429         };

431 DH_METHOD *
432 PK11_DH(void)
433         {
434         return (&pk11_dh);
435         }
436 #endif

438 /* Size of an SSL signature: MD5+SHA1 */
439 #define SSL_SIG_LENGTH          36

441 /* Lengths of DSA data and signature */
442 #define DSA_DATA_LEN            20
443 #define DSA_SIGNATURE_LEN       40

445 static CK_BBOOL true = TRUE;
446 static CK_BBOOL false = FALSE;

448 #ifndef OPENSSL_NO_RSA
449 /*
450  * Similiar to OpenSSL to take advantage of the paddings. The goal is to
451  * support all paddings in this engine although PK11 library does not
452  * support all the paddings used in OpenSSL.
453  * The input errors should have been checked in the padding functions.
454  */
455 static int pk11_RSA_public_encrypt(int flen, const unsigned char *from,
456                 unsigned char *to, RSA *rsa, int padding)
457         {
```



```
458         int i, num = 0, r = -1;
459         unsigned char *buf = NULL;

461         num = BN_num_bytes(rsa->n);
462         if ((buf = (unsigned char *)OPENSSL_malloc(num)) == NULL)
463                 {
464                 RSAerr(PK11_F_RSA_PUB_ENC, PK11_R_MALLOC_FAILURE);
465                 goto err;
466                 }

468         switch (padding)
469                 {
470         case RSA_PKCS1_PADDING:
471                 i = RSA_padding_add_PKCS1_type_2(buf, num, from, flen);
472                 break;
473 #ifndef OPENSSL_NO_SHA
474         case RSA_PKCS1_OAEP_PADDING:
475                 i = RSA_padding_add_PKCS1_OAEP(buf, num, from, flen, NULL, 0);
476                 break;
477 #endif
478         case RSA_SSLV23_PADDING:
479                 i = RSA_padding_add_SSLv23(buf, num, from, flen);
480                 break;
481         case RSA_NO_PADDING:
482                 i = RSA_padding_add_none(buf, num, from, flen);
483                 break;
484         default:
485                 RSAerr(PK11_F_RSA_PUB_ENC, PK11_R_UNKNOWN_PADDING_TYPE);
486                 goto err;
487                 }
488         if (i <= 0) goto err;

490         /* PK11 functions are called here */
491         r = pk11_RSA_public_encrypt_low(num, buf, to, rsa);
492 err:
493         if (buf != NULL)
494                 {
495                 OPENSSL_cleanse(buf, num);
496                 OPENSSL_free(buf);
497                 }
498         return (r);
499         }


502 /*
503  * Similar to Openssl to take advantage of the paddings. The input errors
504  * should be catched in the padding functions
505  */
506 static int pk11_RSA_private_encrypt(int flen, const unsigned char *from,
507         unsigned char *to, RSA *rsa, int padding)
508         {
509         int i, num = 0, r = -1;
510         unsigned char *buf = NULL;

512         num = BN_num_bytes(rsa->n);
513         if ((buf = (unsigned char *)OPENSSL_malloc(num)) == NULL)
514                 {
515                 RSAerr(PK11_F_RSA_PRIV_ENC, PK11_R_MALLOC_FAILURE);
516                 goto err;
517                 }

519         switch (padding)
520                 {
521         case RSA_PKCS1_PADDING:
522                 i = RSA_padding_add_PKCS1_type_1(buf, num, from, flen);
523                 break;
```

```
524         case RSA_NO_PADDING:
525                 i = RSA_padding_add_none(buf, num, from, flen);
526                 break;
527         case RSA_SSLV23_PADDING:
528         default:
529                 RSAerr(PK11_F_RSA_PRIV_ENC, PK11_R_UNKNOWN_PADDING_TYPE);
530                 goto err;
531                 }
532         if (i <= 0) goto err;

534         /* PK11 functions are called here */
535         r = pk11_RSA_private_encrypt_low(num, buf, to, rsa);
536 err:
537         if (buf != NULL)
538                 {
539                 OPENSSL_cleanse(buf, num);
540                 OPENSSL_free(buf);
541                 }
542         return (r);
543         }

545 /* Similar to OpenSSL code. Input errors are also checked here */
546 static int pk11_RSA_private_decrypt(int flen, const unsigned char *from,
547         unsigned char *to, RSA *rsa, int padding)
548         {
549         BIGNUM f;
550         int j, num = 0, r = -1;
551         unsigned char *p;
552         unsigned char *buf = NULL;

554         BN_init(&f);

556         num = BN_num_bytes(rsa->n);

558         if ((buf = (unsigned char *)OPENSSL_malloc(num)) == NULL)
559                 {
560                 RSAerr(PK11_F_RSA_PRIV_DEC, PK11_R_MALLOC_FAILURE);
561                 goto err;
562                 }

564         /*
565          * This check was for equality but PGP does evil things
566          * and chops off the top '0' bytes
567          */
568         if (flen > num)
569                 {
570                 RSAerr(PK11_F_RSA_PRIV_DEC,
571                         PK11_R_DATA_GREATER_THAN_MOD_LEN);
572                 goto err;
573                 }

575         /* make data into a big number */
576         if (BN_bin2bn(from, (int)flen, &f) == NULL)
577                 goto err;

579         if (BN_ucmp(&f, rsa->n) >= 0)
580                 {
581                 RSAerr(PK11_F_RSA_PRIV_DEC,
582                         PK11_R_DATA_TOO_LARGE_FOR_MODULUS);
583                 goto err;
584                 }

586         /* PK11 functions are called here */
587         r = pk11_RSA_private_decrypt_low(flen, from, buf, rsa);

589         /*
```



```
590          * PK11 CKM_RSA_X_509 mechanism pads 0's at the beginning.
591          * Needs to skip these 0's paddings here.
592          */
593         for (j = 0; j < r; j++)
594                 if (buf[j] != 0)
595                         break;

597         p = buf + j;
598         j = r - j;  /* j is only used with no-padding mode */

600         switch (padding)
601                 {
602         case RSA_PKCS1_PADDING:
603                 r = RSA_padding_check_PKCS1_type_2(to, num, p, j, num);
604                 break;
605 #ifndef OPENSSL_NO_SHA
606         case RSA_PKCS1_OAEP_PADDING:
607                 r = RSA_padding_check_PKCS1_OAEP(to, num, p, j, num, NULL, 0);
608                 break;
609 #endif
610         case RSA_SSLV23_PADDING:
611                 r = RSA_padding_check_SSLv23(to, num, p, j, num);
612                 break;
613         case RSA_NO_PADDING:
614                 r = RSA_padding_check_none(to, num, p, j, num);
615                 break;
616         default:
617                 RSAerr(PK11_F_RSA_PRIV_DEC, PK11_R_UNKNOWN_PADDING_TYPE);
618                 goto err;
619                 }
620         if (r < 0)
621                 RSAerr(PK11_F_RSA_PRIV_DEC, PK11_R_PADDING_CHECK_FAILED);

623 err:
624         BN_clear_free(&f);
625         if (buf != NULL)
626                 {
627                 OPENSSL_cleanse(buf, num);
628                 OPENSSL_free(buf);
629                 }
630         return (r);
631         }

633 /* Similar to OpenSSL code. Input errors are also checked here */
634 static int pk11_RSA_public_decrypt(int flen, const unsigned char *from,
635         unsigned char *to, RSA *rsa, int padding)
636         {
637         BIGNUM f;
638         int i, num = 0, r = -1;
639         unsigned char *p;
640         unsigned char *buf = NULL;

642         BN_init(&f);
643         num = BN_num_bytes(rsa->n);
644         buf = (unsigned char *)OPENSSL_malloc(num);
645         if (buf == NULL)
646                 {
647                 RSAerr(PK11_F_RSA_PUB_DEC, PK11_R_MALLOC_FAILURE);
648                 goto err;
649                 }

651         /*
652          * This check was for equality but PGP does evil things
653          * and chops off the top '0' bytes
654          */
655         if (flen > num)
```

```
656                 {
657                 RSAerr(PK11_F_RSA_PUB_DEC, PK11_R_DATA_GREATER_THAN_MOD_LEN);
658                 goto err;
659                 }

661         if (BN_bin2bn(from, flen, &f) == NULL)
662                 goto err;

664         if (BN_ucmp(&f, rsa->n) >= 0)
665                 {
666                 RSAerr(PK11_F_RSA_PUB_DEC,
667                         PK11_R_DATA_TOO_LARGE_FOR_MODULUS);
668                 goto err;
669                 }

671         /* PK11 functions are called here */
672         r = pk11_RSA_public_decrypt_low(flen, from, buf, rsa);

674         /*
675          * PK11 CKM_RSA_X_509 mechanism pads 0's at the beginning.
676          * Needs to skip these 0's here
677          */
678         for (i = 0; i < r; i++)
679                 if (buf[i] != 0)
680                         break;

682         p = buf + i;
683         i = r - i;  /* i is only used with no-padding mode */

685         switch (padding)
686                 {
687         case RSA_PKCS1_PADDING:
688                 r = RSA_padding_check_PKCS1_type_1(to, num, p, i, num);
689                 break;
690         case RSA_NO_PADDING:
691                 r = RSA_padding_check_none(to, num, p, i, num);
692                 break;
693         default:
694                 RSAerr(PK11_F_RSA_PUB_DEC, PK11_R_UNKNOWN_PADDING_TYPE);
695                 goto err;
696                 }
697         if (r < 0)
698                 RSAerr(PK11_F_RSA_PUB_DEC, PK11_R_PADDING_CHECK_FAILED);

700 err:
701         BN_clear_free(&f);
702         if (buf != NULL)
703                 {
704                 OPENSSL_cleanse(buf, num);
705                 OPENSSL_free(buf);
706                 }
707         return (r);
708         }

710 /*
711  * This function implements RSA public encryption using C_EncryptInit and
712  * C_Encrypt pk11 interfaces. Note that the CKM_RSA_X_509 is used here.
713  * The calling function allocated sufficient memory in "to" to store results.
714  */
715 static int pk11_RSA_public_encrypt_low(int flen,
716         const unsigned char *from, unsigned char *to, RSA *rsa)
717         {
718         CK_ULONG bytes_encrypted = flen;
719         int retval = -1;
720         CK_RV rv;
721         CK_MECHANISM mech_rsa = {CKM_RSA_X_509, NULL, 0};
```



```
722         CK_MECHANISM *p_mech = &mech_rsa;
723         CK_OBJECT_HANDLE h_pub_key = CK_INVALID_HANDLE;
724         PK11_SESSION *sp;

726         if ((sp = pk11_get_session(OP_RSA)) == NULL)
727                 return (-1);

729         (void) check_new_rsa_key_pub(sp, rsa);

731         h_pub_key = sp->opdata_rsa_pub_key;
732         if (h_pub_key == CK_INVALID_HANDLE)
733                 h_pub_key = sp->opdata_rsa_pub_key =
734                         pk11_get_public_rsa_key(rsa, &sp->opdata_rsa_pub,
735                             &sp->opdata_rsa_n_num, &sp->opdata_rsa_e_num,
736                             sp->session);

738         if (h_pub_key != CK_INVALID_HANDLE)
739                 {
740                 rv = pFuncList->C_EncryptInit(sp->session, p_mech,
741                         h_pub_key);

743                 if (rv != CKR_OK)
744                         {
745                         PK11err_add_data(PK11_F_RSA_PUB_ENC_LOW,
746                             PK11_R_ENCRYPTINIT, rv);
747                         pk11_return_session(sp, OP_RSA);
748                         return (-1);
749                         }

751                 rv = pFuncList->C_Encrypt(sp->session,
752                         (unsigned char *)from, flen, to, &bytes_encrypted);

754                 if (rv != CKR_OK)
755                         {
756                         PK11err_add_data(PK11_F_RSA_PUB_ENC_LOW,
757                             PK11_R_ENCRYPT, rv);
758                         pk11_return_session(sp, OP_RSA);
759                         return (-1);
760                         }
761                 retval = bytes_encrypted;
762                 }

764         pk11_return_session(sp, OP_RSA);
765         return (retval);
766         }


769 /*
770  * This function implements RSA private encryption using C_SignInit and
771  * C_Sign pk11 APIs. Note that CKM_RSA_X_509 is used here.
772  * The calling function allocated sufficient memory in "to" to store results.
773  */
774 static int pk11_RSA_private_encrypt_low(int flen,
775         const unsigned char *from, unsigned char *to, RSA *rsa)
776         {
777         CK_ULONG ul_sig_len = flen;
778         int retval = -1;
779         CK_RV rv;
780         CK_MECHANISM mech_rsa = {CKM_RSA_X_509, NULL, 0};
781         CK_MECHANISM *p_mech = &mech_rsa;
782         CK_OBJECT_HANDLE h_priv_key = CK_INVALID_HANDLE;
783         PK11_SESSION *sp;

785         if ((sp = pk11_get_session(OP_RSA)) == NULL)
786                 return (-1);
```

```
788         (void) check_new_rsa_key_priv(sp, rsa);

790         h_priv_key = sp->opdata_rsa_priv_key;
791         if (h_priv_key == CK_INVALID_HANDLE)
792                 h_priv_key = sp->opdata_rsa_priv_key =
793                         pk11_get_private_rsa_key(rsa, &sp->opdata_rsa_priv,
794                             &sp->opdata_rsa_d_num, sp->session);

796         if (h_priv_key != CK_INVALID_HANDLE)
797                 {
798                 rv = pFuncList->C_SignInit(sp->session, p_mech,
799                         h_priv_key);

801                 if (rv != CKR_OK)
802                         {
803                         PK11err_add_data(PK11_F_RSA_PRIV_ENC_LOW,
804                             PK11_R_SIGNINIT, rv);
805                         pk11_return_session(sp, OP_RSA);
806                         return (-1);
807                         }

809                 rv = pFuncList->C_Sign(sp->session,
810                         (unsigned char *)from, flen, to, &ul_sig_len);

812                 if (rv != CKR_OK)
813                         {
814                         PK11err_add_data(PK11_F_RSA_PRIV_ENC_LOW, PK11_R_SIGN,
815                             rv);
816                         pk11_return_session(sp, OP_RSA);
817                         return (-1);
818                         }

820                 retval = ul_sig_len;
821                 }

823         pk11_return_session(sp, OP_RSA);
824         return (retval);
825         }


828 /*
829  * This function implements RSA private decryption using C_DecryptInit and
830  * C_Decrypt pk11 APIs. Note that CKM_RSA_X_509 mechanism is used here.
831  * The calling function allocated sufficient memory in "to" to store results.
832  */
833 static int pk11_RSA_private_decrypt_low(int flen,
834         const unsigned char *from, unsigned char *to, RSA *rsa)
835         {
836         CK_ULONG bytes_decrypted = flen;
837         int retval = -1;
838         CK_RV rv;
839         CK_MECHANISM mech_rsa = {CKM_RSA_X_509, NULL, 0};
840         CK_MECHANISM *p_mech = &mech_rsa;
841         CK_OBJECT_HANDLE h_priv_key;
842         PK11_SESSION *sp;

844         if ((sp = pk11_get_session(OP_RSA)) == NULL)
845                 return (-1);

847         (void) check_new_rsa_key_priv(sp, rsa);

849         h_priv_key = sp->opdata_rsa_priv_key;
850         if (h_priv_key == CK_INVALID_HANDLE)
851                 h_priv_key = sp->opdata_rsa_priv_key =
852                         pk11_get_private_rsa_key(rsa, &sp->opdata_rsa_priv,
853                             &sp->opdata_rsa_d_num, sp->session);
```



```
855         if (h_priv_key != CK_INVALID_HANDLE)
856                 {
857                 rv = pFuncList->C_DecryptInit(sp->session, p_mech,
858                         h_priv_key);

860                 if (rv != CKR_OK)
861                         {
862                         PK11err_add_data(PK11_F_RSA_PRIV_DEC_LOW,
863                                 PK11_R_DECRYPTINIT, rv);
864                         pk11_return_session(sp, OP_RSA);
865                         return (-1);
866                         }

868                 rv = pFuncList->C_Decrypt(sp->session,
869                         (unsigned char *)from, flen, to, &bytes_decrypted);

871                 if (rv != CKR_OK)
872                         {
873                         PK11err_add_data(PK11_F_RSA_PRIV_DEC_LOW,
874                             PK11_R_DECRYPT, rv);
875                         pk11_return_session(sp, OP_RSA);
876                         return (-1);
877                         }
878                 retval = bytes_decrypted;
879                 }

881         pk11_return_session(sp, OP_RSA);
882         return (retval);
883         }


886 /*
887  * This function implements RSA public decryption using C_VerifyRecoverInit
888  * and C_VerifyRecover pk11 APIs. Note that CKM_RSA_X_509 is used here.
889  * The calling function allocated sufficient memory in "to" to store results.
890  */
891 static int pk11_RSA_public_decrypt_low(int flen,
892         const unsigned char *from, unsigned char *to, RSA *rsa)
893         {
894         CK_ULONG bytes_decrypted = flen;
895         int retval = -1;
896         CK_RV rv;
897         CK_MECHANISM mech_rsa = {CKM_RSA_X_509, NULL, 0};
898         CK_MECHANISM *p_mech = &mech_rsa;
899         CK_OBJECT_HANDLE h_pub_key = CK_INVALID_HANDLE;
900         PK11_SESSION *sp;

902         if ((sp = pk11_get_session(OP_RSA)) == NULL)
903                 return (-1);

905         (void) check_new_rsa_key_pub(sp, rsa);

907         h_pub_key = sp->opdata_rsa_pub_key;
908         if (h_pub_key == CK_INVALID_HANDLE)
909                 h_pub_key = sp->opdata_rsa_pub_key =
910                         pk11_get_public_rsa_key(rsa, &sp->opdata_rsa_pub,
911                             &sp->opdata_rsa_n_num, &sp->opdata_rsa_e_num,
912                             sp->session);

914         if (h_pub_key != CK_INVALID_HANDLE)
915                 {
916                 rv = pFuncList->C_VerifyRecoverInit(sp->session,
917                         p_mech, h_pub_key);

919                 if (rv != CKR_OK)
```

```
920                         {
921                         PK11err_add_data(PK11_F_RSA_PUB_DEC_LOW,
922                                 PK11_R_VERIFYRECOVERINIT, rv);
923                         pk11_return_session(sp, OP_RSA);
924                         return (-1);
925                         }

927                 rv = pFuncList->C_VerifyRecover(sp->session,
928                         (unsigned char *)from, flen, to, &bytes_decrypted);

930                 if (rv != CKR_OK)
931                         {
932                         PK11err_add_data(PK11_F_RSA_PUB_DEC_LOW,
933                             PK11_R_VERIFYRECOVER, rv);
934                         pk11_return_session(sp, OP_RSA);
935                         return (-1);
936                         }
937                 retval = bytes_decrypted;
938                 }

940         pk11_return_session(sp, OP_RSA);
941         return (retval);
942         }

944 static int pk11_RSA_init(RSA *rsa)
945         {
946         /*
947          * This flag in the RSA_METHOD enables the new rsa_sign,
948          * rsa_verify functions. See rsa.h for details.
949          */
950         rsa->flags |= RSA_FLAG_SIGN_VER;

952         return (1);
953         }

955 static int pk11_RSA_finish(RSA *rsa)
956         {
957         /*
958          * Since we are overloading OpenSSL's native RSA_eay_finish() we need
959          * to do the same as in the original function, i.e. to free bignum
960          * structures.
961          */
962         if (rsa->_method_mod_n != NULL)
963                 BN_MONT_CTX_free(rsa->_method_mod_n);
964         if (rsa->_method_mod_p != NULL)
965                 BN_MONT_CTX_free(rsa->_method_mod_p);
966         if (rsa->_method_mod_q != NULL)
967                 BN_MONT_CTX_free(rsa->_method_mod_q);

969         return (1);
970         }

972 /*
973  * Standard engine interface function. Majority codes here are from
974  * rsa/rsa_sign.c. We replaced the decrypt function call by C_Sign of PKCS#11.
975  * See more details in rsa/rsa_sign.c
976  */
977 static int pk11_RSA_sign(int type, const unsigned char *m, unsigned int m_len,
978         unsigned char *sigret, unsigned int *siglen, const RSA *rsa)
979         {
980         X509_SIG sig;
981         ASN1_TYPE parameter;
982         int i, j=0;
983         unsigned char *p, *s = NULL;
984         X509_ALGOR algor;
985         ASN1_OCTET_STRING digest;
```



```
986         CK_RV rv;
987         CK_MECHANISM mech_rsa = {CKM_RSA_PKCS, NULL, 0};
988         CK_MECHANISM *p_mech = &mech_rsa;
989         CK_OBJECT_HANDLE h_priv_key;
990         PK11_SESSION *sp = NULL;
991         int ret = 0;
992         unsigned long ulsiglen;

994         /* Encode the digest */
995         /* Special case: SSL signature, just check the length */
996         if (type == NID_md5_sha1)
997                 {
998                 if (m_len != SSL_SIG_LENGTH)
999                         {
1000                        PK11err(PK11_F_RSA_SIGN,
1001                                PK11_R_INVALID_MESSAGE_LENGTH);
1002                        goto err;
1003                        }
1004                i = SSL_SIG_LENGTH;
1005                s = (unsigned char *)m;
1006                }
1007        else
1008                {
1009                sig.algor = &algor;
1010                sig.algor->algorithm = OBJ_nid2obj(type);
1011                if (sig.algor->algorithm == NULL)
1012                        {
1013                        PK11err(PK11_F_RSA_SIGN,
1014                                PK11_R_UNKNOWN_ALGORITHM_TYPE);
1015                        goto err;
1016                        }
1017                if (sig.algor->algorithm->length == 0)
1018                        {
1019                        PK11err(PK11_F_RSA_SIGN,
1020                                PK11_R_UNKNOWN_ASN1_OBJECT_ID);
1021                        goto err;
1022                        }
1023                parameter.type = V_ASN1_NULL;
1024                parameter.value.ptr = NULL;
1025                sig.algor->parameter = &parameter;

1027                sig.digest = &digest;
1028                sig.digest->data = (unsigned char *)m;
1029                sig.digest->length = m_len;

1031                i = i2d_X509_SIG(&sig, NULL);
1032                }

1034        j = RSA_size(rsa);
1035        if ((i - RSA_PKCS1_PADDING) > j)
1036                {
1037                PK11err(PK11_F_RSA_SIGN, PK11_R_DIGEST_TOO_BIG);
1038                goto err;
1039                }

1041        if (type != NID_md5_sha1)
1042                {
1043                s = (unsigned char *)OPENSSL_malloc((unsigned int)(j + 1));
1044                if (s == NULL)
1045                        {
1046                        PK11err(PK11_F_RSA_SIGN, PK11_R_MALLOC_FAILURE);
1047                        goto err;
1048                        }
1049                p = s;
1050                (void) i2d_X509_SIG(&sig, &p);
1051                }
```

```
1053         if ((sp = pk11_get_session(OP_RSA)) == NULL)
1054                 goto err;

1056         (void) check_new_rsa_key_priv(sp, rsa);

1058         h_priv_key = sp->opdata_rsa_priv_key;
1059         if (h_priv_key == CK_INVALID_HANDLE)
1060                 h_priv_key = sp->opdata_rsa_priv_key =
1061                         pk11_get_private_rsa_key((RSA *)rsa,
1062                             &sp->opdata_rsa_priv,
1063                             &sp->opdata_rsa_d_num, sp->session);

1065         if (h_priv_key != CK_INVALID_HANDLE)
1066                 {
1067                 rv = pFuncList->C_SignInit(sp->session, p_mech, h_priv_key);

1069                 if (rv != CKR_OK)
1070                         {
1071                         PK11err_add_data(PK11_F_RSA_SIGN, PK11_R_SIGNINIT, rv);
1072                         goto err;
1073                         }

1075                 ulsiglen = j;
1076                 rv = pFuncList->C_Sign(sp->session, s, i, sigret,
1077                         (CK_ULONG_PTR) &ulsiglen);
1078                 *siglen = ulsiglen;

1080                 if (rv != CKR_OK)
1081                         {
1082                         PK11err_add_data(PK11_F_RSA_SIGN, PK11_R_SIGN, rv);
1083                         goto err;
1084                         }
1085                 ret = 1;
1086                 }

1088 err:
1089         if (type != NID_md5_sha1)
1090                 {
1091                 (void) memset(s, 0, (unsigned int)(j + 1));
1092                 OPENSSL_free(s);
1093                 }

1095         pk11_return_session(sp, OP_RSA);
1096         return (ret);
1097         }

1099 static int pk11_RSA_verify(int type, const unsigned char *m,
1100         unsigned int m_len, const unsigned char *sigbuf, unsigned int siglen,
1101         const RSA *rsa)
1102         {
1103         X509_SIG sig;
1104         ASN1_TYPE parameter;
1105         int i, j;
1106         unsigned char *p, *s = NULL;
1107         X509_ALGOR algor;
1108         ASN1_OCTET_STRING digest;
1109         CK_RV rv;
1110         CK_MECHANISM mech_rsa = {CKM_RSA_PKCS, NULL, 0};
1111         CK_MECHANISM *p_mech = &mech_rsa;
1112         CK_OBJECT_HANDLE h_pub_key;
1113         PK11_SESSION *sp = NULL;
1114         int ret = 0;

1116         /* Encode the digest    */
1117         /* Special case: SSL signature, just check the length */
```



```
1118         if (type == NID_md5_sha1)
1119                 {
1120                 if (m_len != SSL_SIG_LENGTH)
1121                         {
1122                         PK11err(PK11_F_RSA_VERIFY,
1123                                 PK11_R_INVALID_MESSAGE_LENGTH);
1124                         goto err;
1125                         }
1126                 i = SSL_SIG_LENGTH;
1127                 s = (unsigned char *)m;
1128                 }
1129         else
1130                 {
1131                 sig.algor = &algor;
1132                 sig.algor->algorithm = OBJ_nid2obj(type);
1133                 if (sig.algor->algorithm == NULL)
1134                         {
1135                         PK11err(PK11_F_RSA_VERIFY,
1136                                 PK11_R_UNKNOWN_ALGORITHM_TYPE);
1137                         goto err;
1138                         }
1139                 if (sig.algor->algorithm->length == 0)
1140                         {
1141                         PK11err(PK11_F_RSA_VERIFY,
1142                                 PK11_R_UNKNOWN_ASN1_OBJECT_ID);
1143                         goto err;
1144                         }
1145                 parameter.type = V_ASN1_NULL;
1146                 parameter.value.ptr = NULL;
1147                 sig.algor->parameter = &parameter;
1148                 sig.digest = &digest;
1149                 sig.digest->data = (unsigned char *)m;
1150                 sig.digest->length = m_len;
1151                 i = i2d_X509_SIG(&sig, NULL);
1152                 }

1154         j = RSA_size(rsa);
1155         if ((i - RSA_PKCS1_PADDING) > j)
1156                 {
1157                 PK11err(PK11_F_RSA_VERIFY, PK11_R_DIGEST_TOO_BIG);
1158                 goto err;
1159                 }

1161         if (type != NID_md5_sha1)
1162                 {
1163                 s = (unsigned char *)OPENSSL_malloc((unsigned int)(j + 1));
1164                 if (s == NULL)
1165                         {
1166                         PK11err(PK11_F_RSA_VERIFY, PK11_R_MALLOC_FAILURE);
1167                         goto err;
1168                         }
1169                 p = s;
1170                 (void) i2d_X509_SIG(&sig, &p);
1171                 }

1173         if ((sp = pk11_get_session(OP_RSA)) == NULL)
1174                 goto err;

1176         (void) check_new_rsa_key_pub(sp, rsa);

1178         h_pub_key = sp->opdata_rsa_pub_key;
1179         if (h_pub_key == CK_INVALID_HANDLE)
1180                 h_pub_key = sp->opdata_rsa_pub_key =
1181                         pk11_get_public_rsa_key((RSA *)rsa, &sp->opdata_rsa_pub,
1182                             &sp->opdata_rsa_n_num, &sp->opdata_rsa_e_num,
1183                             sp->session);
```

```
1185         if (h_pub_key != CK_INVALID_HANDLE)
1186                 {
1187                 rv = pFuncList->C_VerifyInit(sp->session, p_mech,
1188                         h_pub_key);

1190                 if (rv != CKR_OK)
1191                         {
1192                         PK11err_add_data(PK11_F_RSA_VERIFY, PK11_R_VERIFYINIT,
1193                             rv);
1194                         goto err;
1195                         }
1196                 rv = pFuncList->C_Verify(sp->session, s, i, (CK_BYTE_PTR) sigbuf
1197                         (CK_ULONG)siglen);

1199                 if (rv != CKR_OK)
1200                         {
1201                         PK11err_add_data(PK11_F_RSA_VERIFY, PK11_R_VERIFY, rv);
1202                         goto err;
1203                         }
1204                 ret = 1;
1205                 }

1207 err:
1208         if (type != NID_md5_sha1)
1209                 {
1210                 (void) memset(s, 0, (unsigned int)siglen);
1211                 OPENSSL_free(s);
1212                 }

1214         pk11_return_session(sp, OP_RSA);
1215         return (ret);
1216         }

1218 /* load RSA private key from a file */
1219 /* ARGSUSED */
1220 EVP_PKEY *pk11_load_privkey(ENGINE* e, const char *privkey_file,
1221         UI_METHOD *ui_method, void *callback_data)
1222         {
1223         EVP_PKEY *pkey = NULL;
1224         FILE *pubkey;
1225         CK_OBJECT_HANDLE  h_priv_key = CK_INVALID_HANDLE;
1226         RSA *rsa;
1227         PK11_SESSION *sp;

1229         if ((sp = pk11_get_session(OP_RSA)) == NULL)
1230                 return (NULL);

1232         if ((pubkey = fopen(privkey_file, read_mode_flags)) != NULL)
1233                 {
1234                 pkey = PEM_read_PrivateKey(pubkey, NULL, NULL, NULL);
1235                 (void) fclose(pubkey);
1236                 if (pkey != NULL)
1237                         {
1238                         rsa = EVP_PKEY_get1_RSA(pkey);
1239                         if (rsa != NULL)
1240                                 {
1241                                 (void) check_new_rsa_key_priv(sp, rsa);

1243                                 h_priv_key = sp->opdata_rsa_priv_key =
1244                                     pk11_get_private_rsa_key(rsa,
1245                                     &sp->opdata_rsa_priv, &sp->opdata_rsa_d_num,
1246                                     sp->session);
1247                                 if (h_priv_key == CK_INVALID_HANDLE)
1248                                         {
1249                                         EVP_PKEY_free(pkey);
```



```
1250                                         pkey = NULL;
1251                                         }
1252                                 }
1253                         else
1254                                 {
1255                                 EVP_PKEY_free(pkey);
1256                                 pkey = NULL;
1257                                 }
1258                         }
1259                 }

1261         pk11_return_session(sp, OP_RSA);
1262         return (pkey);
1263         }

1265 /* load RSA public key from a file */
1266 /* ARGSUSED */
1267 EVP_PKEY *pk11_load_pubkey(ENGINE* e, const char *pubkey_file,
1268         UI_METHOD *ui_method, void *callback_data)
1269         {
1270         EVP_PKEY *pkey = NULL;
1271         FILE *pubkey;
1272         CK_OBJECT_HANDLE  h_pub_key = CK_INVALID_HANDLE;
1273         RSA *rsa;
1274         PK11_SESSION *sp;

1276         if ((sp = pk11_get_session(OP_RSA)) == NULL)
1277                 return (NULL);

1279         if ((pubkey = fopen(pubkey_file, read_mode_flags)) != NULL)
1280                 {
1281                 pkey = PEM_read_PUBKEY(pubkey, NULL, NULL, NULL);
1282                 (void) fclose(pubkey);
1283                 if (pkey != NULL)
1284                         {
1285                         rsa = EVP_PKEY_get1_RSA(pkey);
1286                         if (rsa != NULL)
1287                                 {
1288                                 (void) check_new_rsa_key_pub(sp, rsa);

1290                                 h_pub_key = sp->opdata_rsa_pub_key =
1291                                     pk11_get_public_rsa_key(rsa,
1292                                     &sp->opdata_rsa_pub, &sp->opdata_rsa_n_num,
1293                                     &sp->opdata_rsa_e_num, sp->session);
1294                                 if (h_pub_key == CK_INVALID_HANDLE)
1295                                         {
1296                                         EVP_PKEY_free(pkey);
1297                                         pkey = NULL;
1298                                         }
1299                                 }
1300                         else
1301                                 {
1302                                 EVP_PKEY_free(pkey);
1303                                 pkey = NULL;
1304                                 }
1305                         }
1306                 }

1308         pk11_return_session(sp, OP_RSA);
1309         return (pkey);
1310         }

1312 /*
1313  * Create a public key object in a session from a given rsa structure.
1314  * The *rsa_n_num and *rsa_e_num pointers are non-NULL for RSA public keys.
1315  */
```

```
1316 static CK_OBJECT_HANDLE pk11_get_public_rsa_key(RSA* rsa,
1317     RSA** key_ptr, BIGNUM **rsa_n_num, BIGNUM **rsa_e_num,
1318     CK_SESSION_HANDLE session)
1319         {
1320         CK_RV rv;
1321         CK_OBJECT_HANDLE h_key = CK_INVALID_HANDLE;
1322         CK_ULONG found;
1323         CK_OBJECT_CLASS o_key = CKO_PUBLIC_KEY;
1324         CK_KEY_TYPE k_type = CKK_RSA;
1325         CK_ULONG ul_key_attr_count = 7;
1326         CK_BBOOL rollback = FALSE;

1328         CK_ATTRIBUTE  a_key_template[] =
1329                 {
1330                 {CKA_CLASS, (void *) NULL, sizeof (CK_OBJECT_CLASS)},
1331                 {CKA_KEY_TYPE, (void *) NULL, sizeof (CK_KEY_TYPE)},
1332                 {CKA_TOKEN, &false, sizeof (true)},
1333                 {CKA_ENCRYPT, &true, sizeof (true)},
1334                 {CKA_VERIFY_RECOVER, &true, sizeof (true)},
1335                 {CKA_MODULUS, (void *)NULL, 0},
1336                 {CKA_PUBLIC_EXPONENT, (void *)NULL, 0}
1337                 };

1339         int i;

1341         a_key_template[0].pValue = &o_key;
1342         a_key_template[1].pValue = &k_type;

1344         a_key_template[5].ulValueLen = BN_num_bytes(rsa->n);
1345         a_key_template[5].pValue = (CK_VOID_PTR)OPENSSL_malloc(
1346                 (size_t)a_key_template[5].ulValueLen);
1347         if (a_key_template[5].pValue == NULL)
1348                 {
1349                 PK11err(PK11_F_GET_PUB_RSA_KEY, PK11_R_MALLOC_FAILURE);
1350                 goto malloc_err;
1351                 }

1353         BN_bn2bin(rsa->n, a_key_template[5].pValue);

1355         a_key_template[6].ulValueLen = BN_num_bytes(rsa->e);
1356         a_key_template[6].pValue = (CK_VOID_PTR)OPENSSL_malloc(
1357                 (size_t)a_key_template[6].ulValueLen);
1358         if (a_key_template[6].pValue == NULL)
1359                 {
1360                 PK11err(PK11_F_GET_PUB_RSA_KEY, PK11_R_MALLOC_FAILURE);
1361                 goto malloc_err;
1362                 }

1364         BN_bn2bin(rsa->e, a_key_template[6].pValue);

1366         /* see find_lock array definition for more info on object locking */
1367         LOCK_OBJSTORE(OP_RSA);
1368         rv = pFuncList->C_FindObjectsInit(session, a_key_template,
1369                 ul_key_attr_count);

1371         if (rv != CKR_OK)
1372                 {
1373                 PK11err_add_data(PK11_F_GET_PUB_RSA_KEY, PK11_R_FINDOBJECTSINIT,
1374                     rv);
1375                 goto err;
1376                 }

1378         rv = pFuncList->C_FindObjects(session, &h_key, 1, &found);

1380         if (rv != CKR_OK)
1381                 {
```



```
1382                 PK11err_add_data(PK11_F_GET_PUB_RSA_KEY,
1383                     PK11_R_FINDOBJECTS, rv);
1384                 goto err;
1385                 }

1387         rv = pFuncList->C_FindObjectsFinal(session);

1389         if (rv != CKR_OK)
1390                 {
1391                 PK11err_add_data(PK11_F_GET_PUB_RSA_KEY,
1392                     PK11_R_FINDOBJECTSFINAL, rv);
1393                 goto err;
1394                 }

1396         if (found == 0)
1397                 {
1398                 rv = pFuncList->C_CreateObject(session,
1399                         a_key_template, ul_key_attr_count, &h_key);
1400                 if (rv != CKR_OK)
1401                         {
1402                         PK11err_add_data(PK11_F_GET_PUB_RSA_KEY,
1403                             PK11_R_CREATEOBJECT, rv);
1404                         goto err;
1405                         }
1406                 }

1408         if (rsa_n_num != NULL)
1409                 if ((*rsa_n_num = BN_dup(rsa->n)) == NULL)
1410                         {
1411                         PK11err(PK11_F_GET_PUB_RSA_KEY, PK11_R_MALLOC_FAILURE);
1412                         rollback = TRUE;
1413                         goto err;
1414                         }
1415         if (rsa_e_num != NULL)
1416                 if ((*rsa_e_num = BN_dup(rsa->e)) == NULL)
1417                         {
1418                         PK11err(PK11_F_GET_PUB_RSA_KEY, PK11_R_MALLOC_FAILURE);
1419                         BN_free(*rsa_n_num);
1420                         *rsa_n_num = NULL;
1421                         rollback = TRUE;
1422                         goto err;
1423                         }

1425         /* LINTED: E_CONSTANT_CONDITION */
1426         KEY_HANDLE_REFHOLD(h_key, OP_RSA, FALSE, rollback, err);
1427         if (key_ptr != NULL)
1428                 *key_ptr = rsa;

1430 err:
1431         if (rollback)
1432                 {
1433                 /*
1434                  * We do not care about the return value from C_DestroyObject()
1435                  * since we are doing rollback.
1436                  */
1437                 if (found == 0)
1438                         (void) pFuncList->C_DestroyObject(session, h_key);
1439                 h_key = CK_INVALID_HANDLE;
1440                 }

1442         UNLOCK_OBJSTORE(OP_RSA);

1444 malloc_err:
1445         for (i = 5; i <= 6; i++)
1446                 {
1447                 if (a_key_template[i].pValue != NULL)
```

```
1448                        {
1449                        OPENSSL_free(a_key_template[i].pValue);
1450                        a_key_template[i].pValue = NULL;
1451                        }
1452                }

1454        return (h_key);
1455        }

1457 /*
1458  * Create a private key object in the session from a given rsa structure.
1459  * The *rsa_d_num pointer is non-NULL for RSA private keys.
1460  */
1461 static CK_OBJECT_HANDLE pk11_get_private_rsa_key(RSA* rsa,
1462     RSA** key_ptr, BIGNUM **rsa_d_num, CK_SESSION_HANDLE session)
1463        {
1464        CK_RV rv;
1465        CK_OBJECT_HANDLE h_key = CK_INVALID_HANDLE;
1466        int i;
1467        CK_ULONG found;
1468        CK_OBJECT_CLASS o_key = CKO_PRIVATE_KEY;
1469        CK_KEY_TYPE k_type = CKK_RSA;
1470        CK_ULONG ul_key_attr_count = 14;
1471        CK_BBOOL rollback = FALSE;

1473        /* Both CKA_TOKEN and CKA_SENSITIVE have to be FALSE for session keys */
1474        CK_ATTRIBUTE  a_key_template[] =
1475                {
1476                {CKA_CLASS, (void *) NULL, sizeof (CK_OBJECT_CLASS)},
1477                {CKA_KEY_TYPE, (void *) NULL, sizeof (CK_KEY_TYPE)},
1478                {CKA_TOKEN, &false, sizeof (true)},
1479                {CKA_SENSITIVE, &false, sizeof (true)},
1480                {CKA_DECRYPT, &true, sizeof (true)},
1481                {CKA_SIGN, &true, sizeof (true)},
1482                {CKA_MODULUS, (void *)NULL, 0},
1483                {CKA_PUBLIC_EXPONENT, (void *)NULL, 0},
1484                {CKA_PRIVATE_EXPONENT, (void *)NULL, 0},
1485                {CKA_PRIME_1, (void *)NULL, 0},
1486                {CKA_PRIME_2, (void *)NULL, 0},
1487                {CKA_EXPONENT_1, (void *)NULL, 0},
1488                {CKA_EXPONENT_2, (void *)NULL, 0},
1489                {CKA_COEFFICIENT, (void *)NULL, 0}
1490                };

1492        a_key_template[0].pValue = &o_key;
1493        a_key_template[1].pValue = &k_type;

1495        /* Put the private key components into the template */
1496        if (init_template_value(rsa->n, &a_key_template[6].pValue,
1497                &a_key_template[6].ulValueLen) == 0 ||
1498            init_template_value(rsa->e, &a_key_template[7].pValue,
1499                &a_key_template[7].ulValueLen) == 0 ||
1500            init_template_value(rsa->d, &a_key_template[8].pValue,
1501                &a_key_template[8].ulValueLen) == 0 ||
1502            init_template_value(rsa->p, &a_key_template[9].pValue,
1503                &a_key_template[9].ulValueLen) == 0 ||
1504            init_template_value(rsa->q, &a_key_template[10].pValue,
1505                &a_key_template[10].ulValueLen) == 0 ||
1506            init_template_value(rsa->dmp1, &a_key_template[11].pValue,
1507                &a_key_template[11].ulValueLen) == 0 ||
1508            init_template_value(rsa->dmq1, &a_key_template[12].pValue,
1509                &a_key_template[12].ulValueLen) == 0 ||
1510            init_template_value(rsa->iqmp, &a_key_template[13].pValue,
1511                &a_key_template[13].ulValueLen) == 0)
1512                {
1513                PK11err(PK11_F_GET_PRIV_RSA_KEY, PK11_R_MALLOC_FAILURE);
```



```
1514                goto malloc_err;
1515                }

1517        /* see find_lock array definition for more info on object locking */
1518        LOCK_OBJSTORE(OP_RSA);
1519        rv = pFuncList->C_FindObjectsInit(session, a_key_template,
1520                ul_key_attr_count);

1522        if (rv != CKR_OK)
1523                {
1524                PK11err_add_data(PK11_F_GET_PRIV_RSA_KEY,
1525                    PK11_R_FINDOBJECTSINIT, rv);
1526                goto err;
1527                }

1529        rv = pFuncList->C_FindObjects(session, &h_key, 1, &found);

1531        if (rv != CKR_OK)
1532                {
1533                PK11err_add_data(PK11_F_GET_PRIV_RSA_KEY,
1534                    PK11_R_FINDOBJECTS, rv);
1535                goto err;
1536                }

1538        rv = pFuncList->C_FindObjectsFinal(session);

1540        if (rv != CKR_OK)
1541                {
1542                PK11err_add_data(PK11_F_GET_PRIV_RSA_KEY,
1543                    PK11_R_FINDOBJECTSFINAL, rv);
1544                goto err;
1545                }

1547        if (found == 0)
1548                {
1549                rv = pFuncList->C_CreateObject(session,
1550                        a_key_template, ul_key_attr_count, &h_key);
1551                if (rv != CKR_OK)
1552                        {
1553                        PK11err_add_data(PK11_F_GET_PRIV_RSA_KEY,
1554                                PK11_R_CREATEOBJECT, rv);
1555                        goto err;
1556                        }
1557                }

1559        if (rsa_d_num != NULL)
1560                if ((*rsa_d_num = BN_dup(rsa->d)) == NULL)
1561                        {
1562                        PK11err(PK11_F_GET_PRIV_RSA_KEY, PK11_R_MALLOC_FAILURE);
1563                        rollback = TRUE;
1564                        goto err;
1565                        }

1567        /* LINTED: E_CONSTANT_CONDITION */
1568        KEY_HANDLE_REFHOLD(h_key, OP_RSA, FALSE, rollback, err);
1569        if (key_ptr != NULL)
1570                *key_ptr = rsa;

1572 err:
1573        if (rollback)
1574                {
1575                /*
1576                 * We do not care about the return value from C_DestroyObject()
1577                 * since we are doing rollback.
1578                 */
1579                if (found == 0)
```

```
1580                        (void) pFuncList->C_DestroyObject(session, h_key);
1581                h_key = CK_INVALID_HANDLE;
1582                }

1584        UNLOCK_OBJSTORE(OP_RSA);

1586 malloc_err:
1587        /*
1588         * 6 to 13 entries in the key template are key components.
1589         * They need to be freed apon exit or error.
1590         */
1591        for (i = 6; i <= 13; i++)
1592                {
1593                if (a_key_template[i].pValue != NULL)
1594                        {
1595                        (void) memset(a_key_template[i].pValue, 0,
1596                                a_key_template[i].ulValueLen);
1597                        OPENSSL_free(a_key_template[i].pValue);
1598                        a_key_template[i].pValue = NULL;
1599                        }
1600                }

1602        return (h_key);
1603        }

1605 /*
1606  * Check for cache miss and clean the object pointer and handle
1607  * in such case. Return 1 for cache hit, 0 for cache miss.
1608  */
1609 static int check_new_rsa_key_pub(PK11_SESSION *sp, const RSA *rsa)
1610        {
1611        /*
1612         * Provide protection against RSA structure reuse by making the
1613         * check for cache hit stronger. Only public components of RSA
1614         * key matter here so it is sufficient to compare them with values
1615         * cached in PK11_SESSION structure.
1616         */
1617        if ((sp->opdata_rsa_pub != rsa) ||
1618            (BN_cmp(sp->opdata_rsa_n_num, rsa->n) != 0) ||
1619            (BN_cmp(sp->opdata_rsa_e_num, rsa->e) != 0))
1620                {
1621                /*
1622                 * We do not check the return value because even in case of
1623                 * failure the sp structure will have both key pointer
1624                 * and object handle cleaned and pk11_destroy_object()
1625                 * reports the failure to the OpenSSL error message buffer.
1626                 */
1627                (void) pk11_destroy_rsa_object_pub(sp, TRUE);
1628                return (0);
1629                }
1630        return (1);
1631        }

1633 /*
1634  * Check for cache miss and clean the object pointer and handle
1635  * in such case. Return 1 for cache hit, 0 for cache miss.
1636  */
1637 static int check_new_rsa_key_priv(PK11_SESSION *sp, const RSA *rsa)
1638        {
1639        /*
1640         * Provide protection against RSA structure reuse by making the
1641         * check for cache hit stronger. Comparing private exponent of RSA
1642         * key with value cached in PK11_SESSION structure should
1643         * be sufficient.
1644         */
1645        if ((sp->opdata_rsa_priv != rsa) ||
```



```
1646            (BN_cmp(sp->opdata_rsa_d_num, rsa->d) != 0))
1647                {
1648                /*
1649                 * We do not check the return value because even in case of
1650                 * failure the sp structure will have both key pointer
1651                 * and object handle cleaned and pk11_destroy_object()
1652                 * reports the failure to the OpenSSL error message buffer.
1653                 */
1654                (void) pk11_destroy_rsa_object_priv(sp, TRUE);
1655                return (0);
1656                }
1657        return (1);
1658        }
1659 #endif

1661 #ifndef OPENSSL_NO_DSA
1662 /* The DSA function implementation */
1663 /* ARGSUSED */
1664 static int pk11_DSA_init(DSA *dsa)
1665        {
1666        return (1);
1667        }

1669 /* ARGSUSED */
1670 static int pk11_DSA_finish(DSA *dsa)
1671        {
1672        return (1);
1673        }


1676 static DSA_SIG *
1677 pk11_dsa_do_sign(const unsigned char *dgst, int dlen, DSA *dsa)
1678        {
1679        BIGNUM *r = NULL, *s = NULL;
1680        int i;
1681        DSA_SIG *dsa_sig = NULL;

1683        CK_RV rv;
1684        CK_MECHANISM Mechanism_dsa = {CKM_DSA, NULL, 0};
1685        CK_MECHANISM *p_mech = &Mechanism_dsa;
1686        CK_OBJECT_HANDLE h_priv_key;

1688        /*
1689         * The signature is the concatenation of r and s,
1690         * each is 20 bytes long
1691         */
1692        unsigned char sigret[DSA_SIGNATURE_LEN];
1693        unsigned long siglen = DSA_SIGNATURE_LEN;
1694        unsigned int siglen2 = DSA_SIGNATURE_LEN / 2;

1696        PK11_SESSION *sp = NULL;

1698        if ((dsa->p == NULL) || (dsa->q == NULL) || (dsa->g == NULL))
1699                {
1700                PK11err(PK11_F_DSA_SIGN, PK11_R_MISSING_KEY_COMPONENT);
1701                goto ret;
1702                }

1704        i = BN_num_bytes(dsa->q); /* should be 20 */
1705        if (dlen > i)
1706                {
1707                PK11err(PK11_F_DSA_SIGN, PK11_R_INVALID_SIGNATURE_LENGTH);
1708                goto ret;
1709                }

1711        if ((sp = pk11_get_session(OP_DSA)) == NULL)
```

```
1712                goto ret;

1714        (void) check_new_dsa_key_priv(sp, dsa);

1716        h_priv_key = sp->opdata_dsa_priv_key;
1717        if (h_priv_key == CK_INVALID_HANDLE)
1718                h_priv_key = sp->opdata_dsa_priv_key =
1719                        pk11_get_private_dsa_key((DSA *)dsa,
1720                            &sp->opdata_dsa_priv,
1721                            &sp->opdata_dsa_priv_num, sp->session);

1723        if (h_priv_key != CK_INVALID_HANDLE)
1724                {
1725                rv = pFuncList->C_SignInit(sp->session, p_mech, h_priv_key);

1727                if (rv != CKR_OK)
1728                        {
1729                        PK11err_add_data(PK11_F_DSA_SIGN, PK11_R_SIGNINIT, rv);
1730                        goto ret;
1731                        }

1733                        (void) memset(sigret, 0, siglen);
1734                        rv = pFuncList->C_Sign(sp->session,
1735                            (unsigned char *) dgst, dlen, sigret,
1736                            (CK_ULONG_PTR) &siglen);

1738                if (rv != CKR_OK)
1739                        {
1740                        PK11err_add_data(PK11_F_DSA_SIGN, PK11_R_SIGN, rv);
1741                        goto ret;
1742                        }
1743                }


1746        if ((s = BN_new()) == NULL)
1747                {
1748                PK11err(PK11_F_DSA_SIGN, PK11_R_MALLOC_FAILURE);
1749                goto ret;
1750                }

1752        if ((r = BN_new()) == NULL)
1753                {
1754                PK11err(PK11_F_DSA_SIGN, PK11_R_MALLOC_FAILURE);
1755                goto ret;
1756                }

1758        if ((dsa_sig = DSA_SIG_new()) == NULL)
1759                {
1760                PK11err(PK11_F_DSA_SIGN, PK11_R_MALLOC_FAILURE);
1761                goto ret;
1762                }

1764        if (BN_bin2bn(sigret, siglen2, r) == NULL ||
1765            BN_bin2bn(&sigret[siglen2], siglen2, s) == NULL)
1766                {
1767                PK11err(PK11_F_DSA_SIGN, PK11_R_MALLOC_FAILURE);
1768                goto ret;
1769                }

1771        dsa_sig->r = r;
1772        dsa_sig->s = s;

1774 ret:
1775        if (dsa_sig == NULL)
1776                {
1777                if (r != NULL)
```



```
1778                        BN_free(r);
1779                if (s != NULL)
1780                        BN_free(s);
1781                }

1783        pk11_return_session(sp, OP_DSA);
1784        return (dsa_sig);
1785        }

1787 static int
1788 pk11_dsa_do_verify(const unsigned char *dgst, int dlen, DSA_SIG *sig,
1789        DSA *dsa)
1790        {
1791        int i;
1792        CK_RV rv;
1793        int retval = 0;
1794        CK_MECHANISM Mechanism_dsa = {CKM_DSA, NULL, 0};
1795        CK_MECHANISM *p_mech = &Mechanism_dsa;
1796        CK_OBJECT_HANDLE h_pub_key;

1798        unsigned char sigbuf[DSA_SIGNATURE_LEN];
1799        unsigned long siglen = DSA_SIGNATURE_LEN;
1800        unsigned long siglen2 = DSA_SIGNATURE_LEN/2;

1802        PK11_SESSION *sp = NULL;

1804        if (BN_is_zero(sig->r) || sig->r->neg || BN_ucmp(sig->r, dsa->q) >= 0)
1805                {
1806                PK11err(PK11_F_DSA_VERIFY,
1807                        PK11_R_INVALID_DSA_SIGNATURE_R);
1808                goto ret;
1809                }

1811        if (BN_is_zero(sig->s) || sig->s->neg || BN_ucmp(sig->s, dsa->q) >= 0)
1812                {
1813                PK11err(PK11_F_DSA_VERIFY,
1814                        PK11_R_INVALID_DSA_SIGNATURE_S);
1815                goto ret;
1816                }

1818        i = BN_num_bytes(dsa->q); /* should be 20 */

1820        if (dlen > i)
1821                {
1822                PK11err(PK11_F_DSA_VERIFY,
1823                        PK11_R_INVALID_SIGNATURE_LENGTH);
1824                goto ret;
1825                }

1827        if ((sp = pk11_get_session(OP_DSA)) == NULL)
1828                goto ret;

1830        (void) check_new_dsa_key_pub(sp, dsa);

1832        h_pub_key = sp->opdata_dsa_pub_key;
1833        if (h_pub_key == CK_INVALID_HANDLE)
1834                h_pub_key = sp->opdata_dsa_pub_key =
1835                        pk11_get_public_dsa_key((DSA *)dsa, &sp->opdata_dsa_pub,
1836                            &sp->opdata_dsa_pub_num, sp->session);

1838        if (h_pub_key != CK_INVALID_HANDLE)
1839                {
1840                rv = pFuncList->C_VerifyInit(sp->session, p_mech,
1841                        h_pub_key);

1843                if (rv != CKR_OK)
```

```
1844                        {
1845                        PK11err_add_data(PK11_F_DSA_VERIFY, PK11_R_VERIFYINIT,
1846                            rv);
1847                        goto ret;
1848                        }

1850                /*
1851                 * The representation of each of the two big numbers could
1852                 * be shorter than DSA_SIGNATURE_LEN/2 bytes so we need
1853                 * to act accordingly and shift if necessary.
1854                 */
1855                (void) memset(sigbuf, 0, siglen);
1856                BN_bn2bin(sig->r, sigbuf + siglen2 - BN_num_bytes(sig->r));
1857                BN_bn2bin(sig->s, &sigbuf[siglen2] + siglen2 -
1858                    BN_num_bytes(sig->s));

1860                rv = pFuncList->C_Verify(sp->session,
1861                        (unsigned char *) dgst, dlen, sigbuf, (CK_ULONG)siglen);

1863                if (rv != CKR_OK)
1864                        {
1865                        PK11err_add_data(PK11_F_DSA_VERIFY, PK11_R_VERIFY, rv);
1866                        goto ret;
1867                        }
1868                }

1870        retval = 1;
1871 ret:

1873        pk11_return_session(sp, OP_DSA);
1874        return (retval);
1875        }


1878 /*
1879  * Create a public key object in a session from a given dsa structure.
1880  * The *dsa_pub_num pointer is non-NULL for DSA public keys.
1881  */
1882 static CK_OBJECT_HANDLE pk11_get_public_dsa_key(DSA* dsa,
1883     DSA **key_ptr, BIGNUM **dsa_pub_num, CK_SESSION_HANDLE session)
1884        {
1885        CK_RV rv;
1886        CK_OBJECT_CLASS o_key = CKO_PUBLIC_KEY;
1887        CK_OBJECT_HANDLE h_key = CK_INVALID_HANDLE;
1888        CK_ULONG found;
1889        CK_KEY_TYPE k_type = CKK_DSA;
1890        CK_ULONG ul_key_attr_count = 8;
1891        CK_BBOOL rollback = FALSE;
1892        int i;

1894        CK_ATTRIBUTE  a_key_template[] =
1895                {
1896                {CKA_CLASS, (void *) NULL, sizeof (CK_OBJECT_CLASS)},
1897                {CKA_KEY_TYPE, (void *) NULL, sizeof (CK_KEY_TYPE)},
1898                {CKA_TOKEN, &false, sizeof (true)},
1899                {CKA_VERIFY, &true, sizeof (true)},
1900                {CKA_PRIME, (void *)NULL, 0},           /* p */
1901                {CKA_SUBPRIME, (void *)NULL, 0},        /* q */
1902                {CKA_BASE, (void *)NULL, 0},            /* g */
1903                {CKA_VALUE, (void *)NULL, 0}            /* pub_key - y */
1904                };

1906        a_key_template[0].pValue = &o_key;
1907        a_key_template[1].pValue = &k_type;

1909        if (init_template_value(dsa->p, &a_key_template[4].pValue,
```



```
1910                &a_key_template[4].ulValueLen) == 0 ||
1911            init_template_value(dsa->q, &a_key_template[5].pValue,
1912                &a_key_template[5].ulValueLen) == 0 ||
1913            init_template_value(dsa->g, &a_key_template[6].pValue,
1914                &a_key_template[6].ulValueLen) == 0 ||
1915            init_template_value(dsa->pub_key, &a_key_template[7].pValue,
1916                &a_key_template[7].ulValueLen) == 0)
1917                {
1918                PK11err(PK11_F_GET_PUB_DSA_KEY, PK11_R_MALLOC_FAILURE);
1919                goto malloc_err;
1920                }

1922        /* see find_lock array definition for more info on object locking */
1923        LOCK_OBJSTORE(OP_DSA);
1924        rv = pFuncList->C_FindObjectsInit(session, a_key_template,
1925                ul_key_attr_count);

1927        if (rv != CKR_OK)
1928                {
1929                PK11err_add_data(PK11_F_GET_PUB_DSA_KEY, PK11_R_FINDOBJECTSINIT,
1930                    rv);
1931                goto err;
1932                }

1934        rv = pFuncList->C_FindObjects(session, &h_key, 1, &found);

1936        if (rv != CKR_OK)
1937                {
1938                PK11err_add_data(PK11_F_GET_PUB_DSA_KEY,
1939                    PK11_R_FINDOBJECTS, rv);
1940                goto err;
1941                }

1943        rv = pFuncList->C_FindObjectsFinal(session);

1945        if (rv != CKR_OK)
1946                {
1947                PK11err_add_data(PK11_F_GET_PUB_DSA_KEY,
1948                    PK11_R_FINDOBJECTSFINAL, rv);
1949                goto err;
1950                }

1952        if (found == 0)
1953                {
1954                rv = pFuncList->C_CreateObject(session,
1955                        a_key_template, ul_key_attr_count, &h_key);
1956                if (rv != CKR_OK)
1957                        {
1958                        PK11err_add_data(PK11_F_GET_PUB_DSA_KEY,
1959                            PK11_R_CREATEOBJECT, rv);
1960                        goto err;
1961                        }
1962                }

1964        if (dsa_pub_num != NULL)
1965                if ((*dsa_pub_num = BN_dup(dsa->pub_key)) == NULL)
1966                        {
1967                        PK11err(PK11_F_GET_PUB_DSA_KEY, PK11_R_MALLOC_FAILURE);
1968                        rollback = TRUE;
1969                        goto err;
1970                        }

1972        /* LINTED: E_CONSTANT_CONDITION */
1973        KEY_HANDLE_REFHOLD(h_key, OP_DSA, FALSE, rollback, err);
1974        if (key_ptr != NULL)
1975                *key_ptr = dsa;
```

```
1977 err:
1978         if (rollback)
1979                 {
1980                 /*
1981                  * We do not care about the return value from C_DestroyObject()
1982                  * since we are doing rollback.
1983                  */
1984                 if (found == 0)
1985                         (void) pFuncList->C_DestroyObject(session, h_key);
1986                 h_key = CK_INVALID_HANDLE;
1987                 }

1989         UNLOCK_OBJSTORE(OP_DSA);

1991 malloc_err:
1992         for (i = 4; i <= 7; i++)
1993                 {
1994                 if (a_key_template[i].pValue != NULL)
1995                         {
1996                         OPENSSL_free(a_key_template[i].pValue);
1997                         a_key_template[i].pValue = NULL;
1998                         }
1999                 }

2001         return (h_key);
2002         }

2004 /*
2005  * Create a private key object in the session from a given dsa structure
2006  * The *dsa_priv_num pointer is non-NULL for DSA private keys.
2007  */
2008 static CK_OBJECT_HANDLE pk11_get_private_dsa_key(DSA* dsa,
2009     DSA **key_ptr, BIGNUM **dsa_priv_num, CK_SESSION_HANDLE session)
2010         {
2011         CK_RV rv;
2012         CK_OBJECT_HANDLE h_key = CK_INVALID_HANDLE;
2013         CK_OBJECT_CLASS o_key = CKO_PRIVATE_KEY;
2014         int i;
2015         CK_ULONG found;
2016         CK_KEY_TYPE k_type = CKK_DSA;
2017         CK_ULONG ul_key_attr_count = 9;
2018         CK_BBOOL rollback = FALSE;

2020         /* Both CKA_TOKEN and CKA_SENSITIVE have to be FALSE for session keys */
2021         CK_ATTRIBUTE  a_key_template[] =
2022                 {
2023                 {CKA_CLASS, (void *) NULL, sizeof (CK_OBJECT_CLASS)},
2024                 {CKA_KEY_TYPE, (void *) NULL, sizeof (CK_KEY_TYPE)},
2025                 {CKA_TOKEN, &false, sizeof (true)},
2026                 {CKA_SENSITIVE, &false, sizeof (true)},
2027                 {CKA_SIGN, &true, sizeof (true)},
2028                 {CKA_PRIME, (void *)NULL, 0},           /* p */
2029                 {CKA_SUBPRIME, (void *)NULL, 0},        /* q */
2030                 {CKA_BASE, (void *)NULL, 0},            /* g */
2031                 {CKA_VALUE, (void *)NULL, 0}            /* priv_key - x */
2032                 };

2034         a_key_template[0].pValue = &o_key;
2035         a_key_template[1].pValue = &k_type;

2037         /* Put the private key components into the template */
2038         if (init_template_value(dsa->p, &a_key_template[5].pValue,
2039                 &a_key_template[5].ulValueLen) == 0 ||
2040             init_template_value(dsa->q, &a_key_template[6].pValue,
2041                 &a_key_template[6].ulValueLen) == 0 ||
```



```
2042             init_template_value(dsa->g, &a_key_template[7].pValue,
2043                 &a_key_template[7].ulValueLen) == 0 ||
2044             init_template_value(dsa->priv_key, &a_key_template[8].pValue,
2045                 &a_key_template[8].ulValueLen) == 0)
2046                 {
2047                 PK11err(PK11_F_GET_PRIV_DSA_KEY, PK11_R_MALLOC_FAILURE);
2048                 goto malloc_err;
2049                 }

2051         /* see find_lock array definition for more info on object locking */
2052         LOCK_OBJSTORE(OP_DSA);
2053         rv = pFuncList->C_FindObjectsInit(session, a_key_template,
2054                 ul_key_attr_count);

2056         if (rv != CKR_OK)
2057                 {
2058                 PK11err_add_data(PK11_F_GET_PRIV_DSA_KEY,
2059                     PK11_R_FINDOBJECTSINIT, rv);
2060                 goto err;
2061                 }

2063         rv = pFuncList->C_FindObjects(session, &h_key, 1, &found);

2065         if (rv != CKR_OK)
2066                 {
2067                 PK11err_add_data(PK11_F_GET_PRIV_DSA_KEY,
2068                     PK11_R_FINDOBJECTS, rv);
2069                 goto err;
2070                 }

2072         rv = pFuncList->C_FindObjectsFinal(session);

2074         if (rv != CKR_OK)
2075                 {
2076                 PK11err_add_data(PK11_F_GET_PRIV_DSA_KEY,
2077                     PK11_R_FINDOBJECTSFINAL, rv);
2078                 goto err;
2079                 }

2081         if (found == 0)
2082                 {
2083                 rv = pFuncList->C_CreateObject(session,
2084                         a_key_template, ul_key_attr_count, &h_key);
2085                 if (rv != CKR_OK)
2086                         {
2087                         PK11err_add_data(PK11_F_GET_PRIV_DSA_KEY,
2088                             PK11_R_CREATEOBJECT, rv);
2089                         goto err;
2090                         }
2091                 }

2093         if (dsa_priv_num != NULL)
2094                 if ((*dsa_priv_num = BN_dup(dsa->priv_key)) == NULL)
2095                         {
2096                         PK11err(PK11_F_GET_PRIV_DSA_KEY, PK11_R_MALLOC_FAILURE);
2097                         rollback = TRUE;
2098                         goto err;
2099                         }

2101         /* LINTED: E_CONSTANT_CONDITION */
2102         KEY_HANDLE_REFHOLD(h_key, OP_DSA, FALSE, rollback, err);
2103         if (key_ptr != NULL)
2104                 *key_ptr = dsa;

2106 err:
2107         if (rollback)
```

```
2108                {
2109                /*
2110                 * We do not care about the return value from C_DestroyObject()
2111                 * since we are doing rollback.
2112                 */
2113                if (found == 0)
2114                        (void) pFuncList->C_DestroyObject(session, h_key);
2115                h_key = CK_INVALID_HANDLE;
2116                }

2118        UNLOCK_OBJSTORE(OP_DSA);

2120 malloc_err:
2121        /*
2122         * 5 to 8 entries in the key template are key components.
2123         * They need to be freed apon exit or error.
2124         */
2125        for (i = 5; i <= 8; i++)
2126                {
2127                if (a_key_template[i].pValue != NULL)
2128                        {
2129                        (void) memset(a_key_template[i].pValue, 0,
2130                                a_key_template[i].ulValueLen);
2131                        OPENSSL_free(a_key_template[i].pValue);
2132                        a_key_template[i].pValue = NULL;
2133                        }
2134                }

2136        return (h_key);
2137        }

2139 /*
2140  * Check for cache miss and clean the object pointer and handle
2141  * in such case. Return 1 for cache hit, 0 for cache miss.
2142  */
2143 static int check_new_dsa_key_pub(PK11_SESSION *sp, DSA *dsa)
2144        {
2145        /*
2146         * Provide protection against DSA structure reuse by making the
2147         * check for cache hit stronger. Only public key component of DSA
2148         * key matters here so it is sufficient to compare it with value
2149         * cached in PK11_SESSION structure.
2150         */
2151        if ((sp->opdata_dsa_pub != dsa) ||
2152            (BN_cmp(sp->opdata_dsa_pub_num, dsa->pub_key) != 0))
2153                {
2154                /*
2155                 * We do not check the return value because even in case of
2156                 * failure the sp structure will have both key pointer
2157                 * and object handle cleaned and pk11_destroy_object()
2158                 * reports the failure to the OpenSSL error message buffer.
2159                 */
2160                (void) pk11_destroy_dsa_object_pub(sp, TRUE);
2161                return (0);
2162                }
2163        return (1);
2164        }

2166 /*
2167  * Check for cache miss and clean the object pointer and handle
2168  * in such case. Return 1 for cache hit, 0 for cache miss.
2169  */
2170 static int check_new_dsa_key_priv(PK11_SESSION *sp, DSA *dsa)
2171        {
2172        /*
2173         * Provide protection against DSA structure reuse by making the
```



```
2174         * check for cache hit stronger. Only private key component of DSA
2175         * key matters here so it is sufficient to compare it with value
2176         * cached in PK11_SESSION structure.
2177         */
2178        if ((sp->opdata_dsa_priv != dsa) ||
2179            (BN_cmp(sp->opdata_dsa_priv_num, dsa->priv_key) != 0))
2180                {
2181                /*
2182                 * We do not check the return value because even in case of
2183                 * failure the sp structure will have both key pointer
2184                 * and object handle cleaned and pk11_destroy_object()
2185                 * reports the failure to the OpenSSL error message buffer.
2186                 */
2187                (void) pk11_destroy_dsa_object_priv(sp, TRUE);
2188                return (0);
2189                }
2190        return (1);
2191        }
2192 #endif


2195 #ifndef OPENSSL_NO_DH
2196 /* The DH function implementation */
2197 /* ARGSUSED */
2198 static int pk11_DH_init(DH *dh)
2199        {
2200        return (1);
2201        }

2203 /* ARGSUSED */
2204 static int pk11_DH_finish(DH *dh)
2205        {
2206        return (1);
2207        }

2209 /*
2210  * Generate DH key-pair.
2211  *
2212  * Warning: Unlike OpenSSL's DH_generate_key(3) we ignore dh->priv_key
2213  * and override it even if it is set. OpenSSL does not touch dh->priv_key
2214  * if set and just computes dh->pub_key. It looks like PKCS#11 standard
2215  * is not capable of providing this functionality. This could be a problem
2216  * for applications relying on OpenSSL's semantics.
2217  */
2218 static int pk11_DH_generate_key(DH *dh)
2219        {
2220        CK_ULONG i;
2221        CK_RV rv, rv1;
2222        int reuse_mem_len = 0, ret = 0;
2223        PK11_SESSION *sp = NULL;
2224        CK_BYTE_PTR reuse_mem;

2226        CK_MECHANISM mechanism = {CKM_DH_PKCS_KEY_PAIR_GEN, NULL_PTR, 0};
2227        CK_OBJECT_HANDLE h_pub_key = CK_INVALID_HANDLE;
2228        CK_OBJECT_HANDLE h_priv_key = CK_INVALID_HANDLE;

2230        CK_ULONG ul_pub_key_attr_count = 3;
2231        CK_ATTRIBUTE pub_key_template[] =
2232                {
2233                {CKA_PRIVATE, &false, sizeof (false)},
2234                {CKA_PRIME, (void *)NULL, 0},
2235                {CKA_BASE, (void *)NULL, 0}
2236                };

2238        CK_ULONG ul_priv_key_attr_count = 3;
2239        CK_ATTRIBUTE priv_key_template[] =
```

```
2240                {
2241                {CKA_PRIVATE, &false, sizeof (false)},
2242                {CKA_SENSITIVE, &false, sizeof (false)},
2243                {CKA_DERIVE, &true, sizeof (true)}
2244                };

2246        CK_ULONG pub_key_attr_result_count = 1;
2247        CK_ATTRIBUTE pub_key_result[] =
2248                {
2249                {CKA_VALUE, (void *)NULL, 0}
2250                };

2252        CK_ULONG priv_key_attr_result_count = 1;
2253        CK_ATTRIBUTE priv_key_result[] =
2254                {
2255                {CKA_VALUE, (void *)NULL, 0}
2256                };

2258        pub_key_template[1].ulValueLen = BN_num_bytes(dh->p);
2259        if (pub_key_template[1].ulValueLen > 0)
2260                {
2261                /*
2262                 * We must not increase ulValueLen by DH_BUF_RESERVE since that
2263                 * could cause the same rounding problem. See definition of
2264                 * DH_BUF_RESERVE above.
2265                 */
2266                pub_key_template[1].pValue =
2267                        OPENSSL_malloc(pub_key_template[1].ulValueLen +
2268                        DH_BUF_RESERVE);
2269                if (pub_key_template[1].pValue == NULL)
2270                        {
2271                        PK11err(PK11_F_DH_GEN_KEY, PK11_R_MALLOC_FAILURE);
2272                        goto err;
2273                        }

2275                i = BN_bn2bin(dh->p, pub_key_template[1].pValue);
2276                }
2277        else
2278                goto err;

2280        pub_key_template[2].ulValueLen = BN_num_bytes(dh->g);
2281        if (pub_key_template[2].ulValueLen > 0)
2282                {
2283                pub_key_template[2].pValue =
2284                        OPENSSL_malloc(pub_key_template[2].ulValueLen +
2285                        DH_BUF_RESERVE);
2286                if (pub_key_template[2].pValue == NULL)
2287                        {
2288                        PK11err(PK11_F_DH_GEN_KEY, PK11_R_MALLOC_FAILURE);
2289                        goto err;
2290                        }

2292                i = BN_bn2bin(dh->g, pub_key_template[2].pValue);
2293                }
2294        else
2295                goto err;

2297        /*
2298         * Note: we are only using PK11_SESSION structure for getting
2299         *       a session handle. The objects created in this function are
2300         *       destroyed before return and thus not cached.
2301         */
2302        if ((sp = pk11_get_session(OP_DH)) == NULL)
2303                goto err;

2305        rv = pFuncList->C_GenerateKeyPair(sp->session,
```



```
2306                &mechanism,
2307                pub_key_template,
2308                ul_pub_key_attr_count,
2309                priv_key_template,
2310                ul_priv_key_attr_count,
2311                &h_pub_key,
2312                &h_priv_key);
2313        if (rv != CKR_OK)
2314                {
2315                PK11err_add_data(PK11_F_DH_GEN_KEY, PK11_R_GEN_KEY, rv);
2316                goto err;
2317                }

2319        /*
2320         * Reuse the larger memory allocated. We know the larger memory
2321         * should be sufficient for reuse.
2322         */
2323        if (pub_key_template[1].ulValueLen > pub_key_template[2].ulValueLen)
2324                {
2325                reuse_mem = pub_key_template[1].pValue;
2326                reuse_mem_len = pub_key_template[1].ulValueLen + DH_BUF_RESERVE;
2327                }
2328        else
2329                {
2330                reuse_mem = pub_key_template[2].pValue;
2331                reuse_mem_len = pub_key_template[2].ulValueLen + DH_BUF_RESERVE;
2332                }

2334        rv = pFuncList->C_GetAttributeValue(sp->session, h_pub_key,
2335                pub_key_result, pub_key_attr_result_count);
2336        rv1 = pFuncList->C_GetAttributeValue(sp->session, h_priv_key,
2337                priv_key_result, priv_key_attr_result_count);

2339        if (rv != CKR_OK || rv1 != CKR_OK)
2340                {
2341                rv = (rv != CKR_OK) ? rv : rv1;
2342                PK11err_add_data(PK11_F_DH_GEN_KEY,
2343                    PK11_R_GETATTRIBUTVALUE, rv);
2344                goto err;
2345                }

2347        if (((CK_LONG) pub_key_result[0].ulValueLen) <= 0 ||
2348                ((CK_LONG) priv_key_result[0].ulValueLen) <= 0)
2349                {
2350                PK11err(PK11_F_DH_GEN_KEY, PK11_R_GETATTRIBUTVALUE);
2351                goto err;
2352                }

2354        /* Reuse the memory allocated */
2355        pub_key_result[0].pValue = reuse_mem;
2356        pub_key_result[0].ulValueLen = reuse_mem_len;

2358        rv = pFuncList->C_GetAttributeValue(sp->session, h_pub_key,
2359                pub_key_result, pub_key_attr_result_count);

2361        if (rv != CKR_OK)
2362                {
2363                PK11err_add_data(PK11_F_DH_GEN_KEY,
2364                    PK11_R_GETATTRIBUTVALUE, rv);
2365                goto err;
2366                }

2368        if (pub_key_result[0].type == CKA_VALUE)
2369                {
2370                if (dh->pub_key == NULL)
2371                        if ((dh->pub_key = BN_new()) == NULL)
```

```
2372                                {
2373                                PK11err(PK11_F_DH_GEN_KEY,
2374                                        PK11_R_MALLOC_FAILURE);
2375                                goto err;
2376                                }
2377                dh->pub_key = BN_bin2bn(pub_key_result[0].pValue,
2378                        pub_key_result[0].ulValueLen, dh->pub_key);
2379                if (dh->pub_key == NULL)
2380                        {
2381                        PK11err(PK11_F_DH_GEN_KEY, PK11_R_MALLOC_FAILURE);
2382                        goto err;
2383                        }
2384                }

2386        /* Reuse the memory allocated */
2387        priv_key_result[0].pValue = reuse_mem;
2388        priv_key_result[0].ulValueLen = reuse_mem_len;

2390        rv = pFuncList->C_GetAttributeValue(sp->session, h_priv_key,
2391                priv_key_result, priv_key_attr_result_count);

2393        if (rv != CKR_OK)
2394                {
2395                PK11err_add_data(PK11_F_DH_GEN_KEY,
2396                    PK11_R_GETATTRIBUTVALUE, rv);
2397                goto err;
2398                }

2400        if (priv_key_result[0].type == CKA_VALUE)
2401                {
2402                if (dh->priv_key == NULL)
2403                        if ((dh->priv_key = BN_new()) == NULL)
2404                                {
2405                                PK11err(PK11_F_DH_GEN_KEY,
2406                                        PK11_R_MALLOC_FAILURE);
2407                                goto err;
2408                                }
2409                dh->priv_key = BN_bin2bn(priv_key_result[0].pValue,
2410                        priv_key_result[0].ulValueLen, dh->priv_key);
2411                if (dh->priv_key == NULL)
2412                        {
2413                        PK11err(PK11_F_DH_GEN_KEY, PK11_R_MALLOC_FAILURE);
2414                        goto err;
2415                        }
2416                }

2418        ret = 1;

2420 err:

2422        if (h_pub_key != CK_INVALID_HANDLE)
2423                {
2424                rv = pFuncList->C_DestroyObject(sp->session, h_pub_key);
2425                if (rv != CKR_OK)
2426                        {
2427                        PK11err_add_data(PK11_F_DH_GEN_KEY,
2428                                PK11_R_DESTROYOBJECT, rv);
2429                        }
2430                }

2432        if (h_priv_key != CK_INVALID_HANDLE)
2433                {
2434                rv = pFuncList->C_DestroyObject(sp->session, h_priv_key);
2435                if (rv != CKR_OK)
2436                        {
2437                        PK11err_add_data(PK11_F_DH_GEN_KEY,
```



```
2438                                PK11_R_DESTROYOBJECT, rv);
2439                        }
2440                }

2442        for (i = 1; i <= 2; i++)
2443                {
2444                if (pub_key_template[i].pValue != NULL)
2445                        {
2446                        OPENSSL_free(pub_key_template[i].pValue);
2447                        pub_key_template[i].pValue = NULL;
2448                        }
2449                }

2451        pk11_return_session(sp, OP_DH);
2452        return (ret);
2453        }

2455 static int pk11_DH_compute_key(unsigned char *key, const BIGNUM *pub_key,
2456        DH *dh)
2457        {
2458        int i;
2459        CK_MECHANISM mechanism = {CKM_DH_PKCS_DERIVE, NULL_PTR, 0};
2460        CK_OBJECT_CLASS key_class = CKO_SECRET_KEY;
2461        CK_KEY_TYPE key_type = CKK_GENERIC_SECRET;
2462        CK_OBJECT_HANDLE h_derived_key = CK_INVALID_HANDLE;
2463        CK_OBJECT_HANDLE h_key = CK_INVALID_HANDLE;

2465        CK_ULONG ul_priv_key_attr_count = 2;
2466        CK_ATTRIBUTE priv_key_template[] =
2467                {
2468                {CKA_CLASS, (void*) NULL, sizeof (key_class)},
2469                {CKA_KEY_TYPE, (void*) NULL, sizeof (key_type)},
2470                };

2472        CK_ULONG priv_key_attr_result_count = 1;
2473        CK_ATTRIBUTE priv_key_result[] =
2474                {
2475                {CKA_VALUE, (void *)NULL, 0}
2476                };

2478        CK_RV rv;
2479        int ret = -1;
2480        PK11_SESSION *sp = NULL;

2482        if (dh->priv_key == NULL)
2483                goto err;

2485        priv_key_template[0].pValue = &key_class;
2486        priv_key_template[1].pValue = &key_type;

2488        if ((sp = pk11_get_session(OP_DH)) == NULL)
2489                goto err;

2491        mechanism.ulParameterLen = BN_num_bytes(pub_key);
2492        mechanism.pParameter = OPENSSL_malloc(mechanism.ulParameterLen);
2493        if (mechanism.pParameter == NULL)
2494                {
2495                PK11err(PK11_F_DH_COMP_KEY, PK11_R_MALLOC_FAILURE);
2496                goto err;
2497                }
2498        BN_bn2bin(pub_key, mechanism.pParameter);

2500        (void) check_new_dh_key(sp, dh);

2502        h_key = sp->opdata_dh_key;
2503        if (h_key == CK_INVALID_HANDLE)
```

```
2504                 h_key = sp->opdata_dh_key =
2505                         pk11_get_dh_key((DH*) dh, &sp->opdata_dh,
2506                             &sp->opdata_dh_priv_num, sp->session);

2508         if (h_key == CK_INVALID_HANDLE)
2509                 {
2510                 PK11err(PK11_F_DH_COMP_KEY, PK11_R_CREATEOBJECT);
2511                 goto err;
2512                 }

2514         rv = pFuncList->C_DeriveKey(sp->session,
2515             &mechanism,
2516             h_key,
2517             priv_key_template,
2518             ul_priv_key_attr_count,
2519             &h_derived_key);
2520         if (rv != CKR_OK)
2521                 {
2522                 PK11err_add_data(PK11_F_DH_COMP_KEY, PK11_R_DERIVEKEY, rv);
2523                 goto err;
2524                 }

2526         rv = pFuncList->C_GetAttributeValue(sp->session, h_derived_key,
2527             priv_key_result, priv_key_attr_result_count);

2529         if (rv != CKR_OK)
2530                 {
2531                 PK11err_add_data(PK11_F_DH_COMP_KEY, PK11_R_GETATTRIBUTVALUE,
2532                     rv);
2533                 goto err;
2534                 }

2536         if (((CK_LONG) priv_key_result[0].ulValueLen) <= 0)
2537                 {
2538                 PK11err(PK11_F_DH_COMP_KEY, PK11_R_GETATTRIBUTVALUE);
2539                 goto err;
2540                 }
2541         priv_key_result[0].pValue =
2542                 OPENSSL_malloc(priv_key_result[0].ulValueLen);
2543         if (!priv_key_result[0].pValue)
2544                 {
2545                 PK11err(PK11_F_DH_COMP_KEY, PK11_R_MALLOC_FAILURE);
2546                 goto err;
2547                 }

2549         rv = pFuncList->C_GetAttributeValue(sp->session, h_derived_key,
2550                 priv_key_result, priv_key_attr_result_count);

2552         if (rv != CKR_OK)
2553                 {
2554                 PK11err_add_data(PK11_F_DH_COMP_KEY, PK11_R_GETATTRIBUTVALUE,
2555                     rv);
2556                 goto err;
2557                 }

2559         /*
2560          * OpenSSL allocates the output buffer 'key' which is the same
2561          * length of the public key. It is long enough for the derived key
2562          */
2563         if (priv_key_result[0].type == CKA_VALUE)
2564                 {
2565                 /*
2566                  * CKM_DH_PKCS_DERIVE mechanism is not supposed to strip
2567                  * leading zeros from a computed shared secret. However,
2568                  * OpenSSL always did it so we must do the same here. The
2569                  * vagueness of the spec regarding leading zero bytes was
```



```
2570                  * finally cleared with TLS 1.1 (RFC 4346) saying that leading
2571                  * zeros are stripped before the computed data is used as the
2572                  * pre-master secret.
2573                  */
2574                 for (i = 0; i < priv_key_result[0].ulValueLen; ++i)
2575                         {
2576                         if (((char *)priv_key_result[0].pValue)[i] != 0)
2577                                 break;
2578                         }

2580                 (void) memcpy(key, ((char *)priv_key_result[0].pValue) + i,
2581                         priv_key_result[0].ulValueLen - i);
2582                 ret = priv_key_result[0].ulValueLen - i;
2583                 }

2585 err:

2587         if (h_derived_key != CK_INVALID_HANDLE)
2588                 {
2589                 rv = pFuncList->C_DestroyObject(sp->session, h_derived_key);
2590                 if (rv != CKR_OK)
2591                         {
2592                         PK11err_add_data(PK11_F_DH_COMP_KEY,
2593                             PK11_R_DESTROYOBJECT, rv);
2594                         }
2595                 }
2596         if (priv_key_result[0].pValue)
2597                 {
2598                 OPENSSL_free(priv_key_result[0].pValue);
2599                 priv_key_result[0].pValue = NULL;
2600                 }

2602         if (mechanism.pParameter)
2603                 {
2604                 OPENSSL_free(mechanism.pParameter);
2605                 mechanism.pParameter = NULL;
2606                 }

2608         pk11_return_session(sp, OP_DH);
2609         return (ret);
2610         }


2613 static CK_OBJECT_HANDLE pk11_get_dh_key(DH* dh,
2614         DH **key_ptr, BIGNUM **dh_priv_num, CK_SESSION_HANDLE session)
2615         {
2616         CK_RV rv;
2617         CK_OBJECT_HANDLE h_key = CK_INVALID_HANDLE;
2618         CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
2619         CK_KEY_TYPE key_type = CKK_DH;
2620         CK_ULONG found;
2621         CK_BBOOL rollback = FALSE;
2622         int i;

2624         CK_ULONG ul_key_attr_count = 7;
2625         CK_ATTRIBUTE key_template[] =
2626                 {
2627                 {CKA_CLASS, (void*) NULL, sizeof (class)},
2628                 {CKA_KEY_TYPE, (void*) NULL, sizeof (key_type)},
2629                 {CKA_DERIVE, &true, sizeof (true)},
2630                 {CKA_PRIVATE, &false, sizeof (false)},
2631                 {CKA_PRIME, (void *) NULL, 0},
2632                 {CKA_BASE, (void *) NULL, 0},
2633                 {CKA_VALUE, (void *) NULL, 0},
2634                 };
```

```
2636         key_template[0].pValue = &class;
2637         key_template[1].pValue = &key_type;

2639         key_template[4].ulValueLen = BN_num_bytes(dh->p);
2640         key_template[4].pValue = (CK_VOID_PTR)OPENSSL_malloc(
2641                 (size_t)key_template[4].ulValueLen);
2642         if (key_template[4].pValue == NULL)
2643                 {
2644                 PK11err(PK11_F_GET_DH_KEY, PK11_R_MALLOC_FAILURE);
2645                 goto malloc_err;
2646                 }

2648         BN_bn2bin(dh->p, key_template[4].pValue);

2650         key_template[5].ulValueLen = BN_num_bytes(dh->g);
2651         key_template[5].pValue = (CK_VOID_PTR)OPENSSL_malloc(
2652                 (size_t)key_template[5].ulValueLen);
2653         if (key_template[5].pValue == NULL)
2654                 {
2655                 PK11err(PK11_F_GET_DH_KEY, PK11_R_MALLOC_FAILURE);
2656                 goto malloc_err;
2657                 }

2659         BN_bn2bin(dh->g, key_template[5].pValue);

2661         key_template[6].ulValueLen = BN_num_bytes(dh->priv_key);
2662         key_template[6].pValue = (CK_VOID_PTR)OPENSSL_malloc(
2663                 (size_t)key_template[6].ulValueLen);
2664         if (key_template[6].pValue == NULL)
2665                 {
2666                 PK11err(PK11_F_GET_DH_KEY, PK11_R_MALLOC_FAILURE);
2667                 goto malloc_err;
2668                 }

2670         BN_bn2bin(dh->priv_key, key_template[6].pValue);

2672         /* see find_lock array definition for more info on object locking */
2673         LOCK_OBJSTORE(OP_DH);
2674         rv = pFuncList->C_FindObjectsInit(session, key_template,
2675                 ul_key_attr_count);

2677         if (rv != CKR_OK)
2678                 {
2679                 PK11err_add_data(PK11_F_GET_DH_KEY, PK11_R_FINDOBJECTSINIT, rv);
2680                 goto err;
2681                 }

2683         rv = pFuncList->C_FindObjects(session, &h_key, 1, &found);

2685         if (rv != CKR_OK)
2686                 {
2687                 PK11err_add_data(PK11_F_GET_DH_KEY, PK11_R_FINDOBJECTS, rv);
2688                 goto err;
2689                 }

2691         rv = pFuncList->C_FindObjectsFinal(session);

2693         if (rv != CKR_OK)
2694                 {
2695                 PK11err_add_data(PK11_F_GET_DH_KEY, PK11_R_FINDOBJECTSFINAL,
2696                     rv);
2697                 goto err;
2698                 }

2700         if (found == 0)
2701                 {
```



```
2702                 rv = pFuncList->C_CreateObject(session,
2703                         key_template, ul_key_attr_count, &h_key);
2704                 if (rv != CKR_OK)
2705                         {
2706                         PK11err_add_data(PK11_F_GET_DH_KEY, PK11_R_CREATEOBJECT,
2707                             rv);
2708                         goto err;
2709                         }
2710                 }

2712         if (dh_priv_num != NULL)
2713                 if ((*dh_priv_num = BN_dup(dh->priv_key)) == NULL)
2714                         {
2715                         PK11err(PK11_F_GET_DH_KEY, PK11_R_MALLOC_FAILURE);
2716                         rollback = TRUE;
2717                         goto err;
2718                         }

2720         /* LINTED: E_CONSTANT_CONDITION */
2721         KEY_HANDLE_REFHOLD(h_key, OP_DH, FALSE, rollback, err);
2722         if (key_ptr != NULL)
2723                 *key_ptr = dh;

2725 err:
2726         if (rollback)
2727                 {
2728                 /*
2729                  * We do not care about the return value from C_DestroyObject()
2730                  * since we are doing rollback.
2731                  */
2732                 if (found == 0)
2733                         (void) pFuncList->C_DestroyObject(session, h_key);
2734                 h_key = CK_INVALID_HANDLE;
2735                 }

2737         UNLOCK_OBJSTORE(OP_DH);

2739 malloc_err:
2740         for (i = 4; i <= 6; i++)
2741                 {
2742                 if (key_template[i].pValue != NULL)
2743                         {
2744                         OPENSSL_free(key_template[i].pValue);
2745                         key_template[i].pValue = NULL;
2746                         }
2747                 }

2749         return (h_key);
2750         }

2752 /*
2753  * Check for cache miss and clean the object pointer and handle
2754  * in such case. Return 1 for cache hit, 0 for cache miss.
2755  *
2756  * Note: we rely on pk11_destroy_dh_key_objects() to set sp->opdata_dh
2757  *       to CK_INVALID_HANDLE even when it fails to destroy the object.
2758  */
2759 static int check_new_dh_key(PK11_SESSION *sp, DH *dh)
2760         {
2761         /*
2762          * Provide protection against DH structure reuse by making the
2763          * check for cache hit stronger. Private key component of DH key
2764          * is unique so it is sufficient to compare it with value cached
2765          * in PK11_SESSION structure.
2766          */
2767         if ((sp->opdata_dh != dh) ||
```

```
2768             (BN_cmp(sp->opdata_dh_priv_num, dh->priv_key) != 0))
2769                 {
2770                 /*
2771                  * We do not check the return value because even in case of
2772                  * failure the sp structure will have both key pointer
2773                  * and object handle cleaned and pk11_destroy_object()
2774                  * reports the failure to the OpenSSL error message buffer.
2775                  */
2776                 (void) pk11_destroy_dh_object(sp, TRUE);
2777                 return (0);
2778                 }
2779         return (1);
2780         }
2781 #endif

2783 /*
2784  * Local function to simplify key template population
2785  * Return 0 -- error, 1 -- no error
2786  */
2787 static int init_template_value(BIGNUM *bn, CK_VOID_PTR *p_value,
2788         CK_ULONG *ul_value_len)
2789         {
2790         CK_ULONG len = BN_num_bytes(bn);
2791         if (len == 0)
2792                 return (1);

2794         *ul_value_len = len;
2795         *p_value = (CK_VOID_PTR)OPENSSL_malloc((size_t)*ul_value_len);
2796         if (*p_value == NULL)
2797                 return (0);

2799         BN_bn2bin(bn, *p_value);

2801         return (1);
2802         }

2804 #endif  /* OPENSSL_NO_HW_PK11 */
2805 #endif  /* OPENSSL_NO_HW */
2806 #endif /* ! codereview */
```

```
**********************************************************
   15521 Wed Aug 13 19:32:59 2014
new/usr/src/lib/openssl/libsunw_crypto/evp/e_aes_cbc_hmac_sha1.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* ====================================================================
 * Copyright (c) 2011-2013 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 */

#include <openssl/opensslconf.h>

#include <stdio.h>
#include <string.h>

#if !defined(OPENSSL_NO_AES) && !defined(OPENSSL_NO_SHA1)

#include <openssl/evp.h>
#include <openssl/objects.h>
#include <openssl/aes.h>
#include <openssl/sha.h>
#include "evp_locl.h"
```



```
#ifndef EVP_CIPH_FLAG_AEAD_CIPHER
#define EVP_CIPH_FLAG_AEAD_CIPHER       0x200000
#define EVP_CTRL_AEAD_TLS1_AAD          0x16
#define EVP_CTRL_AEAD_SET_MAC_KEY       0x17
#endif

#if !defined(EVP_CIPH_FLAG_DEFAULT_ASN1)
#define EVP_CIPH_FLAG_DEFAULT_ASN1 0
#endif

#define TLS1_1_VERSION 0x0302

typedef struct
    {
    AES_KEY             ks;
    SHA_CTX             head,tail,md;
    size_t              payload_length; /* AAD length in decrypt case */
    union {
        unsigned int    tls_ver;
        unsigned char   tls_aad[16];    /* 13 used */
    } aux;
    } EVP_AES_HMAC_SHA1;

#define NO_PAYLOAD_LENGTH       ((size_t)-1)

#if     defined(AES_ASM) &&     ( \
        defined(__x86_64)       || defined(__x86_64__)  || \
        defined(_M_AMD64)       || defined(_M_X64)      || \
        defined(__INTEL__)      )

#if defined(__GNUC__) && __GNUC__>=2 && !defined(PEDANTIC)
# define BSWAP(x) ({ unsigned int r=(x); __asm__ ("bswapl %0":"=r"(r):"0"(r)); r
# define BSWAP(x) ({ unsigned int r=(x); asm ("bswapl %0":"=r"(r):"0"(r)); r; })
#endif

extern unsigned int OPENSSL_ia32cap_P[2];
#define AESNI_CAPABLE   (1<<(57-32))

int aesni_set_encrypt_key(const unsigned char *userKey, int bits,
                          AES_KEY *key);
int aesni_set_decrypt_key(const unsigned char *userKey, int bits,
                          AES_KEY *key);

void aesni_cbc_encrypt(const unsigned char *in,
                       unsigned char *out,
                       size_t length,
                       const AES_KEY *key,
                       unsigned char *ivec, int enc);

void aesni_cbc_sha1_enc (const void *inp, void *out, size_t blocks,
                const AES_KEY *key, unsigned char iv[16],
                SHA_CTX *ctx,const void *in0);

#define data(ctx) ((EVP_AES_HMAC_SHA1 *)(ctx)->cipher_data)

static int aesni_cbc_hmac_sha1_init_key(EVP_CIPHER_CTX *ctx,
                        const unsigned char *inkey,
                        const unsigned char *iv, int enc)
        {
        EVP_AES_HMAC_SHA1 *key = data(ctx);
        int ret;

        if (enc)
                ret=aesni_set_encrypt_key(inkey,ctx->key_len*8,&key->ks);
        else
```

```
 127                 ret=aesni_set_decrypt_key(inkey,ctx->key_len*8,&key->ks);

 129         SHA1_Init(&key->head);  /* handy when benchmarking */
 130         key->tail = key->head;
 131         key->md   = key->head;

 133         key->payload_length = NO_PAYLOAD_LENGTH;

 135         return ret<0?0:1;
 136         }

 138 #define STITCHED_CALL

 140 #if !defined(STITCHED_CALL)
 141 #define aes_off 0
 142 #endif

 144 void sha1_block_data_order (void *c,const void *p,size_t len);

 146 static void sha1_update(SHA_CTX *c,const void *data,size_t len)
 147 {       const unsigned char *ptr = data;
 148         size_t res;

 150         if ((res = c->num)) {
 151                 res = SHA_CBLOCK-res;
 152                 if (len<res) res=len;
 153                 SHA1_Update (c,ptr,res);
 154                 ptr += res;
 155                 len -= res;
 156         }

 158         res = len % SHA_CBLOCK;
 159         len -= res;

 161         if (len) {
 162                 sha1_block_data_order(c,ptr,len/SHA_CBLOCK);

 164                 ptr += len;
 165                 c->Nh += len>>29;
 166                 c->Nl += len<<=3;
 167                 if (c->Nl<(unsigned int)len) c->Nh++;
 168         }

 170         if (res)
 171                 SHA1_Update(c,ptr,res);
 172 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   21398 Wed Aug 13 19:32:59 2014
new/usr/src/lib/openssl/libsunw_crypto/ex_data.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/ex_data.c */

   3 /*
   4  * Overhaul notes;
   5  *
   6  * This code is now *mostly* thread-safe. It is now easier to understand in what
   7  * ways it is safe and in what ways it is not, which is an improvement. Firstly,
   8  * all per-class stacks and index-counters for ex_data are stored in the same
   9  * global LHASH table (keyed by class). This hash table uses locking for all
  10  * access with the exception of CRYPTO_cleanup_all_ex_data(), which must only be
  11  * called when no other threads can possibly race against it (even if it was
  12  * locked, the race would mean it's possible the hash table might have been
  13  * recreated after the cleanup). As classes can only be added to the hash table,
  14  * and within each class, the stack of methods can only be incremented, the
  15  * locking mechanics are simpler than they would otherwise be. For example, the
  16  * new/dup/free ex_data functions will lock the hash table, copy the method
  17  * pointers it needs from the relevant class, then unlock the hash table before
  18  * actually applying those method pointers to the task of the new/dup/free
  19  * operations. As they can't be removed from the method-stack, only
  20  * supplemented, there's no race conditions associated with using them outside
  21  * the lock. The get/set_ex_data functions are not locked because they do not
  22  * involve this global state at all - they operate directly with a previously
  23  * obtained per-class method index and a particular "ex_data" variable. These
  24  * variables are usually instantiated per-context (eg. each RSA structure has
  25  * one) so locking on read/write access to that variable can be locked locally
  26  * if required (eg. using the "RSA" lock to synchronise access to a
  27  * per-RSA-structure ex_data variable if required).
  28  * [Geoff]
  29  */

  31 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
  32  * All rights reserved.
  33  *
  34  * This package is an SSL implementation written
  35  * by Eric Young (eay@cryptsoft.com).
  36  * The implementation was written so as to conform with Netscapes SSL.
  37  *
  38  * This library is free for commercial and non-commercial use as long as
  39  * the following conditions are aheared to.  The following conditions
  40  * apply to all code found in this distribution, be it the RC4, RSA,
  41  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
  42  * included with this distribution is covered by the same copyright terms
  43  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  44  *
  45  * Copyright remains Eric Young's, and as such any Copyright notices in
  46  * the code are not to be removed.
  47  * If this package is used in a product, Eric Young should be given attribution
  48  * as the author of the parts of the library used.
  49  * This can be in the form of a textual message at program startup or
  50  * in documentation (online or textual) provided with the package.
  51  *
  52  * Redistribution and use in source and binary forms, with or without
  53  * modification, are permitted provided that the following conditions
  54  * are met:
  55  * 1. Redistributions of source code must retain the copyright
  56  *    notice, this list of conditions and the following disclaimer.
  57  * 2. Redistributions in binary form must reproduce the above copyright
  58  *    notice, this list of conditions and the following disclaimer in the
  59  *    documentation and/or other materials provided with the distribution.
  60  * 3. All advertising materials mentioning features or use of this software
  61  *    must display the following acknowledgement:
```



```
  62  *    "This product includes cryptographic software written by
  63  *     Eric Young (eay@cryptsoft.com)"
  64  *    The word 'cryptographic' can be left out if the rouines from the library
  65  *    being used are not cryptographic related :-).
  66  * 4. If you include any Windows specific code (or a derivative thereof) from
  67  *    the apps directory (application code) you must include an acknowledgement:
  68  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  69  *
  70  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
  71  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  72  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  73  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
  74  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  75  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  76  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  77  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  78  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  79  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  80  * SUCH DAMAGE.
  81  *
  82  * The licence and distribution terms for any publically available version or
  83  * derivative of this code cannot be changed.  i.e. this code cannot simply be
  84  * copied and put under another distribution licence
  85  * [including the GNU Public Licence.]
  86  */
  87 /* ====================================================================
  88  * Copyright (c) 1998-2001 The OpenSSL Project.  All rights reserved.
  89  *
  90  * Redistribution and use in source and binary forms, with or without
  91  * modification, are permitted provided that the following conditions
  92  * are met:
  93  *
  94  * 1. Redistributions of source code must retain the above copyright
  95  *    notice, this list of conditions and the following disclaimer.
  96  *
  97  * 2. Redistributions in binary form must reproduce the above copyright
  98  *    notice, this list of conditions and the following disclaimer in
  99  *    the documentation and/or other materials provided with the
 100  *    distribution.
 101  *
 102  * 3. All advertising materials mentioning features or use of this
 103  *    software must display the following acknowledgment:
 104  *    "This product includes software developed by the OpenSSL Project
 105  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 106  *
 107  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 108  *    endorse or promote products derived from this software without
 109  *    prior written permission. For written permission, please contact
 110  *    openssl-core@openssl.org.
 111  *
 112  * 5. Products derived from this software may not be called "OpenSSL"
 113  *    nor may "OpenSSL" appear in their names without prior written
 114  *    permission of the OpenSSL Project.
 115  *
 116  * 6. Redistributions of any form whatsoever must retain the following
 117  *    acknowledgment:
 118  *    "This product includes software developed by the OpenSSL Project
 119  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 120  *
 121  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ''AS IS'' AND ANY
 122  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 123  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 124  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 125  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 126  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 127  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
```

```
 128  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 129  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 130  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 131  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 132  * OF THE POSSIBILITY OF SUCH DAMAGE.
 133  * ====================================================================
 134  *
 135  * This product includes cryptographic software written by Eric Young
 136  * (eay@cryptsoft.com).  This product includes software written by Tim
 137  * Hudson (tjh@cryptsoft.com).
 138  *
 139  */

 141 #include <cryptlib.h>
 141 #include "cryptlib.h"
 142 #include <openssl/lhash.h>

 144 /* What an "implementation of ex_data functionality" looks like */
 145 struct st_CRYPTO_EX_DATA_IMPL
 146         {
 147         /*********************/
 148         /* GLOBAL OPERATIONS */
 149         /* Return a new class index */
 150         int (*cb_new_class)(void);
 151         /* Cleanup all state used by the implementation */
 152         void (*cb_cleanup)(void);
 153         /************************/
 154         /* PER-CLASS OPERATIONS */
 155         /* Get a new method index within a class */
 156         int (*cb_get_new_index)(int class_index, long argl, void *argp,
 157                         CRYPTO_EX_new *new_func, CRYPTO_EX_dup *dup_func,
 158                         CRYPTO_EX_free *free_func);
 159         /* Initialise a new CRYPTO_EX_DATA of a given class */
 160         int (*cb_new_ex_data)(int class_index, void *obj,
 161                         CRYPTO_EX_DATA *ad);
 162         /* Duplicate a CRYPTO_EX_DATA of a given class onto a copy */
 163         int (*cb_dup_ex_data)(int class_index, CRYPTO_EX_DATA *to,
 164                         CRYPTO_EX_DATA *from);
 165         /* Cleanup a CRYPTO_EX_DATA of a given class */
 166         void (*cb_free_ex_data)(int class_index, void *obj,
 167                         CRYPTO_EX_DATA *ad);
 168         };

 170 /* The implementation we use at run-time */
 171 static const CRYPTO_EX_DATA_IMPL *impl = NULL;

 173 /* To call "impl" functions, use this macro rather than referring to 'impl' dire
 174  * EX_IMPL(get_new_index)(...); */
 175 #define EX_IMPL(a) impl->cb_##a

 177 /* Predeclare the "default" ex_data implementation */
 178 static int int_new_class(void);
 179 static void int_cleanup(void);
 180 static int int_get_new_index(int class_index, long argl, void *argp,
 181                 CRYPTO_EX_new *new_func, CRYPTO_EX_dup *dup_func,
 182                 CRYPTO_EX_free *free_func);
 183 static int int_new_ex_data(int class_index, void *obj,
 184                 CRYPTO_EX_DATA *ad);
 185 static int int_dup_ex_data(int class_index, CRYPTO_EX_DATA *to,
 186                 CRYPTO_EX_DATA *from);
 187 static void int_free_ex_data(int class_index, void *obj,
 188                 CRYPTO_EX_DATA *ad);
 189 static CRYPTO_EX_DATA_IMPL impl_default =
 190         {
 191         int_new_class,
 192         int_cleanup,
```



```
 193         int_get_new_index,
 194         int_new_ex_data,
 195         int_dup_ex_data,
 196         int_free_ex_data
 197         };

 199 /* Internal function that checks whether "impl" is set and if not, sets it to
 200  * the default. */
 201 static void impl_check(void)
 202         {
 203         CRYPTO_w_lock(CRYPTO_LOCK_EX_DATA);
 204         if(!impl)
 205                 impl = &impl_default;
 206         CRYPTO_w_unlock(CRYPTO_LOCK_EX_DATA);
 207         }
 208 /* A macro wrapper for impl_check that first uses a non-locked test before
 209  * invoking the function (which checks again inside a lock). */
 210 #define IMPL_CHECK if(!impl) impl_check();

 212 /* API functions to get/set the "ex_data" implementation */
 213 const CRYPTO_EX_DATA_IMPL *CRYPTO_get_ex_data_implementation(void)
 214         {
 215         IMPL_CHECK
 216         return impl;
 217         }
 218 int CRYPTO_set_ex_data_implementation(const CRYPTO_EX_DATA_IMPL *i)
 219         {
 220         int toret = 0;
 221         CRYPTO_w_lock(CRYPTO_LOCK_EX_DATA);
 222         if(!impl)
 223                 {
 224                 impl = i;
 225                 toret = 1;
 226                 }
 227         CRYPTO_w_unlock(CRYPTO_LOCK_EX_DATA);
 228         return toret;
 229         }

 231 /****************************************************************************/
 232 /* Interal (default) implementation of "ex_data" support. API functions are
 233  * further down. */

 235 /* The type that represents what each "class" used to implement locally. A STACK
 236  * of CRYPTO_EX_DATA_FUNCS plus a index-counter. The 'class_index' is the global
 237  * value representing the class that is used to distinguish these items. */
 238 typedef struct st_ex_class_item {
 239         int class_index;
 240         STACK_OF(CRYPTO_EX_DATA_FUNCS) *meth;
 241         int meth_num;
 242 } EX_CLASS_ITEM;
______unchanged_portion_omitted_
```

```
**********************************************************
   23509 Wed Aug 13 19:33:00 2014
new/usr/src/lib/openssl/libsunw_crypto/mem_dbg.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/mem_dbg.c */
   2 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
   3  * All rights reserved.
   4  *
   5  * This package is an SSL implementation written
   6  * by Eric Young (eay@cryptsoft.com).
   7  * The implementation was written so as to conform with Netscapes SSL.
   8  *
   9  * This library is free for commercial and non-commercial use as long as
  10  * the following conditions are aheared to.  The following conditions
  11  * apply to all code found in this distribution, be it the RC4, RSA,
  12  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
  13  * included with this distribution is covered by the same copyright terms
  14  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  15  *
  16  * Copyright remains Eric Young's, and as such any Copyright notices in
  17  * the code are not to be removed.
  18  * If this package is used in a product, Eric Young should be given attribution
  19  * as the author of the parts of the library used.
  20  * This can be in the form of a textual message at program startup or
  21  * in documentation (online or textual) provided with the package.
  22  *
  23  * Redistribution and use in source and binary forms, with or without
  24  * modification, are permitted provided that the following conditions
  25  * are met:
  26  * 1. Redistributions of source code must retain the copyright
  27  *    notice, this list of conditions and the following disclaimer.
  28  * 2. Redistributions in binary form must reproduce the above copyright
  29  *    notice, this list of conditions and the following disclaimer in the
  30  *    documentation and/or other materials provided with the distribution.
  31  * 3. All advertising materials mentioning features or use of this software
  32  *    must display the following acknowledgement:
  33  *    "This product includes cryptographic software written by
  34  *     Eric Young (eay@cryptsoft.com)"
  35  *    The word 'cryptographic' can be left out if the rouines from the library
  36  *    being used are not cryptographic related :-).
  37  * 4. If you include any Windows specific code (or a derivative thereof) from
  38  *    the apps directory (application code) you must include an acknowledgement:
  39  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  40  *
  41  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
  42  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  43  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  44  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
  45  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  46  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  47  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  48  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  49  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  50  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  51  * SUCH DAMAGE.
  52  *
  53  * The licence and distribution terms for any publically available version or
  54  * derivative of this code cannot be changed.  i.e. this code cannot simply be
  55  * copied and put under another distribution licence
  56  * [including the GNU Public Licence.]
  57  */
  58 /* ====================================================================
  59  * Copyright (c) 1998-2006 The OpenSSL Project.  All rights reserved.
  60  *
  61  * Redistribution and use in source and binary forms, with or without
```



```
  62  * modification, are permitted provided that the following conditions
  63  * are met:
  64  *
  65  * 1. Redistributions of source code must retain the above copyright
  66  *    notice, this list of conditions and the following disclaimer.
  67  *
  68  * 2. Redistributions in binary form must reproduce the above copyright
  69  *    notice, this list of conditions and the following disclaimer in
  70  *    the documentation and/or other materials provided with the
  71  *    distribution.
  72  *
  73  * 3. All advertising materials mentioning features or use of this
  74  *    software must display the following acknowledgment:
  75  *    "This product includes software developed by the OpenSSL Project
  76  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
  77  *
  78  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  79  *    endorse or promote products derived from this software without
  80  *    prior written permission. For written permission, please contact
  81  *    openssl-core@openssl.org.
  82  *
  83  * 5. Products derived from this software may not be called "OpenSSL"
  84  *    nor may "OpenSSL" appear in their names without prior written
  85  *    permission of the OpenSSL Project.
  86  *
  87  * 6. Redistributions of any form whatsoever must retain the following
  88  *    acknowledgment:
  89  *    "This product includes software developed by the OpenSSL Project
  90  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
  91  *
  92  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ''AS IS'' AND ANY
  93  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  94  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  95  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  96  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  97  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  98  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  99  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 100  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 101  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 102  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 103  * OF THE POSSIBILITY OF SUCH DAMAGE.
 104  * ====================================================================
 105  *
 106  * This product includes cryptographic software written by Eric Young
 107  * (eay@cryptsoft.com).  This product includes software written by Tim
 108  * Hudson (tjh@cryptsoft.com).
 109  *
 110  */

 112 #include <stdio.h>
 113 #include <stdlib.h>
 114 #include <time.h>
 115 #include <cryptlib.h>
 115 #include "cryptlib.h"
 116 #include <openssl/crypto.h>
 117 #include <openssl/buffer.h>
 118 #include <openssl/bio.h>
 119 #include <openssl/lhash.h>

 121 static int mh_mode=CRYPTO_MEM_CHECK_OFF;
 122 /* The state changes to CRYPTO_MEM_CHECK_ON | CRYPTO_MEM_CHECK_ENABLE
 123  * when the application asks for it (usually after library initialisation
 124  * for which no book-keeping is desired).
 125  *
 126  * State CRYPTO_MEM_CHECK_ON exists only temporarily when the library
```

```
 127  * thinks that certain allocations should not be checked (e.g. the data
 128  * structures used for memory checking).  It is not suitable as an initial
 129  * state: the library will unexpectedly enable memory checking when it
 130  * executes one of those sections that want to disable checking
 131  * temporarily.
 132  *
 133  * State CRYPTO_MEM_CHECK_ENABLE without ..._ON makes no sense whatsoever.
 134  */

 136 static unsigned long order = 0; /* number of memory requests */

 138 DECLARE_LHASH_OF(MEM);
 139 static LHASH_OF(MEM) *mh=NULL; /* hash-table of memory requests
 140                                 * (address as key); access requires
 141                                 * MALLOC2 lock */


 144 typedef struct app_mem_info_st
 145 /* For application-defined information (static C-string 'info')
 146  * to be displayed in memory leak list.
 147  * Each thread has its own stack.  For applications, there is
 148  *   CRYPTO_push_info("...")     to push an entry,
 149  *   CRYPTO_pop_info()           to pop an entry,
 150  *   CRYPTO_remove_all_info()    to pop all entries.
 151  */
 152         {
 153         CRYPTO_THREADID threadid;
 154         const char *file;
 155         int line;
 156         const char *info;
 157         struct app_mem_info_st *next; /* tail of thread's stack */
 158         int references;
 159         } APP_INFO;

 161 static void app_info_free(APP_INFO *);

 163 DECLARE_LHASH_OF(APP_INFO);
 164 static LHASH_OF(APP_INFO) *amih=NULL; /* hash-table with those
 165                                        * app_mem_info_st's that are at
 166                                        * the top of their thread's
 167                                        * stack (with 'thread' as key);
 168                                        * access requires MALLOC2
 169                                        * lock */

 171 typedef struct mem_st
 172 /* memory-block description */
 173         {
 174         void *addr;
 175         int num;
 176         const char *file;
 177         int line;
 178         CRYPTO_THREADID threadid;
 179         unsigned long order;
 180         time_t time;
 181         APP_INFO *app_info;
 182         } MEM;

 184 static long options =             /* extra information to be recorded */
 185 #if defined(CRYPTO_MDEBUG_TIME) || defined(CRYPTO_MDEBUG_ALL)
 186         V_CRYPTO_MDEBUG_TIME |
 187 #endif
 188 #if defined(CRYPTO_MDEBUG_THREAD) || defined(CRYPTO_MDEBUG_ALL)
 189         V_CRYPTO_MDEBUG_THREAD |
 190 #endif
 191         0;
```



```
 194 static unsigned int num_disable = 0; /* num_disable > 0
 195                                       *     iff
 196                                       * mh_mode == CRYPTO_MEM_CHECK_ON (w/o ..._
 197                                       */

 199 /* Valid iff num_disable > 0.  CRYPTO_LOCK_MALLOC2 is locked exactly in this
 200  * case (by the thread named in disabling_thread).
 201  */
 202 static CRYPTO_THREADID disabling_threadid;

 204 static void app_info_free(APP_INFO *inf)
 205         {
 206         if (--(inf->references) <= 0)
 207                 {
 208                 if (inf->next != NULL)
 209                         {
 210                         app_info_free(inf->next);
 211                         }
 212                 OPENSSL_free(inf);
 213                 }
 214         }

 216 int CRYPTO_mem_ctrl(int mode)
 217         {
 218         int ret=mh_mode;

 220         CRYPTO_w_lock(CRYPTO_LOCK_MALLOC);
 221         switch (mode)
 222                 {
 223         /* for applications (not to be called while multiple threads
 224          * use the library): */
 225         case CRYPTO_MEM_CHECK_ON: /* aka MemCheck_start() */
 226                 mh_mode = CRYPTO_MEM_CHECK_ON|CRYPTO_MEM_CHECK_ENABLE;
 227                 num_disable = 0;
 228                 break;
 229         case CRYPTO_MEM_CHECK_OFF: /* aka MemCheck_stop() */
 230                 mh_mode = 0;
 231                 num_disable = 0; /* should be true *before* MemCheck_stop is use
 232                                     or there'll be a lot of confusion */
 233                 break;

 235         /* switch off temporarily (for library-internal use): */
 236         case CRYPTO_MEM_CHECK_DISABLE: /* aka MemCheck_off() */
 237                 if (mh_mode & CRYPTO_MEM_CHECK_ON)
 238                         {
 239                         CRYPTO_THREADID cur;
 240                         CRYPTO_THREADID_current(&cur);
 241                         if (!num_disable || CRYPTO_THREADID_cmp(&disabling_threa
 242                                 {
 243                                 /* Long-time lock CRYPTO_LOCK_MALLOC2 must not b
 244                                  * we're holding CRYPTO_LOCK_MALLOC, or we'll de
 245                                  * somebody else holds CRYPTO_LOCK_MALLOC2 (and
 246                                  * it because we block entry to this function).
 247                                  * Give them a chance, first, and then claim the
 248                                  * appropriate order (long-time lock first).
 249                                  */
 250                                 CRYPTO_w_unlock(CRYPTO_LOCK_MALLOC);
 251                                 /* Note that after we have waited for CRYPTO_LOC
 252                                  * and CRYPTO_LOCK_MALLOC, we'll still be in the
 253                                  * "case" and "if" branch because MemCheck_start
 254                                  * MemCheck_stop may never be used while there a
 255                                  * OpenSSL threads. */
 256                                 CRYPTO_w_lock(CRYPTO_LOCK_MALLOC2);
 257                                 CRYPTO_w_lock(CRYPTO_LOCK_MALLOC);
 258                                 mh_mode &= ~CRYPTO_MEM_CHECK_ENABLE;
```

```
259                                 CRYPTO_THREADID_cpy(&disabling_threadid, &cur);
260                                 }
261                         num_disable++;
262                         }
263                 break;
264         case CRYPTO_MEM_CHECK_ENABLE: /* aka MemCheck_on() */
265                 if (mh_mode & CRYPTO_MEM_CHECK_ON)
266                         {
267                         if (num_disable) /* always true, or something is going w
268                                 {
269                                 num_disable--;
270                                 if (num_disable == 0)
271                                         {
272                                         mh_mode|=CRYPTO_MEM_CHECK_ENABLE;
273                                         CRYPTO_w_unlock(CRYPTO_LOCK_MALLOC2);
274                                         }
275                                 }
276                         }
277                 break;

279         default:
280                 break;
281                 }
282         CRYPTO_w_unlock(CRYPTO_LOCK_MALLOC);
283         return(ret);
284         }

286 int CRYPTO_is_mem_check_on(void)
287         {
288         int ret = 0;

290         if (mh_mode & CRYPTO_MEM_CHECK_ON)
291                 {
292                 CRYPTO_THREADID cur;
293                 CRYPTO_THREADID_current(&cur);
294                 CRYPTO_r_lock(CRYPTO_LOCK_MALLOC);

296                 ret = (mh_mode & CRYPTO_MEM_CHECK_ENABLE)
297                         || CRYPTO_THREADID_cmp(&disabling_threadid, &cur);

299                 CRYPTO_r_unlock(CRYPTO_LOCK_MALLOC);
300                 }
301         return(ret);
302         }


305 void CRYPTO_dbg_set_options(long bits)
306         {
307         options = bits;
308         }

310 long CRYPTO_dbg_get_options(void)
311         {
312         return options;
313         }

315 static int mem_cmp(const MEM *a, const MEM *b)
316         {
317 #ifdef _WIN64
318         const char *ap=(const char *)a->addr,
319                    *bp=(const char *)b->addr;
320         if (ap==bp)     return 0;
321         else if (ap>bp) return 1;
322         else            return -1;
323 #else
324         return (const char *)a->addr - (const char *)b->addr;
```



```
325 #endif
326         }
327 static IMPLEMENT_LHASH_COMP_FN(mem, MEM)

329 static unsigned long mem_hash(const MEM *a)
330         {
331         unsigned long ret;

333         ret=(unsigned long)a->addr;

335         ret=ret*17851+(ret>>14)*7+(ret>>4)*251;
336         return(ret);
337         }
338 static IMPLEMENT_LHASH_HASH_FN(mem, MEM)

340 /* static int app_info_cmp(APP_INFO *a, APP_INFO *b) */
341 static int app_info_cmp(const void *a_void, const void *b_void)
342         {
343         return CRYPTO_THREADID_cmp(&((const APP_INFO *)a_void)->threadid,
344                                    &((const APP_INFO *)b_void)->threadid);
345         }
346 static IMPLEMENT_LHASH_COMP_FN(app_info, APP_INFO)

348 static unsigned long app_info_hash(const APP_INFO *a)
349         {
350         unsigned long ret;

352         ret = CRYPTO_THREADID_hash(&a->threadid);
353         /* This is left in as a "who am I to question legacy?" measure */
354         ret=ret*17851+(ret>>14)*7+(ret>>4)*251;
355         return(ret);
356         }
357 static IMPLEMENT_LHASH_HASH_FN(app_info, APP_INFO)

359 static APP_INFO *pop_info(void)
360         {
361         APP_INFO tmp;
362         APP_INFO *ret = NULL;

364         if (amih != NULL)
365                 {
366                 CRYPTO_THREADID_current(&tmp.threadid);
367                 if ((ret=lh_APP_INFO_delete(amih,&tmp)) != NULL)
368                         {
369                         APP_INFO *next=ret->next;

371                         if (next != NULL)
372                                 {
373                                 next->references++;
374                                 (void)lh_APP_INFO_insert(amih,next);
375                                 }
376 #ifdef LEVITTE_DEBUG_MEM
377                         if (CRYPTO_THREADID_cmp(&ret->threadid, &tmp.threadid))
378                                 {
379                                 fprintf(stderr, "pop_info(): deleted info has ot
380                                         CRYPTO_THREADID_hash(&ret->threadid),
381                                         CRYPTO_THREADID_hash(&tmp.threadid));
382                                 abort();
383                                 }
384 #endif
385                         if (--(ret->references) <= 0)
386                                 {
387                                 ret->next = NULL;
388                                 if (next != NULL)
389                                         next->references--;
390                                 OPENSSL_free(ret);
```

```
391                                 }
392                         }
393                 }
394         return(ret);
395         }

397 int CRYPTO_push_info_(const char *info, const char *file, int line)
398         {
399         APP_INFO *ami, *amim;
400         int ret=0;

402         if (is_MemCheck_on())
403                 {
404                 MemCheck_off(); /* obtain MALLOC2 lock */

406                 if ((ami = (APP_INFO *)OPENSSL_malloc(sizeof(APP_INFO))) == NULL
407                         {
408                         ret=0;
409                         goto err;
410                         }
411                 if (amih == NULL)
412                         {
413                         if ((amih=lh_APP_INFO_new()) == NULL)
414                                 {
415                                 OPENSSL_free(ami);
416                                 ret=0;
417                                 goto err;
418                                 }
419                         }

421                 CRYPTO_THREADID_current(&ami->threadid);
422                 ami->file=file;
423                 ami->line=line;
424                 ami->info=info;
425                 ami->references=1;
426                 ami->next=NULL;

428                 if ((amim=lh_APP_INFO_insert(amih,ami)) != NULL)
429                         {
430 #ifdef LEVITTE_DEBUG_MEM
431                         if (CRYPTO_THREADID_cmp(&ami->threadid, &amim->threadid)
432                                 {
433                                 fprintf(stderr, "CRYPTO_push_info(): previous in
434                                         CRYPTO_THREADID_hash(&amim->threadid),
435                                         CRYPTO_THREADID_hash(&ami->threadid));
436                                 abort();
437                                 }
438 #endif
439                         ami->next=amim;
440                         }
441  err:
442                 MemCheck_on(); /* release MALLOC2 lock */
443                 }

445         return(ret);
446         }

448 int CRYPTO_pop_info(void)
449         {
450         int ret=0;

452         if (is_MemCheck_on()) /* _must_ be true, or something went severely wron
453                 {
454                 MemCheck_off(); /* obtain MALLOC2 lock */

456                 ret=(pop_info() != NULL);
```



```
458                 MemCheck_on(); /* release MALLOC2 lock */
459                 }
460         return(ret);
461         }

463 int CRYPTO_remove_all_info(void)
464         {
465         int ret=0;

467         if (is_MemCheck_on()) /* _must_ be true */
468                 {
469                 MemCheck_off(); /* obtain MALLOC2 lock */

471                 while(pop_info() != NULL)
472                         ret++;

474                 MemCheck_on(); /* release MALLOC2 lock */
475                 }
476         return(ret);
477         }


480 static unsigned long break_order_num=0;
481 void CRYPTO_dbg_malloc(void *addr, int num, const char *file, int line,
482         int before_p)
483         {
484         MEM *m,*mm;
485         APP_INFO tmp,*amim;

487         switch(before_p & 127)
488                 {
489         case 0:
490                 break;
491         case 1:
492                 if (addr == NULL)
493                         break;

495                 if (is_MemCheck_on())
496                         {
497                         MemCheck_off(); /* make sure we hold MALLOC2 lock */
498                         if ((m=(MEM *)OPENSSL_malloc(sizeof(MEM))) == NULL)
499                                 {
500                                 OPENSSL_free(addr);
501                                 MemCheck_on(); /* release MALLOC2 lock
502                                                 * if num_disabled drops to 0 */
503                                 return;
504                                 }
505                         if (mh == NULL)
506                                 {
507                                 if ((mh=lh_MEM_new()) == NULL)
508                                         {
509                                         OPENSSL_free(addr);
510                                         OPENSSL_free(m);
511                                         addr=NULL;
512                                         goto err;
513                                         }
514                                 }

516                         m->addr=addr;
517                         m->file=file;
518                         m->line=line;
519                         m->num=num;
520                         if (options & V_CRYPTO_MDEBUG_THREAD)
521                                 CRYPTO_THREADID_current(&m->threadid);
522                         else
```

```
523                                 memset(&m->threadid, 0, sizeof(m->threadid));

525                         if (order == break_order_num)
526                                 {
527                                 /* BREAK HERE */
528                                 m->order=order;
529                                 }
530                         m->order=order++;
531 #ifdef LEVITTE_DEBUG_MEM
532                         fprintf(stderr, "LEVITTE_DEBUG_MEM: [%5ld] %c 0x%p (%d)\
533                                 m->order,
534                                 (before_p & 128) ? '*' : '+',
535                                 m->addr, m->num);
536 #endif
537                         if (options & V_CRYPTO_MDEBUG_TIME)
538                                 m->time=time(NULL);
539                         else
540                                 m->time=0;

542                         CRYPTO_THREADID_current(&tmp.threadid);
543                         m->app_info=NULL;
544                         if (amih != NULL
545                             && (amim=lh_APP_INFO_retrieve(amih,&tmp)) != NULL)
546                                 {
547                                 m->app_info = amim;
548                                 amim->references++;
549                                 }

551                         if ((mm=lh_MEM_insert(mh, m)) != NULL)
552                                 {
553                                 /* Not good, but don't sweat it */
554                                 if (mm->app_info != NULL)
555                                         {
556                                         mm->app_info->references--;
557                                         }
558                                 OPENSSL_free(mm);
559                                 }
560                 err:
561                         MemCheck_on(); /* release MALLOC2 lock
562                                         * if num_disabled drops to 0 */
563                         }
564                 break;
565                 }
566         return;
567         }

569 void CRYPTO_dbg_free(void *addr, int before_p)
570         {
571         MEM m,*mp;

573         switch(before_p)
574                 {
575         case 0:
576                 if (addr == NULL)
577                         break;

579                 if (is_MemCheck_on() && (mh != NULL))
580                         {
581                         MemCheck_off(); /* make sure we hold MALLOC2 lock */

583                         m.addr=addr;
584                         mp=lh_MEM_delete(mh,&m);
585                         if (mp != NULL)
586                                 {
587 #ifdef LEVITTE_DEBUG_MEM
588                         fprintf(stderr, "LEVITTE_DEBUG_MEM: [%5ld] - 0x%p (%d)\n
```



```
589                                 mp->order, mp->addr, mp->num);
590 #endif
591                                 if (mp->app_info != NULL)
592                                         app_info_free(mp->app_info);
593                                 OPENSSL_free(mp);
594                                 }

596                         MemCheck_on(); /* release MALLOC2 lock
597                                         * if num_disabled drops to 0 */
598                         }
599                 break;
600         case 1:
601                 break;
602                 }
603         }

605 void CRYPTO_dbg_realloc(void *addr1, void *addr2, int num,
606         const char *file, int line, int before_p)
607         {
608         MEM m,*mp;

610 #ifdef LEVITTE_DEBUG_MEM
611         fprintf(stderr, "LEVITTE_DEBUG_MEM: --> CRYPTO_dbg_malloc(addr1 = %p, ad
612                 addr1, addr2, num, file, line, before_p);
613 #endif

615         switch(before_p)
616                 {
617         case 0:
618                 break;
619         case 1:
620                 if (addr2 == NULL)
621                         break;

623                 if (addr1 == NULL)
624                         {
625                         CRYPTO_dbg_malloc(addr2, num, file, line, 128 | before_p
626                         break;
627                         }

629                 if (is_MemCheck_on())
630                         {
631                         MemCheck_off(); /* make sure we hold MALLOC2 lock */

633                         m.addr=addr1;
634                         mp=lh_MEM_delete(mh,&m);
635                         if (mp != NULL)
636                                 {
637 #ifdef LEVITTE_DEBUG_MEM
638                                 fprintf(stderr, "LEVITTE_DEBUG_MEM: [%5ld] * 0x%
639                                         mp->order,
640                                         mp->addr, mp->num,
641                                         addr2, num);
642 #endif
643                                 mp->addr=addr2;
644                                 mp->num=num;
645                                 (void)lh_MEM_insert(mh,mp);
646                                 }

648                         MemCheck_on(); /* release MALLOC2 lock
649                                         * if num_disabled drops to 0 */
650                         }
651                 break;
652                 }
653         return;
654         }
```

```
typedef struct mem_leak_st
        {
        BIO *bio;
        int chunks;
        long bytes;
        } MEM_LEAK;

static void print_leak_doall_arg(const MEM *m, MEM_LEAK *l)
        {
        char buf[1024];
        char *bufp = buf;
        APP_INFO *amip;
        int ami_cnt;
        struct tm *lcl = NULL;
        CRYPTO_THREADID ti;

#define BUF_REMAIN (sizeof buf - (size_t)(bufp - buf))

        if(m->addr == (char *)l->bio)
            return;

        if (options & V_CRYPTO_MDEBUG_TIME)
                {
                lcl = localtime(&m->time);

                BIO_snprintf(bufp, BUF_REMAIN, "[%02d:%02d:%02d] ",
                        lcl->tm_hour,lcl->tm_min,lcl->tm_sec);
                bufp += strlen(bufp);
                }

        BIO_snprintf(bufp, BUF_REMAIN, "%5lu file=%s, line=%d, ",
                m->order,m->file,m->line);
        bufp += strlen(bufp);

        if (options & V_CRYPTO_MDEBUG_THREAD)
                {
                BIO_snprintf(bufp, BUF_REMAIN, "thread=%lu, ",
                        CRYPTO_THREADID_hash(&m->threadid));
                bufp += strlen(bufp);
                }

        BIO_snprintf(bufp, BUF_REMAIN, "number=%d, address=%08lX\n",
                m->num,(unsigned long)m->addr);
        bufp += strlen(bufp);

        BIO_puts(l->bio,buf);

        l->chunks++;
        l->bytes+=m->num;

        amip=m->app_info;
        ami_cnt=0;
        if (!amip)
                return;
        CRYPTO_THREADID_cpy(&ti, &amip->threadid);

        do
                {
                int buf_len;
                int info_len;

                ami_cnt++;
                memset(buf,'>',ami_cnt);
                BIO_snprintf(buf + ami_cnt, sizeof buf - ami_cnt,
```



```
                        " thread=%lu, file=%s, line=%d, info=\"",
                        CRYPTO_THREADID_hash(&amip->threadid), amip->file,
                        amip->line);
                buf_len=strlen(buf);
                info_len=strlen(amip->info);
                if (128 - buf_len - 3 < info_len)
                        {
                        memcpy(buf + buf_len, amip->info, 128 - buf_len - 3);
                        buf_len = 128 - 3;
                        }
                else
                        {
                        BUF_strlcpy(buf + buf_len, amip->info,
                                    sizeof buf - buf_len);
                        buf_len = strlen(buf);
                        }
                BIO_snprintf(buf + buf_len, sizeof buf - buf_len, "\"\n");

                BIO_puts(l->bio,buf);

                amip = amip->next;
                }
        while(amip && !CRYPTO_THREADID_cmp(&amip->threadid, &ti));

#ifdef LEVITTE_DEBUG_MEM
        if (amip)
                {
                fprintf(stderr, "Thread switch detected in backtrace!!!!\n");
                abort();
                }
#endif
        }

static IMPLEMENT_LHASH_DOALL_ARG_FN(print_leak, const MEM, MEM_LEAK)

void CRYPTO_mem_leaks(BIO *b)
        {
        MEM_LEAK ml;

        if (mh == NULL && amih == NULL)
                return;

        MemCheck_off(); /* obtain MALLOC2 lock */

        ml.bio=b;
        ml.bytes=0;
        ml.chunks=0;
        if (mh != NULL)
                lh_MEM_doall_arg(mh, LHASH_DOALL_ARG_FN(print_leak), MEM_LEAK,
                                 &ml);
        if (ml.chunks != 0)
                {
                BIO_printf(b,"%ld bytes leaked in %d chunks\n",
                           ml.bytes,ml.chunks);
#ifdef CRYPTO_MDEBUG_ABORT
                abort();
#endif
                }
        else
                {
                /* Make sure that, if we found no leaks, memory-leak debugging i
                 * does not introduce memory leaks (which might irritate
                 * external debugging tools).
                 * (When someone enables leak checking, but does not call
                 * this function, we declare it to be their fault.)
                 *
```

```
787                  * XXX    This should be in CRYPTO_mem_leaks_cb,
788                  * and CRYPTO_mem_leaks should be implemented by
789                  * using CRYPTO_mem_leaks_cb.
790                  * (Also there should be a variant of lh_doall_arg
791                  * that takes a function pointer instead of a void *;
792                  * this would obviate the ugly and illegal
793                  * void_fn_to_char kludge in CRYPTO_mem_leaks_cb.
794                  * Otherwise the code police will come and get us.)
795                  */
796                 int old_mh_mode;

798                 CRYPTO_w_lock(CRYPTO_LOCK_MALLOC);

800                 /* avoid deadlock when lh_free() uses CRYPTO_dbg_free(),
801                  * which uses CRYPTO_is_mem_check_on */
802                 old_mh_mode = mh_mode;
803                 mh_mode = CRYPTO_MEM_CHECK_OFF;

805                 if (mh != NULL)
806                         {
807                         lh_MEM_free(mh);
808                         mh = NULL;
809                         }
810                 if (amih != NULL)
811                         {
812                         if (lh_APP_INFO_num_items(amih) == 0)
813                                 {
814                                 lh_APP_INFO_free(amih);
815                                 amih = NULL;
816                                 }
817                         }

819                 mh_mode = old_mh_mode;
820                 CRYPTO_w_unlock(CRYPTO_LOCK_MALLOC);
821                 }
822         MemCheck_on(); /* release MALLOC2 lock */
823         }

825 #ifndef OPENSSL_NO_FP_API
826 void CRYPTO_mem_leaks_fp(FILE *fp)
827         {
828         BIO *b;

830         if (mh == NULL) return;
831         /* Need to turn off memory checking when allocated BIOs ... especially
832          * as we're creating them at a time when we're trying to check we've not
833          * left anything un-free()'d!! */
834         MemCheck_off();
835         b = BIO_new(BIO_s_file());
836         MemCheck_on();
837         if(!b) return;
838         BIO_set_fp(b,fp,BIO_NOCLOSE);
839         CRYPTO_mem_leaks(b);
840         BIO_free(b);
841         }
842 #endif



846 /* FIXME: We really don't allow much to the callback.  For example, it has
847    no chance of reaching the info stack for the item it processes.  Should
848    it really be this way?  -- Richard Levitte */
849 /* NB: The prototypes have been typedef'd to CRYPTO_MEM_LEAK_CB inside crypto.h
850  * If this code is restructured, remove the callback type if it is no longer
851  * needed. -- Geoff Thorpe */
```



```
853 /* Can't pass CRYPTO_MEM_LEAK_CB directly to lh_MEM_doall_arg because it
854  * is a function pointer and conversion to void * is prohibited. Instead
855  * pass its address
856  */

858 typedef CRYPTO_MEM_LEAK_CB *PCRYPTO_MEM_LEAK_CB;

860 static void cb_leak_doall_arg(const MEM *m, PCRYPTO_MEM_LEAK_CB *cb)
861         {
862         (*cb)(m->order,m->file,m->line,m->num,m->addr);
863         }

865 static IMPLEMENT_LHASH_DOALL_ARG_FN(cb_leak, const MEM, PCRYPTO_MEM_LEAK_CB)

867 void CRYPTO_mem_leaks_cb(CRYPTO_MEM_LEAK_CB *cb)
868         {
869         if (mh == NULL) return;
870         CRYPTO_w_lock(CRYPTO_LOCK_MALLOC2);
871         lh_MEM_doall_arg(mh, LHASH_DOALL_ARG_FN(cb_leak), PCRYPTO_MEM_LEAK_CB,
872                          &cb);
873         CRYPTO_w_unlock(CRYPTO_LOCK_MALLOC2);
874         }
```

```
**********************************************************
    3914 Wed Aug 13 19:33:00 2014
new/usr/src/lib/openssl/libsunw_crypto/o_str.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/o_str.c -*- mode:C; c-file-style: "eay" -*- */
   2 /* Written by Richard Levitte (richard@levitte.org) for the OpenSSL
   3  * project 2003.
   4  */
   5 /* ====================================================================
   6  * Copyright (c) 2003 The OpenSSL Project.  All rights reserved.
   7  *
   8  * Redistribution and use in source and binary forms, with or without
   9  * modification, are permitted provided that the following conditions
  10  * are met:
  11  *
  12  * 1. Redistributions of source code must retain the above copyright
  13  *    notice, this list of conditions and the following disclaimer.
  14  *
  15  * 2. Redistributions in binary form must reproduce the above copyright
  16  *    notice, this list of conditions and the following disclaimer in
  17  *    the documentation and/or other materials provided with the
  18  *    distribution.
  19  *
  20  * 3. All advertising materials mentioning features or use of this
  21  *    software must display the following acknowledgment:
  22  *    "This product includes software developed by the OpenSSL Project
  23  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
  24  *
  25  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  26  *    endorse or promote products derived from this software without
  27  *    prior written permission. For written permission, please contact
  28  *    openssl-core@openssl.org.
  29  *
  30  * 5. Products derived from this software may not be called "OpenSSL"
  31  *    nor may "OpenSSL" appear in their names without prior written
  32  *    permission of the OpenSSL Project.
  33  *
  34  * 6. Redistributions of any form whatsoever must retain the following
  35  *    acknowledgment:
  36  *    "This product includes software developed by the OpenSSL Project
  37  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
  38  *
  39  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
  40  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  41  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  42  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  43  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  44  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  45  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  46  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  47  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  48  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  49  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  50  * OF THE POSSIBILITY OF SUCH DAMAGE.
  51  * ====================================================================
  52  *
  53  * This product includes cryptographic software written by Eric Young
  54  * (eay@cryptsoft.com).  This product includes software written by Tim
  55  * Hudson (tjh@cryptsoft.com).
  56  *
  57  */

  59 #include <ctype.h>
  60 #include <e_os.h>
  61 #include <o_str.h>
```



```
  61 #include "o_str.h"

  63 #if !defined(OPENSSL_IMPLEMENTS_strncasecmp) && \
  64     !defined(OPENSSL_SYSNAME_WIN32) && \
  65     !defined(NETWARE_CLIB)
  66 # include <strings.h>
  67 #endif

  69 int OPENSSL_strncasecmp(const char *str1, const char *str2, size_t n)
  70         {
  71 #if defined(OPENSSL_IMPLEMENTS_strncasecmp)
  72         while (*str1 && *str2 && n)
  73                 {
  74                 int res = toupper(*str1) - toupper(*str2);
  75                 if (res) return res < 0 ? -1 : 1;
  76                 str1++;
  77                 str2++;
  78                 n--;
  79                 }
  80         if (n == 0)
  81                 return 0;
  82         if (*str1)
  83                 return 1;
  84         if (*str2)
  85                 return -1;
  86         return 0;
  87 #else
  88         /* Recursion hazard warning! Whenever strncasecmp is #defined as
  89          * OPENSSL_strncasecmp, OPENSSL_IMPLEMENTS_strncasecmp must be
  90          * defined as well. */
  91         return strncasecmp(str1, str2, n);
  92 #endif
  93         }
  94 int OPENSSL_strcasecmp(const char *str1, const char *str2)
  95         {
  96 #if defined(OPENSSL_IMPLEMENTS_strncasecmp)
  97         return OPENSSL_strncasecmp(str1, str2, (size_t)-1);
  98 #else
  99         return strcasecmp(str1, str2);
 100 #endif
 101         }

 103 int OPENSSL_memcmp(const void *v1,const void *v2,size_t n)
 104         {
 105         const unsigned char *c1=v1,*c2=v2;
 106         int ret=0;

 108         while(n && (ret=*c1-*c2)==0) n--,c1++,c2++;

 110         return ret;
 111         }
```

```
**********************************************************
   11238 Wed Aug 13 19:33:00 2014
new/usr/src/lib/openssl/libsunw_crypto/o_time.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/o_time.c -*- mode:C; c-file-style: "eay" -*- */
   2 /* Written by Richard Levitte (richard@levitte.org) for the OpenSSL
   3  * project 2001.
   4  */
   5 /* Written by Dr Stephen N Henson (steve@openssl.org) for the OpenSSL
   6  * project 2008.
   7  */
   8 /* ====================================================================
   9  * Copyright (c) 2001 The OpenSSL Project.  All rights reserved.
  10  *
  11  * Redistribution and use in source and binary forms, with or without
  12  * modification, are permitted provided that the following conditions
  13  * are met:
  14  *
  15  * 1. Redistributions of source code must retain the above copyright
  16  *    notice, this list of conditions and the following disclaimer.
  17  *
  18  * 2. Redistributions in binary form must reproduce the above copyright
  19  *    notice, this list of conditions and the following disclaimer in
  20  *    the documentation and/or other materials provided with the
  21  *    distribution.
  22  *
  23  * 3. All advertising materials mentioning features or use of this
  24  *    software must display the following acknowledgment:
  25  *    "This product includes software developed by the OpenSSL Project
  26  *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
  27  *
  28  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  29  *    endorse or promote products derived from this software without
  30  *    prior written permission. For written permission, please contact
  31  *    licensing@OpenSSL.org.
  32  *
  33  * 5. Products derived from this software may not be called "OpenSSL"
  34  *    nor may "OpenSSL" appear in their names without prior written
  35  *    permission of the OpenSSL Project.
  36  *
  37  * 6. Redistributions of any form whatsoever must retain the following
  38  *    acknowledgment:
  39  *    "This product includes software developed by the OpenSSL Project
  40  *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
  41  *
  42  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
  43  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  44  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  45  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  46  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  47  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  48  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  49  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  50  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  51  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  52  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  53  * OF THE POSSIBILITY OF SUCH DAMAGE.
  54  * ====================================================================
  55  *
  56  * This product includes cryptographic software written by Eric Young
  57  * (eay@cryptsoft.com).  This product includes software written by Tim
  58  * Hudson (tjh@cryptsoft.com).
  59  *
  60  */
```



```
  62 #include <openssl/e_os2.h>
  63 #include <string.h>
  64 #include <o_time.h>
  64 #include "o_time.h"

  66 #ifdef OPENSSL_SYS_VMS
  67 # if __CRTL_VER >= 70000000 && \
  68      (defined _POSIX_C_SOURCE || !defined _ANSI_C_SOURCE)
  69 #  define VMS_GMTIME_OK
  70 # endif
  71 # ifndef VMS_GMTIME_OK
  72 #  include <libdtdef.h>
  73 #  include <lib$routines.h>
  74 #  include <lnmdef.h>
  75 #  include <starlet.h>
  76 #  include <descrip.h>
  77 #  include <stdlib.h>
  78 # endif /* ndef VMS_GMTIME_OK */
  79 #endif

  81 struct tm *OPENSSL_gmtime(const time_t *timer, struct tm *result)
  82         {
  83         struct tm *ts = NULL;

  85 #if defined(OPENSSL_THREADS) && !defined(OPENSSL_SYS_WIN32) && !defined(OPENSSL_
  86         /* should return &data, but doesn't on some systems,
  87            so we don't even look at the return value */
  88         gmtime_r(timer,result);
  89         ts = result;
  90 #elif !defined(OPENSSL_SYS_VMS) || defined(VMS_GMTIME_OK)
  91         ts = gmtime(timer);
  92         if (ts == NULL)
  93                 return NULL;

  95         memcpy(result, ts, sizeof(struct tm));
  96         ts = result;
  97 #endif
  98 #if defined( OPENSSL_SYS_VMS) && !defined( VMS_GMTIME_OK)
  99         if (ts == NULL)
 100                 {
 101                 static $DESCRIPTOR(tabnam,"LNM$DCL_LOGICAL");
 102                 static $DESCRIPTOR(lognam,"SYS$TIMEZONE_DIFFERENTIAL");
 103                 char logvalue[256];
 104                 unsigned int reslen = 0;
 105                 struct {
 106                         short buflen;
 107                         short code;
 108                         void *bufaddr;
 109                         unsigned int *reslen;
 110                 } itemlist[] = {
 111                         { 0, LNM$_STRING, 0, 0 },
 112                         { 0, 0, 0, 0 },
 113                 };
 114                 int status;
 115                 time_t t;

 117                 /* Get the value for SYS$TIMEZONE_DIFFERENTIAL */
 118                 itemlist[0].buflen = sizeof(logvalue);
 119                 itemlist[0].bufaddr = logvalue;
 120                 itemlist[0].reslen = &reslen;
 121                 status = sys$trnlnm(0, &tabnam, &lognam, 0, itemlist);
 122                 if (!(status & 1))
 123                         return NULL;
 124                 logvalue[reslen] = '\0';

 126                 t = *timer;
```

```
/* The following is extracted from the DEC C header time.h */
/*
**  Beginning in OpenVMS Version 7.0 mktime, time, ctime, strftime
**  have two implementations.  One implementation is provided
**  for compatibility and deals with time in terms of local time,
**  the other __utc_* deals with time in terms of UTC.
*/
/* We use the same conditions as in said time.h to check if we should
   assume that t contains local time (and should therefore be adjusted)
   or UTC (and should therefore be left untouched). */
#if __CRTL_VER < 70000000 || defined _VMS_V6_SOURCE
                /* Get the numerical value of the equivalence string */
                status = atoi(logvalue);

                /* and use it to move time to GMT */
                t -= status;
#endif

                /* then convert the result to the time structure */

                /* Since there was no gmtime_r() to do this stuff for us,
                   we have to do it the hard way. */
                {
                /* The VMS epoch is the astronomical Smithsonian date,
                   if I remember correctly, which is November 17, 1858.
                   Furthermore, time is measure in thenths of microseconds
                   and stored in quadwords (64 bit integers).  unix_epoch
                   below is January 1st 1970 expressed as a VMS time.  The
                   following code was used to get this number:

                   #include <stdio.h>
                   #include <stdlib.h>
                   #include <lib$routines.h>
                   #include <starlet.h>

                   main()
                   {
                     unsigned long systime[2];
                     unsigned short epoch_values[7] =
                       { 1970, 1, 1, 0, 0, 0, 0 };

                     lib$cvt_vectim(epoch_values, systime);

                     printf("%u %u", systime[0], systime[1]);
                   }
                */
                unsigned long unix_epoch[2] = { 1273708544, 8164711 };
                unsigned long deltatime[2];
                unsigned long systime[2];
                struct vms_vectime
                       {
                       short year, month, day, hour, minute, second,
                             centi_second;
                       } time_values;
                long operation;

                /* Turn the number of seconds since January 1st 1970 to
                   an internal delta time.
                   Note that lib$cvt_to_internal_time() will assume
                   that t is signed, and will therefore break on 32-bit
                   systems some time in 2038.
                */
                operation = LIB$K_DELTA_SECONDS;
                status = lib$cvt_to_internal_time(&operation,
                       &t, deltatime);
```



```
                /* Add the delta time with the Unix epoch and we have
                   the current UTC time in internal format */
                status = lib$add_times(unix_epoch, deltatime, systime);

                /* Turn the internal time into a time vector */
                status = sys$numtim(&time_values, systime);

                /* Fill in the struct tm with the result */
                result->tm_sec = time_values.second;
                result->tm_min = time_values.minute;
                result->tm_hour = time_values.hour;
                result->tm_mday = time_values.day;
                result->tm_mon = time_values.month - 1;
                result->tm_year = time_values.year - 1900;

                operation = LIB$K_DAY_OF_WEEK;
                status = lib$cvt_from_internal_time(&operation,
                       &result->tm_wday, systime);
                result->tm_wday %= 7;

                operation = LIB$K_DAY_OF_YEAR;
                status = lib$cvt_from_internal_time(&operation,
                       &result->tm_yday, systime);
                result->tm_yday--;

                result->tm_isdst = 0; /* There's no way to know... */

                ts = result;
                }
                }
#endif
        return ts;
        }

/* Take a tm structure and add an offset to it. This avoids any OS issues
 * with restricted date types and overflows which cause the year 2038
 * problem.
 */

#define SECS_PER_DAY (24 * 60 * 60)

static long date_to_julian(int y, int m, int d);
static void julian_to_date(long jd, int *y, int *m, int *d);

int OPENSSL_gmtime_adj(struct tm *tm, int off_day, long offset_sec)
        {
        int offset_hms, offset_day;
        long time_jd;
        int time_year, time_month, time_day;
        /* split offset into days and day seconds */
        offset_day = offset_sec / SECS_PER_DAY;
        /* Avoid sign issues with % operator */
        offset_hms  = offset_sec - (offset_day * SECS_PER_DAY);
        offset_day += off_day;
        /* Add current time seconds to offset */
        offset_hms += tm->tm_hour * 3600 + tm->tm_min * 60 + tm->tm_sec;
        /* Adjust day seconds if overflow */
        if (offset_hms >= SECS_PER_DAY)
                {
                offset_day++;
                offset_hms -= SECS_PER_DAY;
                }
        else if (offset_hms < 0)
                {
                offset_day--;
```

```
259                 offset_hms += SECS_PER_DAY;
260                 }

262         /* Convert date of time structure into a Julian day number.
263          */

265         time_year = tm->tm_year + 1900;
266         time_month = tm->tm_mon + 1;
267         time_day = tm->tm_mday;

269         time_jd = date_to_julian(time_year, time_month, time_day);

271         /* Work out Julian day of new date */
272         time_jd += offset_day;

274         if (time_jd < 0)
275                 return 0;

277         /* Convert Julian day back to date */

279         julian_to_date(time_jd, &time_year, &time_month, &time_day);

281         if (time_year < 1900 || time_year > 9999)
282                 return 0;

284         /* Update tm structure */

286         tm->tm_year = time_year - 1900;
287         tm->tm_mon = time_month - 1;
288         tm->tm_mday = time_day;

290         tm->tm_hour = offset_hms / 3600;
291         tm->tm_min = (offset_hms / 60) % 60;
292         tm->tm_sec = offset_hms % 60;

294         return 1;

296 }
```

unchanged_portion_omitted_

```
**********************************************************
    6961 Wed Aug 13 19:33:01 2014
new/usr/src/lib/openssl/libsunw_crypto/pl/keysets.pl
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
1 #!/usr/bin/perl
1 #!/usr/local/bin/perl

3 $NUMBER=0x01;
4 $UPPER=0x02;
5 $LOWER=0x04;
6 $UNDER=0x100;
7 $PUNCTUATION=0x200;
8 $WS=0x10;
9 $ESC=0x20;
10 $QUOTE=0x40;
11 $DQUOTE=0x400;
12 $COMMENT=0x80;
13 $FCOMMENT=0x800;
14 $EOF=0x08;
15 $HIGHBIT=0x1000;

17 foreach (0 .. 255)
18         {
19         $v=0;
20         $c=sprintf("%c",$_);
21         $v|=$NUMBER     if ($c =~ /[0-9]/);
22         $v|=$UPPER      if ($c =~ /[A-Z]/);
23         $v|=$LOWER      if ($c =~ /[a-z]/);
24         $v|=$UNDER      if ($c =~ /_/);
25         $v|=$PUNCTUATION if ($c =~ /[!\.%&\*\+,\/;\?\@\^\~\|-]/);
26         $v|=$WS         if ($c =~ /[ \t\r\n]/);
27         $v|=$ESC        if ($c =~ /\\/);
28         $v|=$QUOTE      if ($c =~ /['']/); # for emacs: "''}/)
29         $v|=$COMMENT    if ($c =~ /\#/);
30         $v|=$EOF        if ($c =~ /\0/);
31         $v|=$HIGHBIT    if ($c =~/[\x80-\xff]/);

33         push(@V_def,$v);
34         }

36 foreach (0 .. 255)
37         {
38         $v=0;
39         $c=sprintf("%c",$_);
40         $v|=$NUMBER     if ($c =~ /[0-9]/);
41         $v|=$UPPER      if ($c =~ /[A-Z]/);
42         $v|=$LOWER      if ($c =~ /[a-z]/);
43         $v|=$UNDER      if ($c =~ /_/);
44         $v|=$PUNCTUATION if ($c =~ /[!\.%&\*\+,\/;\?\@\^\~\|-]/);
45         $v|=$WS         if ($c =~ /[ \t\r\n]/);
46         $v|=$DQUOTE     if ($c =~ /["]/); # for emacs: "}/)
47         $v|=$FCOMMENT   if ($c =~ /;/);
48         $v|=$EOF        if ($c =~ /\0/);
49         $v|=$HIGHBIT    if ($c =~/[\x80-\xff]/);

51         push(@V_w32,$v);
52         }

54 print <<"EOF";
55 /* crypto/conf/conf_def.h */
56 /* Copyright (C) 1995-1998 Eric Young (eay\@cryptsoft.com)
57  * All rights reserved.
58  *
59  * This package is an SSL implementation written
60  * by Eric Young (eay\@cryptsoft.com).
```



```
61  * The implementation was written so as to conform with Netscapes SSL.
62  *
63  * This library is free for commercial and non-commercial use as long as
64  * the following conditions are aheared to.  The following conditions
65  * apply to all code found in this distribution, be it the RC4, RSA,
66  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
67  * included with this distribution is covered by the same copyright terms
68  * except that the holder is Tim Hudson (tjh\@cryptsoft.com).
69  *
70  * Copyright remains Eric Young's, and as such any Copyright notices in
71  * the code are not to be removed.
72  * If this package is used in a product, Eric Young should be given attribution
73  * as the author of the parts of the library used.
74  * This can be in the form of a textual message at program startup or
75  * in documentation (online or textual) provided with the package.
76  *
77  * Redistribution and use in source and binary forms, with or without
78  * modification, are permitted provided that the following conditions
79  * are met:
80  * 1. Redistributions of source code must retain the copyright
81  *    notice, this list of conditions and the following disclaimer.
82  * 2. Redistributions in binary form must reproduce the above copyright
83  *    notice, this list of conditions and the following disclaimer in the
84  *    documentation and/or other materials provided with the distribution.
85  * 3. All advertising materials mentioning features or use of this software
86  *    must display the following acknowledgement:
87  *    "This product includes cryptographic software written by
88  *     Eric Young (eay\@cryptsoft.com)"
89  *    The word 'cryptographic' can be left out if the rouines from the library
90  *    being used are not cryptographic related :-).
91  * 4. If you include any Windows specific code (or a derivative thereof) from
92  *    the apps directory (application code) you must include an acknowledgement:
93  *    "This product includes software written by Tim Hudson (tjh\@cryptsoft.com)
94  *
95  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
96  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
97  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
98  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
99  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
100 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
101 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
102 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
103 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
104 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
105 * SUCH DAMAGE.
106 *
107 * The licence and distribution terms for any publically available version or
108 * derivative of this code cannot be changed.  i.e. this code cannot simply be
109 * copied and put under another distribution licence
110 * [including the GNU Public Licence.]
111 */

113 /* THIS FILE WAS AUTOMAGICALLY GENERATED!
114    Please modify and use keysets.pl to regenerate it. */

116 #define CONF_NUMBER             $NUMBER
117 #define CONF_UPPER              $UPPER
118 #define CONF_LOWER              $LOWER
119 #define CONF_UNDER              $UNDER
120 #define CONF_PUNCTUATION        $PUNCTUATION
121 #define CONF_WS                 $WS
122 #define CONF_ESC                $ESC
123 #define CONF_QUOTE              $QUOTE
124 #define CONF_DQUOTE             $DQUOTE
125 #define CONF_COMMENT            $COMMENT
126 #define CONF_FCOMMENT           $FCOMMENT
```

```
 127 #define CONF_EOF                $EOF
 128 #define CONF_HIGHBIT            $HIGHBIT
 129 #define CONF_ALPHA              (CONF_UPPER|CONF_LOWER)
 130 #define CONF_ALPHA_NUMERIC      (CONF_ALPHA|CONF_NUMBER|CONF_UNDER)
 131 #define CONF_ALPHA_NUMERIC_PUNCT (CONF_ALPHA|CONF_NUMBER|CONF_UNDER| \\
 132                                   CONF_PUNCTUATION)

 134 #define KEYTYPES(c)             ((unsigned short *)((c)->meth_data))
 135 #ifndef CHARSET_EBCDIC
 136 #define IS_COMMENT(c,a)         (KEYTYPES(c)[(a)&0xff]&CONF_COMMENT)
 137 #define IS_FCOMMENT(c,a)        (KEYTYPES(c)[(a)&0xff]&CONF_FCOMMENT)
 138 #define IS_EOF(c,a)             (KEYTYPES(c)[(a)&0xff]&CONF_EOF)
 139 #define IS_ESC(c,a)             (KEYTYPES(c)[(a)&0xff]&CONF_ESC)
 140 #define IS_NUMBER(c,a)          (KEYTYPES(c)[(a)&0xff]&CONF_NUMBER)
 141 #define IS_WS(c,a)              (KEYTYPES(c)[(a)&0xff]&CONF_WS)
 142 #define IS_ALPHA_NUMERIC(c,a)   (KEYTYPES(c)[(a)&0xff]&CONF_ALPHA_NUMERIC)
 143 #define IS_ALPHA_NUMERIC_PUNCT(c,a) \\
 144                                 (KEYTYPES(c)[(a)&0xff]&CONF_ALPHA_NUMERIC_PUNCT)
 145 #define IS_QUOTE(c,a)           (KEYTYPES(c)[(a)&0xff]&CONF_QUOTE)
 146 #define IS_DQUOTE(c,a)          (KEYTYPES(c)[(a)&0xff]&CONF_DQUOTE)
 147 #define IS_HIGHBIT(c,a)         (KEYTYPES(c)[(a)&0xff]&CONF_HIGHBIT)

 149 #else /*CHARSET_EBCDIC*/

 151 #define IS_COMMENT(c,a)         (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_COMMENT)
 152 #define IS_FCOMMENT(c,a)        (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_FCOMMENT)
 153 #define IS_EOF(c,a)             (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_EOF)
 154 #define IS_ESC(c,a)             (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_ESC)
 155 #define IS_NUMBER(c,a)          (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_NUMBER)
 156 #define IS_WS(c,a)              (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_WS)
 157 #define IS_ALPHA_NUMERIC(c,a)   (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_ALPHA_NUME
 158 #define IS_ALPHA_NUMERIC_PUNCT(c,a) \\
 159                                 (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_ALPHA_NUME
 160 #define IS_QUOTE(c,a)           (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_QUOTE)
 161 #define IS_DQUOTE(c,a)          (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_DQUOTE)
 162 #define IS_HIGHBIT(c,a)         (KEYTYPES(c)[os_toascii[a]&0xff]&CONF_HIGHBIT)
 163 #endif /*CHARSET_EBCDIC*/

 165 EOF

 167 print "static unsigned short CONF_type_default[256]={";

 169 for ($i=0; $i<256; $i++)
 170         {
 171         print "\n\t" if ($i % 8) == 0;
 172         printf "0x%04X,",$V_def[$i];
 173         }

 175 print "\n\t};\n\n";

 177 print "static unsigned short CONF_type_win32[256]={";

 179 for ($i=0; $i<256; $i++)
 180         {
 181         print "\n\t" if ($i % 8) == 0;
 182         printf "0x%04X,",$V_w32[$i];
 183         }

 185 print "\n\t};\n\n";
```

```
**********************************************************
   34196 Wed Aug 13 19:33:01 2014
new/usr/src/lib/openssl/libsunw_crypto/pl/x86_64-xlate.pl
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
#!/usr/bin/env perl

# Ascetic x86_64 AT&T to MASM/NASM assembler translator by <appro>.
#
# Why AT&T to MASM and not vice versa? Several reasons. Because AT&T
# format is way easier to parse. Because it's simpler to "gear" from
# Unix ABI to Windows one [see cross-reference "card" at the end of
# file]. Because Linux targets were available first...
#
# In addition the script also "distills" code suitable for GNU
# assembler, so that it can be compiled with more rigid assemblers,
# such as Solaris /usr/ccs/bin/as.
#
# This translator is not designed to convert *arbitrary* assembler
# code from AT&T format to MASM one. It's designed to convert just
# enough to provide for dual-ABI OpenSSL modules development...
# There *are* limitations and you might have to modify your assembler
# code or this script to achieve the desired result...
#
# Currently recognized limitations:
#
# - can't use multiple ops per line;
#
# Dual-ABI styling rules.
#
# 1. Adhere to Unix register and stack layout [see cross-reference
#    ABI "card" at the end for explanation].
# 2. Forget about "red zone," stick to more traditional blended
#    stack frame allocation. If volatile storage is actually required
#    that is. If not, just leave the stack as is.
# 3. Functions tagged with ".type name,@function" get crafted with
#    unified Win64 prologue and epilogue automatically. If you want
#    to take care of ABI differences yourself, tag functions as
#    ".type name,@abi-omnipotent" instead.
# 4. To optimize the Win64 prologue you can specify number of input
#    arguments as ".type name,@function,N." Keep in mind that if N is
#    larger than 6, then you *have to* write "abi-omnipotent" code,
#    because >6 cases can't be addressed with unified prologue.
# 5. Name local labels as .L*, do *not* use dynamic labels such as 1:
#    (sorry about latter).
# 6. Don't use [or hand-code with .byte] "rep ret." "ret" mnemonic is
#    required to identify the spots, where to inject Win64 epilogue!
#    But on the pros, it's then prefixed with rep automatically:-)
# 7. Stick to explicit ip-relative addressing. If you have to use
#    GOTPCREL addressing, stick to mov symbol@GOTPCREL(%rip),%r??.
#    Both are recognized and translated to proper Win64 addressing
#    modes. To support legacy code a synthetic directive, .picmeup,
#    is implemented. It puts address of the *next* instruction into
#    target register, e.g.:
#
#               .picmeup        %rax
#               lea             .Label-.(%rax),%rax
#
# 8. In order to provide for structured exception handling unified
#    Win64 prologue copies %rsp value to %rax. For further details
#    see SEH paragraph at the end.
# 9. .init segment is allowed to contain calls to functions only.
# a. If function accepts more than 4 arguments *and* >4th argument
#    is declared as non 64-bit value, do clear its upper part.
```



```
my $flavour = shift;
my $output  = shift;
if ($flavour =~ /\./) { $output = $flavour; undef $flavour; }

open STDOUT,">$output" || die "can't open $output: $!"
        if (defined($output));

my $gas=1;      $gas=0 if ($output =~ /\.asm$/);
my $elf=1;      $elf=0 if (!$gas);
my $win64=0;
my $prefix="sunw_";
my $prefix="";
my $decor=".L";

my $masmref=8 + 50727*2**-32;   # 8.00.50727 shipped with VS2005
my $masm=0;
my $PTR=" PTR";

my $nasmref=2.03;
my $nasm=0;

if    ($flavour eq "mingw64")   { $gas=1; $elf=0; $win64=1;
                                  $prefix=`echo __USER_LABEL_PREFIX__ | $ENV{CC}
                                  chomp($prefix);
                                }
elsif ($flavour eq "macosx")    { $gas=1; $elf=0; $prefix="_"; $decor="L\$"; }
elsif ($flavour eq "masm")      { $gas=0; $elf=0; $masm=$masmref; $win64=1; $dec
elsif ($flavour eq "nasm")      { $gas=0; $elf=0; $nasm=$nasmref; $win64=1; $dec
elsif (!$gas)
{   if ($ENV{ASM} =~ m/nasm/ && `nasm -v` =~ m/version ([0-9]+)\.([0-9]+)/i)
    {   $nasm = $1 + $2*0.01; $PTR="";  }
    elsif (`ml64 2>&1` =~ m/Version ([0-9]+)\.([0-9]+)(\.([0-9]+))?/)
    {   $masm = $1 + $2*2**-16 + $4*2**-32;   }
    die "no assembler found on %PATH" if (!($nasm || $masm));
    $win64=1;
    $elf=0;
    $decor="\$L\$";
}
```

*unchanged_portion_omitted_*

```
{ package directive;    # pick up directives, which start with .
    sub re {
        my      $self = shift;  # single instance is enough...
        local   *line = shift;
        undef   $ret;
        my      $dir;
        my      %opcode =       # lea 2f-1f(%rip),%dst; 1: nop; 2:
                (       "%rax"=>0x01058d48,     "%rcx"=>0x010d8d48,
                        "%rdx"=>0x01158d48,     "%rbx"=>0x011d8d48,
                        "%rsp"=>0x01258d48,     "%rbp"=>0x012d8d48,
                        "%rsi"=>0x01358d48,     "%rdi"=>0x013d8d48,
                        "%r8" =>0x01058d4c,     "%r9" =>0x010d8d4c,
                        "%r10"=>0x01158d4c,     "%r11"=>0x011d8d4c,
                        "%r12"=>0x01258d4c,     "%r13"=>0x012d8d4c,
                        "%r14"=>0x01358d4c,     "%r15"=>0x013d8d4c      );

        if ($line =~ /^\s*(\.\w+)/) {
            $dir = $1;
            $ret = $self;
            undef $self->{value};
            $line = substr($line,@+[0]); $line =~ s/^\s+//;

            SWITCH: for ($dir) {
                /\.picmeup/ && do { if ($line =~ /(%r[\w]+)/i) {
                                        $dir="\t.long";
                                        $line=sprintf "0x%x,0x90000000",$opcode{
                                    }
```

```
                                    last;
                                };
                /\.global|\.globl|\.extern/
                            && do { $globals{$line} = $prefix . $line;
                                    $line = $globals{$line} if ($prefix);
                                    last;
                                };
                /\.type/    && do { ($sym,$type,$narg) = split(',',$line);
                                    if ($type eq "\@function") {
                                        undef $current_function;
                                        $current_function->{name} = $sym;
                                        $current_function->{abi}  = "svr4";
                                        $current_function->{narg} = $narg;
                                        $current_function->{scope} = defined($gl
                                    } elsif ($type eq "\@abi-omnipotent") {
                                        undef $current_function;
                                        $current_function->{name} = $sym;
                                        $current_function->{scope} = defined($gl
                                    }
                                    $line =~ s/\@abi\-omnipotent/\@function/;
                                    $line =~ s/\@function.*/\@function/;
                                    $line =~ s/$sym/$globals{$sym} or $sym/e;
#endif /* ! codereview */
                                    last;
                                };
                /\.asciz/   && do { if ($line =~ /^"(.*)"$/) {
                                        $dir  = ".byte";
                                        $line = join(",",unpack("C*",$1),0);
                                    }
                                    last;
                                };
                /\.rva|\.long|\.quad/
                            && do { $line =~ s/([_a-z][_a-z0-9]*)/$globals{$1} o
                                    $line =~ s/\.L/$decor/g;
                                    last;
                                };
                /\.size/    && do { $line =~ s/([_a-z][_a-z0-9]*)/$globals{$1} o
                                    last;
                                };
#endif /* ! codereview */
            }

            if ($gas) {
                $self->{value} = $dir . "\t" . $line;

                if ($dir =~ /\.extern/) {
                    $self->{value} = ""; # swallow extern
                } elsif (!$self && $dir =~ /\.type/) {
                    $self->{value} = "";
                    $self->{value} = ".def\t" . ($globals{$1} or $1) . ";\t" .
                                (defined($globals{$1})?".scl 2;":".scl 3;") .
                                "\t.type 32;\t.endef"
                                if ($win64 && $line =~ /([^,]+),\@function/);
                } elsif (!$self && $dir =~ /\.size/) {
                    $self->{value} = "";
                    if (defined($current_function)) {
                        $self->{value} .= "${decor}SEH_end_$current_function->{n
                                if ($win64 && $current_function->{abi} eq "svr4"
                        undef $current_function;
                    }
                } elsif (!$self && $dir =~ /\.align/) {
                    $self->{value} = ".p2align\t" . (log($line)/log(2));
                } elsif ($dir eq ".section") {
                    $current_segment=$line;
                    if (!$self && $current_segment eq ".init") {
                        if      ($flavour eq "macosx")  { $self->{value} = ".mod
```



```
                        elsif   ($flavour eq "mingw64") { $self->{value} = ".sec
                    }
                } elsif ($dir =~ /\.(text|data)/) {
                    $current_segment=".$1";
                } elsif ($dir =~ /\.hidden/) {
                    if      ($flavour eq "macosx")  { $self->{value} = ".private_e
                    elsif ($flavour eq "mingw64") { $self->{value} = ""; }
                    else { $self->{value} = ".hidden\t$prefix$line"; }
#endif /* ! codereview */
                } elsif ($dir =~ /\.comm/) {
                    $self->{value} = "$dir\t$prefix$line";
                    $self->{value} =~ s|,([0-9]+),([0-9]+)$|",$1,".log($2)/log(2
                }
                $line = "";
                return $self;
            }

            # non-gas case or nasm/masm
            SWITCH: for ($dir) {
                /\.text/    && do { my $v=undef;
                                    if ($nasm) {
                                        $v="section     .text code align=64\n";
                                    } else {
                                        $v="$current_segment\tENDS\n" if ($curre
                                        $current_segment = ".text\$";
                                        $v.="$current_segment\tSEGMENT ";
                                        $v.=$masm>=$masmref ? "ALIGN(64)" : "PAG
                                        $v.=" 'CODE'";
                                    }
                                    $self->{value} = $v;
                                    last;
                                };
                /\.data/    && do { my $v=undef;
                                    if ($nasm) {
                                        $v="section     .data data align=8\n";
                                    } else {
                                        $v="$current_segment\tENDS\n" if ($curre
                                        $current_segment = "_DATA";
                                        $v.="$current_segment\tSEGMENT";
                                    }
                                    $self->{value} = $v;
                                    last;
                                };
                /\.section/ && do { my $v=undef;
                                    $line =~ s/([^,]*).*/$1/;
                                    $line = ".CRT\$XCU" if ($line eq ".init");
                                    if ($nasm) {
                                        $v="section     $line";
                                        if ($line=~/\.([px])data/) {
                                            $v.=" rdata align=";
                                            $v.=$1 eq "p"? 4 : 8;
                                        } elsif ($line=~/\.CRT\$/i) {
                                            $v.=" rdata align=8";
                                        }
                                    } else {
                                        $v="$current_segment\tENDS\n" if ($curre
                                        $v.="$line\tSEGMENT";
                                        if ($line=~/\.([px])data/) {
                                            $v.=" READONLY";
                                            $v.=" ALIGN(".($1 eq "p" ? 4 : 8).")
                                        } elsif ($line=~/\.CRT\$/i) {
                                            $v.=" READONLY ";
                                            $v.=$masm>=$masmref ? "ALIGN(8)" : "
                                        }
                                    }
                                    $current_segment = $line;
```

```
                                $self->{value} = $v;
                                last;
                              };
        /\.extern/  && do { $self->{value}  = "EXTERN\t".$line;
                                $self->{value} .= ":NEAR" if ($masm);
                                last;
                              };
        /\.globl|.global/
                    && do { $self->{value}  = $masm?"PUBLIC":"global";
                                $self->{value} .= "\t".$line;
                                last;
                              };
        /\.size/    && do { if (defined($current_function)) {
                                undef $self->{value};
                                if ($current_function->{abi} eq "svr4")
                                    $self->{value}="${decor}SEH_end_$cur
                                    $self->{value}.=":\n" if($masm);
                                }
                                $self->{value}.="$current_function->{nam
                                undef $current_function;
                            }
                                last;
                              };
        /\.align/   && do { $self->{value} = "ALIGN\t".$line; last; };
        /\.(value|long|rva|quad)/
                    && do { my $sz  = substr($1,0,1);
                            my @arr = split(/,\s*/,$line);
                            my $last = pop(@arr);
                            my $conv = sub  {   my $var=shift;
                                                $var=~s/^(0b[0-1]+)/oct(
                                                $var=~s/^0x([0-9a-f]+)/0
                                                if ($sz eq "D" && ($curr
                                                { $var=~s/([_a-z\$\@][_a
                                                $var;
                                            };

                            $sz =~ tr/bvlrq/BWDDQ/;
                            $self->{value} = "\tD$sz\t";
                            for (@arr) { $self->{value} .= &$conv($_).",
                            $self->{value} .= &$conv($last);
                            last;
                          };
        /\.byte/    && do { my @str=split(/,\s*/,$line);
                            map(s/(0b[0-1]+)/oct($1)/eig,@str);
                            map(s/0x([0-9a-f]+)/0$1h/ig,@str) if ($masm)
                            while ($#str>15) {
                                $self->{value}.="DB\t"
                                        .join(",",@str[0..15])."\n";
                                foreach (0..15) { shift @str; }
                            }
                            $self->{value}.="DB\t"
                                    .join(",",@str) if (@str);
                            last;
                          };
        /\.comm/    && do { my @str=split(/,\s*/,$line);
                            my $v=undef;
                            if ($nasm) {
                                $v.="common     $prefix@str[0] @str[1]";
                            } else {
                                $v="$current_segment\tENDS\n" if ($curre
                                $current_segment = "_DATA";
                                $v.="$current_segment\tSEGMENT\n";
                                $v.="COMM       @str[0]:DWORD:".@str[1]/
                            }
                            $self->{value} = $v;
                            last;
```



```
                                  };
            }
            $line = "";
        }

        $ret;
    }
    sub out {
        my $self = shift;
        $self->{value};
    }
}

sub rex {
 local *opcode=shift;
 my ($dst,$src,$rex)=@_;

   $rex|=0x04 if($dst>=8);
   $rex|=0x01 if($src>=8);
   push @opcode,($rex|0x40) if ($rex);
}

# older gas and ml64 don't handle SSE>2 instructions
my %regrm = (   "%eax"=>0, "%ecx"=>1, "%edx"=>2, "%ebx"=>3,
                "%esp"=>4, "%ebp"=>5, "%esi"=>6, "%edi"=>7      );

my $movq = sub {        # elderly gas can't handle inter-register movq
  my $arg = shift;
  my @opcode=(0x66);
    if ($arg =~ /%xmm([0-9]+),\s*%r(\w+)/) {
        my ($src,$dst)=($1,$2);
        if ($dst !~ /[0-9]+/)   { $dst = $regrm{"%e$dst"}; }
        rex(\@opcode,$src,$dst,0x8);
        push @opcode,0x0f,0x7e;
        push @opcode,0xc0|(($src&7)<<3)|($dst&7);       # ModR/M
        @opcode;
    } elsif ($arg =~ /%r(\w+),\s*%xmm([0-9]+)/) {
        my ($src,$dst)=($2,$1);
        if ($dst !~ /[0-9]+/)   { $dst = $regrm{"%e$dst"}; }
        rex(\@opcode,$src,$dst,0x8);
        push @opcode,0x0f,0x6e;
        push @opcode,0xc0|(($src&7)<<3)|($dst&7);       # ModR/M
        @opcode;
    } else {
        ();
    }
};

my $pextrd = sub {
    if (shift =~ /\$([0-9]+),\s*%xmm([0-9]+),\s*(%\w+)/) {
      my @opcode=(0x66);
        $imm=$1;
        $src=$2;
        $dst=$3;
        if ($dst =~ /%r([0-9]+)d/)      { $dst = $1; }
        elsif ($dst =~ /%e/)            { $dst = $regrm{$dst}; }
        rex(\@opcode,$src,$dst);
        push @opcode,0x0f,0x3a,0x16;
        push @opcode,0xc0|(($src&7)<<3)|($dst&7);       # ModR/M
        push @opcode,$imm;
        @opcode;
    } else {
        ();
    }
};
```

```
712 my $pinsrd = sub {
713     if (shift =~ /\$([0-9]+),\s*(%\w+),\s*%xmm([0-9]+)/) {
714       my @opcode=(0x66);
715         $imm=$1;
716         $src=$2;
717         $dst=$3;
718         if ($src =~ /%r([0-9]+)/)       { $src = $1; }
719         elsif ($src =~ /%e/)            { $src = $regrm{$src}; }
720         rex(\@opcode,$dst,$src);
721         push @opcode,0x0f,0x3a,0x22;
722         push @opcode,0xc0|(($dst&7)<<3)|($src&7);       # ModR/M
723         push @opcode,$imm;
724         @opcode;
725     } else {
726         ();
727     }
728 };

730 my $pshufb = sub {
731     if (shift =~ /%xmm([0-9]+),\s*%xmm([0-9]+)/) {
732       my @opcode=(0x66);
733         rex(\@opcode,$2,$1);
734         push @opcode,0x0f,0x38,0x00;
735         push @opcode,0xc0|($1&7)|(($2&7)<<3);           # ModR/M
736         @opcode;
737     } else {
738         ();
739     }
740 };

742 my $palignr = sub {
743     if (shift =~ /\$([0-9]+),\s*%xmm([0-9]+),\s*%xmm([0-9]+)/) {
744       my @opcode=(0x66);
745         rex(\@opcode,$3,$2);
746         push @opcode,0x0f,0x3a,0x0f;
747         push @opcode,0xc0|($2&7)|(($3&7)<<3);           # ModR/M
748         push @opcode,$1;
749         @opcode;
750     } else {
751         ();
752     }
753 };

755 my $pclmulqdq = sub {
756     if (shift =~ /\$([x0-9a-f]+),\s*%xmm([0-9]+),\s*%xmm([0-9]+)/) {
757       my @opcode=(0x66);
758         rex(\@opcode,$3,$2);
759         push @opcode,0x0f,0x3a,0x44;
760         push @opcode,0xc0|($2&7)|(($3&7)<<3);           # ModR/M
761         my $c=$1;
762         push @opcode,$c=~/^0/?oct($c):$c;
763         @opcode;
764     } else {
765         ();
766     }
767 };

769 my $rdrand = sub {
770     if (shift =~ /%[er](\w+)/) {
771       my @opcode=();
772       my $dst=$1;
773         if ($dst !~ /[0-9]+/) { $dst = $regrm{"%e$dst"}; }
774         rex(\@opcode,0,$1,8);
775         push @opcode,0x0f,0xc7,0xf0|($dst&7);
776         @opcode;
777     } else {
```



```
778         ();
779     }
780 };

782 if ($nasm) {
783     print <<___;
784 default rel
785 %define XMMWORD
786 ___
787 } elsif ($masm) {
788     print <<___;
789 OPTION  DOTNAME
790 ___
791 }
792 while($line=<>) {

794     chomp($line);

796     $line =~ s|[#!].*$||;       # get rid of asm-style comments...
797     $line =~ s|/\*.*\*/||;      # ... and C-style comments...
798     $line =~ s|^\s+||;          # ... and skip white spaces in beginning

800     undef $label;
801     undef $opcode;
802     undef @args;

804     if ($label=label->re(\$line))       { print $label->out(); }

806     if (directive->re(\$line)) {
807         printf "%s",directive->out();
808     } elsif ($opcode=opcode->re(\$line)) {
809         my $asm = eval("\$".$opcode->mnemonic());
810         undef @bytes;

812         if ((ref($asm) eq 'CODE') && scalar(@bytes=&$asm($line))) {
813             print $gas?".byte\t":"DB\t",join(',',@bytes),"\n";
814             next;
815         }

817         ARGUMENT: while (1) {
818         my $arg;

820         if ($arg=register->re(\$line))  { opcode->size($arg->size()); }
821         elsif ($arg=const->re(\$line))  { }
822         elsif ($arg=ea->re(\$line))     { }
823         elsif ($arg=expr->re(\$line))   { }
824         else                            { last ARGUMENT; }

826         push @args,$arg;

828         last ARGUMENT if ($line !~ /^,/);

830         $line =~ s/^,\s*//;
831         } # ARGUMENT:

833         if ($#args>=0) {
834             my $insn;
835             my $sz=opcode->size();

837             if ($gas) {
838                 $insn = $opcode->out($#args>=1?$args[$#args]->size():$sz);
839                 @args = map($_->out($sz),@args);
840                 printf "\t%s\t%s",$insn,join(",",@args);
841             } else {
842                 $insn = $opcode->out();
843                 foreach (@args) {
```

```
844                     my $arg = $_->out();
845                     # $insn.=$sz compensates for movq, pinsrw, ...
846                     if ($arg =~ /^xmm[0-9]+$/) { $insn.=$sz; $sz="x" if(!$sz); l
847                     if ($arg =~ /^mm[0-9]+$/)  { $insn.=$sz; $sz="q" if(!$sz); l
848                 }
849                 @args = reverse(@args);
850                 undef $sz if ($nasm && $opcode->mnemonic() eq "lea");
851                 printf "\t%s\t%s",$insn,join(",",map($_->out($sz),@args));
852             }
853         } else {
854             printf "\t%s",$opcode->out();
855         }
856     }

858     print $line,"\n";
859 }

861 print "\n$current_segment\tENDS\n"      if ($current_segment && $masm);
862 print "END\n"                           if ($masm);

864 close STDOUT;
```



```
866 ################################################
867 # Cross-reference x86_64 ABI "card"
868 #
869 #               Unix            Win64
870 # %rax          *               *
871 # %rbx          -               -
872 # %rcx          #4              #1
873 # %rdx          #3              #2
874 # %rsi          #2              -
875 # %rdi          #1              -
876 # %rbp          -               -
877 # %rsp          -               -
878 # %r8           #5              #3
879 # %r9           #6              #4
880 # %r10          *               *
881 # %r11          *               *
882 # %r12          -               -
883 # %r13          -               -
884 # %r14          -               -
885 # %r15          -               -
886 #
887 # (*)   volatile register
888 # (-)   preserved by callee
889 # (#)   Nth argument, volatile
890 #
891 # In Unix terms top of stack is argument transfer area for arguments
892 # which could not be accomodated in registers. Or in other words 7th
893 # [integer] argument resides at 8(%rsp) upon function entry point.
894 # 128 bytes above %rsp constitute a "red zone" which is not touched
895 # by signal handlers and can be used as temporal storage without
896 # allocating a frame.
897 #
898 # In Win64 terms N*8 bytes on top of stack is argument transfer area,
899 # which belongs to/can be overwritten by callee. N is the number of
900 # arguments passed to callee, *but* not less than 4! This means that
901 # upon function entry point 5th argument resides at 40(%rsp), as well
902 # as that 32 bytes from 8(%rsp) can always be used as temporal
903 # storage [without allocating a frame]. One can actually argue that
904 # one can assume a "red zone" above stack pointer under Win64 as well.
905 # Point is that at apparently no occasion Windows kernel would alter
906 # the area above user stack pointer in true asynchronous manner...
907 #
908 # All the above means that if assembler programmer adheres to Unix
909 # register and stack layout, but disregards the "red zone" existense,
910 # it's possible to use following prologue and epilogue to "gear" from
911 # Unix to Win64 ABI in leaf functions with not more than 6 arguments.
912 #
913 # omnipotent_function:
914 # ifdef WIN64
915 #       movq    %rdi,8(%rsp)
916 #       movq    %rsi,16(%rsp)
917 #       movq    %rcx,%rdi       ; if 1st argument is actually present
918 #       movq    %rdx,%rsi       ; if 2nd argument is actually ...
919 #       movq    %r8,%rdx        ; if 3rd argument is ...
920 #       movq    %r9,%rcx        ; if 4th argument ...
921 #       movq    40(%rsp),%r8    ; if 5th ...
922 #       movq    48(%rsp),%r9    ; if 6th ...
923 # endif
924 #       ...
925 # ifdef WIN64
926 #       movq    8(%rsp),%rdi
927 #       movq    16(%rsp),%rsi
928 # endif
929 #       ret
930 #
```

```
931 ################################################
932 # Win64 SEH, Structured Exception Handling.
933 #
934 # Unlike on Unix systems(*) lack of Win64 stack unwinding information
935 # has undesired side-effect at run-time: if an exception is raised in
936 # assembler subroutine such as those in question (basically we're
937 # referring to segmentation violations caused by malformed input
938 # parameters), the application is briskly terminated without invoking
939 # any exception handlers, most notably without generating memory dump
940 # or any user notification whatsoever. This poses a problem. It's
941 # possible to address it by registering custom language-specific
942 # handler that would restore processor context to the state at
943 # subroutine entry point and return "exception is not handled, keep
944 # unwinding" code. Writing such handler can be a challenge... But it's
945 # doable, though requires certain coding convention. Consider following
946 # snippet:
947 #
948 # .type function,@function
949 # function:
950 #       movq    %rsp,%rax       # copy rsp to volatile register
951 #       pushq   %r15            # save non-volatile registers
952 #       pushq   %rbx
953 #       pushq   %rbp
954 #       movq    %rsp,%r11
955 #       subq    %rdi,%r11       # prepare [variable] stack frame
956 #       andq    $-64,%r11
957 #       movq    %rax,0(%r11)    # check for exceptions
958 #       movq    %r11,%rsp       # allocate [variable] stack frame
959 #       movq    %rax,0(%rsp)    # save original rsp value
960 # magic_point:
961 #       ...
962 #       movq    0(%rsp),%rcx    # pull original rsp value
963 #       movq    -24(%rcx),%rbp  # restore non-volatile registers
964 #       movq    -16(%rcx),%rbx
965 #       movq    -8(%rcx),%r15
966 #       movq    %rcx,%rsp       # restore original rsp
967 #       ret
968 # .size function,.-function
969 #
970 # The key is that up to magic_point copy of original rsp value remains
971 # in chosen volatile register and no non-volatile register, except for
972 # rsp, is modified. While past magic_point rsp remains constant till
973 # the very end of the function. In this case custom language-specific
974 # exception handler would look like this:
975 #
976 # EXCEPTION_DISPOSITION handler (EXCEPTION_RECORD *rec,ULONG64 frame,
977 #               CONTEXT *context,DISPATCHER_CONTEXT *disp)
978 # {     ULONG64 *rsp = (ULONG64 *)context->Rax;
979 #       if (context->Rip >= magic_point)
980 #       {   rsp = ((ULONG64 **)context->Rsp)[0];
981 #           context->Rbp = rsp[-3];
982 #           context->Rbx = rsp[-2];
983 #           context->R15 = rsp[-1];
984 #       }
985 #       context->Rsp = (ULONG64)rsp;
986 #       context->Rdi = rsp[1];
987 #       context->Rsi = rsp[2];
988 #
989 #       memcpy (disp->ContextRecord,context,sizeof(CONTEXT));
990 #       RtlVirtualUnwind(UNW_FLAG_NHANDLER,disp->ImageBase,
991 #               dips->ControlPc,disp->FunctionEntry,disp->ContextRecord,
992 #               &disp->HandlerData,&disp->EstablisherFrame,NULL);
993 #       return ExceptionContinueSearch;
994 # }
995 #
996 # It's appropriate to implement this handler in assembler, directly in
```



```
997 # function's module. In order to do that one has to know members'
998 # offsets in CONTEXT and DISPATCHER_CONTEXT structures and some constant
999 # values. Here they are:
1000 #
1001 #       CONTEXT.Rax                             120
1002 #       CONTEXT.Rcx                             128
1003 #       CONTEXT.Rdx                             136
1004 #       CONTEXT.Rbx                             144
1005 #       CONTEXT.Rsp                             152
1006 #       CONTEXT.Rbp                             160
1007 #       CONTEXT.Rsi                             168
1008 #       CONTEXT.Rdi                             176
1009 #       CONTEXT.R8                              184
1010 #       CONTEXT.R9                              192
1011 #       CONTEXT.R10                             200
1012 #       CONTEXT.R11                             208
1013 #       CONTEXT.R12                             216
1014 #       CONTEXT.R13                             224
1015 #       CONTEXT.R14                             232
1016 #       CONTEXT.R15                             240
1017 #       CONTEXT.Rip                             248
1018 #       CONTEXT.Xmm6                            512
1019 #       sizeof(CONTEXT)                         1232
1020 #       DISPATCHER_CONTEXT.ControlPc            0
1021 #       DISPATCHER_CONTEXT.ImageBase            8
1022 #       DISPATCHER_CONTEXT.FunctionEntry        16
1023 #       DISPATCHER_CONTEXT.EstablisherFrame     24
1024 #       DISPATCHER_CONTEXT.TargetIp             32
1025 #       DISPATCHER_CONTEXT.ContextRecord        40
1026 #       DISPATCHER_CONTEXT.LanguageHandler      48
1027 #       DISPATCHER_CONTEXT.HandlerData          56
1028 #       UNW_FLAG_NHANDLER                       0
1029 #       ExceptionContinueSearch                 1
1030 #
1031 # In order to tie the handler to the function one has to compose
1032 # couple of structures: one for .xdata segment and one for .pdata.
1033 #
1034 # UNWIND_INFO structure for .xdata segment would be
1035 #
1036 # function_unwind_info:
1037 #       .byte   9,0,0,0
1038 #       .rva    handler
1039 #
1040 # This structure designates exception handler for a function with
1041 # zero-length prologue, no stack frame or frame register.
1042 #
1043 # To facilitate composing of .pdata structures, auto-generated "gear"
1044 # prologue copies rsp value to rax and denotes next instruction with
1045 # .LSEH_begin_{function_name} label. This essentially defines the SEH
1046 # styling rule mentioned in the beginning. Position of this label is
1047 # chosen in such manner that possible exceptions raised in the "gear"
1048 # prologue would be accounted to caller and unwound from latter's frame.
1049 # End of function is marked with respective .LSEH_end_{function_name}
1050 # label. To summarize, .pdata segment would contain
1051 #
1052 #       .rva    .LSEH_begin_function
1053 #       .rva    .LSEH_end_function
1054 #       .rva    function_unwind_info
1055 #
1056 # Reference to functon_unwind_info from .xdata segment is the anchor.
1057 # In case you wonder why references are 32-bit .rvas and not 64-bit
1058 # .quads. References put into these two segments are required to be
1059 # *relative* to the base address of the current binary module, a.k.a.
1060 # image base. No Win64 module, be it .exe or .dll, can be larger than
1061 # 2GB and thus such relative references can be and are accommodated in
1062 # 32 bits.
```

```
1063 #
1064 # Having reviewed the example function code, one can argue that "movq
1065 # %rsp,%rax" above is redundant. It is not! Keep in mind that on Unix
1066 # rax would contain an undefined value. If this "offends" you, use
1067 # another register and refrain from modifying rax till magic_point is
1068 # reached, i.e. as if it was a non-volatile register. If more registers
1069 # are required prior [variable] frame setup is completed, note that
1070 # nobody says that you can have only one "magic point." You can
1071 # "liberate" non-volatile registers by denoting last stack off-load
1072 # instruction and reflecting it in finer grade unwind logic in handler.
1073 # After all, isn't it why it's called *language-specific* handler...
1074 #
1075 # Attentive reader can notice that exceptions would be mishandled in
1076 # auto-generated "gear" epilogue. Well, exception effectively can't
1077 # occur there, because if memory area used by it was subject to
1078 # segmentation violation, then it would be raised upon call to the
1079 # function (and as already mentioned be accounted to caller, which is
1080 # not a problem). If you're still not comfortable, then define tail
1081 # "magic point" just prior ret instruction and have handler treat it...
1082 #
1083 # (*)   Note that we're talking about run-time, not debug-time. Lack of
1084 #       unwind information makes debugging hard on both Windows and
1085 #       Unix. "Unlike" referes to the fact that on Unix signal handler
1086 #       will always be invoked, core dumped and appropriate exit code
1087 #       returned to parent (for user notification).
```

```
**********************************************************
    5680 Wed Aug 13 19:33:01 2014
new/usr/src/lib/openssl/libsunw_crypto/pl/x86_64cpuid.pl
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
#!/usr/bin/env perl

$flavour = shift;
$output  = shift;
if ($flavour =~ /\./) { $output = $flavour; undef $flavour; }

$win64=0; $win64=1 if ($flavour =~ /[nm]asm|mingw64/ || $output =~ /\.asm$/);

$0 =~ m/(.*[\/\\])[^\/\\]+$/; $dir=$1;
( $xlate="${dir}x86_64-xlate.pl" and -f $xlate ) or
( $xlate="${dir}perlasm/x86_64-xlate.pl" and -f $xlate) or
die "can't locate x86_64-xlate.pl";

open OUT,"| \"$^X\" $xlate $flavour $output";
*STDOUT=*OUT;

($arg1,$arg2,$arg3,$arg4)=$win64?("%rcx","%rdx","%r8", "%r9") : # Win64 order
                               ("%rdi","%rsi","%rdx","%rcx"); # Unix order

print<<___;
.extern         OPENSSL_cpuid_setup
.hidden         OPENSSL_cpuid_setup
.extern         illumos_locking_setup
.hidden         illumos_locking_setup
#endif /* ! codereview */
.section        .init
        call    illumos_locking_setup
#endif /* ! codereview */
        call    OPENSSL_cpuid_setup

.hidden OPENSSL_ia32cap_P
.comm   OPENSSL_ia32cap_P,8,4

.text

.globl  OPENSSL_atomic_add
.type   OPENSSL_atomic_add,\@abi-omnipotent
.align  16
OPENSSL_atomic_add:
        movl    ($arg1),%eax
.Lspin: leaq    ($arg2,%rax),%r8
        .byte   0xf0            # lock
        cmpxchgl        %r8d,($arg1)
        jne     .Lspin
        movl    %r8d,%eax
        .byte   0x48,0x98       # cltq/cdqe
        ret
.size   OPENSSL_atomic_add,.-OPENSSL_atomic_add

.globl  OPENSSL_rdtsc
.type   OPENSSL_rdtsc,\@abi-omnipotent
.align  16
OPENSSL_rdtsc:
        rdtsc
        shl     \$32,%rdx
        or      %rdx,%rax
        ret
.size   OPENSSL_rdtsc,.-OPENSSL_rdtsc

.globl  OPENSSL_ia32_cpuid
.type   OPENSSL_ia32_cpuid,\@abi-omnipotent
```



```
.align  16
OPENSSL_ia32_cpuid:
        mov     %rbx,%r8                # save %rbx

        xor     %eax,%eax
        cpuid
        mov     %eax,%r11d              # max value for standard query level

        xor     %eax,%eax
        cmp     \$0x756e6547,%ebx       # "Genu"
        setne   %al
        mov     %eax,%r9d
        cmp     \$0x49656e69,%edx       # "ineI"
        setne   %al
        or      %eax,%r9d
        cmp     \$0x6c65746e,%ecx       # "ntel"
        setne   %al
        or      %eax,%r9d               # 0 indicates Intel CPU
        jz      .Lintel

        cmp     \$0x68747541,%ebx       # "Auth"
        setne   %al
        mov     %eax,%r10d
        cmp     \$0x69746E65,%edx       # "enti"
        setne   %al
        or      %eax,%r10d
        cmp     \$0x444D4163,%ecx       # "cAMD"
        setne   %al
        or      %eax,%r10d              # 0 indicates AMD CPU
        jnz     .Lintel

        # AMD specific
        mov     \$0x80000000,%eax
        cpuid
        cmp     \$0x80000001,%eax
        jb      .Lintel
        mov     %eax,%r10d
        mov     \$0x80000001,%eax
        cpuid
        or      %ecx,%r9d
        and     \$0x00000801,%r9d       # isolate AMD XOP bit, 1<<11

        cmp     \$0x80000008,%r10d
        jb      .Lintel

        mov     \$0x80000008,%eax
        cpuid
        movzb   %cl,%r10                # number of cores - 1
        inc     %r10                    # number of cores

        mov     \$1,%eax
        cpuid
        bt      \$28,%edx               # test hyper-threading bit
        jnc     .Lgeneric
        shr     \$16,%ebx               # number of logical processors
        cmp     %r10b,%bl
        ja      .Lgeneric
        and     \$0xefffffff,%edx       # ~(1<<28)
        jmp     .Lgeneric

.Lintel:
        cmp     \$4,%r11d
        mov     \$-1,%r10d
        jb      .Lnocacheinfo

        mov     \$4,%eax
```

```
128         mov     \$0,%ecx                # query L1D
129         cpuid
130         mov     %eax,%r10d
131         shr     \$14,%r10d
132         and     \$0xfff,%r10d           # number of cores -1 per L1D

134 .Lnocacheinfo:
135         mov     \$1,%eax
136         cpuid
137         and     \$0xbfefffff,%edx       # force reserved bits to 0
138         cmp     \$0,%r9d
139         jne     .Lnotintel
140         or      \$0x40000000,%edx       # set reserved bit#30 on Intel CPUs
141         and     \$15,%ah
142         cmp     \$15,%ah                # examine Family ID
143         jne     .Lnotintel
144         or      \$0x00100000,%edx       # set reserved bit#20 to engage RC4_CHAR
145 .Lnotintel:
146         bt      \$28,%edx               # test hyper-threading bit
147         jnc     .Lgeneric
148         and     \$0xefffffff,%edx       # ~(1<<28)
149         cmp     \$0,%r10d
150         je      .Lgeneric

152         or      \$0x10000000,%edx       # 1<<28
153         shr     \$16,%ebx
154         cmp     \$1,%bl                 # see if cache is shared
155         ja      .Lgeneric
156         and     \$0xefffffff,%edx       # ~(1<<28)
157 .Lgeneric:
158         and     \$0x00000800,%r9d       # isolate AMD XOP flag
159         and     \$0xfffff7ff,%ecx
160         or      %ecx,%r9d               # merge AMD XOP flag

162         mov     %edx,%r10d              # %r9d:%r10d is copy of %ecx:%edx
163         bt      \$27,%r9d               # check OSXSAVE bit
164         jnc     .Lclear_avx
165         xor     %ecx,%ecx               # XCR0
166         .byte   0x0f,0x01,0xd0          # xgetbv
167         and     \$6,%eax                # isolate XMM and YMM state support
168         cmp     \$6,%eax
169         je      .Ldone
170 .Lclear_avx:
171         mov     \$0xefffe7ff,%eax       # ~(1<<28|1<<12|1<<11)
172         and     %eax,%r9d               # clear AVX, FMA and AMD XOP bits
173 .Ldone:
174         shl     \$32,%r9
175         mov     %r10d,%eax
176         mov     %r8,%rbx                # restore %rbx
177         or      %r9,%rax
178         ret
179 .size   OPENSSL_ia32_cpuid,.-OPENSSL_ia32_cpuid

181 .globl  OPENSSL_cleanse
182 .type   OPENSSL_cleanse,\@abi-omnipotent
183 .align  16
184 OPENSSL_cleanse:
185         xor     %rax,%rax
186         cmp     \$15,$arg2
187         jae     .Lot
188         cmp     \$0,$arg2
189         je      .Lret
190 .Little:
191         mov     %al,($arg1)
192         sub     \$1,$arg2
193         lea     1($arg1),$arg1
```



```
194         jnz     .Little
195 .Lret:
196         ret
197 .align  16
198 .Lot:
199         test    \$7,$arg1
200         jz      .Laligned
201         mov     %al,($arg1)
202         lea     -1($arg2),$arg2
203         lea     1($arg1),$arg1
204         jmp     .Lot
205 .Laligned:
206         mov     %rax,($arg1)
207         lea     -8($arg2),$arg2
208         test    \$-8,$arg2
209         lea     8($arg1),$arg1
210         jnz     .Laligned
211         cmp     \$0,$arg2
212         jne     .Little
213         ret
214 .size   OPENSSL_cleanse,.-OPENSSL_cleanse
215 ___

217 print<<___ if (!$win64);
218 .globl  OPENSSL_wipe_cpu
219 .type   OPENSSL_wipe_cpu,\@abi-omnipotent
220 .align  16
221 OPENSSL_wipe_cpu:
222         pxor    %xmm0,%xmm0
223         pxor    %xmm1,%xmm1
224         pxor    %xmm2,%xmm2
225         pxor    %xmm3,%xmm3
226         pxor    %xmm4,%xmm4
227         pxor    %xmm5,%xmm5
228         pxor    %xmm6,%xmm6
229         pxor    %xmm7,%xmm7
230         pxor    %xmm8,%xmm8
231         pxor    %xmm9,%xmm9
232         pxor    %xmm10,%xmm10
233         pxor    %xmm11,%xmm11
234         pxor    %xmm12,%xmm12
235         pxor    %xmm13,%xmm13
236         pxor    %xmm14,%xmm14
237         pxor    %xmm15,%xmm15
238         xorq    %rcx,%rcx
239         xorq    %rdx,%rdx
240         xorq    %rsi,%rsi
241         xorq    %rdi,%rdi
242         xorq    %r8,%r8
243         xorq    %r9,%r9
244         xorq    %r10,%r10
245         xorq    %r11,%r11
246         leaq    8(%rsp),%rax
247         ret
248 .size   OPENSSL_wipe_cpu,.-OPENSSL_wipe_cpu
249 ___
250 print<<___ if ($win64);
251 .globl  OPENSSL_wipe_cpu
252 .type   OPENSSL_wipe_cpu,\@abi-omnipotent
253 .align  16
254 OPENSSL_wipe_cpu:
255         pxor    %xmm0,%xmm0
256         pxor    %xmm1,%xmm1
257         pxor    %xmm2,%xmm2
258         pxor    %xmm3,%xmm3
259         pxor    %xmm4,%xmm4
```

```
260         pxor    %xmm5,%xmm5
261         xorq    %rcx,%rcx
262         xorq    %rdx,%rdx
263         xorq    %r8,%r8
264         xorq    %r9,%r9
265         xorq    %r10,%r10
266         xorq    %r11,%r11
267         leaq    8(%rsp),%rax
268         ret
269 .size   OPENSSL_wipe_cpu,.-OPENSSL_wipe_cpu
270 ___

272 print<<___;
273 .globl  OPENSSL_ia32_rdrand
274 .type   OPENSSL_ia32_rdrand,\@abi-omnipotent
275 .align  16
276 OPENSSL_ia32_rdrand:
277         mov     \$8,%ecx
278 .Loop_rdrand:
279         rdrand  %rax
280         jc      .Lbreak_rdrand
281         loop    .Loop_rdrand
282 .Lbreak_rdrand:
283         cmp     \$0,%rax
284         cmove   %rcx,%rax
285         ret
286 .size   OPENSSL_ia32_rdrand,.-OPENSSL_ia32_rdrand
287 ___

289 close STDOUT;   # flush
```

```
**********************************************************
    6344 Wed Aug 13 19:33:02 2014
new/usr/src/lib/openssl/libsunw_crypto/pl/x86asm.pl
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
______unchanged_portion_omitted_

 134 # label management
 135 $lbdecor="L";           # local label decoration, set by package
 136 $label="000";

 138 sub ::islabel           # see is argument is a known label
 139 { my $i;
 140     if ($_[0] eq "_GLOBAL_OFFSET_TABLE_") { return $_[0]; }
 141 #endif /* ! codereview */
 142     foreach $i (values %label) { return $i if ($i eq $_[0]); }
 143   $label{$_[0]};        # can be undef
 144 }

 146 sub ::label             # instantiate a function-scope label
 147 {   if (!defined($label{$_[0]}))
 148     {   $label{$_[0]}="${lbdecor}${label}${_[0]}"; $label++;   }
 149   $label{$_[0]};
 150 }

 152 sub ::LABEL             # instantiate a file-scope label
 153 {   $label{$_[0]}=$_[1] if (!defined($label{$_[0]}));
 154   $label{$_[0]};
 155 }

 157 sub ::static_label      { &::LABEL($_[0],$lbdecor.$_[0]); }

 159 sub ::set_label_B       { push(@out,"@_:\n"); }
 160 sub ::set_label
 161 { my $label=&::label($_[0]);
 162     &::align($_[1]) if ($_[1]>1);
 163     &::set_label_B($label);
 164   $label;
 165 }

 167 sub ::wipe_labels       # wipes function-scope labels
 168 {   foreach $i (keys %label)
 169     {   delete $label{$i} if ($label{$i} =~ /^\Q${lbdecor}\E[0-9]{3}/); }
 170 }

 172 # subroutine management
 173 sub ::function_begin
 174 {   &function_begin_B(@_);
 175     $stack=4;
 176     &push("ebp");
 177     &push("ebx");
 178     &push("esi");
 179     &push("edi");
 180 }

 182 sub ::function_end
 183 {   &pop("edi");
 184     &pop("esi");
 185     &pop("ebx");
 186     &pop("ebp");
 187     &ret();
 188     &function_end_B(@_);
 189     $stack=0;
 190     &wipe_labels();
 191 }
```



```
 193 sub ::function_end_A
 194 {   &pop("edi");
 195     &pop("esi");
 196     &pop("ebx");
 197     &pop("ebp");
 198     &ret();
 199     $stack+=16; # readjust esp as if we didn't pop anything
 200 }

 202 sub ::asciz
 203 { my @str=unpack("C*",shift);
 204     push @str,0;
 205     while ($#str>15) {
 206         &data_byte(@str[0..15]);
 207         foreach (0..15) { shift @str; }
 208     }
 209     &data_byte(@str) if (@str);
 210 }

 212 sub ::asm_finish
 213 {   &file_end();
 214     print @out;
 215 }

 217 sub ::asm_init
 218 { my ($type,$fn,$cpu)=@_;

 220     $filename=$fn;
 221     $i386=$cpu;

 223     $self=$cpp=$coff=$aout=$macosx=$win32=$netware=$mwerks=$android=0;
 224     if    (($type eq "elf"))
 225     {   $self=1;                    require "x86gas.pl";    }
 226     elsif (($type eq "a\.out"))
 227     {   $aout=1;                    require "x86gas.pl";    }
 228     elsif (($type eq "coff" or $type eq "gaswin"))
 229     {   $coff=1;                    require "x86gas.pl";    }
 230     elsif (($type eq "win32n"))
 231     {   $win32=1;                   require "x86nasm.pl";   }
 232     elsif (($type eq "nw-nasm"))
 233     {   $netware=1;                 require "x86nasm.pl";   }
 234     #elsif (($type eq "nw-mwasm"))
 235     #{  $netware=1; $mwerks=1;      require "x86nasm.pl";   }
 236     elsif (($type eq "win32"))
 237     {   $win32=1;                   require "x86masm.pl";   }
 238     elsif (($type eq "macosx"))
 239     {   $aout=1; $macosx=1;         require "x86gas.pl";    }
 240     elsif (($type eq "android"))
 241     {   $self=1; $android=1;        require "x86gas.pl";    }
 242     else
 243     {   print STDERR <<"EOF";
 244 Pick one target type from
 245         elf     - Linux, FreeBSD, Solaris x86, etc.
 246         a.out   - DJGPP, elder OpenBSD, etc.
 247         coff    - GAS/COFF such as Win32 targets
 248         win32n  - Windows 95/Windows NT NASM format
 249         nw-nasm - NetWare NASM format
 250         macosx  - Mac OS X
 251 EOF
 252         exit(1);
 253     }

 255     $pic=0;
 256     for (@ARGV) { $pic=1 if (/\-[fK]PIC/i); }

 258     $filename =~ s/\.pl$//;
```

```
259     &file($filename);
260 }

262 sub ::hidden {}
263 #endif /* ! codereview */

265 1;
```

```
**********************************************************
    9112 Wed Aug 13 19:33:02 2014
new/usr/src/lib/openssl/libsunw_crypto/pl/x86cpuid.pl
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
#!/usr/bin/env perl

$0 =~ m/(.*[\/\\])[^\/\\]+$/; $dir=$1;
push(@INC, "${dir}", "perlasm");
push(@INC, "${dir}perlasm", "perlasm");
require "x86asm.pl";

&asm_init($ARGV[0],"x86cpuid");

for (@ARGV) { $sse2=1 if (/-DOPENSSL_IA32_SSE2/); }

&function_begin("OPENSSL_ia32_cpuid");
        &xor    ("edx","edx");
        &pushf  ();
        &pop    ("eax");
        &mov    ("ecx","eax");
        &xor    ("eax",1<<21);
        &push   ("eax");
        &popf   ();
        &pushf  ();
        &pop    ("eax");
        &xor    ("ecx","eax");
        &xor    ("eax","eax");
        &bt     ("ecx",21);
        &jnc    (&label("nocpuid"));
        &cpuid  ();
        &mov    ("edi","eax");          # max value for standard query level

        &xor    ("eax","eax");
        &cmp    ("ebx",0x756e6547);     # "Genu"
        &setne  (&LB("eax"));
        &mov    ("ebp","eax");
        &cmp    ("edx",0x49656e69);     # "ineI"
        &setne  (&LB("eax"));
        &or     ("ebp","eax");
        &cmp    ("ecx",0x6c65746e);     # "ntel"
        &setne  (&LB("eax"));
        &or     ("ebp","eax");          # 0 indicates Intel CPU
        &jz     (&label("intel"));

        &cmp    ("ebx",0x68747541);     # "Auth"
        &setne  (&LB("eax"));
        &mov    ("esi","eax");
        &cmp    ("edx",0x69746E65);     # "enti"
        &setne  (&LB("eax"));
        &or     ("esi","eax");
        &cmp    ("ecx",0x444D4163);     # "cAMD"
        &setne  (&LB("eax"));
        &or     ("esi","eax");          # 0 indicates AMD CPU
        &jnz    (&label("intel"));

        # AMD specific
        &mov    ("eax",0x80000000);
        &cpuid  ();
        &cmp    ("eax",0x80000001);
        &jb     (&label("intel"));
        &mov    ("esi","eax");
        &mov    ("eax",0x80000001);
        &cpuid  ();
        &or     ("ebp","ecx");
        &and    ("ebp",1<<11|1);        # isolate XOP bit
```



```
        &cmp    ("esi",0x80000008);
        &jb     (&label("intel"));

        &mov    ("eax",0x80000008);
        &cpuid  ();
        &movz   ("esi",&LB("ecx"));     # number of cores - 1
        &inc    ("esi");                # number of cores

        &mov    ("eax",1);
        &xor    ("ecx","ecx");
        &cpuid  ();
        &bt     ("edx",28);
        &jnc    (&label("generic"));
        &shr    ("ebx",16);
        &and    ("ebx",0xff);
        &cmp    ("ebx","esi");
        &ja     (&label("generic"));
        &and    ("edx",0xefffffff);     # clear hyper-threading bit
        &jmp    (&label("generic"));

&set_label("intel");
        &cmp    ("edi",4);
        &mov    ("edi",-1);
        &jb     (&label("nocacheinfo"));

        &mov    ("eax",4);
        &mov    ("ecx",0);              # query L1D
        &cpuid  ();
        &mov    ("edi","eax");
        &shr    ("edi",14);
        &and    ("edi",0xfff);          # number of cores -1 per L1D

&set_label("nocacheinfo");
        &mov    ("eax",1);
        &xor    ("ecx","ecx");
        &cpuid  ();
        &and    ("edx",0xbfefffff);     # force reserved bits #20, #30 to 0
        &cmp    ("ebp",0);
        &jne    (&label("notintel"));
        &or     ("edx",1<<30);          # set reserved bit#30 on Intel CPUs
        &and    (&HB("eax"),15);        # familiy ID
        &cmp    (&HB("eax"),15);        # P4?
        &jne    (&label("notintel"));
        &or     ("edx",1<<20);          # set reserved bit#20 to engage RC4_CHAR
&set_label("notintel");
        &bt     ("edx",28);             # test hyper-threading bit
        &jnc    (&label("generic"));
        &and    ("edx",0xefffffff);
        &cmp    ("edi",0);
        &je     (&label("generic"));

        &or     ("edx",0x10000000);
        &shr    ("ebx",16);
        &cmp    (&LB("ebx"),1);
        &ja     (&label("generic"));
        &and    ("edx",0xefffffff);     # clear hyper-threading bit if not

&set_label("generic");
        &and    ("ebp",1<<11);          # isolate AMD XOP flag
        &and    ("ecx",0xfffff7ff);     # force 11th bit to 0
        &mov    ("esi","edx");
        &or     ("ebp","ecx");          # merge AMD XOP flag

        &bt     ("ecx",27);             # check OSXSAVE bit
        &jnc    (&label("clear_avx"));
        &xor    ("ecx","ecx");
```

```
127         &data_byte(0x0f,0x01,0xd0);     # xgetbv
128         &and    ("eax",6);
129         &cmp    ("eax",6);
130         &je     (&label("done"));
131         &cmp    ("eax",2);
132         &je     (&label("clear_avx"));
133 &set_label("clear_xmm");
134         &and    ("ebp",0xfdffffff);     # clear AESNI and PCLMULQDQ bits
135         &and    ("esi",0xefffffff);     # clear FXSR
136 &set_label("clear_avx");
137         &and    ("ebp",0xefffe7ff);     # clear AVX, FMA and AMD XOP bits
138 &set_label("done");
139         &mov    ("eax","esi");
140         &mov    ("edx","ebp");
141 &set_label("nocpuid");
142 &function_end("OPENSSL_ia32_cpuid");

144 &external_label("OPENSSL_ia32cap_P");

146 &function_begin_B("OPENSSL_rdtsc","EXTRN\t_OPENSSL_ia32cap_P:DWORD");
147         &xor    ("eax","eax");
148         &xor    ("edx","edx");
149         &picmeup("ecx","OPENSSL_ia32cap_P");
150         &bt     (&DWP(0,"ecx"),4);
151         &jnc    (&label("notsc"));
152         &rdtsc  ();
153 &set_label("notsc");
154         &ret    ();
155 &function_end_B("OPENSSL_rdtsc");

157 # This works in Ring 0 only [read DJGPP+MS-DOS+privileged DPMI host],
158 # but it's safe to call it on any [supported] 32-bit platform...
159 # Just check for [non-]zero return value...
160 &function_begin_B("OPENSSL_instrument_halt","EXTRN\t_OPENSSL_ia32cap_P:DWORD");
161         &picmeup("ecx","OPENSSL_ia32cap_P");
162         &bt     (&DWP(0,"ecx"),4);
163         &jnc    (&label("nohalt"));     # no TSC

165         &data_word(0x9058900e);         # push %cs; pop %eax
166         &and    ("eax",3);
167         &jnz    (&label("nohalt"));     # not enough privileges

169         &pushf  ();
170         &pop    ("eax");
171         &bt     ("eax",9);
172         &jnc    (&label("nohalt"));     # interrupts are disabled

174         &rdtsc  ();
175         &push   ("edx");
176         &push   ("eax");
177         &halt   ();
178         &rdtsc  ();

180         &sub    ("eax",&DWP(0,"esp"));
181         &sbb    ("edx",&DWP(4,"esp"));
182         &add    ("esp",8);
183         &ret    ();

185 &set_label("nohalt");
186         &xor    ("eax","eax");
187         &xor    ("edx","edx");
188         &ret    ();
189 &function_end_B("OPENSSL_instrument_halt");

191 # Essentially there is only one use for this function. Under DJGPP:
192 #
```



```
193 #       #include <go32.h>
194 #       ...
195 #       i=OPENSSL_far_spin(_dos_ds,0x46c);
196 #       ...
197 # to obtain the number of spins till closest timer interrupt.

199 &function_begin_B("OPENSSL_far_spin");
200         &pushf  ();
201         &pop    ("eax")
202         &bt     ("eax",9);
203         &jnc    (&label("nospin"));     # interrupts are disabled

205         &mov    ("eax",&DWP(4,"esp"));
206         &mov    ("ecx",&DWP(8,"esp"));
207         &data_word (0x90d88e1e);        # push %ds, mov %eax,%ds
208         &xor    ("eax","eax");
209         &mov    ("edx",&DWP(0,"ecx"));
210         &jmp    (&label("spin"));

212         &align  (16);
213 &set_label("spin");
214         &inc    ("eax");
215         &cmp    ("edx",&DWP(0,"ecx"));
216         &je     (&label("spin"));

218         &data_word (0x1f909090);        # pop   %ds
219         &ret    ();

221 &set_label("nospin");
222         &xor    ("eax","eax");
223         &xor    ("edx","edx");
224         &ret    ();
225 &function_end_B("OPENSSL_far_spin");

227 &function_begin_B("OPENSSL_wipe_cpu","EXTRN\t_OPENSSL_ia32cap_P:DWORD");
228         &xor    ("eax","eax");
229         &xor    ("edx","edx");
230         &picmeup("ecx","OPENSSL_ia32cap_P");
231         &mov    ("ecx",&DWP(0,"ecx"));
232         &bt     (&DWP(0,"ecx"),1);
233         &jnc    (&label("no_x87"));
234         if ($sse2) {
235                 &and    ("ecx",1<<26|1<<24);    # check SSE2 and FXSR bits
236                 &cmp    ("ecx",1<<26|1<<24);
237                 &jne    (&label("no_sse2"));
238                 &pxor   ("xmm0","xmm0");
239                 &pxor   ("xmm1","xmm1");
240                 &pxor   ("xmm2","xmm2");
241                 &pxor   ("xmm3","xmm3");
242                 &pxor   ("xmm4","xmm4");
243                 &pxor   ("xmm5","xmm5");
244                 &pxor   ("xmm6","xmm6");
245                 &pxor   ("xmm7","xmm7");
246         &set_label("no_sse2");
247         }
248         # just a bunch of fldz to zap the fp/mm bank followed by finit...
249         &data_word(0xeed9eed9,0xeed9eed9,0xeed9eed9,0xeed9eed9,0x90e3db9b);
250 &set_label("no_x87");
251         &lea    ("eax",&DWP(4,"esp"));
252         &ret    ();
253 &function_end_B("OPENSSL_wipe_cpu");

255 &function_begin_B("OPENSSL_atomic_add");
256         &mov    ("edx",&DWP(4,"esp"));  # fetch the pointer, 1st arg
257         &mov    ("ecx",&DWP(8,"esp"));  # fetch the increment, 2nd arg
258         &push   ("ebx");
```

```
259         &nop    ();
260         &mov    ("eax",&DWP(0,"edx"));
261 &set_label("spin");
262         &lea    ("ebx",&DWP(0,"eax","ecx"));
263         &nop    ();
264         &data_word(0x1ab10ff0); # lock; cmpxchg %ebx,(%edx)     # %eax is envolv
265         &jne    (&label("spin"));
266         &mov    ("eax","ebx");  # OpenSSL expects the new value
267         &pop    ("ebx");
268         &ret    ();
269 &function_end_B("OPENSSL_atomic_add");

271 # This function can become handy under Win32 in situations when
272 # we don't know which calling convention, __stdcall or __cdecl(*),
273 # indirect callee is using. In C it can be deployed as
274 #
275 #ifdef OPENSSL_CPUID_OBJ
276 #       type OPENSSL_indirect_call(void *f,...);
277 #       ...
278 #       OPENSSL_indirect_call(func,[up to $max arguments]);
279 #endif
280 #
281 # (*)   it's designed to work even for __fastcall if number of
282 #       arguments is 1 or 2!
283 &function_begin_B("OPENSSL_indirect_call");
284         {
285         my ($max,$i)=(7,);      # $max has to be chosen as 4*n-1
286                                 # in order to preserve eventual
287                                 # stack alignment
288         &push   ("ebp");
289         &mov    ("ebp","esp");
290         &sub    ("esp",$max*4);
291         &mov    ("ecx",&DWP(12,"ebp"));
292         &mov    (&DWP(0,"esp"),"ecx");
293         &mov    ("edx",&DWP(16,"ebp"));
294         &mov    (&DWP(4,"esp"),"edx");
295         for($i=2;$i<$max;$i++)
296                 {
297                 # Some copies will be redundant/bogus...
298                 &mov    ("eax",&DWP(12+$i*4,"ebp"));
299                 &mov    (&DWP(0+$i*4,"esp"),"eax");
300                 }
301         &call_ptr       (&DWP(8,"ebp"));# make the call...
302         &mov    ("esp","ebp");  # ... and just restore the stack pointer
303                                 # without paying attention to what we called,
304                                 # (__cdecl *func) or (__stdcall *one).
305         &pop    ("ebp");
306         &ret    ();
307         }
308 &function_end_B("OPENSSL_indirect_call");

310 &function_begin_B("OPENSSL_cleanse");
311         &mov    ("edx",&wparam(0));
312         &mov    ("ecx",&wparam(1));
313         &xor    ("eax","eax");
314         &cmp    ("ecx",7);
315         &jae    (&label("lot"));
316         &cmp    ("ecx",0);
317         &je     (&label("ret"));
318 &set_label("little");
319         &mov    (&BP(0,"edx"),"al");
320         &sub    ("ecx",1);
321         &lea    ("edx",&DWP(1,"edx"));
322         &jnz    (&label("little"));
323 &set_label("ret");
324         &ret    ();
```



```
326 &set_label("lot",16);
327         &test   ("edx",3);
328         &jz     (&label("aligned"));
329         &mov    (&BP(0,"edx"),"al");
330         &lea    ("ecx",&DWP(-1,"ecx"));
331         &lea    ("edx",&DWP(1,"edx"));
332         &jmp    (&label("lot"));
333 &set_label("aligned");
334         &mov    (&DWP(0,"edx"),"eax");
335         &lea    ("ecx",&DWP(-4,"ecx"));
336         &test   ("ecx",-4);
337         &lea    ("edx",&DWP(4,"edx"));
338         &jnz    (&label("aligned"));
339         &cmp    ("ecx",0);
340         &jne    (&label("little"));
341         &ret    ();
342 &function_end_B("OPENSSL_cleanse");

344 &function_begin_B("OPENSSL_ia32_rdrand");
345         &mov    ("ecx",8);
346 &set_label("loop");
347         &rdrand ("eax");
348         &jc     (&label("break"));
349         &loop   (&label("loop"));
350 &set_label("break");
351         &cmp    ("eax",0);
352         &cmove  ("eax","ecx");
353         &ret    ();
354 &function_end_B("OPENSSL_ia32_rdrand");

356 &initseg("illumos_locking_setup");
357 #endif /* ! codereview */
358 &initseg("OPENSSL_cpuid_setup");

360 &hidden("illumos_locking_setup");
361 &hidden("OPENSSL_cpuid_setup");
362 &hidden("OPENSSL_ia32cap_P");
363 #endif /* ! codereview */

365 &asm_finish();
```

****
5986 Wed Aug 13 19:33:03 2014
new/usr/src/lib/openssl/libsunw_crypto/pl/x86gas.pl
4853 illumos-gate is not lint-clean when built with openssl 1.0
****

```
   1 #!/usr/bin/env perl

   3 package x86gas;

   5 *out=\@::out;

   7 $::lbdecor=$::aout?"L":".L";            # local label decoration
   8 $nmdecor=($::aout or $::coff)?"_":"sunw_";      # external name decoration
   8 $nmdecor=($::aout or $::coff)?"_":"";   # external name decoration

  10 $initseg="";

  12 $align=16;
  13 $align=log($align)/log(2) if ($::aout);
  14 $com_start="#" if ($::aout or $::coff);

  16 sub opsize()
  17 { my $reg=shift;
  18     if    ($reg =~ m/^%e/o)             { "l"; }
  19     elsif ($reg =~ m/^%[a-d][hl]$/o)    { "b"; }
  20     elsif ($reg =~ m/^%[xm]/o)          { undef; }
  21     else                                { "w"; }
  22 }
_____unchanged_portion_omitted_

 250 sub ::dataseg
 251 {   push(@out,".data\n");   }

 253 *::hidden = sub { push(@out,".hidden\t$nmdecor$_[0]\n"); } if ($::elf);
 254 #endif /* ! codereview */

 256 1;
```

```
**********************************************************
    5679 Wed Aug 13 19:33:03 2014
new/usr/src/lib/openssl/libsunw_crypto/pqueue/pqueue.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/pqueue/pqueue.c */
/*
 * DTLS implementation written by Nagendra Modadugu
 * (nagendra@cs.stanford.edu) for the OpenSSL project 2005.
 */
/* ====================================================================
 * Copyright (c) 1999-2005 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

#include "cryptlib.h"
#include <openssl/bn.h>
```



```
#include <openssl/pqueue.h>
#include "pqueue.h"

typedef struct _pqueue
        {
        pitem *items;
        int count;
        } pqueue_s;

pitem *
pitem_new(unsigned char *prio64be, void *data)
        {
        pitem *item = (pitem *) OPENSSL_malloc(sizeof(pitem));
        if (item == NULL) return NULL;

        memcpy(item->priority,prio64be,sizeof(item->priority));

        item->data = data;
        item->next = NULL;

        return item;
        }

void
pitem_free(pitem *item)
        {
        if (item == NULL) return;

        OPENSSL_free(item);
        }

pqueue_s *
pqueue_new()
        {
        pqueue_s *pq = (pqueue_s *) OPENSSL_malloc(sizeof(pqueue_s));
        if (pq == NULL) return NULL;

        memset(pq, 0x00, sizeof(pqueue_s));
        return pq;
        }

void
pqueue_free(pqueue_s *pq)
        {
        if (pq == NULL) return;

        OPENSSL_free(pq);
        }

pitem *
pqueue_insert(pqueue_s *pq, pitem *item)
        {
        pitem *curr, *next;

        if (pq->items == NULL)
                {
                pq->items = item;
                return item;
                }

        for(curr = NULL, next = pq->items;
                next != NULL;
                curr = next, next = next->next)
                {
                /* we can compare 64-bit value in big-endian encoding
                 * with memcmp:-) */
```

```
127                 int cmp = memcmp(next->priority, item->priority,8);
128                 if (cmp > 0)            /* next > item */
129                         {
130                         item->next = next;
131
132                         if (curr == NULL)
133                                 pq->items = item;
134                         else
135                                 curr->next = item;
136
137                         return item;
138                         }
139
140                 else if (cmp == 0)      /* duplicates not allowed */
141                         return NULL;
142                 }
143
144         item->next = NULL;
145         curr->next = item;
146
147         return item;
148         }
149
150 pitem *
151 pqueue_peek(pqueue_s *pq)
152         {
153         return pq->items;
154         }
155
156 pitem *
157 pqueue_pop(pqueue_s *pq)
158         {
159         pitem *item = pq->items;
160
161         if (pq->items != NULL)
162                 pq->items = pq->items->next;
163
164         return item;
165         }
166
167 pitem *
168 pqueue_find(pqueue_s *pq, unsigned char *prio64be)
169         {
170         pitem *next;
171         pitem *found = NULL;
172
173         if ( pq->items == NULL)
174                 return NULL;
175
176         for ( next = pq->items; next->next != NULL; next = next->next)
177                 {
178                 if ( memcmp(next->priority, prio64be,8) == 0)
179                         {
180                         found = next;
181                         break;
182                         }
183                 }
184
185         /* check the one last node */
186         if ( memcmp(next->priority, prio64be,8) ==0)
187                 found = next;
188
189         if ( ! found)
190                 return NULL;
191
192 #if 0 /* find works in peek mode */
```



```
193         if ( prev == NULL)
194                 pq->items = next->next;
195         else
196                 prev->next = next->next;
197 #endif
198
199         return found;
200         }
201
202 void
203 pqueue_print(pqueue_s *pq)
204         {
205         pitem *item = pq->items;
206
207         while(item != NULL)
208                 {
209                 printf("item\t%02x%02x%02x%02x%02x%02x%02x%02x\n",
210                         item->priority[0],item->priority[1],
211                         item->priority[2],item->priority[3],
212                         item->priority[4],item->priority[5],
213                         item->priority[6],item->priority[7]);
214                 item = item->next;
215                 }
216         }
217
218 pitem *
219 pqueue_iterator(pqueue_s *pq)
220         {
221         return pqueue_peek(pq);
222         }
223
224 pitem *
225 pqueue_next(pitem **item)
226         {
227         pitem *ret;
228
229         if ( item == NULL || *item == NULL)
230                 return NULL;
231
232
233         /* *item != NULL */
234         ret = *item;
235         *item = (*item)->next;
236
237         return ret;
238         }
239
240 int
241 pqueue_size(pqueue_s *pq)
242 {
243         pitem *item = pq->items;
244         int count = 0;
245
246         while(item != NULL)
247         {
248                 count++;
249                 item = item->next;
250         }
251         return count;
252 }
```

unchanged_portion_omitted_

```
**********************************************************
    4691 Wed Aug 13 19:33:03 2014
new/usr/src/lib/openssl/libsunw_crypto/ts/ts_lib.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* crypto/ts/ts_lib.c */
/* Written by Zoltan Glozik (zglozik@stones.com) for the OpenSSL
 * project 2002.
 */
/* ====================================================================
 * Copyright (c) 2006 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    licensing@OpenSSL.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

#include <stdio.h>
#include "cryptlib.h"
#include <openssl/objects.h>
```



```
#include <openssl/bn.h>
#include <openssl/x509v3.h>
#include <openssl/ts.h>
#include "ts.h"

/* Local function declarations. */

/* Function definitions. */

int TS_ASN1_INTEGER_print_bio(BIO *bio, const ASN1_INTEGER *num)
        {
        BIGNUM num_bn;
        int result = 0;
        char *hex;

        BN_init(&num_bn);
        ASN1_INTEGER_to_BN(num, &num_bn);
        if ((hex = BN_bn2hex(&num_bn)))
                {
                result = BIO_write(bio, "0x", 2) > 0;
                result = result && BIO_write(bio, hex, strlen(hex)) > 0;
                OPENSSL_free(hex);
                }
        BN_free(&num_bn);

        return result;
        }

int TS_OBJ_print_bio(BIO *bio, const ASN1_OBJECT *obj)
        {
        char obj_txt[128];

        int len = OBJ_obj2txt(obj_txt, sizeof(obj_txt), obj, 0);
        BIO_write(bio, obj_txt, len);
        BIO_write(bio, "\n", 1);

        return 1;
        }

int TS_ext_print_bio(BIO *bio, const STACK_OF(X509_EXTENSION) *extensions)
        {
        int i, critical, n;
        X509_EXTENSION *ex;
        ASN1_OBJECT *obj;

        BIO_printf(bio, "Extensions:\n");
        n = X509v3_get_ext_count(extensions);
        for (i = 0; i < n; i++)
                {
                ex = X509v3_get_ext(extensions, i);
                obj = X509_EXTENSION_get_object(ex);
                i2a_ASN1_OBJECT(bio, obj);
                critical = X509_EXTENSION_get_critical(ex);
                BIO_printf(bio, ": %s\n", critical ? "critical" : "");
                if (!X509V3_EXT_print(bio, ex, 0, 4))
                        {
                        BIO_printf(bio, "%4s", "");
                        M_ASN1_OCTET_STRING_print(bio, ex->value);
                        }
                BIO_write(bio, "\n", 1);
                }

        return 1;
        }

int TS_X509_ALGOR_print_bio(BIO *bio, const X509_ALGOR *alg)
```

```
 127         {
 128         int i = OBJ_obj2nid(alg->algorithm);
 129         return BIO_printf(bio, "Hash Algorithm: %s\n",
 130                 (i == NID_undef) ? "UNKNOWN" : OBJ_nid2ln(i));
 131         }

 133 int TS_MSG_IMPRINT_print_bio(BIO *bio, TS_MSG_IMPRINT *a)
 134         {
 135         const ASN1_OCTET_STRING *msg;

 137         TS_X509_ALGOR_print_bio(bio, TS_MSG_IMPRINT_get_algo(a));

 139         BIO_printf(bio, "Message data:\n");
 140         msg = TS_MSG_IMPRINT_get_msg(a);
 141         BIO_dump_indent(bio, (const char *)M_ASN1_STRING_data(msg),
 142                         M_ASN1_STRING_length(msg), 4);

 144         return 1;
 145         }
```

```
**********************************************************
    8421 Wed Aug 13 19:33:04 2014
new/usr/src/lib/openssl/libsunw_crypto/ts/ts_rsp_print.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* crypto/ts/ts_resp_print.c */
   2 /* Written by Zoltan Glozik (zglozik@stones.com) for the OpenSSL
   3  * project 2002.
   4  */
   5 /* ====================================================================
   6  * Copyright (c) 2006 The OpenSSL Project.  All rights reserved.
   7  *
   8  * Redistribution and use in source and binary forms, with or without
   9  * modification, are permitted provided that the following conditions
  10  * are met:
  11  *
  12  * 1. Redistributions of source code must retain the above copyright
  13  *    notice, this list of conditions and the following disclaimer.
  14  *
  15  * 2. Redistributions in binary form must reproduce the above copyright
  16  *    notice, this list of conditions and the following disclaimer in
  17  *    the documentation and/or other materials provided with the
  18  *    distribution.
  19  *
  20  * 3. All advertising materials mentioning features or use of this
  21  *    software must display the following acknowledgment:
  22  *    "This product includes software developed by the OpenSSL Project
  23  *    for use in the OpenSSL Toolkit. (http://www.OpenSSL.org/)"
  24  *
  25  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  26  *    endorse or promote products derived from this software without
  27  *    prior written permission. For written permission, please contact
  28  *    licensing@OpenSSL.org.
  29  *
  30  * 5. Products derived from this software may not be called "OpenSSL"
  31  *    nor may "OpenSSL" appear in their names without prior written
  32  *    permission of the OpenSSL Project.
  33  *
  34  * 6. Redistributions of any form whatsoever must retain the following
  35  *    acknowledgment:
  36  *    "This product includes software developed by the OpenSSL Project
  37  *    for use in the OpenSSL Toolkit (http://www.OpenSSL.org/)"
  38  *
  39  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
  40  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  41  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  42  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  43  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  44  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  45  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  46  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  47  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  48  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  49  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  50  * OF THE POSSIBILITY OF SUCH DAMAGE.
  51  * ====================================================================
  52  *
  53  * This product includes cryptographic software written by Eric Young
  54  * (eay@cryptsoft.com).  This product includes software written by Tim
  55  * Hudson (tjh@cryptsoft.com).
  56  *
  57  */

  59 #include <stdio.h>
  60 #include "cryptlib.h"
  61 #include <openssl/objects.h>
```



```
  62 #include <openssl/bn.h>
  63 #include <openssl/x509v3.h>
  64 #include <openssl/ts.h>
  64 #include "ts.h"

  66 struct status_map_st
  67         {
  68         int bit;
  69         const char *text;
  70         };

  72 /* Local function declarations. */

  74 static int TS_status_map_print(BIO *bio, struct status_map_st *a,
  75                                ASN1_BIT_STRING *v);
  76 static int TS_ACCURACY_print_bio(BIO *bio, const TS_ACCURACY *accuracy);

  78 /* Function definitions. */

  80 int TS_RESP_print_bio(BIO *bio, TS_RESP *a)
  81         {
  82         TS_TST_INFO *tst_info;

  84         BIO_printf(bio, "Status info:\n");
  85         TS_STATUS_INFO_print_bio(bio, TS_RESP_get_status_info(a));

  87         BIO_printf(bio, "\nTST info:\n");
  88         tst_info = TS_RESP_get_tst_info(a);
  89         if (tst_info != NULL)
  90                 TS_TST_INFO_print_bio(bio, TS_RESP_get_tst_info(a));
  91         else
  92                 BIO_printf(bio, "Not included.\n");

  94         return 1;
  95         }

  97 int TS_STATUS_INFO_print_bio(BIO *bio, TS_STATUS_INFO *a)
  98         {
  99         static const char *status_map[] =
 100                 {
 101                 "Granted.",
 102                 "Granted with modifications.",
 103                 "Rejected.",
 104                 "Waiting.",
 105                 "Revocation warning.",
 106                 "Revoked."
 107                 };
 108         static struct status_map_st failure_map[] =
 109                 {
 110                 { TS_INFO_BAD_ALG,
 111                 "unrecognized or unsupported algorithm identifier" },
 112                 { TS_INFO_BAD_REQUEST,
 113                 "transaction not permitted or supported" },
 114                 { TS_INFO_BAD_DATA_FORMAT,
 115                 "the data submitted has the wrong format" },
 116                 { TS_INFO_TIME_NOT_AVAILABLE,
 117                 "the TSA's time source is not available" },
 118                 { TS_INFO_UNACCEPTED_POLICY,
 119                 "the requested TSA policy is not supported by the TSA" },
 120                 { TS_INFO_UNACCEPTED_EXTENSION,
 121                 "the requested extension is not supported by the TSA" },
 122                 { TS_INFO_ADD_INFO_NOT_AVAILABLE,
 123                 "the additional information requested could not be understood "
 124                 "or is not available" },
 125                 { TS_INFO_SYSTEM_FAILURE,
 126                 "the request cannot be handled due to system failure" },
```

```
127                 { -1, NULL }
128                 };
129         long status;
130         int i, lines = 0;

132         /* Printing status code. */
133         BIO_printf(bio, "Status: ");
134         status = ASN1_INTEGER_get(a->status);
135         if (0 <= status && status < (long)(sizeof(status_map)/sizeof(status_map[
136                 BIO_printf(bio, "%s\n", status_map[status]);
137         else
138                 BIO_printf(bio, "out of bounds\n");

140         /* Printing status description. */
141         BIO_printf(bio, "Status description: ");
142         for (i = 0; i < sk_ASN1_UTF8STRING_num(a->text); ++i)
143                 {
144                 if (i > 0)
145                         BIO_puts(bio, "\t");
146                 ASN1_STRING_print_ex(bio, sk_ASN1_UTF8STRING_value(a->text, i),
147                                      0);
148                 BIO_puts(bio, "\n");
149                 }
150         if (i == 0)
151                 BIO_printf(bio, "unspecified\n");

153         /* Printing failure information. */
154         BIO_printf(bio, "Failure info: ");
155         if (a->failure_info != NULL)
156                 lines = TS_status_map_print(bio, failure_map,
157                                             a->failure_info);
158         if (lines == 0)
159                 BIO_printf(bio, "unspecified");
160         BIO_printf(bio, "\n");

162         return 1;
163         }

165 static int TS_status_map_print(BIO *bio, struct status_map_st *a,
166                                ASN1_BIT_STRING *v)
167         {
168         int lines = 0;

170         for (; a->bit >= 0; ++a)
171                 {
172                 if (ASN1_BIT_STRING_get_bit(v, a->bit))
173                         {
174                         if (++lines > 1)
175                                 BIO_printf(bio, ", ");
176                         BIO_printf(bio, "%s", a->text);
177                         }
178                 }

180         return lines;
181         }

183 int TS_TST_INFO_print_bio(BIO *bio, TS_TST_INFO *a)
184         {
185         int v;
186         ASN1_OBJECT *policy_id;
187         const ASN1_INTEGER *serial;
188         const ASN1_GENERALIZEDTIME *gtime;
189         TS_ACCURACY *accuracy;
190         const ASN1_INTEGER *nonce;
191         GENERAL_NAME *tsa_name;
```



```
193         if (a == NULL) return 0;

195         /* Print version. */
196         v = TS_TST_INFO_get_version(a);
197         BIO_printf(bio, "Version: %d\n", v);

199         /* Print policy id. */
200         BIO_printf(bio, "Policy OID: ");
201         policy_id = TS_TST_INFO_get_policy_id(a);
202         TS_OBJ_print_bio(bio, policy_id);

204         /* Print message imprint. */
205         TS_MSG_IMPRINT_print_bio(bio, TS_TST_INFO_get_msg_imprint(a));

207         /* Print serial number. */
208         BIO_printf(bio, "Serial number: ");
209         serial = TS_TST_INFO_get_serial(a);
210         if (serial == NULL)
211                 BIO_printf(bio, "unspecified");
212         else
213                 TS_ASN1_INTEGER_print_bio(bio, serial);
214         BIO_write(bio, "\n", 1);

216         /* Print time stamp. */
217         BIO_printf(bio, "Time stamp: ");
218         gtime = TS_TST_INFO_get_time(a);
219         ASN1_GENERALIZEDTIME_print(bio, gtime);
220         BIO_write(bio, "\n", 1);

222         /* Print accuracy. */
223         BIO_printf(bio, "Accuracy: ");
224         accuracy = TS_TST_INFO_get_accuracy(a);
225         if (accuracy == NULL)
226                 BIO_printf(bio, "unspecified");
227         else
228                 TS_ACCURACY_print_bio(bio, accuracy);
229         BIO_write(bio, "\n", 1);

231         /* Print ordering. */
232         BIO_printf(bio, "Ordering: %s\n",
233                    TS_TST_INFO_get_ordering(a) ? "yes" : "no");

235         /* Print nonce. */
236         BIO_printf(bio, "Nonce: ");
237         nonce = TS_TST_INFO_get_nonce(a);
238         if (nonce == NULL)
239                 BIO_printf(bio, "unspecified");
240         else
241                 TS_ASN1_INTEGER_print_bio(bio, nonce);
242         BIO_write(bio, "\n", 1);

244         /* Print TSA name. */
245         BIO_printf(bio, "TSA: ");
246         tsa_name = TS_TST_INFO_get_tsa(a);
247         if (tsa_name == NULL)
248                 BIO_printf(bio, "unspecified");
249         else
250                 {
251                 STACK_OF(CONF_VALUE) *nval;
252                 if ((nval = i2v_GENERAL_NAME(NULL, tsa_name, NULL)))
253                         X509V3_EXT_val_prn(bio, nval, 0, 0);
254                 sk_CONF_VALUE_pop_free(nval, X509V3_conf_free);
255                 }
256         BIO_write(bio, "\n", 1);

258         /* Print extensions. */
```

```
259         TS_ext_print_bio(bio, TS_TST_INFO_get_exts(a));

261         return 1;
262         }

264 static int TS_ACCURACY_print_bio(BIO *bio, const TS_ACCURACY *accuracy)
265         {
266         const ASN1_INTEGER *seconds = TS_ACCURACY_get_seconds(accuracy);
267         const ASN1_INTEGER *millis = TS_ACCURACY_get_millis(accuracy);
268         const ASN1_INTEGER *micros = TS_ACCURACY_get_micros(accuracy);

270         if (seconds != NULL)
271                 TS_ASN1_INTEGER_print_bio(bio, seconds);
272         else
273                 BIO_printf(bio, "unspecified");
274         BIO_printf(bio, " seconds, ");
275         if (millis != NULL)
276                 TS_ASN1_INTEGER_print_bio(bio, millis);
277         else
278                 BIO_printf(bio, "unspecified");
279         BIO_printf(bio, " millis, ");
280         if (micros != NULL)
281                 TS_ASN1_INTEGER_print_bio(bio, micros);
282         else
283                 BIO_printf(bio, "unspecified");
284         BIO_printf(bio, " micros");

286         return 1;
287         }
```

```
**********************************************************
   21051 Wed Aug 13 19:33:04 2014
new/usr/src/lib/openssl/libsunw_crypto/x509v3/v3_utl.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
______unchanged_portion_omitted_

 356 /* hex string utilities */

 358 /* Given a buffer of length 'len' return a OPENSSL_malloc'ed string with its
 359  * hex representation
 360  * @@@ (Contents of buffer are always kept in ASCII, also on EBCDIC machines)
 361  */

 363 char *hex_to_string(const unsigned char *buffer, long len)
 364 {
 365         char *tmp, *q;
 366         const unsigned char *p;
 367         int i;
 368         static const char hexdig[] = "0123456789ABCDEF";
 368         const static char hexdig[] = "0123456789ABCDEF";
 369         if(!buffer || !len) return NULL;
 370         if(!(tmp = OPENSSL_malloc(len * 3 + 1))) {
 371                 X509V3err(X509V3_F_HEX_TO_STRING,ERR_R_MALLOC_FAILURE);
 372                 return NULL;
 373         }
 374         q = tmp;
 375         for(i = 0, p = buffer; i < len; i++,p++) {
 376                 *q++ = hexdig[(*p >> 4) & 0xf];
 377                 *q++ = hexdig[*p & 0xf];
 378                 *q++ = ':';
 379         }
 380         q[-1] = 0;
 381 #ifdef CHARSET_EBCDIC
 382         ebcdic2ascii(tmp, tmp, q - tmp - 1);
 383 #endif

 385         return tmp;
 386 }
______unchanged_portion_omitted_
```

```
**********************************************************
   14275 Wed Aug 13 19:33:04 2014
new/usr/src/lib/openssl/libsunw_ssl/bio_ssl.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* ssl/bio_ssl.c */
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```



```
#include <errno.h>
#include <openssl/opensslconf.h>
#endif /* ! codereview */
#include <openssl/crypto.h>
#include <openssl/bio.h>
#include <openssl/err.h>
#include <openssl/ssl.h>

static int ssl_write(BIO *h, const char *buf, int num);
static int ssl_read(BIO *h, char *buf, int size);
static int ssl_puts(BIO *h, const char *str);
static long ssl_ctrl(BIO *h, int cmd, long arg1, void *arg2);
static int ssl_new(BIO *h);
static int ssl_free(BIO *data);
static long ssl_callback_ctrl(BIO *h, int cmd, bio_info_cb *fp);
typedef struct bio_ssl_st
        {
        SSL *ssl; /* The ssl handle :-) */
        /* re-negotiate every time the total number of bytes is this size */
        int num_renegotiates;
        unsigned long renegotiate_count;
        unsigned long byte_count;
        unsigned long renegotiate_timeout;
        unsigned long last_time;
        } BIO_SSL;

static BIO_METHOD methods_sslp=
        {
        BIO_TYPE_SSL,"ssl",
        ssl_write,
        ssl_read,
        ssl_puts,
        NULL, /* ssl_gets, */
        ssl_ctrl,
        ssl_new,
        ssl_free,
        ssl_callback_ctrl,
        };

BIO_METHOD *BIO_f_ssl(void)
        {
        return(&methods_sslp);
        }

static int ssl_new(BIO *bi)
        {
        BIO_SSL *bs;

        bs=(BIO_SSL *)OPENSSL_malloc(sizeof(BIO_SSL));
        if (bs == NULL)
                {
                BIOerr(BIO_F_SSL_NEW,ERR_R_MALLOC_FAILURE);
                return(0);
                }
        memset(bs,0,sizeof(BIO_SSL));
        bi->init=0;
        bi->ptr=(char *)bs;
        bi->flags=0;
        return(1);
        }

static int ssl_free(BIO *a)
        {
        BIO_SSL *bs;

        if (a == NULL) return(0);
```

```
128         bs=(BIO_SSL *)a->ptr;
129         if (bs->ssl != NULL) SSL_shutdown(bs->ssl);
130         if (a->shutdown)
131                 {
132                 if (a->init && (bs->ssl != NULL))
133                         SSL_free(bs->ssl);
134                 a->init=0;
135                 a->flags=0;
136                 }
137         if (a->ptr != NULL)
138                 OPENSSL_free(a->ptr);
139         return(1);
140         }

142 static int ssl_read(BIO *b, char *out, int outl)
143         {
144         int ret=1;
145         BIO_SSL *sb;
146         SSL *ssl;
147         int retry_reason=0;
148         int r=0;

150         if (out == NULL) return(0);
151         sb=(BIO_SSL *)b->ptr;
152         ssl=sb->ssl;

154         BIO_clear_retry_flags(b);

156 #if 0
157         if (!SSL_is_init_finished(ssl))
158                 {
159 /*              ret=SSL_do_handshake(ssl); */
160                 if (ret > 0)
161                         {

163                         outflags=(BIO_FLAGS_READ|BIO_FLAGS_SHOULD_RETRY);
164                         ret= -1;
165                         goto end;
166                         }
167                 }
168 #endif
169 /*      if (ret > 0) */
170         ret=SSL_read(ssl,out,outl);

172         switch (SSL_get_error(ssl,ret))
173                 {
174         case SSL_ERROR_NONE:
175                 if (ret <= 0) break;
176                 if (sb->renegotiate_count > 0)
177                         {
178                         sb->byte_count+=ret;
179                         if (sb->byte_count > sb->renegotiate_count)
180                                 {
181                                 sb->byte_count=0;
182                                 sb->num_renegotiates++;
183                                 SSL_renegotiate(ssl);
184                                 r=1;
185                                 }
186                         }
187                 if ((sb->renegotiate_timeout > 0) && (!r))
188                         {
189                         unsigned long tm;

191                         tm=(unsigned long)time(NULL);
192                         if (tm > sb->last_time+sb->renegotiate_timeout)
193                                 {
```



```
194                                 sb->last_time=tm;
195                                 sb->num_renegotiates++;
196                                 SSL_renegotiate(ssl);
197                                 }
198                         }

200                 break;
201         case SSL_ERROR_WANT_READ:
202                 BIO_set_retry_read(b);
203                 break;
204         case SSL_ERROR_WANT_WRITE:
205                 BIO_set_retry_write(b);
206                 break;
207         case SSL_ERROR_WANT_X509_LOOKUP:
208                 BIO_set_retry_special(b);
209                 retry_reason=BIO_RR_SSL_X509_LOOKUP;
210                 break;
211         case SSL_ERROR_WANT_ACCEPT:
212                 BIO_set_retry_special(b);
213                 retry_reason=BIO_RR_ACCEPT;
214                 break;
215         case SSL_ERROR_WANT_CONNECT:
216                 BIO_set_retry_special(b);
217                 retry_reason=BIO_RR_CONNECT;
218                 break;
219         case SSL_ERROR_SYSCALL:
220         case SSL_ERROR_SSL:
221         case SSL_ERROR_ZERO_RETURN:
222         default:
223                 break;
224                 }

226         b->retry_reason=retry_reason;
227         return(ret);
228         }

230 static int ssl_write(BIO *b, const char *out, int outl)
231         {
232         int ret,r=0;
233         int retry_reason=0;
234         SSL *ssl;
235         BIO_SSL *bs;

237         if (out == NULL) return(0);
238         bs=(BIO_SSL *)b->ptr;
239         ssl=bs->ssl;

241         BIO_clear_retry_flags(b);

243 /*      ret=SSL_do_handshake(ssl);
244         if (ret > 0) */
245         ret=SSL_write(ssl,out,outl);

247         switch (SSL_get_error(ssl,ret))
248                 {
249         case SSL_ERROR_NONE:
250                 if (ret <= 0) break;
251                 if (bs->renegotiate_count > 0)
252                         {
253                         bs->byte_count+=ret;
254                         if (bs->byte_count > bs->renegotiate_count)
255                                 {
256                                 bs->byte_count=0;
257                                 bs->num_renegotiates++;
258                                 SSL_renegotiate(ssl);
259                                 r=1;
```

```
260                                 }
261                         }
262                 if ((bs->renegotiate_timeout > 0) && (!r))
263                         {
264                         unsigned long tm;

266                         tm=(unsigned long)time(NULL);
267                         if (tm > bs->last_time+bs->renegotiate_timeout)
268                                 {
269                                 bs->last_time=tm;
270                                 bs->num_renegotiates++;
271                                 SSL_renegotiate(ssl);
272                                 }
273                         }
274                 break;
275         case SSL_ERROR_WANT_WRITE:
276                 BIO_set_retry_write(b);
277                 break;
278         case SSL_ERROR_WANT_READ:
279                 BIO_set_retry_read(b);
280                 break;
281         case SSL_ERROR_WANT_X509_LOOKUP:
282                 BIO_set_retry_special(b);
283                 retry_reason=BIO_RR_SSL_X509_LOOKUP;
284                 break;
285         case SSL_ERROR_WANT_CONNECT:
286                 BIO_set_retry_special(b);
287                 retry_reason=BIO_RR_CONNECT;
288         case SSL_ERROR_SYSCALL:
289         case SSL_ERROR_SSL:
290         default:
291                 break;
292                 }

294         b->retry_reason=retry_reason;
295         return(ret);
296         }

298 static long ssl_ctrl(BIO *b, int cmd, long num, void *ptr)
299         {
300         SSL **sslp,*ssl;
301         BIO_SSL *bs;
302         BIO *dbio,*bio;
303         long ret=1;

305         bs=(BIO_SSL *)b->ptr;
306         ssl=bs->ssl;
307         if ((ssl == NULL)  && (cmd != BIO_C_SET_SSL))
308                 return(0);
309         switch (cmd)
310                 {
311         case BIO_CTRL_RESET:
312                 SSL_shutdown(ssl);

314                 if (ssl->handshake_func == ssl->method->ssl_connect)
315                         SSL_set_connect_state(ssl);
316                 else if (ssl->handshake_func == ssl->method->ssl_accept)
317                         SSL_set_accept_state(ssl);

319                 SSL_clear(ssl);

321                 if (b->next_bio != NULL)
322                         ret=BIO_ctrl(b->next_bio,cmd,num,ptr);
323                 else if (ssl->rbio != NULL)
324                         ret=BIO_ctrl(ssl->rbio,cmd,num,ptr);
325                 else
```



```
326                         ret=1;
327                 break;
328         case BIO_CTRL_INFO:
329                 ret=0;
330                 break;
331         case BIO_C_SSL_MODE:
332                 if (num) /* client mode */
333                         SSL_set_connect_state(ssl);
334                 else
335                         SSL_set_accept_state(ssl);
336                 break;
337         case BIO_C_SET_SSL_RENEGOTIATE_TIMEOUT:
338                 ret=bs->renegotiate_timeout;
339                 if (num < 60) num=5;
340                 bs->renegotiate_timeout=(unsigned long)num;
341                 bs->last_time=(unsigned long)time(NULL);
342                 break;
343         case BIO_C_SET_SSL_RENEGOTIATE_BYTES:
344                 ret=bs->renegotiate_count;
345                 if ((long)num >=512)
346                         bs->renegotiate_count=(unsigned long)num;
347                 break;
348         case BIO_C_GET_SSL_NUM_RENEGOTIATES:
349                 ret=bs->num_renegotiates;
350                 break;
351         case BIO_C_SET_SSL:
352                 if (ssl != NULL)
353                         {
354                         ssl_free(b);
355                         if (!ssl_new(b))
356                                 return 0;
357                         }
358                 b->shutdown=(int)num;
359                 ssl=(SSL *)ptr;
360                 ((BIO_SSL *)b->ptr)->ssl=ssl;
361                 bio=SSL_get_rbio(ssl);
362                 if (bio != NULL)
363                         {
364                         if (b->next_bio != NULL)
365                                 BIO_push(bio,b->next_bio);
366                         b->next_bio=bio;
367                         CRYPTO_add(&bio->references,1,CRYPTO_LOCK_BIO);
368                         }
369                 b->init=1;
370                 break;
371         case BIO_C_GET_SSL:
372                 if (ptr != NULL)
373                         {
374                         sslp=(SSL **)ptr;
375                         *sslp=ssl;
376                         }
377                 else
378                         ret=0;
379                 break;
380         case BIO_CTRL_GET_CLOSE:
381                 ret=b->shutdown;
382                 break;
383         case BIO_CTRL_SET_CLOSE:
384                 b->shutdown=(int)num;
385                 break;
386         case BIO_CTRL_WPENDING:
387                 ret=BIO_ctrl(ssl->wbio,cmd,num,ptr);
388                 break;
389         case BIO_CTRL_PENDING:
390                 ret=SSL_pending(ssl);
391                 if (ret == 0)
```

```
392                         ret=BIO_pending(ssl->rbio);
393                 break;
394         case BIO_CTRL_FLUSH:
395                 BIO_clear_retry_flags(b);
396                 ret=BIO_ctrl(ssl->wbio,cmd,num,ptr);
397                 BIO_copy_next_retry(b);
398                 break;
399         case BIO_CTRL_PUSH:
400                 if ((b->next_bio != NULL) && (b->next_bio != ssl->rbio))
401                         {
402                         SSL_set_bio(ssl,b->next_bio,b->next_bio);
403                         CRYPTO_add(&b->next_bio->references,1,CRYPTO_LOCK_BIO);
404                         }
405                 break;
406         case BIO_CTRL_POP:
407                 /* Only detach if we are the BIO explicitly being popped */
408                 if (b == ptr)
409                         {
410                         /* Shouldn't happen in practice because the
411                          * rbio and wbio are the same when pushed.
412                          */
413                         if (ssl->rbio != ssl->wbio)
414                                 BIO_free_all(ssl->wbio);
415                         if (b->next_bio != NULL)
416                                 CRYPTO_add(&b->next_bio->references,-1,CRYPTO_LO
417                         ssl->wbio=NULL;
418                         ssl->rbio=NULL;
419                         }
420                 break;
421         case BIO_C_DO_STATE_MACHINE:
422                 BIO_clear_retry_flags(b);

424                 b->retry_reason=0;
425                 ret=(int)SSL_do_handshake(ssl);

427                 switch (SSL_get_error(ssl,(int)ret))
428                         {
429                 case SSL_ERROR_WANT_READ:
430                         BIO_set_flags(b,
431                                 BIO_FLAGS_READ|BIO_FLAGS_SHOULD_RETRY);
432                         break;
433                 case SSL_ERROR_WANT_WRITE:
434                         BIO_set_flags(b,
435                                 BIO_FLAGS_WRITE|BIO_FLAGS_SHOULD_RETRY);
436                         break;
437                 case SSL_ERROR_WANT_CONNECT:
438                         BIO_set_flags(b,
439                                 BIO_FLAGS_IO_SPECIAL|BIO_FLAGS_SHOULD_RETRY);
440                         b->retry_reason=b->next_bio->retry_reason;
441                         break;
442                 default:
443                         break;
444                         }
445                 break;
446         case BIO_CTRL_DUP:
447                 dbio=(BIO *)ptr;
448                 if (((BIO_SSL *)dbio->ptr)->ssl != NULL)
449                         SSL_free(((BIO_SSL *)dbio->ptr)->ssl);
450                 ((BIO_SSL *)dbio->ptr)->ssl=SSL_dup(ssl);
451                 ((BIO_SSL *)dbio->ptr)->renegotiate_count=
452                         ((BIO_SSL *)b->ptr)->renegotiate_count;
453                 ((BIO_SSL *)dbio->ptr)->byte_count=
454                         ((BIO_SSL *)b->ptr)->byte_count;
455                 ((BIO_SSL *)dbio->ptr)->renegotiate_timeout=
456                         ((BIO_SSL *)b->ptr)->renegotiate_timeout;
457                 ((BIO_SSL *)dbio->ptr)->last_time=
```



```
458                         ((BIO_SSL *)b->ptr)->last_time;
459                 ret=(((BIO_SSL *)dbio->ptr)->ssl != NULL);
460                 break;
461         case BIO_C_GET_FD:
462                 ret=BIO_ctrl(ssl->rbio,cmd,num,ptr);
463                 break;
464         case BIO_CTRL_SET_CALLBACK:
465                 {
466 #if 0 /* FIXME: Should this be used?  -- Richard Levitte */
467                 SSLerr(SSL_F_SSL_CTRL, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
468                 ret = -1;
469 #else
470                 ret=0;
471 #endif
472                 }
473                 break;
474         case BIO_CTRL_GET_CALLBACK:
475                 {
476                 void (**fptr)(const SSL *xssl,int type,int val);

478                 fptr=(void (**)(const SSL *xssl,int type,int val))ptr;
479                 *fptr=SSL_get_info_callback(ssl);
480                 }
481                 break;
482         default:
483                 ret=BIO_ctrl(ssl->rbio,cmd,num,ptr);
484                 break;
485                 }
486         return(ret);
487         }

489 static long ssl_callback_ctrl(BIO *b, int cmd, bio_info_cb *fp)
490         {
491         SSL *ssl;
492         BIO_SSL *bs;
493         long ret=1;

495         bs=(BIO_SSL *)b->ptr;
496         ssl=bs->ssl;
497         switch (cmd)
498                 {
499         case BIO_CTRL_SET_CALLBACK:
500                 {
501                 /* FIXME: setting this via a completely different prototype
502                    seems like a crap idea */
503                 SSL_set_info_callback(ssl,(void (*)(const SSL *,int,int))fp);
504                 }
505                 break;
506         default:
507                 ret=BIO_callback_ctrl(ssl->rbio,cmd,fp);
508                 break;
509                 }
510         return(ret);
511         }

513 static int ssl_puts(BIO *bp, const char *str)
514         {
515         int n,ret;

517         n=strlen(str);
518         ret=BIO_write(bp,str,n);
519         return(ret);
520         }

522 BIO *BIO_new_buffer_ssl_connect(SSL_CTX *ctx)
523         {
```

```
524 #ifndef OPENSSL_NO_SOCK
525         BIO *ret=NULL,*buf=NULL,*ssl=NULL;

527         if ((buf=BIO_new(BIO_f_buffer())) == NULL)
528                 return(NULL);
529         if ((ssl=BIO_new_ssl_connect(ctx)) == NULL)
530                 goto err;
531         if ((ret=BIO_push(buf,ssl)) == NULL)
532                 goto err;
533         return(ret);
534 err:
535         if (buf != NULL) BIO_free(buf);
536         if (ssl != NULL) BIO_free(ssl);
537 #endif
538         return(NULL);
539         }

541 BIO *BIO_new_ssl_connect(SSL_CTX *ctx)
542         {
543 #ifndef OPENSSL_NO_SOCK
544         BIO *ret=NULL,*con=NULL,*ssl=NULL;

546         if ((con=BIO_new(BIO_s_connect())) == NULL)
547                 return(NULL);
548         if ((ssl=BIO_new_ssl(ctx,1)) == NULL)
549                 goto err;
550         if ((ret=BIO_push(ssl,con)) == NULL)
551                 goto err;
552         return(ret);
553 err:
554         if (con != NULL) BIO_free(con);
555 #endif
556         return(NULL);
557         }

559 BIO *BIO_new_ssl(SSL_CTX *ctx, int client)
560         {
561         BIO *ret;
562         SSL *ssl;

564         if ((ret=BIO_new(BIO_f_ssl())) == NULL)
565                 return(NULL);
566         if ((ssl=SSL_new(ctx)) == NULL)
567                 {
568                 BIO_free(ret);
569                 return(NULL);
570                 }
571         if (client)
572                 SSL_set_connect_state(ssl);
573         else
574                 SSL_set_accept_state(ssl);

576         BIO_set_ssl(ret,ssl,BIO_CLOSE);
577         return(ret);
578         }

580 int BIO_ssl_copy_session_id(BIO *t, BIO *f)
581         {
582         t=BIO_find_type(t,BIO_TYPE_SSL);
583         f=BIO_find_type(f,BIO_TYPE_SSL);
584         if ((t == NULL) || (f == NULL))
585                 return(0);
586         if (    (((BIO_SSL *)t->ptr)->ssl == NULL) ||
587                 (((BIO_SSL *)f->ptr)->ssl == NULL))
588                 return(0);
589         SSL_copy_session_id(((BIO_SSL *)t->ptr)->ssl,((BIO_SSL *)f->ptr)->ssl);
```



```
590         return(1);
591         }

593 void BIO_ssl_shutdown(BIO *b)
594         {
595         SSL *s;

597         while (b != NULL)
598                 {
599                 if (b->method->type == BIO_TYPE_SSL)
600                         {
601                         s=((BIO_SSL *)b->ptr)->ssl;
602                         SSL_shutdown(s);
603                         break;
604                         }
605                 b=b->next_bio;
606                 }
607         }
```

```
**********************************************************
   45471 Wed Aug 13 19:33:05 2014
new/usr/src/lib/openssl/libsunw_ssl/d1_both.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* ssl/d1_both.c */
   2 /*
   3  * DTLS implementation written by Nagendra Modadugu
   4  * (nagendra@cs.stanford.edu) for the OpenSSL project 2005.
   5  */
   6 /* ====================================================================
   7  * Copyright (c) 1998-2005 The OpenSSL Project.  All rights reserved.
   8  *
   9  * Redistribution and use in source and binary forms, with or without
  10  * modification, are permitted provided that the following conditions
  11  * are met:
  12  *
  13  * 1. Redistributions of source code must retain the above copyright
  14  *    notice, this list of conditions and the following disclaimer.
  15  *
  16  * 2. Redistributions in binary form must reproduce the above copyright
  17  *    notice, this list of conditions and the following disclaimer in
  18  *    the documentation and/or other materials provided with the
  19  *    distribution.
  20  *
  21  * 3. All advertising materials mentioning features or use of this
  22  *    software must display the following acknowledgment:
  23  *    "This product includes software developed by the OpenSSL Project
  24  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
  25  *
  26  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  27  *    endorse or promote products derived from this software without
  28  *    prior written permission. For written permission, please contact
  29  *    openssl-core@openssl.org.
  30  *
  31  * 5. Products derived from this software may not be called "OpenSSL"
  32  *    nor may "OpenSSL" appear in their names without prior written
  33  *    permission of the OpenSSL Project.
  34  *
  35  * 6. Redistributions of any form whatsoever must retain the following
  36  *    acknowledgment:
  37  *    "This product includes software developed by the OpenSSL Project
  38  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
  39  *
  40  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
  41  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  42  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  43  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  44  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  45  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  46  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  47  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  48  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
  49  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  50  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
  51  * OF THE POSSIBILITY OF SUCH DAMAGE.
  52  * ====================================================================
  53  *
  54  * This product includes cryptographic software written by Eric Young
  55  * (eay@cryptsoft.com).  This product includes software written by Tim
  56  * Hudson (tjh@cryptsoft.com).
  57  *
  58  */
  59 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
  60  * All rights reserved.
  61  *
```



```
  62  * This package is an SSL implementation written
  63  * by Eric Young (eay@cryptsoft.com).
  64  * The implementation was written so as to conform with Netscapes SSL.
  65  *
  66  * This library is free for commercial and non-commercial use as long as
  67  * the following conditions are aheared to.  The following conditions
  68  * apply to all code found in this distribution, be it the RC4, RSA,
  69  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
  70  * included with this distribution is covered by the same copyright terms
  71  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  72  *
  73  * Copyright remains Eric Young's, and as such any Copyright notices in
  74  * the code are not to be removed.
  75  * If this package is used in a product, Eric Young should be given attribution
  76  * as the author of the parts of the library used.
  77  * This can be in the form of a textual message at program startup or
  78  * in documentation (online or textual) provided with the package.
  79  *
  80  * Redistribution and use in source and binary forms, with or without
  81  * modification, are permitted provided that the following conditions
  82  * are met:
  83  * 1. Redistributions of source code must retain the copyright
  84  *    notice, this list of conditions and the following disclaimer.
  85  * 2. Redistributions in binary form must reproduce the above copyright
  86  *    notice, this list of conditions and the following disclaimer in the
  87  *    documentation and/or other materials provided with the distribution.
  88  * 3. All advertising materials mentioning features or use of this software
  89  *    must display the following acknowledgement:
  90  *    "This product includes cryptographic software written by
  91  *     Eric Young (eay@cryptsoft.com)"
  92  *    The word 'cryptographic' can be left out if the rouines from the library
  93  *    being used are not cryptographic related :-).
  94  * 4. If you include any Windows specific code (or a derivative thereof) from
  95  *    the apps directory (application code) you must include an acknowledgement:
  96  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  97  *
  98  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
  99  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 100  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 101  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 102  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 103  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 104  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 105  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 106  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 107  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 108  * SUCH DAMAGE.
 109  *
 110  * The licence and distribution terms for any publically available version or
 111  * derivative of this code cannot be changed.  i.e. this code cannot simply be
 112  * copied and put under another distribution licence
 113  * [including the GNU Public Licence.]
 114  */

 116 #include <limits.h>
 117 #include <string.h>
 118 #include <stdio.h>
 119 #include "ssl_locl.h"
 120 #include <openssl/buffer.h>
 121 #include <openssl/rand.h>
 122 #include <openssl/objects.h>
 123 #include <openssl/evp.h>
 124 #include <openssl/x509.h>

 126 #define RSMBLY_BITMASK_SIZE(msg_len) (((msg_len) + 7) / 8)
```

```
128 #define RSMBLY_BITMASK_MARK(bitmask, start, end) { \
129                         if ((end) - (start) <= 8) { \
130                                 long ii; \
131                                 for (ii = (start); ii < (end); ii++) bitmask[((i
132                         } else { \
133                                 long ii; \
134                                 bitmask[((start) >> 3)] |= bitmask_start_values[
135                                 for (ii = (((start) >> 3) + 1); ii < ((((end) -
136                                 bitmask[(((end) - 1) >> 3)] |= bitmask_end_value
137                         } }

139 #define RSMBLY_BITMASK_IS_COMPLETE(bitmask, msg_len, is_complete) { \
140                         long ii; \
141                         OPENSSL_assert((msg_len) > 0); \
142                         is_complete = 1; \
143                         if (bitmask[(((msg_len) - 1) >> 3)] != bitmask_end_value
144                         if (is_complete) for (ii = (((msg_len) - 1) >> 3) - 1; i
145                                 if (bitmask[ii] != 0xff) { is_complete = 0; brea

147 #if 0
148 #define RSMBLY_BITMASK_PRINT(bitmask, msg_len) { \
149                         long ii; \
150                         printf("bitmask: "); for (ii = 0; ii < (msg_len); ii++)
151                         printf("%d ", (bitmask[ii >> 3] & (1 << (ii & 7))) >> (i
152                         printf("\n"); }
153 #endif

155 static unsigned char bitmask_start_values[] = {0xff, 0xfe, 0xfc, 0xf8, 0xf0, 0xe
156 static unsigned char bitmask_end_values[]   = {0xff, 0x01, 0x03, 0x07, 0x0f, 0x1

158 /* XDTLS:  figure out the right values */
159 static unsigned int g_probable_mtu[] = {1500 - 28, 512 - 28, 256 - 28};

161 static unsigned int dtls1_guess_mtu(unsigned int curr_mtu);
162 static void dtls1_fix_message_header(SSL *s, unsigned long frag_off,
163         unsigned long frag_len);
164 static unsigned char *dtls1_write_message_header(SSL *s,
165         unsigned char *p);
166 static void dtls1_set_message_header_int(SSL *s, unsigned char mt,
167         unsigned long len, unsigned short seq_num, unsigned long frag_off,
168         unsigned long frag_len);
169 static long dtls1_get_message_fragment(SSL *s, int st1, int stn,
170         long max, int *ok);

172 static hm_fragment *
173 dtls1_hm_fragment_new(unsigned long frag_len, int reassembly)
174         {
175         hm_fragment *frag = NULL;
176         unsigned char *buf = NULL;
177         unsigned char *bitmask = NULL;

179         frag = (hm_fragment *)OPENSSL_malloc(sizeof(hm_fragment));
180         if ( frag == NULL)
181                 return NULL;

183         if (frag_len)
184                 {
185                 buf = (unsigned char *)OPENSSL_malloc(frag_len);
186                 if ( buf == NULL)
187                         {
188                         OPENSSL_free(frag);
189                         return NULL;
190                         }
191                 }

193         /* zero length fragment gets zero frag->fragment */
```



```
194         frag->fragment = buf;

196         /* Initialize reassembly bitmask if necessary */
197         if (reassembly)
198                 {
199                 bitmask = (unsigned char *)OPENSSL_malloc(RSMBLY_BITMASK_SIZE(fr
200                 if (bitmask == NULL)
201                         {
202                         if (buf != NULL) OPENSSL_free(buf);
203                         OPENSSL_free(frag);
204                         return NULL;
205                         }
206                 memset(bitmask, 0, RSMBLY_BITMASK_SIZE(frag_len));
207                 }

209         frag->reassembly = bitmask;

211         return frag;
212         }

214 static void
215 dtls1_hm_fragment_free(hm_fragment *frag)
216         {

218         if (frag->msg_header.is_ccs)
219                 {
220                 EVP_CIPHER_CTX_free(frag->msg_header.saved_retransmit_state.enc_
221                 EVP_MD_CTX_destroy(frag->msg_header.saved_retransmit_state.write
222                 }
223         if (frag->fragment) OPENSSL_free(frag->fragment);
224         if (frag->reassembly) OPENSSL_free(frag->reassembly);
225         OPENSSL_free(frag);
226         }

228 /* send s->init_buf in records of type 'type' (SSL3_RT_HANDSHAKE or SSL3_RT_CHAN
229 int dtls1_do_write(SSL *s, int type)
230         {
231         int ret;
232         int curr_mtu;
233         unsigned int len, frag_off, mac_size, blocksize;

235         /* AHA!  Figure out the MTU, and stick to the right size */
236         if (s->d1->mtu < dtls1_min_mtu() && !(SSL_get_options(s) & SSL_OP_NO_QUE
237                 {
238                 s->d1->mtu =
239                         BIO_ctrl(SSL_get_wbio(s), BIO_CTRL_DGRAM_QUERY_MTU, 0, N

241                 /* I've seen the kernel return bogus numbers when it doesn't kno
242                  * (initial write), so just make sure we have a reasonable numbe
243                 if (s->d1->mtu < dtls1_min_mtu())
244                         {
245                         s->d1->mtu = 0;
246                         s->d1->mtu = dtls1_guess_mtu(s->d1->mtu);
247                         BIO_ctrl(SSL_get_wbio(s), BIO_CTRL_DGRAM_SET_MTU,
248                                 s->d1->mtu, NULL);
249                         }
250                 }
251 #if 0
252         mtu = s->d1->mtu;

254         fprintf(stderr, "using MTU = %d\n", mtu);

256         mtu -= (DTLS1_HM_HEADER_LENGTH + DTLS1_RT_HEADER_LENGTH);

258         curr_mtu = mtu - BIO_wpending(SSL_get_wbio(s));
```

```
260         if ( curr_mtu > 0)
261                 mtu = curr_mtu;
262         else if ( ( ret = BIO_flush(SSL_get_wbio(s))) <= 0)
263                 return ret;
264
265         if ( BIO_wpending(SSL_get_wbio(s)) + s->init_num >= mtu)
266                 {
267                 ret = BIO_flush(SSL_get_wbio(s));
268                 if ( ret <= 0)
269                         return ret;
270                 mtu = s->d1->mtu - (DTLS1_HM_HEADER_LENGTH + DTLS1_RT_HEADER_LEN
271                 }
272 #endif
273
274         OPENSSL_assert(s->d1->mtu >= dtls1_min_mtu());  /* should have something
275
276         if ( s->init_off == 0  && type == SSL3_RT_HANDSHAKE)
277                 OPENSSL_assert(s->init_num ==
278                         (int)s->d1->w_msg_hdr.msg_len + DTLS1_HM_HEADER_LENGTH);
279
280         if (s->write_hash)
281                 mac_size = EVP_MD_CTX_size(s->write_hash);
282         else
283                 mac_size = 0;
284
285         if (s->enc_write_ctx &&
286                 (EVP_CIPHER_mode( s->enc_write_ctx->cipher) & EVP_CIPH_CBC_MODE)
287                 blocksize = 2 * EVP_CIPHER_block_size(s->enc_write_ctx->cipher);
288         else
289                 blocksize = 0;
290
291         frag_off = 0;
292         while( s->init_num)
293                 {
294                 curr_mtu = s->d1->mtu - BIO_wpending(SSL_get_wbio(s)) -
295                         DTLS1_RT_HEADER_LENGTH - mac_size - blocksize;
296
297                 if ( curr_mtu <= DTLS1_HM_HEADER_LENGTH)
298                         {
299                         /* grr.. we could get an error if MTU picked was wrong *
300                         ret = BIO_flush(SSL_get_wbio(s));
301                         if ( ret <= 0)
302                                 return ret;
303                         curr_mtu = s->d1->mtu - DTLS1_RT_HEADER_LENGTH -
304                                 mac_size - blocksize;
305                         }
306
307                 if ( s->init_num > curr_mtu)
308                         len = curr_mtu;
309                 else
310                         len = s->init_num;
311
312
313                 /* XDTLS: this function is too long.  split out the CCS part */
314                 if ( type == SSL3_RT_HANDSHAKE)
315                         {
316                         if ( s->init_off != 0)
317                                 {
318                                 OPENSSL_assert(s->init_off > DTLS1_HM_HEADER_LEN
319                                 s->init_off -= DTLS1_HM_HEADER_LENGTH;
320                                 s->init_num += DTLS1_HM_HEADER_LENGTH;
321
322                                 if ( s->init_num > curr_mtu)
323                                         len = curr_mtu;
324                                 else
325                                         len = s->init_num;
```



```
326                                 }
327
328                         dtls1_fix_message_header(s, frag_off,
329                                 len - DTLS1_HM_HEADER_LENGTH);
330
331                         dtls1_write_message_header(s, (unsigned char *)&s->init_
332
333                         OPENSSL_assert(len >= DTLS1_HM_HEADER_LENGTH);
334                         }
335
336                 ret=dtls1_write_bytes(s,type,&s->init_buf->data[s->init_off],
337                         len);
338                 if (ret < 0)
339                         {
340                         /* might need to update MTU here, but we don't know
341                          * which previous packet caused the failure -- so can't
342                          * really retransmit anything.  continue as if everythin
343                          * is fine and wait for an alert to handle the
344                          * retransmit
345                          */
346                         if ( BIO_ctrl(SSL_get_wbio(s),
347                                 BIO_CTRL_DGRAM_MTU_EXCEEDED, 0, NULL) > 0 )
348                                 s->d1->mtu = BIO_ctrl(SSL_get_wbio(s),
349                                         BIO_CTRL_DGRAM_QUERY_MTU, 0, NULL);
350                         else
351                                 return(-1);
352                         }
353                 else
354                         {
355
356                         /* bad if this assert fails, only part of the handshake
357                          * message got sent.  but why would this happen? */
358                         OPENSSL_assert(len == (unsigned int)ret);
359
360                         if (type == SSL3_RT_HANDSHAKE && ! s->d1->retransmitting
361                                 {
362                                 /* should not be done for 'Hello Request's, but
363                                  * we'll ignore the result anyway */
364                                 unsigned char *p = (unsigned char *)&s->init_buf
365                                 const struct hm_header_st *msg_hdr = &s->d1->w_m
366                                 int xlen;
367
368                                 if (frag_off == 0 && s->version != DTLS1_BAD_VER
369                                         {
370                                         /* reconstruct message header is if it
371                                          * is being sent in single fragment */
372                                         *p++ = msg_hdr->type;
373                                         l2n3(msg_hdr->msg_len,p);
374                                         s2n (msg_hdr->seq,p);
375                                         l2n3(0,p);
376                                         l2n3(msg_hdr->msg_len,p);
377                                         p  -= DTLS1_HM_HEADER_LENGTH;
378                                         xlen = ret;
379                                         }
380                                 else
381                                         {
382                                         p  += DTLS1_HM_HEADER_LENGTH;
383                                         xlen = ret - DTLS1_HM_HEADER_LENGTH;
384                                         }
385
386                                 ssl3_finish_mac(s, p, xlen);
387                                 }
388
389                         if (ret == s->init_num)
390                                 {
391                                 if (s->msg_callback)
```

```
392                                     s->msg_callback(1, s->version, type, s->
393                                             (size_t)(s->init_off + s->init_n
394                                             s->msg_callback_arg);

396                                 s->init_off = 0;  /* done writing this message *
397                                 s->init_num = 0;

399                                 return(1);
400                                 }
401                         s->init_off+=ret;
402                         s->init_num-=ret;
403                         frag_off += (ret -= DTLS1_HM_HEADER_LENGTH);
404                         }
405                 }
406         return(0);
407         }


410 /* Obtain handshake message of message type 'mt' (any if mt == -1),
411  * maximum acceptable body length 'max'.
412  * Read an entire handshake message.  Handshake messages arrive in
413  * fragments.
414  */
415 long dtls1_get_message(SSL *s, int st1, int stn, int mt, long max, int *ok)
416         {
417         int i, al;
418         struct hm_header_st *msg_hdr;
419         unsigned char *p;
420         unsigned long msg_len;

422         /* s3->tmp is used to store messages that are unexpected, caused
423          * by the absence of an optional handshake message */
424         if (s->s3->tmp.reuse_message)
425                 {
426                 s->s3->tmp.reuse_message=0;
427                 if ((mt >= 0) && (s->s3->tmp.message_type != mt))
428                         {
429                         al=SSL_AD_UNEXPECTED_MESSAGE;
430                         SSLerr(SSL_F_DTLS1_GET_MESSAGE,SSL_R_UNEXPECTED_MESSAGE)
431                         goto f_err;
432                         }
433                 *ok=1;
434                 s->init_msg = s->init_buf->data + DTLS1_HM_HEADER_LENGTH;
435                 s->init_num = (int)s->s3->tmp.message_size;
436                 return s->init_num;
437                 }

439         msg_hdr = &s->d1->r_msg_hdr;
440         memset(msg_hdr, 0x00, sizeof(struct hm_header_st));

442 again:
443         i = dtls1_get_message_fragment(s, st1, stn, max, ok);
444         if ( i == DTLS1_HM_BAD_FRAGMENT ||
445                 i == DTLS1_HM_FRAGMENT_RETRY)  /* bad fragment received */
446                 goto again;
447         else if ( i <= 0 && !*ok)
448                 return i;

450         p = (unsigned char *)s->init_buf->data;
451         msg_len = msg_hdr->msg_len;

453         /* reconstruct message header */
454         *(p++) = msg_hdr->type;
455         l2n3(msg_len,p);
456         s2n (msg_hdr->seq,p);
457         l2n3(0,p);
```



```
458         l2n3(msg_len,p);
459         if (s->version != DTLS1_BAD_VER) {
460                 p       -= DTLS1_HM_HEADER_LENGTH;
461                 msg_len += DTLS1_HM_HEADER_LENGTH;
462         }

464         ssl3_finish_mac(s, p, msg_len);
465         if (s->msg_callback)
466                 s->msg_callback(0, s->version, SSL3_RT_HANDSHAKE,
467                         p, msg_len,
468                         s, s->msg_callback_arg);

470         memset(msg_hdr, 0x00, sizeof(struct hm_header_st));

472         /* Don't change sequence numbers while listening */
473         if (!s->d1->listen)
474                 s->d1->handshake_read_seq++;

476         s->init_msg = s->init_buf->data + DTLS1_HM_HEADER_LENGTH;
477         return s->init_num;

479 f_err:
480         ssl3_send_alert(s,SSL3_AL_FATAL,al);
481         *ok = 0;
482         return -1;
483         }


486 static int dtls1_preprocess_fragment(SSL *s,struct hm_header_st *msg_hdr,int max
487         {
488         size_t frag_off,frag_len,msg_len;

490         msg_len  = msg_hdr->msg_len;
491         frag_off = msg_hdr->frag_off;
492         frag_len = msg_hdr->frag_len;

494         /* sanity checking */
495         if ( (frag_off+frag_len) > msg_len)
496                 {
497                 SSLerr(SSL_F_DTLS1_PREPROCESS_FRAGMENT,SSL_R_EXCESSIVE_MESSAGE_S
498                 return SSL_AD_ILLEGAL_PARAMETER;
499                 }

501         if ( (frag_off+frag_len) > (unsigned long)max)
502                 {
503                 SSLerr(SSL_F_DTLS1_PREPROCESS_FRAGMENT,SSL_R_EXCESSIVE_MESSAGE_S
504                 return SSL_AD_ILLEGAL_PARAMETER;
505                 }

507         if ( s->d1->r_msg_hdr.frag_off == 0) /* first fragment */
508                 {
509                 /* msg_len is limited to 2^24, but is effectively checked
510                  * against max above */
511                 if (!BUF_MEM_grow_clean(s->init_buf,msg_len+DTLS1_HM_HEADER_LENG
512                         {
513                         SSLerr(SSL_F_DTLS1_PREPROCESS_FRAGMENT,ERR_R_BUF_LIB);
514                         return SSL_AD_INTERNAL_ERROR;
515                         }

517                 s->s3->tmp.message_size  = msg_len;
518                 s->d1->r_msg_hdr.msg_len = msg_len;
519                 s->s3->tmp.message_type  = msg_hdr->type;
520                 s->d1->r_msg_hdr.type    = msg_hdr->type;
521                 s->d1->r_msg_hdr.seq     = msg_hdr->seq;
522                 }
523         else if (msg_len != s->d1->r_msg_hdr.msg_len)
```

```
524                 {
525                 /* They must be playing with us! BTW, failure to enforce
526                  * upper limit would open possibility for buffer overrun. */
527                 SSLerr(SSL_F_DTLS1_PREPROCESS_FRAGMENT,SSL_R_EXCESSIVE_MESSAGE_S
528                 return SSL_AD_ILLEGAL_PARAMETER;
529                 }

531         return 0; /* no error */
532         }


535 static int
536 dtls1_retrieve_buffered_fragment(SSL *s, long max, int *ok)
537         {
538         /* (0) check whether the desired fragment is available
539          * if so:
540          * (1) copy over the fragment to s->init_buf->data[]
541          * (2) update s->init_num
542          */
543         pitem *item;
544         hm_fragment *frag;
545         int al;

547         *ok = 0;
548         item = pqueue_peek(s->d1->buffered_messages);
549         if ( item == NULL)
550                 return 0;

552         frag = (hm_fragment *)item->data;

554         /* Don't return if reassembly still in progress */
555         if (frag->reassembly != NULL)
556                 return 0;

558         if ( s->d1->handshake_read_seq == frag->msg_header.seq)
559                 {
560                 unsigned long frag_len = frag->msg_header.frag_len;
561                 pqueue_pop(s->d1->buffered_messages);

563                 al=dtls1_preprocess_fragment(s,&frag->msg_header,max);

565                 if (al==0) /* no alert */
566                         {
567                         unsigned char *p = (unsigned char *)s->init_buf->data+DT
568                         memcpy(&p[frag->msg_header.frag_off],
569                                 frag->fragment,frag->msg_header.frag_len);
570                         }

572                 dtls1_hm_fragment_free(frag);
573                 pitem_free(item);

575                 if (al==0)
576                         {
577                         *ok = 1;
578                         return frag_len;
579                         }

581                 ssl3_send_alert(s,SSL3_AL_FATAL,al);
582                 s->init_num = 0;
583                 *ok = 0;
584                 return -1;
585                 }
586         else
587                 return 0;
588         }
```



```
590 /* dtls1_max_handshake_message_len returns the maximum number of bytes
591  * permitted in a DTLS handshake message for |s|. The minimum is 16KB, but may
592  * be greater if the maximum certificate list size requires it. */
593 static unsigned long dtls1_max_handshake_message_len(const SSL *s)
594         {
595         unsigned long max_len = DTLS1_HM_HEADER_LENGTH + SSL3_RT_MAX_ENCRYPTED_L
596         if (max_len < (unsigned long)s->max_cert_list)
597                 return s->max_cert_list;
598         return max_len;
599         }

601 static int
602 dtls1_reassemble_fragment(SSL *s, const struct hm_header_st* msg_hdr, int *ok)
603         {
604         hm_fragment *frag = NULL;
605         pitem *item = NULL;
606         int i = -1, is_complete;
607         unsigned char seq64be[8];
608         unsigned long frag_len = msg_hdr->frag_len;

610         if ((msg_hdr->frag_off+frag_len) > msg_hdr->msg_len ||
611                 msg_hdr->msg_len > dtls1_max_handshake_message_len(s))
612                 goto err;

614         if (frag_len == 0)
615                 return DTLS1_HM_FRAGMENT_RETRY;

617         /* Try to find item in queue */
618         memset(seq64be,0,sizeof(seq64be));
619         seq64be[6] = (unsigned char) (msg_hdr->seq>>8);
620         seq64be[7] = (unsigned char) msg_hdr->seq;
621         item = pqueue_find(s->d1->buffered_messages, seq64be);

623         if (item == NULL)
624                 {
625                 frag = dtls1_hm_fragment_new(msg_hdr->msg_len, 1);
626                 if ( frag == NULL)
627                         goto err;
628                 memcpy(&(frag->msg_header), msg_hdr, sizeof(*msg_hdr));
629                 frag->msg_header.frag_len = frag->msg_header.msg_len;
630                 frag->msg_header.frag_off = 0;
631                 }
632         else
633                 {
634                 frag = (hm_fragment*) item->data;
635                 if (frag->msg_header.msg_len != msg_hdr->msg_len)
636                         {
637                         item = NULL;
638                         frag = NULL;
639                         goto err;
640                         }
641                 }


643         /* If message is already reassembled, this must be a
644          * retransmit and can be dropped. In this case item != NULL and so frag
645          * does not need to be freed.
646          */
647         if (frag->reassembly == NULL)
648                 {
649                 unsigned char devnull [256];

651                 while (frag_len)
652                         {
653                         i = s->method->ssl_read_bytes(s,SSL3_RT_HANDSHAKE,
654                                 devnull,
```

```
655                                 frag_len>sizeof(devnull)?sizeof(devnull):frag_le
656                         if (i<=0) goto err;
657                         frag_len -= i;
658                         }
659                 return DTLS1_HM_FRAGMENT_RETRY;
660                 }

662         /* read the body of the fragment (header has already been read */
663         i = s->method->ssl_read_bytes(s,SSL3_RT_HANDSHAKE,
664                 frag->fragment + msg_hdr->frag_off,frag_len,0);
665         if ((unsigned long)i!=frag_len)
666                 i=-1;
667         if (i<=0)
668                 goto err;

670         RSMBLY_BITMASK_MARK(frag->reassembly, (long)msg_hdr->frag_off,
671                             (long)(msg_hdr->frag_off + frag_len));

673         RSMBLY_BITMASK_IS_COMPLETE(frag->reassembly, (long)msg_hdr->msg_len,
674                                    is_complete);

676         if (is_complete)
677                 {
678                 OPENSSL_free(frag->reassembly);
679                 frag->reassembly = NULL;
680                 }

682         if (item == NULL)
683                 {
684                 item = pitem_new(seq64be, frag);
685                 if (item == NULL)
686                         {
687                         i = -1;
688                         goto err;
689                         }

691                 item = pqueue_insert(s->d1->buffered_messages, item);
692                 /* pqueue_insert fails iff a duplicate item is inserted.
693                  * However, |item| cannot be a duplicate. If it were,
694                  * |pqueue_find|, above, would have returned it and control
695                  * would never have reached this branch. */
696                 OPENSSL_assert(item != NULL);
697                 }

699         return DTLS1_HM_FRAGMENT_RETRY;

701 err:
702         if (frag != NULL && item == NULL) dtls1_hm_fragment_free(frag);
703         *ok = 0;
704         return i;
705         }


708 static int
709 dtls1_process_out_of_seq_message(SSL *s, const struct hm_header_st* msg_hdr, int
710 {
711         int i=-1;
712         hm_fragment *frag = NULL;
713         pitem *item = NULL;
714         unsigned char seq64be[8];
715         unsigned long frag_len = msg_hdr->frag_len;

717         if ((msg_hdr->frag_off+frag_len) > msg_hdr->msg_len)
718                 goto err;

720         /* Try to find item in queue, to prevent duplicate entries */
```



```
721         memset(seq64be,0,sizeof(seq64be));
722         seq64be[6] = (unsigned char) (msg_hdr->seq>>8);
723         seq64be[7] = (unsigned char) msg_hdr->seq;
724         item = pqueue_find(s->d1->buffered_messages, seq64be);

726         /* If we already have an entry and this one is a fragment,
727          * don't discard it and rather try to reassemble it.
728          */
729         if (item != NULL && frag_len != msg_hdr->msg_len)
730                 item = NULL;

732         /* Discard the message if sequence number was already there, is
733          * too far in the future, already in the queue or if we received
734          * a FINISHED before the SERVER_HELLO, which then must be a stale
735          * retransmit.
736          */
737         if (msg_hdr->seq <= s->d1->handshake_read_seq ||
738                 msg_hdr->seq > s->d1->handshake_read_seq + 10 || item != NULL ||
739                 (s->d1->handshake_read_seq == 0 && msg_hdr->type == SSL3_MT_FINI
740                 {
741                 unsigned char devnull [256];

743                 while (frag_len)
744                         {
745                         i = s->method->ssl_read_bytes(s,SSL3_RT_HANDSHAKE,
746                                 devnull,
747                                 frag_len>sizeof(devnull)?sizeof(devnull):frag_le
748                         if (i<=0) goto err;
749                         frag_len -= i;
750                         }
751                 }
752         else
753                 {
754                 if (frag_len != msg_hdr->msg_len)
755                         return dtls1_reassemble_fragment(s, msg_hdr, ok);

757                 if (frag_len > dtls1_max_handshake_message_len(s))
758                         goto err;

760                 frag = dtls1_hm_fragment_new(frag_len, 0);
761                 if ( frag == NULL)
762                         goto err;

764                 memcpy(&(frag->msg_header), msg_hdr, sizeof(*msg_hdr));

766                 if (frag_len)
767                         {
768                         /* read the body of the fragment (header has already bee
769                         i = s->method->ssl_read_bytes(s,SSL3_RT_HANDSHAKE,
770                                 frag->fragment,frag_len,0);
771                         if ((unsigned long)i!=frag_len)
772                                 i=-1;
773                                 i = -1;
773                         if (i<=0)
774                                 goto err;
775                         }

777                 item = pitem_new(seq64be, frag);
778                 if ( item == NULL)
779                         goto err;

781                 item = pqueue_insert(s->d1->buffered_messages, item);
782                 /* pqueue_insert fails iff a duplicate item is inserted.
783                  * However, |item| cannot be a duplicate. If it were,
784                  * |pqueue_find|, above, would have returned it. Then, either
785                  * |frag_len| != |msg_hdr->msg_len| in which case |item| is set
```

```
786                  * to NULL and it will have been processed with
787                  * |dtls1_reassemble_fragment|, above, or the record will have
788                  * been discarded. */
789                 OPENSSL_assert(item != NULL);
790                 }

792         return DTLS1_HM_FRAGMENT_RETRY;

794 err:
795         if (frag != NULL && item == NULL) dtls1_hm_fragment_free(frag);
796         *ok = 0;
797         return i;
798         }


801 static long
802 dtls1_get_message_fragment(SSL *s, int st1, int stn, long max, int *ok)
803         {
804         unsigned char wire[DTLS1_HM_HEADER_LENGTH];
805         unsigned long len, frag_off, frag_len;
806         int i,al;
807         struct hm_header_st msg_hdr;

809         redo:
810         /* see if we have the required fragment already */
811         if ((frag_len = dtls1_retrieve_buffered_fragment(s,max,ok)) || *ok)
812                 {
813                 if (*ok)        s->init_num = frag_len;
814                 return frag_len;
815                 }

817         /* read handshake message header */
818         i=s->method->ssl_read_bytes(s,SSL3_RT_HANDSHAKE,wire,
819                 DTLS1_HM_HEADER_LENGTH, 0);
820         if (i <= 0)     /* nbio, or an error */
821                 {
822                 s->rwstate=SSL_READING;
823                 *ok = 0;
824                 return i;
825                 }
826         /* Handshake fails if message header is incomplete */
827         if (i != DTLS1_HM_HEADER_LENGTH)
828                 {
829                 al=SSL_AD_UNEXPECTED_MESSAGE;
830                 SSLerr(SSL_F_DTLS1_GET_MESSAGE_FRAGMENT,SSL_R_UNEXPECTED_MESSAGE
831                 goto f_err;
832                 }

834         /* parse the message fragment header */
835         dtls1_get_message_header(wire, &msg_hdr);

837         /*
838          * if this is a future (or stale) message it gets buffered
839          * (or dropped)--no further processing at this time
840          * While listening, we accept seq 1 (ClientHello with cookie)
841          * although we're still expecting seq 0 (ClientHello)
842          */
843         if (msg_hdr.seq != s->d1->handshake_read_seq && !(s->d1->listen && msg_h
844                 return dtls1_process_out_of_seq_message(s, &msg_hdr, ok);

846         len = msg_hdr.msg_len;
847         frag_off = msg_hdr.frag_off;
848         frag_len = msg_hdr.frag_len;

850         if (frag_len && frag_len < len)
851                 return dtls1_reassemble_fragment(s, &msg_hdr, ok);
```



```
853         if (!s->server && s->d1->r_msg_hdr.frag_off == 0 &&
854                 wire[0] == SSL3_MT_HELLO_REQUEST)
855                 {
856                 /* The server may always send 'Hello Request' messages --
857                  * we are doing a handshake anyway now, so ignore them
858                  * if their format is correct. Does not count for
859                  * 'Finished' MAC. */
860                 if (wire[1] == 0 && wire[2] == 0 && wire[3] == 0)
861                         {
862                         if (s->msg_callback)
863                                 s->msg_callback(0, s->version, SSL3_RT_HANDSHAKE
864                                         wire, DTLS1_HM_HEADER_LENGTH, s,
865                                         s->msg_callback_arg);

867                         s->init_num = 0;
868                         goto redo;
869                         }
870                 else /* Incorrectly formated Hello request */
871                         {
872                         al=SSL_AD_UNEXPECTED_MESSAGE;
873                         SSLerr(SSL_F_DTLS1_GET_MESSAGE_FRAGMENT,SSL_R_UNEXPECTED
874                         goto f_err;
875                         }
876                 }

878         if ((al=dtls1_preprocess_fragment(s,&msg_hdr,max)))
879                 goto f_err;

881         /* XDTLS:  ressurect this when restart is in place */
882         s->state=stn;

884         if ( frag_len > 0)
885                 {
886                 unsigned char *p=(unsigned char *)s->init_buf->data+DTLS1_HM_HEA

888                 i=s->method->ssl_read_bytes(s,SSL3_RT_HANDSHAKE,
889                         &p[frag_off],frag_len,0);
890                 /* XDTLS:  fix this--message fragments cannot span multiple pack
891                 if (i <= 0)
892                         {
893                         s->rwstate=SSL_READING;
894                         *ok = 0;
895                         return i;
896                         }
897                 }
898         else
899                 i = 0;

901         /* XDTLS:  an incorrectly formatted fragment should cause the
902          * handshake to fail */
903         if (i != (int)frag_len)
904                 {
905                 al=SSL3_AD_ILLEGAL_PARAMETER;
906                 SSLerr(SSL_F_DTLS1_GET_MESSAGE_FRAGMENT,SSL3_AD_ILLEGAL_PARAMETE
907                 goto f_err;
908                 }

910         *ok = 1;

912         /* Note that s->init_num is *not* used as current offset in
913          * s->init_buf->data, but as a counter summing up fragments'
914          * lengths: as soon as they sum up to handshake packet
915          * length, we assume we have got all the fragments. */
916         s->init_num = frag_len;
917         return frag_len;
```

```
 919 f_err:
 920         ssl3_send_alert(s,SSL3_AL_FATAL,al);
 921         s->init_num = 0;

 923         *ok=0;
 924         return(-1);
 925         }

 927 int dtls1_send_finished(SSL *s, int a, int b, const char *sender, int slen)
 928         {
 929         unsigned char *p,*d;
 930         int i;
 931         unsigned long l;

 933         if (s->state == a)
 934                 {
 935                 d=(unsigned char *)s->init_buf->data;
 936                 p= &(d[DTLS1_HM_HEADER_LENGTH]);

 938                 i=s->method->ssl3_enc->final_finish_mac(s,
 939                         sender,slen,s->s3->tmp.finish_md);
 940                 s->s3->tmp.finish_md_len = i;
 941                 memcpy(p, s->s3->tmp.finish_md, i);
 942                 p+=i;
 943                 l=i;

 945         /* Copy the finished so we can use it for
 946          * renegotiation checks
 947          */
 948         if(s->type == SSL_ST_CONNECT)
 949                 {
 950                 OPENSSL_assert(i <= EVP_MAX_MD_SIZE);
 951                 memcpy(s->s3->previous_client_finished,
 952                        s->s3->tmp.finish_md, i);
 953                 s->s3->previous_client_finished_len=i;
 954                 }
 955         else
 956                 {
 957                 OPENSSL_assert(i <= EVP_MAX_MD_SIZE);
 958                 memcpy(s->s3->previous_server_finished,
 959                        s->s3->tmp.finish_md, i);
 960                 s->s3->previous_server_finished_len=i;
 961                 }

 963 #ifdef OPENSSL_SYS_WIN16
 964                 /* MSVC 1.5 does not clear the top bytes of the word unless
 965                  * I do this.
 966                  */
 967                 l&=0xffff;
 968 #endif

 970                 d = dtls1_set_message_header(s, d, SSL3_MT_FINISHED, l, 0, l);
 971                 s->init_num=(int)l+DTLS1_HM_HEADER_LENGTH;
 972                 s->init_off=0;

 974                 /* buffer the message to handle re-xmits */
 975                 dtls1_buffer_message(s, 0);

 977                 s->state=b;
 978                 }

 980         /* SSL3_ST_SEND_xxxxxx_HELLO_B */
 981         return(dtls1_do_write(s,SSL3_RT_HANDSHAKE));
 982         }
```



```
 984 /* for these 2 messages, we need to
 985  * ssl->enc_read_ctx                    re-init
 986  * ssl->s3->read_sequence               zero
 987  * ssl->s3->read_mac_secret             re-init
 988  * ssl->session->read_sym_enc           assign
 989  * ssl->session->read_compression       assign
 990  * ssl->session->read_hash              assign
 991  */
 992 int dtls1_send_change_cipher_spec(SSL *s, int a, int b)
 993         {
 994         unsigned char *p;

 996         if (s->state == a)
 997                 {
 998                 p=(unsigned char *)s->init_buf->data;
 999                 *p++=SSL3_MT_CCS;
1000                 s->d1->handshake_write_seq = s->d1->next_handshake_write_seq;
1001                 s->init_num=DTLS1_CCS_HEADER_LENGTH;

1003                 if (s->version == DTLS1_BAD_VER) {
1004                         s->d1->next_handshake_write_seq++;
1005                         s2n(s->d1->handshake_write_seq,p);
1006                         s->init_num+=2;
1007                 }

1009                 s->init_off=0;

1011                 dtls1_set_message_header_int(s, SSL3_MT_CCS, 0,
1012                         s->d1->handshake_write_seq, 0, 0);

1014                 /* buffer the message to handle re-xmits */
1015                 dtls1_buffer_message(s, 1);

1017                 s->state=b;
1018                 }

1020         /* SSL3_ST_CW_CHANGE_B */
1021         return(dtls1_do_write(s,SSL3_RT_CHANGE_CIPHER_SPEC));
1022         }

1024 static int dtls1_add_cert_to_buf(BUF_MEM *buf, unsigned long *l, X509 *x)
1025         {
1026         int n;
1027         unsigned char *p;

1029         n=i2d_X509(x,NULL);
1030         if (!BUF_MEM_grow_clean(buf,(int)(n+(*l)+3)))
1031                 {
1032                 SSLerr(SSL_F_DTLS1_ADD_CERT_TO_BUF,ERR_R_BUF_LIB);
1033                 return 0;
1034                 }
1035         p=(unsigned char *)&(buf->data[*l]);
1036         l2n3(n,p);
1037         i2d_X509(x,&p);
1038         *l+=n+3;

1040         return 1;
1041         }
1042 unsigned long dtls1_output_cert_chain(SSL *s, X509 *x)
1043         {
1044         unsigned char *p;
1045         int i;
1046         unsigned long l= 3 + DTLS1_HM_HEADER_LENGTH;
1047         BUF_MEM *buf;

1049         /* TLSv1 sends a chain with nothing in it, instead of an alert */
```

```
1050         buf=s->init_buf;
1051         if (!BUF_MEM_grow_clean(buf,10))
1052                 {
1053                 SSLerr(SSL_F_DTLS1_OUTPUT_CERT_CHAIN,ERR_R_BUF_LIB);
1054                 return(0);
1055                 }
1056         if (x != NULL)
1057                 {
1058                 X509_STORE_CTX xs_ctx;

1060                 if (!X509_STORE_CTX_init(&xs_ctx,s->ctx->cert_store,x,NULL))
1061                         {
1062                         SSLerr(SSL_F_DTLS1_OUTPUT_CERT_CHAIN,ERR_R_X509_LIB);
1063                         return(0);
1064                         }

1066                 X509_verify_cert(&xs_ctx);
1067                 /* Don't leave errors in the queue */
1068                 ERR_clear_error();
1069                 for (i=0; i < sk_X509_num(xs_ctx.chain); i++)
1070                         {
1071                         x = sk_X509_value(xs_ctx.chain, i);

1073                         if (!dtls1_add_cert_to_buf(buf, &l, x))
1074                                 {
1075                                 X509_STORE_CTX_cleanup(&xs_ctx);
1076                                 return 0;
1077                                 }
1078                         }
1079                 X509_STORE_CTX_cleanup(&xs_ctx);
1080                 }
1081         /* Thawte special :-) */
1082         for (i=0; i<sk_X509_num(s->ctx->extra_certs); i++)
1083                 {
1084                 x=sk_X509_value(s->ctx->extra_certs,i);
1085                 if (!dtls1_add_cert_to_buf(buf, &l, x))
1086                         return 0;
1087                 }

1089         l-= (3 + DTLS1_HM_HEADER_LENGTH);

1091         p=(unsigned char *)&(buf->data[DTLS1_HM_HEADER_LENGTH]);
1092         l2n3(l,p);
1093         l+=3;
1094         p=(unsigned char *)&(buf->data[0]);
1095         p = dtls1_set_message_header(s, p, SSL3_MT_CERTIFICATE, l, 0, l);

1097         l+=DTLS1_HM_HEADER_LENGTH;
1098         return(l);
1099         }

1101 int dtls1_read_failed(SSL *s, int code)
1102         {
1103         if ( code > 0)
1104                 {
1105                 fprintf( stderr, "invalid state reached %s:%d", __FILE__, __LINE
1106                 return 1;
1107                 }

1109         if (!dtls1_is_timer_expired(s))
1110                 {
1111                 /* not a timeout, none of our business,
1112                    let higher layers handle this.  in fact it's probably an erro
1113                 return code;
1114                 }
```



```
1116 #ifndef OPENSSL_NO_HEARTBEATS
1117         if (!SSL_in_init(s) && !s->tlsext_hb_pending)  /* done, no need to send
1118 #else
1119         if (!SSL_in_init(s))  /* done, no need to send a retransmit */
1120 #endif
1121                 {
1122                 BIO_set_flags(SSL_get_rbio(s), BIO_FLAGS_READ);
1123                 return code;
1124                 }

1126 #if 0 /* for now, each alert contains only one record number */
1127         item = pqueue_peek(state->rcvd_records);
1128         if ( item )
1129                 {
1130                 /* send an alert immediately for all the missing records */
1131                 }
1132         else
1133 #endif

1135 #if 0  /* no more alert sending, just retransmit the last set of messages */
1136         if ( state->timeout.read_timeouts >= DTLS1_TMO_READ_COUNT)
1137                 ssl3_send_alert(s,SSL3_AL_WARNING,
1138                         DTLS1_AD_MISSING_HANDSHAKE_MESSAGE);
1139 #endif

1141         return dtls1_handle_timeout(s);
1142         }

1144 int
1145 dtls1_get_queue_priority(unsigned short seq, int is_ccs)
1146         {
1147         /* The index of the retransmission queue actually is the message sequenc
1148          * since the queue only contains messages of a single handshake. However
1149          * ChangeCipherSpec has no message sequence number and so using only the
1150          * will result in the CCS and Finished having the same index. To prevent
1151          * the sequence number is multiplied by 2. In case of a CCS 1 is subtrac
1152          * This does not only differ CSS and Finished, it also maintains the ord
1153          * index (important for priority queues) and fits in the unsigned short
1154          */
1155         return seq * 2 - is_ccs;
1156         }

1158 int
1159 dtls1_retransmit_buffered_messages(SSL *s)
1160         {
1161         pqueue sent = s->d1->sent_messages;
1162         piterator iter;
1163         pitem *item;
1164         hm_fragment *frag;
1165         int found = 0;

1167         iter = pqueue_iterator(sent);

1169         for ( item = pqueue_next(&iter); item != NULL; item = pqueue_next(&iter)
1170                 {
1171                 frag = (hm_fragment *)item->data;
1172                         if ( dtls1_retransmit_message(s,
1173                                 (unsigned short)dtls1_get_queue_priority(frag->m
1174                                 0, &found) <= 0 && found)
1175                         {
1176                         fprintf(stderr, "dtls1_retransmit_message() failed\n");
1177                         return -1;
1178                         }
1179                 }

1181         return 1;
```

```
1182         }

1184 int
1185 dtls1_buffer_message(SSL *s, int is_ccs)
1186         {
1187         pitem *item;
1188         hm_fragment *frag;
1189         unsigned char seq64be[8];

1191         /* this function is called immediately after a message has
1192          * been serialized */
1193         OPENSSL_assert(s->init_off == 0);

1195         frag = dtls1_hm_fragment_new(s->init_num, 0);
1196         if (!frag)
1197                 return 0;

1199         memcpy(frag->fragment, s->init_buf->data, s->init_num);

1201         if ( is_ccs)
1202                 {
1203                 OPENSSL_assert(s->d1->w_msg_hdr.msg_len +
1204                                 ((s->version==DTLS1_VERSION)?DTLS1_CCS_HEADER_LEN
1205                 }
1206         else
1207                 {
1208                 OPENSSL_assert(s->d1->w_msg_hdr.msg_len +
1209                         DTLS1_HM_HEADER_LENGTH == (unsigned int)s->init_num);
1210                 }

1212         frag->msg_header.msg_len = s->d1->w_msg_hdr.msg_len;
1213         frag->msg_header.seq = s->d1->w_msg_hdr.seq;
1214         frag->msg_header.type = s->d1->w_msg_hdr.type;
1215         frag->msg_header.frag_off = 0;
1216         frag->msg_header.frag_len = s->d1->w_msg_hdr.msg_len;
1217         frag->msg_header.is_ccs = is_ccs;

1219         /* save current state*/
1220         frag->msg_header.saved_retransmit_state.enc_write_ctx = s->enc_write_ctx
1221         frag->msg_header.saved_retransmit_state.write_hash = s->write_hash;
1222         frag->msg_header.saved_retransmit_state.compress = s->compress;
1223         frag->msg_header.saved_retransmit_state.session = s->session;
1224         frag->msg_header.saved_retransmit_state.epoch = s->d1->w_epoch;

1226         memset(seq64be,0,sizeof(seq64be));
1227         seq64be[6] = (unsigned char)(dtls1_get_queue_priority(frag->msg_header.s
1228
1229         seq64be[7] = (unsigned char)(dtls1_get_queue_priority(frag->msg_header.s
1230


1232         item = pitem_new(seq64be, frag);
1233         if ( item == NULL)
1234                 {
1235                 dtls1_hm_fragment_free(frag);
1236                 return 0;
1237                 }

1239 #if 0
1240         fprintf( stderr, "buffered messge: \ttype = %xx\n", msg_buf->type);
1241         fprintf( stderr, "\t\t\t\t\tlen = %d\n", msg_buf->len);
1242         fprintf( stderr, "\t\t\t\t\tseq_num = %d\n", msg_buf->seq_num);
1243 #endif

1245         pqueue_insert(s->d1->sent_messages, item);
1246         return 1;
1247         }
```



```
1249 int
1250 dtls1_retransmit_message(SSL *s, unsigned short seq, unsigned long frag_off,
1251         int *found)
1252         {
1253         int ret;
1254         /* XDTLS: for now assuming that read/writes are blocking */
1255         pitem *item;
1256         hm_fragment *frag ;
1257         unsigned long header_length;
1258         unsigned char seq64be[8];
1259         struct dtls1_retransmit_state saved_state;
1260         unsigned char save_write_sequence[8];

1262         /*
1263           OPENSSL_assert(s->init_num == 0);
1264           OPENSSL_assert(s->init_off == 0);
1265          */

1267         /* XDTLS:  the requested message ought to be found, otherwise error */
1268         memset(seq64be,0,sizeof(seq64be));
1269         seq64be[6] = (unsigned char)(seq>>8);
1270         seq64be[7] = (unsigned char)seq;

1272         item = pqueue_find(s->d1->sent_messages, seq64be);
1273         if ( item == NULL)
1274                 {
1275                 fprintf(stderr, "retransmit:  message %d non-existant\n", seq);
1276                 *found = 0;
1277                 return 0;
1278                 }

1280         *found = 1;
1281         frag = (hm_fragment *)item->data;

1283         if ( frag->msg_header.is_ccs)
1284                 header_length = DTLS1_CCS_HEADER_LENGTH;
1285         else
1286                 header_length = DTLS1_HM_HEADER_LENGTH;

1288         memcpy(s->init_buf->data, frag->fragment,
1289                 frag->msg_header.msg_len + header_length);
1290                 s->init_num = frag->msg_header.msg_len + header_length;

1292         dtls1_set_message_header_int(s, frag->msg_header.type,
1293                 frag->msg_header.msg_len, frag->msg_header.seq, 0,
1294                 frag->msg_header.frag_len);

1296         /* save current state */
1297         saved_state.enc_write_ctx = s->enc_write_ctx;
1298         saved_state.write_hash = s->write_hash;
1299         saved_state.compress = s->compress;
1300         saved_state.session = s->session;
1301         saved_state.epoch = s->d1->w_epoch;
1302         saved_state.epoch = s->d1->w_epoch;

1304         s->d1->retransmitting = 1;

1306         /* restore state in which the message was originally sent */
1307         s->enc_write_ctx = frag->msg_header.saved_retransmit_state.enc_write_ctx
1308         s->write_hash = frag->msg_header.saved_retransmit_state.write_hash;
1309         s->compress = frag->msg_header.saved_retransmit_state.compress;
1310         s->session = frag->msg_header.saved_retransmit_state.session;
1311         s->d1->w_epoch = frag->msg_header.saved_retransmit_state.epoch;

1313         if (frag->msg_header.saved_retransmit_state.epoch == saved_state.epoch -
```

```
1314         {
1315                 memcpy(save_write_sequence, s->s3->write_sequence, sizeof(s->s3-
1316                 memcpy(s->s3->write_sequence, s->d1->last_write_sequence, sizeof
1317         }

1319         ret = dtls1_do_write(s, frag->msg_header.is_ccs ?
1320                                                 SSL3_RT_CHANGE_CIPHER_SPEC : SS

1322         /* restore current state */
1323         s->enc_write_ctx = saved_state.enc_write_ctx;
1324         s->write_hash = saved_state.write_hash;
1325         s->compress = saved_state.compress;
1326         s->session = saved_state.session;
1327         s->d1->w_epoch = saved_state.epoch;

1329         if (frag->msg_header.saved_retransmit_state.epoch == saved_state.epoch -
1330         {
1331                 memcpy(s->d1->last_write_sequence, s->s3->write_sequence, sizeof
1332                 memcpy(s->s3->write_sequence, save_write_sequence, sizeof(s->s3-
1333         }

1335         s->d1->retransmitting = 0;

1337         (void)BIO_flush(SSL_get_wbio(s));
1338         return ret;
1339         }

1341 /* call this function when the buffered messages are no longer needed */
1342 void
1343 dtls1_clear_record_buffer(SSL *s)
1344         {
1345         pitem *item;

1347         for(item = pqueue_pop(s->d1->sent_messages);
1348                 item != NULL; item = pqueue_pop(s->d1->sent_messages))
1349                 {
1350                 dtls1_hm_fragment_free((hm_fragment *)item->data);
1351                 pitem_free(item);
1352                 }
1353         }


1356 unsigned char *
1357 dtls1_set_message_header(SSL *s, unsigned char *p, unsigned char mt,
1358                         unsigned long len, unsigned long frag_off, unsigned long
1359         {
1360         /* Don't change sequence numbers while listening */
1361         if (frag_off == 0 && !s->d1->listen)
1362                 {
1363                 s->d1->handshake_write_seq = s->d1->next_handshake_write_seq;
1364                 s->d1->next_handshake_write_seq++;
1365                 }

1367         dtls1_set_message_header_int(s, mt, len, s->d1->handshake_write_seq,
1368                 frag_off, frag_len);

1370         return p += DTLS1_HM_HEADER_LENGTH;
1371         }


1374 /* don't actually do the writing, wait till the MTU has been retrieved */
1375 static void
1376 dtls1_set_message_header_int(SSL *s, unsigned char mt,
1377                              unsigned long len, unsigned short seq_num, unsigned
1378                              unsigned long frag_len)
1379         {
```



```
1380         struct hm_header_st *msg_hdr = &s->d1->w_msg_hdr;

1382         msg_hdr->type = mt;
1383         msg_hdr->msg_len = len;
1384         msg_hdr->seq = seq_num;
1385         msg_hdr->frag_off = frag_off;
1386         msg_hdr->frag_len = frag_len;
1387         }

1389 static void
1390 dtls1_fix_message_header(SSL *s, unsigned long frag_off,
1391                          unsigned long frag_len)
1392         {
1393         struct hm_header_st *msg_hdr = &s->d1->w_msg_hdr;

1395         msg_hdr->frag_off = frag_off;
1396         msg_hdr->frag_len = frag_len;
1397         }

1399 static unsigned char *
1400 dtls1_write_message_header(SSL *s, unsigned char *p)
1401         {
1402         struct hm_header_st *msg_hdr = &s->d1->w_msg_hdr;

1404         *p++ = msg_hdr->type;
1405         l2n3(msg_hdr->msg_len, p);

1407         s2n(msg_hdr->seq, p);
1408         l2n3(msg_hdr->frag_off, p);
1409         l2n3(msg_hdr->frag_len, p);

1411         return p;
1412         }

1414 unsigned int
1415 dtls1_min_mtu(void)
1416         {
1417         return (g_probable_mtu[(sizeof(g_probable_mtu) /
1418                 sizeof(g_probable_mtu[0])) - 1]);
1419         }

1421 static unsigned int
1422 dtls1_guess_mtu(unsigned int curr_mtu)
1423         {
1424         unsigned int i;

1426         if ( curr_mtu == 0 )
1427                 return g_probable_mtu[0] ;

1429         for ( i = 0; i < sizeof(g_probable_mtu)/sizeof(g_probable_mtu[0]); i++)
1430                 if ( curr_mtu > g_probable_mtu[i])
1431                         return g_probable_mtu[i];

1433         return curr_mtu;
1434         }

1436 void
1437 dtls1_get_message_header(unsigned char *data, struct hm_header_st *msg_hdr)
1438         {
1439         memset(msg_hdr, 0x00, sizeof(struct hm_header_st));
1440         msg_hdr->type = *(data++);
1441         n2l3(data, msg_hdr->msg_len);

1443         n2s(data, msg_hdr->seq);
1444         n2l3(data, msg_hdr->frag_off);
1445         n2l3(data, msg_hdr->frag_len);
```

```
1446         }

1448 void
1449 dtls1_get_ccs_header(unsigned char *data, struct ccs_header_st *ccs_hdr)
1450         {
1451         memset(ccs_hdr, 0x00, sizeof(struct ccs_header_st));

1453         ccs_hdr->type = *(data++);
1454         }

1456 int dtls1_shutdown(SSL *s)
1457         {
1458         int ret;
1459 #ifndef OPENSSL_NO_SCTP
1460         if (BIO_dgram_is_sctp(SSL_get_wbio(s)) &&
1461             !(s->shutdown & SSL_SENT_SHUTDOWN))
1462                 {
1463                 ret = BIO_dgram_sctp_wait_for_dry(SSL_get_wbio(s));
1464                 if (ret < 0) return -1;

1466                 if (ret == 0)
1467                         BIO_ctrl(SSL_get_wbio(s), BIO_CTRL_DGRAM_SCTP_SAVE_SHUTD
1468                 }
1469 #endif
1470         ret = ssl3_shutdown(s);
1471 #ifndef OPENSSL_NO_SCTP
1472         BIO_ctrl(SSL_get_wbio(s), BIO_CTRL_DGRAM_SCTP_SAVE_SHUTDOWN, 0, NULL);
1473 #endif
1474         return ret;
1475         }

1477 #ifndef OPENSSL_NO_HEARTBEATS
1478 int
1479 dtls1_process_heartbeat(SSL *s)
1480         {
1481         unsigned char *p = &s->s3->rrec.data[0], *pl;
1482         unsigned short hbtype;
1483         unsigned int payload;
1484         unsigned int padding = 16; /* Use minimum padding */

1486         if (s->msg_callback)
1487                 s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
1488                         &s->s3->rrec.data[0], s->s3->rrec.length,
1489                         s, s->msg_callback_arg);

1491         /* Read type and payload length first */
1492         if (1 + 2 + 16 > s->s3->rrec.length)
1493                 return 0; /* silently discard */
1494         hbtype = *p++;
1495         n2s(p, payload);
1496         if (1 + 2 + payload + 16 > s->s3->rrec.length)
1497                 return 0; /* silently discard per RFC 6520 sec. 4 */
1498         pl = p;

1500         if (hbtype == TLS1_HB_REQUEST)
1501                 {
1502                 unsigned char *buffer, *bp;
1503                 unsigned int write_length = 1 /* heartbeat type */ +
1504                                             2 /* heartbeat length */ +
1505                                             payload + padding;
1506                 int r;

1508                 if (write_length > SSL3_RT_MAX_PLAIN_LENGTH)
1509                         return 0;

1511                 /* Allocate memory for the response, size is 1 byte
```



```
1512                  * message type, plus 2 bytes payload length, plus
1513                  * payload, plus padding
1514                  */
1515                 buffer = OPENSSL_malloc(write_length);
1516                 bp = buffer;

1518                 /* Enter response type, length and copy payload */
1519                 *bp++ = TLS1_HB_RESPONSE;
1520                 s2n(payload, bp);
1521                 memcpy(bp, pl, payload);
1522                 bp += payload;
1523                 /* Random padding */
1524                 RAND_pseudo_bytes(bp, padding);

1526                 r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, write_length

1528                 if (r >= 0 && s->msg_callback)
1529                         s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
1530                                 buffer, write_length,
1531                                 s, s->msg_callback_arg);

1533                 OPENSSL_free(buffer);

1535                 if (r < 0)
1536                         return r;
1537                 }
1538         else if (hbtype == TLS1_HB_RESPONSE)
1539                 {
1540                 unsigned int seq;

1542                 /* We only send sequence numbers (2 bytes unsigned int),
1543                  * and 16 random bytes, so we just try to read the
1544                  * sequence number */
1545                 n2s(pl, seq);

1547                 if (payload == 18 && seq == s->tlsext_hb_seq)
1548                         {
1549                         dtls1_stop_timer(s);
1550                         s->tlsext_hb_seq++;
1551                         s->tlsext_hb_pending = 0;
1552                         }
1553                 }

1555         return 0;
1556         }

1558 int
1559 dtls1_heartbeat(SSL *s)
1560         {
1561         unsigned char *buf, *p;
1562         int ret;
1563         unsigned int payload = 18; /* Sequence number + random bytes */
1564         unsigned int padding = 16; /* Use minimum padding */

1566         /* Only send if peer supports and accepts HB requests... */
1567         if (!(s->tlsext_heartbeat & SSL_TLSEXT_HB_ENABLED) ||
1568             s->tlsext_heartbeat & SSL_TLSEXT_HB_DONT_SEND_REQUESTS)
1569                 {
1570                 SSLerr(SSL_F_DTLS1_HEARTBEAT,SSL_R_TLS_HEARTBEAT_PEER_DOESNT_ACC
1571                 return -1;
1572                 }

1574         /* ...and there is none in flight yet... */
1575         if (s->tlsext_hb_pending)
1576                 {
1577                 SSLerr(SSL_F_DTLS1_HEARTBEAT,SSL_R_TLS_HEARTBEAT_PENDING);
```

```
1578                return -1;
1579                }

1581        /* ...and no handshake in progress. */
1582        if (SSL_in_init(s) || s->in_handshake)
1583                {
1584                SSLerr(SSL_F_DTLS1_HEARTBEAT,SSL_R_UNEXPECTED_MESSAGE);
1585                return -1;
1586                }

1588        /* Check if padding is too long, payload and padding
1589         * must not exceed 2^14 - 3 = 16381 bytes in total.
1590         */
1591        OPENSSL_assert(payload + padding <= 16381);

1593        /* Create HeartBeat message, we just use a sequence number
1594         * as payload to distuingish different messages and add
1595         * some random stuff.
1596         *  - Message Type, 1 byte
1597         *  - Payload Length, 2 bytes (unsigned int)
1598         *  - Payload, the sequence number (2 bytes uint)
1599         *  - Payload, random bytes (16 bytes uint)
1600         *  - Padding
1601         */
1602        buf = OPENSSL_malloc(1 + 2 + payload + padding);
1603        p = buf;
1604        /* Message Type */
1605        *p++ = TLS1_HB_REQUEST;
1606        /* Payload length (18 bytes here) */
1607        s2n(payload, p);
1608        /* Sequence number */
1609        s2n(s->tlsext_hb_seq, p);
1610        /* 16 random bytes */
1611        RAND_pseudo_bytes(p, 16);
1612        p += 16;
1613        /* Random padding */
1614        RAND_pseudo_bytes(p, padding);

1616        ret = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buf, 3 + payload + padding
1617        if (ret >= 0)
1618                {
1619                if (s->msg_callback)
1620                        s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
1621                                buf, 3 + payload + padding,
1622                                s, s->msg_callback_arg);

1624                dtls1_start_timer(s);
1625                s->tlsext_hb_pending = 1;
1626                }

1628        OPENSSL_free(buf);

1630        return ret;
1631        }
1632 #endif
```

```
**********************************************************
   13303 Wed Aug 13 19:33:05 2014
new/usr/src/lib/openssl/libsunw_ssl/d1_srtp.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
/* ssl/t1_lib.c */
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
/* ====================================================================
 * Copyright (c) 1998-2006 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
```



```
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ''AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */
/*
  DTLS code by Eric Rescorla <ekr@rtfm.com>

  Copyright (C) 2006, Network Resonance, Inc.
  Copyright (C) 2011, RTFM, Inc.
*/

#include <stdio.h>
#include <openssl/objects.h>
#include "ssl_locl.h"

#ifndef OPENSSL_NO_SRTP

#include <openssl/srtp.h>
#include "srtp.h"
```

```
127 static SRTP_PROTECTION_PROFILE srtp_known_profiles[]=
128     {
129     {
130     "SRTP_AES128_CM_SHA1_80",
131     SRTP_AES128_CM_SHA1_80,
132     },
133     {
134     "SRTP_AES128_CM_SHA1_32",
135     SRTP_AES128_CM_SHA1_32,
136     },
137 #if 0
138     {
139     "SRTP_NULL_SHA1_80",
140     SRTP_NULL_SHA1_80,
141     },
142     {
143     "SRTP_NULL_SHA1_32",
144     SRTP_NULL_SHA1_32,
145     },
146 #endif
147     {0}
148     };

150 static int find_profile_by_name(char *profile_name,
151                                 SRTP_PROTECTION_PROFILE **pptr,unsigned len)
152         {
153         SRTP_PROTECTION_PROFILE *p;

155         p=srtp_known_profiles;
156         while(p->name)
157                 {
158                 if((len == strlen(p->name)) && !strncmp(p->name,profile_name,
159                                                         len))
160                         {
161                         *pptr=p;
162                         return 0;
163                         }

165                 p++;
166                 }

168         return 1;
169         }

171 static int find_profile_by_num(unsigned profile_num,
172                                 SRTP_PROTECTION_PROFILE **pptr)
173         {
174         SRTP_PROTECTION_PROFILE *p;

176         p=srtp_known_profiles;
177         while(p->name)
178                 {
179                 if(p->id == profile_num)
180                         {
181                         *pptr=p;
182                         return 0;
183                         }
184                 p++;
185                 }

187         return 1;
188         }

190 static int ssl_ctx_make_profiles(const char *profiles_string,STACK_OF(SRTP_PROTE
191         {
192         STACK_OF(SRTP_PROTECTION_PROFILE) *profiles;
```



```
194         char *col;
195         char *ptr=(char *)profiles_string;

197         SRTP_PROTECTION_PROFILE *p;

199         if(!(profiles=sk_SRTP_PROTECTION_PROFILE_new_null()))
200                 {
201                 SSLerr(SSL_F_SSL_CTX_MAKE_PROFILES, SSL_R_SRTP_COULD_NOT_ALLOCAT
202                 return 1;
203                 }

205         do
206                 {
207                 col=strchr(ptr,':');

209                 if(!find_profile_by_name(ptr,&p,
210                                          col ? col-ptr : (int)strlen(ptr)))
211                         {
212                         sk_SRTP_PROTECTION_PROFILE_push(profiles,p);
213                         }
214                 else
215                         {
216                         SSLerr(SSL_F_SSL_CTX_MAKE_PROFILES,SSL_R_SRTP_UNKNOWN_PR
217                         return 1;
218                         }

220                 if(col) ptr=col+1;
221                 } while (col);

223         *out=profiles;

225         return 0;
226         }

228 int SSL_CTX_set_tlsext_use_srtp(SSL_CTX *ctx,const char *profiles)
229         {
230         return ssl_ctx_make_profiles(profiles,&ctx->srtp_profiles);
231         }

233 int SSL_set_tlsext_use_srtp(SSL *s,const char *profiles)
234         {
235         return ssl_ctx_make_profiles(profiles,&s->srtp_profiles);
236         }


239 STACK_OF(SRTP_PROTECTION_PROFILE) *SSL_get_srtp_profiles(SSL *s)
240         {
241         if(s != NULL)
242                 {
243                 if(s->srtp_profiles != NULL)
244                         {
245                         return s->srtp_profiles;
246                         }
247                 else if((s->ctx != NULL) &&
248                         (s->ctx->srtp_profiles != NULL))
249                         {
250                         return s->ctx->srtp_profiles;
251                         }
252                 }

254         return NULL;
255         }

257 SRTP_PROTECTION_PROFILE *SSL_get_selected_srtp_profile(SSL *s)
258         {
```

```
259         return s->srtp_profile;
260         }

262 /* Note: this function returns 0 length if there are no
263    profiles specified */
264 int ssl_add_clienthello_use_srtp_ext(SSL *s, unsigned char *p, int *len, int max
265         {
266         int ct=0;
267         int i;
268         STACK_OF(SRTP_PROTECTION_PROFILE) *clnt=0;
269         SRTP_PROTECTION_PROFILE *prof;

271         clnt=SSL_get_srtp_profiles(s);
272         ct=sk_SRTP_PROTECTION_PROFILE_num(clnt); /* -1 if clnt == 0 */

274         if(p)
275                 {
276                 if(ct==0)
277                         {
278                         SSLerr(SSL_F_SSL_ADD_CLIENTHELLO_USE_SRTP_EXT,SSL_R_EMPT
279                         return 1;
280                         }

282                 if((2 + ct*2 + 1) > maxlen)
283                         {
284                         SSLerr(SSL_F_SSL_ADD_CLIENTHELLO_USE_SRTP_EXT,SSL_R_SRTP
285                         return 1;
286                         }

288                 /* Add the length */
289                 s2n(ct * 2, p);
290                 for(i=0;i<ct;i++)
291                         {
292                         prof=sk_SRTP_PROTECTION_PROFILE_value(clnt,i);
293                         s2n(prof->id,p);
294                         }

296                 /* Add an empty use_mki value */
297                 *p++ = 0;
298                 }

300         *len=2 + ct*2 + 1;

302         return 0;
303         }


306 int ssl_parse_clienthello_use_srtp_ext(SSL *s, unsigned char *d, int len,int *al
307         {
308         SRTP_PROTECTION_PROFILE *cprof,*sprof;
309         STACK_OF(SRTP_PROTECTION_PROFILE) *clnt=0,*srvr;
310         int ct;
311         int mki_len;
312         int i,j;
313         int id;
314         int ret;

316          /* Length value + the MKI length */
317         if(len < 3)
318                 {
319                 SSLerr(SSL_F_SSL_PARSE_CLIENTHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_P
320                 *al=SSL_AD_DECODE_ERROR;
321                 return 1;
322                 }

324         /* Pull off the length of the cipher suite list */
```



```
325         n2s(d, ct);
326         len -= 2;

328         /* Check that it is even */
329         if(ct%2)
330                 {
331                 SSLerr(SSL_F_SSL_PARSE_CLIENTHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_P
332                 *al=SSL_AD_DECODE_ERROR;
333                 return 1;
334                 }

336         /* Check that lengths are consistent */
337         if(len < (ct + 1))
338                 {
339                 SSLerr(SSL_F_SSL_PARSE_CLIENTHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_P
340                 *al=SSL_AD_DECODE_ERROR;
341                 return 1;
342                 }


345         clnt=sk_SRTP_PROTECTION_PROFILE_new_null();

347         while(ct)
348                 {
349                 n2s(d,id);
350                 ct-=2;
351                 len-=2;

353                 if(!find_profile_by_num(id,&cprof))
354                         {
355                         sk_SRTP_PROTECTION_PROFILE_push(clnt,cprof);
356                         }
357                 else
358                         {
359                         ; /* Ignore */
360                         }
361                 }

363         /* Now extract the MKI value as a sanity check, but discard it for now *
364         mki_len = *d;
365         d++; len--;

367         if (mki_len != len)
368                 {
369                 SSLerr(SSL_F_SSL_PARSE_CLIENTHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_M
370                 *al=SSL_AD_DECODE_ERROR;
371                 return 1;
372                 }

374         srvr=SSL_get_srtp_profiles(s);

376         /* Pick our most preferred profile. If no profiles have been
377          configured then the outer loop doesn't run
378          (sk_SRTP_PROTECTION_PROFILE_num() = -1)
379          and so we just return without doing anything */
380         for(i=0;i<sk_SRTP_PROTECTION_PROFILE_num(srvr);i++)
381                 {
382                 sprof=sk_SRTP_PROTECTION_PROFILE_value(srvr,i);

384                 for(j=0;j<sk_SRTP_PROTECTION_PROFILE_num(clnt);j++)
385                         {
386                         cprof=sk_SRTP_PROTECTION_PROFILE_value(clnt,j);

388                         if(cprof->id==sprof->id)
389                                 {
390                                 s->srtp_profile=sprof;
```

```
391                                 *al=0;
392                                 ret=0;
393                                 goto done;
394                                 }
395                         }
396                 }

398         ret=0;

400 done:
401         if(clnt) sk_SRTP_PROTECTION_PROFILE_free(clnt);

403         return ret;
404         }

406 int ssl_add_serverhello_use_srtp_ext(SSL *s, unsigned char *p, int *len, int max
407         {
408         if(p)
409                 {
410                 if(maxlen < 5)
411                         {
412                         SSLerr(SSL_F_SSL_ADD_SERVERHELLO_USE_SRTP_EXT,SSL_R_SRTP
413                         return 1;
414                         }

416                 if(s->srtp_profile==0)
417                         {
418                         SSLerr(SSL_F_SSL_ADD_SERVERHELLO_USE_SRTP_EXT,SSL_R_USE_
419                         return 1;
420                         }
421                 s2n(2, p);
422                 s2n(s->srtp_profile->id,p);
423                 *p++ = 0;
424                 }
425         *len=5;

427         return 0;
428         }


431 int ssl_parse_serverhello_use_srtp_ext(SSL *s, unsigned char *d, int len,int *al
432         {
433         unsigned id;
434         int i;
435         int ct;

437         STACK_OF(SRTP_PROTECTION_PROFILE) *clnt;
438         SRTP_PROTECTION_PROFILE *prof;

440         if(len!=5)
441                 {
442                 SSLerr(SSL_F_SSL_PARSE_SERVERHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_P
443                 *al=SSL_AD_DECODE_ERROR;
444                 return 1;
445                 }

447         n2s(d, ct);
448         if(ct!=2)
449                 {
450                 SSLerr(SSL_F_SSL_PARSE_SERVERHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_P
451                 *al=SSL_AD_DECODE_ERROR;
452                 return 1;
453                 }

455         n2s(d,id);
456         if (*d)  /* Must be no MKI, since we never offer one */
```



```
457                 {
458                 SSLerr(SSL_F_SSL_PARSE_SERVERHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_M
459                 *al=SSL_AD_ILLEGAL_PARAMETER;
460                 return 1;
461                 }

463         clnt=SSL_get_srtp_profiles(s);

465         /* Throw an error if the server gave us an unsolicited extension */
466         if (clnt == NULL)
467                 {
468                 SSLerr(SSL_F_SSL_PARSE_SERVERHELLO_USE_SRTP_EXT,SSL_R_NO_SRTP_PR
469                 *al=SSL_AD_DECODE_ERROR;
470                 return 1;
471                 }

473         /* Check to see if the server gave us something we support
474            (and presumably offered)
475         */
476         for(i=0;i<sk_SRTP_PROTECTION_PROFILE_num(clnt);i++)
477                 {
478                 prof=sk_SRTP_PROTECTION_PROFILE_value(clnt,i);

480                 if(prof->id == id)
481                         {
482                         s->srtp_profile=prof;
483                         *al=0;
484                         return 0;
485                         }
486                 }

488         SSLerr(SSL_F_SSL_PARSE_SERVERHELLO_USE_SRTP_EXT,SSL_R_BAD_SRTP_PROTECTIO
489         *al=SSL_AD_DECODE_ERROR;
490         return 1;
491         }


494 #endif
```

```
**********************************************************
   86215 Wed Aug 13 19:33:05 2014
new/usr/src/lib/openssl/libsunw_ssl/s3_clnt.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
______unchanged_portion_omitted_
1975 #endif

1977         if ((unsigned long)(p - d + llen) != n)
1978                 {
1979                 ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_DECODE_ERROR);
1980                 SSLerr(SSL_F_SSL3_GET_CERTIFICATE_REQUEST,SSL_R_LENGTH_MISMATCH)
1981                 goto err;
1982                 }

1984         for (nc=0; nc<llen; )
1985                 {
1986                 n2s(p,l);
1987                 if ((l+nc+2) > llen)
1988                         {
1989                         if ((s->options & SSL_OP_NETSCAPE_CA_DN_BUG))
1990                                 goto cont; /* netscape bugs */
1991                         ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_DECODE_ERROR);
1992                         SSLerr(SSL_F_SSL3_GET_CERTIFICATE_REQUEST,SSL_R_CA_DN_TO
1993                         goto err;
1994                         }

1996                 q=p;

1998                 if ((xn=d2i_X509_NAME(NULL,&q,l)) == NULL)
1999                         {
2000                         /* If netscape tolerance is on, ignore errors */
2001                         if (s->options & SSL_OP_NETSCAPE_CA_DN_BUG)
2002                                 goto cont;
2003                         else
2004                                 {
2005                                 ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_DECODE_ER
2006                                 SSLerr(SSL_F_SSL3_GET_CERTIFICATE_REQUEST,ERR_R_
2007                                 goto err;
2008                                 }
2009                         }

2011                 if (q != (p+l))
2012                         {
2013                         ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_DECODE_ERROR);
2014                         SSLerr(SSL_F_SSL3_GET_CERTIFICATE_REQUEST,SSL_R_CA_DN_LE
2015                         goto err;
2016                         }
2017                 if (!sk_X509_NAME_push(ca_sk,xn))
2018                         {
2019                         SSLerr(SSL_F_SSL3_GET_CERTIFICATE_REQUEST,ERR_R_MALLOC_F
2020                         goto err;
2021                         }

2023                 p+=l;
2024                 nc+=l+2;
2025                 }

2027         if (0)
2028                 {
2029 cont:
2030                 ERR_clear_error();
2031                 }

2033         /* we should setup a certificate to return.... */
2034         s->s3->tmp.cert_req=1;
```



```
2035         s->s3->tmp.ctype_num=ctype_num;
2036         if (s->s3->tmp.ca_names != NULL)
2037                 sk_X509_NAME_pop_free(s->s3->tmp.ca_names,X509_NAME_free);
2038         s->s3->tmp.ca_names=ca_sk;
2039         ca_sk=NULL;

2041         ret=1;
2042 err:
2043         if (ca_sk != NULL) sk_X509_NAME_pop_free(ca_sk,X509_NAME_free);
2044         return(ret);
2045         }

2047 static int ca_dn_cmp(const X509_NAME * const *a, const X509_NAME * const *b)
2048         {
2049         return(X509_NAME_cmp(*a,*b));
2050         }
2051 #ifndef OPENSSL_NO_TLSEXT
2052 int ssl3_get_new_session_ticket(SSL *s)
2053         {
2054         int ok,al,ret=0, ticklen;
2055         long n;
2056         const unsigned char *p;
2057         unsigned char *d;

2059         n=s->method->ssl_get_message(s,
2060                 SSL3_ST_CR_SESSION_TICKET_A,
2061                 SSL3_ST_CR_SESSION_TICKET_B,
2062                 -1,
2063                 16384,
2064                 &ok);

2066         if (!ok)
2067                 return((int)n);

2069         if (s->s3->tmp.message_type == SSL3_MT_FINISHED)
2070                 {
2071                 s->s3->tmp.reuse_message=1;
2072                 return(1);
2073                 }
2074         if (s->s3->tmp.message_type != SSL3_MT_NEWSESSION_TICKET)
2075                 {
2076                 al=SSL_AD_UNEXPECTED_MESSAGE;
2077                 SSLerr(SSL_F_SSL3_GET_NEW_SESSION_TICKET,SSL_R_BAD_MESSAGE_TYPE)
2078                 goto f_err;
2079                 }
2080         if (n < 6)
2081                 {
2082                 /* need at least ticket_lifetime_hint + ticket length */
2083                 al = SSL_AD_DECODE_ERROR;
2084                 SSLerr(SSL_F_SSL3_GET_NEW_SESSION_TICKET,SSL_R_LENGTH_MISMATCH);
2085                 goto f_err;
2086                 }

2088         p=d=(unsigned char *)s->init_msg;
2089         n2l(p, s->session->tlsext_tick_lifetime_hint);
2090         n2s(p, ticklen);
2091         /* ticket_lifetime_hint + ticket_length + ticket */
2092         if (ticklen + 6 != n)
2093                 {
2094                 al = SSL_AD_DECODE_ERROR;
2095                 SSLerr(SSL_F_SSL3_GET_NEW_SESSION_TICKET,SSL_R_LENGTH_MISMATCH);
2096                 goto f_err;
2097                 }
2098         if (s->session->tlsext_tick)
2099                 {
2100                 OPENSSL_free(s->session->tlsext_tick);
```

```
2101                 s->session->tlsext_ticklen = 0;
2102                 }
2103         s->session->tlsext_tick = OPENSSL_malloc(ticklen);
2104         if (!s->session->tlsext_tick)
2105                 {
2106                 SSLerr(SSL_F_SSL3_GET_NEW_SESSION_TICKET,ERR_R_MALLOC_FAILURE);
2107                 goto err;
2108                 }
2109         memcpy(s->session->tlsext_tick, p, ticklen);
2110         s->session->tlsext_ticklen = ticklen;
2111         /* There are two ways to detect a resumed ticket sesion.
2112          * One is to set an appropriate session ID and then the server
2113          * must return a match in ServerHello. This allows the normal
2114          * client session ID matching to work and we know much
2115          * earlier that the ticket has been accepted.
2116          *
2117          * The other way is to set zero length session ID when the
2118          * ticket is presented and rely on the handshake to determine
2119          * session resumption.
2120          *
2121          * We choose the former approach because this fits in with
2122          * assumptions elsewhere in OpenSSL. The session ID is set
2123          * to the SHA256 (or SHA1 is SHA256 is disabled) hash of the
2124          * ticket.
2125          */
2126         EVP_Digest(p, ticklen,
2127                         s->session->session_id, &s->session->session_id_length,
2128 #ifndef OPENSSL_NO_SHA256
2129                                                         EVP_sha256(), NULL);
2130 #else
2131                                                         EVP_sha1(), NULL);
2132 #endif
2133         ret=1;
2134         return(ret);
2135 f_err:
2136         ssl3_send_alert(s,SSL3_AL_FATAL,al);
2137 err:
2138         return(-1);
2139         }

2141 int ssl3_get_cert_status(SSL *s)
2142         {
2143         int ok, al;
2144         unsigned long resplen,n;
2145         const unsigned char *p;

2147         n=s->method->ssl_get_message(s,
2148                 SSL3_ST_CR_CERT_STATUS_A,
2149                 SSL3_ST_CR_CERT_STATUS_B,
2150                 SSL3_MT_CERTIFICATE_STATUS,
2151                 16384,
2152                 &ok);

2154         if (!ok) return((int)n);
2155         if (n < 4)
2156                 {
2157                 /* need at least status type + length */
2158                 al = SSL_AD_DECODE_ERROR;
2159                 SSLerr(SSL_F_SSL3_GET_CERT_STATUS,SSL_R_LENGTH_MISMATCH);
2160                 goto f_err;
2161                 }
2162         p = (unsigned char *)s->init_msg;
2163         if (*p++ != TLSEXT_STATUSTYPE_ocsp)
2164                 {
2165                 al = SSL_AD_DECODE_ERROR;
2166                 SSLerr(SSL_F_SSL3_GET_CERT_STATUS,SSL_R_UNSUPPORTED_STATUS_TYPE)
```



```
2167                 goto f_err;
2168                 }
2169         n2l3(p, resplen);
2170         if (resplen + 4 != n)
2171                 {
2172                 al = SSL_AD_DECODE_ERROR;
2173                 SSLerr(SSL_F_SSL3_GET_CERT_STATUS,SSL_R_LENGTH_MISMATCH);
2174                 goto f_err;
2175                 }
2176         if (s->tlsext_ocsp_resp)
2177                 OPENSSL_free(s->tlsext_ocsp_resp);
2178         s->tlsext_ocsp_resp = BUF_memdup(p, resplen);
2179         if (!s->tlsext_ocsp_resp)
2180                 {
2181                 al = SSL_AD_INTERNAL_ERROR;
2182                 SSLerr(SSL_F_SSL3_GET_CERT_STATUS,ERR_R_MALLOC_FAILURE);
2183                 goto f_err;
2184                 }
2185         s->tlsext_ocsp_resplen = resplen;
2186         if (s->ctx->tlsext_status_cb)
2187                 {
2188                 int ret;
2189                 ret = s->ctx->tlsext_status_cb(s, s->ctx->tlsext_status_arg);
2190                 if (ret == 0)
2191                         {
2192                         al = SSL_AD_BAD_CERTIFICATE_STATUS_RESPONSE;
2193                         SSLerr(SSL_F_SSL3_GET_CERT_STATUS,SSL_R_INVALID_STATUS_R
2194                         goto f_err;
2195                         }
2196                 if (ret < 0)
2197                         {
2198                         al = SSL_AD_INTERNAL_ERROR;
2199                         SSLerr(SSL_F_SSL3_GET_CERT_STATUS,ERR_R_MALLOC_FAILURE);
2200                         goto f_err;
2201                         }
2202                 }
2203         return 1;
2204 f_err:
2205         ssl3_send_alert(s,SSL3_AL_FATAL,al);
2206         return(-1);
2207         }
2208 #endif

2210 int ssl3_get_server_done(SSL *s)
2211         {
2212         int ok,ret=0;
2213         long n;

2215         n=s->method->ssl_get_message(s,
2216                 SSL3_ST_CR_SRVR_DONE_A,
2217                 SSL3_ST_CR_SRVR_DONE_B,
2218                 SSL3_MT_SERVER_DONE,
2219                 30, /* should be very small, like 0 :-) */
2220                 &ok);

2222         if (!ok) return((int)n);
2223         if (n > 0)
2224                 {
2225                 /* should contain no data */
2226                 ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_DECODE_ERROR);
2227                 SSLerr(SSL_F_SSL3_GET_SERVER_DONE,SSL_R_LENGTH_MISMATCH);
2228                 return -1;
2229                 }
2230         ret=1;
2231         return(ret);
2232         }
```

```
2235 int ssl3_send_client_key_exchange(SSL *s)
2236         {
2237         unsigned char *p,*d;
2238         int n;
2239         unsigned long alg_k;
2240 #ifndef OPENSSL_NO_RSA
2241         unsigned char *q;
2242         EVP_PKEY *pkey=NULL;
2243 #endif
2244 #ifndef OPENSSL_NO_KRB5
2245         KSSL_ERR kssl_err;
2246 #endif /* OPENSSL_NO_KRB5 */
2247 #ifndef OPENSSL_NO_ECDH
2248         EC_KEY *clnt_ecdh = NULL;
2249         const EC_POINT *srvr_ecpoint = NULL;
2250         EVP_PKEY *srvr_pub_pkey = NULL;
2251         unsigned char *encodedPoint = NULL;
2252         int encoded_pt_len = 0;
2253         BN_CTX * bn_ctx = NULL;
2254 #endif

2256         if (s->state == SSL3_ST_CW_KEY_EXCH_A)
2257                 {
2258                 d=(unsigned char *)s->init_buf->data;
2259                 p= &(d[4]);

2261                 alg_k=s->s3->tmp.new_cipher->algorithm_mkey;

2263                 /* Fool emacs indentation */
2264                 if (0) {}
2265 #ifndef OPENSSL_NO_RSA
2266                 else if (alg_k & SSL_kRSA)
2267                         {
2268                         RSA *rsa;
2269                         unsigned char tmp_buf[SSL_MAX_MASTER_KEY_LENGTH];

2271                         if (s->session->sess_cert == NULL)
2272                                 {
2273                                 /* We should always have a server certificate wi
2274                                 SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2275                                 goto err;
2276                                 }

2279 #endif /* ! codereview */
2280                         if (s->session->sess_cert->peer_rsa_tmp != NULL)
2281                                 rsa=s->session->sess_cert->peer_rsa_tmp;
2282                         else
2283                                 {
2284                                 pkey=X509_get_pubkey(s->session->sess_cert->peer
2285                                 if ((pkey == NULL) ||
2286                                         (pkey->type != EVP_PKEY_RSA) ||
2287                                         (pkey->pkey.rsa == NULL))
2288                                         {
2289                                         SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2290                                         goto err;
2291                                         }
2292                                 rsa=pkey->pkey.rsa;
2293                                 EVP_PKEY_free(pkey);
2294                                 }

2296                         tmp_buf[0]=s->client_version>>8;
2297                         tmp_buf[1]=s->client_version&0xff;
2298                         if (RAND_bytes(&(tmp_buf[2]),sizeof tmp_buf-2) <= 0)
```



```
2299                                         goto err;

2301                         s->session->master_key_length=sizeof tmp_buf;

2303                         q=p;
2304                         /* Fix buf for TLS and beyond */
2305                         if (s->version > SSL3_VERSION)
2306                                 p+=2;
2307                         n=RSA_public_encrypt(sizeof tmp_buf,
2308                                 tmp_buf,p,rsa,RSA_PKCS1_PADDING);
2309 #ifdef PKCS1_CHECK
2310                         if (s->options & SSL_OP_PKCS1_CHECK_1) p[1]++;
2311                         if (s->options & SSL_OP_PKCS1_CHECK_2) tmp_buf[0]=0x70;
2312 #endif
2313                         if (n <= 0)
2314                                 {
2315                                 SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,SSL_R
2316                                 goto err;
2317                                 }

2319                         /* Fix buf for TLS and beyond */
2320                         if (s->version > SSL3_VERSION)
2321                                 {
2322                                 s2n(n,q);
2323                                 n+=2;
2324                                 }

2326                         s->session->master_key_length=
2327                                 s->method->ssl3_enc->generate_master_secret(s,
2328                                         s->session->master_key,
2329                                         tmp_buf,sizeof tmp_buf);
2330                         OPENSSL_cleanse(tmp_buf,sizeof tmp_buf);
2331                         }
2332 #endif
2333 #ifndef OPENSSL_NO_KRB5
2334                 else if (alg_k & SSL_kKRB5)
2335                         {
2336                         krb5_error_code krb5rc;
2337                         KSSL_CTX        *kssl_ctx = s->kssl_ctx;
2338                         /*  krb5_data   krb5_ap_req;  */
2339                         krb5_data       *enc_ticket;
2340                         krb5_data       authenticator, *authp = NULL;
2341                         EVP_CIPHER_CTX  ciph_ctx;
2342                         const EVP_CIPHER *enc = NULL;
2343                         unsigned char   iv[EVP_MAX_IV_LENGTH];
2344                         unsigned char   tmp_buf[SSL_MAX_MASTER_KEY_LENGTH];
2345                         unsigned char   epms[SSL_MAX_MASTER_KEY_LENGTH
2346                                             + EVP_MAX_IV_LENGTH];
2347                         int             padl, outl = sizeof(epms);

2349                         EVP_CIPHER_CTX_init(&ciph_ctx);

2351 #ifdef KSSL_DEBUG
2352                         printf("ssl3_send_client_key_exchange(%lx & %lx)\n",
2353                                 alg_k, SSL_kKRB5);
2354 #endif  /* KSSL_DEBUG */

2356                         authp = NULL;
2357 #ifdef KRB5SENDAUTH
2358                         if (KRB5SENDAUTH)  authp = &authenticator;
2359 #endif  /* KRB5SENDAUTH */

2361                         krb5rc = kssl_cget_tkt(kssl_ctx, &enc_ticket, authp,
2362                                 &kssl_err);
2363                         enc = kssl_map_enc(kssl_ctx->enctype);
2364                         if (enc == NULL)
```

```
2365                            goto err;
2366 #ifdef KSSL_DEBUG
2367                         {
2368                         printf("kssl_cget_tkt rtn %d\n", krb5rc);
2369                         if (krb5rc && kssl_err.text)
2370                           printf("kssl_cget_tkt kssl_err=%s\n", kssl_err.text);
2371                         }
2372 #endif  /* KSSL_DEBUG */

2374                         if (krb5rc)
2375                               {
2376                               ssl3_send_alert(s,SSL3_AL_FATAL,
2377                                        SSL_AD_HANDSHAKE_FAILURE);
2378                               SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2379                                        kssl_err.reason);
2380                               goto err;
2381                               }

2383                         /*  20010406 VRS - Earlier versions used KRB5 AP_REQ
2384                         **  in place of RFC 2712 KerberosWrapper, as in:
2385                         **
2386                         **  Send ticket (copy to *p, set n = length)
2387                         **  n = krb5_ap_req.length;
2388                         **  memcpy(p, krb5_ap_req.data, krb5_ap_req.length);
2389                         **  if (krb5_ap_req.data)
2390                         **    kssl_krb5_free_data_contents(NULL,&krb5_ap_req);
2391                         **
2392                         **  Now using real RFC 2712 KerberosWrapper
2393                         **  (Thanks to Simon Wilkinson <sxw@sxw.org.uk>)
2394                         **  Note: 2712 "opaque" types are here replaced
2395                         **  with a 2-byte length followed by the value.
2396                         **  Example:
2397                         **  KerberosWrapper= xx xx asn1ticket 0 0 xx xx encpms
2398                         **  Where "xx xx" = length bytes.  Shown here with
2399                         **  optional authenticator omitted.
2400                         */

2402                         /*  KerberosWrapper.Ticket              */
2403                         s2n(enc_ticket->length,p);
2404                         memcpy(p, enc_ticket->data, enc_ticket->length);
2405                         p+= enc_ticket->length;
2406                         n = enc_ticket->length + 2;

2408                         /*  KerberosWrapper.Authenticator       */
2409                         if (authp  &&  authp->length)
2410                               {
2411                               s2n(authp->length,p);
2412                               memcpy(p, authp->data, authp->length);
2413                               p+= authp->length;
2414                               n+= authp->length + 2;

2416                               free(authp->data);
2417                               authp->data = NULL;
2418                               authp->length = 0;
2419                               }
2420                         else
2421                               {
2422                               s2n(0,p);/*  null authenticator length  */
2423                               n+=2;
2424                               }

2426                            tmp_buf[0]=s->client_version>>8;
2427                            tmp_buf[1]=s->client_version&0xff;
2428                            if (RAND_bytes(&(tmp_buf[2]),sizeof tmp_buf-2) <= 0)
2429                                goto err;
```



```
2431                         /*  20010420 VRS.  Tried it this way; failed.
2432                         **      EVP_EncryptInit_ex(&ciph_ctx,enc, NULL,NULL);
2433                         **      EVP_CIPHER_CTX_set_key_length(&ciph_ctx,
2434                         **                               kssl_ctx->length);
2435                         **      EVP_EncryptInit_ex(&ciph_ctx,NULL, key,iv);
2436                         */

2438                         memset(iv, 0, sizeof iv);  /* per RFC 1510 */
2439                         EVP_EncryptInit_ex(&ciph_ctx,enc, NULL,
2440                                kssl_ctx->key,iv);
2441                         EVP_EncryptUpdate(&ciph_ctx,epms,&outl,tmp_buf,
2442                                sizeof tmp_buf);
2443                         EVP_EncryptFinal_ex(&ciph_ctx,&(epms[outl]),&padl);
2444                         outl += padl;
2445                         if (outl > (int)sizeof epms)
2446                                {
2447                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE, ERR_
2448                                goto err;
2449                                }
2450                         EVP_CIPHER_CTX_cleanup(&ciph_ctx);

2452                         /*  KerberosWrapper.EncryptedPreMasterSecret    */
2453                         s2n(outl,p);
2454                         memcpy(p, epms, outl);
2455                         p+=outl;
2456                         n+=outl + 2;

2458                         s->session->master_key_length=
2459                                s->method->ssl3_enc->generate_master_secret(s,
2460                                       s->session->master_key,
2461                                       tmp_buf, sizeof tmp_buf);

2463                         OPENSSL_cleanse(tmp_buf, sizeof tmp_buf);
2464                         OPENSSL_cleanse(epms, outl);
2465                         }
2466 #endif
2467 #ifndef OPENSSL_NO_DH
2468                      else if (alg_k & (SSL_kEDH|SSL_kDHr|SSL_kDHd))
2469                         {
2470                         DH *dh_srvr,*dh_clnt;

2472                         if (s->session->sess_cert == NULL)
2473                                {
2474                                ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_UNEXPECTE
2475                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,SSL_R
2476                                goto err;
2477                                }

2479                         if (s->session->sess_cert->peer_dh_tmp != NULL)
2480                                dh_srvr=s->session->sess_cert->peer_dh_tmp;
2481                         else
2482                                {
2483                                /* we get them from the cert */
2484                                ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_HANDSHAKE
2485                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,SSL_R
2486                                goto err;
2487                                }

2489                         /* generate a new random key */
2490                         if ((dh_clnt=DHparams_dup(dh_srvr)) == NULL)
2491                                {
2492                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2493                                goto err;
2494                                }
2495                         if (!DH_generate_key(dh_clnt))
2496                                {
```

```
2497                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2498                                DH_free(dh_clnt);
2499                                goto err;
2500                                }

2502                        /* use the 'p' output buffer for the DH key, but
2503                         * make sure to clear it out afterwards */

2505                        n=DH_compute_key(p,dh_srvr->pub_key,dh_clnt);

2507                        if (n <= 0)
2508                                {
2509                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2510                                DH_free(dh_clnt);
2511                                goto err;
2512                                }

2514                        /* generate master key from the result */
2515                        s->session->master_key_length=
2516                                s->method->ssl3_enc->generate_master_secret(s,
2517                                        s->session->master_key,p,n);
2518                        /* clean up */
2519                        memset(p,0,n);

2521                        /* send off the data */
2522                        n=BN_num_bytes(dh_clnt->pub_key);
2523                        s2n(n,p);
2524                        BN_bn2bin(dh_clnt->pub_key,p);
2525                        n+=2;

2527                        DH_free(dh_clnt);

2529                        /* perhaps clean things up a bit EAY EAY EAY EAY*/
2530                        }
2531 #endif

2533 #ifndef OPENSSL_NO_ECDH
2534                 else if (alg_k & (SSL_kEECDH|SSL_kECDHr|SSL_kECDHe))
2535                        {
2536                        const EC_GROUP *srvr_group = NULL;
2537                        EC_KEY *tkey;
2538                        int ecdh_clnt_cert = 0;
2539                        int field_size = 0;

2541                        if (s->session->sess_cert == NULL)
2542                                {
2543                                ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_UNEXPECTE
2544                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,SSL_R
2545                                goto err;
2546                                }

2548                        /* Did we send out the client's
2549                         * ECDH share for use in premaster
2550                         * computation as part of client certificate?
2551                         * If so, set ecdh_clnt_cert to 1.
2552                         */
2553                        if ((alg_k & (SSL_kECDHr|SSL_kECDHe)) && (s->cert != NUL
2554                                {
2555                                /* XXX: For now, we do not support client
2556                                 * authentication using ECDH certificates.
2557                                 * To add such support, one needs to add
2558                                 * code that checks for appropriate
2559                                 * conditions and sets ecdh_clnt_cert to 1.
2560                                 * For example, the cert have an ECC
2561                                 * key on the same curve as the server's
2562                                 * and the key should be authorized for
```



```
2563                                 * key agreement.
2564                                 *
2565                                 * One also needs to add code in ssl3_connect
2566                                 * to skip sending the certificate verify
2567                                 * message.
2568                                 *
2569                                 * if ((s->cert->key->privatekey != NULL) &&
2570                                 *     (s->cert->key->privatekey->type ==
2571                                 *      EVP_PKEY_EC) && ...)
2572                                 * ecdh_clnt_cert = 1;
2573                                 */
2574                                }

2576                        if (s->session->sess_cert->peer_ecdh_tmp != NULL)
2577                                {
2578                                tkey = s->session->sess_cert->peer_ecdh_tmp;
2579                                }
2580                        else
2581                                {
2582                                /* Get the Server Public Key from Cert */
2583                                srvr_pub_pkey = X509_get_pubkey(s->session-> \
2584                                    sess_cert->peer_pkeys[SSL_PKEY_ECC].x509);
2585                                if ((srvr_pub_pkey == NULL) ||
2586                                    (srvr_pub_pkey->type != EVP_PKEY_EC) ||
2587                                    (srvr_pub_pkey->pkey.ec == NULL))
2588                                        {
2589                                        SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2590                                            ERR_R_INTERNAL_ERROR);
2591                                        goto err;
2592                                        }

2594                                tkey = srvr_pub_pkey->pkey.ec;
2595                                }

2597                        srvr_group   = EC_KEY_get0_group(tkey);
2598                        srvr_ecpoint = EC_KEY_get0_public_key(tkey);

2600                        if ((srvr_group == NULL) || (srvr_ecpoint == NULL))
2601                                {
2602                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2603                                    ERR_R_INTERNAL_ERROR);
2604                                goto err;
2605                                }

2607                        if ((clnt_ecdh=EC_KEY_new()) == NULL)
2608                                {
2609                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2610                                goto err;
2611                                }

2613                        if (!EC_KEY_set_group(clnt_ecdh, srvr_group))
2614                                {
2615                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2616                                goto err;
2617                                }
2618                        if (ecdh_clnt_cert)
2619                                {
2620                                /* Reuse key info from our certificate
2621                                 * We only need our private key to perform
2622                                 * the ECDH computation.
2623                                 */
2624                                const BIGNUM *priv_key;
2625                                tkey = s->cert->key->privatekey->pkey.ec;
2626                                priv_key = EC_KEY_get0_private_key(tkey);
2627                                if (priv_key == NULL)
2628                                        {
```

```
2629                                    SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2630                                    goto err;
2631                                    }
2632                            if (!EC_KEY_set_private_key(clnt_ecdh, priv_key)
2633                                    {
2634                                    SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2635                                    goto err;
2636                                    }
2637                            }
2638                    else
2639                            {
2640                            /* Generate a new ECDH key pair */
2641                            if (!(EC_KEY_generate_key(clnt_ecdh)))
2642                                    {
2643                                    SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2644                                    goto err;
2645                                    }
2646                            }

2648                    /* use the 'p' output buffer for the ECDH key, but
2649                     * make sure to clear it out afterwards
2650                     */

2652                    field_size = EC_GROUP_get_degree(srvr_group);
2653                    if (field_size <= 0)
2654                            {
2655                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2656                                   ERR_R_ECDH_LIB);
2657                            goto err;
2658                            }
2659                    n=ECDH_compute_key(p, (field_size+7)/8, srvr_ecpoint, cl
2660                    if (n <= 0)
2661                            {
2662                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2663                                   ERR_R_ECDH_LIB);
2664                            goto err;
2665                            }

2667                    /* generate master key from the result */
2668                    s->session->master_key_length = s->method->ssl3_enc \
2669                        -> generate_master_secret(s,
2670                            s->session->master_key,
2671                            p, n);

2673                    memset(p, 0, n); /* clean up */

2675                    if (ecdh_clnt_cert)
2676                            {
2677                            /* Send empty client key exch message */
2678                            n = 0;
2679                            }
2680                    else
2681                            {
2682                            /* First check the size of encoding and
2683                             * allocate memory accordingly.
2684                             */
2685                            encoded_pt_len =
2686                                EC_POINT_point2oct(srvr_group,
2687                                    EC_KEY_get0_public_key(clnt_ecdh),
2688                                    POINT_CONVERSION_UNCOMPRESSED,
2689                                    NULL, 0, NULL);

2691                            encodedPoint = (unsigned char *)
2692                                OPENSSL_malloc(encoded_pt_len *
2693                                    sizeof(unsigned char));
2694                            bn_ctx = BN_CTX_new();
```



```
2695                                if ((encodedPoint == NULL) ||
2696                                    (bn_ctx == NULL))
2697                                        {
2698                                        SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2699                                        goto err;
2700                                        }

2702                                /* Encode the public key */
2703                                n = EC_POINT_point2oct(srvr_group,
2704                                    EC_KEY_get0_public_key(clnt_ecdh),
2705                                    POINT_CONVERSION_UNCOMPRESSED,
2706                                    encodedPoint, encoded_pt_len, bn_ctx);

2708                                *p = n; /* length of encoded point */
2709                                /* Encoded point will be copied here */
2710                                p += 1;
2711                                /* copy the point */
2712                                memcpy((unsigned char *)p, encodedPoint, n);
2713                                /* increment n to account for length field */
2714                                n += 1;
2715                                }

2717                        /* Free allocated memory */
2718                        BN_CTX_free(bn_ctx);
2719                        if (encodedPoint != NULL) OPENSSL_free(encodedPoint);
2720                        if (clnt_ecdh != NULL)
2721                                EC_KEY_free(clnt_ecdh);
2722                        EVP_PKEY_free(srvr_pub_pkey);
2723                        }
2724 #endif /* !OPENSSL_NO_ECDH */
2725                 else if (alg_k & SSL_kGOST)
2726                        {
2727                        /* GOST key exchange message creation */
2728                        EVP_PKEY_CTX *pkey_ctx;
2729                        X509 *peer_cert;
2730                        size_t msglen;
2731                        unsigned int md_len;
2732                        int keytype;
2733                        unsigned char premaster_secret[32],shared_ukm[32], tmp[2
2734                        EVP_MD_CTX *ukm_hash;
2735                        EVP_PKEY *pub_key;

2737                        /* Get server sertificate PKEY and create ctx from it */
2738                        peer_cert=s->session->sess_cert->peer_pkeys[(keytype=SSL
2739                        if (!peer_cert)
2740                                peer_cert=s->session->sess_cert->peer_pkeys[(key
2741                        if (!peer_cert)         {
2742                                        SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2743                                        goto err;
2744                                }

2746                        pkey_ctx=EVP_PKEY_CTX_new(pub_key=X509_get_pubkey(peer_c
2747                        /* If we have send a certificate, and certificate key

2749                         * parameters match those of server certificate, use
2750                         * certificate key for key exchange
2751                         */

2753                         /* Otherwise, generate ephemeral key pair */

2755                        EVP_PKEY_encrypt_init(pkey_ctx);
2756                          /* Generate session key */
2757                    RAND_bytes(premaster_secret,32);
2758                        /* If we have client certificate, use its secret as peer
2759                        if (s->s3->tmp.cert_req && s->cert->key->privatekey) {
2760                                if (EVP_PKEY_derive_set_peer(pkey_ctx,s->cert->k
```

```
2761                                    /* If there was an error - just ignore i
2762                                    * would be used
2763                                    */
2764                                    ERR_clear_error();
2765                            }
2766                    }
2767                    /* Compute shared IV and store it in algorithm-specific
2768                     * context data */
2769                    ukm_hash = EVP_MD_CTX_create();
2770                    EVP_DigestInit(ukm_hash,EVP_get_digestbynid(NID_id_GostR
2771                    EVP_DigestUpdate(ukm_hash,s->s3->client_random,SSL3_RAND
2772                    EVP_DigestUpdate(ukm_hash,s->s3->server_random,SSL3_RAND
2773                    EVP_DigestFinal_ex(ukm_hash, shared_ukm, &md_len);
2774                    EVP_MD_CTX_destroy(ukm_hash);
2775                    if (EVP_PKEY_CTX_ctrl(pkey_ctx,-1,EVP_PKEY_OP_ENCRYPT,EV
2776                            8,shared_ukm)<0) {
2777                                    SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHAN
2778                                            SSL_R_LIBRARY_BUG);
2779                                    goto err;
2780                            }
2781                    /* Make GOST keytransport blob message */
2782                    /*Encapsulate it into sequence */
2783                    *(p++)=V_ASN1_SEQUENCE | V_ASN1_CONSTRUCTED;
2784                    msglen=255;
2785                    if (EVP_PKEY_encrypt(pkey_ctx,tmp,&msglen,premaster_secr
2786                    SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2787                                    SSL_R_LIBRARY_BUG);
2788                            goto err;
2789                    }
2790                    if (msglen >= 0x80)
2791                            {
2792                            *(p++)=0x81;
2793                            *(p++)= msglen & 0xff;
2794                            n=msglen+3;
2795                            }
2796                    else
2797                            {
2798                            *(p++)= msglen & 0xff;
2799                            n=msglen+2;
2800                            }
2801                    memcpy(p, tmp, msglen);
2802                    /* Check if pubkey from client certificate was used */
2803                    if (EVP_PKEY_CTX_ctrl(pkey_ctx, -1, -1, EVP_PKEY_CTRL_PE
2804                            {
2805                            /* Set flag "skip certificate verify" */
2806                            s->s3->flags |= TLS1_FLAGS_SKIP_CERT_VERIFY;
2807                            }
2808                    EVP_PKEY_CTX_free(pkey_ctx);
2809                    s->session->master_key_length=
2810                            s->method->ssl3_enc->generate_master_secret(s,
2811                                    s->session->master_key,premaster_secret,
2812                    EVP_PKEY_free(pub_key);

2814                    }
2815 #ifndef OPENSSL_NO_SRP
2816                 else if (alg_k & SSL_kSRP)
2817                    {
2818                    if (s->srp_ctx.A != NULL)
2819                            {
2820                            /* send off the data */
2821                            n=BN_num_bytes(s->srp_ctx.A);
2822                            s2n(n,p);
2823                            BN_bn2bin(s->srp_ctx.A,p);
2824                            n+=2;
2825                            }
2826                    else
```



```
2827                            {
2828                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2829                            goto err;
2830                            }
2831                    if (s->session->srp_username != NULL)
2832                            OPENSSL_free(s->session->srp_username);
2833                    s->session->srp_username = BUF_strdup(s->srp_ctx.login);
2834                    if (s->session->srp_username == NULL)
2835                            {
2836                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2837                                    ERR_R_MALLOC_FAILURE);
2838                            goto err;
2839                            }

2841                    if ((s->session->master_key_length = SRP_generate_client
2842                            {
2843                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,ERR_R
2844                            goto err;
2845                            }
2846                    }
2847 #endif
2848 #ifndef OPENSSL_NO_PSK
2849                 else if (alg_k & SSL_kPSK)
2850                    {
2851                    char identity[PSK_MAX_IDENTITY_LEN];
2852                    unsigned char *t = NULL;
2853                    unsigned char psk_or_pre_ms[PSK_MAX_PSK_LEN*2+4];
2854                    unsigned int pre_ms_len = 0, psk_len = 0;
2855                    int psk_err = 1;

2857                    n = 0;
2858                    if (s->psk_client_callback == NULL)
2859                            {
2860                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2861                                    SSL_R_PSK_NO_CLIENT_CB);
2862                            goto err;
2863                            }

2865                    psk_len = s->psk_client_callback(s, s->ctx->psk_identity
2866                            identity, PSK_MAX_IDENTITY_LEN,
2867                            psk_or_pre_ms, sizeof(psk_or_pre_ms));
2868                    if (psk_len > PSK_MAX_PSK_LEN)
2869                            {
2870                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2871                                    ERR_R_INTERNAL_ERROR);
2872                            goto psk_err;
2873                            }
2874                    else if (psk_len == 0)
2875                            {
2876                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2877                                    SSL_R_PSK_IDENTITY_NOT_FOUND);
2878                            goto psk_err;
2879                            }

2881                    /* create PSK pre_master_secret */
2882                    pre_ms_len = 2+psk_len+2+psk_len;
2883                    t = psk_or_pre_ms;
2884                    memmove(psk_or_pre_ms+psk_len+4, psk_or_pre_ms, psk_len)
2885                    s2n(psk_len, t);
2886                    memset(t, 0, psk_len);
2887                    t+=psk_len;
2888                    s2n(psk_len, t);

2890                    if (s->session->psk_identity_hint != NULL)
2891                            OPENSSL_free(s->session->psk_identity_hint);
2892                    s->session->psk_identity_hint = BUF_strdup(s->ctx->psk_i
```

```
2893                        if (s->ctx->psk_identity_hint != NULL &&
2894                                s->session->psk_identity_hint == NULL)
2895                                {
2896                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2897                                        ERR_R_MALLOC_FAILURE);
2898                                goto psk_err;
2899                                }

2901                        if (s->session->psk_identity != NULL)
2902                                OPENSSL_free(s->session->psk_identity);
2903                        s->session->psk_identity = BUF_strdup(identity);
2904                        if (s->session->psk_identity == NULL)
2905                                {
2906                                SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2907                                        ERR_R_MALLOC_FAILURE);
2908                                goto psk_err;
2909                                }

2911                        s->session->master_key_length =
2912                                s->method->ssl3_enc->generate_master_secret(s,
2913                                        s->session->master_key,
2914                                        psk_or_pre_ms, pre_ms_len);
2915                        n = strlen(identity);
2916                        s2n(n, p);
2917                        memcpy(p, identity, n);
2918                        n+=2;
2919                        psk_err = 0;
2920                psk_err:
2921                        OPENSSL_cleanse(identity, PSK_MAX_IDENTITY_LEN);
2922                        OPENSSL_cleanse(psk_or_pre_ms, sizeof(psk_or_pre_ms));
2923                        if (psk_err != 0)
2924                                {
2925                                ssl3_send_alert(s, SSL3_AL_FATAL, SSL_AD_HANDSHA
2926                                goto err;
2927                                }
2928                        }
2929 #endif
2930                else
2931                        {
2932                        ssl3_send_alert(s, SSL3_AL_FATAL,
2933                            SSL_AD_HANDSHAKE_FAILURE);
2934                        SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
2935                            ERR_R_INTERNAL_ERROR);
2936                        goto err;
2937                        }

2939                *(d++)=SSL3_MT_CLIENT_KEY_EXCHANGE;
2940                l2n3(n,d);

2942                s->state=SSL3_ST_CW_KEY_EXCH_B;
2943                /* number of bytes to write */
2944                s->init_num=n+4;
2945                s->init_off=0;
2946                }

2948        /* SSL3_ST_CW_KEY_EXCH_B */
2949        return(ssl3_do_write(s,SSL3_RT_HANDSHAKE));
2950 err:
2951 #ifndef OPENSSL_NO_ECDH
2952        BN_CTX_free(bn_ctx);
2953        if (encodedPoint != NULL) OPENSSL_free(encodedPoint);
2954        if (clnt_ecdh != NULL)
2955                EC_KEY_free(clnt_ecdh);
2956        EVP_PKEY_free(srvr_pub_pkey);
2957 #endif
2958        return(-1);
```



```
2959        }

2961 int ssl3_send_client_verify(SSL *s)
2962        {
2963        unsigned char *p,*d;
2964        unsigned char data[MD5_DIGEST_LENGTH+SHA_DIGEST_LENGTH];
2965        EVP_PKEY *pkey;
2966        EVP_PKEY_CTX *pctx=NULL;
2967        EVP_MD_CTX mctx;
2968        unsigned u=0;
2969        unsigned long n;
2970        int j;

2972        EVP_MD_CTX_init(&mctx);

2974        if (s->state == SSL3_ST_CW_CERT_VRFY_A)
2975                {
2976                d=(unsigned char *)s->init_buf->data;
2977                p= &(d[4]);
2978                pkey=s->cert->key->privatekey;
2979 /* Create context from key and test if sha1 is allowed as digest */
2980                pctx = EVP_PKEY_CTX_new(pkey,NULL);
2981                EVP_PKEY_sign_init(pctx);
2982                if (EVP_PKEY_CTX_set_signature_md(pctx, EVP_sha1())>0)
2983                        {
2984                        if (TLS1_get_version(s) < TLS1_2_VERSION)
2985                                s->method->ssl3_enc->cert_verify_mac(s,
2986                                                NID_sha1,
2987                                                &(data[MD5_DIGEST_LENGTH]));
2988                        }
2989                else
2990                        {
2991                        ERR_clear_error();
2992                        }
2993                /* For TLS v1.2 send signature algorithm and signature
2994                 * using agreed digest and cached handshake records.
2995                 */
2996                if (TLS1_get_version(s) >= TLS1_2_VERSION)
2997                        {
2998                        long hdatalen = 0;
2999                        void *hdata;
3000                        const EVP_MD *md = s->cert->key->digest;
3001                        hdatalen = BIO_get_mem_data(s->s3->handshake_buffer,
3002                                                            &hdata);
3003                        if (hdatalen <= 0 || !tls12_get_sigandhash(p, pkey, md))
3004                                {
3005                                SSLerr(SSL_F_SSL3_SEND_CLIENT_VERIFY,
3006                                                ERR_R_INTERNAL_ERROR);
3007                                goto err;
3008                                }
3009                        p += 2;
3010 #ifdef SSL_DEBUG
3011                        fprintf(stderr, "Using TLS 1.2 with client alg %s\n",
3012                                                        EVP_MD_name(md));
3013 #endif
3014                        if (!EVP_SignInit_ex(&mctx, md, NULL)
3015                                || !EVP_SignUpdate(&mctx, hdata, hdatalen)
3016                                || !EVP_SignFinal(&mctx, p + 2, &u, pkey))
3017                                {
3018                                SSLerr(SSL_F_SSL3_SEND_CLIENT_VERIFY,
3019                                                ERR_R_EVP_LIB);
3020                                goto err;
3021                                }
3022                        s2n(u,p);
3023                        n = u + 4;
3024                        if (!ssl3_digest_cached_records(s))
```

```
3025                                    goto err;
3026                            }
3027                    else
3028 #ifndef OPENSSL_NO_RSA
3029                    if (pkey->type == EVP_PKEY_RSA)
3030                            {
3031                            s->method->ssl3_enc->cert_verify_mac(s,
3032                                    NID_md5,
3033                                    &(data[0]));
3034                            if (RSA_sign(NID_md5_sha1, data,
3035                                            MD5_DIGEST_LENGTH+SHA_DIGEST_LENGTH,
3036                                           &(p[2]), &u, pkey->pkey.rsa) <= 0 )
3037                                    {
3038                                    SSLerr(SSL_F_SSL3_SEND_CLIENT_VERIFY,ERR_R_RSA_L
3039                                    goto err;
3040                                    }
3041                            s2n(u,p);
3042                            n=u+2;
3043                            }
3044                    else
3045 #endif
3046 #ifndef OPENSSL_NO_DSA
3047                            if (pkey->type == EVP_PKEY_DSA)
3048                            {
3049                            if (!DSA_sign(pkey->save_type,
3050                                    &(data[MD5_DIGEST_LENGTH]),
3051                                    SHA_DIGEST_LENGTH,&(p[2]),
3052                                    (unsigned int *)&j,pkey->pkey.dsa))
3053                                    {
3054                                    SSLerr(SSL_F_SSL3_SEND_CLIENT_VERIFY,ERR_R_DSA_L
3055                                    goto err;
3056                                    }
3057                            s2n(j,p);
3058                            n=j+2;
3059                            }
3060                    else
3061 #endif
3062 #ifndef OPENSSL_NO_ECDSA
3063                            if (pkey->type == EVP_PKEY_EC)
3064                            {
3065                            if (!ECDSA_sign(pkey->save_type,
3066                                    &(data[MD5_DIGEST_LENGTH]),
3067                                    SHA_DIGEST_LENGTH,&(p[2]),
3068                                    (unsigned int *)&j,pkey->pkey.ec))
3069                                    {
3070                                    SSLerr(SSL_F_SSL3_SEND_CLIENT_VERIFY,
3071                                        ERR_R_ECDSA_LIB);
3072                                    goto err;
3073                                    }
3074                            s2n(j,p);
3075                            n=j+2;
3076                            }
3077                    else
3078 #endif
3079                    if (pkey->type == NID_id_GostR3410_94 || pkey->type == NID_id_Go
3080                    {
3081                    unsigned char signbuf[64];
3082                    int i;
3083                    size_t sigsize=64;
3084                    s->method->ssl3_enc->cert_verify_mac(s,
3085                            NID_id_GostR3411_94,
3086                            data);
3087                    if (EVP_PKEY_sign(pctx, signbuf, &sigsize, data, 32) <= 0) {
3088                            SSLerr(SSL_F_SSL3_SEND_CLIENT_VERIFY,
3089                            ERR_R_INTERNAL_ERROR);
3090                            goto err;
```



```
3091                    }
3092                    for (i=63,j=0; i>=0; j++, i--) {
3093                            p[2+j]=signbuf[i];
3094                    }
3095                    s2n(j,p);
3096                    n=j+2;
3097                    }
3098                    else
3099                    {
3100                            SSLerr(SSL_F_SSL3_SEND_CLIENT_VERIFY,ERR_R_INTERNAL_ERRO
3101                            goto err;
3102                    }
3103                    *(d++)=SSL3_MT_CERTIFICATE_VERIFY;
3104                    l2n3(n,d);

3106                    s->state=SSL3_ST_CW_CERT_VRFY_B;
3107                    s->init_num=(int)n+4;
3108                    s->init_off=0;
3109                    }
3110            EVP_MD_CTX_cleanup(&mctx);
3111            EVP_PKEY_CTX_free(pctx);
3112            return(ssl3_do_write(s,SSL3_RT_HANDSHAKE));
3113 err:
3114            EVP_MD_CTX_cleanup(&mctx);
3115            EVP_PKEY_CTX_free(pctx);
3116            return(-1);
3117            }

3119 int ssl3_send_client_certificate(SSL *s)
3120            {
3121            X509 *x509=NULL;
3122            EVP_PKEY *pkey=NULL;
3123            int i;
3124            unsigned long l;

3126            if (s->state == SSL3_ST_CW_CERT_A)
3127                    {
3128                    if ((s->cert == NULL) ||
3129                            (s->cert->key->x509 == NULL) ||
3130                            (s->cert->key->privatekey == NULL))
3131                            s->state=SSL3_ST_CW_CERT_B;
3132                    else
3133                            s->state=SSL3_ST_CW_CERT_C;
3134                    }

3136            /* We need to get a client cert */
3137            if (s->state == SSL3_ST_CW_CERT_B)
3138                    {
3139                    /* If we get an error, we need to
3140                     * ssl->rwstate=SSL_X509_LOOKUP; return(-1);
3141                     * We then get retied later */
3142                    i=0;
3143                    i = ssl_do_client_cert_cb(s, &x509, &pkey);
3144                    if (i < 0)
3145                            {
3146                            s->rwstate=SSL_X509_LOOKUP;
3147                            return(-1);
3148                            }
3149                    s->rwstate=SSL_NOTHING;
3150                    if ((i == 1) && (pkey != NULL) && (x509 != NULL))
3151                            {
3152                            s->state=SSL3_ST_CW_CERT_B;
3153                            if (    !SSL_use_certificate(s,x509) ||
3154                                    !SSL_use_PrivateKey(s,pkey))
3155                                    i=0;
3156                            }
```

```
3157                 else if (i == 1)
3158                         {
3159                         i=0;
3160                         SSLerr(SSL_F_SSL3_SEND_CLIENT_CERTIFICATE,SSL_R_BAD_DATA
3161                         }

3163                 if (x509 != NULL) X509_free(x509);
3164                 if (pkey != NULL) EVP_PKEY_free(pkey);
3165                 if (i == 0)
3166                         {
3167                         if (s->version == SSL3_VERSION)
3168                                 {
3169                                 s->s3->tmp.cert_req=0;
3170                                 ssl3_send_alert(s,SSL3_AL_WARNING,SSL_AD_NO_CERT
3171                                 return(1);
3172                                 }
3173                         else
3174                                 {
3175                                 s->s3->tmp.cert_req=2;
3176                                 }
3177                         }

3179                 /* Ok, we have a cert */
3180                 s->state=SSL3_ST_CW_CERT_C;
3181                 }

3183         if (s->state == SSL3_ST_CW_CERT_C)
3184                 {
3185                 s->state=SSL3_ST_CW_CERT_D;
3186                 l=ssl3_output_cert_chain(s,
3187                         (s->s3->tmp.cert_req == 2)?NULL:s->cert->key->x509);
3188                 s->init_num=(int)l;
3189                 s->init_off=0;
3190                 }
3191         /* SSL3_ST_CW_CERT_D */
3192         return(ssl3_do_write(s,SSL3_RT_HANDSHAKE));
3193         }

3195 #define has_bits(i,m)   (((i)&(m)) == (m))

3197 int ssl3_check_cert_and_algorithm(SSL *s)
3198         {
3199         int i,idx;
3200         long alg_k,alg_a;
3201         EVP_PKEY *pkey=NULL;
3202         SESS_CERT *sc;
3203 #ifndef OPENSSL_NO_RSA
3204         RSA *rsa;
3205 #endif
3206 #ifndef OPENSSL_NO_DH
3207         DH *dh;
3208 #endif

3210         alg_k=s->s3->tmp.new_cipher->algorithm_mkey;
3211         alg_a=s->s3->tmp.new_cipher->algorithm_auth;

3213         /* we don't have a certificate */
3214         if ((alg_a & (SSL_aDH|SSL_aNULL|SSL_aKRB5)) || (alg_k & SSL_kPSK))
3215                 return(1);

3217         sc=s->session->sess_cert;
3218         if (sc == NULL)
3219                 {
3220                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,ERR_R_INTERNAL_ERROR)
3221                 goto err;
3222                 }
```



```
3224 #ifndef OPENSSL_NO_RSA
3225         rsa=s->session->sess_cert->peer_rsa_tmp;
3226 #endif
3227 #ifndef OPENSSL_NO_DH
3228         dh=s->session->sess_cert->peer_dh_tmp;
3229 #endif

3231         /* This is the passed certificate */

3233         idx=sc->peer_cert_type;
3234 #ifndef OPENSSL_NO_ECDH
3235         if (idx == SSL_PKEY_ECC)
3236                 {
3237                 if (ssl_check_srvr_ecc_cert_and_alg(sc->peer_pkeys[idx].x509,
3238                                                         s) == 0)
3239                         { /* check failed */
3240                         SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_BAD_ECC
3241                         goto f_err;
3242                         }
3243                 else
3244                         {
3245                         return 1;
3246                         }
3247                 }
3248 #endif
3249         pkey=X509_get_pubkey(sc->peer_pkeys[idx].x509);
3250         i=X509_certificate_type(sc->peer_pkeys[idx].x509,pkey);
3251         EVP_PKEY_free(pkey);


3254         /* Check that we have a certificate if we require one */
3255         if ((alg_a & SSL_aRSA) && !has_bits(i,EVP_PK_RSA|EVP_PKT_SIGN))
3256                 {
3257                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_MISSING_RSA_SIG
3258                 goto f_err;
3259                 }
3260 #ifndef OPENSSL_NO_DSA
3261         else if ((alg_a & SSL_aDSS) && !has_bits(i,EVP_PK_DSA|EVP_PKT_SIGN))
3262                 {
3263                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_MISSING_DSA_SIG
3264                 goto f_err;
3265                 }
3266 #endif
3267 #ifndef OPENSSL_NO_RSA
3268         if ((alg_k & SSL_kRSA) &&
3269                 !(has_bits(i,EVP_PK_RSA|EVP_PKT_ENC) || (rsa != NULL)))
3270                 {
3271                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_MISSING_RSA_ENC
3272                 goto f_err;
3273                 }
3274 #endif
3275 #ifndef OPENSSL_NO_DH
3276         if ((alg_k & SSL_kEDH) &&
3277                 !(has_bits(i,EVP_PK_DH|EVP_PKT_EXCH) || (dh != NULL)))
3278                 {
3279                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_MISSING_DH_KEY)
3280                 goto f_err;
3281                 }
3282         else if ((alg_k & SSL_kDHr) && !has_bits(i,EVP_PK_DH|EVP_PKS_RSA))
3283                 {
3284                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_MISSING_DH_RSA_
3285                 goto f_err;
3286                 }
3287 #ifndef OPENSSL_NO_DSA
3288         else if ((alg_k & SSL_kDHd) && !has_bits(i,EVP_PK_DH|EVP_PKS_DSA))
```

```
3289                 {
3290                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_MISSING_DH_DSA_
3291                 goto f_err;
3292                 }
3293 #endif
3294 #endif

3296         if (SSL_C_IS_EXPORT(s->s3->tmp.new_cipher) && !has_bits(i,EVP_PKT_EXP))
3297                 {
3298 #ifndef OPENSSL_NO_RSA
3299                 if (alg_k & SSL_kRSA)
3300                         {
3301                         if (rsa == NULL
3302                             || RSA_size(rsa)*8 > SSL_C_EXPORT_PKEYLENGTH(s->s3->
3303                                 {
3304                                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R
3305                                 goto f_err;
3306                                 }
3307                         }
3308                 else
3309 #endif
3310 #ifndef OPENSSL_NO_DH
3311                         if (alg_k & (SSL_kEDH|SSL_kDHr|SSL_kDHd))
3312                             {
3313                             if (dh == NULL
3314                                 || DH_size(dh)*8 > SSL_C_EXPORT_PKEYLENGTH(s->s3
3315                                 {
3316                                 SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R
3317                                 goto f_err;
3318                                 }
3319                         }
3320                 else
3321 #endif
3322                         {
3323                         SSLerr(SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM,SSL_R_UNKNOWN
3324                         goto f_err;
3325                         }
3326                 }
3327         return(1);
3328 f_err:
3329         ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_HANDSHAKE_FAILURE);
3330 err:
3331         return(0);
3332         }

3334 #if !defined(OPENSSL_NO_TLSEXT) && !defined(OPENSSL_NO_NEXTPROTONEG)
3335 int ssl3_send_next_proto(SSL *s)
3336         {
3337         unsigned int len, padding_len;
3338         unsigned char *d;

3340         if (s->state == SSL3_ST_CW_NEXT_PROTO_A)
3341                 {
3342                 len = s->next_proto_negotiated_len;
3343                 padding_len = 32 - ((len + 2) % 32);
3344                 d = (unsigned char *)s->init_buf->data;
3345                 d[4] = len;
3346                 memcpy(d + 5, s->next_proto_negotiated, len);
3347                 d[5 + len] = padding_len;
3348                 memset(d + 6 + len, 0, padding_len);
3349                 *(d++)=SSL3_MT_NEXT_PROTO;
3350                 l2n3(2 + len + padding_len, d);
3351                 s->state = SSL3_ST_CW_NEXT_PROTO_B;
3352                 s->init_num = 4 + 2 + len + padding_len;
3353                 s->init_off = 0;
3354                 }
```



```
3356         return ssl3_do_write(s, SSL3_RT_HANDSHAKE);
3357 }
3358 #endif  /* !OPENSSL_NO_TLSEXT && !OPENSSL_NO_NEXTPROTONEG */

3360 /* Check to see if handshake is full or resumed. Usually this is just a
3361  * case of checking to see if a cache hit has occurred. In the case of
3362  * session tickets we have to check the next message to be sure.
3363  */

3365 #ifndef OPENSSL_NO_TLSEXT
3366 int ssl3_check_finished(SSL *s)
3367         {
3368         int ok;
3369         long n;
3370         /* If we have no ticket it cannot be a resumed session. */
3371         if (!s->session->tlsext_tick)
3372                 return 1;
3373         /* this function is called when we really expect a Certificate
3374          * message, so permit appropriate message length */
3375         n=s->method->ssl_get_message(s,
3376                 SSL3_ST_CR_CERT_A,
3377                 SSL3_ST_CR_CERT_B,
3378                 -1,
3379                 s->max_cert_list,
3380                 &ok);
3381         if (!ok) return((int)n);
3382         s->s3->tmp.reuse_message = 1;
3383         if ((s->s3->tmp.message_type == SSL3_MT_FINISHED)
3384                 || (s->s3->tmp.message_type == SSL3_MT_NEWSESSION_TICKET))
3385                 return 2;

3387         return 1;
3388         }
3389 #endif

3391 int ssl_do_client_cert_cb(SSL *s, X509 **px509, EVP_PKEY **ppkey)
3392         {
3393         int i = 0;
3394 #ifndef OPENSSL_NO_ENGINE
3395         if (s->ctx->client_cert_engine)
3396                 {
3397                 i = ENGINE_load_ssl_client_cert(s->ctx->client_cert_engine, s,
3398                                                 SSL_get_client_CA_list(s),
3399                                                 px509, ppkey, NULL, NULL, NULL);
3400                 if (i != 0)
3401                         return i;
3402                 }
3403 #endif
3404         if (s->ctx->client_cert_cb)
3405                 i = s->ctx->client_cert_cb(s,px509,ppkey);
3406         return i;
3407         }
```

```
**********************************************************
   83244 Wed Aug 13 19:33:06 2014
new/usr/src/lib/openssl/libsunw_ssl/s3_lib.c
4853 illumos-gate is not lint-clean when built with openssl 1.0
**********************************************************
```

```
   1 /* ssl/s3_lib.c */
   2 /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
   3  * All rights reserved.
   4  *
   5  * This package is an SSL implementation written
   6  * by Eric Young (eay@cryptsoft.com).
   7  * The implementation was written so as to conform with Netscapes SSL.
   8  *
   9  * This library is free for commercial and non-commercial use as long as
  10  * the following conditions are aheared to.  The following conditions
  11  * apply to all code found in this distribution, be it the RC4, RSA,
  12  * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
  13  * included with this distribution is covered by the same copyright terms
  14  * except that the holder is Tim Hudson (tjh@cryptsoft.com).
  15  *
  16  * Copyright remains Eric Young's, and as such any Copyright notices in
  17  * the code are not to be removed.
  18  * If this package is used in a product, Eric Young should be given attribution
  19  * as the author of the parts of the library used.
  20  * This can be in the form of a textual message at program startup or
  21  * in documentation (online or textual) provided with the package.
  22  *
  23  * Redistribution and use in source and binary forms, with or without
  24  * modification, are permitted provided that the following conditions
  25  * are met:
  26  * 1. Redistributions of source code must retain the copyright
  27  *    notice, this list of conditions and the following disclaimer.
  28  * 2. Redistributions in binary form must reproduce the above copyright
  29  *    notice, this list of conditions and the following disclaimer in the
  30  *    documentation and/or other materials provided with the distribution.
  31  * 3. All advertising materials mentioning features or use of this software
  32  *    must display the following acknowledgement:
  33  *    "This product includes cryptographic software written by
  34  *     Eric Young (eay@cryptsoft.com)"
  35  *    The word 'cryptographic' can be left out if the rouines from the library
  36  *    being used are not cryptographic related :-).
  37  * 4. If you include any Windows specific code (or a derivative thereof) from
  38  *    the apps directory (application code) you must include an acknowledgement:
  39  *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
  40  *
  41  * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
  42  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  43  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  44  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
  45  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  46  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  47  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  48  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  49  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  50  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  51  * SUCH DAMAGE.
  52  *
  53  * The licence and distribution terms for any publically available version or
  54  * derivative of this code cannot be changed.  i.e. this code cannot simply be
  55  * copied and put under another distribution licence
  56  * [including the GNU Public Licence.]
  57  */
  58 /* ====================================================================
  59  * Copyright (c) 1998-2007 The OpenSSL Project.  All rights reserved.
  60  *
  61  * Redistribution and use in source and binary forms, with or without
```



```
  62  * modification, are permitted provided that the following conditions
  63  * are met:
  64  *
  65  * 1. Redistributions of source code must retain the above copyright
  66  *    notice, this list of conditions and the following disclaimer.
  67  *
  68  * 2. Redistributions in binary form must reproduce the above copyright
  69  *    notice, this list of conditions and the following disclaimer in
  70  *    the documentation and/or other materials provided with the
  71  *    distribution.
  72  *
  73  * 3. All advertising materials mentioning features or use of this
  74  *    software must display the following acknowledgment:
  75  *    "This product includes software developed by the OpenSSL Project
  76  *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
  77  *
  78  * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
  79  *    endorse or promote products derived from this software without
  80  *    prior written permission. For written permission, please contact
  81  *    openssl-core@openssl.org.
  82  *
  83  * 5. Products derived from this software may not be called "OpenSSL"
  84  *    nor may "OpenSSL" appear in their names without prior written
  85  *    permission of the OpenSSL Project.
  86  *
  87  * 6. Redistributions of any form whatsoever must retain the following
  88  *    acknowledgment:
  89  *    "This product includes software developed by the OpenSSL Project
  90  *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
  91  *
  92  * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ''AS IS'' AND ANY
  93  * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  94  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
  95  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
  96  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  97  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
  98  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  99  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 100  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 101  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 102  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 103  * OF THE POSSIBILITY OF SUCH DAMAGE.
 104  * ====================================================================
 105  *
 106  * This product includes cryptographic software written by Eric Young
 107  * (eay@cryptsoft.com).  This product includes software written by Tim
 108  * Hudson (tjh@cryptsoft.com).
 109  *
 110  */
 111 /* ====================================================================
 112  * Copyright 2002 Sun Microsystems, Inc. ALL RIGHTS RESERVED.
 113  *
 114  * Portions of the attached software ("Contribution") are developed by
 115  * SUN MICROSYSTEMS, INC., and are contributed to the OpenSSL project.
 116  *
 117  * The Contribution is licensed pursuant to the OpenSSL open source
 118  * license provided above.
 119  *
 120  * ECC cipher suite support in OpenSSL originally written by
 121  * Vipul Gupta and Sumit Gupta of Sun Microsystems Laboratories.
 122  *
 123  */
 124 /* ====================================================================
 125  * Copyright 2005 Nokia. All rights reserved.
 126  *
 127  * The portions of the attached software ("Contribution") is developed by
```

```
 128  * Nokia Corporation and is licensed pursuant to the OpenSSL open source
 129  * license.
 130  *
 131  * The Contribution, originally written by Mika Kousa and Pasi Eronen of
 132  * Nokia Corporation, consists of the "PSK" (Pre-Shared Key) ciphersuites
 133  * support (see RFC 4279) to OpenSSL.
 134  *
 135  * No patent licenses or other rights except those expressly stated in
 136  * the OpenSSL open source license shall be deemed granted or received
 137  * expressly, by implication, estoppel, or otherwise.
 138  *
 139  * No assurances are provided by Nokia that the Contribution does not
 140  * infringe the patent or other intellectual property rights of any third
 141  * party or that the license provides you with all the necessary rights
 142  * to make use of the Contribution.
 143  *
 144  * THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. IN
 145  * ADDITION TO THE DISCLAIMERS INCLUDED IN THE LICENSE, NOKIA
 146  * SPECIFICALLY DISCLAIMS ANY LIABILITY FOR CLAIMS BROUGHT BY YOU OR ANY
 147  * OTHER ENTITY BASED ON INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OR
 148  * OTHERWISE.
 149  */

 151 #include <stdio.h>
 152 #include <openssl/objects.h>
 153 #include "ssl_locl.h"
 154 #include "kssl_lcl.h"
 155 #ifndef OPENSSL_NO_TLSEXT
 156 #ifndef OPENSSL_NO_EC
 157 #include <ec_lcl.h>
 157 #include "../crypto/ec/ec_lcl.h"
 158 #endif /* OPENSSL_NO_EC */
 159 #endif /* OPENSSL_NO_TLSEXT */
 160 #include <openssl/md5.h>
 161 #ifndef OPENSSL_NO_DH
 162 #include <openssl/dh.h>
 163 #endif

 165 const char ssl3_version_str[]="SSLv3" OPENSSL_VERSION_PTEXT;

 167 #define SSL3_NUM_CIPHERS        (sizeof(ssl3_ciphers)/sizeof(SSL_CIPHER))

 169 /* list of available SSLv3 ciphers (sorted by id) */
 170 OPENSSL_GLOBAL SSL_CIPHER ssl3_ciphers[]={

 172 /* The RSA ciphers */
 173 /* Cipher 01 */
 174         {
 175         1,
 176         SSL3_TXT_RSA_NULL_MD5,
 177         SSL3_CK_RSA_NULL_MD5,
 178         SSL_kRSA,
 179         SSL_aRSA,
 180         SSL_eNULL,
 181         SSL_MD5,
 182         SSL_SSLV3,
 183         SSL_NOT_EXP|SSL_STRONG_NONE,
 184         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
 185         0,
 186         0,
 187         },

 189 /* Cipher 02 */
 190         {
 191         1,
 192         SSL3_TXT_RSA_NULL_SHA,
```



```
 193         SSL3_CK_RSA_NULL_SHA,
 194         SSL_kRSA,
 195         SSL_aRSA,
 196         SSL_eNULL,
 197         SSL_SHA1,
 198         SSL_SSLV3,
 199         SSL_NOT_EXP|SSL_STRONG_NONE|SSL_FIPS,
 200         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
 201         0,
 202         0,
 203         },

 205 /* Cipher 03 */
 206         {
 207         1,
 208         SSL3_TXT_RSA_RC4_40_MD5,
 209         SSL3_CK_RSA_RC4_40_MD5,
 210         SSL_kRSA,
 211         SSL_aRSA,
 212         SSL_RC4,
 213         SSL_MD5,
 214         SSL_SSLV3,
 215         SSL_EXPORT|SSL_EXP40,
 216         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
 217         40,
 218         128,
 219         },

 221 /* Cipher 04 */
 222         {
 223         1,
 224         SSL3_TXT_RSA_RC4_128_MD5,
 225         SSL3_CK_RSA_RC4_128_MD5,
 226         SSL_kRSA,
 227         SSL_aRSA,
 228         SSL_RC4,
 229         SSL_MD5,
 230         SSL_SSLV3,
 231         SSL_NOT_EXP|SSL_MEDIUM,
 232         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
 233         128,
 234         128,
 235         },

 237 /* Cipher 05 */
 238         {
 239         1,
 240         SSL3_TXT_RSA_RC4_128_SHA,
 241         SSL3_CK_RSA_RC4_128_SHA,
 242         SSL_kRSA,
 243         SSL_aRSA,
 244         SSL_RC4,
 245         SSL_SHA1,
 246         SSL_SSLV3,
 247         SSL_NOT_EXP|SSL_MEDIUM,
 248         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
 249         128,
 250         128,
 251         },

 253 /* Cipher 06 */
 254         {
 255         1,
 256         SSL3_TXT_RSA_RC2_40_MD5,
 257         SSL3_CK_RSA_RC2_40_MD5,
 258         SSL_kRSA,
```

```
259         SSL_aRSA,
260         SSL_RC2,
261         SSL_MD5,
262         SSL_SSLV3,
263         SSL_EXPORT|SSL_EXP40,
264         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
265         40,
266         128,
267         },

269 /* Cipher 07 */
270 #ifndef OPENSSL_NO_IDEA
271         {
272         1,
273         SSL3_TXT_RSA_IDEA_128_SHA,
274         SSL3_CK_RSA_IDEA_128_SHA,
275         SSL_kRSA,
276         SSL_aRSA,
277         SSL_IDEA,
278         SSL_SHA1,
279         SSL_SSLV3,
280         SSL_NOT_EXP|SSL_MEDIUM,
281         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
282         128,
283         128,
284         },
285 #endif

287 /* Cipher 08 */
288         {
289         1,
290         SSL3_TXT_RSA_DES_40_CBC_SHA,
291         SSL3_CK_RSA_DES_40_CBC_SHA,
292         SSL_kRSA,
293         SSL_aRSA,
294         SSL_DES,
295         SSL_SHA1,
296         SSL_SSLV3,
297         SSL_EXPORT|SSL_EXP40,
298         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
299         40,
300         56,
301         },

303 /* Cipher 09 */
304         {
305         1,
306         SSL3_TXT_RSA_DES_64_CBC_SHA,
307         SSL3_CK_RSA_DES_64_CBC_SHA,
308         SSL_kRSA,
309         SSL_aRSA,
310         SSL_DES,
311         SSL_SHA1,
312         SSL_SSLV3,
313         SSL_NOT_EXP|SSL_LOW,
314         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
315         56,
316         56,
317         },

319 /* Cipher 0A */
320         {
321         1,
322         SSL3_TXT_RSA_DES_192_CBC3_SHA,
323         SSL3_CK_RSA_DES_192_CBC3_SHA,
324         SSL_kRSA,
```



```
325         SSL_aRSA,
326         SSL_3DES,
327         SSL_SHA1,
328         SSL_SSLV3,
329         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
330         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
331         112,
332         168,
333         },

335 /* The DH ciphers */
336 /* Cipher 0B */
337         {
338         0,
339         SSL3_TXT_DH_DSS_DES_40_CBC_SHA,
340         SSL3_CK_DH_DSS_DES_40_CBC_SHA,
341         SSL_kDHd,
342         SSL_aDH,
343         SSL_DES,
344         SSL_SHA1,
345         SSL_SSLV3,
346         SSL_EXPORT|SSL_EXP40,
347         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
348         40,
349         56,
350         },

352 /* Cipher 0C */
353         {
354         0, /* not implemented (non-ephemeral DH) */
355         SSL3_TXT_DH_DSS_DES_64_CBC_SHA,
356         SSL3_CK_DH_DSS_DES_64_CBC_SHA,
357         SSL_kDHd,
358         SSL_aDH,
359         SSL_DES,
360         SSL_SHA1,
361         SSL_SSLV3,
362         SSL_NOT_EXP|SSL_LOW,
363         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
364         56,
365         56,
366         },

368 /* Cipher 0D */
369         {
370         0, /* not implemented (non-ephemeral DH) */
371         SSL3_TXT_DH_DSS_DES_192_CBC3_SHA,
372         SSL3_CK_DH_DSS_DES_192_CBC3_SHA,
373         SSL_kDHd,
374         SSL_aDH,
375         SSL_3DES,
376         SSL_SHA1,
377         SSL_SSLV3,
378         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
379         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
380         112,
381         168,
382         },

384 /* Cipher 0E */
385         {
386         0, /* not implemented (non-ephemeral DH) */
387         SSL3_TXT_DH_RSA_DES_40_CBC_SHA,
388         SSL3_CK_DH_RSA_DES_40_CBC_SHA,
389         SSL_kDHr,
390         SSL_aDH,
```

```
391         SSL_DES,
392         SSL_SHA1,
393         SSL_SSLV3,
394         SSL_EXPORT|SSL_EXP40,
395         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
396         40,
397         56,
398         },

400 /* Cipher 0F */
401         {
402         0, /* not implemented (non-ephemeral DH) */
403         SSL3_TXT_DH_RSA_DES_64_CBC_SHA,
404         SSL3_CK_DH_RSA_DES_64_CBC_SHA,
405         SSL_kDHr,
406         SSL_aDH,
407         SSL_DES,
408         SSL_SHA1,
409         SSL_SSLV3,
410         SSL_NOT_EXP|SSL_LOW,
411         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
412         56,
413         56,
414         },

416 /* Cipher 10 */
417         {
418         0, /* not implemented (non-ephemeral DH) */
419         SSL3_TXT_DH_RSA_DES_192_CBC3_SHA,
420         SSL3_CK_DH_RSA_DES_192_CBC3_SHA,
421         SSL_kDHr,
422         SSL_aDH,
423         SSL_3DES,
424         SSL_SHA1,
425         SSL_SSLV3,
426         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
427         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
428         112,
429         168,
430         },

432 /* The Ephemeral DH ciphers */
433 /* Cipher 11 */
434         {
435         1,
436         SSL3_TXT_EDH_DSS_DES_40_CBC_SHA,
437         SSL3_CK_EDH_DSS_DES_40_CBC_SHA,
438         SSL_kEDH,
439         SSL_aDSS,
440         SSL_DES,
441         SSL_SHA1,
442         SSL_SSLV3,
443         SSL_EXPORT|SSL_EXP40,
444         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
445         40,
446         56,
447         },

449 /* Cipher 12 */
450         {
451         1,
452         SSL3_TXT_EDH_DSS_DES_64_CBC_SHA,
453         SSL3_CK_EDH_DSS_DES_64_CBC_SHA,
454         SSL_kEDH,
455         SSL_aDSS,
456         SSL_DES,
```



```
457         SSL_SHA1,
458         SSL_SSLV3,
459         SSL_NOT_EXP|SSL_LOW,
460         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
461         56,
462         56,
463         },

465 /* Cipher 13 */
466         {
467         1,
468         SSL3_TXT_EDH_DSS_DES_192_CBC3_SHA,
469         SSL3_CK_EDH_DSS_DES_192_CBC3_SHA,
470         SSL_kEDH,
471         SSL_aDSS,
472         SSL_3DES,
473         SSL_SHA1,
474         SSL_SSLV3,
475         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
476         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
477         112,
478         168,
479         },

481 /* Cipher 14 */
482         {
483         1,
484         SSL3_TXT_EDH_RSA_DES_40_CBC_SHA,
485         SSL3_CK_EDH_RSA_DES_40_CBC_SHA,
486         SSL_kEDH,
487         SSL_aRSA,
488         SSL_DES,
489         SSL_SHA1,
490         SSL_SSLV3,
491         SSL_EXPORT|SSL_EXP40,
492         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
493         40,
494         56,
495         },

497 /* Cipher 15 */
498         {
499         1,
500         SSL3_TXT_EDH_RSA_DES_64_CBC_SHA,
501         SSL3_CK_EDH_RSA_DES_64_CBC_SHA,
502         SSL_kEDH,
503         SSL_aRSA,
504         SSL_DES,
505         SSL_SHA1,
506         SSL_SSLV3,
507         SSL_NOT_EXP|SSL_LOW,
508         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
509         56,
510         56,
511         },

513 /* Cipher 16 */
514         {
515         1,
516         SSL3_TXT_EDH_RSA_DES_192_CBC3_SHA,
517         SSL3_CK_EDH_RSA_DES_192_CBC3_SHA,
518         SSL_kEDH,
519         SSL_aRSA,
520         SSL_3DES,
521         SSL_SHA1,
522         SSL_SSLV3,
```

```
523         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
524         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
525         112,
526         168,
527         },

529 /* Cipher 17 */
530         {
531         1,
532         SSL3_TXT_ADH_RC4_40_MD5,
533         SSL3_CK_ADH_RC4_40_MD5,
534         SSL_kEDH,
535         SSL_aNULL,
536         SSL_RC4,
537         SSL_MD5,
538         SSL_SSLV3,
539         SSL_EXPORT|SSL_EXP40,
540         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
541         40,
542         128,
543         },

545 /* Cipher 18 */
546         {
547         1,
548         SSL3_TXT_ADH_RC4_128_MD5,
549         SSL3_CK_ADH_RC4_128_MD5,
550         SSL_kEDH,
551         SSL_aNULL,
552         SSL_RC4,
553         SSL_MD5,
554         SSL_SSLV3,
555         SSL_NOT_EXP|SSL_MEDIUM,
556         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
557         128,
558         128,
559         },

561 /* Cipher 19 */
562         {
563         1,
564         SSL3_TXT_ADH_DES_40_CBC_SHA,
565         SSL3_CK_ADH_DES_40_CBC_SHA,
566         SSL_kEDH,
567         SSL_aNULL,
568         SSL_DES,
569         SSL_SHA1,
570         SSL_SSLV3,
571         SSL_EXPORT|SSL_EXP40,
572         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
573         40,
574         128,
575         },

577 /* Cipher 1A */
578         {
579         1,
580         SSL3_TXT_ADH_DES_64_CBC_SHA,
581         SSL3_CK_ADH_DES_64_CBC_SHA,
582         SSL_kEDH,
583         SSL_aNULL,
584         SSL_DES,
585         SSL_SHA1,
586         SSL_SSLV3,
587         SSL_NOT_EXP|SSL_LOW,
588         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
```



```
589         56,
590         56,
591         },

593 /* Cipher 1B */
594         {
595         1,
596         SSL3_TXT_ADH_DES_192_CBC_SHA,
597         SSL3_CK_ADH_DES_192_CBC_SHA,
598         SSL_kEDH,
599         SSL_aNULL,
600         SSL_3DES,
601         SSL_SHA1,
602         SSL_SSLV3,
603         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
604         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
605         112,
606         168,
607         },

609 /* Fortezza ciphersuite from SSL 3.0 spec */
610 #if 0
611 /* Cipher 1C */
612         {
613         0,
614         SSL3_TXT_FZA_DMS_NULL_SHA,
615         SSL3_CK_FZA_DMS_NULL_SHA,
616         SSL_kFZA,
617         SSL_aFZA,
618         SSL_eNULL,
619         SSL_SHA1,
620         SSL_SSLV3,
621         SSL_NOT_EXP|SSL_STRONG_NONE,
622         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
623         0,
624         0,
625         },

627 /* Cipher 1D */
628         {
629         0,
630         SSL3_TXT_FZA_DMS_FZA_SHA,
631         SSL3_CK_FZA_DMS_FZA_SHA,
632         SSL_kFZA,
633         SSL_aFZA,
634         SSL_eFZA,
635         SSL_SHA1,
636         SSL_SSLV3,
637         SSL_NOT_EXP|SSL_STRONG_NONE,
638         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
639         0,
640         0,
641         },

643 /* Cipher 1E */
644         {
645         0,
646         SSL3_TXT_FZA_DMS_RC4_SHA,
647         SSL3_CK_FZA_DMS_RC4_SHA,
648         SSL_kFZA,
649         SSL_aFZA,
650         SSL_RC4,
651         SSL_SHA1,
652         SSL_SSLV3,
653         SSL_NOT_EXP|SSL_MEDIUM,
654         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
```

```
655         128,
656         128,
657         },
658 #endif

660 #ifndef OPENSSL_NO_KRB5
661 /* The Kerberos ciphers*/
662 /* Cipher 1E */
663         {
664         1,
665         SSL3_TXT_KRB5_DES_64_CBC_SHA,
666         SSL3_CK_KRB5_DES_64_CBC_SHA,
667         SSL_kKRB5,
668         SSL_aKRB5,
669         SSL_DES,
670         SSL_SHA1,
671         SSL_SSLV3,
672         SSL_NOT_EXP|SSL_LOW,
673         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
674         56,
675         56,
676         },

678 /* Cipher 1F */
679         {
680         1,
681         SSL3_TXT_KRB5_DES_192_CBC3_SHA,
682         SSL3_CK_KRB5_DES_192_CBC3_SHA,
683         SSL_kKRB5,
684         SSL_aKRB5,
685         SSL_3DES,
686         SSL_SHA1,
687         SSL_SSLV3,
688         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
689         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
690         112,
691         168,
692         },

694 /* Cipher 20 */
695         {
696         1,
697         SSL3_TXT_KRB5_RC4_128_SHA,
698         SSL3_CK_KRB5_RC4_128_SHA,
699         SSL_kKRB5,
700         SSL_aKRB5,
701         SSL_RC4,
702         SSL_SHA1,
703         SSL_SSLV3,
704         SSL_NOT_EXP|SSL_MEDIUM,
705         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
706         128,
707         128,
708         },

710 /* Cipher 21 */
711         {
712         1,
713         SSL3_TXT_KRB5_IDEA_128_CBC_SHA,
714         SSL3_CK_KRB5_IDEA_128_CBC_SHA,
715         SSL_kKRB5,
716         SSL_aKRB5,
717         SSL_IDEA,
718         SSL_SHA1,
719         SSL_SSLV3,
720         SSL_NOT_EXP|SSL_MEDIUM,
```



```
721         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
722         128,
723         128,
724         },

726 /* Cipher 22 */
727         {
728         1,
729         SSL3_TXT_KRB5_DES_64_CBC_MD5,
730         SSL3_CK_KRB5_DES_64_CBC_MD5,
731         SSL_kKRB5,
732         SSL_aKRB5,
733         SSL_DES,
734         SSL_MD5,
735         SSL_SSLV3,
736         SSL_NOT_EXP|SSL_LOW,
737         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
738         56,
739         56,
740         },

742 /* Cipher 23 */
743         {
744         1,
745         SSL3_TXT_KRB5_DES_192_CBC3_MD5,
746         SSL3_CK_KRB5_DES_192_CBC3_MD5,
747         SSL_kKRB5,
748         SSL_aKRB5,
749         SSL_3DES,
750         SSL_MD5,
751         SSL_SSLV3,
752         SSL_NOT_EXP|SSL_HIGH,
753         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
754         112,
755         168,
756         },

758 /* Cipher 24 */
759         {
760         1,
761         SSL3_TXT_KRB5_RC4_128_MD5,
762         SSL3_CK_KRB5_RC4_128_MD5,
763         SSL_kKRB5,
764         SSL_aKRB5,
765         SSL_RC4,
766         SSL_MD5,
767         SSL_SSLV3,
768         SSL_NOT_EXP|SSL_MEDIUM,
769         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
770         128,
771         128,
772         },

774 /* Cipher 25 */
775         {
776         1,
777         SSL3_TXT_KRB5_IDEA_128_CBC_MD5,
778         SSL3_CK_KRB5_IDEA_128_CBC_MD5,
779         SSL_kKRB5,
780         SSL_aKRB5,
781         SSL_IDEA,
782         SSL_MD5,
783         SSL_SSLV3,
784         SSL_NOT_EXP|SSL_MEDIUM,
785         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
786         128,
```

```
787         128,
788         },

790 /* Cipher 26 */
791         {
792         1,
793         SSL3_TXT_KRB5_DES_40_CBC_SHA,
794         SSL3_CK_KRB5_DES_40_CBC_SHA,
795         SSL_kKRB5,
796         SSL_aKRB5,
797         SSL_DES,
798         SSL_SHA1,
799         SSL_SSLV3,
800         SSL_EXPORT|SSL_EXP40,
801         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
802         40,
803         56,
804         },

806 /* Cipher 27 */
807         {
808         1,
809         SSL3_TXT_KRB5_RC2_40_CBC_SHA,
810         SSL3_CK_KRB5_RC2_40_CBC_SHA,
811         SSL_kKRB5,
812         SSL_aKRB5,
813         SSL_RC2,
814         SSL_SHA1,
815         SSL_SSLV3,
816         SSL_EXPORT|SSL_EXP40,
817         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
818         40,
819         128,
820         },

822 /* Cipher 28 */
823         {
824         1,
825         SSL3_TXT_KRB5_RC4_40_SHA,
826         SSL3_CK_KRB5_RC4_40_SHA,
827         SSL_kKRB5,
828         SSL_aKRB5,
829         SSL_RC4,
830         SSL_SHA1,
831         SSL_SSLV3,
832         SSL_EXPORT|SSL_EXP40,
833         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
834         40,
835         128,
836         },

838 /* Cipher 29 */
839         {
840         1,
841         SSL3_TXT_KRB5_DES_40_CBC_MD5,
842         SSL3_CK_KRB5_DES_40_CBC_MD5,
843         SSL_kKRB5,
844         SSL_aKRB5,
845         SSL_DES,
846         SSL_MD5,
847         SSL_SSLV3,
848         SSL_EXPORT|SSL_EXP40,
849         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
850         40,
851         56,
852         },
```



```
854 /* Cipher 2A */
855         {
856         1,
857         SSL3_TXT_KRB5_RC2_40_CBC_MD5,
858         SSL3_CK_KRB5_RC2_40_CBC_MD5,
859         SSL_kKRB5,
860         SSL_aKRB5,
861         SSL_RC2,
862         SSL_MD5,
863         SSL_SSLV3,
864         SSL_EXPORT|SSL_EXP40,
865         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
866         40,
867         128,
868         },

870 /* Cipher 2B */
871         {
872         1,
873         SSL3_TXT_KRB5_RC4_40_MD5,
874         SSL3_CK_KRB5_RC4_40_MD5,
875         SSL_kKRB5,
876         SSL_aKRB5,
877         SSL_RC4,
878         SSL_MD5,
879         SSL_SSLV3,
880         SSL_EXPORT|SSL_EXP40,
881         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
882         40,
883         128,
884         },
885 #endif  /* OPENSSL_NO_KRB5 */

887 /* New AES ciphersuites */
888 /* Cipher 2F */
889         {
890         1,
891         TLS1_TXT_RSA_WITH_AES_128_SHA,
892         TLS1_CK_RSA_WITH_AES_128_SHA,
893         SSL_kRSA,
894         SSL_aRSA,
895         SSL_AES128,
896         SSL_SHA1,
897         SSL_TLSV1,
898         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
899         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
900         128,
901         128,
902         },
903 /* Cipher 30 */
904         {
905         0,
906         TLS1_TXT_DH_DSS_WITH_AES_128_SHA,
907         TLS1_CK_DH_DSS_WITH_AES_128_SHA,
908         SSL_kDHd,
909         SSL_aDH,
910         SSL_AES128,
911         SSL_SHA1,
912         SSL_TLSV1,
913         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
914         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
915         128,
916         128,
917         },
918 /* Cipher 31 */
```

```
        {
        0,
        TLS1_TXT_DH_RSA_WITH_AES_128_SHA,
        TLS1_CK_DH_RSA_WITH_AES_128_SHA,
        SSL_kDHr,
        SSL_aDH,
        SSL_AES128,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        128,
        128,
        },
/* Cipher 32 */
        {
        1,
        TLS1_TXT_DHE_DSS_WITH_AES_128_SHA,
        TLS1_CK_DHE_DSS_WITH_AES_128_SHA,
        SSL_kEDH,
        SSL_aDSS,
        SSL_AES128,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        128,
        128,
        },
/* Cipher 33 */
        {
        1,
        TLS1_TXT_DHE_RSA_WITH_AES_128_SHA,
        TLS1_CK_DHE_RSA_WITH_AES_128_SHA,
        SSL_kEDH,
        SSL_aRSA,
        SSL_AES128,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        128,
        128,
        },
/* Cipher 34 */
        {
        1,
        TLS1_TXT_ADH_WITH_AES_128_SHA,
        TLS1_CK_ADH_WITH_AES_128_SHA,
        SSL_kEDH,
        SSL_aNULL,
        SSL_AES128,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        128,
        128,
        },

/* Cipher 35 */
        {
        1,
        TLS1_TXT_RSA_WITH_AES_256_SHA,
        TLS1_CK_RSA_WITH_AES_256_SHA,
        SSL_kRSA,
```



```
        SSL_aRSA,
        SSL_AES256,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        256,
        256,
        },
/* Cipher 36 */
        {
        0,
        TLS1_TXT_DH_DSS_WITH_AES_256_SHA,
        TLS1_CK_DH_DSS_WITH_AES_256_SHA,
        SSL_kDHd,
        SSL_aDH,
        SSL_AES256,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        256,
        256,
        },

/* Cipher 37 */
        {
        0, /* not implemented (non-ephemeral DH) */
        TLS1_TXT_DH_RSA_WITH_AES_256_SHA,
        TLS1_CK_DH_RSA_WITH_AES_256_SHA,
        SSL_kDHr,
        SSL_aDH,
        SSL_AES256,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        256,
        256,
        },

/* Cipher 38 */
        {
        1,
        TLS1_TXT_DHE_DSS_WITH_AES_256_SHA,
        TLS1_CK_DHE_DSS_WITH_AES_256_SHA,
        SSL_kEDH,
        SSL_aDSS,
        SSL_AES256,
        SSL_SHA1,
        SSL_TLSV1,
        SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
        SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
        256,
        256,
        },

/* Cipher 39 */
        {
        1,
        TLS1_TXT_DHE_RSA_WITH_AES_256_SHA,
        TLS1_CK_DHE_RSA_WITH_AES_256_SHA,
        SSL_kEDH,
        SSL_aRSA,
        SSL_AES256,
        SSL_SHA1,
```

```
1051         SSL_TLSV1,
1052         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1053         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1054         256,
1055         256,
1056         },

1058         /* Cipher 3A */
1059         {
1060         1,
1061         TLS1_TXT_ADH_WITH_AES_256_SHA,
1062         TLS1_CK_ADH_WITH_AES_256_SHA,
1063         SSL_kEDH,
1064         SSL_aNULL,
1065         SSL_AES256,
1066         SSL_SHA1,
1067         SSL_TLSV1,
1068         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1069         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1070         256,
1071         256,
1072         },

1074         /* TLS v1.2 ciphersuites */
1075         /* Cipher 3B */
1076         {
1077         1,
1078         TLS1_TXT_RSA_WITH_NULL_SHA256,
1079         TLS1_CK_RSA_WITH_NULL_SHA256,
1080         SSL_kRSA,
1081         SSL_aRSA,
1082         SSL_eNULL,
1083         SSL_SHA256,
1084         SSL_TLSV1_2,
1085         SSL_NOT_EXP|SSL_STRONG_NONE|SSL_FIPS,
1086         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1087         0,
1088         0,
1089         },

1091         /* Cipher 3C */
1092         {
1093         1,
1094         TLS1_TXT_RSA_WITH_AES_128_SHA256,
1095         TLS1_CK_RSA_WITH_AES_128_SHA256,
1096         SSL_kRSA,
1097         SSL_aRSA,
1098         SSL_AES128,
1099         SSL_SHA256,
1100         SSL_TLSV1_2,
1101         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1102         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1103         128,
1104         128,
1105         },

1107         /* Cipher 3D */
1108         {
1109         1,
1110         TLS1_TXT_RSA_WITH_AES_256_SHA256,
1111         TLS1_CK_RSA_WITH_AES_256_SHA256,
1112         SSL_kRSA,
1113         SSL_aRSA,
1114         SSL_AES256,
1115         SSL_SHA256,
1116         SSL_TLSV1_2,
```



```
1117         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1118         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1119         256,
1120         256,
1121         },

1123         /* Cipher 3E */
1124         {
1125         0, /* not implemented (non-ephemeral DH) */
1126         TLS1_TXT_DH_DSS_WITH_AES_128_SHA256,
1127         TLS1_CK_DH_DSS_WITH_AES_128_SHA256,
1128         SSL_kDHd,
1129         SSL_aDH,
1130         SSL_AES128,
1131         SSL_SHA256,
1132         SSL_TLSV1_2,
1133         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1134         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1135         128,
1136         128,
1137         },

1139         /* Cipher 3F */
1140         {
1141         0, /* not implemented (non-ephemeral DH) */
1142         TLS1_TXT_DH_RSA_WITH_AES_128_SHA256,
1143         TLS1_CK_DH_RSA_WITH_AES_128_SHA256,
1144         SSL_kDHr,
1145         SSL_aDH,
1146         SSL_AES128,
1147         SSL_SHA256,
1148         SSL_TLSV1_2,
1149         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1150         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1151         128,
1152         128,
1153         },

1155         /* Cipher 40 */
1156         {
1157         1,
1158         TLS1_TXT_DHE_DSS_WITH_AES_128_SHA256,
1159         TLS1_CK_DHE_DSS_WITH_AES_128_SHA256,
1160         SSL_kEDH,
1161         SSL_aDSS,
1162         SSL_AES128,
1163         SSL_SHA256,
1164         SSL_TLSV1_2,
1165         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1166         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1167         128,
1168         128,
1169         },

1171 #ifndef OPENSSL_NO_CAMELLIA
1172         /* Camellia ciphersuites from RFC4132 (128-bit portion) */

1174         /* Cipher 41 */
1175         {
1176         1,
1177         TLS1_TXT_RSA_WITH_CAMELLIA_128_CBC_SHA,
1178         TLS1_CK_RSA_WITH_CAMELLIA_128_CBC_SHA,
1179         SSL_kRSA,
1180         SSL_aRSA,
1181         SSL_CAMELLIA128,
1182         SSL_SHA1,
```

```
1183         SSL_TLSV1,
1184         SSL_NOT_EXP|SSL_HIGH,
1185         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1186         128,
1187         128,
1188         },

1190         /* Cipher 42 */
1191         {
1192         0, /* not implemented (non-ephemeral DH) */
1193         TLS1_TXT_DH_DSS_WITH_CAMELLIA_128_CBC_SHA,
1194         TLS1_CK_DH_DSS_WITH_CAMELLIA_128_CBC_SHA,
1195         SSL_kDHd,
1196         SSL_aDH,
1197         SSL_CAMELLIA128,
1198         SSL_SHA1,
1199         SSL_TLSV1,
1200         SSL_NOT_EXP|SSL_HIGH,
1201         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1202         128,
1203         128,
1204         },

1206         /* Cipher 43 */
1207         {
1208         0, /* not implemented (non-ephemeral DH) */
1209         TLS1_TXT_DH_RSA_WITH_CAMELLIA_128_CBC_SHA,
1210         TLS1_CK_DH_RSA_WITH_CAMELLIA_128_CBC_SHA,
1211         SSL_kDHr,
1212         SSL_aDH,
1213         SSL_CAMELLIA128,
1214         SSL_SHA1,
1215         SSL_TLSV1,
1216         SSL_NOT_EXP|SSL_HIGH,
1217         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1218         128,
1219         128,
1220         },

1222         /* Cipher 44 */
1223         {
1224         1,
1225         TLS1_TXT_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA,
1226         TLS1_CK_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA,
1227         SSL_kEDH,
1228         SSL_aDSS,
1229         SSL_CAMELLIA128,
1230         SSL_SHA1,
1231         SSL_TLSV1,
1232         SSL_NOT_EXP|SSL_HIGH,
1233         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1234         128,
1235         128,
1236         },

1238         /* Cipher 45 */
1239         {
1240         1,
1241         TLS1_TXT_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA,
1242         TLS1_CK_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA,
1243         SSL_kEDH,
1244         SSL_aRSA,
1245         SSL_CAMELLIA128,
1246         SSL_SHA1,
1247         SSL_TLSV1,
1248         SSL_NOT_EXP|SSL_HIGH,
```



```
1249         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1250         128,
1251         128,
1252         },

1254         /* Cipher 46 */
1255         {
1256         1,
1257         TLS1_TXT_ADH_WITH_CAMELLIA_128_CBC_SHA,
1258         TLS1_CK_ADH_WITH_CAMELLIA_128_CBC_SHA,
1259         SSL_kEDH,
1260         SSL_aNULL,
1261         SSL_CAMELLIA128,
1262         SSL_SHA1,
1263         SSL_TLSV1,
1264         SSL_NOT_EXP|SSL_HIGH,
1265         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1266         128,
1267         128,
1268         },
1269 #endif /* OPENSSL_NO_CAMELLIA */

1271 #if TLS1_ALLOW_EXPERIMENTAL_CIPHERSUITES
1272         /* New TLS Export CipherSuites from expired ID */
1273 #if 0
1274         /* Cipher 60 */
1275         {
1276         1,
1277         TLS1_TXT_RSA_EXPORT1024_WITH_RC4_56_MD5,
1278         TLS1_CK_RSA_EXPORT1024_WITH_RC4_56_MD5,
1279         SSL_kRSA,
1280         SSL_aRSA,
1281         SSL_RC4,
1282         SSL_MD5,
1283         SSL_TLSV1,
1284         SSL_EXPORT|SSL_EXP56,
1285         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1286         56,
1287         128,
1288         },

1290         /* Cipher 61 */
1291         {
1292         1,
1293         TLS1_TXT_RSA_EXPORT1024_WITH_RC2_CBC_56_MD5,
1294         TLS1_CK_RSA_EXPORT1024_WITH_RC2_CBC_56_MD5,
1295         SSL_kRSA,
1296         SSL_aRSA,
1297         SSL_RC2,
1298         SSL_MD5,
1299         SSL_TLSV1,
1300         SSL_EXPORT|SSL_EXP56,
1301         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1302         56,
1303         128,
1304         },
1305 #endif

1307         /* Cipher 62 */
1308         {
1309         1,
1310         TLS1_TXT_RSA_EXPORT1024_WITH_DES_CBC_SHA,
1311         TLS1_CK_RSA_EXPORT1024_WITH_DES_CBC_SHA,
1312         SSL_kRSA,
1313         SSL_aRSA,
1314         SSL_DES,
```

```
1315         SSL_SHA1,
1316         SSL_TLSV1,
1317         SSL_EXPORT|SSL_EXP56,
1318         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1319         56,
1320         56,
1321         },

1323         /* Cipher 63 */
1324         {
1325         1,
1326         TLS1_TXT_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA,
1327         TLS1_CK_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA,
1328         SSL_kEDH,
1329         SSL_aDSS,
1330         SSL_DES,
1331         SSL_SHA1,
1332         SSL_TLSV1,
1333         SSL_EXPORT|SSL_EXP56,
1334         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1335         56,
1336         56,
1337         },

1339         /* Cipher 64 */
1340         {
1341         1,
1342         TLS1_TXT_RSA_EXPORT1024_WITH_RC4_56_SHA,
1343         TLS1_CK_RSA_EXPORT1024_WITH_RC4_56_SHA,
1344         SSL_kRSA,
1345         SSL_aRSA,
1346         SSL_RC4,
1347         SSL_SHA1,
1348         SSL_TLSV1,
1349         SSL_EXPORT|SSL_EXP56,
1350         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1351         56,
1352         128,
1353         },

1355         /* Cipher 65 */
1356         {
1357         1,
1358         TLS1_TXT_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA,
1359         TLS1_CK_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA,
1360         SSL_kEDH,
1361         SSL_aDSS,
1362         SSL_RC4,
1363         SSL_SHA1,
1364         SSL_TLSV1,
1365         SSL_EXPORT|SSL_EXP56,
1366         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1367         56,
1368         128,
1369         },

1371         /* Cipher 66 */
1372         {
1373         1,
1374         TLS1_TXT_DHE_DSS_WITH_RC4_128_SHA,
1375         TLS1_CK_DHE_DSS_WITH_RC4_128_SHA,
1376         SSL_kEDH,
1377         SSL_aDSS,
1378         SSL_RC4,
1379         SSL_SHA1,
1380         SSL_TLSV1,
```



```
1381         SSL_NOT_EXP|SSL_MEDIUM,
1382         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1383         128,
1384         128,
1385         },
1386 #endif

1388         /* TLS v1.2 ciphersuites */
1389         /* Cipher 67 */
1390         {
1391         1,
1392         TLS1_TXT_DHE_RSA_WITH_AES_128_SHA256,
1393         TLS1_CK_DHE_RSA_WITH_AES_128_SHA256,
1394         SSL_kEDH,
1395         SSL_aRSA,
1396         SSL_AES128,
1397         SSL_SHA256,
1398         SSL_TLSV1_2,
1399         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1400         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1401         128,
1402         128,
1403         },

1405         /* Cipher 68 */
1406         {
1407         0, /* not implemented (non-ephemeral DH) */
1408         TLS1_TXT_DH_DSS_WITH_AES_256_SHA256,
1409         TLS1_CK_DH_DSS_WITH_AES_256_SHA256,
1410         SSL_kDHd,
1411         SSL_aDH,
1412         SSL_AES256,
1413         SSL_SHA256,
1414         SSL_TLSV1_2,
1415         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1416         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1417         256,
1418         256,
1419         },

1421         /* Cipher 69 */
1422         {
1423         0, /* not implemented (non-ephemeral DH) */
1424         TLS1_TXT_DH_RSA_WITH_AES_256_SHA256,
1425         TLS1_CK_DH_RSA_WITH_AES_256_SHA256,
1426         SSL_kDHr,
1427         SSL_aDH,
1428         SSL_AES256,
1429         SSL_SHA256,
1430         SSL_TLSV1_2,
1431         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1432         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1433         256,
1434         256,
1435         },

1437         /* Cipher 6A */
1438         {
1439         1,
1440         TLS1_TXT_DHE_DSS_WITH_AES_256_SHA256,
1441         TLS1_CK_DHE_DSS_WITH_AES_256_SHA256,
1442         SSL_kEDH,
1443         SSL_aDSS,
1444         SSL_AES256,
1445         SSL_SHA256,
1446         SSL_TLSV1_2,
```

```
1447         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1448         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1449         256,
1450         256,
1451         },

1453         /* Cipher 6B */
1454         {
1455         1,
1456         TLS1_TXT_DHE_RSA_WITH_AES_256_SHA256,
1457         TLS1_CK_DHE_RSA_WITH_AES_256_SHA256,
1458         SSL_kEDH,
1459         SSL_aRSA,
1460         SSL_AES256,
1461         SSL_SHA256,
1462         SSL_TLSV1_2,
1463         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1464         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1465         256,
1466         256,
1467         },

1469         /* Cipher 6C */
1470         {
1471         1,
1472         TLS1_TXT_ADH_WITH_AES_128_SHA256,
1473         TLS1_CK_ADH_WITH_AES_128_SHA256,
1474         SSL_kEDH,
1475         SSL_aNULL,
1476         SSL_AES128,
1477         SSL_SHA256,
1478         SSL_TLSV1_2,
1479         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1480         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1481         128,
1482         128,
1483         },

1485         /* Cipher 6D */
1486         {
1487         1,
1488         TLS1_TXT_ADH_WITH_AES_256_SHA256,
1489         TLS1_CK_ADH_WITH_AES_256_SHA256,
1490         SSL_kEDH,
1491         SSL_aNULL,
1492         SSL_AES256,
1493         SSL_SHA256,
1494         SSL_TLSV1_2,
1495         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1496         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1497         256,
1498         256,
1499         },

1501         /* GOST Ciphersuites */

1503         {
1504         1,
1505         "GOST94-GOST89-GOST89",
1506         0x3000080,
1507         SSL_kGOST,
1508         SSL_aGOST94,
1509         SSL_eGOST2814789CNT,
1510         SSL_GOST89MAC,
1511         SSL_TLSV1,
1512         SSL_NOT_EXP|SSL_HIGH,
```



```
1513         SSL_HANDSHAKE_MAC_GOST94|TLS1_PRF_GOST94|TLS1_STREAM_MAC,
1514         256,
1515         256
1516         },
1517         {
1518         1,
1519         "GOST2001-GOST89-GOST89",
1520         0x3000081,
1521         SSL_kGOST,
1522         SSL_aGOST01,
1523         SSL_eGOST2814789CNT,
1524         SSL_GOST89MAC,
1525         SSL_TLSV1,
1526         SSL_NOT_EXP|SSL_HIGH,
1527         SSL_HANDSHAKE_MAC_GOST94|TLS1_PRF_GOST94|TLS1_STREAM_MAC,
1528         256,
1529         256
1530         },
1531         {
1532         1,
1533         "GOST94-NULL-GOST94",
1534         0x3000082,
1535         SSL_kGOST,
1536         SSL_aGOST94,
1537         SSL_eNULL,
1538         SSL_GOST94,
1539         SSL_TLSV1,
1540         SSL_NOT_EXP|SSL_STRONG_NONE,
1541         SSL_HANDSHAKE_MAC_GOST94|TLS1_PRF_GOST94,
1542         0,
1543         0
1544         },
1545         {
1546         1,
1547         "GOST2001-NULL-GOST94",
1548         0x3000083,
1549         SSL_kGOST,
1550         SSL_aGOST01,
1551         SSL_eNULL,
1552         SSL_GOST94,
1553         SSL_TLSV1,
1554         SSL_NOT_EXP|SSL_STRONG_NONE,
1555         SSL_HANDSHAKE_MAC_GOST94|TLS1_PRF_GOST94,
1556         0,
1557         0
1558         },

1560 #ifndef OPENSSL_NO_CAMELLIA
1561         /* Camellia ciphersuites from RFC4132 (256-bit portion) */

1563         /* Cipher 84 */
1564         {
1565         1,
1566         TLS1_TXT_RSA_WITH_CAMELLIA_256_CBC_SHA,
1567         TLS1_CK_RSA_WITH_CAMELLIA_256_CBC_SHA,
1568         SSL_kRSA,
1569         SSL_aRSA,
1570         SSL_CAMELLIA256,
1571         SSL_SHA1,
1572         SSL_TLSV1,
1573         SSL_NOT_EXP|SSL_HIGH,
1574         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1575         256,
1576         256,
1577         },
1578         /* Cipher 85 */
```

```
1579         {
1580         0, /* not implemented (non-ephemeral DH) */
1581         TLS1_TXT_DH_DSS_WITH_CAMELLIA_256_CBC_SHA,
1582         TLS1_CK_DH_DSS_WITH_CAMELLIA_256_CBC_SHA,
1583         SSL_kDHd,
1584         SSL_aDH,
1585         SSL_CAMELLIA256,
1586         SSL_SHA1,
1587         SSL_TLSV1,
1588         SSL_NOT_EXP|SSL_HIGH,
1589         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1590         256,
1591         256,
1592         },

1594         /* Cipher 86 */
1595         {
1596         0, /* not implemented (non-ephemeral DH) */
1597         TLS1_TXT_DH_RSA_WITH_CAMELLIA_256_CBC_SHA,
1598         TLS1_CK_DH_RSA_WITH_CAMELLIA_256_CBC_SHA,
1599         SSL_kDHr,
1600         SSL_aDH,
1601         SSL_CAMELLIA256,
1602         SSL_SHA1,
1603         SSL_TLSV1,
1604         SSL_NOT_EXP|SSL_HIGH,
1605         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1606         256,
1607         256,
1608         },

1610         /* Cipher 87 */
1611         {
1612         1,
1613         TLS1_TXT_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA,
1614         TLS1_CK_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA,
1615         SSL_kEDH,
1616         SSL_aDSS,
1617         SSL_CAMELLIA256,
1618         SSL_SHA1,
1619         SSL_TLSV1,
1620         SSL_NOT_EXP|SSL_HIGH,
1621         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1622         256,
1623         256,
1624         },

1626         /* Cipher 88 */
1627         {
1628         1,
1629         TLS1_TXT_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA,
1630         TLS1_CK_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA,
1631         SSL_kEDH,
1632         SSL_aRSA,
1633         SSL_CAMELLIA256,
1634         SSL_SHA1,
1635         SSL_TLSV1,
1636         SSL_NOT_EXP|SSL_HIGH,
1637         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1638         256,
1639         256,
1640         },

1642         /* Cipher 89 */
1643         {
1644         1,
```



```
1645         TLS1_TXT_ADH_WITH_CAMELLIA_256_CBC_SHA,
1646         TLS1_CK_ADH_WITH_CAMELLIA_256_CBC_SHA,
1647         SSL_kEDH,
1648         SSL_aNULL,
1649         SSL_CAMELLIA256,
1650         SSL_SHA1,
1651         SSL_TLSV1,
1652         SSL_NOT_EXP|SSL_HIGH,
1653         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1654         256,
1655         256,
1656         },
1657 #endif /* OPENSSL_NO_CAMELLIA */

1659 #ifndef OPENSSL_NO_PSK
1660         /* Cipher 8A */
1661         {
1662         1,
1663         TLS1_TXT_PSK_WITH_RC4_128_SHA,
1664         TLS1_CK_PSK_WITH_RC4_128_SHA,
1665         SSL_kPSK,
1666         SSL_aPSK,
1667         SSL_RC4,
1668         SSL_SHA1,
1669         SSL_TLSV1,
1670         SSL_NOT_EXP|SSL_MEDIUM,
1671         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1672         128,
1673         128,
1674         },

1676         /* Cipher 8B */
1677         {
1678         1,
1679         TLS1_TXT_PSK_WITH_3DES_EDE_CBC_SHA,
1680         TLS1_CK_PSK_WITH_3DES_EDE_CBC_SHA,
1681         SSL_kPSK,
1682         SSL_aPSK,
1683         SSL_3DES,
1684         SSL_SHA1,
1685         SSL_TLSV1,
1686         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1687         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1688         112,
1689         168,
1690         },

1692         /* Cipher 8C */
1693         {
1694         1,
1695         TLS1_TXT_PSK_WITH_AES_128_CBC_SHA,
1696         TLS1_CK_PSK_WITH_AES_128_CBC_SHA,
1697         SSL_kPSK,
1698         SSL_aPSK,
1699         SSL_AES128,
1700         SSL_SHA1,
1701         SSL_TLSV1,
1702         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1703         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1704         128,
1705         128,
1706         },

1708         /* Cipher 8D */
1709         {
1710         1,
```

```
1711         TLS1_TXT_PSK_WITH_AES_256_CBC_SHA,
1712         TLS1_CK_PSK_WITH_AES_256_CBC_SHA,
1713         SSL_kPSK,
1714         SSL_aPSK,
1715         SSL_AES256,
1716         SSL_SHA1,
1717         SSL_TLSV1,
1718         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1719         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1720         256,
1721         256,
1722         },
1723 #endif  /* OPENSSL_NO_PSK */

1725 #ifndef OPENSSL_NO_SEED
1726         /* SEED ciphersuites from RFC4162 */

1728         /* Cipher 96 */
1729         {
1730         1,
1731         TLS1_TXT_RSA_WITH_SEED_SHA,
1732         TLS1_CK_RSA_WITH_SEED_SHA,
1733         SSL_kRSA,
1734         SSL_aRSA,
1735         SSL_SEED,
1736         SSL_SHA1,
1737         SSL_TLSV1,
1738         SSL_NOT_EXP|SSL_MEDIUM,
1739         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1740         128,
1741         128,
1742         },

1744         /* Cipher 97 */
1745         {
1746         0, /* not implemented (non-ephemeral DH) */
1747         TLS1_TXT_DH_DSS_WITH_SEED_SHA,
1748         TLS1_CK_DH_DSS_WITH_SEED_SHA,
1749         SSL_kDHd,
1750         SSL_aDH,
1751         SSL_SEED,
1752         SSL_SHA1,
1753         SSL_TLSV1,
1754         SSL_NOT_EXP|SSL_MEDIUM,
1755         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1756         128,
1757         128,
1758         },

1760         /* Cipher 98 */
1761         {
1762         0, /* not implemented (non-ephemeral DH) */
1763         TLS1_TXT_DH_RSA_WITH_SEED_SHA,
1764         TLS1_CK_DH_RSA_WITH_SEED_SHA,
1765         SSL_kDHr,
1766         SSL_aDH,
1767         SSL_SEED,
1768         SSL_SHA1,
1769         SSL_TLSV1,
1770         SSL_NOT_EXP|SSL_MEDIUM,
1771         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1772         128,
1773         128,
1774         },

1776         /* Cipher 99 */
```



```
1777         {
1778         1,
1779         TLS1_TXT_DHE_DSS_WITH_SEED_SHA,
1780         TLS1_CK_DHE_DSS_WITH_SEED_SHA,
1781         SSL_kEDH,
1782         SSL_aDSS,
1783         SSL_SEED,
1784         SSL_SHA1,
1785         SSL_TLSV1,
1786         SSL_NOT_EXP|SSL_MEDIUM,
1787         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1788         128,
1789         128,
1790         },

1792         /* Cipher 9A */
1793         {
1794         1,
1795         TLS1_TXT_DHE_RSA_WITH_SEED_SHA,
1796         TLS1_CK_DHE_RSA_WITH_SEED_SHA,
1797         SSL_kEDH,
1798         SSL_aRSA,
1799         SSL_SEED,
1800         SSL_SHA1,
1801         SSL_TLSV1,
1802         SSL_NOT_EXP|SSL_MEDIUM,
1803         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1804         128,
1805         128,
1806         },

1808         /* Cipher 9B */
1809         {
1810         1,
1811         TLS1_TXT_ADH_WITH_SEED_SHA,
1812         TLS1_CK_ADH_WITH_SEED_SHA,
1813         SSL_kEDH,
1814         SSL_aNULL,
1815         SSL_SEED,
1816         SSL_SHA1,
1817         SSL_TLSV1,
1818         SSL_NOT_EXP|SSL_MEDIUM,
1819         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
1820         128,
1821         128,
1822         },

1824 #endif /* OPENSSL_NO_SEED */

1826         /* GCM ciphersuites from RFC5288 */

1828         /* Cipher 9C */
1829         {
1830         1,
1831         TLS1_TXT_RSA_WITH_AES_128_GCM_SHA256,
1832         TLS1_CK_RSA_WITH_AES_128_GCM_SHA256,
1833         SSL_kRSA,
1834         SSL_aRSA,
1835         SSL_AES128GCM,
1836         SSL_AEAD,
1837         SSL_TLSV1_2,
1838         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1839         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
1840         128,
1841         128,
1842         },
```

```
1844         /* Cipher 9D */
1845         {
1846         1,
1847         TLS1_TXT_RSA_WITH_AES_256_GCM_SHA384,
1848         TLS1_CK_RSA_WITH_AES_256_GCM_SHA384,
1849         SSL_kRSA,
1850         SSL_aRSA,
1851         SSL_AES256GCM,
1852         SSL_AEAD,
1853         SSL_TLSV1_2,
1854         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1855         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
1856         256,
1857         256,
1858         },

1860         /* Cipher 9E */
1861         {
1862         1,
1863         TLS1_TXT_DHE_RSA_WITH_AES_128_GCM_SHA256,
1864         TLS1_CK_DHE_RSA_WITH_AES_128_GCM_SHA256,
1865         SSL_kEDH,
1866         SSL_aRSA,
1867         SSL_AES128GCM,
1868         SSL_AEAD,
1869         SSL_TLSV1_2,
1870         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1871         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
1872         128,
1873         128,
1874         },

1876         /* Cipher 9F */
1877         {
1878         1,
1879         TLS1_TXT_DHE_RSA_WITH_AES_256_GCM_SHA384,
1880         TLS1_CK_DHE_RSA_WITH_AES_256_GCM_SHA384,
1881         SSL_kEDH,
1882         SSL_aRSA,
1883         SSL_AES256GCM,
1884         SSL_AEAD,
1885         SSL_TLSV1_2,
1886         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1887         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
1888         256,
1889         256,
1890         },

1892         /* Cipher A0 */
1893         {
1894         0,
1895         TLS1_TXT_DH_RSA_WITH_AES_128_GCM_SHA256,
1896         TLS1_CK_DH_RSA_WITH_AES_128_GCM_SHA256,
1897         SSL_kDHr,
1898         SSL_aDH,
1899         SSL_AES128GCM,
1900         SSL_AEAD,
1901         SSL_TLSV1_2,
1902         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1903         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
1904         128,
1905         128,
1906         },

1908         /* Cipher A1 */
```



```
1909         {
1910         0,
1911         TLS1_TXT_DH_RSA_WITH_AES_256_GCM_SHA384,
1912         TLS1_CK_DH_RSA_WITH_AES_256_GCM_SHA384,
1913         SSL_kDHr,
1914         SSL_aDH,
1915         SSL_AES256GCM,
1916         SSL_AEAD,
1917         SSL_TLSV1_2,
1918         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1919         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
1920         256,
1921         256,
1922         },

1924         /* Cipher A2 */
1925         {
1926         1,
1927         TLS1_TXT_DHE_DSS_WITH_AES_128_GCM_SHA256,
1928         TLS1_CK_DHE_DSS_WITH_AES_128_GCM_SHA256,
1929         SSL_kEDH,
1930         SSL_aDSS,
1931         SSL_AES128GCM,
1932         SSL_AEAD,
1933         SSL_TLSV1_2,
1934         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1935         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
1936         128,
1937         128,
1938         },

1940         /* Cipher A3 */
1941         {
1942         1,
1943         TLS1_TXT_DHE_DSS_WITH_AES_256_GCM_SHA384,
1944         TLS1_CK_DHE_DSS_WITH_AES_256_GCM_SHA384,
1945         SSL_kEDH,
1946         SSL_aDSS,
1947         SSL_AES256GCM,
1948         SSL_AEAD,
1949         SSL_TLSV1_2,
1950         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1951         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
1952         256,
1953         256,
1954         },

1956         /* Cipher A4 */
1957         {
1958         0,
1959         TLS1_TXT_DH_DSS_WITH_AES_128_GCM_SHA256,
1960         TLS1_CK_DH_DSS_WITH_AES_128_GCM_SHA256,
1961         SSL_kDHd,
1962         SSL_aDH,
1963         SSL_AES128GCM,
1964         SSL_AEAD,
1965         SSL_TLSV1_2,
1966         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1967         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
1968         128,
1969         128,
1970         },

1972         /* Cipher A5 */
1973         {
1974         0,
```

```
1975         TLS1_TXT_DH_DSS_WITH_AES_256_GCM_SHA384,
1976         TLS1_CK_DH_DSS_WITH_AES_256_GCM_SHA384,
1977         SSL_kDHd,
1978         SSL_aDH,
1979         SSL_AES256GCM,
1980         SSL_AEAD,
1981         SSL_TLSV1_2,
1982         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1983         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
1984         256,
1985         256,
1986         },

1988         /* Cipher A6 */
1989         {
1990         1,
1991         TLS1_TXT_ADH_WITH_AES_128_GCM_SHA256,
1992         TLS1_CK_ADH_WITH_AES_128_GCM_SHA256,
1993         SSL_kEDH,
1994         SSL_aNULL,
1995         SSL_AES128GCM,
1996         SSL_AEAD,
1997         SSL_TLSV1_2,
1998         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
1999         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2000         128,
2001         128,
2002         },

2004         /* Cipher A7 */
2005         {
2006         1,
2007         TLS1_TXT_ADH_WITH_AES_256_GCM_SHA384,
2008         TLS1_CK_ADH_WITH_AES_256_GCM_SHA384,
2009         SSL_kEDH,
2010         SSL_aNULL,
2011         SSL_AES256GCM,
2012         SSL_AEAD,
2013         SSL_TLSV1_2,
2014         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2015         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2016         256,
2017         256,
2018         },

2020 #ifndef OPENSSL_NO_ECDH
2021         /* Cipher C001 */
2022         {
2023         1,
2024         TLS1_TXT_ECDH_ECDSA_WITH_NULL_SHA,
2025         TLS1_CK_ECDH_ECDSA_WITH_NULL_SHA,
2026         SSL_kECDHe,
2027         SSL_aECDH,
2028         SSL_eNULL,
2029         SSL_SHA1,
2030         SSL_TLSV1,
2031         SSL_NOT_EXP|SSL_STRONG_NONE|SSL_FIPS,
2032         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2033         0,
2034         0,
2035         },

2037         /* Cipher C002 */
2038         {
2039         1,
2040         TLS1_TXT_ECDH_ECDSA_WITH_RC4_128_SHA,
```



```
2041         TLS1_CK_ECDH_ECDSA_WITH_RC4_128_SHA,
2042         SSL_kECDHe,
2043         SSL_aECDH,
2044         SSL_RC4,
2045         SSL_SHA1,
2046         SSL_TLSV1,
2047         SSL_NOT_EXP|SSL_MEDIUM,
2048         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2049         128,
2050         128,
2051         },

2053         /* Cipher C003 */
2054         {
2055         1,
2056         TLS1_TXT_ECDH_ECDSA_WITH_DES_192_CBC3_SHA,
2057         TLS1_CK_ECDH_ECDSA_WITH_DES_192_CBC3_SHA,
2058         SSL_kECDHe,
2059         SSL_aECDH,
2060         SSL_3DES,
2061         SSL_SHA1,
2062         SSL_TLSV1,
2063         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2064         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2065         112,
2066         168,
2067         },

2069         /* Cipher C004 */
2070         {
2071         1,
2072         TLS1_TXT_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
2073         TLS1_CK_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
2074         SSL_kECDHe,
2075         SSL_aECDH,
2076         SSL_AES128,
2077         SSL_SHA1,
2078         SSL_TLSV1,
2079         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2080         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2081         128,
2082         128,
2083         },

2085         /* Cipher C005 */
2086         {
2087         1,
2088         TLS1_TXT_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
2089         TLS1_CK_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
2090         SSL_kECDHe,
2091         SSL_aECDH,
2092         SSL_AES256,
2093         SSL_SHA1,
2094         SSL_TLSV1,
2095         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2096         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2097         256,
2098         256,
2099         },

2101         /* Cipher C006 */
2102         {
2103         1,
2104         TLS1_TXT_ECDHE_ECDSA_WITH_NULL_SHA,
2105         TLS1_CK_ECDHE_ECDSA_WITH_NULL_SHA,
2106         SSL_kEECDH,
```

```
2107         SSL_aECDSA,
2108         SSL_eNULL,
2109         SSL_SHA1,
2110         SSL_TLSV1,
2111         SSL_NOT_EXP|SSL_STRONG_NONE|SSL_FIPS,
2112         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2113         0,
2114         0,
2115         },

2117         /* Cipher C007 */
2118         {
2119         1,
2120         TLS1_TXT_ECDHE_ECDSA_WITH_RC4_128_SHA,
2121         TLS1_CK_ECDHE_ECDSA_WITH_RC4_128_SHA,
2122         SSL_kEECDH,
2123         SSL_aECDSA,
2124         SSL_RC4,
2125         SSL_SHA1,
2126         SSL_TLSV1,
2127         SSL_NOT_EXP|SSL_MEDIUM,
2128         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2129         128,
2130         128,
2131         },

2133         /* Cipher C008 */
2134         {
2135         1,
2136         TLS1_TXT_ECDHE_ECDSA_WITH_DES_192_CBC3_SHA,
2137         TLS1_CK_ECDHE_ECDSA_WITH_DES_192_CBC3_SHA,
2138         SSL_kEECDH,
2139         SSL_aECDSA,
2140         SSL_3DES,
2141         SSL_SHA1,
2142         SSL_TLSV1,
2143         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2144         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2145         112,
2146         168,
2147         },

2149         /* Cipher C009 */
2150         {
2151         1,
2152         TLS1_TXT_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
2153         TLS1_CK_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
2154         SSL_kEECDH,
2155         SSL_aECDSA,
2156         SSL_AES128,
2157         SSL_SHA1,
2158         SSL_TLSV1,
2159         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2160         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2161         128,
2162         128,
2163         },

2165         /* Cipher C00A */
2166         {
2167         1,
2168         TLS1_TXT_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
2169         TLS1_CK_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
2170         SSL_kEECDH,
2171         SSL_aECDSA,
2172         SSL_AES256,
```



```
2173         SSL_SHA1,
2174         SSL_TLSV1,
2175         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2176         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2177         256,
2178         256,
2179         },

2181         /* Cipher C00B */
2182         {
2183         1,
2184         TLS1_TXT_ECDH_RSA_WITH_NULL_SHA,
2185         TLS1_CK_ECDH_RSA_WITH_NULL_SHA,
2186         SSL_kECDHr,
2187         SSL_aECDH,
2188         SSL_eNULL,
2189         SSL_SHA1,
2190         SSL_TLSV1,
2191         SSL_NOT_EXP|SSL_STRONG_NONE|SSL_FIPS,
2192         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2193         0,
2194         0,
2195         },

2197         /* Cipher C00C */
2198         {
2199         1,
2200         TLS1_TXT_ECDH_RSA_WITH_RC4_128_SHA,
2201         TLS1_CK_ECDH_RSA_WITH_RC4_128_SHA,
2202         SSL_kECDHr,
2203         SSL_aECDH,
2204         SSL_RC4,
2205         SSL_SHA1,
2206         SSL_TLSV1,
2207         SSL_NOT_EXP|SSL_MEDIUM,
2208         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2209         128,
2210         128,
2211         },

2213         /* Cipher C00D */
2214         {
2215         1,
2216         TLS1_TXT_ECDH_RSA_WITH_DES_192_CBC3_SHA,
2217         TLS1_CK_ECDH_RSA_WITH_DES_192_CBC3_SHA,
2218         SSL_kECDHr,
2219         SSL_aECDH,
2220         SSL_3DES,
2221         SSL_SHA1,
2222         SSL_TLSV1,
2223         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2224         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2225         112,
2226         168,
2227         },

2229         /* Cipher C00E */
2230         {
2231         1,
2232         TLS1_TXT_ECDH_RSA_WITH_AES_128_CBC_SHA,
2233         TLS1_CK_ECDH_RSA_WITH_AES_128_CBC_SHA,
2234         SSL_kECDHr,
2235         SSL_aECDH,
2236         SSL_AES128,
2237         SSL_SHA1,
2238         SSL_TLSV1,
```

```
2239         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2240         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2241         128,
2242         128,
2243         },

2245         /* Cipher C00F */
2246         {
2247         1,
2248         TLS1_TXT_ECDH_RSA_WITH_AES_256_CBC_SHA,
2249         TLS1_CK_ECDH_RSA_WITH_AES_256_CBC_SHA,
2250         SSL_kECDHr,
2251         SSL_aECDH,
2252         SSL_AES256,
2253         SSL_SHA1,
2254         SSL_TLSV1,
2255         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2256         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2257         256,
2258         256,
2259         },

2261         /* Cipher C010 */
2262         {
2263         1,
2264         TLS1_TXT_ECDHE_RSA_WITH_NULL_SHA,
2265         TLS1_CK_ECDHE_RSA_WITH_NULL_SHA,
2266         SSL_kEECDH,
2267         SSL_aRSA,
2268         SSL_eNULL,
2269         SSL_SHA1,
2270         SSL_TLSV1,
2271         SSL_NOT_EXP|SSL_STRONG_NONE|SSL_FIPS,
2272         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2273         0,
2274         0,
2275         },

2277         /* Cipher C011 */
2278         {
2279         1,
2280         TLS1_TXT_ECDHE_RSA_WITH_RC4_128_SHA,
2281         TLS1_CK_ECDHE_RSA_WITH_RC4_128_SHA,
2282         SSL_kEECDH,
2283         SSL_aRSA,
2284         SSL_RC4,
2285         SSL_SHA1,
2286         SSL_TLSV1,
2287         SSL_NOT_EXP|SSL_MEDIUM,
2288         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2289         128,
2290         128,
2291         },

2293         /* Cipher C012 */
2294         {
2295         1,
2296         TLS1_TXT_ECDHE_RSA_WITH_DES_192_CBC3_SHA,
2297         TLS1_CK_ECDHE_RSA_WITH_DES_192_CBC3_SHA,
2298         SSL_kEECDH,
2299         SSL_aRSA,
2300         SSL_3DES,
2301         SSL_SHA1,
2302         SSL_TLSV1,
2303         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2304         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
```



```
2305         112,
2306         168,
2307         },

2309         /* Cipher C013 */
2310         {
2311         1,
2312         TLS1_TXT_ECDHE_RSA_WITH_AES_128_CBC_SHA,
2313         TLS1_CK_ECDHE_RSA_WITH_AES_128_CBC_SHA,
2314         SSL_kEECDH,
2315         SSL_aRSA,
2316         SSL_AES128,
2317         SSL_SHA1,
2318         SSL_TLSV1,
2319         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2320         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2321         128,
2322         128,
2323         },

2325         /* Cipher C014 */
2326         {
2327         1,
2328         TLS1_TXT_ECDHE_RSA_WITH_AES_256_CBC_SHA,
2329         TLS1_CK_ECDHE_RSA_WITH_AES_256_CBC_SHA,
2330         SSL_kEECDH,
2331         SSL_aRSA,
2332         SSL_AES256,
2333         SSL_SHA1,
2334         SSL_TLSV1,
2335         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2336         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2337         256,
2338         256,
2339         },

2341         /* Cipher C015 */
2342         {
2343         1,
2344         TLS1_TXT_ECDH_anon_WITH_NULL_SHA,
2345         TLS1_CK_ECDH_anon_WITH_NULL_SHA,
2346         SSL_kEECDH,
2347         SSL_aNULL,
2348         SSL_eNULL,
2349         SSL_SHA1,
2350         SSL_TLSV1,
2351         SSL_NOT_EXP|SSL_STRONG_NONE|SSL_FIPS,
2352         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2353         0,
2354         0,
2355         },

2357         /* Cipher C016 */
2358         {
2359         1,
2360         TLS1_TXT_ECDH_anon_WITH_RC4_128_SHA,
2361         TLS1_CK_ECDH_anon_WITH_RC4_128_SHA,
2362         SSL_kEECDH,
2363         SSL_aNULL,
2364         SSL_RC4,
2365         SSL_SHA1,
2366         SSL_TLSV1,
2367         SSL_NOT_EXP|SSL_MEDIUM,
2368         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2369         128,
2370         128,
```

```
2371         },

2373         /* Cipher C017 */
2374         {
2375         1,
2376         TLS1_TXT_ECDH_anon_WITH_DES_192_CBC3_SHA,
2377         TLS1_CK_ECDH_anon_WITH_DES_192_CBC3_SHA,
2378         SSL_kEECDH,
2379         SSL_aNULL,
2380         SSL_3DES,
2381         SSL_SHA1,
2382         SSL_TLSV1,
2383         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2384         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2385         112,
2386         168,
2387         },

2389         /* Cipher C018 */
2390         {
2391         1,
2392         TLS1_TXT_ECDH_anon_WITH_AES_128_CBC_SHA,
2393         TLS1_CK_ECDH_anon_WITH_AES_128_CBC_SHA,
2394         SSL_kEECDH,
2395         SSL_aNULL,
2396         SSL_AES128,
2397         SSL_SHA1,
2398         SSL_TLSV1,
2399         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2400         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2401         128,
2402         128,
2403         },

2405         /* Cipher C019 */
2406         {
2407         1,
2408         TLS1_TXT_ECDH_anon_WITH_AES_256_CBC_SHA,
2409         TLS1_CK_ECDH_anon_WITH_AES_256_CBC_SHA,
2410         SSL_kEECDH,
2411         SSL_aNULL,
2412         SSL_AES256,
2413         SSL_SHA1,
2414         SSL_TLSV1,
2415         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2416         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2417         256,
2418         256,
2419         },
2420 #endif  /* OPENSSL_NO_ECDH */

2422 #ifndef OPENSSL_NO_SRP
2423         /* Cipher C01A */
2424         {
2425         1,
2426         TLS1_TXT_SRP_SHA_WITH_3DES_EDE_CBC_SHA,
2427         TLS1_CK_SRP_SHA_WITH_3DES_EDE_CBC_SHA,
2428         SSL_kSRP,
2429         SSL_aSRP,
2430         SSL_3DES,
2431         SSL_SHA1,
2432         SSL_TLSV1,
2433         SSL_NOT_EXP|SSL_HIGH,
2434         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2435         112,
2436         168,
```



```
2437         },

2439         /* Cipher C01B */
2440         {
2441         1,
2442         TLS1_TXT_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA,
2443         TLS1_CK_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA,
2444         SSL_kSRP,
2445         SSL_aRSA,
2446         SSL_3DES,
2447         SSL_SHA1,
2448         SSL_TLSV1,
2449         SSL_NOT_EXP|SSL_HIGH,
2450         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2451         112,
2452         168,
2453         },

2455         /* Cipher C01C */
2456         {
2457         1,
2458         TLS1_TXT_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA,
2459         TLS1_CK_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA,
2460         SSL_kSRP,
2461         SSL_aDSS,
2462         SSL_3DES,
2463         SSL_SHA1,
2464         SSL_TLSV1,
2465         SSL_NOT_EXP|SSL_HIGH,
2466         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2467         112,
2468         168,
2469         },

2471         /* Cipher C01D */
2472         {
2473         1,
2474         TLS1_TXT_SRP_SHA_WITH_AES_128_CBC_SHA,
2475         TLS1_CK_SRP_SHA_WITH_AES_128_CBC_SHA,
2476         SSL_kSRP,
2477         SSL_aSRP,
2478         SSL_AES128,
2479         SSL_SHA1,
2480         SSL_TLSV1,
2481         SSL_NOT_EXP|SSL_HIGH,
2482         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2483         128,
2484         128,
2485         },

2487         /* Cipher C01E */
2488         {
2489         1,
2490         TLS1_TXT_SRP_SHA_RSA_WITH_AES_128_CBC_SHA,
2491         TLS1_CK_SRP_SHA_RSA_WITH_AES_128_CBC_SHA,
2492         SSL_kSRP,
2493         SSL_aRSA,
2494         SSL_AES128,
2495         SSL_SHA1,
2496         SSL_TLSV1,
2497         SSL_NOT_EXP|SSL_HIGH,
2498         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2499         128,
2500         128,
2501         },
```

```
2503         /* Cipher C01F */
2504         {
2505         1,
2506         TLS1_TXT_SRP_SHA_DSS_WITH_AES_128_CBC_SHA,
2507         TLS1_CK_SRP_SHA_DSS_WITH_AES_128_CBC_SHA,
2508         SSL_kSRP,
2509         SSL_aDSS,
2510         SSL_AES128,
2511         SSL_SHA1,
2512         SSL_TLSV1,
2513         SSL_NOT_EXP|SSL_HIGH,
2514         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2515         128,
2516         128,
2517         },

2519         /* Cipher C020 */
2520         {
2521         1,
2522         TLS1_TXT_SRP_SHA_WITH_AES_256_CBC_SHA,
2523         TLS1_CK_SRP_SHA_WITH_AES_256_CBC_SHA,
2524         SSL_kSRP,
2525         SSL_aSRP,
2526         SSL_AES256,
2527         SSL_SHA1,
2528         SSL_TLSV1,
2529         SSL_NOT_EXP|SSL_HIGH,
2530         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2531         256,
2532         256,
2533         },

2535         /* Cipher C021 */
2536         {
2537         1,
2538         TLS1_TXT_SRP_SHA_RSA_WITH_AES_256_CBC_SHA,
2539         TLS1_CK_SRP_SHA_RSA_WITH_AES_256_CBC_SHA,
2540         SSL_kSRP,
2541         SSL_aRSA,
2542         SSL_AES256,
2543         SSL_SHA1,
2544         SSL_TLSV1,
2545         SSL_NOT_EXP|SSL_HIGH,
2546         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2547         256,
2548         256,
2549         },

2551         /* Cipher C022 */
2552         {
2553         1,
2554         TLS1_TXT_SRP_SHA_DSS_WITH_AES_256_CBC_SHA,
2555         TLS1_CK_SRP_SHA_DSS_WITH_AES_256_CBC_SHA,
2556         SSL_kSRP,
2557         SSL_aDSS,
2558         SSL_AES256,
2559         SSL_SHA1,
2560         SSL_TLSV1,
2561         SSL_NOT_EXP|SSL_HIGH,
2562         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2563         256,
2564         256,
2565         },
2566 #endif  /* OPENSSL_NO_SRP */
2567 #ifndef OPENSSL_NO_ECDH
```



```
2569         /* HMAC based TLS v1.2 ciphersuites from RFC5289 */

2571         /* Cipher C023 */
2572         {
2573         1,
2574         TLS1_TXT_ECDHE_ECDSA_WITH_AES_128_SHA256,
2575         TLS1_CK_ECDHE_ECDSA_WITH_AES_128_SHA256,
2576         SSL_kEECDH,
2577         SSL_aECDSA,
2578         SSL_AES128,
2579         SSL_SHA256,
2580         SSL_TLSV1_2,
2581         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2582         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2583         128,
2584         128,
2585         },

2587         /* Cipher C024 */
2588         {
2589         1,
2590         TLS1_TXT_ECDHE_ECDSA_WITH_AES_256_SHA384,
2591         TLS1_CK_ECDHE_ECDSA_WITH_AES_256_SHA384,
2592         SSL_kEECDH,
2593         SSL_aECDSA,
2594         SSL_AES256,
2595         SSL_SHA384,
2596         SSL_TLSV1_2,
2597         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2598         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2599         256,
2600         256,
2601         },

2603         /* Cipher C025 */
2604         {
2605         1,
2606         TLS1_TXT_ECDH_ECDSA_WITH_AES_128_SHA256,
2607         TLS1_CK_ECDH_ECDSA_WITH_AES_128_SHA256,
2608         SSL_kECDHe,
2609         SSL_aECDH,
2610         SSL_AES128,
2611         SSL_SHA256,
2612         SSL_TLSV1_2,
2613         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2614         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2615         128,
2616         128,
2617         },

2619         /* Cipher C026 */
2620         {
2621         1,
2622         TLS1_TXT_ECDH_ECDSA_WITH_AES_256_SHA384,
2623         TLS1_CK_ECDH_ECDSA_WITH_AES_256_SHA384,
2624         SSL_kECDHe,
2625         SSL_aECDH,
2626         SSL_AES256,
2627         SSL_SHA384,
2628         SSL_TLSV1_2,
2629         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2630         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2631         256,
2632         256,
2633         },
```

```
2635         /* Cipher C027 */
2636         {
2637         1,
2638         TLS1_TXT_ECDHE_RSA_WITH_AES_128_SHA256,
2639         TLS1_CK_ECDHE_RSA_WITH_AES_128_SHA256,
2640         SSL_kEECDH,
2641         SSL_aRSA,
2642         SSL_AES128,
2643         SSL_SHA256,
2644         SSL_TLSV1_2,
2645         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2646         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2647         128,
2648         128,
2649         },

2651         /* Cipher C028 */
2652         {
2653         1,
2654         TLS1_TXT_ECDHE_RSA_WITH_AES_256_SHA384,
2655         TLS1_CK_ECDHE_RSA_WITH_AES_256_SHA384,
2656         SSL_kEECDH,
2657         SSL_aRSA,
2658         SSL_AES256,
2659         SSL_SHA384,
2660         SSL_TLSV1_2,
2661         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2662         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2663         256,
2664         256,
2665         },

2667         /* Cipher C029 */
2668         {
2669         1,
2670         TLS1_TXT_ECDH_RSA_WITH_AES_128_SHA256,
2671         TLS1_CK_ECDH_RSA_WITH_AES_128_SHA256,
2672         SSL_kECDHr,
2673         SSL_aECDH,
2674         SSL_AES128,
2675         SSL_SHA256,
2676         SSL_TLSV1_2,
2677         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2678         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2679         128,
2680         128,
2681         },

2683         /* Cipher C02A */
2684         {
2685         1,
2686         TLS1_TXT_ECDH_RSA_WITH_AES_256_SHA384,
2687         TLS1_CK_ECDH_RSA_WITH_AES_256_SHA384,
2688         SSL_kECDHr,
2689         SSL_aECDH,
2690         SSL_AES256,
2691         SSL_SHA384,
2692         SSL_TLSV1_2,
2693         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2694         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2695         256,
2696         256,
2697         },

2699         /* GCM based TLS v1.2 ciphersuites from RFC5289 */
```



```
2701         /* Cipher C02B */
2702         {
2703         1,
2704         TLS1_TXT_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
2705         TLS1_CK_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
2706         SSL_kEECDH,
2707         SSL_aECDSA,
2708         SSL_AES128GCM,
2709         SSL_AEAD,
2710         SSL_TLSV1_2,
2711         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2712         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2713         128,
2714         128,
2715         },

2717         /* Cipher C02C */
2718         {
2719         1,
2720         TLS1_TXT_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
2721         TLS1_CK_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
2722         SSL_kEECDH,
2723         SSL_aECDSA,
2724         SSL_AES256GCM,
2725         SSL_AEAD,
2726         SSL_TLSV1_2,
2727         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2728         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2729         256,
2730         256,
2731         },

2733         /* Cipher C02D */
2734         {
2735         1,
2736         TLS1_TXT_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,
2737         TLS1_CK_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,
2738         SSL_kECDHe,
2739         SSL_aECDH,
2740         SSL_AES128GCM,
2741         SSL_AEAD,
2742         SSL_TLSV1_2,
2743         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2744         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2745         128,
2746         128,
2747         },

2749         /* Cipher C02E */
2750         {
2751         1,
2752         TLS1_TXT_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,
2753         TLS1_CK_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,
2754         SSL_kECDHe,
2755         SSL_aECDH,
2756         SSL_AES256GCM,
2757         SSL_AEAD,
2758         SSL_TLSV1_2,
2759         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2760         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2761         256,
2762         256,
2763         },

2765         /* Cipher C02F */
2766         {
```

```
2767         1,
2768         TLS1_TXT_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
2769         TLS1_CK_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
2770         SSL_kEECDH,
2771         SSL_aRSA,
2772         SSL_AES128GCM,
2773         SSL_AEAD,
2774         SSL_TLSV1_2,
2775         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2776         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2777         128,
2778         128,
2779         },

2781         /* Cipher C030 */
2782         {
2783         1,
2784         TLS1_TXT_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
2785         TLS1_CK_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
2786         SSL_kEECDH,
2787         SSL_aRSA,
2788         SSL_AES256GCM,
2789         SSL_AEAD,
2790         SSL_TLSV1_2,
2791         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2792         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2793         256,
2794         256,
2795         },

2797         /* Cipher C031 */
2798         {
2799         1,
2800         TLS1_TXT_ECDH_RSA_WITH_AES_128_GCM_SHA256,
2801         TLS1_CK_ECDH_RSA_WITH_AES_128_GCM_SHA256,
2802         SSL_kECDHr,
2803         SSL_aECDH,
2804         SSL_AES128GCM,
2805         SSL_AEAD,
2806         SSL_TLSV1_2,
2807         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2808         SSL_HANDSHAKE_MAC_SHA256|TLS1_PRF_SHA256,
2809         128,
2810         128,
2811         },

2813         /* Cipher C032 */
2814         {
2815         1,
2816         TLS1_TXT_ECDH_RSA_WITH_AES_256_GCM_SHA384,
2817         TLS1_CK_ECDH_RSA_WITH_AES_256_GCM_SHA384,
2818         SSL_kECDHr,
2819         SSL_aECDH,
2820         SSL_AES256GCM,
2821         SSL_AEAD,
2822         SSL_TLSV1_2,
2823         SSL_NOT_EXP|SSL_HIGH|SSL_FIPS,
2824         SSL_HANDSHAKE_MAC_SHA384|TLS1_PRF_SHA384,
2825         256,
2826         256,
2827         },

2829 #endif /* OPENSSL_NO_ECDH */


2832 #ifdef TEMP_GOST_TLS
```



```
2833 /* Cipher FF00 */
2834         {
2835         1,
2836         "GOST-MD5",
2837         0x0300ff00,
2838         SSL_kRSA,
2839         SSL_aRSA,
2840         SSL_eGOST2814789CNT,
2841         SSL_MD5,
2842         SSL_TLSV1,
2843         SSL_NOT_EXP|SSL_HIGH,
2844         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2845         256,
2846         256,
2847         },
2848         {
2849         1,
2850         "GOST-GOST94",
2851         0x0300ff01,
2852         SSL_kRSA,
2853         SSL_aRSA,
2854         SSL_eGOST2814789CNT,
2855         SSL_GOST94,
2856         SSL_TLSV1,
2857         SSL_NOT_EXP|SSL_HIGH,
2858         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2859         256,
2860         256
2861         },
2862         {
2863         1,
2864         "GOST-GOST89MAC",
2865         0x0300ff02,
2866         SSL_kRSA,
2867         SSL_aRSA,
2868         SSL_eGOST2814789CNT,
2869         SSL_GOST89MAC,
2870         SSL_TLSV1,
2871         SSL_NOT_EXP|SSL_HIGH,
2872         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF,
2873         256,
2874         256
2875         },
2876         {
2877         1,
2878         "GOST-GOST89STREAM",
2879         0x0300ff03,
2880         SSL_kRSA,
2881         SSL_aRSA,
2882         SSL_eGOST2814789CNT,
2883         SSL_GOST89MAC,
2884         SSL_TLSV1,
2885         SSL_NOT_EXP|SSL_HIGH,
2886         SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF|TLS1_STREAM_MAC,
2887         256,
2888         256
2889         },
2890 #endif

2892 /* end of list */
2893         };

2895 SSL3_ENC_METHOD SSLv3_enc_data={
2896         ssl3_enc,
2897         n_ssl3_mac,
2898         ssl3_setup_key_block,
```

```
2899         ssl3_generate_master_secret,
2900         ssl3_change_cipher_state,
2901         ssl3_final_finish_mac,
2902         MD5_DIGEST_LENGTH+SHA_DIGEST_LENGTH,
2903         ssl3_cert_verify_mac,
2904         SSL3_MD_CLIENT_FINISHED_CONST,4,
2905         SSL3_MD_SERVER_FINISHED_CONST,4,
2906         ssl3_alert_code,
2907         (int (*)(SSL *, unsigned char *, size_t, const char *,
2908                  size_t, const unsigned char *, size_t,
2909                  int use_context))ssl_undefined_function,
2910         };

2912 long ssl3_default_timeout(void)
2913         {
2914         /* 2 hours, the 24 hours mentioned in the SSLv3 spec
2915          * is way too long for http, the cache would over fill */
2916         return(60*60*2);
2917         }

2919 int ssl3_num_ciphers(void)
2920         {
2921         return(SSL3_NUM_CIPHERS);
2922         }

2924 const SSL_CIPHER *ssl3_get_cipher(unsigned int u)
2925         {
2926         if (u < SSL3_NUM_CIPHERS)
2927                 return(&(ssl3_ciphers[SSL3_NUM_CIPHERS-1-u]));
2928         else
2929                 return(NULL);
2930         }

2932 int ssl3_pending(const SSL *s)
2933         {
2934         if (s->rstate == SSL_ST_READ_BODY)
2935                 return 0;

2937         return (s->s3->rrec.type == SSL3_RT_APPLICATION_DATA) ? s->s3->rrec.leng
2938         }

2940 int ssl3_new(SSL *s)
2941         {
2942         SSL3_STATE *s3;

2944         if ((s3=OPENSSL_malloc(sizeof *s3)) == NULL) goto err;
2945         memset(s3,0,sizeof *s3);
2946         memset(s3->rrec.seq_num,0,sizeof(s3->rrec.seq_num));
2947         memset(s3->wrec.seq_num,0,sizeof(s3->wrec.seq_num));

2949         s->s3=s3;

2951 #ifndef OPENSSL_NO_SRP
2952         SSL_SRP_CTX_init(s);
2953 #endif
2954         s->method->ssl_clear(s);
2955         return(1);
2956 err:
2957         return(0);
2958         }

2960 void ssl3_free(SSL *s)
2961         {
2962         if(s == NULL)
2963             return;
```



```
2965 #ifdef TLSEXT_TYPE_opaque_prf_input
2966         if (s->s3->client_opaque_prf_input != NULL)
2967                 OPENSSL_free(s->s3->client_opaque_prf_input);
2968         if (s->s3->server_opaque_prf_input != NULL)
2969                 OPENSSL_free(s->s3->server_opaque_prf_input);
2970 #endif

2972         ssl3_cleanup_key_block(s);
2973         if (s->s3->rbuf.buf != NULL)
2974                 ssl3_release_read_buffer(s);
2975         if (s->s3->wbuf.buf != NULL)
2976                 ssl3_release_write_buffer(s);
2977         if (s->s3->rrec.comp != NULL)
2978                 OPENSSL_free(s->s3->rrec.comp);
2979 #ifndef OPENSSL_NO_DH
2980         if (s->s3->tmp.dh != NULL)
2981                 DH_free(s->s3->tmp.dh);
2982 #endif
2983 #ifndef OPENSSL_NO_ECDH
2984         if (s->s3->tmp.ecdh != NULL)
2985                 EC_KEY_free(s->s3->tmp.ecdh);
2986 #endif

2988         if (s->s3->tmp.ca_names != NULL)
2989                 sk_X509_NAME_pop_free(s->s3->tmp.ca_names,X509_NAME_free);
2990         if (s->s3->handshake_buffer) {
2991                 BIO_free(s->s3->handshake_buffer);
2992         }
2993         if (s->s3->handshake_dgst) ssl3_free_digest_list(s);
2994 #ifndef OPENSSL_NO_SRP
2995         SSL_SRP_CTX_free(s);
2996 #endif
2997         OPENSSL_cleanse(s->s3,sizeof *s->s3);
2998         OPENSSL_free(s->s3);
2999         s->s3=NULL;
3000         }

3002 void ssl3_clear(SSL *s)
3003         {
3004         unsigned char *rp,*wp;
3005         size_t rlen, wlen;
3006         int init_extra;

3008 #ifdef TLSEXT_TYPE_opaque_prf_input
3009         if (s->s3->client_opaque_prf_input != NULL)
3010                 OPENSSL_free(s->s3->client_opaque_prf_input);
3011         s->s3->client_opaque_prf_input = NULL;
3012         if (s->s3->server_opaque_prf_input != NULL)
3013                 OPENSSL_free(s->s3->server_opaque_prf_input);
3014         s->s3->server_opaque_prf_input = NULL;
3015 #endif

3017         ssl3_cleanup_key_block(s);
3018         if (s->s3->tmp.ca_names != NULL)
3019                 sk_X509_NAME_pop_free(s->s3->tmp.ca_names,X509_NAME_free);

3021         if (s->s3->rrec.comp != NULL)
3022                 {
3023                 OPENSSL_free(s->s3->rrec.comp);
3024                 s->s3->rrec.comp=NULL;
3025                 }
3026 #ifndef OPENSSL_NO_DH
3027         if (s->s3->tmp.dh != NULL)
3028                 {
3029                 DH_free(s->s3->tmp.dh);
3030                 s->s3->tmp.dh = NULL;
```

```
3031                }
3032 #endif
3033 #ifndef OPENSSL_NO_ECDH
3034         if (s->s3->tmp.ecdh != NULL)
3035                 {
3036                 EC_KEY_free(s->s3->tmp.ecdh);
3037                 s->s3->tmp.ecdh = NULL;
3038                 }
3039 #endif
3040 #ifndef OPENSSL_NO_TLSEXT
3041 #ifndef OPENSSL_NO_EC
3042         s->s3->is_probably_safari = 0;
3043 #endif /* !OPENSSL_NO_EC */
3044 #endif /* !OPENSSL_NO_TLSEXT */

3046         rp = s->s3->rbuf.buf;
3047         wp = s->s3->wbuf.buf;
3048         rlen = s->s3->rbuf.len;
3049         wlen = s->s3->wbuf.len;
3050         init_extra = s->s3->init_extra;
3051         if (s->s3->handshake_buffer) {
3052                 BIO_free(s->s3->handshake_buffer);
3053                 s->s3->handshake_buffer = NULL;
3054         }
3055         if (s->s3->handshake_dgst) {
3056                 ssl3_free_digest_list(s);
3057         }
3058         memset(s->s3,0,sizeof *s->s3);
3059         s->s3->rbuf.buf = rp;
3060         s->s3->wbuf.buf = wp;
3061         s->s3->rbuf.len = rlen;
3062         s->s3->wbuf.len = wlen;
3063         s->s3->init_extra = init_extra;

3065         ssl_free_wbio_buffer(s);

3067         s->packet_length=0;
3068         s->s3->renegotiate=0;
3069         s->s3->total_renegotiations=0;
3070         s->s3->num_renegotiations=0;
3071         s->s3->in_read_app_data=0;
3072         s->version=SSL3_VERSION;

3074 #if !defined(OPENSSL_NO_TLSEXT) && !defined(OPENSSL_NO_NEXTPROTONEG)
3075         if (s->next_proto_negotiated)
3076                 {
3077                 OPENSSL_free(s->next_proto_negotiated);
3078                 s->next_proto_negotiated = NULL;
3079                 s->next_proto_negotiated_len = 0;
3080                 }
3081 #endif
3082         }

3084 #ifndef OPENSSL_NO_SRP
3085 static char * MS_CALLBACK srp_password_from_info_cb(SSL *s, void *arg)
3086         {
3087         return BUF_strdup(s->srp_ctx.info) ;
3088         }
3089 #endif

3091 long ssl3_ctrl(SSL *s, int cmd, long larg, void *parg)
3092         {
3093         int ret=0;

3095 #if !defined(OPENSSL_NO_DSA) || !defined(OPENSSL_NO_RSA)
3096         if (
```



```
3097 #ifndef OPENSSL_NO_RSA
3098             cmd == SSL_CTRL_SET_TMP_RSA ||
3099             cmd == SSL_CTRL_SET_TMP_RSA_CB ||
3100 #endif
3101 #ifndef OPENSSL_NO_DSA
3102             cmd == SSL_CTRL_SET_TMP_DH ||
3103             cmd == SSL_CTRL_SET_TMP_DH_CB ||
3104 #endif
3105                 0)
3106                 {
3107                 if (!ssl_cert_inst(&s->cert))
3108                         {
3109                         SSLerr(SSL_F_SSL3_CTRL, ERR_R_MALLOC_FAILURE);
3110                         return(0);
3111                         }
3112                 }
3113 #endif

3115         switch (cmd)
3116                 {
3117         case SSL_CTRL_GET_SESSION_REUSED:
3118                 ret=s->hit;
3119                 break;
3120         case SSL_CTRL_GET_CLIENT_CERT_REQUEST:
3121                 break;
3122         case SSL_CTRL_GET_NUM_RENEGOTIATIONS:
3123                 ret=s->s3->num_renegotiations;
3124                 break;
3125         case SSL_CTRL_CLEAR_NUM_RENEGOTIATIONS:
3126                 ret=s->s3->num_renegotiations;
3127                 s->s3->num_renegotiations=0;
3128                 break;
3129         case SSL_CTRL_GET_TOTAL_RENEGOTIATIONS:
3130                 ret=s->s3->total_renegotiations;
3131                 break;
3132         case SSL_CTRL_GET_FLAGS:
3133                 ret=(int)(s->s3->flags);
3134                 break;
3135 #ifndef OPENSSL_NO_RSA
3136         case SSL_CTRL_NEED_TMP_RSA:
3137                 if ((s->cert != NULL) && (s->cert->rsa_tmp == NULL) &&
3138                     ((s->cert->pkeys[SSL_PKEY_RSA_ENC].privatekey == NULL) ||
3139                      (EVP_PKEY_size(s->cert->pkeys[SSL_PKEY_RSA_ENC].privatekey)
3140                         ret = 1;
3141                 break;
3142         case SSL_CTRL_SET_TMP_RSA:
3143                 {
3144                         RSA *rsa = (RSA *)parg;
3145                         if (rsa == NULL)
3146                                 {
3147                                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_PASSED_NULL_PARAME
3148                                 return(ret);
3149                                 }
3150                         if ((rsa = RSAPrivateKey_dup(rsa)) == NULL)
3151                                 {
3152                                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_RSA_LIB);
3153                                 return(ret);
3154                                 }
3155                         if (s->cert->rsa_tmp != NULL)
3156                                 RSA_free(s->cert->rsa_tmp);
3157                         s->cert->rsa_tmp = rsa;
3158                         ret = 1;
3159                 }
3160                 break;
3161         case SSL_CTRL_SET_TMP_RSA_CB:
3162                 {
```

```
3163                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
3164                 return(ret);
3165                 }
3166                 break;
3167 #endif
3168 #ifndef OPENSSL_NO_DH
3169         case SSL_CTRL_SET_TMP_DH:
3170                 {
3171                         DH *dh = (DH *)parg;
3172                         if (dh == NULL)
3173                                 {
3174                                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_PASSED_NULL_PARAME
3175                                 return(ret);
3176                                 }
3177                         if ((dh = DHparams_dup(dh)) == NULL)
3178                                 {
3179                                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_DH_LIB);
3180                                 return(ret);
3181                                 }
3182                         if (!(s->options & SSL_OP_SINGLE_DH_USE))
3183                                 {
3184                                 if (!DH_generate_key(dh))
3185                                         {
3186                                         DH_free(dh);
3187                                         SSLerr(SSL_F_SSL3_CTRL, ERR_R_DH_LIB);
3188                                         return(ret);
3189                                         }
3190                                 }
3191                         if (s->cert->dh_tmp != NULL)
3192                                 DH_free(s->cert->dh_tmp);
3193                         s->cert->dh_tmp = dh;
3194                         ret = 1;
3195                 }
3196                 break;
3197         case SSL_CTRL_SET_TMP_DH_CB:
3198                 {
3199                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
3200                 return(ret);
3201                 }
3202                 break;
3203 #endif
3204 #ifndef OPENSSL_NO_ECDH
3205         case SSL_CTRL_SET_TMP_ECDH:
3206                 {
3207                 EC_KEY *ecdh = NULL;

3209                 if (parg == NULL)
3210                         {
3211                         SSLerr(SSL_F_SSL3_CTRL, ERR_R_PASSED_NULL_PARAMETER);
3212                         return(ret);
3213                         }
3214                 if (!EC_KEY_up_ref((EC_KEY *)parg))
3215                         {
3216                         SSLerr(SSL_F_SSL3_CTRL,ERR_R_ECDH_LIB);
3217                         return(ret);
3218                         }
3219                 ecdh = (EC_KEY *)parg;
3220                 if (!(s->options & SSL_OP_SINGLE_ECDH_USE))
3221                         {
3222                         if (!EC_KEY_generate_key(ecdh))
3223                                 {
3224                                 EC_KEY_free(ecdh);
3225                                 SSLerr(SSL_F_SSL3_CTRL,ERR_R_ECDH_LIB);
3226                                 return(ret);
3227                                 }
3228                         }
```



```
3229                 if (s->cert->ecdh_tmp != NULL)
3230                         EC_KEY_free(s->cert->ecdh_tmp);
3231                 s->cert->ecdh_tmp = ecdh;
3232                 ret = 1;
3233                 }
3234                 break;
3235         case SSL_CTRL_SET_TMP_ECDH_CB:
3236                 {
3237                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
3238                 return(ret);
3239                 }
3240                 break;
3241 #endif /* !OPENSSL_NO_ECDH */
3242 #ifndef OPENSSL_NO_TLSEXT
3243         case SSL_CTRL_SET_TLSEXT_HOSTNAME:
3244                 if (larg == TLSEXT_NAMETYPE_host_name)
3245                         {
3246                         if (s->tlsext_hostname != NULL)
3247                                 OPENSSL_free(s->tlsext_hostname);
3248                         s->tlsext_hostname = NULL;

3250                         ret = 1;
3251                         if (parg == NULL)
3252                                 break;
3253                         if (strlen((char *)parg) > TLSEXT_MAXLEN_host_name)
3254                                 {
3255                                 SSLerr(SSL_F_SSL3_CTRL, SSL_R_SSL3_EXT_INVALID_S
3256                                 return 0;
3257                                 }
3258                         if ((s->tlsext_hostname = BUF_strdup((char *)parg)) == N
3259                                 {
3260                                 SSLerr(SSL_F_SSL3_CTRL, ERR_R_INTERNAL_ERROR);
3261                                 return 0;
3262                                 }
3263                         }
3264                 else
3265                         {
3266                         SSLerr(SSL_F_SSL3_CTRL, SSL_R_SSL3_EXT_INVALID_SERVERNAM
3267                         return 0;
3268                         }
3269                 break;
3270         case SSL_CTRL_SET_TLSEXT_DEBUG_ARG:
3271                 s->tlsext_debug_arg=parg;
3272                 ret = 1;
3273                 break;

3275 #ifdef TLSEXT_TYPE_opaque_prf_input
3276         case SSL_CTRL_SET_TLSEXT_OPAQUE_PRF_INPUT:
3277                 if (larg > 12288) /* actual internal limit is 2^16 for the compl
3278                                    * (including the cert chain and everything) *
3279                         {
3280                         SSLerr(SSL_F_SSL3_CTRL, SSL_R_OPAQUE_PRF_INPUT_TOO_LONG)
3281                         break;
3282                         }
3283                 if (s->tlsext_opaque_prf_input != NULL)
3284                         OPENSSL_free(s->tlsext_opaque_prf_input);
3285                 if ((size_t)larg == 0)
3286                         s->tlsext_opaque_prf_input = OPENSSL_malloc(1); /* dummy
3287                 else
3288                         s->tlsext_opaque_prf_input = BUF_memdup(parg, (size_t)la
3289                 if (s->tlsext_opaque_prf_input != NULL)
3290                         {
3291                         s->tlsext_opaque_prf_input_len = (size_t)larg;
3292                         ret = 1;
3293                         }
3294                 else
```

```
3295                        s->tlsext_opaque_prf_input_len = 0;
3296                break;
3297 #endif

3299        case SSL_CTRL_SET_TLSEXT_STATUS_REQ_TYPE:
3300                s->tlsext_status_type=larg;
3301                ret = 1;
3302                break;

3304        case SSL_CTRL_GET_TLSEXT_STATUS_REQ_EXTS:
3305                *(STACK_OF(X509_EXTENSION) **)parg = s->tlsext_ocsp_exts;
3306                ret = 1;
3307                break;

3309        case SSL_CTRL_SET_TLSEXT_STATUS_REQ_EXTS:
3310                s->tlsext_ocsp_exts = parg;
3311                ret = 1;
3312                break;

3314        case SSL_CTRL_GET_TLSEXT_STATUS_REQ_IDS:
3315                *(STACK_OF(OCSP_RESPID) **)parg = s->tlsext_ocsp_ids;
3316                ret = 1;
3317                break;

3319        case SSL_CTRL_SET_TLSEXT_STATUS_REQ_IDS:
3320                s->tlsext_ocsp_ids = parg;
3321                ret = 1;
3322                break;

3324        case SSL_CTRL_GET_TLSEXT_STATUS_REQ_OCSP_RESP:
3325                *(unsigned char **)parg = s->tlsext_ocsp_resp;
3326                return s->tlsext_ocsp_resplen;

3328        case SSL_CTRL_SET_TLSEXT_STATUS_REQ_OCSP_RESP:
3329                if (s->tlsext_ocsp_resp)
3330                        OPENSSL_free(s->tlsext_ocsp_resp);
3331                s->tlsext_ocsp_resp = parg;
3332                s->tlsext_ocsp_resplen = larg;
3333                ret = 1;
3334                break;

3336 #ifndef OPENSSL_NO_HEARTBEATS
3337        case SSL_CTRL_TLS_EXT_SEND_HEARTBEAT:
3338                if (SSL_version(s) == DTLS1_VERSION || SSL_version(s) == DTLS1_B
3339                        ret = dtls1_heartbeat(s);
3340                else
3341                        ret = tls1_heartbeat(s);
3342                break;

3344        case SSL_CTRL_GET_TLS_EXT_HEARTBEAT_PENDING:
3345                ret = s->tlsext_hb_pending;
3346                break;

3348        case SSL_CTRL_SET_TLS_EXT_HEARTBEAT_NO_REQUESTS:
3349                if (larg)
3350                        s->tlsext_heartbeat |= SSL_TLSEXT_HB_DONT_RECV_REQUESTS;
3351                else
3352                        s->tlsext_heartbeat &= ~SSL_TLSEXT_HB_DONT_RECV_REQUESTS
3353                ret = 1;
3354                break;
3355 #endif

3357 #endif /* !OPENSSL_NO_TLSEXT */
3358        default:
3359                break;
3360                }
```



```
3361        return(ret);
3362        }

3364 long ssl3_callback_ctrl(SSL *s, int cmd, void (*fp)(void))
3365        {
3366        int ret=0;

3368 #if !defined(OPENSSL_NO_DSA) || !defined(OPENSSL_NO_RSA)
3369        if (
3370 #ifndef OPENSSL_NO_RSA
3371            cmd == SSL_CTRL_SET_TMP_RSA_CB ||
3372 #endif
3373 #ifndef OPENSSL_NO_DSA
3374            cmd == SSL_CTRL_SET_TMP_DH_CB ||
3375 #endif
3376                0)
3377                {
3378                if (!ssl_cert_inst(&s->cert))
3379                        {
3380                        SSLerr(SSL_F_SSL3_CALLBACK_CTRL, ERR_R_MALLOC_FAILURE);
3381                        return(0);
3382                        }
3383                }
3384 #endif

3386        switch (cmd)
3387                {
3388 #ifndef OPENSSL_NO_RSA
3389        case SSL_CTRL_SET_TMP_RSA_CB:
3390                {
3391                s->cert->rsa_tmp_cb = (RSA *(*)(SSL *, int, int))fp;
3392                }
3393                break;
3394 #endif
3395 #ifndef OPENSSL_NO_DH
3396        case SSL_CTRL_SET_TMP_DH_CB:
3397                {
3398                s->cert->dh_tmp_cb = (DH *(*)(SSL *, int, int))fp;
3399                }
3400                break;
3401 #endif
3402 #ifndef OPENSSL_NO_ECDH
3403        case SSL_CTRL_SET_TMP_ECDH_CB:
3404                {
3405                s->cert->ecdh_tmp_cb = (EC_KEY *(*)(SSL *, int, int))fp;
3406                }
3407                break;
3408 #endif
3409 #ifndef OPENSSL_NO_TLSEXT
3410        case SSL_CTRL_SET_TLSEXT_DEBUG_CB:
3411                s->tlsext_debug_cb=(void (*)(SSL *,int ,int,
3412                                             unsigned char *, int, void *))fp;
3413                break;
3414 #endif
3415        default:
3416                break;
3417                }
3418        return(ret);
3419        }

3421 long ssl3_ctx_ctrl(SSL_CTX *ctx, int cmd, long larg, void *parg)
3422        {
3423        CERT *cert;

3425        cert=ctx->cert;
```

```
3427         switch (cmd)
3428                 {
3429 #ifndef OPENSSL_NO_RSA
3430         case SSL_CTRL_NEED_TMP_RSA:
3431                 if (    (cert->rsa_tmp == NULL) &&
3432                         ((cert->pkeys[SSL_PKEY_RSA_ENC].privatekey == NULL) ||
3433                          (EVP_PKEY_size(cert->pkeys[SSL_PKEY_RSA_ENC].privatekey
3434                         )
3435                         return(1);
3436                 else
3437                         return(0);
3438                 /* break; */
3439         case SSL_CTRL_SET_TMP_RSA:
3440                 {
3441                 RSA *rsa;
3442                 int i;

3444                 rsa=(RSA *)parg;
3445                 i=1;
3446                 if (rsa == NULL)
3447                         i=0;
3448                 else
3449                         {
3450                         if ((rsa=RSAPrivateKey_dup(rsa)) == NULL)
3451                                 i=0;
3452                         }
3453                 if (!i)
3454                         {
3455                         SSLerr(SSL_F_SSL3_CTX_CTRL,ERR_R_RSA_LIB);
3456                         return(0);
3457                         }
3458                 else
3459                         {
3460                         if (cert->rsa_tmp != NULL)
3461                                 RSA_free(cert->rsa_tmp);
3462                         cert->rsa_tmp=rsa;
3463                         return(1);
3464                         }
3465                 }
3466                 /* break; */
3467         case SSL_CTRL_SET_TMP_RSA_CB:
3468                 {
3469                 SSLerr(SSL_F_SSL3_CTX_CTRL, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
3470                 return(0);
3471                 }
3472                 break;
3473 #endif
3474 #ifndef OPENSSL_NO_DH
3475         case SSL_CTRL_SET_TMP_DH:
3476                 {
3477                 DH *new=NULL,*dh;

3479                 dh=(DH *)parg;
3480                 if ((new=DHparams_dup(dh)) == NULL)
3481                         {
3482                         SSLerr(SSL_F_SSL3_CTX_CTRL,ERR_R_DH_LIB);
3483                         return 0;
3484                         }
3485                 if (!(ctx->options & SSL_OP_SINGLE_DH_USE))
3486                         {
3487                         if (!DH_generate_key(new))
3488                                 {
3489                                 SSLerr(SSL_F_SSL3_CTX_CTRL,ERR_R_DH_LIB);
3490                                 DH_free(new);
3491                                 return 0;
3492                                 }
```



```
3493                         }
3494                 if (cert->dh_tmp != NULL)
3495                         DH_free(cert->dh_tmp);
3496                 cert->dh_tmp=new;
3497                 return 1;
3498                 }
3499                 /*break; */
3500         case SSL_CTRL_SET_TMP_DH_CB:
3501                 {
3502                 SSLerr(SSL_F_SSL3_CTX_CTRL, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
3503                 return(0);
3504                 }
3505                 break;
3506 #endif
3507 #ifndef OPENSSL_NO_ECDH
3508         case SSL_CTRL_SET_TMP_ECDH:
3509                 {
3510                 EC_KEY *ecdh = NULL;

3512                 if (parg == NULL)
3513                         {
3514                         SSLerr(SSL_F_SSL3_CTX_CTRL,ERR_R_ECDH_LIB);
3515                         return 0;
3516                         }
3517                 ecdh = EC_KEY_dup((EC_KEY *)parg);
3518                 if (ecdh == NULL)
3519                         {
3520                         SSLerr(SSL_F_SSL3_CTX_CTRL,ERR_R_EC_LIB);
3521                         return 0;
3522                         }
3523                 if (!(ctx->options & SSL_OP_SINGLE_ECDH_USE))
3524                         {
3525                         if (!EC_KEY_generate_key(ecdh))
3526                                 {
3527                                 EC_KEY_free(ecdh);
3528                                 SSLerr(SSL_F_SSL3_CTX_CTRL,ERR_R_ECDH_LIB);
3529                                 return 0;
3530                                 }
3531                         }

3533                 if (cert->ecdh_tmp != NULL)
3534                         {
3535                         EC_KEY_free(cert->ecdh_tmp);
3536                         }
3537                 cert->ecdh_tmp = ecdh;
3538                 return 1;
3539                 }
3540                 /* break; */
3541         case SSL_CTRL_SET_TMP_ECDH_CB:
3542                 {
3543                 SSLerr(SSL_F_SSL3_CTX_CTRL, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
3544                 return(0);
3545                 }
3546                 break;
3547 #endif /* !OPENSSL_NO_ECDH */
3548 #ifndef OPENSSL_NO_TLSEXT
3549         case SSL_CTRL_SET_TLSEXT_SERVERNAME_ARG:
3550                 ctx->tlsext_servername_arg=parg;
3551                 break;
3552         case SSL_CTRL_SET_TLSEXT_TICKET_KEYS:
3553         case SSL_CTRL_GET_TLSEXT_TICKET_KEYS:
3554                 {
3555                 unsigned char *keys = parg;
3556                 if (!keys)
3557                         return 48;
3558                 if (larg != 48)
```

```
3559                        {
3560                        SSLerr(SSL_F_SSL3_CTX_CTRL, SSL_R_INVALID_TICKET_KEYS_LE
3561                        return 0;
3562                        }
3563                if (cmd == SSL_CTRL_SET_TLSEXT_TICKET_KEYS)
3564                        {
3565                        memcpy(ctx->tlsext_tick_key_name, keys, 16);
3566                        memcpy(ctx->tlsext_tick_hmac_key, keys + 16, 16);
3567                        memcpy(ctx->tlsext_tick_aes_key, keys + 32, 16);
3568                        }
3569                else
3570                        {
3571                        memcpy(keys, ctx->tlsext_tick_key_name, 16);
3572                        memcpy(keys + 16, ctx->tlsext_tick_hmac_key, 16);
3573                        memcpy(keys + 32, ctx->tlsext_tick_aes_key, 16);
3574                        }
3575                return 1;
3576                }

3578 #ifdef TLSEXT_TYPE_opaque_prf_input
3579         case SSL_CTRL_SET_TLSEXT_OPAQUE_PRF_INPUT_CB_ARG:
3580                 ctx->tlsext_opaque_prf_input_callback_arg = parg;
3581                 return 1;
3582 #endif

3584         case SSL_CTRL_SET_TLSEXT_STATUS_REQ_CB_ARG:
3585                 ctx->tlsext_status_arg=parg;
3586                 return 1;
3587                 break;

3589 #ifndef OPENSSL_NO_SRP
3590         case SSL_CTRL_SET_TLS_EXT_SRP_USERNAME:
3591                 ctx->srp_ctx.srp_Mask|=SSL_kSRP;
3592                 if (ctx->srp_ctx.login != NULL)
3593                         OPENSSL_free(ctx->srp_ctx.login);
3594                 ctx->srp_ctx.login = NULL;
3595                 if (parg == NULL)
3596                         break;
3597                 if (strlen((const char *)parg) > 255 || strlen((const char *)par
3598                         {
3599                         SSLerr(SSL_F_SSL3_CTX_CTRL, SSL_R_INVALID_SRP_USERNAME);
3600                         return 0;
3601                         }
3602                 if ((ctx->srp_ctx.login = BUF_strdup((char *)parg)) == NULL)
3603                         {
3604                         SSLerr(SSL_F_SSL3_CTX_CTRL, ERR_R_INTERNAL_ERROR);
3605                         return 0;
3606                         }
3607                 break;
3608         case SSL_CTRL_SET_TLS_EXT_SRP_PASSWORD:
3609                 ctx->srp_ctx.SRP_give_srp_client_pwd_callback=srp_password_from_
3610                 ctx->srp_ctx.info=parg;
3611                 break;
3612         case SSL_CTRL_SET_SRP_ARG:
3613                 ctx->srp_ctx.srp_Mask|=SSL_kSRP;
3614                 ctx->srp_ctx.SRP_cb_arg=parg;
3615                 break;

3617         case SSL_CTRL_SET_TLS_EXT_SRP_STRENGTH:
3618                 ctx->srp_ctx.strength=larg;
3619                 break;
3620 #endif
3621 #endif /* !OPENSSL_NO_TLSEXT */

3623         /* A Thawte special :-) */
3624         case SSL_CTRL_EXTRA_CHAIN_CERT:
```



```
3625                 if (ctx->extra_certs == NULL)
3626                         {
3627                         if ((ctx->extra_certs=sk_X509_new_null()) == NULL)
3628                                 return(0);
3629                         }
3630                 sk_X509_push(ctx->extra_certs,(X509 *)parg);
3631                 break;

3633         case SSL_CTRL_GET_EXTRA_CHAIN_CERTS:
3634                 *(STACK_OF(X509) **)parg =  ctx->extra_certs;
3635                 break;

3637         case SSL_CTRL_CLEAR_EXTRA_CHAIN_CERTS:
3638                 if (ctx->extra_certs)
3639                         {
3640                         sk_X509_pop_free(ctx->extra_certs, X509_free);
3641                         ctx->extra_certs = NULL;
3642                         }
3643                 break;

3645         default:
3646                 return(0);
3647                 }
3648         return(1);
3649         }

3651 long ssl3_ctx_callback_ctrl(SSL_CTX *ctx, int cmd, void (*fp)(void))
3652         {
3653         CERT *cert;

3655         cert=ctx->cert;

3657         switch (cmd)
3658                 {
3659 #ifndef OPENSSL_NO_RSA
3660         case SSL_CTRL_SET_TMP_RSA_CB:
3661                 {
3662                 cert->rsa_tmp_cb = (RSA *(*)(SSL *, int, int))fp;
3663                 }
3664                 break;
3665 #endif
3666 #ifndef OPENSSL_NO_DH
3667         case SSL_CTRL_SET_TMP_DH_CB:
3668                 {
3669                 cert->dh_tmp_cb = (DH *(*)(SSL *, int, int))fp;
3670                 }
3671                 break;
3672 #endif
3673 #ifndef OPENSSL_NO_ECDH
3674         case SSL_CTRL_SET_TMP_ECDH_CB:
3675                 {
3676                 cert->ecdh_tmp_cb = (EC_KEY *(*)(SSL *, int, int))fp;
3677                 }
3678                 break;
3679 #endif
3680 #ifndef OPENSSL_NO_TLSEXT
3681         case SSL_CTRL_SET_TLSEXT_SERVERNAME_CB:
3682                 ctx->tlsext_servername_callback=(int (*)(SSL *,int *,void *))fp;
3683                 break;

3685 #ifdef TLSEXT_TYPE_opaque_prf_input
3686         case SSL_CTRL_SET_TLSEXT_OPAQUE_PRF_INPUT_CB:
3687                 ctx->tlsext_opaque_prf_input_callback = (int (*)(SSL *,void *, s
3688                 break;
3689 #endif
```

```
3691         case SSL_CTRL_SET_TLSEXT_STATUS_REQ_CB:
3692                 ctx->tlsext_status_cb=(int (*)(SSL *,void *))fp;
3693                 break;

3695         case SSL_CTRL_SET_TLSEXT_TICKET_KEY_CB:
3696                 ctx->tlsext_ticket_key_cb=(int (*)(SSL *,unsigned char  *,
3697                                                    unsigned char *,
3698                                                    EVP_CIPHER_CTX *,
3699                                                    HMAC_CTX *, int))fp;
3700                 break;

3702 #ifndef OPENSSL_NO_SRP
3703         case SSL_CTRL_SET_SRP_VERIFY_PARAM_CB:
3704                 ctx->srp_ctx.srp_Mask|=SSL_kSRP;
3705                 ctx->srp_ctx.SRP_verify_param_callback=(int (*)(SSL *,void *))fp
3706                 break;
3707         case SSL_CTRL_SET_TLS_EXT_SRP_USERNAME_CB:
3708                 ctx->srp_ctx.srp_Mask|=SSL_kSRP;
3709                 ctx->srp_ctx.TLS_ext_srp_username_callback=(int (*)(SSL *,int *,
3710                 break;
3711         case SSL_CTRL_SET_SRP_GIVE_CLIENT_PWD_CB:
3712                 ctx->srp_ctx.srp_Mask|=SSL_kSRP;
3713                 ctx->srp_ctx.SRP_give_srp_client_pwd_callback=(char *(*)(SSL *,v
3714                 break;
3715 #endif
3716 #endif
3717         default:
3718                 return(0);
3719                 }
3720         return(1);
3721         }

3723 /* This function needs to check if the ciphers required are actually
3724  * available */
3725 const SSL_CIPHER *ssl3_get_cipher_by_char(const unsigned char *p)
3726         {
3727         SSL_CIPHER c;
3728         const SSL_CIPHER *cp;
3729         unsigned long id;

3731         id=0x03000000L|((unsigned long)p[0]<<8L)|(unsigned long)p[1];
3732         c.id=id;
3733         cp = OBJ_bsearch_ssl_cipher_id(&c, ssl3_ciphers, SSL3_NUM_CIPHERS);
3734 #ifdef DEBUG_PRINT_UNKNOWN_CIPHERSUITES
3735 if (cp == NULL) fprintf(stderr, "Unknown cipher ID %x\n", (p[0] << 8) | p[1]);
3736 #endif
3737         if (cp == NULL || cp->valid == 0)
3738                 return NULL;
3739         else
3740                 return cp;
3741         }

3743 int ssl3_put_cipher_by_char(const SSL_CIPHER *c, unsigned char *p)
3744         {
3745         long l;

3747         if (p != NULL)
3748                 {
3749                 l=c->id;
3750                 if ((l & 0xff000000) != 0x03000000) return(0);
3751                 p[0]=((unsigned char)(l>> 8L))&0xFF;
3752                 p[1]=((unsigned char)(l     ))&0xFF;
3753                 }
3754         return(2);
3755         }
```



```
3757 SSL_CIPHER *ssl3_choose_cipher(SSL *s, STACK_OF(SSL_CIPHER) *clnt,
3758              STACK_OF(SSL_CIPHER) *srvr)
3759         {
3760         SSL_CIPHER *c,*ret=NULL;
3761         STACK_OF(SSL_CIPHER) *prio, *allow;
3762         int i,ii,ok;
3763 #if !defined(OPENSSL_NO_TLSEXT) && !defined(OPENSSL_NO_EC)
3764         unsigned int j;
3765         int ec_ok, ec_nid;
3766         unsigned char ec_search1 = 0, ec_search2 = 0;
3767 #endif
3768         CERT *cert;
3769         unsigned long alg_k,alg_a,mask_k,mask_a,emask_k,emask_a;

3771         /* Let's see which ciphers we can support */
3772         cert=s->cert;

3774 #if 0
3775         /* Do not set the compare functions, because this may lead to a
3776          * reordering by "id". We want to keep the original ordering.
3777          * We may pay a price in performance during sk_SSL_CIPHER_find(),
3778          * but would have to pay with the price of sk_SSL_CIPHER_dup().
3779          */
3780         sk_SSL_CIPHER_set_cmp_func(srvr, ssl_cipher_ptr_id_cmp);
3781         sk_SSL_CIPHER_set_cmp_func(clnt, ssl_cipher_ptr_id_cmp);
3782 #endif

3784 #ifdef CIPHER_DEBUG
3785         printf("Server has %d from %p:\n", sk_SSL_CIPHER_num(srvr), (void *)srvr
3786         for(i=0 ; i < sk_SSL_CIPHER_num(srvr) ; ++i)
3787                 {
3788                 c=sk_SSL_CIPHER_value(srvr,i);
3789                 printf("%p:%s\n",(void *)c,c->name);
3790                 }
3791         printf("Client sent %d from %p:\n", sk_SSL_CIPHER_num(clnt), (void *)cln
3792         for(i=0 ; i < sk_SSL_CIPHER_num(clnt) ; ++i)
3793             {
3794             c=sk_SSL_CIPHER_value(clnt,i);
3795             printf("%p:%s\n",(void *)c,c->name);
3796             }
3797 #endif

3799         if (s->options & SSL_OP_CIPHER_SERVER_PREFERENCE)
3800                 {
3801                 prio = srvr;
3802                 allow = clnt;
3803                 }
3804         else
3805                 {
3806                 prio = clnt;
3807                 allow = srvr;
3808                 }

3810         for (i=0; i<sk_SSL_CIPHER_num(prio); i++)
3811                 {
3812                 c=sk_SSL_CIPHER_value(prio,i);

3814                 /* Skip TLS v1.2 only ciphersuites if lower than v1.2 */
3815                 if ((c->algorithm_ssl & SSL_TLSV1_2) &&
3816                         (TLS1_get_version(s) < TLS1_2_VERSION))
3817                         continue;

3819                 ssl_set_cert_masks(cert,c);
3820                 mask_k = cert->mask_k;
3821                 mask_a = cert->mask_a;
3822                 emask_k = cert->export_mask_k;
```

```
3823                 emask_a = cert->export_mask_a;
3824 #ifndef OPENSSL_NO_SRP
3825                 mask_k=cert->mask_k | s->srp_ctx.srp_Mask;
3826                 emask_k=cert->export_mask_k | s->srp_ctx.srp_Mask;
3827 #endif

3829 #ifdef KSSL_DEBUG
3830 /*               printf("ssl3_choose_cipher %d alg= %lx\n", i,c->algorithms);*/
3831 #endif    /* KSSL_DEBUG */

3833                 alg_k=c->algorithm_mkey;
3834                 alg_a=c->algorithm_auth;

3836 #ifndef OPENSSL_NO_KRB5
3837                 if (alg_k & SSL_kKRB5)
3838                         {
3839                         if ( !kssl_keytab_is_available(s->kssl_ctx) )
3840                           continue;
3841                         }
3842 #endif /* OPENSSL_NO_KRB5 */
3843 #ifndef OPENSSL_NO_PSK
3844                 /* with PSK there must be server callback set */
3845                 if ((alg_k & SSL_kPSK) && s->psk_server_callback == NULL)
3846                         continue;
3847 #endif /* OPENSSL_NO_PSK */

3849                 if (SSL_C_IS_EXPORT(c))
3850                         {
3851                         ok = (alg_k & emask_k) && (alg_a & emask_a);
3852 #ifdef CIPHER_DEBUG
3853                         printf("%d:[%08lX:%08lX:%08lX:%08lX]%p:%s (export)\n",ok
3854                                 (void *)c,c->name);
3855 #endif
3856                         }
3857                 else
3858                         {
3859                         ok = (alg_k & mask_k) && (alg_a & mask_a);
3860 #ifdef CIPHER_DEBUG
3861                         printf("%d:[%08lX:%08lX:%08lX:%08lX]%p:%s\n",ok,alg_k,al
3862                                 c->name);
3863 #endif
3864                         }

3866 #ifndef OPENSSL_NO_TLSEXT
3867 #ifndef OPENSSL_NO_EC
3868                 if (
3869                         /* if we are considering an ECC cipher suite that uses o
3870                         (alg_a & SSL_aECDSA || alg_a & SSL_aECDH)
3871                         /* and we have an ECC certificate */
3872                         && (s->cert->pkeys[SSL_PKEY_ECC].x509 != NULL)
3873                         /* and the client specified a Supported Point Formats ex
3874                         && ((s->session->tlsext_ecpointformatlist_length > 0) &&
3875                         /* and our certificate's point is compressed */
3876                         && (
3877                                 (s->cert->pkeys[SSL_PKEY_ECC].x509->cert_info !=
3878                                 && (s->cert->pkeys[SSL_PKEY_ECC].x509->cert_info
3879                                 && (s->cert->pkeys[SSL_PKEY_ECC].x509->cert_info
3880                                 && (s->cert->pkeys[SSL_PKEY_ECC].x509->cert_info
3881                                 && (
3882                                         (*(s->cert->pkeys[SSL_PKEY_ECC].x509->ce
3883                                         || (*(s->cert->pkeys[SSL_PKEY_ECC].x509-
3884                                         )
3885                                 )
3886                 )
3887                         {
3888                         ec_ok = 0;
```



```
3889                         /* if our certificate's curve is over a field type that
3890                          * then do not allow this cipher suite to be negotiated
3891                         if (
3892                                 (s->cert->pkeys[SSL_PKEY_ECC].privatekey->pkey.e
3893                                 && (s->cert->pkeys[SSL_PKEY_ECC].privatekey->pke
3894                                 && (s->cert->pkeys[SSL_PKEY_ECC].privatekey->pke
3895                                 && (EC_METHOD_get_field_type(s->cert->pkeys[SSL_
3896                         )
3897                                 {
3898                                 for (j = 0; j < s->session->tlsext_ecpointformat
3899                                         {
3900                                         if (s->session->tlsext_ecpointformatlist
3901                                                 {
3902                                                 ec_ok = 1;
3903                                                 break;
3904                                                 }
3905                                         }
3906                                 }
3907                         else if (EC_METHOD_get_field_type(s->cert->pkeys[SSL_PKE
3908                                 {
3909                                 for (j = 0; j < s->session->tlsext_ecpointformat
3910                                         {
3911                                         if (s->session->tlsext_ecpointformatlist
3912                                                 {
3913                                                 ec_ok = 1;
3914                                                 break;
3915                                                 }
3916                                         }
3917                                 }
3918                         ok = ok && ec_ok;
3919                         }
3920                 if (
3921                         /* if we are considering an ECC cipher suite that uses o
3922                         (alg_a & SSL_aECDSA || alg_a & SSL_aECDH)
3923                         /* and we have an ECC certificate */
3924                         && (s->cert->pkeys[SSL_PKEY_ECC].x509 != NULL)
3925                         /* and the client specified an EllipticCurves extension
3926                         && ((s->session->tlsext_ellipticcurvelist_length > 0) &&
3927                 )
3928                         {
3929                         ec_ok = 0;
3930                         if (
3931                                 (s->cert->pkeys[SSL_PKEY_ECC].privatekey->pkey.e
3932                                 && (s->cert->pkeys[SSL_PKEY_ECC].privatekey->pke
3933                         )
3934                                 {
3935                                 ec_nid = EC_GROUP_get_curve_name(s->cert->pkeys[
3936                                 if ((ec_nid == 0)
3937                                         && (s->cert->pkeys[SSL_PKEY_ECC].private
3938                                 )
3939                                         {
3940                                         if (EC_METHOD_get_field_type(s->cert->pk
3941                                                 {
3942                                                 ec_search1 = 0xFF;
3943                                                 ec_search2 = 0x01;
3944                                                 }
3945                                         else if (EC_METHOD_get_field_type(s->cer
3946                                                 {
3947                                                 ec_search1 = 0xFF;
3948                                                 ec_search2 = 0x02;
3949                                                 }
3950                                         }
3951                                 else
3952                                         {
3953                                         ec_search1 = 0x00;
3954                                         ec_search2 = tls1_ec_nid2curve_id(ec_nid
```

```
3955                                    }
3956                            if ((ec_search1 != 0) || (ec_search2 != 0))
3957                                    {
3958                                    for (j = 0; j < s->session->tlsext_ellip
3959                                            {
3960                                            if ((s->session->tlsext_elliptic
3961                                                    {
3962                                                    ec_ok = 1;
3963                                                    break;
3964                                                    }
3965                                            }
3966                                    }
3967                            }
3968                    ok = ok && ec_ok;
3969                    }
3970            if (
3971                    /* if we are considering an ECC cipher suite that uses a
3972                    (alg_k & SSL_kEECDH)
3973                    /* and we have an ephemeral EC key */
3974                    && (s->cert->ecdh_tmp != NULL)
3975                    /* and the client specified an EllipticCurves extension
3976                    && ((s->session->tlsext_ellipticcurvelist_length > 0) &&
3977            )
3978                    {
3979                    ec_ok = 0;
3980                    if (s->cert->ecdh_tmp->group != NULL)
3981                            {
3982                            ec_nid = EC_GROUP_get_curve_name(s->cert->ecdh_t
3983                            if ((ec_nid == 0)
3984                                    && (s->cert->ecdh_tmp->group->meth != NU
3985                            )
3986                                    {
3987                                    if (EC_METHOD_get_field_type(s->cert->ec
3988                                            {
3989                                            ec_search1 = 0xFF;
3990                                            ec_search2 = 0x01;
3991                                            }
3992                                    else if (EC_METHOD_get_field_type(s->cer
3993                                            {
3994                                            ec_search1 = 0xFF;
3995                                            ec_search2 = 0x02;
3996                                            }
3997                                    }
3998                            else
3999                                    {
4000                                    ec_search1 = 0x00;
4001                                    ec_search2 = tls1_ec_nid2curve_id(ec_nid
4002                                    }
4003                            if ((ec_search1 != 0) || (ec_search2 != 0))
4004                                    {
4005                                    for (j = 0; j < s->session->tlsext_ellip
4006                                            {
4007                                            if ((s->session->tlsext_elliptic
4008                                                    {
4009                                                    ec_ok = 1;
4010                                                    break;
4011                                                    }
4012                                            }
4013                                    }
4014                            }
4015                    ok = ok && ec_ok;
4016                    }
4017 #endif /* OPENSSL_NO_EC */
4018 #endif /* OPENSSL_NO_TLSEXT */

4020                    if (!ok) continue;
```



```
4021                    ii=sk_SSL_CIPHER_find(allow,c);
4022                    if (ii >= 0)
4023                            {
4024 #if !defined(OPENSSL_NO_EC) && !defined(OPENSSL_NO_TLSEXT)
4025                            if ((alg_k & SSL_kEECDH) && (alg_a & SSL_aECDSA) && s->s
4026                                    {
4027                                    if (!ret) ret=sk_SSL_CIPHER_value(allow,ii);
4028                                    continue;
4029                                    }
4030 #endif
4031                            ret=sk_SSL_CIPHER_value(allow,ii);
4032                            break;
4033                            }
4034                    }
4035            return(ret);
4036            }

4038 int ssl3_get_req_cert_type(SSL *s, unsigned char *p)
4039            {
4040            int ret=0;
4041            unsigned long alg_k;

4043            alg_k = s->s3->tmp.new_cipher->algorithm_mkey;

4045 #ifndef OPENSSL_NO_GOST
4046            if (s->version >= TLS1_VERSION)
4047                    {
4048                    if (alg_k & SSL_kGOST)
4049                            {
4050                            p[ret++]=TLS_CT_GOST94_SIGN;
4051                            p[ret++]=TLS_CT_GOST01_SIGN;
4052                            return(ret);
4053                            }
4054                    }
4055 #endif

4057 #ifndef OPENSSL_NO_DH
4058            if (alg_k & (SSL_kDHr|SSL_kEDH))
4059                    {
4060 #  ifndef OPENSSL_NO_RSA
4061                    p[ret++]=SSL3_CT_RSA_FIXED_DH;
4062 #  endif
4063 #  ifndef OPENSSL_NO_DSA
4064                    p[ret++]=SSL3_CT_DSS_FIXED_DH;
4065 #  endif
4066                    }
4067            if ((s->version == SSL3_VERSION) &&
4068                    (alg_k & (SSL_kEDH|SSL_kDHd|SSL_kDHr)))
4069                    {
4070 #  ifndef OPENSSL_NO_RSA
4071                    p[ret++]=SSL3_CT_RSA_EPHEMERAL_DH;
4072 #  endif
4073 #  ifndef OPENSSL_NO_DSA
4074                    p[ret++]=SSL3_CT_DSS_EPHEMERAL_DH;
4075 #  endif
4076                    }
4077 #endif /* !OPENSSL_NO_DH */
4078 #ifndef OPENSSL_NO_RSA
4079            p[ret++]=SSL3_CT_RSA_SIGN;
4080 #endif
4081 #ifndef OPENSSL_NO_DSA
4082            p[ret++]=SSL3_CT_DSS_SIGN;
4083 #endif
4084 #ifndef OPENSSL_NO_ECDH
4085            if ((alg_k & (SSL_kECDHr|SSL_kECDHe)) && (s->version >= TLS1_VERSION))
4086                    {
```

```
4087                 p[ret++]=TLS_CT_RSA_FIXED_ECDH;
4088                 p[ret++]=TLS_CT_ECDSA_FIXED_ECDH;
4089                 }
4090 #endif

4092 #ifndef OPENSSL_NO_ECDSA
4093         /* ECDSA certs can be used with RSA cipher suites as well
4094          * so we don't need to check for SSL_kECDH or SSL_kEECDH
4095          */
4096         if (s->version >= TLS1_VERSION)
4097                 {
4098                 p[ret++]=TLS_CT_ECDSA_SIGN;
4099                 }
4100 #endif
4101         return(ret);
4102         }

4104 int ssl3_shutdown(SSL *s)
4105         {
4106         int ret;

4108         /* Don't do anything much if we have not done the handshake or
4109          * we don't want to send messages :-) */
4110         if ((s->quiet_shutdown) || (s->state == SSL_ST_BEFORE))
4111                 {
4112                 s->shutdown=(SSL_SENT_SHUTDOWN|SSL_RECEIVED_SHUTDOWN);
4113                 return(1);
4114                 }

4116         if (!(s->shutdown & SSL_SENT_SHUTDOWN))
4117                 {
4118                 s->shutdown|=SSL_SENT_SHUTDOWN;
4119 #if 1
4120                 ssl3_send_alert(s,SSL3_AL_WARNING,SSL_AD_CLOSE_NOTIFY);
4121 #endif
4122                 /* our shutdown alert has been sent now, and if it still needs
4123                  * to be written, s->s3->alert_dispatch will be true */
4124                 if (s->s3->alert_dispatch)
4125                         return(-1);     /* return WANT_WRITE */
4126                 }
4127         else if (s->s3->alert_dispatch)
4128                 {
4129                 /* resend it if not sent */
4130 #if 1
4131                 ret=s->method->ssl_dispatch_alert(s);
4132                 if(ret == -1)
4133                         {
4134                         /* we only get to return -1 here the 2nd/Nth
4135                          * invocation, we must  have already signalled
4136                          * return 0 upon a previous invoation,
4137                          * return WANT_WRITE */
4138                         return(ret);
4139                         }
4140 #endif
4141                 }
4142         else if (!(s->shutdown & SSL_RECEIVED_SHUTDOWN))
4143                 {
4144                 /* If we are waiting for a close from our peer, we are closed */
4145                 s->method->ssl_read_bytes(s,0,NULL,0,0);
4146                 if(!(s->shutdown & SSL_RECEIVED_SHUTDOWN))
4147                         {
4148                         return(-1);     /* return WANT_READ */
4149                         }
4150                 }

4152         if ((s->shutdown == (SSL_SENT_SHUTDOWN|SSL_RECEIVED_SHUTDOWN)) &&
```



```
4153                 !s->s3->alert_dispatch)
4154                 return(1);
4155         else
4156                 return(0);
4157         }

4159 int ssl3_write(SSL *s, const void *buf, int len)
4160         {
4161         int ret,n;

4163 #if 0
4164         if (s->shutdown & SSL_SEND_SHUTDOWN)
4165                 {
4166                 s->rwstate=SSL_NOTHING;
4167                 return(0);
4168                 }
4169 #endif
4170         clear_sys_error();
4171         if (s->s3->renegotiate) ssl3_renegotiate_check(s);

4173         /* This is an experimental flag that sends the
4174          * last handshake message in the same packet as the first
4175          * use data - used to see if it helps the TCP protocol during
4176          * session-id reuse */
4177         /* The second test is because the buffer may have been removed */
4178         if ((s->s3->flags & SSL3_FLAGS_POP_BUFFER) && (s->wbio == s->bbio))
4179                 {
4180                 /* First time through, we write into the buffer */
4181                 if (s->s3->delay_buf_pop_ret == 0)
4182                         {
4183                         ret=ssl3_write_bytes(s,SSL3_RT_APPLICATION_DATA,
4184                                                 buf,len);
4185                         if (ret <= 0) return(ret);

4187                         s->s3->delay_buf_pop_ret=ret;
4188                         }

4190                 s->rwstate=SSL_WRITING;
4191                 n=BIO_flush(s->wbio);
4192                 if (n <= 0) return(n);
4193                 s->rwstate=SSL_NOTHING;

4195                 /* We have flushed the buffer, so remove it */
4196                 ssl_free_wbio_buffer(s);
4197                 s->s3->flags&= ~SSL3_FLAGS_POP_BUFFER;

4199                 ret=s->s3->delay_buf_pop_ret;
4200                 s->s3->delay_buf_pop_ret=0;
4201                 }
4202         else
4203                 {
4204                 ret=s->method->ssl_write_bytes(s,SSL3_RT_APPLICATION_DATA,
4205                         buf,len);
4206                 if (ret <= 0) return(ret);
4207                 }

4209         return(ret);
4210         }

4212 static int ssl3_read_internal(SSL *s, void *buf, int len, int peek)
4213         {
4214         int ret;

4216         clear_sys_error();
4217         if (s->s3->renegotiate) ssl3_renegotiate_check(s);
4218         s->s3->in_read_app_data=1;
```

```
4219         ret=s->method->ssl_read_bytes(s,SSL3_RT_APPLICATION_DATA,buf,len,peek);
4220         if ((ret == -1) && (s->s3->in_read_app_data == 2))
4221                 {
4222                 /* ssl3_read_bytes decided to call s->handshake_func, which
4223                  * called ssl3_read_bytes to read handshake data.
4224                  * However, ssl3_read_bytes actually found application data
4225                  * and thinks that application data makes sense here; so disable
4226                  * handshake processing and try to read application data again.
4227                 s->in_handshake++;
4228                 ret=s->method->ssl_read_bytes(s,SSL3_RT_APPLICATION_DATA,buf,len
4229                 s->in_handshake--;
4230                 }
4231         else
4232                 s->s3->in_read_app_data=0;

4234         return(ret);
4235         }

4237 int ssl3_read(SSL *s, void *buf, int len)
4238         {
4239         return ssl3_read_internal(s, buf, len, 0);
4240         }

4242 int ssl3_peek(SSL *s, void *buf, int len)
4243         {
4244         return ssl3_read_internal(s, buf, len, 1);
4245         }

4247 int ssl3_renegotiate(SSL *s)
4248         {
4249         if (s->handshake_func == NULL)
4250                 return(1);

4252         if (s->s3->flags & SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS)
4253                 return(0);

4255         s->s3->renegotiate=1;
4256         return(1);
4257         }

4259 int ssl3_renegotiate_check(SSL *s)
4260         {
4261         int ret=0;

4263         if (s->s3->renegotiate)
4264                 {
4265                 if (    (s->s3->rbuf.left == 0) &&
4266                         (s->s3->wbuf.left == 0) &&
4267                         !SSL_in_init(s))
4268                         {
4269 /*
4270 if we are the server, and we have sent a 'RENEGOTIATE' message, we
4271 need to go to SSL_ST_ACCEPT.
4272 */
4273                         /* SSL_ST_ACCEPT */
4274                         s->state=SSL_ST_RENEGOTIATE;
4275                         s->s3->renegotiate=0;
4276                         s->s3->num_renegotiations++;
4277                         s->s3->total_renegotiations++;
4278                         ret=1;
4279                         }
4280                 }
4281         return(ret);
4282         }
4283 /* If we are using TLS v1.2 or later and default SHA1+MD5 algorithms switch
4284  * to new SHA256 PRF and handshake macs
```



```
4285  */
4286 long ssl_get_algorithm2(SSL *s)
4287         {
4288         long alg2 = s->s3->tmp.new_cipher->algorithm2;
4289         if (s->method->version == TLS1_2_VERSION &&
4290             alg2 == (SSL_HANDSHAKE_MAC_DEFAULT|TLS1_PRF))
4291                 return SSL_HANDSHAKE_MAC_SHA256 | TLS1_PRF_SHA256;
4292         return alg2;
4293         }
```