```
**********************************************************
  126040 Wed Feb 10 17:30:49 2016
new/usr/src/uts/common/crypto/io/dca.c
6640 dca gets the instance number a lot, never actually uses it
**********************************************************
______unchanged_portion_omitted_

/* Convenience macros */
#define DCA_SOFTC_FROM_CTX(ctx) ((dca_t *)(ctx)->cc_provider)
/* Retrieve the softc and instance number from a SPI crypto context */
#define DCA_SOFTC_FROM_CTX(ctx, softc, instance) {              \
        (softc) = (dca_t *)(ctx)->cc_provider;                  \
        (instance) = ddi_get_instance((softc)->dca_dip);        \
}

#define DCA_MECH_FROM_CTX(ctx) \
        (((dca_request_t *)(ctx)->cc_provider_private)->dr_ctx.ctx_cm_type)

static int dca_bindchains_one(dca_request_t *reqp, size_t cnt, int dr_offset,
    caddr_t kaddr, ddi_dma_handle_t handle, uint_t flags,
    dca_chain_t *head, int *n_chain);
static uint64_t dca_ena(uint64_t ena);
static caddr_t dca_bufdaddr_out(crypto_data_t *data);
static char *dca_fma_eclass_string(char *model, dca_fma_eclass_t index);
static int dca_check_acc_handle(dca_t *dca, ddi_acc_handle_t handle,
    dca_fma_eclass_t eclass_index);

static void dca_fma_init(dca_t *dca);
static void dca_fma_fini(dca_t *dca);
static int dca_fm_error_cb(dev_info_t *dip, ddi_fm_error_t *err,
    const void *impl_data);


static dca_device_t dca_devices[] = {
        /* Broadcom vanilla variants */
        {       0x14e4, 0x5820, "Broadcom 5820" },
        {       0x14e4, 0x5821, "Broadcom 5821" },
        {       0x14e4, 0x5822, "Broadcom 5822" },
        {       0x14e4, 0x5825, "Broadcom 5825" },
        /* Sun specific OEMd variants */
        {       0x108e, 0x5454, "SCA" },
        {       0x108e, 0x5455, "SCA 1000" },
        {       0x108e, 0x5457, "SCA 500" },
        /* subsysid should be 0x5457, but got 0x1 from HW. Assume both here. */
        {       0x108e, 0x1, "SCA 500" },
};
______unchanged_portion_omitted_

/*
 * Cipher (encrypt/decrypt) entry points.
 */

/* ARGSUSED */
static int
dca_encrypt_init(crypto_ctx_t *ctx, crypto_mechanism_t *mechanism,
    crypto_key_t *key, crypto_spi_ctx_template_t ctx_template,
    crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
```

```
        DBG(softc, DENTRY, "dca_encrypt_init: started");

        /* check mechanism */
        switch (mechanism->cm_type) {
        case DES_CBC_MECH_INFO_TYPE:
                error = dca_3desctxinit(ctx, mechanism, key, KM_SLEEP,
                    DR_ENCRYPT);
                break;
        case DES3_CBC_MECH_INFO_TYPE:
                error = dca_3desctxinit(ctx, mechanism, key, KM_SLEEP,
                    DR_ENCRYPT | DR_TRIPLE);
                break;
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsainit(ctx, mechanism, key, KM_SLEEP);
                break;
        default:
                cmn_err(CE_WARN, "dca_encrypt_init: unexpected mech type "
                    "0x%llx\n", (unsigned long long)mechanism->cm_type);
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_encrypt_init: done, err = 0x%x", error);

        if (error == CRYPTO_SUCCESS)
                dca_enlist2(&softc->dca_ctx_list, ctx->cc_provider_private,
                    &softc->dca_ctx_list_lock);

        return (error);
}

/* ARGSUSED */
static int
dca_encrypt(crypto_ctx_t *ctx, crypto_data_t *plaintext,
    crypto_data_t *ciphertext, crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_encrypt: started");

        /* handle inplace ops */
        if (!ciphertext) {
                dca_request_t *reqp = ctx->cc_provider_private;
                reqp->dr_flags |= DR_INPLACE;
                ciphertext = plaintext;
        }

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case DES_CBC_MECH_INFO_TYPE:
                error = dca_3des(ctx, plaintext, ciphertext, req, DR_ENCRYPT);
                break;
        case DES3_CBC_MECH_INFO_TYPE:
                error = dca_3des(ctx, plaintext, ciphertext, req,
                    DR_ENCRYPT | DR_TRIPLE);
                break;
        case RSA_PKCS_MECH_INFO_TYPE:
```

```
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsastart(ctx, plaintext, ciphertext, req,
                    DCA_RSA_ENC);
                break;
        default:
                /* Should never reach here */
                cmn_err(CE_WARN, "dca_encrypt: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                error = CRYPTO_MECHANISM_INVALID;
        }

        if ((error != CRYPTO_QUEUED) && (error != CRYPTO_SUCCESS) &&
            (error != CRYPTO_BUFFER_TOO_SMALL)) {
                ciphertext->cd_length = 0;
        }

        DBG(softc, DENTRY, "dca_encrypt: done, err = 0x%x", error);

        return (error);
}

/* ARGSUSED */
static int
dca_encrypt_update(crypto_ctx_t *ctx, crypto_data_t *plaintext,
    crypto_data_t *ciphertext, crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_encrypt_update: started");

        /* handle inplace ops */
        if (!ciphertext) {
                dca_request_t *reqp = ctx->cc_provider_private;
                reqp->dr_flags |= DR_INPLACE;
                ciphertext = plaintext;
        }

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case DES_CBC_MECH_INFO_TYPE:
                error = dca_3desupdate(ctx, plaintext, ciphertext, req,
                    DR_ENCRYPT);
                break;
        case DES3_CBC_MECH_INFO_TYPE:
                error = dca_3desupdate(ctx, plaintext, ciphertext, req,
                    DR_ENCRYPT | DR_TRIPLE);
                break;
        default:
                /* Should never reach here */
                cmn_err(CE_WARN, "dca_encrypt_update: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_encrypt_update: done, err = 0x%x", error);

        return (error);
```

```
}

/* ARGSUSED */
static int
dca_encrypt_final(crypto_ctx_t *ctx, crypto_data_t *ciphertext,
    crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_encrypt_final: started");

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case DES_CBC_MECH_INFO_TYPE:
                error = dca_3desfinal(ctx, ciphertext, DR_ENCRYPT);
                break;
        case DES3_CBC_MECH_INFO_TYPE:
                error = dca_3desfinal(ctx, ciphertext, DR_ENCRYPT | DR_TRIPLE);
                break;
        default:
                /* Should never reach here */
                cmn_err(CE_WARN, "dca_encrypt_final: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_encrypt_final: done, err = 0x%x", error);

        return (error);
}
______unchanged_portion_omitted_

/* ARGSUSED */
static int
dca_decrypt_init(crypto_ctx_t *ctx, crypto_mechanism_t *mechanism,
    crypto_key_t *key, crypto_spi_ctx_template_t ctx_template,
    crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_decrypt_init: started");

        /* check mechanism */
        switch (mechanism->cm_type) {
        case DES_CBC_MECH_INFO_TYPE:
                error = dca_3desctxinit(ctx, mechanism, key, KM_SLEEP,
                    DR_DECRYPT);
                break;
        case DES3_CBC_MECH_INFO_TYPE:
                error = dca_3desctxinit(ctx, mechanism, key, KM_SLEEP,
                    DR_DECRYPT | DR_TRIPLE);
```

```
                        break;
                case RSA_PKCS_MECH_INFO_TYPE:
                case RSA_X_509_MECH_INFO_TYPE:
                        error = dca_rsainit(ctx, mechanism, key, KM_SLEEP);
                        break;
                default:
                        cmn_err(CE_WARN, "dca_decrypt_init: unexpected mech type "
                            "0x%llx\n", (unsigned long long)mechanism->cm_type);
                        error = CRYPTO_MECHANISM_INVALID;
                }

                DBG(softc, DENTRY, "dca_decrypt_init: done, err = 0x%x", error);

                if (error == CRYPTO_SUCCESS)
                        dca_enlist2(&softc->dca_ctx_list, ctx->cc_provider_private,
                            &softc->dca_ctx_list_lock);

                return (error);
        }

        /* ARGSUSED */
        static int
        dca_decrypt(crypto_ctx_t *ctx, crypto_data_t *ciphertext,
            crypto_data_t *plaintext, crypto_req_handle_t req)
        {
                int error = CRYPTO_FAILED;
                dca_t *softc;
                /* LINTED E_FUNC_SET_NOT_USED */
                int instance;

                if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                        return (CRYPTO_OPERATION_NOT_INITIALIZED);

                softc = DCA_SOFTC_FROM_CTX(ctx);
                /* extract softc and instance number from context */
                DCA_SOFTC_FROM_CTX(ctx, softc, instance);
                DBG(softc, DENTRY, "dca_decrypt: started");

                /* handle inplace ops */
                if (!plaintext) {
                        dca_request_t *reqp = ctx->cc_provider_private;
                        reqp->dr_flags |= DR_INPLACE;
                        plaintext = ciphertext;
                }

                /* check mechanism */
                switch (DCA_MECH_FROM_CTX(ctx)) {
                case DES_CBC_MECH_INFO_TYPE:
                        error = dca_3des(ctx, ciphertext, plaintext, req, DR_DECRYPT);
                        break;
                case DES3_CBC_MECH_INFO_TYPE:
                        error = dca_3des(ctx, ciphertext, plaintext, req,
                            DR_DECRYPT | DR_TRIPLE);
                        break;
                case RSA_PKCS_MECH_INFO_TYPE:
                case RSA_X_509_MECH_INFO_TYPE:
                        error = dca_rsastart(ctx, ciphertext, plaintext, req,
                            DCA_RSA_DEC);
                        break;
                default:
                        /* Should never reach here */
                        cmn_err(CE_WARN, "dca_decrypt: unexpected mech type "
                            "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                        error = CRYPTO_MECHANISM_INVALID;
                }
```

```
                if ((error != CRYPTO_QUEUED) && (error != CRYPTO_SUCCESS) &&
                    (error != CRYPTO_BUFFER_TOO_SMALL)) {
                        if (plaintext)
                                plaintext->cd_length = 0;
                }

                DBG(softc, DENTRY, "dca_decrypt: done, err = 0x%x", error);

                return (error);
        }

        /* ARGSUSED */
        static int
        dca_decrypt_update(crypto_ctx_t *ctx, crypto_data_t *ciphertext,
            crypto_data_t *plaintext, crypto_req_handle_t req)
        {
                int error = CRYPTO_FAILED;
                dca_t *softc;
                /* LINTED E_FUNC_SET_NOT_USED */
                int instance;

                if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                        return (CRYPTO_OPERATION_NOT_INITIALIZED);

                softc = DCA_SOFTC_FROM_CTX(ctx);
                /* extract softc and instance number from context */
                DCA_SOFTC_FROM_CTX(ctx, softc, instance);
                DBG(softc, DENTRY, "dca_decrypt_update: started");

                /* handle inplace ops */
                if (!plaintext) {
                        dca_request_t *reqp = ctx->cc_provider_private;
                        reqp->dr_flags |= DR_INPLACE;
                        plaintext = ciphertext;
                }

                /* check mechanism */
                switch (DCA_MECH_FROM_CTX(ctx)) {
                case DES_CBC_MECH_INFO_TYPE:
                        error = dca_3desupdate(ctx, ciphertext, plaintext, req,
                            DR_DECRYPT);
                        break;
                case DES3_CBC_MECH_INFO_TYPE:
                        error = dca_3desupdate(ctx, ciphertext, plaintext, req,
                            DR_DECRYPT | DR_TRIPLE);
                        break;
                default:
                        /* Should never reach here */
                        cmn_err(CE_WARN, "dca_decrypt_update: unexpected mech type "
                            "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                        error = CRYPTO_MECHANISM_INVALID;
                }

                DBG(softc, DENTRY, "dca_decrypt_update: done, err = 0x%x", error);

                return (error);
        }

        /* ARGSUSED */
        static int
        dca_decrypt_final(crypto_ctx_t *ctx, crypto_data_t *plaintext,
            crypto_req_handle_t req)
        {
                int error = CRYPTO_FAILED;
                dca_t *softc;
                /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_decrypt_final: started");

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case DES_CBC_MECH_INFO_TYPE:
                error = dca_3desfinal(ctx, plaintext, DR_DECRYPT);
                break;
        case DES3_CBC_MECH_INFO_TYPE:
                error = dca_3desfinal(ctx, plaintext, DR_DECRYPT | DR_TRIPLE);
                break;
        default:
                /* Should never reach here */
                cmn_err(CE_WARN, "dca_decrypt_final: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_decrypt_final: done, err = 0x%x", error);

        return (error);
}
______unchanged_portion_omitted_

/*
 * Sign entry points.
 */

/* ARGSUSED */
static int
dca_sign_init(crypto_ctx_t *ctx, crypto_mechanism_t *mechanism,
    crypto_key_t *key, crypto_spi_ctx_template_t ctx_template,
    crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_sign_init: started\n");

        if (ctx_template != NULL)
                return (CRYPTO_ARGUMENTS_BAD);

        /* check mechanism */
        switch (mechanism->cm_type) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsainit(ctx, mechanism, key, KM_SLEEP);
                break;
        case DSA_MECH_INFO_TYPE:
                error = dca_dsainit(ctx, mechanism, key, KM_SLEEP,
                    DCA_DSA_SIGN);
                break;
        default:
                cmn_err(CE_WARN, "dca_sign_init: unexpected mech type "
```

```
                    "0x%llx\n", (unsigned long long)mechanism->cm_type);
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_sign_init: done, err = 0x%x", error);

        if (error == CRYPTO_SUCCESS)
                dca_enlist2(&softc->dca_ctx_list, ctx->cc_provider_private,
                    &softc->dca_ctx_list_lock);

        return (error);
}

static int
dca_sign(crypto_ctx_t *ctx, crypto_data_t *data,
    crypto_data_t *signature, crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_sign: started\n");

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsastart(ctx, data, signature, req, DCA_RSA_SIGN);
                break;
        case DSA_MECH_INFO_TYPE:
                error = dca_dsa_sign(ctx, data, signature, req);
                break;
        default:
                cmn_err(CE_WARN, "dca_sign: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_sign: done, err = 0x%x", error);

        return (error);
}

/* ARGSUSED */
static int
dca_sign_update(crypto_ctx_t *ctx, crypto_data_t *data,
    crypto_req_handle_t req)
{
        int error = CRYPTO_MECHANISM_INVALID;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_sign_update: started\n");

        cmn_err(CE_WARN, "dca_sign_update: unexpected mech type "
            "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));

        DBG(softc, DENTRY, "dca_sign_update: done, err = 0x%x", error);

        return (error);
}

/* ARGSUSED */
static int
dca_sign_final(crypto_ctx_t *ctx, crypto_data_t *signature,
    crypto_req_handle_t req)
{
        int error = CRYPTO_MECHANISM_INVALID;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_sign_final: started\n");

        cmn_err(CE_WARN, "dca_sign_final: unexpected mech type "
            "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));

        DBG(softc, DENTRY, "dca_sign_final: done, err = 0x%x", error);

        return (error);
}
```

______***unchanged_portion_omitted_***

```
/* ARGSUSED */
static int
dca_sign_recover_init(crypto_ctx_t *ctx, crypto_mechanism_t *mechanism,
    crypto_key_t *key, crypto_spi_ctx_template_t ctx_template,
    crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_sign_recover_init: started\n");

        if (ctx_template != NULL)
                return (CRYPTO_ARGUMENTS_BAD);

        /* check mechanism */
        switch (mechanism->cm_type) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsainit(ctx, mechanism, key, KM_SLEEP);
                break;
        default:
                cmn_err(CE_WARN, "dca_sign_recover_init: unexpected mech type "
                    "0x%llx\n", (unsigned long long)mechanism->cm_type);
                error = CRYPTO_MECHANISM_INVALID;
```

```
        }

        DBG(softc, DENTRY, "dca_sign_recover_init: done, err = 0x%x", error);

        if (error == CRYPTO_SUCCESS)
                dca_enlist2(&softc->dca_ctx_list, ctx->cc_provider_private,
                    &softc->dca_ctx_list_lock);

        return (error);
}

static int
dca_sign_recover(crypto_ctx_t *ctx, crypto_data_t *data,
    crypto_data_t *signature, crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_sign_recover: started\n");

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsastart(ctx, data, signature, req, DCA_RSA_SIGNR);
                break;
        default:
                cmn_err(CE_WARN, "dca_sign_recover: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_sign_recover: done, err = 0x%x", error);

        return (error);
}

static int
dca_sign_recover_atomic(crypto_provider_handle_t provider,
    crypto_session_id_t session_id, crypto_mechanism_t *mechanism,
    crypto_key_t *key, crypto_data_t *data, crypto_data_t *signature,
    crypto_spi_ctx_template_t ctx_template, crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc = (dca_t *)provider;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        instance = ddi_get_instance(softc->dca_dip);
        DBG(softc, DENTRY, "dca_sign_recover_atomic: started\n");

        if (ctx_template != NULL)
                return (CRYPTO_ARGUMENTS_BAD);

        /* check mechanism */
        switch (mechanism->cm_type) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsaatomic(provider, session_id, mechanism, key,
                    data, signature, KM_SLEEP, req, DCA_RSA_SIGNR);
                break;
        default:
                cmn_err(CE_WARN, "dca_sign_recover_atomic: unexpected mech type"
                    " 0x%llx\n", (unsigned long long)mechanism->cm_type);
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_sign_recover_atomic: done, err = 0x%x", error);

        return (error);
}

/*
 * Verify entry points.
 */

/* ARGSUSED */
static int
dca_verify_init(crypto_ctx_t *ctx, crypto_mechanism_t *mechanism,
    crypto_key_t *key, crypto_spi_ctx_template_t ctx_template,
    crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_verify_init: started\n");

        if (ctx_template != NULL)
                return (CRYPTO_ARGUMENTS_BAD);

        /* check mechanism */
        switch (mechanism->cm_type) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsainit(ctx, mechanism, key, KM_SLEEP);
                break;
        case DSA_MECH_INFO_TYPE:
                error = dca_dsainit(ctx, mechanism, key, KM_SLEEP,
                    DCA_DSA_VRFY);
                break;
        default:
                cmn_err(CE_WARN, "dca_verify_init: unexpected mech type "
                    "0x%llx\n", (unsigned long long)mechanism->cm_type);
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_verify_init: done, err = 0x%x", error);

        if (error == CRYPTO_SUCCESS)
                dca_enlist2(&softc->dca_ctx_list, ctx->cc_provider_private,
                    &softc->dca_ctx_list_lock);

        return (error);
}

static int
dca_verify(crypto_ctx_t *ctx, crypto_data_t *data, crypto_data_t *signature,
    crypto_req_handle_t req)
{
```

```
        int error = CRYPTO_FAILED;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_verify: started\n");

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsastart(ctx, signature, data, req, DCA_RSA_VRFY);
                break;
        case DSA_MECH_INFO_TYPE:
                error = dca_dsa_verify(ctx, data, signature, req);
                break;
        default:
                cmn_err(CE_WARN, "dca_verify: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
                error = CRYPTO_MECHANISM_INVALID;
        }

        DBG(softc, DENTRY, "dca_verify: done, err = 0x%x", error);

        return (error);
}

/* ARGSUSED */
static int
dca_verify_update(crypto_ctx_t *ctx, crypto_data_t *data,
    crypto_req_handle_t req)
{
        int error = CRYPTO_MECHANISM_INVALID;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_verify_update: started\n");

        cmn_err(CE_WARN, "dca_verify_update: unexpected mech type "
            "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));

        DBG(softc, DENTRY, "dca_verify_update: done, err = 0x%x", error);

        return (error);
}

/* ARGSUSED */
static int
dca_verify_final(crypto_ctx_t *ctx, crypto_data_t *signature,
    crypto_req_handle_t req)
{
        int error = CRYPTO_MECHANISM_INVALID;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_verify_final: started\n");

        cmn_err(CE_WARN, "dca_verify_final: unexpected mech type "
            "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));

        DBG(softc, DENTRY, "dca_verify_final: done, err = 0x%x", error);

        return (error);
}
______unchanged_portion_omitted_

/* ARGSUSED */
static int
dca_verify_recover_init(crypto_ctx_t *ctx, crypto_mechanism_t *mechanism,
    crypto_key_t *key, crypto_spi_ctx_template_t ctx_template,
    crypto_req_handle_t req)
{
        int error = CRYPTO_MECHANISM_INVALID;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_verify_recover_init: started\n");

        if (ctx_template != NULL)
                return (CRYPTO_ARGUMENTS_BAD);

        /* check mechanism */
        switch (mechanism->cm_type) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsainit(ctx, mechanism, key, KM_SLEEP);
                break;
        default:
                cmn_err(CE_WARN, "dca_verify_recover_init: unexpected mech type"
                    " 0x%llx\n", (unsigned long long)mechanism->cm_type);
        }

        DBG(softc, DENTRY, "dca_verify_recover_init: done, err = 0x%x", error);

        if (error == CRYPTO_SUCCESS)
                dca_enlist2(&softc->dca_ctx_list, ctx->cc_provider_private,
                    &softc->dca_ctx_list_lock);

        return (error);
}

static int
dca_verify_recover(crypto_ctx_t *ctx, crypto_data_t *signature,
    crypto_data_t *data, crypto_req_handle_t req)
{
        int error = CRYPTO_MECHANISM_INVALID;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
```

```
        int instance;

        if (!ctx || !ctx->cc_provider || !ctx->cc_provider_private)
                return (CRYPTO_OPERATION_NOT_INITIALIZED);

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_verify_recover: started\n");

        /* check mechanism */
        switch (DCA_MECH_FROM_CTX(ctx)) {
        case RSA_PKCS_MECH_INFO_TYPE:
        case RSA_X_509_MECH_INFO_TYPE:
                error = dca_rsastart(ctx, signature, data, req, DCA_RSA_VRFYR);
                break;
        default:
                cmn_err(CE_WARN, "dca_verify_recover: unexpected mech type "
                    "0x%llx\n", (unsigned long long)DCA_MECH_FROM_CTX(ctx));
        }

        DBG(softc, DENTRY, "dca_verify_recover: done, err = 0x%x", error);

        return (error);
}
______unchanged_portion_omitted_

/*
 * Random number entry points.
 */

/* ARGSUSED */
static int
dca_generate_random(crypto_provider_handle_t provider,
    crypto_session_id_t session_id,
    uchar_t *buf, size_t len, crypto_req_handle_t req)
{
        int error = CRYPTO_FAILED;
        dca_t *softc = (dca_t *)provider;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        instance = ddi_get_instance(softc->dca_dip);
        DBG(softc, DENTRY, "dca_generate_random: started");

        error = dca_rng(softc, buf, len, req);

        DBG(softc, DENTRY, "dca_generate_random: done, err = 0x%x", error);

        return (error);
}

/*
 * Context management entry points.
 */

int
dca_free_context(crypto_ctx_t *ctx)
{
        int error = CRYPTO_SUCCESS;
        dca_t *softc;
        /* LINTED E_FUNC_SET_NOT_USED */
        int instance;

        softc = DCA_SOFTC_FROM_CTX(ctx);
        /* extract softc and instance number from context */
        DCA_SOFTC_FROM_CTX(ctx, softc, instance);
        DBG(softc, DENTRY, "dca_free_context: entered");

        if (ctx->cc_provider_private == NULL)
                return (error);

        dca_rmlist2(ctx->cc_provider_private, &softc->dca_ctx_list_lock);

        error = dca_free_context_low(ctx);

        DBG(softc, DENTRY, "dca_free_context: done, err = 0x%x", error);

        return (error);
}
```
______***unchanged_portion_omitted_***

****
    2401 Wed Feb 10 17:30:50 2016
new/usr/src/uts/intel/dca/Makefile
6640 dca gets the instance number a lot, never actually uses it
****

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
#       This makefile drives the production of the DCA kCF provider.
#
#       intel implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = dca
OBJECTS         = $(DCA_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(DCA_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/crypto/io

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#       set signing mode
ELFSIGN_MOD     = $(ELFSIGN_CRYPTO)

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(SRC_CONFFILE)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOTLINK) $(ROOT_CONFFILE)
```

```
# C99 mode is needed for dca
CFLAGS += $(C99_ENABLE)

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-unused-variable

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

$(ROOTLINK):    $(ROOT_CRYPTO_DIR) $(ROOTMODULE)
        -$(RM) $@; ln $(ROOTMODULE) $@

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ
```