```
**********************************************************
    6926 Thu Jan 17 14:59:34 2019
new/usr/src/lib/gss_mechs/mech_dh/dh_common/dh_template.c
10116 mech_dh needs smatch fixes
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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
 */
/*
 *      dh_template.c
 *
 *      Copyright (c) 1997, by Sun Microsystems, Inc.
 *      All rights reserved.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <dh_gssapi.h>
#include <dlfcn.h>
#include "../dh_common/dh_common.h"

extern int key_encryptsession_pk_g();
extern int key_decryptsession_pk_g();
extern int key_gendes_g();
extern int key_secretkey_is_set_g();

static int __encrypt(const char *remotename, des_block deskeys[], int no_keys);
static int __decrypt(const char *remotename,
                     des_block deskeys[], int no_keys, int *key_cached);
static int __gendes(des_block deskeys[], int no_keys);
static int __secret_is_set(void);
static char *__get_principal(void);

/*
 * This module defines the entry point for gss_mech_initialize and the
 * key opts for Diffie-Hellman mechanism of type algorithm 0. Each algorithm
 * 0 mechanism defines its OID, MODULUS, ROOT, KEYLEN, ALGTYPE (which should
 * be zero) and HEX_KEY_BYTES. That module then will #include this file.
 */

/* The keyopts for the per mechanism context */
static dh_keyopts_desc dh_keyopts = {
```



```
        __encrypt,
        __decrypt,
        __gendes,
        __secret_is_set,
        __get_principal
};
______unchanged_portion_omitted_

/*
 * A NIS+ server will define the function __rpcsec_gss_is_server.
 * This function will return one when it is appropriate to get public
 * keys out of the per process public key cache. Appropriateness here
 * is when the name server just put the public key in the cache from a
 * received directory object, typically from the cold start file.
 */
static int
dh_getpublickey(const char *remote, keylen_t keylen, algtype_t algtype,
                char *pk, size_t pklen)
{
        static mutex_t init_nis_pubkey_lock = DEFAULTMUTEX;
        static int init_nis_pubkey = 0;
        static int (*nis_call)();
        static const char NIS_SYMBOL[] = "__rpcsec_gss_is_server";

        if (!init_nis_pubkey) {
                (void) mutex_lock(&init_nis_pubkey_lock);
                mutex_lock(&init_nis_pubkey_lock);
                if (!init_nis_pubkey) {
                        void *dlhandle = dlopen(0, RTLD_NOLOAD);
                        if (dlhandle == 0) {
                                syslog(LOG_ERR, "dh: Could not dlopen "
                                    "in dh_getpublickey for %s. "
                                    "dlopen returned %s", remote, dlerror());
                        } else {
                                nis_call = (int (*)())
                                        dlsym(dlhandle, NIS_SYMBOL);
                        }
                        init_nis_pubkey = 1;
                }
                (void) mutex_unlock(&init_nis_pubkey_lock);
                mutex_unlock(&init_nis_pubkey_lock);
        }
        if (nis_call && (*nis_call)()) {
                int key_cached;
                return (__getpublickey_cached_g(remote, keylen, algtype,
                                                pk, pklen, &key_cached));
        }

        /*
         * If we're not being called by a nis plus server or that
         * server does not want to get the keys from the cache we
         * get the key in the normal manner.
         */

        return (getpublickey_g(remote, keylen, algtype, pk, pklen));
}
______unchanged_portion_omitted_
```