```
**********************************************************
   35970 Thu Aug 14 10:10:50 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_kef.c
6621176 $SRC/cmd/cmd-crypto/cryptoadm/*.c seem to have syntax errors in the tran
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <locale.h>
#include <libgen.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/crypto/ioctladmin.h>
#include <signal.h>
#include <sys/crypto/elfsign.h>
#include "cryptoadm.h"

static int err; /* to store the value of errno in case being overwritten */
static int check_hardware_provider(char *, char *, int *, int *);

/*
 * Display the mechanism list for a kernel software provider.
 */
int
list_mechlist_for_soft(char *provname)
{
        mechlist_t *pmechlist;
        int rc;

        if (provname == NULL) {
                return (FAILURE);
        }

        rc = get_soft_info(provname, &pmechlist);
        if (rc == SUCCESS) {
                (void) filter_mechlist(&pmechlist, RANDOM);
```



```
                print_mechlist(provname, pmechlist);
                free_mechlist(pmechlist);
        } else {
                cryptoerror(LOG_STDERR, gettext(
                    "failed to retrieve the mechanism list for %s."),
                    provname);
        }

        return (rc);

}
```

```
_____unchanged_portion_omitted_
```

```


/*
 * Display the policy information for a kernel hardware provider.
 */
int
list_policy_for_hard(char *provname)
{
        entry_t *pent;
        boolean_t is_active;
        mechlist_t *pmechlist;
        char    devname[MAXNAMELEN];
        int     inst_num;
        int     count;
        int rc = SUCCESS;
        boolean_t has_random = B_FALSE;
        boolean_t has_mechs = B_FALSE;

        if (provname == NULL) {
                return (FAILURE);
        }

        /*
         * Check if the provider is valid. If it is valid, get the number of
         * mechanisms also.
         */
        if (check_hardware_provider(provname, devname, &inst_num, &count) ==
            FAILURE) {
                return (FAILURE);
        }

        /* Get the mechanism list for the kernel hardware provider */
        if ((rc = get_dev_info(devname, inst_num, count, &pmechlist)) ==
            SUCCESS) {
                has_random = filter_mechlist(&pmechlist, RANDOM);

                if (pmechlist != NULL) {
                        has_mechs = B_TRUE;
                        free_mechlist(pmechlist);
                }
        } else {
                cryptoerror(LOG_STDERR, gettext(
                    "failed to retrieve the mechanism list for %s."),
                    devname);
                return (rc);
        }

        /*
         * If the hardware provider has an entry in the kcf.conf file,
         * some of its mechanisms must have been disabled.  Print out
         * the disabled list from the config file entry.  Otherwise,
         * if it is active, then all the mechanisms for it are enabled.
         */
```

```
202         if ((pent = getent_kef(provname)) != NULL) {
203                 print_kef_policy(pent, has_random, has_mechs);
204                 free_entry(pent);
205                 return (SUCCESS);
206         } else {
207                 if (check_active_for_hard(provname, &is_active) ==
208                     FAILURE) {
209                         return (FAILURE);
210                 } else if (is_active == B_TRUE) {
211                         (void) printf(gettext(
212                             "%s: all mechanisms are enabled."), provname);
213                         if (has_random)
214                                 /*
215                                  * TRANSLATION_NOTE
217                                  * TRANSLATION_NOTE:
216                                  * "random" is a keyword and not to be
217                                  * translated.
218                                  */
219                                 (void) printf(gettext(" %s is enabled.\n"),
220                                     "random");
221                         else
222                                 (void) printf("\n");
223                         return (SUCCESS);
224                 } else {
225                         cryptoerror(LOG_STDERR,
226                             gettext("%s does not exist."), provname);
227                         return (FAILURE);
228                 }
229         }
230 }
```

*unchanged_portion_omitted_*

```
357 int
358 disable_kef_software(char *provname, boolean_t rndflag, boolean_t allflag,
359     mechlist_t *dislist)
360 {
361         crypto_load_soft_disabled_t     *pload_soft_dis = NULL;
362         mechlist_t      *infolist;
363         entry_t         *pent;
364         boolean_t       is_active;
365         int     fd;

367         if (provname == NULL) {
368                 return (FAILURE);
369         }

371         /* Get the entry of this provider from the config file. */
372         if ((pent = getent_kef(provname)) == NULL) {
373                 cryptoerror(LOG_STDERR,
374                     gettext("%s does not exist."), provname);
375                 return (FAILURE);
376         }

378         /*
379          * Check if the kernel software provider is currently unloaded.
380          * If it is unloaded, return FAILURE, because the disable subcommand
381          * can not perform on inactive (unloaded) providers.
382          */
383         if (check_active_for_soft(provname, &is_active) == FAILURE) {
384                 free_entry(pent);
385                 return (FAILURE);
386         } else if (is_active == B_FALSE) {
387                 /*
388                  * TRANSLATION_NOTE
```



```
390                  * TRANSLATION_NOTE:
389                  * "disable" is a keyword and not to be translated.
390                  */
391                 cryptoerror(LOG_STDERR,
392                     gettext("can not do %1$s on an unloaded "
393                     "kernel software provider -- %2$s."), "disable", provname);
394                 free_entry(pent);
395                 return (FAILURE);
396         }

398         /* Get the mechanism list for the software provider */
399         if (get_soft_info(provname, &infolist) == FAILURE) {
400                 free(pent);
401                 return (FAILURE);
402         }

404         /* See comments in disable_kef_hardware() */
405         if (!rndflag) {
406                 (void) filter_mechlist(&infolist, RANDOM);
407         }

409         /* Calculate the new disabled list */
410         if (disable_mechs(&pent, infolist, allflag, dislist) == FAILURE) {
411                 free_entry(pent);
412                 free_mechlist(infolist);
413                 return (FAILURE);
414         }

416         /* infolist is no longer needed; free it */
417         free_mechlist(infolist);

419         /* Update the kcf.conf file with the updated entry */
420         if (update_kcfconf(pent, MODIFY_MODE) == FAILURE) {
421                 free_entry(pent);
422                 return (FAILURE);
423         }

425         /* Inform kernel about the new disabled list. */
426         if ((pload_soft_dis = setup_soft_dis(pent)) == NULL) {
427                 free_entry(pent);
428                 return (FAILURE);
429         }

431         /* pent is no longer needed; free it. */
432         free_entry(pent);

434         if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
435                 cryptoerror(LOG_STDERR,
436                     gettext("failed to open %s for RW: %s"),
437                     ADMIN_IOCTL_DEVICE, strerror(errno));
438                 free(pload_soft_dis);
439                 return (FAILURE);
440         }

442         if (ioctl(fd, CRYPTO_LOAD_SOFT_DISABLED, pload_soft_dis) == -1) {
443                 cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl failed: %s",
444                     strerror(errno));
445                 free(pload_soft_dis);
446                 (void) close(fd);
447                 return (FAILURE);
448         }

450         if (pload_soft_dis->sd_return_value != CRYPTO_SUCCESS) {
451                 cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl return_value = "
452                     "%d", pload_soft_dis->sd_return_value);
453                 free(pload_soft_dis);
```

```
 454                 (void) close(fd);
 455                 return (FAILURE);
 456         }

 458         free(pload_soft_dis);
 459         (void) close(fd);
 460         return (SUCCESS);
 461 }
_____unchanged_portion_omitted_

1261 /*
1262  * Unload the kernel software provider. Before calling this function, the
1263  * caller should check if the provider is in the config file and if it
1264  * is kernel. This routine makes 3 ioctl calls to remove it from kernel
1265  * completely. The argument do_check set to B_FALSE means that the
1266  * caller knows the provider is not the config file and hence the check
1267  * is skipped.
1268  */
1269 int
1270 unload_kef_soft(char *provname, boolean_t do_check)
1271 {
1272         crypto_unload_soft_module_t     *punload_soft = NULL;
1273         crypto_load_soft_config_t       *pload_soft_conf = NULL;
1274         crypto_load_soft_disabled_t     *pload_soft_dis = NULL;
1275         entry_t *pent = NULL;
1276         int     fd;

1278         if (provname == NULL) {
1279                 cryptoerror(LOG_STDERR, gettext("internal error."));
1280                 return (FAILURE);
1281         }

1283         if (!do_check) {
1284                 /* Construct an entry using the provname */
1285                 pent = calloc(1, sizeof (entry_t));
1286                 if (pent == NULL) {
1287                         cryptoerror(LOG_STDERR, gettext("out of memory."));
1288                         return (FAILURE);
1289                 }
1290                 (void) strlcpy(pent->name, provname, MAXNAMELEN);
1291         } else if ((pent = getent_kef(provname)) == NULL) {
1292                 cryptoerror(LOG_STDERR, gettext("%s does not exist."),
1293                     provname);
1294                 return (FAILURE);
1295         }

1297         /* Open the admin_ioctl_device */
1298         if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
1299                 err = errno;
1300                 cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
1301                     ADMIN_IOCTL_DEVICE, strerror(err));
1302                 return (FAILURE);
1303         }

1305         /* Inform kernel to unload this software module */
1306         if ((punload_soft = setup_unload_soft(pent)) == NULL) {
1307                 (void) close(fd);
1308                 return (FAILURE);
1309         }

1311         if (ioctl(fd, CRYPTO_UNLOAD_SOFT_MODULE, punload_soft) == -1) {
1312                 cryptodebug("CRYPTO_UNLOAD_SOFT_MODULE ioctl failed: %s",
1313                     strerror(errno));
1314                 free_entry(pent);
1315                 free(punload_soft);
1316                 (void) close(fd);
```



```
1317                 return (FAILURE);
1318         }

1320         if (punload_soft->sm_return_value != CRYPTO_SUCCESS) {
1321                 cryptodebug("CRYPTO_UNLOAD_SOFT_MODULE ioctl return_value = "
1322                     "%d", punload_soft->sm_return_value);
1323                 /*
1324                  * If the return value is CRYPTO_UNKNOWN_PROVIDER, it means
1326                  * If the return value is CRYPTO_UNKNOWN_PRIVDER, it means
1325                  * that the provider is not registered yet.  Should just
1326                  * continue.
1327                  */
1328                 if (punload_soft->sm_return_value != CRYPTO_UNKNOWN_PROVIDER) {
1329                         free_entry(pent);
1330                         free(punload_soft);
1331                         (void) close(fd);
1332                         return (FAILURE);
1333                 }
1334         }

1336         free(punload_soft);

1338         /*
1339          * Inform kernel to remove the configuration of this software
1340          * module.
1341          */
1342         free_mechlist(pent->suplist);
1343         pent->suplist = NULL;
1344         pent->sup_count = 0;
1345         if ((pload_soft_conf = setup_soft_conf(pent)) == NULL) {
1346                 free_entry(pent);
1347                 (void) close(fd);
1348                 return (FAILURE);
1349         }

1351         if (ioctl(fd, CRYPTO_LOAD_SOFT_CONFIG, pload_soft_conf) == -1) {
1352                 cryptodebug("CRYPTO_LOAD_SOFT_CONFIG ioctl failed: %s",
1353                     strerror(errno));
1354                 free_entry(pent);
1355                 free(pload_soft_conf);
1356                 (void) close(fd);
1357                 return (FAILURE);
1358         }

1360         if (pload_soft_conf->sc_return_value != CRYPTO_SUCCESS) {
1361                 cryptodebug("CRYPTO_LOAD_SOFT_CONFIG ioctl return_value = "
1362                     "%d", pload_soft_conf->sc_return_value);
1363                 free_entry(pent);
1364                 free(pload_soft_conf);
1365                 (void) close(fd);
1366                 return (FAILURE);
1367         }

1369         free(pload_soft_conf);

1371         /* Inform kernel to remove the disabled entries if any */
1372         if (pent->dis_count == 0) {
1373                 free_entry(pent);
1374                 (void) close(fd);
1375                 return (SUCCESS);
1376         } else {
1377                 free_mechlist(pent->dislist);
1378                 pent->dislist = NULL;
1379                 pent->dis_count = 0;
1380         }
```

```
1382         if ((pload_soft_dis = setup_soft_dis(pent)) == NULL) {
1383                 free_entry(pent);
1384                 (void) close(fd);
1385                 return (FAILURE);
1386         }

1388         /* pent is no longer needed; free it */
1389         free_entry(pent);

1391         if (ioctl(fd, CRYPTO_LOAD_SOFT_DISABLED, pload_soft_dis) == -1) {
1392                 cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl failed: %S",
1393                     strerror(errno));
1394                 free(pload_soft_dis);
1395                 (void) close(fd);
1396                 return (FAILURE);
1397         }

1399         if (pload_soft_dis->sd_return_value != CRYPTO_SUCCESS) {
1400                 cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl return_value = "
1401                     "%d", pload_soft_dis->sd_return_value);
1402                 free(pload_soft_dis);
1403                 (void) close(fd);
1404                 return (FAILURE);
1405         }

1407         free(pload_soft_dis);
1408         (void) close(fd);
1409         return (SUCCESS);
1410 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   28317 Thu Aug 14 10:10:54 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_kef_util.c
6621176 $SRC/cmd/cmd-crypto/cryptoadm/*.c seem to have syntax errors in the tran
**********************************************************
```

```
/*
 * 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.
 * 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
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <time.h>
#include <unistd.h>
#include <locale.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "cryptoadm.h"

static int err; /* To store errno which may be overwritten by gettext() */
static int build_entrylist(entry_t *, entrylist_t **);
static entry_t *dup_entry(entry_t *);
static mechlist_t *dup_mechlist(mechlist_t *);
static entry_t *getent(char *, entrylist_t *);
static int interpret(char *, entry_t **);
static int parse_dislist(char *, entry_t *);


/*
 * Duplicate the mechanism list.  A null pointer is returned if the storage
 * space available is insufficient or the input argument is NULL.
 */
static mechlist_t *
dup_mechlist(mechlist_t *plist)
{
        mechlist_t *pres = NULL;
        mechlist_t *pcur;
```



```
        mechlist_t *ptmp;
        int rc = SUCCESS;

        while (plist != NULL) {
                if (!(ptmp = create_mech(plist->name))) {
                        rc = FAILURE;
                        break;
                }

                if (pres == NULL) {
                        pres = pcur = ptmp;
                } else {
                        pcur->next = ptmp;
                        pcur = pcur->next;
                }
                plist = plist->next;
        }

        if (rc != SUCCESS) {
                free_mechlist(pres);
                return (NULL);
        }

        return (pres);
}
```

*unchanged_portion_omitted_*

```
/*
 * Print out the mechanism policy for a kernel provider that has an entry
 * in the kcf.conf file.
 *
 * The flag has_random is set to B_TRUE if the provider does random
 * numbers. The flag has_mechs is set by the caller to B_TRUE if the provider
 * has some mechanisms.
 */
void
print_kef_policy(entry_t *pent, boolean_t has_random, boolean_t has_mechs)
{
        mechlist_t *ptr;
        boolean_t rnd_disabled = B_FALSE;

        if (pent == NULL) {
                return;
        }

        rnd_disabled = filter_mechlist(&pent->dislist, RANDOM);
        ptr = pent->dislist;

        (void) printf("%s:", pent->name);

        if (has_mechs == B_TRUE) {
                /*
                 * TRANSLATION_NOTE
                 * TRANSLATION_NOTE:
                 * This code block may need to be modified a bit to avoid
                 * constructing the text message on the fly.
                 */
                (void) printf(gettext(" all mechanisms are enabled"));
                if (ptr != NULL)
                        (void) printf(gettext(", except "));
                while (ptr != NULL) {
                        (void) printf("%s", ptr->name);
                        ptr = ptr->next;
                        if (ptr != NULL)
```

```
1204                                    (void) printf(",");
1205                 }
1206                 if (ptr == NULL)
1207                         (void) printf(".");
1208         }

1210         /*
1211          * TRANSLATION_NOTE
1214          * TRANSLATION_NOTE:
1212          * "random" is a keyword and not to be translated.
1213          */
1214         if (rnd_disabled)
1215                 (void) printf(gettext(" %s is disabled."), "random");
1216         else if (has_random)
1217                 (void) printf(gettext(" %s is enabled."), "random");
1218         (void) printf("\n");
1219 }
```

unchanged_portion_omitted_

```
**********************************************************
   13950 Thu Aug 14 10:10:58 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_metaslot.c
6621176 $SRC/cmd/cmd-crypto/cryptoadm/*.c seem to have syntax errors in the tran
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Administration for metaslot
 *
 * All the "list" operations will call functions in libpkcs11.so
 * Normally, it doesn't make sense to call functions in libpkcs11.so directly
 * because libpkcs11.so depends on the configuration file (pkcs11.conf) the
 * cryptoadm command is trying to administer.  However, since metaslot
 * is part of the framework, it is not possible to get information about
 * it without actually calling functions in libpkcs11.so.
 *
 * So, for the listing operation, which won't modify the value of pkcs11.conf
 * it is safe to call libpkcs11.so.
 *
 * For other operations that modifies the pkcs11.conf file, libpkcs11.so
 * will not be called.
 *
 */

#include <cryptoutil.h>
#include <stdio.h>
#include <libintl.h>
#include <dlfcn.h>
#include <link.h>
#include <strings.h>
#include <security/cryptoki.h>
#include <cryptoutil.h>
#include "cryptoadm.h"

#define METASLOT_ID     0

int
list_metaslot_info(boolean_t show_mechs, boolean_t verbose,
    mechlist_t *mechlist)
{
        int rc = SUCCESS;
        CK_RV rv;
```



```
        CK_SLOT_INFO slot_info;
        CK_TOKEN_INFO token_info;
        CK_MECHANISM_TYPE_PTR pmech_list = NULL;
        CK_ULONG mech_count;
        int i;
        CK_RV (*Tmp_C_GetFunctionList)(CK_FUNCTION_LIST_PTR_PTR);
        CK_FUNCTION_LIST_PTR    funcs;
        void *dldesc = NULL;
        boolean_t lib_initialized = B_FALSE;
        uentry_t *puent;
        char buf[128];


        /*
         * Display the system-wide metaslot settings as specified
         * in pkcs11.conf file.
         */
        if ((puent = getent_uef(METASLOT_KEYWORD)) == NULL) {
                cryptoerror(LOG_STDERR,
                    gettext("metaslot entry doesn't exist."));
                return (FAILURE);
        }

        (void) printf(gettext("System-wide Meta Slot Configuration:\n"));
        /*
         * TRANSLATION_NOTE
         * TRANSLATION_NOTE:
         * Strictly for appearance's sake, this line should be as long as
         * the length of the translated text above.
         */
        (void) printf(gettext("-----------------------------------\n"));
        (void) printf(gettext("Status: %s\n"), puent->flag_metaslot_enabled ?
            gettext("enabled") : gettext("disabled"));
        (void) printf(gettext("Sensitive Token Object Automatic Migrate: %s\n"),
            puent->flag_metaslot_auto_key_migrate ? gettext("enabled") :
            gettext("disabled"));

        bzero(buf, sizeof (buf));
        if (memcmp(puent->metaslot_ks_slot, buf, SLOT_DESCRIPTION_SIZE) != 0) {
                (void) printf(gettext("Persistent object store slot: %s\n"),
                    puent->metaslot_ks_slot);
        }

        if (memcmp(puent->metaslot_ks_token, buf, TOKEN_LABEL_SIZE) != 0) {
                (void) printf(gettext("Persistent object store token: %s\n"),
                    puent->metaslot_ks_token);
        }

        if ((!verbose) && (!show_mechs)) {
                return (SUCCESS);
        }

        if (verbose) {
                (void) printf(gettext("\nDetailed Meta Slot Information:\n"));
                /*
                 * TRANSLATION_NOTE
                 * TRANSLATION_NOTE:
                 * Strictly for appearance's sake, this line should be as
                 * long as the length of the translated text above.
                 */
                (void) printf(gettext("-------------------------------\n"));
        }

        /*
         * Need to actually make calls to libpkcs11.so to get
         * information about metaslot.
```

```
126          */

128         dldesc = dlopen(UEF_FRAME_LIB, RTLD_NOW);
129         if (dldesc == NULL) {
130                 char *dl_error;
131                 dl_error = dlerror();
132                 cryptodebug("Cannot load PKCS#11 framework library. "
133                     "dlerror:%s", dl_error);
134                 return (FAILURE);
135         }

137         /* Get the pointer to library's C_GetFunctionList() */
138         Tmp_C_GetFunctionList = (CK_RV(*)())dlsym(dldesc, "C_GetFunctionList");
139         if (Tmp_C_GetFunctionList == NULL) {
140                 cryptodebug("Cannot get the address of the C_GetFunctionList "
141                     "from framework");
142                 rc = FAILURE;
143                 goto finish;
144         }


147         /* Get the provider's function list */
148         rv = Tmp_C_GetFunctionList(&funcs);
149         if (rv != CKR_OK) {
150                 cryptodebug("failed to call C_GetFunctionList in "
151                     "framework library");
152                 rc = FAILURE;
153                 goto finish;
154         }

156         /* Initialize this provider */
157         rv = funcs->C_Initialize(NULL_PTR);
158         if (rv != CKR_OK) {
159                 cryptodebug("C_Initialize failed with error code 0x%x\n", rv);
160                 rc = FAILURE;
161                 goto finish;
162         } else {
163                 lib_initialized = B_TRUE;
164         }

166         /*
167          * We know for sure that metaslot is slot 0 in the framework,
168          * so, we will do a C_GetSlotInfo() trying to see if it works.
169          * If it fails with CKR_SLOT_ID_INVALID, we know that metaslot
170          * is not really enabled.
171          */
172         rv = funcs->C_GetSlotInfo(METASLOT_ID, &slot_info);
173         if (rv == CKR_SLOT_ID_INVALID) {
174                 (void) printf(gettext("actual status: disabled.\n"));
175                 /*
176                  * Even if the -m and -v flag is supplied, there's nothing
177                  * interesting to display about metaslot since it is disabled,
178                  * so, just stop right here.
179                  */
180                 goto finish;
181         }

183         if (rv != CKR_OK) {
184                 cryptodebug("C_GetSlotInfo failed with error "
185                     "code 0x%x\n", rv);
186                 rc = FAILURE;
187                 goto finish;
188         }

190         if (!verbose) {
191                 goto display_mechs;
```



```
192         }

194         (void) printf(gettext("actual status: enabled.\n"));

196         (void) printf(gettext("Description: %.64s\n"),
197             slot_info.slotDescription);

199         (void) printf(gettext("Token Present: %s\n"),
200             (slot_info.flags & CKF_TOKEN_PRESENT ?
201             gettext("True") : gettext("False")));

203         rv = funcs->C_GetTokenInfo(METASLOT_ID, &token_info);
204         if (rv != CKR_OK) {
205                 cryptodebug("C_GetTokenInfo failed with error "
206                     "code 0x%x\n", rv);
207                 rc = FAILURE;
208                 goto finish;
209         }

211         (void) printf(gettext("Token Label: %.32s\n"
212             "Manufacturer ID: %.32s\n"
213             "Model: %.16s\n"
214             "Serial Number: %.16s\n"
215             "Hardware Version: %d.%d\n"
216             "Firmware Version: %d.%d\n"
217             "UTC Time: %.16s\n"
218             "PIN Min Length: %d\n"
219             "PIN Max Length: %d\n"),
220             token_info.label,
221             token_info.manufacturerID,
222             token_info.model,
223             token_info.serialNumber,
224             token_info.hardwareVersion.major,
225             token_info.hardwareVersion.minor,
226             token_info.firmwareVersion.major,
227             token_info.firmwareVersion.minor,
228             token_info.utcTime,
229             token_info.ulMinPinLen,
230             token_info.ulMaxPinLen);

232         display_token_flags(token_info.flags);

234         if (!show_mechs) {
235                 goto finish;
236         }

238 display_mechs:

240         if (mechlist == NULL) {
241                 rv = funcs->C_GetMechanismList(METASLOT_ID, NULL_PTR,
242                     &mech_count);
243                 if (rv != CKR_OK) {
244                         cryptodebug("C_GetMechanismList failed with error "
245                             "code 0x%x\n", rv);
246                         rc = FAILURE;
247                         goto finish;
248                 }

250                 if (mech_count > 0) {
251                         pmech_list = malloc(mech_count *
252                             sizeof (CK_MECHANISM_TYPE));
253                         if (pmech_list == NULL) {
254                                 cryptodebug("out of memory");
255                                 rc = FAILURE;
256                                 goto finish;
257                         }
```

```
258                         rv = funcs->C_GetMechanismList(METASLOT_ID, pmech_list,
259                             &mech_count);
260                         if (rv != CKR_OK) {
261                                 cryptodebug("C_GetMechanismList failed with "
262                                     "error code 0x%x\n", rv);
263                                 rc = FAILURE;
264                                 goto finish;
265                         }
266                 }
267         } else {
268                 rc = convert_mechlist(&pmech_list, &mech_count, mechlist);
269                 if (rc != SUCCESS) {
270                         goto finish;
271                 }
272         }

274         (void) printf(gettext("Mechanisms:\n"));
275         if (mech_count == 0) {
276                 /* should never be this case */
277                 (void) printf(gettext("No mechanisms\n"));
278                 goto finish;
279         }
280         if (verbose) {
281                 display_verbose_mech_header();
282         }

284         for (i = 0; i < mech_count; i++) {
285                 CK_MECHANISM_TYPE       mech = pmech_list[i];

287                 if (mech >= CKM_VENDOR_DEFINED) {
288                         (void) printf("%#lx", mech);
289                 } else {
290                         (void) printf("%-29s", pkcs11_mech2str(mech));
291                 }

293                 if (verbose) {
294                         CK_MECHANISM_INFO mech_info;
295                         rv = funcs->C_GetMechanismInfo(METASLOT_ID,
296                             mech, &mech_info);
297                         if (rv != CKR_OK) {
298                                 cryptodebug("C_GetMechanismInfo failed with "
299                                     "error code 0x%x\n", rv);
300                                 rc = FAILURE;
301                                 goto finish;
302                         }
303                         display_mech_info(&mech_info);
304                 }
305                 (void) printf("\n");
306         }

308 finish:

310         if ((rc == FAILURE) && (show_mechs)) {
311                 (void) printf(gettext(
312                     "metaslot: failed to retrieve the mechanism list.\n"));
313         }

315         if (lib_initialized) {
316                 (void) funcs->C_Finalize(NULL_PTR);
317         }

319         if (dldesc != NULL) {
320                 (void) dlclose(dldesc);
321         }

323         if (pmech_list != NULL) {
```



```
324                 (void) free(pmech_list);
325         }

327         return (rc);
328 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   45470 Thu Aug 14 10:11:02 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_uef.c
6621176 $SRC/cmd/cmd-crypto/cryptoadm/*.c seem to have syntax errors in the tran
**********************************************************
______unchanged_portion_omitted_

 178 /*
 179  * Display the mechanism list for a user-level library
 180  */
 181 int
 182 list_mechlist_for_lib(char *libname, mechlist_t *mlist,
 183                 flag_val_t *rng_flag, boolean_t no_warn,
 184                 boolean_t verbose, boolean_t show_mechs)
 185 {
 186         CK_RV   rv = CKR_OK;
 187         CK_RV   (*Tmp_C_GetFunctionList)(CK_FUNCTION_LIST_PTR_PTR);
 188         CK_FUNCTION_LIST_PTR    prov_funcs; /* Provider's function list */
 189         CK_SLOT_ID_PTR          prov_slots = NULL; /* Provider's slot list */
 190         CK_MECHANISM_TYPE_PTR   pmech_list; /* mechanism list for a slot */
 191         CK_SLOT_INFO    slotinfo;
 192         CK_ULONG        slot_count;
 193         CK_ULONG        mech_count;
 194         uentry_t        *puent = NULL;
 195         boolean_t       lib_initialized = B_FALSE;
 196         void            *dldesc = NULL;
 197         char            *dl_error;
 198         const char      *mech_name;
 199         char            *isa;
 200         char            libpath[MAXPATHLEN];
 201         char            buf[MAXPATHLEN];
 202         int             i, j;
 203         int             rc = SUCCESS;

 205         if (libname == NULL) {
 206                 /* should not happen */
 207                 cryptoerror(LOG_STDERR, gettext("internal error."));
 208                 cryptodebug("list_mechlist_for_lib() - libname is NULL.");
 209                 return (FAILURE);
 210         }

 212         /* Check if the library is in the pkcs11.conf file */
 213         if ((puent = getent_uef(libname)) == NULL) {
 214                 cryptoerror(LOG_STDERR,
 215                     gettext("%s does not exist."), libname);
 216                 return (FAILURE);
 217         }
 218         free_uentry(puent);

 220         /* Remove $ISA from the library name */
 221         if (strlcpy(buf, libname, sizeof (buf)) >= sizeof (buf)) {
 222                 (void) printf(gettext("%s: the provider name is too long."),
 223                     libname);
 224                 return (FAILURE);
 225         }

 227         if ((isa = strstr(buf, PKCS11_ISA)) != NULL) {
 228                 *isa = '\000';
 229                 isa += strlen(PKCS11_ISA);
 230                 (void) snprintf(libpath, MAXPATHLEN, "%s%s%s", buf, "/", isa);
 231         } else {
 232                 (void) strlcpy(libpath, libname, sizeof (libpath));
 233         }

 235         /* Open the provider */
 236         dldesc = dlopen(libpath, RTLD_NOW);
 237         if (dldesc == NULL) {
 238                 dl_error = dlerror();
 239                 cryptodebug("Cannot load PKCS#11 library %s.  dlerror: %s",
 240                     libname, dl_error != NULL ? dl_error : "Unknown");
 241                 rc = FAILURE;
```



```
 242                 goto clean_exit;
 243         }

 245         /* Get the pointer to provider's C_GetFunctionList() */
 246         Tmp_C_GetFunctionList = (CK_RV(*)())dlsym(dldesc, "C_GetFunctionList");
 247         if (Tmp_C_GetFunctionList == NULL) {
 248                 cryptodebug("Cannot get the address of the C_GetFunctionList "
 249                     "from %s", libname);
 250                 rc = FAILURE;
 251                 goto clean_exit;
 252         }

 254         /* Get the provider's function list */
 255         rv = Tmp_C_GetFunctionList(&prov_funcs);
 256         if (rv != CKR_OK) {
 257                 cryptodebug("failed to call C_GetFunctionList from %s",
 258                     libname);
 259                 rc = FAILURE;
 260                 goto clean_exit;
 261         }

 263         /* Initialize this provider */
 264         rv = prov_funcs->C_Initialize(NULL_PTR);
 265         if (rv != CKR_OK) {
 266                 cryptodebug("failed to call C_Initialize from %s, "
 267                     "return code = %d", libname, rv);
 268                 rc = FAILURE;
 269                 goto clean_exit;
 270         } else {
 271                 lib_initialized = B_TRUE;
 272         }

 274         /*
 275          * Find out how many slots this provider has, call with tokenPresent
 276          * set to FALSE so all potential slots are returned.
 277          */
 278         rv = prov_funcs->C_GetSlotList(FALSE, NULL_PTR, &slot_count);
 279         if (rv != CKR_OK) {
 280                 cryptodebug("failed to get the slotlist from %s.", libname);
 281                 rc = FAILURE;
 282                 goto clean_exit;
 283         } else if (slot_count == 0) {
 284                 if (!no_warn)
 285                         (void) printf(gettext("%s: no slots presented.\n"),
 286                             libname);
 287                 rc = SUCCESS;
 288                 goto clean_exit;
 289         }

 291         /* Allocate memory for the slot list */
 292         prov_slots = malloc(slot_count * sizeof (CK_SLOT_ID));
 293         if (prov_slots == NULL) {
 294                 cryptodebug("out of memory.");
 295                 rc = FAILURE;
 296                 goto clean_exit;
 297         }

 299         /* Get the slot list from provider */
 300         rv = prov_funcs->C_GetSlotList(FALSE, prov_slots, &slot_count);
 301         if (rv != CKR_OK) {
 302                 cryptodebug("failed to call C_GetSlotList() from %s.",
 303                     libname);
 304                 rc = FAILURE;
 305                 goto clean_exit;
 306         }
```

```
308         if (verbose) {
309                 (void) printf(gettext("Number of slots: %d\n"), slot_count);
310         }

312         /* Get the mechanism list for each slot */
313         for (i = 0; i < slot_count; i++) {
314                 if (verbose)
315                         /*
316                          * TRANSLATION_NOTE
316                          * TRANSLATION_NOTE:
317                          * In some languages, the # symbol should be
318                          * converted to "no", an "n" followed by a
319                          * superscript "o"..
320                          */
321                         (void) printf(gettext("\nSlot #%d\n"), i+1);

323                 if ((rng_flag != NULL) && (*rng_flag == NO_RNG)) {
324                         if (check_random(prov_slots[i], prov_funcs)) {
325                                 *rng_flag = HAS_RNG;
326                                 rc = SUCCESS;
327                                 goto clean_exit;
328                         } else
329                                 continue;
330                 }

332                 rv = prov_funcs->C_GetSlotInfo(prov_slots[i], &slotinfo);
333                 if (rv != CKR_OK) {
334                         cryptodebug("failed to get slotinfo from %s", libname);
335                         rc = FAILURE;
336                         break;
337                 }
338                 if (verbose) {
339                         CK_TOKEN_INFO tokeninfo;

341                         (void) printf(gettext("Description: %.64s\n"
342                             "Manufacturer: %.32s\n"
343                             "PKCS#11 Version: %d.%d\n"),
344                             slotinfo.slotDescription,
345                             slotinfo.manufacturerID,
346                             prov_funcs->version.major,
347                             prov_funcs->version.minor);

349                         (void) printf(gettext("Hardware Version: %d.%d\n"
350                             "Firmware Version: %d.%d\n"),
351                             slotinfo.hardwareVersion.major,
352                             slotinfo.hardwareVersion.minor,
353                             slotinfo.firmwareVersion.major,
354                             slotinfo.firmwareVersion.minor);

356                         (void) printf(gettext("Token Present: %s\n"),
357                             (slotinfo.flags & CKF_TOKEN_PRESENT ?
358                             gettext("True") : gettext("False")));

360                         display_slot_flags(slotinfo.flags);

362                         rv = prov_funcs->C_GetTokenInfo(prov_slots[i],
363                             &tokeninfo);
364                         if (rv != CKR_OK) {
365                                 cryptodebug("Failed to get "
366                                     "token info from %s", libname);
367                                 rc = FAILURE;
368                                 break;
369                         }

371                         (void) printf(gettext("Token Label: %.32s\n"
372                             "Manufacturer ID: %.32s\n"
```



```
373                             "Model: %.16s\n"
374                             "Serial Number: %.16s\n"
375                             "Hardware Version: %d.%d\n"
376                             "Firmware Version: %d.%d\n"
377                             "UTC Time: %.16s\n"
378                             "PIN Min Length: %d\n"
379                             "PIN Max Length: %d\n"),
380                             tokeninfo.label,
381                             tokeninfo.manufacturerID,
382                             tokeninfo.model,
383                             tokeninfo.serialNumber,
384                             tokeninfo.hardwareVersion.major,
385                             tokeninfo.hardwareVersion.minor,
386                             tokeninfo.firmwareVersion.major,
387                             tokeninfo.firmwareVersion.minor,
388                             tokeninfo.utcTime,
389                             tokeninfo.ulMinPinLen,
390                             tokeninfo.ulMaxPinLen);

392                         display_token_flags(tokeninfo.flags);
393                 }

395                 if (mlist == NULL) {
396                         rv = prov_funcs->C_GetMechanismList(prov_slots[i],
397                             NULL_PTR, &mech_count);
398                         if (rv != CKR_OK) {
399                                 cryptodebug(
400                                     "failed to call C_GetMechanismList() "
401                                     "from %s.", libname);
402                                 rc = FAILURE;
403                                 break;
404                         }

406                         if (mech_count == 0) {
407                                 /* no mechanisms in this slot */
408                                 continue;
409                         }

411                         pmech_list = malloc(mech_count *
412                             sizeof (CK_MECHANISM_TYPE));
413                         if (pmech_list == NULL) {
414                                 cryptodebug("out of memory");
415                                 rc = FAILURE;
416                                 break;
417                         }

419                         /* Get the actual mechanism list */
420                         rv = prov_funcs->C_GetMechanismList(prov_slots[i],
421                             pmech_list, &mech_count);
422                         if (rv != CKR_OK) {
423                                 cryptodebug(
424                                     "failed to call C_GetMechanismList() "
425                                     "from %s.", libname);
426                                 (void) free(pmech_list);
427                                 rc = FAILURE;
428                                 break;
429                         }
430                 } else  {
431                         /* use the mechanism list passed in */
432                         rc = convert_mechlist(&pmech_list, &mech_count, mlist);
433                         if (rc != SUCCESS) {
434                                 goto clean_exit;
435                         }
436                 }
437                 if (show_mechs)
438                         (void) printf(gettext("Mechanisms:\n"));
```

```
440                 if (verbose && show_mechs) {
441                         display_verbose_mech_header();
442                 }
443                 /*
444                  * Merge the current mechanism list into the returning
445                  * mechanism list.
446                  */
447                 for (j = 0; show_mechs && j < mech_count; j++) {
448                         CK_MECHANISM_TYPE       mech = pmech_list[j];

450                         if (mech >= CKM_VENDOR_DEFINED) {
451                                 (void) printf("%#lx", mech);
452                         } else {
453                                 mech_name = pkcs11_mech2str(mech);
454                                 (void) printf("%-29s", mech_name);
455                         }

457                         if (verbose) {
458                                 CK_MECHANISM_INFO mech_info;
459                                 rv = prov_funcs->C_GetMechanismInfo(
460                                     prov_slots[i], mech, &mech_info);
461                                 if (rv != CKR_OK) {
462                                         cryptodebug(
463                                             "failed to call "
464                                             "C_GetMechanismInfo() from %s.",
465                                             libname);
466                                         (void) free(pmech_list);
467                                         rc = FAILURE;
468                                         break;
469                                 }
470                                 display_mech_info(&mech_info);
471                         }
472                         (void) printf("\n");
473                 }
474                 (void) free(pmech_list);
475                 if (rc == FAILURE) {
476                         break;
477                 }
478         }

480         if (rng_flag != NULL || rc == FAILURE) {
481                 goto clean_exit;
482         }

484 clean_exit:

486         if (rc == FAILURE) {
487                 (void) printf(gettext(
488                     "%s: failed to retrieve the mechanism list.\n"), libname);
489         }

491         if (lib_initialized) {
492                 (void) prov_funcs->C_Finalize(NULL_PTR);
493         }

495         if (dldesc != NULL) {
496                 (void) dlclose(dldesc);
497         }

499         if (prov_slots != NULL) {
500                 (void) free(prov_slots);
501         }

503         return (rc);
504 }
```

unchanged_portion_omitted_


```
1199 /*
1200  * Print out the mechanism policy for a user-level provider pointed by puent.
1201  */
1202 int
1203 print_uef_policy(uentry_t *puent)
1204 {
1205         flag_val_t rng_flag;

1207         if (puent == NULL) {
1208                 return (FAILURE);
1209         }

1211         rng_flag = NO_RNG;
1212         if (list_mechlist_for_lib(puent->name, NULL, &rng_flag, B_TRUE,
1213             B_FALSE, B_FALSE) != SUCCESS) {
1214                 cryptoerror(LOG_STDERR,
1215                     gettext("%s internal error."), puent->name);
1216                 return (FAILURE);
1217         }

1219         if (display_policy(puent) != SUCCESS) {
1220                 goto failed_exit;
1221         }


1224         if (puent->flag_norandom == B_TRUE)
1225                 /*
1226                  * TRANSLATION_NOTE
1226                  * TRANSLATION_NOTE:
1227                  * "random" is a keyword and not to be translated.
1228                  */
1229                 (void) printf(gettext(" %s is disabled."), "random");
1230         else {
1231                 if (rng_flag == HAS_RNG)
1232                         /*
1233                          * TRANSLATION_NOTE
1233                          * TRANSLATION_NOTE:
1234                          * "random" is a keyword and not to be translated.
1235                          */
1236                         (void) printf(gettext(" %s is enabled."), "random");
1237         }
1238         (void) printf("\n");

1240         return (SUCCESS);

1242 failed_exit:

1244         (void) printf(gettext("\nout of memory.\n"));
1245         return (FAILURE);
1246 }
```

unchanged_portion_omitted_

```
1802 void
1803 display_verbose_mech_header()
1804 {
1805         (void) printf("%28s %s", " ", HDR1);
1806         (void) printf("%28s %s", " ", HDR2);
1807         (void) printf("%28s %s", " ", HDR3);
1808         (void) printf("%28s %s", " ", HDR4);
1809         (void) printf("%28s %s", " ", HDR5);
1810         (void) printf("%28s %s", " ", HDR6);
1811         (void) printf("%-28.28s %s", gettext("mechanism name"), HDR7);
1812         /*
```

```
1813          * TRANSLATION_NOTE
1813          * TRANSLATION_NOTE:
1814          * Strictly for appearance's sake, the first header line should be
1815          * as long as the length of the translated text above.  The format
1816          * lengths should all match too.
1817          */
1818         (void) printf("%28s ---- ---- "
1819             "-  -  -  -  -  -  -  -  -  -  -  -  -\n",
1820             gettext("----------------------------"));
1821 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   40432 Thu Aug 14 10:11:06 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/cryptoadm.c
6621176 $SRC/cmd/cmd-crypto/cryptoadm/*.c seem to have syntax errors in the tran
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <locale.h>
#include <libgen.h>
#include <sys/types.h>
#include <zone.h>
#include <sys/crypto/ioctladmin.h>
#include <cryptoutil.h>
#include "cryptoadm.h"

#define REQ_ARG_CNT     2

/* subcommand index */
enum subcommand_index {
        CRYPTO_LIST,
        CRYPTO_DISABLE,
        CRYPTO_ENABLE,
        CRYPTO_INSTALL,
        CRYPTO_UNINSTALL,
        CRYPTO_UNLOAD,
        CRYPTO_REFRESH,
        CRYPTO_START,
        CRYPTO_STOP,
        CRYPTO_HELP };

/*
 * TRANSLATION_NOTE
 * TRANSLATION_NOTE:
 * Command keywords are not to be translated.
 */
```



```
static char *cmd_table[] = {
        "list",
        "disable",
        "enable",
        "install",
        "uninstall",
        "unload",
        "refresh",
        "start",
        "stop",
        "--help" };

/* provider type */
enum provider_type_index {
        PROV_UEF_LIB,
        PROV_KEF_SOFT,
        PROV_KEF_HARD,
        METASLOT,
        PROV_BADNAME };

typedef struct {
        char cp_name[MAXPATHLEN];
        enum provider_type_index cp_type;
} cryptoadm_provider_t;

/*
 * TRANSLATION_NOTE
 * TRANSLATION_NOTE:
 * Operand keywords are not to be translated.
 */
static const char *KN_PROVIDER = "provider=";
static const char *KN_MECH = "mechanism=";
static const char *KN_ALL = "all";
static const char *KN_TOKEN = "token=";
static const char *KN_SLOT = "slot=";
static const char *KN_DEFAULT_KS = "default-keystore";
static const char *KN_AUTO_KEY_MIGRATE = "auto-key-migrate";

/* static variables */
static boolean_t        allflag = B_FALSE;
static boolean_t        rndflag = B_FALSE;
static mechlist_t       *mecharglist = NULL;

/* static functions */
static void usage(void);
static int get_provider_type(char *);
static int process_mech_operands(int, char **, boolean_t);
static int do_list(int, char **);
static int do_disable(int, char **);
static int do_enable(int, char **);
static int do_install(int, char **);
static int do_uninstall(int, char **);
static int do_unload(int, char **);
static int do_refresh(int);
static int do_start(int);
static int do_stop(int);
static int list_simple_for_all(boolean_t);
static int list_mechlist_for_all(boolean_t);
static int list_policy_for_all(void);

int
main(int argc, char *argv[])
{
        char    *subcmd;
        int     cmdnum;
        int     cmd_index = 0;
```

```
124         int     rc = SUCCESS;

126         (void) setlocale(LC_ALL, "");

128 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
129 #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
130 #endif
131         (void) textdomain(TEXT_DOMAIN);

133         cryptodebug_init(basename(argv[0]));

135         if (argc < REQ_ARG_CNT) {
136                 usage();
137                 return (ERROR_USAGE);
138         }

140         /* get the subcommand index */
141         cmd_index = 0;
142         subcmd = argv[1];
143         cmdnum = sizeof (cmd_table)/sizeof (cmd_table[0]);

145         while ((cmd_index < cmdnum) &&
146             (strcmp(subcmd, cmd_table[cmd_index]) != 0)) {
147                 cmd_index++;
148         }
149         if (cmd_index >= cmdnum) {
150                 usage();
151                 return (ERROR_USAGE);
152         }

154         /* do the subcommand */
155         switch (cmd_index) {
156         case CRYPTO_LIST:
157                 rc = do_list(argc, argv);
158                 break;
159         case CRYPTO_DISABLE:
160                 rc = do_disable(argc, argv);
161                 break;
162         case CRYPTO_ENABLE:
163                 rc = do_enable(argc, argv);
164                 break;
165         case CRYPTO_INSTALL:
166                 rc = do_install(argc, argv);
167                 break;
168         case CRYPTO_UNINSTALL:
169                 rc = do_uninstall(argc, argv);
170                 break;
171         case CRYPTO_UNLOAD:
172                 rc = do_unload(argc, argv);
173                 break;
174         case CRYPTO_REFRESH:
175                 rc = do_refresh(argc);
176                 break;
177         case CRYPTO_START:
178                 rc = do_start(argc);
179                 break;
180         case CRYPTO_STOP:
181                 rc = do_stop(argc);
182                 break;
183         case CRYPTO_HELP:
184                 usage();
185                 rc = SUCCESS;
186                 break;
187         default: /* should not come here */
188                 usage();
189                 rc = ERROR_USAGE;
```



```
190                 break;
191         }
192         return (rc);
193 }


196 static void
197 usage(void)
198 {
199         /*
200          * TRANSLATION_NOTE
201          * TRANSLATION_NOTE:
201          * Command usage is not to be translated.  Only the word "Usage:"
202          * along with localized expressions indicating what kind of value
203          * is expected for arguments.
204          */
205         (void) fprintf(stderr, gettext("Usage:\n"));
206         (void) fprintf(stderr,
207             "  cryptoadm list [-mpv] [provider=<%s> | metaslot]"
208             " [mechanism=<%s>]\n",
209             gettext("provider-name"), gettext("mechanism-list"));
210         (void) fprintf(stderr,
211             "  cryptoadm disable provider=<%s>"
212             " mechanism=<%s> | random | all\n",
213             gettext("provider-name"), gettext("mechanism-list"));
214         (void) fprintf(stderr,
215             "  cryptoadm disable metaslot"
216             " [auto-key-migrate] [mechanism=<%s>]\n",
217             gettext("mechanism-list"));
218         (void) fprintf(stderr,
219             "  cryptoadm enable provider=<%s>"
220             " mechanism=<%s> | random | all\n",
221             gettext("provider-name"), gettext("mechanism-list"));
222         (void) fprintf(stderr,
223             "  cryptoadm enable metaslot [mechanism=<%s>]"
224             " [[token=<%s>] [slot=<%s>]"
225             " | [default-keystore]] | [auto-key-migrate]\n",
226             gettext("mechanism-list"), gettext("token-label"),
227             gettext("slot-description"));
228         (void) fprintf(stderr,
229             "  cryptoadm install provider=<%s>\n",
230             gettext("provider-name"));
231         (void) fprintf(stderr,
232             "  cryptoadm install provider=<%s> [mechanism=<%s>]\n",
233             gettext("provider-name"), gettext("mechanism-list"));
234         (void) fprintf(stderr,
235             "  cryptoadm uninstall provider=<%s>\n",
236             gettext("provider-name"));
237         (void) fprintf(stderr,
238             "  cryptoadm unload provider=<%s>\n",
239             gettext("provider-name"));
240         (void) fprintf(stderr,
241             "  cryptoadm refresh\n"
242             "  cryptoadm start\n"
243             "  cryptoadm stop\n"
244             "  cryptoadm --help\n");
245 }


248 /*
249  * Get the provider type.  This function returns
250  * - PROV_UEF_LIB if provname contains an absolute path name
251  * - PROV_KEF_SOFT if provname is a base name only
252  * - PROV_KEF_HARD if provname contains one slash only and the slash is not
253  *      the 1st character.
254  * - PROV_BADNAME otherwise.
```

```
 * - PROV_BADNAME othewise.
 */
static int
get_provider_type(char *provname)
{
        char *pslash1;
        char *pslash2;

        if (provname == NULL) {
                return (FAILURE);
        }

        if (provname[0] == '/') {
                return (PROV_UEF_LIB);
        } else if ((pslash1 = strchr(provname, SEP_SLASH)) == NULL) {
                /* no slash */
                return (PROV_KEF_SOFT);
        } else {
                pslash2 = strrchr(provname, SEP_SLASH);
                if (pslash1 == pslash2) {
                        return (PROV_KEF_HARD);
                } else {
                        return (PROV_BADNAME);
                }
        }
}

/*
 * Get the provider structure.  This function returns NULL if no valid
 * provider= is found in argv[], otherwise a cryptoadm_provider_t is returned.
 * If provider= is found but has no argument, then a cryptoadm_provider_t
 * with cp_type = PROV_BADNAME is returned.
 */
static cryptoadm_provider_t *
get_provider(int argc, char **argv)
{
        int c = 0;
        boolean_t found = B_FALSE;
        cryptoadm_provider_t *provider = NULL;
        char *provstr = NULL, *savstr;
        boolean_t is_metaslot = B_FALSE;

        while (!found && ++c < argc) {
                if (strncmp(argv[c], METASLOT_KEYWORD,
                    strlen(METASLOT_KEYWORD)) == 0) {
                        is_metaslot = B_TRUE;
                        found = B_TRUE;
                } else if (strncmp(argv[c], KN_PROVIDER,
                    strlen(KN_PROVIDER)) == 0 &&
                    strlen(argv[c]) > strlen(KN_PROVIDER)) {
                        if ((provstr = strdup(argv[c])) == NULL) {
                                int err = errno;
                                /*
                                 * TRANSLATION_NOTE
                                 * TRANSLATION_NOTE:
                                 * "get_provider" is a function name and should
                                 * not be translated.
                                 */
                                cryptoerror(LOG_STDERR, "get_provider: %s.",
                                    strerror(err));
                                return (NULL);
                        }
                        found = B_TRUE;
                }
        }
        if (!found)
```



```
                return (NULL);

        provider = malloc(sizeof (cryptoadm_provider_t));
        if (provider == NULL) {
                cryptoerror(LOG_STDERR, gettext("out of memory."));
                if (provstr) {
                        free(provstr);
                }
                return (NULL);
        }

        if (is_metaslot) {
                (void) strlcpy(provider->cp_name, METASLOT_KEYWORD,
                    strlen(METASLOT_KEYWORD));
                provider->cp_type = METASLOT;
        } else {

                savstr = provstr;
                (void) strtok(provstr, "=");
                provstr = strtok(NULL, "=");
                if (provstr == NULL) {
                        cryptoerror(LOG_STDERR, gettext("bad provider name."));
                        provider->cp_type = PROV_BADNAME;
                        free(savstr);
                        return (provider);
                }

                (void) strlcpy(provider->cp_name, provstr,
                    sizeof (provider->cp_name));
                provider->cp_type = get_provider_type(provider->cp_name);

                free(savstr);
        }
        return (provider);
}
```

unchanged_portion_omitted_

```
/*
 * Process the mechanism operands for the disable, enable and install
 * subcommands.  This function sets the static variable allflag to be B_TRUE
 * if the keyword "all" is specified, otherwise builds a link list of the
 * mechanism operands and save it in the static variable mecharglist.
 *
 * This function returns
 *      ERROR_USAGE: mechanism operand is missing.
 *      FAILURE: out of memory.
 *      SUCCESS: otherwise.
 */
static int
process_mech_operands(int argc, char **argv, boolean_t quiet)
{
        mechlist_t *pmech;
        mechlist_t *pcur = NULL;
        mechlist_t *phead = NULL;
        boolean_t found = B_FALSE;
        char *mechliststr = NULL;
        char *curmech = NULL;
        int c = -1;
        int rc = SUCCESS;

        while (!found && ++c < argc) {
                if ((strncmp(argv[c], KN_MECH, strlen(KN_MECH)) == 0) &&
                    strlen(argv[c]) > strlen(KN_MECH)) {
                        found = B_TRUE;
                }
        }
```

```
482         if (!found) {
483                 if (!quiet)
484                         /*
485                          * TRANSLATION_NOTE
486                          * TRANSLATION_NOTE:
487                          * "mechanism" could be either a literal keyword
488                          * and hence not to be translated, or a descriptive
489                          * word and translatable.  A choice was made to
490                          * view it as a literal keyword.
491                          */
492                         cryptoerror(LOG_STDERR,
493                                 gettext("the %s operand is missing.\n"),
494                                 "mechanism");
495                 return (ERROR_USAGE);
496         }
497         (void) strtok(argv[c], "=");
498         mechliststr = strtok(NULL, "=");
499
500         if (strcmp(mechliststr, "all") == 0) {
501                 allflag = B_TRUE;
502                 mecharglist = NULL;
503                 return (SUCCESS);
504         }
505
506         curmech = strtok(mechliststr, ",");
507         do {
508                 if ((pmech = create_mech(curmech)) == NULL) {
509                         rc = FAILURE;
510                         break;
511                 } else {
512                         if (phead == NULL) {
513                                 phead = pcur = pmech;
514                         } else {
515                                 pcur->next = pmech;
516                                 pcur = pmech;
517                         }
518                 }
519         } while ((curmech = strtok(NULL, ",")) != NULL);
520
521         if (rc == FAILURE) {
522                 cryptoerror(LOG_STDERR, gettext("out of memory."));
523                 free_mechlist(phead);
524         } else {
525                 mecharglist = phead;
526                 rc = SUCCESS;
527         }
528         return (rc);
529 }
530
531
532 /*
533  * The top level function for the list subcommand and options.
534  */
535 static int
536 do_list(int argc, char **argv)
537 {
538         boolean_t       mflag = B_FALSE;
539         boolean_t       pflag = B_FALSE;
540         boolean_t       vflag = B_FALSE;
541         char    ch;
542         cryptoadm_provider_t    *prov = NULL;
543         int     rc = SUCCESS;
544
545         argc -= 1;
546         argv += 1;
```



```
547
548         if (argc == 1) {
549                 rc = list_simple_for_all(B_FALSE);
550                 goto out;
551         }
552
553         /*
554          * [-v] [-m] [-p] [provider=<>] [mechanism=<>]
555          */
556         if (argc > 5) {
557                 usage();
558                 return (rc);
559         }
560
561         while ((ch = getopt(argc, argv, "mpv")) != EOF) {
562                 switch (ch) {
563                 case 'm':
564                         mflag = B_TRUE;
565                         if (pflag) {
566                                 rc = ERROR_USAGE;
567                         }
568                         break;
569                 case 'p':
570                         pflag = B_TRUE;
571                         if (mflag || vflag) {
572                                 rc = ERROR_USAGE;
573                         }
574                         break;
575                 case 'v':
576                         vflag = B_TRUE;
577                         if (pflag)
578                                 rc = ERROR_USAGE;
579                         break;
580                 default:
581                         rc = ERROR_USAGE;
582                         break;
583                 }
584         }
585
586         if (rc == ERROR_USAGE) {
587                 usage();
588                 return (rc);
589         }
590
591         if ((rc = process_feature_operands(argc, argv)) != SUCCESS) {
592                 goto out;
593         }
594
595         prov = get_provider(argc, argv);
596
597         if (mflag || vflag) {
598                 if (argc > 0) {
599                         rc = process_mech_operands(argc, argv, B_TRUE);
600                         if (rc == FAILURE)
601                                 goto out;
602                         /* "-m" is implied when a mechanism list is given */
603                         if (mecharglist != NULL || allflag)
604                                 mflag = B_TRUE;
605                 }
606         }
607
608         if (prov == NULL) {
609                 if (mflag) {
610                         rc = list_mechlist_for_all(vflag);
611                 } else if (pflag) {
612                         rc = list_policy_for_all();
```

```
613                 } else if (vflag) {
614                         rc = list_simple_for_all(vflag);
615                 }
616         } else if (prov->cp_type == METASLOT) {
617                 if ((!mflag) && (!vflag) && (!pflag)) {
618                         /* no flag is specified, just list metaslot status */
619                         rc = list_metaslot_info(mflag, vflag, mecharglist);
620                 } else if (mflag || vflag) {
621                         rc = list_metaslot_info(mflag, vflag, mecharglist);
622                 } else if (pflag) {
623                         rc = list_metaslot_policy();
624                 } else {
625                         /* error message */
626                         usage();
627                         rc = ERROR_USAGE;
628                 }
629         } else if (prov->cp_type == PROV_BADNAME) {
630                 usage();
631                 rc = ERROR_USAGE;
632                 goto out;
633         } else { /* do the listing for a provider only */
634                 if (mflag || vflag) {
635                         if (vflag)
636                                 (void) printf(gettext("Provider: %s\n"),
637                                         prov->cp_name);
638                         switch (prov->cp_type) {
639                         case PROV_UEF_LIB:
640                                 rc = list_mechlist_for_lib(prov->cp_name,
641                                         mecharglist, NULL, B_FALSE,
642                                         vflag, mflag);
643                                 break;
644                         case PROV_KEF_SOFT:
645                                 rc = list_mechlist_for_soft(prov->cp_name);
646                                 break;
647                         case PROV_KEF_HARD:
648                                 rc = list_mechlist_for_hard(prov->cp_name);
649                                 break;
650                         default: /* should not come here */
651                                 rc = FAILURE;
652                                 break;
653                         }
654                 } else if (pflag) {
655                         switch (prov->cp_type) {
656                         case PROV_UEF_LIB:
657                                 rc = list_policy_for_lib(prov->cp_name);
658                                 break;
659                         case PROV_KEF_SOFT:
660                                 if (getzoneid() == GLOBAL_ZONEID) {
661                                         rc = list_policy_for_soft(
662                                             prov->cp_name);
663                                 } else {
664                                         /*
665                                          * TRANSLATION_NOTE
666                                          * TRANSLATION_NOTE:
666                                          * "global" is keyword and not to
667                                          * be translated.
668                                          */
669                                         cryptoerror(LOG_STDERR, gettext(
670                                             "policy information for kernel "
671                                             "providers is available "
672                                             "in the %s zone only"), "global");
673                                         rc = FAILURE;
674                                 }
675                                 break;
676                         case PROV_KEF_HARD:
677                                 if (getzoneid() == GLOBAL_ZONEID) {
```



```
678                                         rc = list_policy_for_hard(
679                                             prov->cp_name);
680                                 } else {
681                                         /*
682                                          * TRANSLATION_NOTE
683                                          * TRANSLATION_NOTE:
683                                          * "global" is keyword and not to
684                                          * be translated.
685                                          */
686                                         cryptoerror(LOG_STDERR, gettext(
687                                             "policy information for kernel "
688                                             "providers is available "
689                                             "in the %s zone only"), "global");
690                                         rc = FAILURE;
691                                 }

693                                 break;
694                         default: /* should not come here */
695                                 rc = FAILURE;
696                                 break;
697                         }
698                 } else {
699                         /* error message */
700                         usage();
701                         rc = ERROR_USAGE;
702                 }
703         }

705 out:
706         if (prov != NULL)
707                 free(prov);

709         if (mecharglist != NULL)
710                 free_mechlist(mecharglist);
711         return (rc);
712 }


715 /*
716  * The top level function for the disable subcommand.
717  */
718 static int
719 do_disable(int argc, char **argv)
720 {
721         cryptoadm_provider_t    *prov = NULL;
722         int     rc = SUCCESS;
723         boolean_t auto_key_migrate_flag = B_FALSE;

725         if ((argc < 3) || (argc > 5)) {
726                 usage();
727                 return (ERROR_USAGE);
728         }

730         prov = get_provider(argc, argv);
731         if (prov == NULL) {
732                 usage();
733                 return (ERROR_USAGE);
734         }
735         if (prov->cp_type == PROV_BADNAME) {
736                 return (FAILURE);
737         }

739         if ((rc = process_feature_operands(argc, argv)) != SUCCESS) {
740                 goto out;
741         }
```

```
743         /*
744          * If allflag or rndflag has already been set there is no reason to
745          * process mech=
746          */
747         if (prov->cp_type == METASLOT) {
748                 if ((argc > 3) &&
749                     (rc = process_metaslot_operands(argc, argv,
750                     NULL, NULL, NULL, &auto_key_migrate_flag)) != SUCCESS) {
751                         usage();
752                         return (rc);
753                 }
754         } else if (!allflag && !rndflag &&
755                 (rc = process_mech_operands(argc, argv, B_FALSE)) != SUCCESS) {
756                         return (rc);
757         }

759         switch (prov->cp_type) {
760         case METASLOT:
761                 rc = disable_metaslot(mecharglist, allflag,
762                     auto_key_migrate_flag);
763                 break;
764         case PROV_UEF_LIB:
765                 rc = disable_uef_lib(prov->cp_name, rndflag, allflag,
766                     mecharglist);
767                 break;
768         case PROV_KEF_SOFT:
769                 if (rndflag && !allflag) {
770                         if ((mecharglist = create_mech(RANDOM)) == NULL) {
771                                 rc = FAILURE;
772                                 break;
773                         }
774                 }
775                 if (getzoneid() == GLOBAL_ZONEID) {
776                         rc = disable_kef_software(prov->cp_name, rndflag,
777                             allflag, mecharglist);
778                 } else {
779                         /*
780                          * TRANSLATION_NOTE
781                          * TRANSLATION_NOTE:
781                          * "disable" could be either a literal keyword
782                          * and hence not to be translated, or a verb and
783                          * translatable.  A choice was made to view it as
784                          * a literal keyword.  "global" is keyword and not
785                          * to be translated.
786                          */
787                         cryptoerror(LOG_STDERR, gettext("%1$s for kernel "
788                             "providers is supported in the %2$s zone only"),
789                             "disable", "global");
790                         rc = FAILURE;
791                 }
792                 break;
793         case PROV_KEF_HARD:
794                 if (rndflag && !allflag) {
795                         if ((mecharglist = create_mech(RANDOM)) == NULL) {
796                                 rc = FAILURE;
797                                 break;
798                         }
799                 }
800                 if (getzoneid() == GLOBAL_ZONEID) {
801                         rc = disable_kef_hardware(prov->cp_name, rndflag,
802                             allflag, mecharglist);
803                 } else {
804                         /*
805                          * TRANSLATION_NOTE
806                          * TRANSLATION_NOTE:
806                          * "disable" could be either a literal keyword
```



```
807                          * and hence not to be translated, or a verb and
808                          * translatable.  A choice was made to view it as
809                          * a literal keyword.  "global" is keyword and not
810                          * to be translated.
811                          */
812                         cryptoerror(LOG_STDERR, gettext("%1$s for kernel "
813                             "providers is supported in the %2$s zone only"),
814                             "disable", "global");
815                         rc = FAILURE;
816                 }
817                 break;
818         default: /* should not come here */
819                 rc = FAILURE;
820                 break;
821         }

823 out:
824         free(prov);
825         if (mecharglist != NULL) {
826                 free_mechlist(mecharglist);
827         }
828         return (rc);
829 }


832 /*
833  * The top level function fo the enable subcommand.
834  */
835 static int
836 do_enable(int argc, char **argv)
837 {
838         cryptoadm_provider_t    *prov = NULL;
839         int     rc = SUCCESS;
840         char *alt_token = NULL, *alt_slot = NULL;
841         boolean_t use_default = B_FALSE, auto_key_migrate_flag = B_FALSE;

843         if ((argc < 3) || (argc > 6)) {
844                 usage();
845                 return (ERROR_USAGE);
846         }

848         prov = get_provider(argc, argv);
849         if (prov == NULL) {
850                 usage();
851                 return (ERROR_USAGE);
852         }
853         if ((prov->cp_type != METASLOT) && (argc != 4)) {
854                 usage();
855                 return (ERROR_USAGE);
856         }
857         if (prov->cp_type == PROV_BADNAME) {
858                 rc = FAILURE;
859                 goto out;
860         }


863         if (prov->cp_type == METASLOT) {
864                 if ((rc = process_metaslot_operands(argc, argv, &alt_token,
865                     &alt_slot, &use_default, &auto_key_migrate_flag))
866                     != SUCCESS) {
867                         usage();
868                         goto out;
869                 }
870                 if ((alt_slot || alt_token) && use_default) {
871                         usage();
872                         rc = FAILURE;
```

```
873                         goto out;
874                 }
875         } else {
876                 if ((rc = process_feature_operands(argc, argv)) != SUCCESS) {
877                         goto out;
878                 }

880                 /*
881                  * If allflag or rndflag has already been set there is
882                  * no reason to process mech=
883                  */
884                 if (!allflag && !rndflag &&
885                     (rc = process_mech_operands(argc, argv, B_FALSE))
886                     != SUCCESS) {
887                         goto out;
888                 }
889         }

891         switch (prov->cp_type) {
892         case METASLOT:
893                 rc = enable_metaslot(alt_token, alt_slot, use_default,
894                     mecharglist, allflag, auto_key_migrate_flag);
895                 break;
896         case PROV_UEF_LIB:
897                 rc = enable_uef_lib(prov->cp_name, rndflag, allflag,
898                     mecharglist);
899                 break;
900         case PROV_KEF_SOFT:
901         case PROV_KEF_HARD:
902                 if (rndflag && !allflag) {
903                         if ((mecharglist = create_mech(RANDOM)) == NULL) {
904                                 rc = FAILURE;
905                                 break;
906                         }
907                 }
908                 if (getzoneid() == GLOBAL_ZONEID) {
909                         rc = enable_kef(prov->cp_name, rndflag, allflag,
910                             mecharglist);
911                 } else {
912                         /*
913                          * TRANSLATION_NOTE
914                          * TRANSLATION_NOTE:
914                          * "enable" could be either a literal keyword
915                          * and hence not to be translated, or a verb and
916                          * translatable.  A choice was made to view it as
917                          * a literal keyword.  "global" is keyword and not
918                          * to be translated.
919                          */
920                         cryptoerror(LOG_STDERR, gettext("%1$s for kernel "
921                             "providers is supported in the %2$s zone only"),
922                             "enable", "global");
923                         rc = FAILURE;
924                 }
925                 break;
926         default: /* should not come here */
927                 rc = FAILURE;
928                 break;
929         }
930 out:
931         free(prov);
932         if (mecharglist != NULL) {
933                 free_mechlist(mecharglist);
934         }
935         if (alt_token != NULL) {
936                 free(alt_token);
937         }
```



```
938         if (alt_slot != NULL) {
939                 free(alt_slot);
940         }
941         return (rc);
942 }



946 /*
947  * The top level function fo the install subcommand.
948  */
949 static int
950 do_install(int argc, char **argv)
951 {
952         cryptoadm_provider_t    *prov = NULL;
953         int     rc;

955         if (argc < 3) {
956                 usage();
957                 return (ERROR_USAGE);
958         }

960         prov = get_provider(argc, argv);
961         if (prov == NULL ||
962             prov->cp_type == PROV_BADNAME || prov->cp_type == PROV_KEF_HARD) {
963                 /*
964                  * TRANSLATION_NOTE
965                  * TRANSLATION_NOTE:
965                  * "install" could be either a literal keyword and hence
966                  * not to be translated, or a verb and translatable.  A
967                  * choice was made to view it as a literal keyword.
968                  */
969                 cryptoerror(LOG_STDERR,
970                     gettext("bad provider name for %s."), "install");
971                 rc = FAILURE;
972                 goto out;
973         }

975         if (prov->cp_type == PROV_UEF_LIB) {
976                 rc = install_uef_lib(prov->cp_name);
977                 goto out;
978         }

980         /* It is the PROV_KEF_SOFT type now  */

982         /* check if there are mechanism operands */
983         if (argc < 4) {
984                 /*
985                  * TRANSLATION_NOTE
986                  * TRANSLATION_NOTE:
986                  * "mechanism" could be either a literal keyword and hence
987                  * not to be translated, or a descriptive word and
988                  * translatable.  A choice was made to view it as a literal
989                  * keyword.
990                  */
991                 cryptoerror(LOG_STDERR,
992                     gettext("need %s operands for installing a"
993                     " kernel software provider."), "mechanism");
994                 rc = ERROR_USAGE;
995                 goto out;
996         }

998         if ((rc = process_mech_operands(argc, argv, B_FALSE)) != SUCCESS) {
999                 goto out;
1000        }
```

```
1002         if (allflag == B_TRUE) {
1003                 /*
1004                  * TRANSLATION_NOTE
1005                  * TRANSLATION_NOTE:
1005                  * "all", "mechanism", and "install" are all keywords and
1006                  * not to be translated.
1007                  */
1008                 cryptoerror(LOG_STDERR,
1009                     gettext("can not use the %1$s keyword for %2$s "
1010                     "in the %3$s subcommand."), "all", "mechanism", "install");
1011                 rc = ERROR_USAGE;
1012                 goto out;
1013         }

1015         if (getzoneid() == GLOBAL_ZONEID) {
1016                 rc = install_kef(prov->cp_name, mecharglist);
1017         } else {
1018                 /*
1019                  * TRANSLATION_NOTE
1020                  * TRANSLATION_NOTE:
1020                  * "install" could be either a literal keyword and hence
1021                  * not to be translated, or a verb and translatable.  A
1022                  * choice was made to view it as a literal keyword.
1023                  * "global" is keyword and not to be translated.
1024                  */
1025                 cryptoerror(LOG_STDERR, gettext("%1$s for kernel providers "
1026                     "is supported in the %2$s zone only"), "install", "global");
1027                 rc = FAILURE;
1028         }
1029 out:
1030         free(prov);
1031         return (rc);
1032 }



1036 /*
1037  * The top level function for the uninstall subcommand.
1038  */
1039 static int
1040 do_uninstall(int argc, char **argv)
1041 {
1042         cryptoadm_provider_t    *prov = NULL;
1043         int     rc = SUCCESS;

1045         if (argc != 3) {
1046                 usage();
1047                 return (ERROR_USAGE);
1048         }

1050         prov = get_provider(argc, argv);
1051         if (prov == NULL ||
1052             prov->cp_type == PROV_BADNAME || prov->cp_type == PROV_KEF_HARD) {
1053                 /*
1054                  * TRANSLATION_NOTE
1055                  * TRANSLATION_NOTE:
1055                  * "uninstall" could be either a literal keyword and hence
1056                  * not to be translated, or a verb and translatable.  A
1057                  * choice was made to view it as a literal keyword.
1058                  */
1059                 cryptoerror(LOG_STDERR,
1060                     gettext("bad provider name for %s."), "uninstall");
1061                 free(prov);
1062                 return (FAILURE);
1063         }
```



```
1065         if (prov->cp_type == PROV_UEF_LIB) {
1066                 rc = uninstall_uef_lib(prov->cp_name);
1067         } else if (prov->cp_type == PROV_KEF_SOFT) {
1068                 if (getzoneid() == GLOBAL_ZONEID) {
1069                         rc = uninstall_kef(prov->cp_name);
1070                 } else {
1071                         /*
1072                          * TRANSLATION_NOTE
1073                          * TRANSLATION_NOTE:
1073                          * "uninstall" could be either a literal keyword and
1074                          * hence not to be translated, or a verb and
1075                          * translatable.  A choice was made to view it as a
1076                          * literal keyword.  "global" is keyword and not to
1077                          * be translated.
1078                          */
1079                         cryptoerror(LOG_STDERR, gettext("%1$s for kernel "
1080                             "providers is supported in the %2$s zone only"),
1081                             "uninstall", "global");
1082                         rc = FAILURE;
1083                 }
1084         }

1086         free(prov);
1087         return (rc);
1088 }


1091 /*
1092  * The top level function for the unload subcommand.
1093  */
1094 static int
1095 do_unload(int argc, char **argv)
1096 {
1097         cryptoadm_provider_t    *prov = NULL;
1098         entry_t *pent;
1099         boolean_t       is_active;
1100         int rc = SUCCESS;

1102         if (argc != 3) {
1103                 usage();
1104                 return (ERROR_USAGE);
1105         }

1107         /* check if it is a kernel software provider */
1108         prov = get_provider(argc, argv);
1109         if (prov == NULL) {
1110                 cryptoerror(LOG_STDERR,
1111                     gettext("unable to determine provider name."));
1112                 goto out;
1113         }
1114         if (prov->cp_type != PROV_KEF_SOFT) {
1115                 cryptoerror(LOG_STDERR,
1116                     gettext("%s is not a valid kernel software provider."),
1117                     prov->cp_name);
1118                 rc = FAILURE;
1119                 goto out;
1120         }

1122         if (getzoneid() != GLOBAL_ZONEID) {
1123                 /*
1124                  * TRANSLATION_NOTE
1125                  * TRANSLATION_NOTE:
1125                  * "unload" could be either a literal keyword and hence
1126                  * not to be translated, or a verb and translatable.
1127                  * A choice was made to view it as a literal keyword.
1128                  * "global" is keyword and not to be translated.
```

```
1129                 */
1130                cryptoerror(LOG_STDERR, gettext("%1$s for kernel providers "
1131                    "is supported in the %2$s zone only"), "unload", "global");
1132                rc = FAILURE;
1133                goto out;
1134        }

1136        /* Check if it is in the kcf.conf file first */
1137        if ((pent = getent_kef(prov->cp_name)) == NULL) {
1138                cryptoerror(LOG_STDERR,
1139                    gettext("provider %s does not exist."), prov->cp_name);
1140                rc = FAILURE;
1141                goto out;
1142        }
1143        free_entry(pent);

1145        /* If it is unloaded already, return  */
1146        if (check_active_for_soft(prov->cp_name, &is_active) == FAILURE) {
1147                cryptodebug("internal error");
1148                cryptoerror(LOG_STDERR,
1149                    gettext("failed to unload %s."), prov->cp_name);
1150                rc = FAILURE;
1151                goto out;
1152        }

1154        if (is_active == B_FALSE) { /* unloaded already */
1155                rc = SUCCESS;
1156                goto out;
1157        } else if (unload_kef_soft(prov->cp_name, B_TRUE) == FAILURE) {
1158                cryptoerror(LOG_STDERR,
1159                    gettext("failed to unload %s."), prov->cp_name);
1160                rc = FAILURE;
1161        } else {
1162                rc = SUCCESS;
1163        }
1164 out:
1165        free(prov);
1166        return (rc);
1167 }
```

*unchanged_portion_omitted_*

```
1357 /*
1358  * List all the providers. And for each provider, list the mechanism list.
1359  */
1360 static int
1361 list_mechlist_for_all(boolean_t verbose)
1362 {
1363        crypto_get_dev_list_t   *pdevlist_kernel;
1364        uentrylist_t    *pliblist;
1365        uentrylist_t    *plibptr;
1366        entrylist_t     *pdevlist_conf;
1367        entrylist_t     *psoftlist_conf;
1368        entrylist_t     *pdevlist_zone;
1369        entrylist_t     *psoftlist_zone;
1370        entrylist_t     *ptr;
1371        mechlist_t      *pmechlist;
1372        boolean_t       is_active;
1373        char    provname[MAXNAMELEN];
1374        char    devname[MAXNAMELEN];
1375        int     inst_num;
1376        int     count;
1377        int     i;
1378        int     rv;
1379        int     rc = SUCCESS;
```



```
1381        /* get user-level providers */
1382        (void) printf(gettext("\nUser-level providers:\n"));
1383        /*
1384         * TRANSLATION_NOTE
1385         * TRANSLATION_NOTE:
1385         * Strictly for appearance's sake, this line should be as long as
1386         * the length of the translated text above.
1387         */
1388        (void) printf(gettext("====================\n"));
1389        if (get_pkcs11conf_info(&pliblist) != SUCCESS) {
1390                cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1391                    "the list of user-level providers.\n"));
1392                rc = FAILURE;
1393        }

1395        plibptr = pliblist;
1396        while (plibptr != NULL) {
1397                /* skip metaslot entry */
1398                if (strcmp(plibptr->puent->name, METASLOT_KEYWORD) != 0) {
1399                        (void) printf(gettext("\nProvider: %s\n"),
1400                            plibptr->puent->name);
1401                        rv = list_mechlist_for_lib(plibptr->puent->name,
1402                            mecharglist, NULL, B_FALSE, verbose, B_TRUE);
1403                        if (rv == FAILURE) {
1404                                rc = FAILURE;
1405                        }
1406                }
1407                plibptr = plibptr->next;
1408        }
1409        free_uentrylist(pliblist);

1411        /* get kernel software providers */
1412        (void) printf(gettext("\nKernel software providers:\n"));
1413        /*
1414         * TRANSLATION_NOTE
1415         * TRANSLATION_NOTE:
1415         * Strictly for appearance's sake, this line should be as long as
1416         * the length of the translated text above.
1417         */
1418        (void) printf(gettext("==========================\n"));
1419        if (getzoneid() == GLOBAL_ZONEID) {
1420                /* use kcf.conf for kernel software providers in global zone */
1421                pdevlist_conf = NULL;
1422                psoftlist_conf = NULL;

1424                if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf) !=
1425                    SUCCESS) {
1426                        cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1427                            "the list of kernel software providers.\n"));
1428                        rc = FAILURE;
1429                }

1431                ptr = psoftlist_conf;
1432                while (ptr != NULL) {
1433                        if (check_active_for_soft(ptr->pent->name, &is_active)
1434                            == SUCCESS) {
1435                                if (is_active) {
1436                                        rv = list_mechlist_for_soft(
1437                                            ptr->pent->name);
1438                                        if (rv == FAILURE) {
1439                                                rc = FAILURE;
1440                                        }
1441                                } else {
1442                                        (void) printf(gettext(
1443                                            "%s: (inactive)\n"),
```

```
1444                                            ptr->pent->name);
1445                                }
1446                        } else {
1447                                /* should not happen */
1448                                (void) printf(gettext(
1449                                    "%s: failed to get the mechanism list.\n"),
1450                                    ptr->pent->name);
1451                                rc = FAILURE;
1452                        }
1453                        ptr = ptr->next;
1454                }

1456                free_entrylist(pdevlist_conf);
1457                free_entrylist(psoftlist_conf);
1458        } else {
1459                /* kcf.conf not there in non-global zone, use /dev/cryptoadm */
1460                pdevlist_zone = NULL;
1461                psoftlist_zone = NULL;

1463                if (get_admindev_info(&pdevlist_zone, &psoftlist_zone) !=
1464                    SUCCESS) {
1465                        cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1466                            "the list of kernel software providers.\n"));
1467                        rc = FAILURE;
1468                }

1470                ptr = psoftlist_zone;
1471                while (ptr != NULL) {
1472                        rv = list_mechlist_for_soft(ptr->pent->name);
1473                        if (rv == FAILURE) {
1474                                (void) printf(gettext(
1475                                    "%s: failed to get the mechanism list.\n"),
1476                                    ptr->pent->name);
1477                                rc = FAILURE;
1478                        }
1479                        ptr = ptr->next;
1480                }

1482                free_entrylist(pdevlist_zone);
1483                free_entrylist(psoftlist_zone);
1484        }

1486        /* Get kernel hardware providers and their mechanism lists */
1487        (void) printf(gettext("\nKernel hardware providers:\n"));
1488        /*
1489         * TRANSLATION_NOTE
1490         * TRANSLATION_NOTE:
1490         * Strictly for appearance's sake, this line should be as long as
1491         * the length of the translated text above.
1492         */
1493        (void) printf(gettext("=========================\n"));
1494        if (get_dev_list(&pdevlist_kernel) != SUCCESS) {
1495                cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1496                    "the list of hardware providers.\n"));
1497                return (FAILURE);
1498        }

1500        for (i = 0; i < pdevlist_kernel->dl_dev_count; i++) {
1501                (void) strlcpy(devname,
1502                    pdevlist_kernel->dl_devs[i].le_dev_name, MAXNAMELEN);
1503                inst_num = pdevlist_kernel->dl_devs[i].le_dev_instance;
1504                count = pdevlist_kernel->dl_devs[i].le_mechanism_count;
1505                (void) snprintf(provname, sizeof (provname), "%s/%d", devname,
1506                    inst_num);
1507                if (get_dev_info(devname, inst_num, count, &pmechlist) ==
1508                    SUCCESS) {
```



```
1509                        (void) filter_mechlist(&pmechlist, RANDOM);
1510                        print_mechlist(provname, pmechlist);
1511                        free_mechlist(pmechlist);
1512                } else {
1513                        (void) printf(gettext("%s: failed to get the mechanism"
1514                            " list.\n"), provname);
1515                        rc = FAILURE;
1516                }
1517        }
1518        free(pdevlist_kernel);
1519        return (rc);
1520 }


1523 /*
1524  * List all the providers. And for each provider, list the policy information.
1525  */
1526 static int
1527 list_policy_for_all(void)
1528 {
1529        crypto_get_dev_list_t   *pdevlist_kernel;
1530        uentrylist_t    *pliblist;
1531        uentrylist_t    *plibptr;
1532        entrylist_t     *pdevlist_conf;
1533        entrylist_t     *psoftlist_conf;
1534        entrylist_t     *ptr;
1535        entrylist_t     *phead;
1536        boolean_t       found;
1537        char    provname[MAXNAMELEN];
1538        int     i;
1539        int     rc = SUCCESS;

1541        /* Get user-level providers */
1542        (void) printf(gettext("\nUser-level providers:\n"));
1543        /*
1544         * TRANSLATION_NOTE
1545         * TRANSLATION_NOTE:
1545         * Strictly for appearance's sake, this line should be as long as
1546         * the length of the translated text above.
1547         */
1548        (void) printf(gettext("====================\n"));
1549        if (get_pkcs11conf_info(&pliblist) == FAILURE) {
1550                cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1551                    "the list of user-level providers.\n"));
1552        } else {
1553                plibptr = pliblist;
1554                while (plibptr != NULL) {
1555                        /* skip metaslot entry */
1556                        if (strcmp(plibptr->puent->name,
1557                            METASLOT_KEYWORD) != 0) {
1558                                if (print_uef_policy(plibptr->puent)
1559                                    == FAILURE) {
1560                                        rc = FAILURE;
1561                                }
1562                        }
1563                        plibptr = plibptr->next;
1564                }
1565                free_uentrylist(pliblist);
1566        }

1568        /* kernel software providers */
1569        (void) printf(gettext("\nKernel software providers:\n"));
1570        /*
1571         * TRANSLATION_NOTE
1572         * TRANSLATION_NOTE:
1572         * Strictly for appearance's sake, this line should be as long as
```

```
1573          * the length of the translated text above.
1574          */
1575         (void) printf(gettext("=========================\n"));

1577         /* Get all entries from the kcf.conf file */
1578         pdevlist_conf = NULL;
1579         if (getzoneid() == GLOBAL_ZONEID) {
1580                 /* use kcf.conf for kernel software providers in global zone */
1581                 psoftlist_conf = NULL;

1583                 if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf) ==
1584                     FAILURE) {
1585                         cryptoerror(LOG_STDERR, gettext(
1586                             "failed to retrieve the list of kernel "
1587                             "providers.\n"));
1588                         return (FAILURE);
1589                 }

1591                 ptr = psoftlist_conf;
1592                 while (ptr != NULL) {
1593                         (void) list_policy_for_soft(ptr->pent->name);
1594                         ptr = ptr->next;
1595                 }

1597                 free_entrylist(psoftlist_conf);
1598         } else {
1599                 /* kcf.conf not there in non-global zone, no policy info */

1601                 /*
1602                  * TRANSLATION_NOTE
1603                  * TRANSLATION_NOTE:
1603                  * "global" is keyword and not to be translated.
1604                  */
1605                 cryptoerror(LOG_STDERR, gettext(
1606                     "policy information for kernel software providers is "
1607                     "available in the %s zone only"), "global");
1608         }

1610         /* Kernel hardware providers */
1611         (void) printf(gettext("\nKernel hardware providers:\n"));
1612         /*
1613          * TRANSLATION_NOTE
1614          * TRANSLATION_NOTE:
1614          * Strictly for appearance's sake, this line should be as long as
1615          * the length of the translated text above.
1616          */
1617         (void) printf(gettext("=========================\n"));

1619         if (getzoneid() != GLOBAL_ZONEID) {
1620                 /*
1621                  * TRANSLATION_NOTE
1622                  * TRANSLATION_NOTE:
1622                  * "global" is keyword and not to be translated.
1623                  */
1624                 cryptoerror(LOG_STDERR, gettext(
1625                     "policy information for kernel hardware providers is "
1626                     "available in the %s zone only"), "global");
1627                 return (FAILURE);
1628         }

1630         /* Get the hardware provider list from kernel */
1631         if (get_dev_list(&pdevlist_kernel) != SUCCESS) {
1632                 cryptoerror(LOG_STDERR, gettext(
1633                     "failed to retrieve the list of hardware providers.\n"));
1634                 free_entrylist(pdevlist_conf);
1635                 return (FAILURE);
```



```
1636         }

1638         /*
1639          * For each hardware provider from kernel, check if it has an entry
1640          * in the config file.  If it has an entry, print out the policy from
1641          * config file and remove the entry from the hardware provider list
1642          * of the config file.  If it does not have an entry in the config
1643          * file, no mechanisms of it have been disabled. But, we still call
1644          * list_policy_for_hard() to account for the "random" feature.
1645          */
1646         for (i = 0; i < pdevlist_kernel->dl_dev_count; i++) {
1647                 (void) snprintf(provname, sizeof (provname), "%s/%d",
1648                     pdevlist_kernel->dl_devs[i].le_dev_name,
1649                     pdevlist_kernel->dl_devs[i].le_dev_instance);
1650                 found = B_FALSE;
1651                 phead = ptr = pdevlist_conf;
1652                 while (!found && ptr) {
1653                         if (strcmp(ptr->pent->name, provname) == 0) {
1654                                 found = B_TRUE;
1655                         } else {
1656                                 phead = ptr;
1657                                 ptr = ptr->next;
1658                         }
1659                 }

1661                 if (found) {
1662                         (void) list_policy_for_hard(ptr->pent->name);
1663                         if (phead == ptr) {
1664                                 pdevlist_conf = pdevlist_conf->next;
1665                         } else {
1666                                 phead->next = ptr->next;
1667                         }
1668                         free_entry(ptr->pent);
1669                         free(ptr);
1670                 } else {
1671                         (void) list_policy_for_hard(provname);
1672                 }
1673         }

1675         /*
1676          * If there are still entries left in the pdevlist_conf list from
1677          * the config file, these providers must have been detached.
1678          * Should print out their policy information also.
1679          */
1680         ptr = pdevlist_conf;
1681         while (ptr != NULL) {
1682                 print_kef_policy(ptr->pent, B_FALSE, B_TRUE);
1683                 ptr = ptr->next;
1684         }

1686         free_entrylist(pdevlist_conf);
1687         free(pdevlist_kernel);

1689         return (rc);
1690 }
```

*unchanged_portion_omitted_*