```
**********************************************************
   13975 Mon Jun 30 15:20:55 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_metaslot.c
5031131 perf: pkcs11_kernel can benefit from a more efficient pkcs11_mech2str()
**********************************************************
```

```
/*
 * 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   "@(#)adm_metaslot.c     1.4     08/06/27 SMI"
#pragma ident   "@(#)adm_metaslot.c     1.3     05/06/08 SMI"

/*
 * 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:
         * 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:
                 * Strictly for appearance's sake, this line should be as
                 * long as the length of the translated text above.
                 */
                (void) printf(gettext("------------------------------\n"));
```

```
123         }
124
125         /*
126          * Need to actually make calls to libpkcs11.so to get
127          * information about metaslot.
128          */
129
130         dldesc = dlopen(UEF_FRAME_LIB, RTLD_NOW);
131         if (dldesc == NULL) {
132                 char *dl_error;
133                 dl_error = dlerror();
134                 cryptodebug("Cannot load PKCS#11 framework library. "
135                     "dlerror:%s", dl_error);
136                 return (FAILURE);
137         }
138
139         /* Get the pointer to library's C_GetFunctionList() */
140         Tmp_C_GetFunctionList = (CK_RV(*)())dlsym(dldesc, "C_GetFunctionList");
141         if (Tmp_C_GetFunctionList == NULL) {
142                 cryptodebug("Cannot get the address of the C_GetFunctionList "
143                     "from framework");
144                 rc = FAILURE;
145                 goto finish;
146         }
147
148
149         /* Get the provider's function list */
150         rv = Tmp_C_GetFunctionList(&funcs);
151         if (rv != CKR_OK) {
152                 cryptodebug("failed to call C_GetFunctionList in "
153                     "framework library");
154                 rc = FAILURE;
155                 goto finish;
156         }
157
158         /* Initialize this provider */
159         rv = funcs->C_Initialize(NULL_PTR);
160         if (rv != CKR_OK) {
161                 cryptodebug("C_Initialize failed with error code 0x%x\n", rv);
162                 rc = FAILURE;
163                 goto finish;
164         } else {
165                 lib_initialized = B_TRUE;
166         }
167
168         /*
169          * We know for sure that metaslot is slot 0 in the framework,
170          * so, we will do a C_GetSlotInfo() trying to see if it works.
171          * If it fails with CKR_SLOT_ID_INVALID, we know that metaslot
172          * If it failes with CKR_SLOT_ID_INVALID, we know that metaslot
173          * is not really enabled.
174          */
175         rv = funcs->C_GetSlotInfo(METASLOT_ID, &slot_info);
176         if (rv == CKR_SLOT_ID_INVALID) {
177                 (void) printf(gettext("actual status: disabled.\n"));
178                 /*
179                  * Even if the -m and -v flag is supplied, there's nothing
180                  * interesting to display about metaslot since it is disabled,
181                  * so, just stop right here.
182                  */
183                 goto finish;
184         }
185
186         if (rv != CKR_OK) {
187                 cryptodebug("C_GetSlotInfo failed with error "
188                     "code 0x%x\n", rv);
```



```
189                 rc = FAILURE;
190                 goto finish;
191         }
192
193         if (!verbose) {
194                 goto display_mechs;
195         }
196
197         (void) printf(gettext("actual status: enabled.\n"));
198
199         (void) printf(gettext("Description: %.64s\n"),
200             slot_info.slotDescription);
201
202         (void) printf(gettext("Token Present: %s\n"),
203             (slot_info.flags & CKF_TOKEN_PRESENT ?
204             gettext("True") : gettext("False")));
205
206         rv = funcs->C_GetTokenInfo(METASLOT_ID, &token_info);
207         if (rv != CKR_OK) {
208                 cryptodebug("C_GetTokenInfo failed with error "
209                     "code 0x%x\n", rv);
210                 rc = FAILURE;
211                 goto finish;
212         }
213
214         (void) printf(gettext("Token Label: %.32s\n"
215             "Manufacturer ID: %.32s\n"
216             "Model: %.16s\n"
217             "Serial Number: %.16s\n"
218             "Hardware Version: %d.%d\n"
219             "Firmware Version: %d.%d\n"
220             "UTC Time: %.16s\n"
221             "PIN Length: %d-%d\n"),
222             token_info.label,
223             token_info.manufacturerID,
224             token_info.model,
225             token_info.serialNumber,
226             token_info.hardwareVersion.major,
227             token_info.hardwareVersion.minor,
228             token_info.firmwareVersion.major,
229             token_info.firmwareVersion.minor,
230             token_info.utcTime,
231             token_info.ulMinPinLen,
232             token_info.ulMaxPinLen);
233
234         display_token_flags(token_info.flags);
235
236         if (!show_mechs) {
237                 goto finish;
238         }
239
240 display_mechs:
241
242         if (mechlist == NULL) {
243                 rv = funcs->C_GetMechanismList(METASLOT_ID, NULL_PTR,
244                     &mech_count);
245                 if (rv != CKR_OK) {
246                         cryptodebug("C_GetMechanismList failed with error "
247                             "code 0x%x\n", rv);
248                         rc = FAILURE;
249                         goto finish;
250                 }
251
252                 if (mech_count > 0) {
253                         pmech_list = malloc(mech_count *
254                             sizeof (CK_MECHANISM_TYPE));
```

```
254                         if (pmech_list == NULL) {
255                                 cryptodebug("out of memory");
256                                 rc = FAILURE;
257                                 goto finish;
258                         }
259                         rv = funcs->C_GetMechanismList(METASLOT_ID, pmech_list,
260                             &mech_count);
261                         if (rv != CKR_OK) {
262                                 cryptodebug("C_GetMechanismList failed with "
263                                     "error code 0x%x\n", rv);
264                                 rc = FAILURE;
265                                 goto finish;
266                         }
267                 }
268         } else {
269                 rc = convert_mechlist(&pmech_list, &mech_count, mechlist);
270                 if (rc != SUCCESS) {
271                         goto finish;
272                 }
273         }

275         (void) printf(gettext("Mechanisms:\n"));
276         if (mech_count == 0) {
277                 /* should never be this case */
278                 (void) printf(gettext("No mechanisms\n"));
279                 goto finish;
280         }
281         if (verbose) {
282                 display_verbose_mech_header();
283         }

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

288                 if (mech > CKM_VENDOR_DEFINED) {
289                         (void) printf("%#lx", mech);
290                 } else {
291                         (void) printf("%-29s", pkcs11_mech2str(mech));
292                 }

287                 (void) printf("%-29s", pkcs11_mech2str(pmech_list[i]));
294                 if (verbose) {
295                         CK_MECHANISM_INFO mech_info;
296                         rv = funcs->C_GetMechanismInfo(METASLOT_ID,
297                             mech, &mech_info);
291                             pmech_list[i], &mech_info);
298                         if (rv != CKR_OK) {
299                                 cryptodebug("C_GetMechanismInfo failed with "
300                                     "error code 0x%x\n", rv);
301                                 rc = FAILURE;
302                                 goto finish;
303                         }
304                         display_mech_info(&mech_info);
305                 }
306                 (void) printf("\n");
307         }

309 finish:

311         if ((rc == FAILURE) && (show_mechs)) {
312                 (void) printf(gettext(
313                     "metaslot: failed to retrieve the mechanism list.\n"));
314         }

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



```
318         }

320         if (dldesc != NULL) {
321                 (void) dlclose(dldesc);
322         }

324         if (pmech_list != NULL) {
325                 (void) free(pmech_list);
326         }

328         return (rc);
329 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   45490 Mon Jun 30 15:20:57 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_uef.c
5031131 perf: pkcs11_kernel can benefit from a more efficient pkcs11_mech2str()
**********************************************************
```

```
/*
 * 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   "@(#)adm_uef.c  1.13    08/06/27 SMI"
#pragma ident   "@(#)adm_uef.c  1.12    06/11/02 SMI"

#include <cryptoutil.h>
#include <fcntl.h>
#include <libintl.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <errno.h>
#include <dlfcn.h>
#include <link.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <security/cryptoki.h>
#include "cryptoadm.h"

#define HDR1 "                                  P\n"
#define HDR2 "                        S     V  K  a     U  D\n"
#define HDR3 "                        i     e  e  i     n  e\n"
#define HDR4 "                     S  g  V  r  y  r  W  w  r\n"
#define HDR5 "            E  D  D  i  n  e  i  G  G  r  r  i\n"
#define HDR6 "         H  n  e  i  g  +  r  +  e  e  a  a  v  E\n"
#define HDR7 "min  max  W  c  c  g  n  R  i  R  n  n  p  p  e  C\n"


static int err; /* To store errno which may be overwritten by gettext() */
static boolean_t is_in_policylist(midstr_t, umechlist_t *);
static char *uent2str(uentry_t *);
static boolean_t check_random(CK_SLOT_ID, CK_FUNCTION_LIST_PTR);

static void display_slot_flags(CK_FLAGS flags)
{
        (void) printf(gettext("Slot Flags: "));
```



```
        if (flags & CKF_TOKEN_PRESENT)
                (void) printf("CKF_TOKEN_PRESENT ");
        if (flags & CKF_REMOVABLE_DEVICE)
                (void) printf("CKF_REMOVABLE_DEVICE ");
        if (flags & CKF_HW_SLOT)
                (void) printf("CKF_HW_SLOT ");
        (void) printf("\n");
}
______unchanged_portion_omitted_

/*
 * Converts the provided list of mechanism names in their string format to
 * their corresponding PKCS#11 mechanism IDs.
 * their corrsponding PKCS#11 mechanism IDs.
 *
 * The list of mechanism names to be converted is provided in the
 * "mlist" argument.  The list of converted mechanism IDs is returned
 * in the "pmech_list" argument.
 *
 * This function is called by list_metaslot_info() and
 * list_mechlist_for_lib() functions.
 */
int
convert_mechlist(CK_MECHANISM_TYPE **pmech_list, CK_ULONG *mech_count,
    mechlist_t *mlist)
{
        int i, n = 0;
        mechlist_t *p = mlist;

        while (p != NULL) {
                p = p->next;
                n++;
        }

        *pmech_list = malloc(n * sizeof (CK_MECHANISM_TYPE));
        if (pmech_list == NULL) {
                cryptodebug("out of memory");
                return (FAILURE);
        }
        p = mlist;
        for (i = 0; i < n; i++) {
                if (pkcs11_str2mech(p->name, &(*pmech_list[i])) != CKR_OK) {
                        free(*pmech_list);
                        return (FAILURE);
                }
                p = p->next;
        }
        *mech_count = n;
        return (SUCCESS);
}

/*
 * Display the mechanism list for a user-level library
 */
int
list_mechlist_for_lib(char *libname, mechlist_t *mlist,
                flag_val_t *rng_flag, boolean_t no_warn,
                boolean_t verbose, boolean_t show_mechs)
{
        CK_RV   rv = CKR_OK;
        CK_RV   (*Tmp_C_GetFunctionList)(CK_FUNCTION_LIST_PTR_PTR);
        CK_FUNCTION_LIST_PTR    prov_funcs; /* Provider's function list */
        CK_SLOT_ID_PTR          prov_slots = NULL; /* Provider's slot list */
        CK_MECHANISM_TYPE_PTR   pmech_list; /* mechanism list for a slot */
        CK_SLOT_INFO    slotinfo;
        CK_ULONG        slot_count;
```

```
195         CK_ULONG        mech_count;
196         uentry_t        *puent = NULL;
197         boolean_t       lib_initialized = B_FALSE;
198         void            *dldesc = NULL;
199         char            *dl_error;
200         const char      *mech_name;
200         char    *mech_name;
201         char            *isa;
202         char            libpath[MAXPATHLEN];
203         char            buf[MAXPATHLEN];
204         int             i, j;
205         int             rc = SUCCESS;

207         if (libname == NULL) {
208                 /* should not happen */
209                 cryptoerror(LOG_STDERR, gettext("internal error."));
210                 cryptodebug("list_mechlist_for_lib() - libname is NULL.");
211                 return (FAILURE);
212         }

214         /* Check if the library is in the pkcs11.conf file */
215         if ((puent = getent_uef(libname)) == NULL) {
216                 cryptoerror(LOG_STDERR,
217                     gettext("%s does not exist."), libname);
218                 return (FAILURE);
219         }
220         free_uentry(puent);

222         /* Remove $ISA from the library name */
223         if (strlcpy(buf, libname, sizeof (buf)) >= sizeof (buf)) {
224                 (void) printf(gettext("%s: the provider name is too long."),
225                     libname);
226                 return (FAILURE);
227         }

229         if ((isa = strstr(buf, PKCS11_ISA)) != NULL) {
230                 *isa = '\000';
231                 isa += strlen(PKCS11_ISA);
232                 (void) snprintf(libpath, MAXPATHLEN, "%s%s%s", buf, "/", isa);
233         } else {
234                 (void) strlcpy(libpath, libname, sizeof (libpath));
235         }

237         /* Open the provider */
238         dldesc = dlopen(libpath, RTLD_NOW);
239         if (dldesc == NULL) {
240                 dl_error = dlerror();
241                 cryptodebug("Cannot load PKCS#11 library %s.  dlerror: %s",
242                     libname, dl_error != NULL ? dl_error : "Unknown");
243                 rc = FAILURE;
244                 goto clean_exit;
245         }

247         /* Get the pointer to provider's C_GetFunctionList() */
248         Tmp_C_GetFunctionList = (CK_RV(*)())dlsym(dldesc, "C_GetFunctionList");
249         if (Tmp_C_GetFunctionList == NULL) {
250                 cryptodebug("Cannot get the address of the C_GetFunctionList "
251                     "from %s", libname);
252                 rc = FAILURE;
253                 goto clean_exit;
254         }

256         /* Get the provider's function list */
257         rv = Tmp_C_GetFunctionList(&prov_funcs);
258         if (rv != CKR_OK) {
259                 cryptodebug("failed to call C_GetFunctionList from %s",
```



```
260                     libname);
261                 rc = FAILURE;
262                 goto clean_exit;
263         }

265         /* Initialize this provider */
266         rv = prov_funcs->C_Initialize(NULL_PTR);
267         if (rv != CKR_OK) {
268                 cryptodebug("failed to call C_Initialize from %s, "
269                     "return code = %d", libname, rv);
270                 rc = FAILURE;
271                 goto clean_exit;
272         } else {
273                 lib_initialized = B_TRUE;
274         }

276         /*
277          * Find out how many slots this provider has, call with tokenPresent
278          * set to FALSE so all potential slots are returned.
279          */
280         rv = prov_funcs->C_GetSlotList(FALSE, NULL_PTR, &slot_count);
281         if (rv != CKR_OK) {
282                 cryptodebug("failed to get the slotlist from %s.", libname);
283                 rc = FAILURE;
284                 goto clean_exit;
285         } else if (slot_count == 0) {
286                 if (!no_warn)
287                         (void) printf(gettext("%s: no slots presented.\n"),
288                             libname);
289                 rc = SUCCESS;
290                 goto clean_exit;
291         }

293         /* Allocate memory for the slot list */
294         prov_slots = malloc(slot_count * sizeof (CK_SLOT_ID));
295         if (prov_slots == NULL) {
296                 cryptodebug("out of memory.");
297                 rc = FAILURE;
298                 goto clean_exit;
299         }

301         /* Get the slot list from provider */
302         rv = prov_funcs->C_GetSlotList(FALSE, prov_slots, &slot_count);
303         if (rv != CKR_OK) {
304                 cryptodebug("failed to call C_GetSlotList() from %s.",
305                     libname);
306                 rc = FAILURE;
307                 goto clean_exit;
308         }

310         if (verbose) {
311                 (void) printf(gettext("Number of slots: %d\n"), slot_count);
312         }

314         /* Get the mechanism list for each slot */
315         for (i = 0; i < slot_count; i++) {
316                 if (verbose)
317                         /*
318                          * TRANSLATION_NOTE:
319                          * In some languages, the # symbol should be
320                          * converted to "no", an "n" followed by a
321                          * superscript "o"..
322                          */
323                         (void) printf(gettext("\nSlot #%d\n"), i+1);

325                 if ((rng_flag != NULL) && (*rng_flag == NO_RNG)) {
```

```
326                         if (check_random(prov_slots[i], prov_funcs)) {
327                                 *rng_flag = HAS_RNG;
328                                 rc = SUCCESS;
329                                 goto clean_exit;
330                         } else
331                                 continue;
332                 }

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

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

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

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

362                         display_slot_flags(slotinfo.flags);

364                         rv = prov_funcs->C_GetTokenInfo(prov_slots[i],
365                             &tokeninfo);
366                         if (rv != CKR_OK) {
367                                 cryptodebug("Failed to get "
368                                     "token info from %s", libname);
369                                 rc = FAILURE;
370                                 break;
371                         }

373                         (void) printf(gettext("Token Label: %.32s\n"
374                             "Manufacturer ID: %.32s\n"
375                             "Model: %.16s\n"
376                             "Serial Number: %.16s\n"
377                             "Hardware Version: %d.%d\n"
378                             "Firmware Version: %d.%d\n"
379                             "UTC Time: %.16s\n"
380                             "PIN Length: %d-%d\n"),
381                             tokeninfo.label,
382                             tokeninfo.manufacturerID,
383                             tokeninfo.model,
384                             tokeninfo.serialNumber,
385                             tokeninfo.hardwareVersion.major,
386                             tokeninfo.hardwareVersion.minor,
387                             tokeninfo.firmwareVersion.major,
388                             tokeninfo.firmwareVersion.minor,
389                             tokeninfo.utcTime,
390                             tokeninfo.ulMinPinLen,
391                             tokeninfo.ulMaxPinLen);
```



```
393                         display_token_flags(tokeninfo.flags);
394                 }

396                 if (mlist == NULL) {
397                         rv = prov_funcs->C_GetMechanismList(prov_slots[i],
398                             NULL_PTR, &mech_count);
399                         if (rv != CKR_OK) {
400                                 cryptodebug(
401                                     "failed to call C_GetMechanismList() "
402                                     "from %s.", libname);
403                                 rc = FAILURE;
404                                 break;
405                         }

407                         if (mech_count == 0) {
408                                 /* no mechanisms in this slot */
409                                 continue;
410                         }

412                         pmech_list = malloc(mech_count *
413                             sizeof (CK_MECHANISM_TYPE));
414                         if (pmech_list == NULL) {
415                                 cryptodebug("out of memory");
416                                 rc = FAILURE;
417                                 break;
418                         }

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

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

451                         if (mech > CKM_VENDOR_DEFINED) {
452                                 (void) printf("%#lx", mech);
453                         } else {
454                                 mech_name = pkcs11_mech2str(mech);
449                         mech_name = pkcs11_mech2str(pmech_list[j]);
455                                 (void) printf("%-29s", mech_name);
456                         }
```

```
458                         if (verbose) {
459                                 CK_MECHANISM_INFO mech_info;
460                                 rv = prov_funcs->C_GetMechanismInfo(
461                                     prov_slots[i], mech, &mech_info);
454                                     prov_slots[i], pmech_list[j], &mech_info);
462                                 if (rv != CKR_OK) {
463                                       cryptodebug(
464                                           "failed to call "
465                                           "C_GetMechanismInfo() from %s.",
466                                           libname);
467                                       (void) free(pmech_list);
468                                       rc = FAILURE;
469                                       break;
470                                 }
471                                 display_mech_info(&mech_info);
472                         }
473                         (void) printf("\n");
474                 }
475                 (void) free(pmech_list);
476                 if (rc == FAILURE) {
477                         break;
478                 }
479         }

481         if (rng_flag != NULL || rc == FAILURE) {
482                 goto clean_exit;
483         }

485 clean_exit:

487         if (rc == FAILURE) {
488                 (void) printf(gettext(
489                     "%s: failed to retrieve the mechanism list.\n"), libname);
490         }

492         if (lib_initialized) {
493                 (void) prov_funcs->C_Finalize(NULL_PTR);
494         }

496         if (dldesc != NULL) {
497                 (void) dlclose(dldesc);
498         }

500         if (prov_slots != NULL) {
501                 (void) free(prov_slots);
502         }

504         return (rc);
505 }
```

unchanged_portion_omitted_

```
1121 int
1122 display_policy(uentry_t *puent)
1123 {
1124         CK_MECHANISM_TYPE       mech_id;
1125         const char              *mech_name;
1118         char *mech_name;
1126         umechlist_t             *ptr;

1128         if (puent == NULL) {
1129                 return (SUCCESS);
1130         }

1132         if (puent->flag_enabledlist == B_FALSE) {
```



```
1133                 (void) printf(gettext("%s: all mechanisms are enabled"),
1134                     puent->name);
1135                 ptr = puent->policylist;
1136                 if (ptr == NULL) {
1137                         (void) printf(".");
1138                 } else {
1139                         (void) printf(gettext(", except "));
1140                         while (ptr != NULL) {
1141                                 mech_id = strtoul(ptr->name, NULL, 0);
1142                                 if (mech_id & CKO_VENDOR_DEFINED) {
1143                                         /* vendor defined mechanism */
1144                                         (void) printf("%s", ptr->name);
1145                                 } else {
1146                                         if (mech_id > CKM_VENDOR_DEFINED) {
1147                                                 (void) printf("%#lx", mech_id);
1148                                         } else {
1149                                                 mech_name = pkcs11_mech2str(
1150                                                     mech_id);
1139                                         mech_name = pkcs11_mech2str(mech_id);
1151                                                 if (mech_name == NULL) {
1152                                                         return (FAILURE);
1153                                                 }
1154                                                 (void) printf("%s", mech_name);
1144                                         free(mech_name);
1155                                         }
1156                                 }

1158                                 ptr = ptr->next;
1159                                 if (ptr == NULL) {
1160                                         (void) printf(".");
1161                                 } else {
1162                                         (void) printf(",");
1163                                 }
1164                         }
1165                 }
1166         } else { /* puent->flag_enabledlist == B_TRUE */
1167                 (void) printf(gettext("%s: all mechanisms are disabled"),
1168                     puent->name);
1169                 ptr = puent->policylist;
1170                 if (ptr == NULL) {
1171                         (void) printf(".");
1172                 } else {
1173                         (void) printf(gettext(", except "));
1174                         while (ptr != NULL) {
1175                                 mech_id = strtoul(ptr->name, NULL, 0);
1176                                 if (mech_id & CKO_VENDOR_DEFINED) {
1177                                         /* vendor defined mechanism */
1178                                         (void) printf("%s", ptr->name);
1179                                 } else {
1180                                         mech_name = pkcs11_mech2str(mech_id);
1181                                         if (mech_name == NULL) {
1182                                                 return (FAILURE);
1183                                         }
1184                                         (void) printf("%s", mech_name);
1174                                         free(mech_name);
1185                                 }
1186                                 ptr = ptr->next;
1187                                 if (ptr == NULL) {
1188                                         (void) printf(".");
1189                                 } else {
1190                                         (void) printf(",");
1191                                 }
1192                         }
1193                 }
1194         }
1195         return (SUCCESS);
```

```
1196 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4965 Mon Jun 30 15:21:00 2008
new/usr/src/lib/libcryptoutil/common/cryptoutil.h
5031131 perf: pkcs11_kernel can benefit from a more efficient pkcs11_mech2str()
**********************************************************
```

```
/*
 * 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.
 */

#ifndef _CRYPTOUTIL_H
#define _CRYPTOUTIL_H

#pragma ident   "@(#)cryptoutil.h       1.12    08/06/27 SMI"
#pragma ident   "@(#)cryptoutil.h       1.11    08/02/20 SMI"

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <syslog.h>
#include <security/cryptoki.h>
#include <sys/param.h>

#define LOG_STDERR      -1
#define SUCCESS         0
#define FAILURE         1
#define MECH_ID_HEX_LEN 11      /* length of mechanism id in hex form */

#define _PATH_PKCS11_CONF       "/etc/crypto/pkcs11.conf"
#define _PATH_KCFD_LOCK         "/var/run/kcfd.lock"

/* $ISA substitution for parsing pkcs11.conf data */
#define PKCS11_ISA      "/$ISA/"
#if defined(_LP64)
#define PKCS11_ISA_DIR  "/64/"
#else   /* !_LP64 */
#define PKCS11_ISA_DIR  "/"
#endif

/* keywords and delimiters for parsing configuration files */
#define SEP_COLON       ":"
#define SEP_SEMICOLON   ";"
#define SEP_EQUAL       "="
#define SEP_COMMA       ","
```



```
#define METASLOT_KEYWORD        "metaslot"
#define EF_DISABLED     "disabledlist="
#define EF_ENABLED      "enabledlist="
#define EF_NORANDOM     "NO_RANDOM"
#define METASLOT_TOKEN  "metaslot_token="
#define METASLOT_SLOT   "metaslot_slot="
#define METASLOT_STATUS "metaslot_status="
#define METASLOT_AUTO_KEY_MIGRATE       "metaslot_auto_key_migrate="
#define METASLOT_ENABLED        "enabled"
#define METASLOT_DISABLED       "disabled"
#define SLOT_DESCRIPTION_SIZE   64
#define TOKEN_LABEL_SIZE        32
#define TOKEN_MANUFACTURER_SIZE 32
#define TOKEN_SERIAL_SIZE       16

/*
 * Define the following softtoken values that are used by softtoken
 * library, cryptoadm and pktool command.
 */
#define SOFT_SLOT_DESCRIPTION   \
                        "Sun Crypto Softtoken           " \
                        "                               "
#define SOFT_TOKEN_LABEL        "Sun Software PKCS#11 softtoken  "
#define SOFT_TOKEN_SERIAL       "                "
#define SOFT_MANUFACTURER_ID    "Sun Microsystems, Inc.         "
#define SOFT_DEFAULT_PIN        "changeme"

typedef char libname_t[MAXPATHLEN];
typedef char midstr_t[MECH_ID_HEX_LEN];

typedef struct umechlist {
        midstr_t                name;   /* mechanism name in hex form */
        struct umechlist        *next;
} umechlist_t;
```

unchanged_portion_omitted_

```
extern void cryptodebug(const char *fmt, ...);
extern void cryptoerror(int priority, const char *fmt, ...);
extern void cryptodebug_init(const char *prefix);

extern const char *pkcs11_mech2str(CK_MECHANISM_TYPE mech);
extern char *pkcs11_mech2str(CK_MECHANISM_TYPE mech);
extern CK_RV pkcs11_str2mech(char *mech_str, CK_MECHANISM_TYPE_PTR mech);

extern int get_pkcs11conf_info(uentrylist_t **);
extern umechlist_t *create_umech(char *);
extern void free_umechlist(umechlist_t *);
extern void free_uentrylist(uentrylist_t *);
extern void free_uentry(uentry_t *);
extern uentry_t *getent_uef(char *);

extern void tohexstr(uchar_t *bytes, size_t blen, char *hexstr, size_t hexlen);
extern CK_RV pkcs11_mech2keytype(CK_MECHANISM_TYPE mech_type,
    CK_KEY_TYPE *ktype);
extern CK_RV pkcs11_mech2keygen(CK_MECHANISM_TYPE mech_type,
    CK_MECHANISM_TYPE *gen_mech);
extern char *pkcs11_strerror(CK_RV rv);

extern int
get_metaslot_info(boolean_t  *status_enabled, boolean_t *migrate_enabled,
    char **objectstore_slot_info, char **objectstore_token_info);

extern char *get_fullpath(char *dir, char *filepath);
extern int str2lifetime(char *ltimestr, uint32_t *ltime);

extern char *pkcs11_default_token(void);
```

```
 142 extern int pkcs11_get_pass(char *token_name, char **pdata, size_t *psize,
 143     size_t min_psize, boolean_t with_confirmation);

 145 extern int pkcs11_random_data(void *dbuf, size_t dlen);
 146 extern int pkcs11_nzero_random_data(void *dbuf, size_t dlen);
 147 extern int pkcs11_read_data(char *filename, void **dbuf, size_t *dlen);

 149 #ifdef __cplusplus
 150 }
```

unchanged_portion_omitted_

```
**********************************************************
   15677 Mon Jun 30 15:21:02 2008
new/usr/src/lib/libcryptoutil/common/mechstr.c
5031131 perf: pkcs11_kernel can benefit from a more efficient pkcs11_mech2str()
4947627 improve libcrypto string/mechanism conversion functions in edge cases
**********************************************************
```

```
/*
 * 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.
 */

#pragma ident   "@(#)mechstr.c  1.7     08/06/30 SMI"
#pragma ident   "@(#)mechstr.c  1.6     08/01/07 SMI"

/*
 * Convert Algorithm names as strings to PKCS#11 Mech numbers and vice versa.
 */

#include <limits.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <security/cryptoki.h>
#include <security/pkcs11t.h>

#include <cryptoutil.h>

/*
 * This table is a one-to-one mapping between mechanism names and numbers.
 * As such, it should not contain deprecated mechanism names (aliases).
 */
typedef struct {
static const struct {
        const char              *str;
        CK_MECHANISM_TYPE       mech;
} pkcs11_mapping_t;

/*
 * Note: elements in this table MUST be in numeric order,
 * since bsearch(3C) is used to search this table.
 */
static const pkcs11_mapping_t mapping[] = {
} mapping[] = {
        { "CKM_RSA_PKCS_KEY_PAIR_GEN", CKM_RSA_PKCS_KEY_PAIR_GEN },
        { "CKM_RSA_PKCS", CKM_RSA_PKCS },
        { "CKM_RSA_9796", CKM_RSA_9796 },
```



```
        { "CKM_RSA_X_509", CKM_RSA_X_509 },
        { "CKM_MD2_RSA_PKCS", CKM_MD2_RSA_PKCS },
        { "CKM_MD5_RSA_PKCS", CKM_MD5_RSA_PKCS },
        { "CKM_SHA1_RSA_PKCS", CKM_SHA1_RSA_PKCS },
        { "CKM_RIPEMD128_RSA_PKCS", CKM_RIPEMD128_RSA_PKCS },
        { "CKM_RIPEMD160_RSA_PKCS", CKM_RIPEMD160_RSA_PKCS },
        { "CKM_RSA_PKCS_OAEP", CKM_RSA_PKCS_OAEP },
        { "CKM_RSA_X9_31_KEY_PAIR_GEN", CKM_RSA_X9_31_KEY_PAIR_GEN },
        { "CKM_RSA_X9_31", CKM_RSA_X9_31 },
        { "CKM_SHA1_RSA_X9_31", CKM_SHA1_RSA_X9_31 },
        { "CKM_RSA_PKCS_PSS", CKM_RSA_PKCS_PSS },
        { "CKM_SHA1_RSA_PKCS_PSS", CKM_SHA1_RSA_PKCS_PSS },
        { "CKM_DSA_KEY_PAIR_GEN", CKM_DSA_KEY_PAIR_GEN },
        { "CKM_DSA", CKM_DSA },
        { "CKM_DSA_SHA1", CKM_DSA_SHA1 },
        { "CKM_DH_PKCS_KEY_PAIR_GEN", CKM_DH_PKCS_KEY_PAIR_GEN },
        { "CKM_DH_PKCS_DERIVE", CKM_DH_PKCS_DERIVE },
        { "CKM_X9_42_DH_KEY_PAIR_GEN", CKM_X9_42_DH_KEY_PAIR_GEN },
        { "CKM_X9_42_DH_DERIVE", CKM_X9_42_DH_DERIVE },
        { "CKM_X9_42_DH_HYBRID_DERIVE", CKM_X9_42_DH_HYBRID_DERIVE },
        { "CKM_X9_42_MQV_DERIVE", CKM_X9_42_MQV_DERIVE },
        { "CKM_SHA256_RSA_PKCS", CKM_SHA256_RSA_PKCS },
        { "CKM_SHA384_RSA_PKCS", CKM_SHA384_RSA_PKCS },
        { "CKM_SHA512_RSA_PKCS", CKM_SHA512_RSA_PKCS },
        { "CKM_SHA256_RSA_PKCS_PSS", CKM_SHA256_RSA_PKCS_PSS },
        { "CKM_SHA384_RSA_PKCS_PSS", CKM_SHA384_RSA_PKCS_PSS },
        { "CKM_SHA512_RSA_PKCS_PSS", CKM_SHA512_RSA_PKCS_PSS },
        { "CKM_SHA224_RSA_PKCS", CKM_SHA224_RSA_PKCS },
        { "CKM_SHA224_RSA_PKCS_PSS", CKM_SHA224_RSA_PKCS_PSS },
        { "CKM_RC2_KEY_GEN", CKM_RC2_KEY_GEN },
        { "CKM_RC2_ECB", CKM_RC2_ECB },
        { "CKM_RC2_CBC", CKM_RC2_CBC },
        { "CKM_RC2_MAC", CKM_RC2_MAC },
        { "CKM_RC2_MAC_GENERAL", CKM_RC2_MAC_GENERAL },
        { "CKM_RC2_CBC_PAD", CKM_RC2_CBC_PAD },
        { "CKM_RC4_KEY_GEN", CKM_RC4_KEY_GEN },
        { "CKM_RC4", CKM_RC4 },
        { "CKM_DES_KEY_GEN", CKM_DES_KEY_GEN },
        { "CKM_DES_ECB", CKM_DES_ECB },
        { "CKM_DES_CBC", CKM_DES_CBC },
        { "CKM_DES_MAC", CKM_DES_MAC },
        { "CKM_DES_MAC_GENERAL", CKM_DES_MAC_GENERAL },
        { "CKM_DES_CBC_PAD", CKM_DES_CBC_PAD },
        { "CKM_DES2_KEY_GEN", CKM_DES2_KEY_GEN },
        { "CKM_DES3_KEY_GEN", CKM_DES3_KEY_GEN },
        { "CKM_DES3_ECB", CKM_DES3_ECB },
        { "CKM_DES3_CBC", CKM_DES3_CBC },
        { "CKM_DES3_MAC", CKM_DES3_MAC },
        { "CKM_DES3_MAC_GENERAL", CKM_DES3_MAC_GENERAL },
        { "CKM_DES3_CBC_PAD", CKM_DES3_CBC_PAD },
        { "CKM_CDMF_KEY_GEN", CKM_CDMF_KEY_GEN },
        { "CKM_CDMF_ECB", CKM_CDMF_ECB },
        { "CKM_CDMF_CBC", CKM_CDMF_CBC },
        { "CKM_CDMF_MAC", CKM_CDMF_MAC },
        { "CKM_CDMF_MAC_GENERAL", CKM_CDMF_MAC_GENERAL },
        { "CKM_CDMF_CBC_PAD", CKM_CDMF_CBC_PAD },
        { "CKM_DES_OFB64", CKM_DES_OFB64 },
        { "CKM_DES_OFB8", CKM_DES_OFB8 },
        { "CKM_DES_CFB64", CKM_DES_CFB64 },
        { "CKM_DES_CFB8", CKM_DES_CFB8 },
        { "CKM_MD2", CKM_MD2 },
        { "CKM_MD2_HMAC", CKM_MD2_HMAC },
        { "CKM_MD2_HMAC_GENERAL", CKM_MD2_HMAC_GENERAL },
        { "CKM_MD5", CKM_MD5 },
        { "CKM_MD5_HMAC", CKM_MD5_HMAC },
        { "CKM_MD5_HMAC_GENERAL", CKM_MD5_HMAC_GENERAL },
```

```
124         { "CKM_SHA_1", CKM_SHA_1 },
125         { "CKM_SHA_1_HMAC", CKM_SHA_1_HMAC },
126         { "CKM_SHA_1_HMAC_GENERAL", CKM_SHA_1_HMAC_GENERAL },
127         { "CKM_RIPEMD128", CKM_RIPEMD128 },
128         { "CKM_RIPEMD128_HMAC", CKM_RIPEMD128_HMAC },
129         { "CKM_RIPEMD128_HMAC_GENERAL", CKM_RIPEMD128_HMAC_GENERAL },
130         { "CKM_RIPEMD160", CKM_RIPEMD160 },
131         { "CKM_RIPEMD160_HMAC", CKM_RIPEMD160_HMAC },
132         { "CKM_RIPEMD160_HMAC_GENERAL", CKM_RIPEMD160_HMAC_GENERAL },
133         { "CKM_SHA256", CKM_SHA256 },
134         { "CKM_SHA256_HMAC", CKM_SHA256_HMAC },
135         { "CKM_SHA256_HMAC_GENERAL", CKM_SHA256_HMAC_GENERAL },
136         { "CKM_SHA224", CKM_SHA224 },
137         { "CKM_SHA224_HMAC", CKM_SHA224_HMAC },
138         { "CKM_SHA224_HMAC_GENERAL", CKM_SHA224_HMAC_GENERAL },
139         { "CKM_SHA384", CKM_SHA384 },
140         { "CKM_SHA384_HMAC", CKM_SHA384_HMAC },
141         { "CKM_SHA384_HMAC_GENERAL", CKM_SHA384_HMAC_GENERAL },
142         { "CKM_SHA512", CKM_SHA512 },
143         { "CKM_SHA512_HMAC", CKM_SHA512_HMAC },
144         { "CKM_SHA512_HMAC_GENERAL", CKM_SHA512_HMAC_GENERAL },
145         { "CKM_SECURID_KEY_GEN", CKM_SECURID_KEY_GEN },
146         { "CKM_SECURID", CKM_SECURID },
147         { "CKM_HOTP_KEY_GEN", CKM_HOTP_KEY_GEN },
148         { "CKM_HOTP", CKM_HOTP },
149         { "CKM_ACTI", CKM_ACTI },
150         { "CKM_ACTI_KEY_GEN", CKM_ACTI_KEY_GEN },
151         { "CKM_CAST_KEY_GEN", CKM_CAST_KEY_GEN },
152         { "CKM_CAST_ECB", CKM_CAST_ECB },
153         { "CKM_CAST_CBC", CKM_CAST_CBC },
154         { "CKM_CAST_MAC", CKM_CAST_MAC },
155         { "CKM_CAST_MAC_GENERAL", CKM_CAST_MAC_GENERAL },
156         { "CKM_CAST_CBC_PAD", CKM_CAST_CBC_PAD },
157         { "CKM_CAST3_KEY_GEN", CKM_CAST3_KEY_GEN },
158         { "CKM_CAST3_ECB", CKM_CAST3_ECB },
159         { "CKM_CAST3_CBC", CKM_CAST3_CBC },
160         { "CKM_CAST3_MAC", CKM_CAST3_MAC },
161         { "CKM_CAST3_MAC_GENERAL", CKM_CAST3_MAC_GENERAL },
162         { "CKM_CAST3_CBC_PAD", CKM_CAST3_CBC_PAD },
163         { "CKM_CAST5_KEY_GEN", CKM_CAST5_KEY_GEN },
164         { "CKM_CAST128_KEY_GEN", CKM_CAST128_KEY_GEN },
165         { "CKM_CAST5_ECB", CKM_CAST5_ECB },
166         { "CKM_CAST128_ECB", CKM_CAST128_ECB },
167         { "CKM_CAST5_CBC", CKM_CAST5_CBC },
168         { "CKM_CAST128_CBC", CKM_CAST128_CBC },
169         { "CKM_CAST5_MAC", CKM_CAST5_MAC },
170         { "CKM_CAST128_MAC", CKM_CAST128_MAC },
171         { "CKM_CAST5_MAC_GENERAL", CKM_CAST5_MAC_GENERAL },
172         { "CKM_CAST128_MAC_GENERAL", CKM_CAST128_MAC_GENERAL },
173         { "CKM_CAST5_CBC_PAD", CKM_CAST5_CBC_PAD },
174         { "CKM_CAST128_CBC_PAD", CKM_CAST128_CBC_PAD },
175         { "CKM_RC5_KEY_GEN", CKM_RC5_KEY_GEN },
176         { "CKM_RC5_ECB", CKM_RC5_ECB },
177         { "CKM_RC5_CBC", CKM_RC5_CBC },
178         { "CKM_RC5_MAC", CKM_RC5_MAC },
179         { "CKM_RC5_MAC_GENERAL", CKM_RC5_MAC_GENERAL },
180         { "CKM_RC5_CBC_PAD", CKM_RC5_CBC_PAD },
181         { "CKM_IDEA_KEY_GEN", CKM_IDEA_KEY_GEN },
182         { "CKM_IDEA_ECB", CKM_IDEA_ECB },
183         { "CKM_IDEA_CBC", CKM_IDEA_CBC },
184         { "CKM_IDEA_MAC", CKM_IDEA_MAC },
185         { "CKM_IDEA_MAC_GENERAL", CKM_IDEA_MAC_GENERAL },
186         { "CKM_IDEA_CBC_PAD", CKM_IDEA_CBC_PAD },
187         { "CKM_GENERIC_SECRET_KEY_GEN", CKM_GENERIC_SECRET_KEY_GEN },
188         { "CKM_CONCATENATE_BASE_AND_KEY", CKM_CONCATENATE_BASE_AND_KEY },
189         { "CKM_CONCATENATE_BASE_AND_DATA", CKM_CONCATENATE_BASE_AND_DATA },
```



```
190         { "CKM_CONCATENATE_DATA_AND_BASE", CKM_CONCATENATE_DATA_AND_BASE },
191         { "CKM_XOR_BASE_AND_DATA", CKM_XOR_BASE_AND_DATA },
192         { "CKM_EXTRACT_KEY_FROM_KEY", CKM_EXTRACT_KEY_FROM_KEY },
193         { "CKM_SSL3_PRE_MASTER_KEY_GEN", CKM_SSL3_PRE_MASTER_KEY_GEN },
194         { "CKM_SSL3_MASTER_KEY_DERIVE", CKM_SSL3_MASTER_KEY_DERIVE },
195         { "CKM_SSL3_KEY_AND_MAC_DERIVE", CKM_SSL3_KEY_AND_MAC_DERIVE },
196         { "CKM_SSL3_MASTER_KEY_DERIVE_DH", CKM_SSL3_MASTER_KEY_DERIVE_DH },
197         { "CKM_TLS_PRE_MASTER_KEY_GEN", CKM_TLS_PRE_MASTER_KEY_GEN },
198         { "CKM_TLS_MASTER_KEY_DERIVE", CKM_TLS_MASTER_KEY_DERIVE },
199         { "CKM_TLS_KEY_AND_MAC_DERIVE", CKM_TLS_KEY_AND_MAC_DERIVE },
200         { "CKM_TLS_MASTER_KEY_DERIVE_DH", CKM_TLS_MASTER_KEY_DERIVE_DH },
201         { "CKM_TLS_PRF", CKM_TLS_PRF },
202         { "CKM_SSL3_MD5_MAC", CKM_SSL3_MD5_MAC },
203         { "CKM_SSL3_SHA1_MAC", CKM_SSL3_SHA1_MAC },
204         { "CKM_MD5_KEY_DERIVATION", CKM_MD5_KEY_DERIVATION },
205         { "CKM_MD2_KEY_DERIVATION", CKM_MD2_KEY_DERIVATION },
206         { "CKM_SHA1_KEY_DERIVATION", CKM_SHA1_KEY_DERIVATION },
207         { "CKM_SHA256_KEY_DERIVATION", CKM_SHA256_KEY_DERIVATION },
208         { "CKM_SHA384_KEY_DERIVATION", CKM_SHA384_KEY_DERIVATION },
209         { "CKM_SHA512_KEY_DERIVATION", CKM_SHA512_KEY_DERIVATION },
210         { "CKM_SHA224_KEY_DERIVATION", CKM_SHA224_KEY_DERIVATION },
211         { "CKM_PBE_MD2_DES_CBC", CKM_PBE_MD2_DES_CBC },
212         { "CKM_PBE_MD5_DES_CBC", CKM_PBE_MD5_DES_CBC },
213         { "CKM_PBE_MD5_CAST_CBC", CKM_PBE_MD5_CAST_CBC },
214         { "CKM_PBE_MD5_CAST3_CBC", CKM_PBE_MD5_CAST3_CBC },
215         { "CKM_PBE_MD5_CAST5_CBC", CKM_PBE_MD5_CAST5_CBC },
216         { "CKM_PBE_MD5_CAST128_CBC", CKM_PBE_MD5_CAST128_CBC },
217         { "CKM_PBE_SHA1_CAST5_CBC", CKM_PBE_SHA1_CAST5_CBC },
218         { "CKM_PBE_SHA1_CAST128_CBC", CKM_PBE_SHA1_CAST128_CBC },
219         { "CKM_PBE_SHA1_RC4_128", CKM_PBE_SHA1_RC4_128 },
220         { "CKM_PBE_SHA1_RC4_40", CKM_PBE_SHA1_RC4_40 },
221         { "CKM_PBE_SHA1_DES3_EDE_CBC", CKM_PBE_SHA1_DES3_EDE_CBC },
222         { "CKM_PBE_SHA1_DES2_EDE_CBC", CKM_PBE_SHA1_DES2_EDE_CBC },
223         { "CKM_PBE_SHA1_RC2_128_CBC", CKM_PBE_SHA1_RC2_128_CBC },
224         { "CKM_PBE_SHA1_RC2_40_CBC", CKM_PBE_SHA1_RC2_40_CBC },
225         { "CKM_PKCS5_PBKD2", CKM_PKCS5_PBKD2 },
226         { "CKM_PBA_SHA1_WITH_SHA1_HMAC", CKM_PBA_SHA1_WITH_SHA1_HMAC },
227         { "CKM_KEY_WRAP_LYNKS", CKM_KEY_WRAP_LYNKS },
228         { "CKM_KEY_WRAP_SET_OAEP", CKM_KEY_WRAP_SET_OAEP },
229         { "CKM_KIP_DERIVE", CKM_KIP_DERIVE },
230         { "CKM_KIP_WRAP", CKM_KIP_WRAP },
231         { "CKM_KIP_MAC", CKM_KIP_MAC },
232         { "CKM_CAMELLIA_KEY_GEN", CKM_CAMELLIA_KEY_GEN },
233         { "CKM_CAMELLIA_ECB", CKM_CAMELLIA_ECB },
234         { "CKM_CAMELLIA_CBC", CKM_CAMELLIA_CBC },
235         { "CKM_CAMELLIA_MAC", CKM_CAMELLIA_MAC },
236         { "CKM_CAMELLIA_MAC_GENERAL", CKM_CAMELLIA_MAC_GENERAL },
237         { "CKM_CAMELLIA_CBC_PAD", CKM_CAMELLIA_CBC_PAD },
238         { "CKM_CAMELLIA_ECB_ENCRYPT_DATA", CKM_CAMELLIA_ECB_ENCRYPT_DATA },
239         { "CKM_CAMELLIA_CBC_ENCRYPT_DATA", CKM_CAMELLIA_CBC_ENCRYPT_DATA },
240         { "CKM_CAMELLIA_CTR", CKM_CAMELLIA_CTR },
241         { "CKM_ARIA_KEY_GEN", CKM_ARIA_KEY_GEN },
242         { "CKM_ARIA_ECB", CKM_ARIA_ECB },
243         { "CKM_ARIA_CBC", CKM_ARIA_CBC },
244         { "CKM_ARIA_MAC", CKM_ARIA_MAC },
245         { "CKM_ARIA_MAC_GENERAL", CKM_ARIA_MAC_GENERAL },
246         { "CKM_ARIA_CBC_PAD", CKM_ARIA_CBC_PAD },
247         { "CKM_ARIA_ECB_ENCRYPT_DATA", CKM_ARIA_ECB_ENCRYPT_DATA },
248         { "CKM_ARIA_CBC_ENCRYPT_DATA", CKM_ARIA_CBC_ENCRYPT_DATA },
249         { "CKM_SKIPJACK_KEY_GEN", CKM_SKIPJACK_KEY_GEN },
250         { "CKM_SKIPJACK_ECB64", CKM_SKIPJACK_ECB64 },
251         { "CKM_SKIPJACK_CBC64", CKM_SKIPJACK_CBC64 },
252         { "CKM_SKIPJACK_OFB64", CKM_SKIPJACK_OFB64 },
253         { "CKM_SKIPJACK_CFB64", CKM_SKIPJACK_CFB64 },
254         { "CKM_SKIPJACK_CFB32", CKM_SKIPJACK_CFB32 },
255         { "CKM_SKIPJACK_CFB16", CKM_SKIPJACK_CFB16 },
```

```
256         { "CKM_SKIPJACK_CFB8", CKM_SKIPJACK_CFB8 },
257         { "CKM_SKIPJACK_WRAP", CKM_SKIPJACK_WRAP },
258         { "CKM_SKIPJACK_PRIVATE_WRAP", CKM_SKIPJACK_PRIVATE_WRAP },
259         { "CKM_SKIPJACK_RELAYX", CKM_SKIPJACK_RELAYX },
260         { "CKM_KEA_KEY_PAIR_GEN", CKM_KEA_KEY_PAIR_GEN },
261         { "CKM_KEA_KEY_DERIVE", CKM_KEA_KEY_DERIVE },
262         { "CKM_FORTEZZA_TIMESTAMP", CKM_FORTEZZA_TIMESTAMP },
263         { "CKM_BATON_KEY_GEN", CKM_BATON_KEY_GEN },
264         { "CKM_BATON_ECB128", CKM_BATON_ECB128 },
265         { "CKM_BATON_ECB96", CKM_BATON_ECB96 },
266         { "CKM_BATON_CBC128", CKM_BATON_CBC128 },
267         { "CKM_BATON_COUNTER", CKM_BATON_COUNTER },
268         { "CKM_BATON_SHUFFLE", CKM_BATON_SHUFFLE },
269         { "CKM_BATON_WRAP", CKM_BATON_WRAP },
270         { "CKM_EC_KEY_PAIR_GEN", CKM_EC_KEY_PAIR_GEN },
271         { "CKM_ECDSA", CKM_ECDSA },
272         { "CKM_ECDSA_SHA1", CKM_ECDSA_SHA1 },
273         { "CKM_ECDH1_DERIVE", CKM_ECDH1_DERIVE },
274         { "CKM_ECDH1_COFACTOR_DERIVE", CKM_ECDH1_COFACTOR_DERIVE },
275         { "CKM_ECMQV_DERIVE", CKM_ECMQV_DERIVE },
276         { "CKM_JUNIPER_KEY_GEN", CKM_JUNIPER_KEY_GEN },
277         { "CKM_JUNIPER_ECB128", CKM_JUNIPER_ECB128 },
278         { "CKM_JUNIPER_CBC128", CKM_JUNIPER_CBC128 },
279         { "CKM_JUNIPER_COUNTER", CKM_JUNIPER_COUNTER },
280         { "CKM_JUNIPER_SHUFFLE", CKM_JUNIPER_SHUFFLE },
281         { "CKM_JUNIPER_WRAP", CKM_JUNIPER_WRAP },
282         { "CKM_FASTHASH", CKM_FASTHASH },
283         { "CKM_AES_KEY_GEN", CKM_AES_KEY_GEN },
284         { "CKM_AES_ECB", CKM_AES_ECB },
285         { "CKM_AES_CBC", CKM_AES_CBC },
286         { "CKM_AES_MAC", CKM_AES_MAC },
287         { "CKM_AES_MAC_GENERAL", CKM_AES_MAC_GENERAL },
288         { "CKM_AES_CBC_PAD", CKM_AES_CBC_PAD },
289         { "CKM_AES_CTR", CKM_AES_CTR },
290         { "CKM_BLOWFISH_KEY_GEN", CKM_BLOWFISH_KEY_GEN },
291         { "CKM_BLOWFISH_CBC", CKM_BLOWFISH_CBC },
292         { "CKM_TWOFISH_KEY_GEN", CKM_TWOFISH_KEY_GEN },
293         { "CKM_TWOFISH_CBC", CKM_TWOFISH_CBC },
294         { "CKM_DES_ECB_ENCRYPT_DATA", CKM_DES_ECB_ENCRYPT_DATA },
295         { "CKM_DES_CBC_ENCRYPT_DATA", CKM_DES_CBC_ENCRYPT_DATA },
296         { "CKM_DES3_ECB_ENCRYPT_DATA", CKM_DES3_ECB_ENCRYPT_DATA },
297         { "CKM_DES3_CBC_ENCRYPT_DATA", CKM_DES3_CBC_ENCRYPT_DATA },
298         { "CKM_AES_ECB_ENCRYPT_DATA", CKM_AES_ECB_ENCRYPT_DATA },
299         { "CKM_AES_CBC_ENCRYPT_DATA", CKM_AES_CBC_ENCRYPT_DATA },
300         { "CKM_DSA_PARAMETER_GEN", CKM_DSA_PARAMETER_GEN },
301         { "CKM_DH_PKCS_PARAMETER_GEN", CKM_DH_PKCS_PARAMETER_GEN },
302         { "CKM_X9_42_DH_PARAMETER_GEN", CKM_X9_42_DH_PARAMETER_GEN },
303         /*
304          * Values above 0x8000000 (CKM_VENDOR_DEFINED) are represented
305          * as strings with hexadecimal numbers (e.g., "0x8123456").
306          */
296         { "CKM_VENDOR_DEFINED", CKM_VENDOR_DEFINED },
307         { NULL, 0 }
308 };


311 /*
312  * pkcs11_mech_comp - compare two pkcs11_mapping_t structures
313  *
314  * Return a strcmp-like result (positive, zero, or negative).
315  * For use with bsearch(3C) in pkcs11_mech2str().
316  */
317 static int
318 pkcs11_mech_comp(const void *mapping1, const void *mapping2) {
319         return (((pkcs11_mapping_t *)mapping1)->mech -
320                 ((pkcs11_mapping_t *)mapping2)->mech);
```



```
321 }


324 /*
325  * pkcs11_mech2str - convert PKCS#11 mech to a string
326  *
327  * Anything below CKM_VENDOR_DEFINED that wasn't in the mapping table
328  * at build time causes NULL to be returned.  Anything above it also
329  * returns NULL since we have no way to know its real name.
305  * returns NULL since we have no way to know what its real name is.
330  */
331 const char
307 char
332 *pkcs11_mech2str(CK_MECHANISM_TYPE mech)
333 {
334         pkcs11_mapping_t        target;
335         pkcs11_mapping_t        *result = NULL;
310         int i;
311         char buf[11];   /* Num chars for representing ulong in ASCII */

337         if (mech > CKM_VENDOR_DEFINED) {
338                 return (NULL);
314                 (void) snprintf(buf, sizeof (buf), "%#lx", mech);
315                 return (strdup(buf));
339         }

341         /* Search for the mechanism number using bsearch(3C) */
342         target.mech = mech;
343         target.str = NULL;
344         result = (pkcs11_mapping_t *)bsearch((void *)&target, (void *)mapping,
345             (sizeof (mapping) / sizeof (pkcs11_mapping_t)) - 1,
346             sizeof (pkcs11_mapping_t), pkcs11_mech_comp);
347         if (result != NULL) {
348                 return (result->str);
318         for (i = 0; mapping[i].str; i++) {
319                 if (mapping[i].mech == mech)
320                         return (strdup(mapping[i].str));
349         }

351         return (NULL);
352 }

354 /*
355  * pkcs11_str2mech - convert a string into a PKCS#11 mech number.
356  *
357  * Since there isn't a reserved value for an invalid mech we return
329  * Since there isn't reserved value for an invalid mech we return
358  * CKR_MECHANISM_INVALID for anything we don't recognise.
359  * The value in mech isn't meaningful in these cases.
360  */
361 CK_RV
362 pkcs11_str2mech(char *mech_str, CK_MECHANISM_TYPE_PTR mech)
363 {
364         int     i;
365         int     compare_off = 0;
337         char *tmech_str;

367         if (mech_str == NULL)
368                 return (CKR_MECHANISM_INVALID);

370         if (strncasecmp(mech_str, "0x", 2) == 0) {
371                 long long llnum;
342         if (strncasecmp(mech_str, "0x8", 3) == 0) {
372                 cryptodebug("pkcs11_str2mech: hex string passed in: %s",
373                     mech_str);
374                 llnum = strtoll(mech_str, NULL, 16);
```

```
375                 if ((llnum >= CKM_VENDOR_DEFINED) && (llnum <= UINT_MAX)) {
376                         *mech = llnum;
345                 *mech = strtoll(mech_str, NULL, 16);
377                         return (CKR_OK);
378                 } else {
379                         return (CKR_MECHANISM_INVALID);
380                 }
381         }

383         /* If there's no CKM_ prefix, then ignore it in comparisons */
384         if (strncasecmp(mech_str, "CKM_", 4) != 0) {
350                 size_t tmech_strlen = strlen(mech_str) + 4 + 1;
385                 cryptodebug("pkcs11_str2mech: no CKM_ prefix: %s", mech_str);
386                 cryptodebug("pkcs11_str2mech: with prefix: CKM_%s", mech_str);
387                 compare_off = 4;
352                 tmech_str = malloc(tmech_strlen * sizeof (char));
353                 (void) snprintf(tmech_str, tmech_strlen, "CKM_%s", mech_str);
354                 cryptodebug("pkcs11_str2mech: with prefix: %s", tmech_str);
355         } else {
356                 tmech_str = mech_str;
388         }

390         /* Linear search for a matching string */
391         for (i = 0; mapping[i].str; i++) {
392                 if (strcasecmp(&mapping[i].str[compare_off], mech_str) == 0) {
360                 if (strcasecmp(mapping[i].str, tmech_str) == 0) {
393                         *mech = mapping[i].mech;
362                         if (tmech_str != mech_str)
363                                 free(tmech_str);
394                         return (CKR_OK);
395                 }
396         }
367         if (tmech_str != mech_str)
368                 free(tmech_str);

398         return (CKR_MECHANISM_INVALID);
399 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   35884 Mon Jun 30 15:21:05 2008
new/usr/src/lib/pkcs11/pkcs11_kernel/common/kernelUtil.c
5031131 perf: pkcs11_kernel can benefit from a more efficient pkcs11_mech2str()
**********************************************************
```

```
/*
 * 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 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "@(#)kernelUtil.c       1.17    08/06/30 SMI"
#pragma ident   "@(#)kernelUtil.c       1.16    07/09/11 SMI"

#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <stdio.h>
#include <cryptoutil.h>
#include <errno.h>
#include <security/cryptoki.h>
#include <sys/crypto/common.h>
#include <sys/crypto/ioctl.h>
#include "kernelGlobal.h"
#include "kernelObject.h"
#include "kernelSlot.h"

#define ENCODE_ATTR(type, value, len) {         \
        cur_attr->oa_type = type;               \
        (void) memcpy(ptr, value, len);         \
        cur_attr->oa_value = ptr;               \
        cur_attr->oa_value_len = len;           \
        cur_attr++;                             \
}
```

*_____unchanged_portion_omitted_*

```
CK_RV
kernel_mech(CK_MECHANISM_TYPE type, crypto_mech_type_t *k_number)
{
        crypto_get_mechanism_number_t get_number;
        const char *string;
        char *string;
        CK_RV rv;
        int r;
        kmh_elem_t *elem;
        uint_t h;
```



```
        char buf[11];   /* Num chars for representing ulong in ASCII */

        /*
         * Search for an existing entry. No need to lock since we are
         * just a reader and we never free the entries in the hash table.
         */
        h = MECH_HASH(type);
        for (elem = kernel_mechhash[h]; elem != NULL; elem = elem->knext) {
                if (type == elem->type) {
                        *k_number = elem->kmech;
                        return (CKR_OK);
                }
        }

        if (type > CKM_VENDOR_DEFINED) {
                (void) snprintf(buf, sizeof (buf), "%#lx", type);
                string = buf;
        } else {
                string = pkcs11_mech2str(type);
        }

        if (string == NULL)
                return (CKR_MECHANISM_INVALID);

        get_number.pn_mechanism_string = (char *)string;
        get_number.pn_mechanism_string = string;
        get_number.pn_mechanism_len = strlen(string) + 1;

        while ((r = ioctl(kernel_fd, CRYPTO_GET_MECHANISM_NUMBER,
            &get_number)) < 0) {
                if (errno != EINTR)
                        break;
        }
        if (r < 0) {
                rv = CKR_MECHANISM_INVALID;
        } else {
                if (get_number.pn_return_value != CRYPTO_SUCCESS) {
                        rv = crypto2pkcs11_error_number(
                            get_number.pn_return_value);
                } else {
                        rv = CKR_OK;
                }
        }

        if (rv == CKR_OK) {
                *k_number = get_number.pn_internal_number;
                /* Add this to the hash table */
                (void) kmech_hash_insert(type, *k_number);
        }

        free(string);
        return (rv);
}
```

*_____unchanged_portion_omitted_*

```
/*
 * Get the value of the CKA_PRIVATE attribute for the object just returned
 * from the HW provider.  This function will be called by any function
 * that creates a new object, because the CKA_PRIVATE value of an object is
 * token specific.  The CKA_PRIVATE attribute value of the new object will be
 * token sepecific.  The CKA_PRIVATE attribute value of the new object will be
 * stored in the object structure in the library, which will be used later at
 * C_Logout to clean up all private objects.
 */
CK_RV
```

```
1189 get_cka_private_value(kernel_session_t *sp, crypto_object_id_t oid,
1190     CK_BBOOL *is_pri_obj)
1191 {
1192         CK_RV  rv = CKR_OK;
1193         crypto_object_get_attribute_value_t obj_ga;
1194         crypto_object_attribute_t obj_attr;
1195         CK_BBOOL pri_value;
1196         int r;

1198         obj_ga.og_session = sp->k_session;
1199         obj_ga.og_handle = oid;
1200         obj_ga.og_count = 1;

1202         obj_attr.oa_type = CKA_PRIVATE;
1203         obj_attr.oa_value = (char *)&pri_value;
1204         obj_attr.oa_value_len = sizeof (CK_BBOOL);
1205         obj_ga.og_attributes = (char *)&obj_attr;

1207         while ((r = ioctl(kernel_fd, CRYPTO_OBJECT_GET_ATTRIBUTE_VALUE,
1208             &obj_ga)) < 0) {
1209                 if (errno != EINTR)
1210                         break;
1211         }
1212         if (r < 0) {
1213                 rv = CKR_FUNCTION_FAILED;
1214         } else {
1215                 rv = crypto2pkcs11_error_number(obj_ga.og_return_value);
1216         }

1218         if (rv == CKR_OK) {
1219                 *is_pri_obj = *(CK_BBOOL *)obj_attr.oa_value;
1220         }

1222         return (rv);
1223 }


1226 CK_RV
1227 get_mechanism_info(kernel_slot_t *pslot, CK_MECHANISM_TYPE type,
1228     CK_MECHANISM_INFO_PTR pInfo, uint32_t *k_mi_flags)
1229 {
1230         crypto_get_provider_mechanism_info_t mechanism_info;
1231         const char *string;
1224         char *string;
1232         CK_FLAGS flags, mi_flags;
1233         CK_RV rv;
1234         int r;
1235         char buf[11];   /* Num chars for representing ulong in ASCII */

1237         if (type > CKM_VENDOR_DEFINED) {
1238                 /* allocate/build a string containing the mechanism number */
1239                 (void) snprintf(buf, sizeof (buf), "%#lx", type);
1240                 string = buf;
1241         } else {
1242                 string = pkcs11_mech2str(type);
1243         }

1245         if (string == NULL)
1246                 return (CKR_MECHANISM_INVALID);

1248         (void) strcpy(mechanism_info.mi_mechanism_name, string);
1249         mechanism_info.mi_provider_id = pslot->sl_provider_id;

1251         while ((r = ioctl(kernel_fd, CRYPTO_GET_PROVIDER_MECHANISM_INFO,
1252             &mechanism_info)) < 0) {
1253                 if (errno != EINTR)
```



```
1254                         break;
1255         }
1256         if (r < 0) {
1257                 rv = CKR_FUNCTION_FAILED;
1258         } else {
1259                 rv = crypto2pkcs11_error_number(
1260                     mechanism_info.mi_return_value);
1261         }

1263         if (rv != CKR_OK) {
1264                 return (rv);
1249                 goto out;
1265         }

1267         /*
1268          * Atomic flags are not part of PKCS#11 so we filter
1269          * them out here.
1270          */
1271         mi_flags = mechanism_info.mi_flags;
1272         mi_flags &= ~(CRYPTO_FG_DIGEST_ATOMIC | CRYPTO_FG_ENCRYPT_ATOMIC |
1273             CRYPTO_FG_DECRYPT_ATOMIC | CRYPTO_FG_MAC_ATOMIC |
1274             CRYPTO_FG_SIGN_ATOMIC | CRYPTO_FG_VERIFY_ATOMIC |
1275             CRYPTO_FG_SIGN_RECOVER_ATOMIC |
1276             CRYPTO_FG_VERIFY_RECOVER_ATOMIC |
1277             CRYPTO_FG_ENCRYPT_MAC_ATOMIC |
1278             CRYPTO_FG_MAC_DECRYPT_ATOMIC);

1280         if (mi_flags == 0) {
1281                 return (CKR_MECHANISM_INVALID);
1266                 rv = CKR_MECHANISM_INVALID;
1267                 goto out;
1282         }

1284         if (rv == CKR_OK) {
1285                 /* set the value of k_mi_flags first */
1286                 *k_mi_flags = mi_flags;

1288                 /* convert KEF flags into pkcs11 flags */
1289                 flags = CKF_HW;
1290                 if (mi_flags & CRYPTO_FG_ENCRYPT)
1291                         flags |= CKF_ENCRYPT;
1292                 if (mi_flags & CRYPTO_FG_DECRYPT) {
1293                         flags |= CKF_DECRYPT;
1294                         /*
1295                          * Since we'll be emulating C_UnwrapKey() for some
1296                          * cases, we can go ahead and claim CKF_UNWRAP
1297                          */
1298                         flags |= CKF_UNWRAP;
1299                 }
1300                 if (mi_flags & CRYPTO_FG_DIGEST)
1301                         flags |= CKF_DIGEST;
1302                 if (mi_flags & CRYPTO_FG_SIGN)
1303                         flags |= CKF_SIGN;
1304                 if (mi_flags & CRYPTO_FG_SIGN_RECOVER)
1305                         flags |= CKF_SIGN_RECOVER;
1306                 if (mi_flags & CRYPTO_FG_VERIFY)
1307                         flags |= CKF_VERIFY;
1308                 if (mi_flags & CRYPTO_FG_VERIFY_RECOVER)
1309                         flags |= CKF_VERIFY_RECOVER;
1310                 if (mi_flags & CRYPTO_FG_GENERATE)
1311                         flags |= CKF_GENERATE;
1312                 if (mi_flags & CRYPTO_FG_GENERATE_KEY_PAIR)
1313                         flags |= CKF_GENERATE_KEY_PAIR;
1314                 if (mi_flags & CRYPTO_FG_WRAP)
1315                         flags |= CKF_WRAP;
1316                 if (mi_flags & CRYPTO_FG_UNWRAP)
```

```
1317                        flags |= CKF_UNWRAP;
1318                if (mi_flags & CRYPTO_FG_DERIVE)
1319                        flags |= CKF_DERIVE;

1321                pInfo->ulMinKeySize = mechanism_info.mi_min_key_size;
1322                pInfo->ulMaxKeySize = mechanism_info.mi_max_key_size;
1323                pInfo->flags = flags;

1325        }

1313 out:
1314        free(string);
1327        return (rv);
1328 }
```

_____unchanged_portion_omitted_