

```

*****
34920 Tue Oct 28 16:45:20 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_kef.c
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #include <fcntl.h>
27 #include <stdio.h>
28 #include <stdlib.h>
29 #include <strings.h>
30 #include <unistd.h>
31 #include <locale.h>
32 #include <libgen.h>
33 #include <sys/types.h>
34 #include <sys/stat.h>
35 #include <sys/crypto/ioctladmin.h>
36 #include <signal.h>
37 #include <sys/crypto/elfsign.h>
38 #include "cryptoadm.h"

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

42 /*
43  * Display the mechanism list for a kernel software provider.
44  * This implements part of the "cryptoadm list -m" command.
45  *
46  * Parameters phardlist and psoftlist are supplied by get_kcfconf_info().
47  * If NULL, this function obtains it by calling get_kcfconf_info() internally.
48  */
49 int
50 list_mechlist_for_soft(char *provname,
51 entrylist_t *phardlist, entrylist_t *psoftlist)
47 list_mechlist_for_soft(char *provname)
52 {
53     mechlist_t *pmechlist = NULL;
49 mechlist_t *pmechlist;
54     int rc;

56     if (provname == NULL) {
57         return (FAILURE);
58     }

```

```

60     rc = get_soft_info(provname, &pmechlist, phardlist, psoftlist);
56     rc = get_soft_info(provname, &pmechlist);
61     if (rc == SUCCESS) {
62         (void) filter_mechlist(&pmechlist, RANDOM);
63         print_mechlist(provname, pmechlist);
64         free_mechlist(pmechlist);
65     } else {
66         cryptoerror(LOG_STDERR, gettext(
67             "failed to retrieve the mechanism list for %s."),
68             provname);
69     }
71     return (rc);
72 }

74 /*
75  * Display the mechanism list for a kernel hardware provider.
76  * This implements part of the "cryptoadm list -m" command.
77  */
78 int
79 list_mechlist_for_hard(char *provname)
80 {
81     mechlist_t *pmechlist = NULL;
77 mechlist_t *pmechlist;
82     char devname[MAXNAMELEN];
83     int inst_num;
84     int count;
85     int rc = SUCCESS;

87     if (provname == NULL) {
88         return (FAILURE);
89     }

91     /*
92      * Check if the provider is valid. If it is valid, get the number of
93      * mechanisms also.
94      */
95     if (check_hardware_provider(provname, devname, &inst_num, &count) ==
96         FAILURE) {
97         return (FAILURE);
98     }

100     /* Get the mechanism list for the kernel hardware provider */
101     if ((rc = get_dev_info(devname, inst_num, count, &pmechlist)) ==
102         SUCCESS) {
103         (void) filter_mechlist(&pmechlist, RANDOM);
104         print_mechlist(provname, pmechlist);
105         free_mechlist(pmechlist);
106     }

108     return (rc);
109 }

112 /*
113  * Display the policy information for a kernel software provider.
114  * This implements part of the "cryptoadm list -p" command.
115  *
116  * Parameters phardlist and psoftlist are supplied by get_kcfconf_info().
117  * If NULL, this function obtains it by calling get_kcfconf_info() internally.
118  */
119 int
120 list_policy_for_soft(char *provname,
121 entrylist_t *phardlist, entrylist_t *psoftlist)

```

```

112 list_policy_for_soft(char *provname)
122 {
123     int          rc;
124     entry_t      *pent = NULL;
125     mechlist_t   *pmechlist = NULL;
116     mechlist_t *pmechlist;
126     boolean_t    has_random = B_FALSE;
127     boolean_t    has_mechs = B_FALSE;
128     boolean_t    in_kernel = B_FALSE;

130     if (provname == NULL) {
131         return (FAILURE);
132     }

134     if (check_kernel_for_soft(provname, NULL, &in_kernel) == FAILURE) {
135         return (FAILURE);
136     } else if (in_kernel == B_FALSE) {
124     if ((pent = getent_kef(provname)) == NULL) {
137         cryptoerror(LOG_STDERR, gettext("%s does not exist."),
138             provname);
139         return (FAILURE);
140     }
141     pent = getent_kef(provname, phardlist, psoftlist);

143     rc = get_soft_info(provname, &pmechlist, phardlist, psoftlist);
130     rc = get_soft_info(provname, &pmechlist);
144     if (rc == SUCCESS) {
145         has_random = filter_mechlist(&pmechlist, RANDOM);
146         if (pmechlist != NULL) {
147             has_mechs = B_TRUE;
148             free_mechlist(pmechlist);
149         }
150     } else {
151         cryptoerror(LOG_STDERR, gettext(
152             "failed to retrieve the mechanism list for %s."),
153             provname);
154         return (rc);
155     }

157     print_kef_policy(provname, pent, has_random, has_mechs);
144     print_kef_policy(pent, has_random, has_mechs);
158     free_entry(pent);
159     return (SUCCESS);
160 }

164 /*
165  * Display the policy information for a kernel hardware provider.
166  * This implements part of the "cryptoadm list -p" command.
167  *
168  * Parameters phardlist and psoftlist are supplied by get_kcfconf_info().
169  * If NULL, this function obtains it by calling get_kcfconf_info() internally.
170  * Parameter pdevlist is supplied by get_dev_list().
171  * If NULL, this function obtains it by calling get_dev_list() internally.
172  */
173 int
174 list_policy_for_hard(char *provname,
175     entrylist_t *phardlist, entrylist_t *psoftlist,
176     crypto_get_dev_list_t *pdevlist)
155 list_policy_for_hard(char *provname)
177 {
178     entry_t      *pent = NULL;
179     boolean_t    in_kernel;
180     mechlist_t   *pmechlist = NULL;
157     entry_t *pent;

```

```

158     boolean_t is_active;
159     mechlist_t *pmechlist;
181     char        devname[MAXNAMELEN];
182     int          inst_num;
183     int          count;
184     int          rc = SUCCESS;
185     boolean_t    has_random = B_FALSE;
186     boolean_t    has_mechs = B_FALSE;

188     if (provname == NULL) {
189         return (FAILURE);
190     }

192     /*
193     * Check if the provider is valid. If it is valid, get the number of
194     * mechanisms also.
195     */
196     if (check_hardware_provider(provname, devname, &inst_num, &count) ==
197         FAILURE) {
198         return (FAILURE);
199     }

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

206         if (pmechlist != NULL) {
207             has_mechs = B_TRUE;
208             free_mechlist(pmechlist);
209         }
210     } else {
211         cryptoerror(LOG_STDERR, gettext(
212             "failed to retrieve the mechanism list for %s."),
213             devname);
214         return (rc);
215     }

217     /*
218     * If the hardware provider has an entry in the kcf.conf file,
219     * some of its mechanisms must have been disabled. Print out
220     * the disabled list from the config file entry. Otherwise,
221     * if it is active, then all the mechanisms for it are enabled.
222     */
223     if ((pent = getent_kef(provname, phardlist, psoftlist)) != NULL) {
224         print_kef_policy(provname, pent, has_random, has_mechs);
202     if ((pent = getent_kef(provname)) != NULL) {
203         print_kef_policy(pent, has_random, has_mechs);
225         free_entry(pent);
226         return (SUCCESS);
227     } else {
228         if (check_kernel_for_hard(provname, pdevlist,
229             &in_kernel) == FAILURE) {
207     if (check_active_for_hard(provname, &is_active) ==
208         FAILURE) {
230         return (FAILURE);
231     } else if (in_kernel == B_TRUE) {
210     } else if (is_active == B_TRUE) {
232         (void) printf(gettext(
233             "%s: all mechanisms are enabled."), provname);
234         if (has_random)
235             /*
236             * TRANSLATION_NOTE
237             * "random" is a keyword and not to be
238             * translated.
239             */

```

```

240             (void) printf(gettext("%s is enabled.\n"),
241                "random");
242     else
243         (void) printf("\n");
244     return (SUCCESS);
245 } else {
246     cryptoerror(LOG_STDERR,
247                gettext("%s does not exist."), provname);
248     return (FAILURE);
249 }
250 }
251 }

254 /*
255  * Disable a kernel hardware provider.
256  * This implements the "cryptoadm disable" command for
257  * kernel hardware providers.
258  */

259 int
260 disable_kef_hardware(char *provname, boolean_t rndflag, boolean_t allflag,
261                     mechlist_t *dislist)
262 {
263     crypto_load_dev_disabled_t *pload_dev_dis = NULL;
264     mechlist_t *infolist = NULL;
265     entry_t *pent = NULL;
266     crypto_load_dev_disabled_t *pload_dev_dis;
267     mechlist_t *infolist;
268     entry_t *pent;
269     boolean_t new_dev_entry = B_FALSE;
270     char devname[MAXNAMELEN];
271     int inst_num;
272     int count;
273     int fd = -1;
274     int fd;
275     int rc = SUCCESS;

276     if (provname == NULL) {
277         return (FAILURE);
278     }

279     /*
280      * Check if the provider is valid. If it is valid, get the number of
281      * mechanisms also.
282      */
283     if (check_hardware_provider(provname, devname, &inst_num, &count)
284         == FAILURE) {
285         return (FAILURE);
286     }

287     /* Get the mechanism list for the kernel hardware provider */
288     if (get_dev_info(devname, inst_num, count, &infolist) == FAILURE) {
289         return (FAILURE);
290     }

291     /*
292      * Get the entry of this hardware provider from the config file.
293      * If there is no entry yet, create one for it.
294      */
295     if ((pent = getent_kef(provname, NULL, NULL)) == NULL) {
296         if ((pent = create_entry(provname)) == NULL) {
297             if ((pent = getent_kef(provname)) == NULL) {
298                 if ((pent = malloc(sizeof(entry_t))) == NULL) {
299                     cryptoerror(LOG_STDERR, gettext("out of memory."));
300                     free_mechlist(infolist);

```

```

299         return (FAILURE);
300     }
301     new_dev_entry = B_TRUE;
302     (void) strncpy(pent->name, provname, MAXNAMELEN);
303     pent->suplist = NULL;
304     pent->sup_count = 0;
305     pent->dislist = NULL;
306     pent->dis_count = 0;
307 }

308 /*
309  * kCF treats random as an internal mechanism. So, we need to
310  * filter it from the mechanism list here, if we are NOT disabling
311  * or enabling the random feature. Note that we map random feature at
312  * cryptoadm(lm) level to the "random" mechanism in kCF.
313  */
314 if (!rndflag) {
315     (void) filter_mechlist(&dislist, RANDOM);
316 }

317 /* Calculate the new disabled list */
318 if (disable_mechs(&pent, infolist, allflag, dislist) == FAILURE) {
319     free_mechlist(infolist);
320     free_entry(pent);
321     return (FAILURE);
322 }
323 free_mechlist(infolist);

324 /* If no mechanisms are to be disabled, return */
325 if (pent->dis_count == 0) {
326     free_entry(pent);
327     return (SUCCESS);
328 }

329 /* Update the config file with the new entry or the updated entry */
330 if (new_dev_entry) {
331     rc = update_kcfconf(pent, ADD_MODE);
332 } else {
333     rc = update_kcfconf(pent, MODIFY_MODE);
334 }

335 if (rc == FAILURE) {
336     free_entry(pent);
337     return (FAILURE);
338 }

339 /* Inform kernel about the new disabled mechanism list */
340 if ((pload_dev_dis = setup_dev_dis(pent)) == NULL) {
341     free_entry(pent);
342     return (FAILURE);
343 }
344 free_entry(pent);

345 if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
346     cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
347                ADMIN_IOCTL_DEVICE, strerror(errno));
348     free(pload_dev_dis);
349     return (FAILURE);
350 }

351 if (ioctl(fd, CRYPTO_LOAD_DEV_DISABLED, pload_dev_dis) == -1) {
352     cryptodebug("CRYPTO_LOAD_DEV_DISABLED ioctl failed: %s",
353                strerror(errno));
354     free(pload_dev_dis);
355     (void) close(fd);
356     return (FAILURE);

```

```

360     }
362     if (pload_dev_dis->dd_return_value != CRYPTO_SUCCESS) {
363         cryptodebug("CRYPTO_LOAD_DEV_DISABLED ioctl return_value = "
364             "%d", pload_dev_dis->dd_return_value);
365         free(pload_dev_dis);
366         (void) close(fd);
367         return (FAILURE);
368     }
370     free(pload_dev_dis);
371     (void) close(fd);
372     return (SUCCESS);
373 }
376 /*
377  * Disable a kernel software provider.
378  * This implements the "cryptoadm disable" command for
379  * kernel software providers.
380  */
381 int
382 disable_kef_software(char *provname, boolean_t rndflag, boolean_t allflag,
383     mechlist_t *dislist)
384 {
385     crypto_load_soft_disabled_t    *pload_soft_dis = NULL;
386     mechlist_t                    *infolist = NULL;
387     entry_t                        *pent = NULL;
388     entrylist_t                   *phardlist = NULL;
389     entrylist_t                   *psoftlist = NULL;
390     boolean_t                      in_kernel = B_FALSE;
391     int                            fd = -1;
392     int                            rc = SUCCESS;
393     mechlist_t                    *infolist;
394     entry_t                        *pent;
395     boolean_t                      is_active;
396     int                            fd;
397
398     if (provname == NULL) {
399         return (FAILURE);
400     }
401
402     /* Get the entry of this provider from the config file. */
403     if ((pent = getent_kef(provname)) == NULL) {
404         cryptodebug(LOG_STDERR,
405             gettext("%s does not exist."), provname);
406         return (FAILURE);
407     }
408
409     /*
410      * Check if the kernel software provider is currently unloaded.
411      * If it is unloaded, return FAILURE, because the disable subcommand
412      * can not perform on inactive (unloaded) providers.
413      */
414     if (check_kernel_for_soft(provname, NULL, &in_kernel) == FAILURE) {
415         if (check_active_for_soft(provname, &is_active) == FAILURE) {
416             free_entry(pent);
417             return (FAILURE);
418         } else if (in_kernel == B_FALSE) {
419             /* TRANSLATION_NOTE
420              * "disable" is a keyword and not to be translated.
421              */
422             cryptodebug(LOG_STDERR,

```

```

407         gettext("%s is not loaded or does not exist."),
408         provname);
409         gettext("can not do %1$s on an unloaded "
410             "kernel software provider -- %2$s."), "disable", provname);
411         free_entry(pent);
412         return (FAILURE);
413     }
414
415     if (get_kcfconf_info(&phardlist, &psoftlist) == FAILURE) {
416         cryptodebug(LOG_ERR,
417             "failed to retrieve the providers' "
418             "information from the configuration file - %s.",
419             _PATH_KCF_CONF);
420         /* Get the mechanism list for the software provider */
421         if (get_soft_info(provname, &infolist) == FAILURE) {
422             free(pent);
423             return (FAILURE);
424         }
425     }
426
427     /* Get the entry of this provider from the kcf.conf file, if any.
428     * Otherwise, create a new kcf.conf entry for writing back to the file.
429     */
430     pent = getent_kef(provname, phardlist, psoftlist);
431     if (pent == NULL) { /* create a new entry */
432         pent = create_entry(provname);
433         if (pent == NULL) {
434             cryptodebug("out of memory.");
435             rc = FAILURE;
436             goto out;
437         }
438     }
439
440     /* Get the mechanism list for the software provider from the kernel */
441     if (get_soft_info(provname, &infolist, phardlist, psoftlist) ==
442         FAILURE) {
443         rc = FAILURE;
444         goto out;
445     }
446
447     if ((infolist != NULL) && (infolist->name[0] != '\0')) {
448         /*
449          * Replace the supportedlist from kcf.conf with possibly
450          * more-up-to-date list from the kernel. This is the case
451          * for default software providers that had more mechanisms
452          * added in the current version of the kernel.
453          */
454         free_mechlist(pent->suplist);
455         pent->suplist = infolist;
456     }
457
458     /*
459     * KCF treats random as an internal mechanism. So, we need to
460     * filter it from the mechanism list here, if we are NOT disabling
461     * or enabling the random feature. Note that we map random feature at
462     * cryptoadm(1M) level to the "random" mechanism in kcf.
463     */
464     /* See comments in disable_kef_hardware() */
465     if (!rndflag) {
466         (void) filter_mechlist(&infolist, RANDOM);
467     }
468
469     /* Calculate the new disabled list */
470     if (disable_mechs(&pent, infolist, allflag, dislist) == FAILURE) {
471         rc = FAILURE;
472         goto out;
473     }

```

```

411     free_entry(pent);
412     free_mechlist(infolist);
413     return (FAILURE);
414 }

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

468 /* Update the kcf.conf file with the updated entry */
469 if (update_kcfconf(pent, MODIFY_MODE) == FAILURE) {
470     rc = FAILURE;
471     goto out;
472     free_entry(pent);
473     return (FAILURE);
474 }

474 /* Setup argument to inform kernel about the new disabled list. */
475 /* Inform kernel about the new disabled list. */
476 if ((pload_soft_dis = setup_soft_dis(pent)) == NULL) {
477     rc = FAILURE;
478     goto out;
479     free_entry(pent);
480     return (FAILURE);
481 }

482 /* pent is no longer needed; free it. */
483 free_entry(pent);

484 if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
485     cryptoerror(LOG_STDERR,
486         gettext("failed to open %s for RW: %s"),
487         ADMIN_IOCTL_DEVICE, strerror(errno));
488     rc = FAILURE;
489     goto out;
490     free(pload_soft_dis);
491     return (FAILURE);
492 }

493 /* Inform kernel about the new disabled list. */
494 if (ioctl(fd, CRYPTO_LOAD_SOFT_DISABLED, pload_soft_dis) == -1) {
495     cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl failed: %s",
496         strerror(errno));
497     rc = FAILURE;
498     goto out;
499     free(pload_soft_dis);
500     (void) close(fd);
501     return (FAILURE);
502 }

503 out:
504 free_entrylist(phardlist);
505 free_entrylist(psoftlist);
506 free_mechlist(infolist);
507 free_entry(pent);
508 free(pload_soft_dis);
509 if (fd != -1)

```

```

510     (void) close(fd);
511     return (rc);
512     return (SUCCESS);
513 }

515 /*
516  * Enable a kernel software or hardware provider.
517  * This implements the "cryptoadm enable" command for kernel providers.
518  */
519 int
520 enable_kef(char *provname, boolean_t rndflag, boolean_t allflag,
521     mechlist_t *mlist)
522 {
523     crypto_load_soft_disabled_t *pload_soft_dis = NULL;
524     crypto_load_dev_disabled_t *pload_dev_dis = NULL;
525     entry_t *pent = NULL;
526     entry_t *pent;
527     boolean_t redo_flag = B_FALSE;
528     boolean_t in_kernel = B_FALSE;
529     int fd = -1;
530     int rc = SUCCESS;

532 /* Get the entry of this provider from the kcf.conf file, if any. */
533 pent = getent_kef(provname, NULL, NULL);
534 /* Get the entry with the provider name from the kcf.conf file */
535 pent = getent_kef(provname);

536 if (is_device(provname)) {
537     if (pent == NULL) {
538         /*
539          * This device doesn't have an entry in the config
540          * file, therefore nothing is disabled.
541          */
542         cryptoerror(LOG_STDERR, gettext(
543             "all mechanisms are enabled already for %s."),
544             provname);
545         free_entry(pent);
546         return (SUCCESS);
547     } else { /* a software module */
548         if (check_kernel_for_soft(provname, NULL, &in_kernel) ==
549             FAILURE) {
550             free_entry(pent);
551             if (pent == NULL) {
552                 cryptoerror(LOG_STDERR,
553                     gettext("%s does not exist."), provname);
554                 return (FAILURE);
555             } else if (in_kernel == B_FALSE) {
556                 cryptoerror(LOG_STDERR, gettext("%s does not exist."),
557                     provname);
558                 free_entry(pent);
559                 return (FAILURE);
560             } else if ((pent == NULL) || (pent->dis_count == 0)) {
561                 /* nothing to be enabled. */
562                 cryptoerror(LOG_STDERR, gettext(
563                     "all mechanisms are enabled already for %s."),
564                     provname);
565                 free_entry(pent);
566                 return (SUCCESS);
567             }
568         }
569     }
570 }

```

```

567 /*
568  * kCF treats random as an internal mechanism. So, we need to
569  * filter it from the mechanism list here, if we are NOT disabling
570  * or enabling the random feature. Note that we map random feature at
571  * cryptoadm(1M) level to the "random" mechanism in kCF.
572  */
573 if (!rndflag) {
574     /* See comments in disable_kef_hardware() */
575     redo_flag = filter_mechlist(&pent->dislist, RANDOM);
576     if (redo_flag)
577         pent->dis_count--;
578
579     /* Update the entry by enabling mechanisms for this provider */
580     if ((rc = enable_mechs(&pent, allflag, mlist)) != SUCCESS) {
581         free_entry(pent);
582         return (rc);
583     }
584
585     if (redo_flag) {
586         mechlist_t *tmp;
587
588         if ((tmp = create_mech(RANDOM)) == NULL) {
589             free_entry(pent);
590             return (FAILURE);
591         }
592         tmp->next = pent->dislist;
593         pent->dislist = tmp;
594         pent->dis_count++;
595     }
596
597     /*
598     * Update the kcf.conf file with the updated entry.
599     * For a hardware provider, if there is no more disabled mechanism,
600     * remove the entire kcf.conf entry.
601     * the entire entry in the config file should be removed.
602     */
603     if (is_device(pent->name) && (pent->dis_count == 0)) {
604         rc = update_kcfconf(pent, DELETE_MODE);
605     } else {
606         rc = update_kcfconf(pent, MODIFY_MODE);
607     }
608
609     if (rc == FAILURE) {
610         free_entry(pent);
611         return (FAILURE);
612     }
613
614     /* Inform Kernel about the policy change */
615
616     if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
617         cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
618             ADMIN_IOCTL_DEVICE, strerror(errno));
619         free_entry(pent);
620         return (FAILURE);
621     }
622
623     if (is_device(provname)) {
624         /* LOAD_DEV_DISABLED */
625         if ((pload_dev_dis = setup_dev_dis(pent)) == NULL) {
626             free_entry(pent);
627             return (FAILURE);
628         }
629     }
630
631     if (ioctl(fd, CRYPTO_LOAD_DEV_DISABLED, pload_dev_dis) == -1) {

```

```

631         cryptodebug("CRYPTO_LOAD_DEV_DISABLED ioctl failed: "
632             "%s", strerror(errno));
633         free_entry(pent);
634         free(pload_dev_dis);
635         (void) close(fd);
636         return (FAILURE);
637     }
638
639     if (pload_dev_dis->dd_return_value != CRYPTO_SUCCESS) {
640         cryptodebug("CRYPTO_LOAD_DEV_DISABLED ioctl "
641             "return_value = %d",
642             pload_dev_dis->dd_return_value);
643         free_entry(pent);
644         free(pload_dev_dis);
645         (void) close(fd);
646         return (FAILURE);
647     }
648
649     } else { /* a software module */
650     } else {
651         /* LOAD_SOFT_DISABLED */
652         if ((pload_soft_dis = setup_soft_dis(pent)) == NULL) {
653             free_entry(pent);
654             return (FAILURE);
655         }
656
657         if (ioctl(fd, CRYPTO_LOAD_SOFT_DISABLED, pload_soft_dis)
658             == -1) {
659             cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl failed: "
660                 "%s", strerror(errno));
661             free_entry(pent);
662             free(pload_soft_dis);
663             (void) close(fd);
664             return (FAILURE);
665         }
666
667         if (pload_soft_dis->sd_return_value != CRYPTO_SUCCESS) {
668             cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl "
669                 "return_value = %d",
670                 pload_soft_dis->sd_return_value);
671             free_entry(pent);
672             free(pload_soft_dis);
673             (void) close(fd);
674             return (FAILURE);
675         }
676     }
677
678     free_entry(pent);
679     free(pload_soft_dis);
680     (void) close(fd);
681     return (SUCCESS);
682 }
683
684 /*
685  * Install a software module with the specified mechanism list into the system.
686  * This routine adds an entry into the config file for this software module
687  * first, then makes a CRYPTO_LOAD_SOFT_CONFIG ioctl call to inform kernel
688  * about the new addition.
689  */
690 int
691 install_kef(char *provname, mechlist_t *mlist)
692 {
693     crypto_load_soft_config_t *pload_soft_conf = NULL;
694     boolean_t found;
695     entry_t *pent = NULL;

```

```

696 FILE *pfile = NULL;
697 FILE *pfile_tmp = NULL;
619 entry_t *pent;
620 FILE *pfile;
621 FILE *pfile_tmp;
698 char tmpfile_name[MAXPATHLEN];
699 char *ptr;
700 char *str;
701 char *name;
702 char buffer[BUFSIZ];
703 char buffer2[BUFSIZ];
704 int found_count;
705 int fd = -1;
629 int fd;
706 int rc = SUCCESS;
707 int err;

709 if ((provname == NULL) || (mlist == NULL)) {
710     return (FAILURE);
711 }

713 /* Check if the provider already exists */
714 if ((pent = getent_kef(provname, NULL, NULL)) != NULL) {
637 if ((pent = getent_kef(provname)) != NULL) {
715     cryptoerror(LOG_STDERR, gettext("%s exists already."),
716                 provname);
717     free_entry(pent);
718     return (FAILURE);
719 }

721 /* Create an entry with provname and mlist. */
722 if ((pent = create_entry(provname)) == NULL) {
645 if ((pent = malloc(sizeof (entry_t))) == NULL) {
723     cryptoerror(LOG_STDERR, gettext("out of memory."));
724     return (FAILURE);
725 }

650 (void) strncpy(pent->name, provname, MAXNAMELEN);
726 pent->sup_count = get_mech_count(mlist);
727 pent->suplist = mlist;
653 pent->dis_count = 0;
654 pent->dislist = NULL;

729 /* Append an entry for this software module to the kcf.conf file. */
730 if ((str = ent2str(pent)) == NULL) {
731     free_entry(pent);
732     return (FAILURE);
733 }

735 if ((pfile = fopen(_PATH_KCF_CONF, "r+")) == NULL) {
736     err = errno;
737     cryptoerror(LOG_STDERR,
738                 gettext("failed to update the configuration - %s"),
739                 strerror(err));
740     cryptodebug("failed to open %s for write.", _PATH_KCF_CONF);
741     free_entry(pent);
742     return (FAILURE);
743 }

745 if (lockf(fileno(pfile), F_TLOCK, 0) == -1) {
746     err = errno;
747     cryptoerror(LOG_STDERR,
748                 gettext("failed to lock the configuration - %s"),
749                 strerror(err));
750     free_entry(pent);
751     (void) fclose(pfile);

```

```

752     return (FAILURE);
753 }

755 /*
756  * Create a temporary file in the /etc/crypto directory.
757  */
758 (void) strncpy(tmpfile_name, TMPFILE_TEMPLATE, sizeof (tmpfile_name));
759 if (mkstemp(tmpfile_name) == -1) {
760     err = errno;
761     cryptoerror(LOG_STDERR,
762                 gettext("failed to create a temporary file - %s"),
763                 strerror(err));
764     free_entry(pent);
765     (void) fclose(pfile);
766     return (FAILURE);
767 }

769 if ((pfile_tmp = fopen(tmpfile_name, "w")) == NULL) {
770     err = errno;
771     cryptoerror(LOG_STDERR, gettext("failed to open %s - %s"),
772                 tmpfile_name, strerror(err));
773     free_entry(pent);
774     (void) fclose(pfile);
775     return (FAILURE);
776 }

779 /*
780  * Loop thru the config file. If the provider was reserved within a
781  * package bracket, just uncomment it. Otherwise, append it at
782  * the end. The resulting file will be saved in the temp file first.
783  */
784 found_count = 0;
785 rc = SUCCESS;
786 while (fgets(buffer, BUFSIZ, pfile) != NULL) {
787     found = B_FALSE;
788     if (buffer[0] == '#') {
789         (void) strncpy(buffer2, buffer, BUFSIZ);
790         ptr = buffer2;
791         ptr++;
792         if ((name = strtok(ptr, SEP_COLON)) == NULL) {
793             rc = FAILURE;
794             break;
795         } else if (strcmp(provname, name) == 0) {
796             found = B_TRUE;
797             found_count++;
798         }
799     }

801 if (found == B_FALSE) {
802     if (fputs(buffer, pfile_tmp) == EOF) {
803         rc = FAILURE;
804     }
805 } else {
806     if (found_count == 1) {
807         if (fputs(str, pfile_tmp) == EOF) {
808             rc = FAILURE;
809         }
810     } else {
811         /*
812          * Found a second entry with #libname.
813          * Should not happen. The kcf.conf file
814          * Should not happen. The kcf.conf file
815          * is corrupted. Give a warning and skip
816          * this entry.
817          */

```

```

817         cryptoerror(LOG_STDERR, gettext(
818             "(Warning) Found an additional reserved "
819             "entry for %s.", provname);
820     }
821 }
822
823     if (rc == FAILURE) {
824         break;
825     }
826 }
827 (void) fclose(pfile);
828
829 if (rc == FAILURE) {
830     cryptoerror(LOG_STDERR, gettext("write error.));
831     (void) fclose(pfile_tmp);
832     if (unlink(tmpfile_name) != 0) {
833         err = errno;
834         cryptoerror(LOG_STDERR, gettext(
835             "(Warning) failed to remove %s: %s", tmpfile_name,
836             strerror(err));
837     }
838     free_entry(pent);
839     return (FAILURE);
840 }
841
842 if (found_count == 0) {
843     /*
844     * This libname was not in package before, append it to the
845     * end of the temp file.
846     */
847     if (fputs(str, pfile_tmp) == EOF) {
848         cryptoerror(LOG_STDERR, gettext(
849             "failed to write to %s: %s", tmpfile_name,
850             strerror(errno));
851         (void) fclose(pfile_tmp);
852         if (unlink(tmpfile_name) != 0) {
853             err = errno;
854             cryptoerror(LOG_STDERR, gettext(
855                 "(Warning) failed to remove %s: %s",
856                 tmpfile_name, strerror(err));
857         }
858         free_entry(pent);
859         return (FAILURE);
860     }
861 }
862
863 if (fclose(pfile_tmp) != 0) {
864     err = errno;
865     cryptoerror(LOG_STDERR,
866         gettext("failed to close %s: %s"), tmpfile_name,
867         strerror(err));
868     free_entry(pent);
869     return (FAILURE);
870 }
871
872 if (rename(tmpfile_name, _PATH_KCF_CONF) == -1) {
873     err = errno;
874     cryptoerror(LOG_STDERR,
875         gettext("failed to update the configuration - %s"),
876         strerror(err));
877     cryptodebug("failed to rename %s to %s: %s", tmpfile_name,
878         _PATH_KCF_CONF, strerror(err));
879     rc = FAILURE;
880 } else if (chmod(_PATH_KCF_CONF,
881     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) == -1) {
882     err = errno;

```

```

883         cryptoerror(LOG_STDERR,
884             gettext("failed to update the configuration - %s"),
885             strerror(err));
886     cryptodebug("failed to chmod to %s: %s", _PATH_KCF_CONF,
887         strerror(err));
888     rc = FAILURE;
889 } else {
890     rc = SUCCESS;
891 }
892
893 if (rc == FAILURE) {
894     if (unlink(tmpfile_name) != 0) {
895         err = errno;
896         cryptoerror(LOG_STDERR, gettext(
897             "(Warning) failed to remove %s: %s",
898             tmpfile_name, strerror(err));
899     }
900     free_entry(pent);
901     return (FAILURE);
902 }
903
904 /* Inform kernel of this new software module. */
905
906 if ((pload_soft_conf = setup_soft_conf(pent)) == NULL) {
907     free_entry(pent);
908     return (FAILURE);
909 }
910
911 if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
912     cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
913         ADMIN_IOCTL_DEVICE, strerror(errno));
914     free_entry(pent);
915     free(pload_soft_conf);
916     return (FAILURE);
917 }
918
919 if (ioctl(fd, CRYPTO_LOAD_SOFT_CONFIG, pload_soft_conf) == -1) {
920     cryptodebug("CRYPTO_LOAD_SOFT_CONFIG ioctl failed: %s",
921         strerror(errno));
922     free_entry(pent);
923     free(pload_soft_conf);
924     (void) close(fd);
925     return (FAILURE);
926 }
927
928 if (pload_soft_conf->sc_return_value != CRYPTO_SUCCESS) {
929     cryptodebug("CRYPTO_LOAD_SOFT_CONFIG ioctl failed, "
930         "return_value = %d", pload_soft_conf->sc_return_value);
931     free_entry(pent);
932     free(pload_soft_conf);
933     (void) close(fd);
934     return (FAILURE);
935 }
936
937 free_entry(pent);
938 free(pload_soft_conf);
939 (void) close(fd);
940 return (SUCCESS);
941 }
942
943 /*
944 * Uninstall the software module. This routine first unloads the software
945 * module with 3 ioctl calls, then deletes its entry from the config file.
946 * Removing an entry from the config file needs to be done last to ensure
947 * that there is still an entry if the earlier unload failed for any reason.

```



```

949 */
950 int
951 uninstall_kef(char *provname)
952 {
953     entry_t      *pent = NULL;
954     entry_t      *pent;
955     boolean_t    is_active;
956     boolean_t    in_package;
957     boolean_t    found;
958     FILE         *pfile;
959     FILE         *pfile_tmp;
960     char         tmpfile_name[MAXPATHLEN];
961     char         *name;
962     char         strbuf[BUFSIZ];
963     char         buffer[BUFSIZ];
964     char         buffer2[BUFSIZ];
965     char         *str;
966     int          len;
967     int          rc = SUCCESS;
968     boolean_t    in_kernel = B_FALSE;
969     boolean_t    in_kcfconf = B_FALSE;
970     int         fd = -1;
971     crypto_load_soft_config_t *pload_soft_conf = NULL;
972
973     /* Check to see if the provider exists first. */
974     if (check_kernel_for_soft(provname, NULL, &in_kernel) == FAILURE) {
975
976         /* Check if it is in the kcf.conf file first. */
977         if ((pent = getent_kef(provname)) == NULL) {
978             cryptoerror(LOG_STDERR,
979                 gettext("%s does not exist."), provname);
980             return (FAILURE);
981         } else if (in_kernel == B_FALSE) {
982             cryptoerror(LOG_STDERR, gettext("%s does not exist."),
983                 provname);
984             return (FAILURE);
985         }
986     }
987
988     /*
989     * If it is loaded, unload it first. This does 2 ioctl calls:
990     * CRYPTO_UNLOAD_SOFT_MODULE and CRYPTO_LOAD_SOFT_DISABLED.
991     * Get rid of the disabled list for the provider and get the converted
992     * string for the entry. This is to prepare the string for a provider
993     * that is in a package.
994     */
995     if (unload_kef_soft(provname) == FAILURE) {
996         free_mechlist(pent->dislist);
997         pent->dis_count = 0;
998         pent->dislist = NULL;
999         str = ent2str(pent);
1000         free_entry(pent);
1001         if (str == NULL) {
1002             cryptoerror(LOG_STDERR, gettext("internal error."));
1003             return (FAILURE);
1004         }
1005         (void) snprintf(strbuf, sizeof (strbuf), "%s%s", "#", str);
1006         free(str);
1007
1008         /* If it is not loaded, unload it first */
1009         if (check_active_for_soft(provname, &is_active) == FAILURE) {
1010             return (FAILURE);
1011         } else if ((is_active == B_TRUE) &&
1012             (unload_kef_soft(provname, B_TRUE) == FAILURE)) {
1013             cryptoerror(LOG_STDERR,
1014                 gettext("failed to unload %s during uninstall.\n"),

```

```

1015         provname);
1016         gettext("failed to uninstall %s.\n"), provname);
1017         return (FAILURE);
1018     }
1019
1020     /*
1021     * Inform kernel to remove the configuration of this software module.
1022     * Remove the entry from the config file. If the provider to be
1023     * uninstalled is in a package, just comment it off.
1024     */
1025     if ((pfile = fopen(_PATH_KCF_CONF, "r+")) == NULL) {
1026         err = errno;
1027         cryptoerror(LOG_STDERR,
1028             gettext("failed to update the configuration - %s"),
1029             strerror(err));
1030         cryptoerror(LOG_STDERR,
1031             gettext("failed to open %s for write.", _PATH_KCF_CONF));
1032         return (FAILURE);
1033     }
1034
1035     /* Setup ioctl() parameter */
1036     pent = getent_kef(provname, NULL, NULL);
1037     if (pent != NULL) { /* in kcf.conf */
1038         in_kcfconf = B_TRUE;
1039         free_mechlist(pent->suplist);
1040         pent->suplist = NULL;
1041         pent->sup_count = 0;
1042     } else if ((pent = create_entry(provname)) == NULL) {
1043         cryptoerror(LOG_STDERR, gettext("out of memory."));
1044         if (lockf(fileno(pfile), F_TLOCK, 0) == -1) {
1045             err = errno;
1046             cryptoerror(LOG_STDERR,
1047                 gettext("failed to lock the configuration - %s"),
1048                 strerror(err));
1049             (void) fclose(pfile);
1050             return (FAILURE);
1051         }
1052     }
1053     if ((pload_soft_conf = setup_soft_conf(pent)) == NULL) {
1054         free_entry(pent);
1055
1056         /*
1057         * Create a temporary file in the /etc/crypto directory to save
1058         * the new configuration file first.
1059         */
1060         (void) strncpy(tmpfile_name, TMPFILE_TEMPLATE, sizeof (tmpfile_name));
1061         if (mkstemp(tmpfile_name) == -1) {
1062             err = errno;
1063             cryptoerror(LOG_STDERR,
1064                 gettext("failed to create a temporary file - %s"),
1065                 strerror(err));
1066             (void) fclose(pfile);
1067             return (FAILURE);
1068         }
1069
1070         /* Open the /dev/cryptoadm device */
1071         if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
1072             int err = errno;
1073             cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
1074                 ADMIN_IOCTL_DEVICE, strerror(err));
1075             free_entry(pent);
1076             free(pload_soft_conf);
1077             if ((pfile_tmp = fopen(tmpfile_name, "w")) == NULL) {
1078                 err = errno;
1079                 cryptoerror(LOG_STDERR, gettext("failed to open %s - %s"),
1080                     tmpfile_name, strerror(err));
1081                 if (unlink(tmpfile_name) != 0) {
1082                     err = errno;

```

```

971         cryptoerror(LOG_STDERR, gettext(
972             "(Warning) failed to remove %s: %s"), tmpfile_name,
973             strerror(err));
974     }
975     (void) fclose(pfile);
1007     return (FAILURE);
1008 }

1010 if (ioctl(fd, CRYPTO_LOAD_SOFT_CONFIG,
1011     pload_soft_conf) == -1) {
1012     cryptodebug("CRYPTO_LOAD_SOFT_CONFIG ioctl failed: %s",
1013         strerror(errno));
1014     free_entry(pent);
1015     free(pload_soft_conf);
1016     (void) close(fd);
979 /*
980  * Loop thru the config file.  If the kernel software provider
981  * to be uninstalled is in a package, just comment it off.
982  */
983 in_package = B_FALSE;
984 while (fgets(buffer, BUFSIZ, pfile) != NULL) {
985     found = B_FALSE;
986     if (!(buffer[0] == ' ' || buffer[0] == '\n' ||
987         buffer[0] == '\t')) {
988         if (strstr(buffer, " Start ") != NULL) {
989             in_package = B_TRUE;
990         } else if (strstr(buffer, " End ") != NULL) {
991             in_package = B_FALSE;
992         } else if (buffer[0] != '#') {
993             (void) strncpy(buffer2, buffer, BUFSIZ);

995             /* get rid of trailing '\n' */
996             len = strlen(buffer2);
997             if (buffer2[len-1] == '\n') {
998                 len--;
999             }
1000             buffer2[len] = '\0';

1002             if ((name = strtok(buffer2, SEP_COLON))
1003                 == NULL) {
1004                 rc = FAILURE;
1005                 break;
1006             } else if (strcmp(provname, name) == 0) {
1007                 found = B_TRUE;
1008             }
1009         }
1010     }

1012     if (found) {
1013         if (in_package) {
1014             if (fputs(strbuf, pfile_tmp) == EOF) {
1015                 rc = FAILURE;
1016             }
1017         }
1018     } else {
1019         if (fputs(buffer, pfile_tmp) == EOF) {
1020             rc = FAILURE;
1021         }
1022     }

1024     if (rc == FAILURE) {
1025         break;
1026     }
1027 }

1029 if (rc == FAILURE) {

```

```

1030         cryptoerror(LOG_STDERR, gettext("write error."));
1031         (void) fclose(pfile);
1032         (void) fclose(pfile_tmp);
1033         if (unlink(tmpfile_name) != 0) {
1034             err = errno;
1035             cryptoerror(LOG_STDERR, gettext(
1036                 "(Warning) failed to remove %s: %s"), tmpfile_name,
1037                 strerror(err));
1038         }
1039     }
1040     return (FAILURE);
1041 }

1020 if (pload_soft_conf->sc_return_value != CRYPTO_SUCCESS) {
1021     cryptodebug("CRYPTO_LOAD_SOFT_CONFIG ioctl = return_value = %d",
1022         pload_soft_conf->sc_return_value);
1023     free_entry(pent);
1024     free(pload_soft_conf);
1025     (void) close(fd);
1042     (void) fclose(pfile);
1043     if (fclose(pfile_tmp) != 0) {
1044         err = errno;
1045         cryptoerror(LOG_STDERR,
1046             gettext("failed to close %s: %s"), tmpfile_name,
1047             strerror(err));
1026     }
1027     return (FAILURE);
1028 }

1029 /* ioctl cleanup */
1030 free(pload_soft_conf);
1031 (void) close(fd);
1051 /* Now update the real config file */
1052 if (rename(tmpfile_name, _PATH_KCF_CONF) == -1) {
1053     err = errno;
1054     cryptoerror(LOG_STDERR,
1055         gettext("failed to update the configuration - %s"),
1056         strerror(err));
1057     cryptodebug("failed to rename %1$s to %2$s: %3$s", tmpfile,
1058         _PATH_KCF_CONF, strerror(err));
1059     rc = FAILURE;
1060 } else if (chmod(_PATH_KCF_CONF,
1061     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) == -1) {
1062     err = errno;
1063     cryptoerror(LOG_STDERR,
1064         gettext("failed to update the configuration - %s"),
1065         strerror(err));
1066     cryptodebug("failed to chmod to %s: %s", _PATH_KCF_CONF,
1067         strerror(err));
1068     rc = FAILURE;
1069 } else {
1070     rc = SUCCESS;
1071 }

1034 /* Finally, remove entry from kcf.conf, if present */
1035 if (in_kcfconf && (pent != NULL)) {
1036     rc = update_kcfconf(pent, DELETE_MODE);
1073     if ((rc == FAILURE) && (unlink(tmpfile_name) != 0)) {
1074         err = errno;
1075         cryptoerror(LOG_STDERR, gettext(
1076             "(Warning) failed to remove %s: %s"), tmpfile_name,
1077             strerror(err));
1037     }
1038 }

1039 free_entry(pent);
1040 return (rc);

```

```

1041 }

1044 /*
1045 * Implement the "cryptoadm refresh" command for global zones.
1046 * That is, send the current contents of kcf.conf to the kernel via ioctl().
1047 */
1048 int
1049 refresh(void)
1050 {
1051     crypto_get_soft_list_t      *psftlist_kernel = NULL;
1052     crypto_load_soft_config_t   *pload_soft_conf = NULL;
1053     crypto_load_soft_disabled_t *pload_soft_dis = NULL;
1054     crypto_load_dev_disabled_t *pload_dev_dis = NULL;
1055     entrylist_t                 *pdevlist = NULL;
1056     entrylist_t                 *psftlist = NULL;
1057     entrylist_t                 *ptr;
1058     int                          fd = -1;
1059     boolean_t                    found;
1060     char                         *psftname;
1061     int                          rc;
1062     int                          err;
1063     int                          i;

1064     if (get_soft_list(&psftlist_kernel) == FAILURE) {
1065         cryptoerror(LOG_ERR, gettext("Failed to retrieve the "
1066             "software provider list from kernel."));
1067         return (FAILURE);
1068     }

1069     if (get_kcfconf_info(&pdevlist, &psftlist) == FAILURE) {
1070         cryptoerror(LOG_ERR, "failed to retrieve the providers' "
1071             "information from the configuration file - %s.",
1072             _PATH_KCF_CONF);
1073         return (FAILURE);
1074     }

1075     /*
1076      * If a kernel software provider is in kernel, but it is not in the
1077      * kcf.conf file, it must have been pkgrm'ed and needs to be unloaded
1078      * now.
1079      */
1080     if (psftlist_kernel->sl_soft_count > 0) {
1081         psftname = psftlist_kernel->sl_soft_names;
1082         for (i = 0; i < psftlist_kernel->sl_soft_count; i++) {
1083             ptr = psftlist;
1084             found = B_FALSE;
1085             while (ptr != NULL) {
1086                 if (strcmp(psftname, ptr->pent->name) == 0) {
1087                     found = B_TRUE;
1088                     break;
1089                 }
1090                 ptr = ptr->next;
1091             }

1092             if (!found) {
1093                 rc = unload_kef_soft(psftname, B_FALSE);
1094                 if (rc == FAILURE) {
1095                     cryptoerror(LOG_ERR, gettext(
1096                         "WARNING - the provider %s is "
1097                         "still in kernel."), psftname);
1098                 }
1099             }
1100             psftname = psftname + strlen(psftname) + 1;
1101         }
1102     }

```

```

1142     }
1143     free(psftlist_kernel);

1144     if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
1145         err = errno;
1146         cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
1147             ADMIN_IOCTL_DEVICE, strerror(err));
1148         free(psftlist);
1149         free(pdevlist);
1150         return (FAILURE);
1151     }

1152     /*
1153      * For each software provider module, pass two sets of information to
1154      * the kernel: the supported list and the disabled list.
1155      * For each software module, pass two sets of information to kernel
1156      * - the supported list and the disabled list
1157      */
1158     for (ptr = psftlist; ptr != NULL; ptr = ptr->next) {
1159         entry_t                 *pent = ptr->pent;

1160         ptr = psftlist;
1161         while (ptr != NULL) {
1162             /* load the supported list */
1163             if ((pload_soft_conf = setup_soft_conf(pent)) == NULL) {
1164                 cryptoerror(LOG_ERR, "setup_soft_conf() failed");
1165                 if ((pload_soft_conf = setup_soft_conf(ptr->pent)) == NULL) {
1166                     rc = FAILURE;
1167                     break;
1168                 }
1169             }

1170             if (!pent->load) { /* unloaded--mark as loaded */
1171                 pent->load = B_TRUE;
1172                 rc = update_kcfconf(pent, MODIFY_MODE);
1173                 if (rc != SUCCESS) {
1174                     free(pload_soft_conf);
1175                     break;
1176                 }
1177             }

1178             if (ioctl(fd, CRYPTO_LOAD_SOFT_CONFIG, pload_soft_conf)
1179                 == -1) {
1180                 cryptoerror(LOG_ERR, "CRYPTO_LOAD_SOFT_CONFIG ioctl failed: %s",
1181                     strerror(errno));
1182                 free(pload_soft_conf);
1183                 rc = FAILURE;
1184                 break;
1185             }

1186             if (pload_soft_conf->sc_return_value != CRYPTO_SUCCESS) {
1187                 cryptoerror(LOG_ERR, "CRYPTO_LOAD_SOFT_CONFIG ioctl "
1188                     "return_value = %d",
1189                     pload_soft_conf->sc_return_value);
1190                 free(pload_soft_conf);
1191                 rc = FAILURE;
1192                 break;
1193             }

1194             free(pload_soft_conf);
1195         }
1196     }

1197     free(pload_soft_conf);

1198     /* load the disabled list */
1199     if (ptr->pent->dis_count != 0) {
1200         pload_soft_dis = setup_soft_dis(ptr->pent);
1201         if (pload_soft_dis == NULL) {
1202             cryptoerror(LOG_ERR, "setup_soft_dis() failed");
1203             free(pload_soft_dis);
1204         }
1205     }

```

```

1126         rc = FAILURE;
1127         break;
1128     }
1130     if (ioctl(fd, CRYPTO_LOAD_SOFT_DISABLED,
1131         pload_soft_dis) == -1) {
1132         cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl "
1133             "failed: %s", strerror(errno));
1134         free(pload_soft_dis);
1135         rc = FAILURE;
1136         break;
1137     }
1139     if (pload_soft_dis->sd_return_value !=
1140         CRYPTO_SUCCESS) {
1141         cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl "
1142             "return_value = %d",
1143             pload_soft_dis->sd_return_value);
1144         free(pload_soft_dis);
1145         rc = FAILURE;
1146         break;
1147     }
1148     free(pload_soft_dis);
1149 }
1213     free(pload_soft_conf);
1214     ptr = ptr->next;
1150 }
1152 if (rc != SUCCESS) {
1153     (void) close(fd);
1154     return (rc);
1155 }
1158 /*
1159  * For each hardware provider module, pass the disabled list
1160  * information to the kernel.
1161  */
1162 for (ptr = pdevlist; ptr != NULL; ptr = ptr->next) {
1223 /* Pass the disabledlist information for Device to kernel */
1224 ptr = pdevlist;
1225 while (ptr != NULL) {
1163     /* load the disabled list */
1164     if (ptr->pent->dis_count != 0) {
1165         pload_dev_dis = setup_dev_dis(ptr->pent);
1166         if (pload_dev_dis == NULL) {
1167             rc = FAILURE;
1168             break;
1169         }
1171         if (ioctl(fd, CRYPTO_LOAD_DEV_DISABLED, pload_dev_dis)
1172             == -1) {
1173             cryptodebug("CRYPTO_LOAD_DEV_DISABLED ioctl "
1174                 "failed: %s", strerror(errno));
1175             free(pload_dev_dis);
1176             rc = FAILURE;
1177             break;
1178         }
1180         if (pload_dev_dis->dd_return_value != CRYPTO_SUCCESS) {
1181             cryptodebug("CRYPTO_LOAD_DEV_DISABLED ioctl "
1182                 "return_value = %d",
1183                 pload_dev_dis->dd_return_value);
1184             free(pload_dev_dis);
1185             rc = FAILURE;

```

```

1186         break;
1187     }
1188     free(pload_dev_dis);
1189 }
1254     ptr = ptr->next;
1190 }
1192     (void) close(fd);
1193     return (rc);
1194 }
1196 /*
1197  * Unload the kernel software provider. Before calling this function, the
1198  * caller should check to see if the provider is in the kernel.
1199  */
1200 * This routine makes 2 ioctl calls to remove it completely from the kernel:
1201 * CRYPTO_UNLOAD_SOFT_MODULE - does a modunload of the KCF module
1202 * CRYPTO_LOAD_SOFT_DISABLED - updates kernel disabled mechanism list
1203 *
1204 * This implements part of "cryptoadm unload" and "cryptoadm uninstall".
1205 */
1206 int
1207 unload_kef_soft(char *provname)
1208 {
1209     crypto_unload_soft_module_t *punload_soft = NULL;
1210     crypto_load_soft_config_t *pload_soft_conf = NULL;
1211     crypto_load_soft_disabled_t *pload_soft_dis = NULL;
1212     entry_t *pent = NULL;
1213     int fd = -1;
1214     int err;
1215     if (provname == NULL) {
1216         cryptoerror(LOG_STDERR, gettext("internal error.));
1217         return (FAILURE);
1218     }
1220     pent = getent_kef(provname, NULL, NULL);
1221     if (pent == NULL) { /* not in kcf.conf */
1222         if (!do_check) {
1223             /* Construct an entry using the provname */
1224             pent = create_entry(provname);
1225             pent = calloc(1, sizeof(entry_t));
1226             if (pent == NULL) {
1227                 cryptoerror(LOG_STDERR, gettext("out of memory.));
1228                 return (FAILURE);
1229             }
1230             (void) strncpy(pent->name, provname, MAXNAMELEN);
1231         } else if ((pent = getent_kef(provname)) == NULL) {
1232             cryptoerror(LOG_STDERR, gettext("%s does not exist."),
1233                 provname);
1234             return (FAILURE);
1235         }
1236     }
1237     /* Open the admin_ioctl_device */
1238     if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDWR)) == -1) {
1239         err = errno;
1240         cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
1241             ADMIN_IOCTL_DEVICE, strerror(err));

```

```

1235     free_entry(pent);
1236     return (FAILURE);
1237 }

1239 /* Inform kernel to unload this software module */
1240 if ((punload_soft = setup_unload_soft(pent)) == NULL) {
1241     free_entry(pent);
1242     (void) close(fd);
1243     return (FAILURE);
1244 }

1246 if (ioctl(fd, CRYPTO_UNLOAD_SOFT_MODULE, punload_soft) == -1) {
1247     cryptodebug("CRYPTO_UNLOAD_SOFT_MODULE ioctl failed: %s",
1248               strerror(errno));
1249     free_entry(pent);
1250     free(punload_soft);
1251     (void) close(fd);
1252     return (FAILURE);
1253 }

1255 if (punload_soft->sm_return_value != CRYPTO_SUCCESS) {
1256     cryptodebug("CRYPTO_UNLOAD_SOFT_MODULE ioctl return_value = "
1257               "%d", punload_soft->sm_return_value);
1258     /*
1259      * If the return value is CRYPTO_UNKNOWN_PROVIDER, it means
1260      * that the provider is not registered yet. Should just
1261      * continue.
1262      */
1263     if (punload_soft->sm_return_value != CRYPTO_UNKNOWN_PROVIDER) {
1264         free_entry(pent);
1265         free(punload_soft);
1266         (void) close(fd);
1267         return (FAILURE);
1268     }
1269 }

1271 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);

1273 /* Inform kernel to remove the disabled entries if any */
1274 if (pent->dis_count == 0) {
1275     free_entry(pent);
1276     (void) close(fd);
1277     return (SUCCESS);
1278 } else {
1279     free_mechlist(pent->dislist);
1280     pent->dislist = NULL;
1281     pent->dis_count = 0;
1282 }

1284 if ((pload_soft_dis = setup_soft_dis(pent)) == NULL) {
1285     free_entry(pent);
1286     (void) close(fd);
1287     return (FAILURE);
1288 }

1290 /* pent is no longer needed; free it */
1291 free_entry(pent);

1293 if (ioctl(fd, CRYPTO_LOAD_SOFT_DISABLED, pload_soft_dis) == -1) {
1294     cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl failed: %s",
1295               strerror(errno));
1296     free(pload_soft_dis);
1297     (void) close(fd);
1298     return (FAILURE);
1299 }

1301 if (pload_soft_dis->sd_return_value != CRYPTO_SUCCESS) {
1302     cryptodebug("CRYPTO_LOAD_SOFT_DISABLED ioctl return_value = "
1303               "%d", pload_soft_dis->sd_return_value);
1304     free(pload_soft_dis);
1305     (void) close(fd);
1306     return (FAILURE);
1307 }

1309 free(pload_soft_dis);
1310 (void) close(fd);
1311 return (SUCCESS);
1312 }

```

unchanged portion omitted

new/usr/src/cmd/cmd-crypto/cryptoadm/adm_kef_ioctl.c

1

```
*****
14657 Tue Oct 28 16:45:23 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_kef_ioctl.c
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  * Common Development and Distribution License, Version 1.0 only
8  * (the "License"). You may not use this file except in compliance
9  * with the License.
10 *
11 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
12 * or http://www.opensolaris.org/os/licensing.
13 * See the License for the specific language governing permissions
14 * and limitations under the License.
15 *
16 * When distributing Covered Code, include this CDDL HEADER in each
17 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
18 * If applicable, add the following below this CDDL HEADER, with the
19 * fields enclosed by brackets "[]" replaced with your own identifying
20 * information: Portions Copyright [yyyy] [name of copyright owner]
21 *
22 * CDDL HEADER END
23 */
24 /*
25 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
26 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
27 * Use is subject to license terms.
28 */
29 #pragma ident "%Z%M% %I% %E% SMI"
30
31 #include <fcntl.h>
32 #include <stdio.h>
33 #include <stdlib.h>
34 #include <strings.h>
35 #include <unistd.h>
36 #include <locale.h>
37 #include <libgen.h>
38 #include <sys/types.h>
39 #include <zone.h>
40 #include <sys/crypto/ioctladmin.h>
41 #include "cryptoadm.h"
42
43 #define DEFAULT_DEV_NUM 5
44 #define DEFAULT_SOFT_NUM 10
45
46 static crypto_get_soft_info_t *setup_get_soft_info(char *, int);
47
48 /*
49  * Prepare the argument for the LOAD_SOFT_CONFIG ioctl call for the
50  * provider pointed by pent. Return NULL if out of memory.
51  */
52 crypto_load_soft_config_t *
53 setup_soft_conf(entry_t *pent)
54 {
55     crypto_load_soft_config_t *pload_soft_conf;
56     mechlist_t *plist;
57     uint_t sup_count;
58     size_t extra_mech_size = 0;
59     int i;
```

new/usr/src/cmd/cmd-crypto/cryptoadm/adm_kef_ioctl.c

2

```
56     if (pent == NULL) {
57         return (NULL);
58     }
59
60     sup_count = pent->sup_count;
61     if (sup_count > 1) {
62         extra_mech_size = sizeof (crypto_mech_name_t) *
63             (sup_count - 1);
64     }
65
66     pload_soft_conf = malloc(sizeof (crypto_load_soft_config_t) +
67         extra_mech_size);
68     if (pload_soft_conf == NULL) {
69         cryptodebug("out of memory.");
70         return (NULL);
71     }
72
73     (void) strncpy(pload_soft_conf->sc_name, pent->name, MAXNAMELEN);
74     pload_soft_conf->sc_count = sup_count;
75
76     i = 0;
77     plist = pent->suplist;
78     while (i < sup_count) {
79         (void) strncpy(pload_soft_conf->sc_list[i++],
80             plist->name, CRYPTO_MAX_MECH_NAME);
81         plist = plist->next;
82     }
83
84     return (pload_soft_conf);
85 }
86
87 /*
88  * Prepare the argument for the LOAD_SOFT_DISABLED ioctl call for the
89  * provider pointed by pent. Return NULL if out of memory.
90  */
91 crypto_load_soft_disabled_t *
92 setup_soft_dis(entry_t *pent)
93 {
94     crypto_load_soft_disabled_t *pload_soft_dis = NULL;
95     mechlist_t *plist = NULL;
96     crypto_load_soft_disabled_t *pload_soft_dis;
97     mechlist_t *plist;
98     size_t extra_mech_size = 0;
99     uint_t dis_count;
100     int i;
101
102     if (pent == NULL) {
103         return (NULL);
104     }
105
106     dis_count = pent->dis_count;
107     if (dis_count > 1) {
108         extra_mech_size = sizeof (crypto_mech_name_t) *
109             (dis_count - 1);
110     }
111
112     pload_soft_dis = malloc(sizeof (crypto_load_soft_disabled_t) +
113         extra_mech_size);
114     if (pload_soft_dis == NULL) {
115         cryptodebug("out of memory.");
116         return (NULL);
117     }
118
119     (void) strncpy(pload_soft_dis->sd_name, pent->name, MAXNAMELEN);
120     pload_soft_dis->sd_count = dis_count;
```

```

121     i = 0;
122     plist = pent->dislist;
123     while (i < dis_count) {
124         (void) strcpy(pload_soft_dis->sd_list[i++],
125             plist->name, CRYPTO_MAX_MECH_NAME);
126         plist = plist->next;
127     }
129     return (pload_soft_dis);
130 }

133 /*
134  * Prepare the argument for the LOAD_DEV_DISABLED ioctl call for the
135  * provider pointed by pent. Return NULL if out of memory.
136  */
137 crypto_load_dev_disabled_t *
138 setup_dev_dis(entry_t *pent)
139 {
140     crypto_load_dev_disabled_t *pload_dev_dis = NULL;
141     mechlist_t *plist = NULL;
142     crypto_load_dev_disabled_t *pload_dev_dis;
143     mechlist_t *plist;
144     size_t extra_mech_size = 0;
145     uint_t dis_count;
146     int i;
147     char pname[MAXNAMELEN];
148     int inst_num;

149     if (pent == NULL) {
150         return (NULL);
151     }

152     /* get the device name and the instance number */
153     if (split_hw_provname(pent->name, pname, &inst_num) == FAILURE) {
154         return (NULL);
155     }

156     /* allocate space for pload_dev_dis */
157     dis_count = pent->dis_count;
158     if (dis_count > 1) {
159         extra_mech_size = sizeof (crypto_mech_name_t) *
160             (dis_count - 1);
161     }

162     pload_dev_dis = malloc(sizeof (crypto_load_dev_disabled_t) +
163         extra_mech_size);
164     if (pload_dev_dis == NULL) {
165         cryptodebug("out of memory.");
166         return (NULL);
167     }

168     /* set the values for pload_dev_dis */
169     (void) strcpy(pload_dev_dis->dd_dev_name, pname, MAXNAMELEN);
170     pload_dev_dis->dd_dev_instance = inst_num;
171     pload_dev_dis->dd_count = dis_count;

172     i = 0;
173     plist = pent->dislist;
174     while (i < dis_count) {
175         (void) strcpy(pload_dev_dis->dd_list[i++],
176             plist->name, CRYPTO_MAX_MECH_NAME);
177         plist = plist->next;
178     }

```

```

184         return (pload_dev_dis);
185     }
186     _____unchanged_portion_omitted_____

213 /*
214  * Prepare the calling argument for the GET_SOFT_INFO call for the provider
215  * with the number of mechanisms specified in the second argument.
216  *
217  * Called by get_soft_info().
218  */
219 static crypto_get_soft_info_t *
220 setup_get_soft_info(char *provname, int count)
221 {
222     crypto_get_soft_info_t *psoft_info;
223     size_t extra_mech_size = 0;

224     if (provname == NULL) {
225         return (NULL);
226     }

227     if (count > 1) {
228         extra_mech_size = sizeof (crypto_mech_name_t) * (count - 1);
229     }

230     psoft_info = malloc(sizeof (crypto_get_soft_info_t) + extra_mech_size);
231     if (psoft_info == NULL) {
232         cryptodebug("out of memory.");
233         return (NULL);
234     }

235     (void) strcpy(psoft_info->si_name, provname, MAXNAMELEN);
236     psoft_info->si_count = count;

237     return (psoft_info);
238 }

246 /*
247  * Get the device list from kernel.
248  */
249 int
250 get_dev_list(crypto_get_dev_list_t **ppdevlist)
251 {
252     crypto_get_dev_list_t *pdevlist;
253     int fd = -1;
254     int count = DEFAULT_DEV_NUM;

255     pdevlist = malloc(sizeof (crypto_get_dev_list_t) +
256         sizeof (crypto_dev_list_entry_t) * (count - 1));
257     if (pdevlist == NULL) {
258         cryptodebug("out of memory.");
259         return (FAILURE);
260     }

261     if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDONLY)) == -1) {
262         cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
263             ADMIN_IOCTL_DEVICE, strerror(errno));
264         return (FAILURE);
265     }

266     pdevlist->dl_dev_count = count;
267     if (ioctl(fd, CRYPTO_GET_DEV_LIST, pdevlist) == -1) {
268         cryptodebug("CRYPTO_GET_DEV_LIST ioctl failed: %s",
269             strerror(errno));

```

```

273         free(pdevlist);
274         (void) close(fd);
275         return (FAILURE);
276     }

278     /* BUFFER is too small, get the number of devices and retry it. */
279     if (pdevlist->dl_return_value == CRYPTO_BUFFER_TOO_SMALL) {
280         count = pdevlist->dl_dev_count;
281         free(pdevlist);
282         pdevlist = malloc(sizeof (crypto_get_dev_list_t) +
283             sizeof (crypto_dev_list_entry_t) * (count - 1));
284         if (pdevlist == NULL) {
285             cryptodebug("out of memory.");
286             (void) close(fd);
287             return (FAILURE);
288         }

290         if (ioctl(fd, CRYPTO_GET_DEV_LIST, pdevlist) == -1) {
291             cryptodebug("CRYPTO_GET_DEV_LIST ioctl failed: %s",
292                 strerror(errno));
293             free(pdevlist);
294             (void) close(fd);
295             return (FAILURE);
296         }
297     }

299     if (pdevlist->dl_return_value != CRYPTO_SUCCESS) {
300         cryptodebug("CRYPTO_GET_DEV_LIST ioctl failed, "
301             "return_value = %d", pdevlist->dl_return_value);
302         free(pdevlist);
303         (void) close(fd);
304         return (FAILURE);
305     }

307     *ppdevlist = pdevlist;
308     (void) close(fd);
309     return (SUCCESS);
310 }

313 /*
314  * Get all the mechanisms supported by the hardware provider.
315  * The result will be stored in the second argument.
316  */
317 int
318 get_dev_info(char *devname, int inst_num, int count, mechlist_t **ppmechlist)
319 {
320     crypto_get_dev_info_t *dev_info;
321     mechlist_t *phead;
322     mechlist_t *pcur;
323     mechlist_t *pmech;
324     int fd = -1;
325     int rc;
326     int i;

328     if (devname == NULL || count < 1) {
329         cryptodebug("get_dev_info(): devname is NULL or bogus count");
330         return (FAILURE);
331     }

333     /* Set up the argument for the CRYPTO_GET_DEV_INFO ioctl call */
334     dev_info = malloc(sizeof (crypto_get_dev_info_t) +
335         sizeof (crypto_mech_name_t) * (count - 1));
336     if (dev_info == NULL) {
337         cryptodebug("out of memory.");

```

```

338         return (FAILURE);
339     }
340     (void) strncpy(dev_info->di_dev_name, devname, MAXNAMELEN);
341     dev_info->di_dev_instance = inst_num;
342     dev_info->di_count = count;

344     /* Open the ioctl device */
345     if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDONLY)) == -1) {
346         cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
347             ADMIN_IOCTL_DEVICE, strerror(errno));
348         free(dev_info);
349         return (FAILURE);
350     }

352     if (ioctl(fd, CRYPTO_GET_DEV_INFO, dev_info) == -1) {
353         cryptodebug("CRYPTO_GET_DEV_INFO ioctl failed: %s",
354             strerror(errno));
355         free(dev_info);
356         (void) close(fd);
357         return (FAILURE);
358     }

360     if (dev_info->di_return_value != CRYPTO_SUCCESS) {
361         cryptodebug("CRYPTO_GET_DEV_INFO ioctl failed, "
362             "return_value = %d", dev_info->di_return_value);
363         free(dev_info);
364         (void) close(fd);
365         return (FAILURE);
366     }

368     phead = pcur = NULL;
369     rc = SUCCESS;
370     for (i = 0; i < dev_info->di_count; i++) {
371         pmech = create_mech(&dev_info->di_list[i][0]);
372         if (pmech == NULL) {
373             rc = FAILURE;
374             break;
375         } else {
376             if (phead == NULL) {
377                 phead = pcur = pmech;
378             } else {
379                 pcur->next = pmech;
380                 pcur = pmech;
381             }
382         }
383     }

385     if (rc == SUCCESS) {
386         *ppmechlist = phead;
387     } else {
388         free_mechlist(phead);
389     }

391     free(dev_info);
392     (void) close(fd);
393     return (rc);
394 }

397 /*
398  * Get the supported mechanism list of the software provider from kernel.
399  *
400  * Parameters pheadlist and psoftlist are supplied by get_kcfconf_info().
401  * If NULL, this function calls get_kcfconf_info() internally.
402  */

```



```

403 int
404 get_soft_info(char *provname, mechlist_t **ppmechlist,
405               entrylist_t *phardlist, entrylist_t *psoftlist)
406 get_soft_info(char *provname, mechlist_t **ppmechlist)
407 {
408     boolean_t    in_kernel = B_FALSE;
409     crypto_get_soft_info_t *psoft_info;
410     mechlist_t   *phead;
411     mechlist_t   *pmech;
412     mechlist_t   *pcur;
413     entry_t      *pent = NULL;
414     entry_t      *pent;
415     int          count;
416     int          fd = -1;
417     int          rc;
418     int          i;
419
420     if (provname == NULL) {
421         return (FAILURE);
422     }
423
424     if (getzoneid() == GLOBAL_ZONEID) {
425         /* use kcf.conf for kernel software providers in global zone */
426         if ((pent = getent_kef(provname, phardlist, psoftlist)) ==
427             NULL) {
428             /* No kcf.conf entry for this provider */
429             if (check_kernel_for_soft(provname, NULL, &in_kernel)
430                 == FAILURE) {
431                 if ((pent = getent_kef(provname)) == NULL) {
432                     cryptoerror(LOG_STDERR, gettext("%s does not exist."),
433                                 provname);
434                     return (FAILURE);
435                 } else if (in_kernel == B_FALSE) {
436                     cryptoerror(LOG_STDERR,
437                                 gettext("%s does not exist."), provname);
438                     return (FAILURE);
439                 }
440             }
441             /*
442              * Set mech count to 1. It will be reset to the
443              * correct value later if the setup buffer is too small.
444              */
445             count = 1;
446         } else {
447             count = pent->sup_count;
448             free_entry(pent);
449         }
450     } else {
451         /*
452          * kcf.conf not there in non-global zone: set mech count to 1.
453          * It will be reset to the correct value later if the setup
454          * buffer is too small.
455          * kcf.conf not there in non-global zone, set mech count to 1;
456          * it will be reset to the correct value later if the setup
457          * buffer is too small
458          */
459         count = 1;
460     }
461
462     if ((psoft_info = setup_get_soft_info(provname, count)) == NULL) {
463         return (FAILURE);
464     }
465
466     if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDONLY)) == -1) {

```

```

467         cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
468                     ADMIN_IOCTL_DEVICE, strerror(errno));
469         free(psoft_info);
470         return (FAILURE);
471     }
472
473     /* make GET_SOFT_INFO ioctl call */
474     if ((rc = ioctl(fd, CRYPTO_GET_SOFT_INFO, psoft_info)) == -1) {
475         cryptodebug("CRYPTO_GET_SOFT_INFO ioctl failed: %s",
476                     strerror(errno));
477         (void) close(fd);
478         free(psoft_info);
479         return (FAILURE);
480     }
481
482     /* BUFFER is too small, get the number of mechanisms and retry it. */
483     if (psoft_info->si_return_value == CRYPTO_BUFFER_TOO_SMALL) {
484         count = psoft_info->si_count;
485         free(psoft_info);
486         if ((psoft_info = setup_get_soft_info(provname, count))
487             == NULL) {
488             (void) close(fd);
489             return (FAILURE);
490         } else {
491             rc = ioctl(fd, CRYPTO_GET_SOFT_INFO, psoft_info);
492             if (rc == -1) {
493                 cryptodebug("CRYPTO_GET_SOFT_INFO ioctl "
494                             "failed: %s", strerror(errno));
495                 (void) close(fd);
496                 free(psoft_info);
497                 return (FAILURE);
498             }
499         }
500     }
501
502     (void) close(fd);
503     if (psoft_info->si_return_value != CRYPTO_SUCCESS) {
504         cryptodebug("CRYPTO_GET_SOFT_INFO ioctl failed, "
505                     "return_value = %d", psoft_info->si_return_value);
506         free(psoft_info);
507         return (FAILURE);
508     }
509
510     /* Build the mechanism linked list and return it */
511     /* Get the mechanism list and return it */
512     rc = SUCCESS;
513     phead = pcur = NULL;
514     for (i = 0; i < psoft_info->si_count; i++) {
515         pmech = create_mech(&psoft_info->si_list[i][0]);
516         if (pmech == NULL) {
517             rc = FAILURE;
518             break;
519         } else {
520             if (phead == NULL) {
521                 phead = pcur = pmech;
522             } else {
523                 pcur->next = pmech;
524                 pcur = pmech;
525             }
526         }
527     }
528
529     if (rc == FAILURE) {
530         free_mechlist(phead);
531     } else {

```

```

525         *ppmechlist = phead;
526     }

528     free(psoft_info);
529     return (rc);
530 }

533 /*
534  * Get the kernel software provider list from kernel.
535  */
536 int
537 get_soft_list(crypto_get_soft_list_t **ppsoftlist)
538 {
539     crypto_get_soft_list_t *psoftlist = NULL;
540     int count = DEFAULT_SOFT_NUM;
541     int len;
542     int fd = -1;
543     int fd;

544     if ((fd = open(ADMIN_IOCTL_DEVICE, O_RDONLY)) == -1) {
545         cryptoerror(LOG_STDERR, gettext("failed to open %s: %s"),
546             ADMIN_IOCTL_DEVICE, strerror(errno));
547         return (FAILURE);
548     }

549     len = MAXNAMELEN * count;
550     psoftlist = malloc(sizeof (crypto_get_soft_list_t) + len);
551     if (psoftlist == NULL) {
552         cryptodebug("out of memory.");
553         (void) close(fd);
554         return (FAILURE);
555     }
556     psoftlist->sl_soft_names = (caddr_t)(psoftlist + 1);
557     psoftlist->sl_soft_count = count;
558     psoftlist->sl_soft_len = len;

559     if (ioctl(fd, CRYPTO_GET_SOFT_LIST, psoftlist) == -1) {
560         cryptodebug("CRYPTO_GET_SOFT_LIST ioctl failed: %s",
561             strerror(errno));
562         free(psoftlist);
563         (void) close(fd);
564         return (FAILURE);
565     }

566     /*
567      * if BUFFER is too small, get the number of software providers and
568      * the minimum length needed for names and length and retry it.
569      */
570     if (psoftlist->sl_return_value == CRYPTO_BUFFER_TOO_SMALL) {
571         count = psoftlist->sl_soft_count;
572         len = psoftlist->sl_soft_len;
573         free(psoftlist);
574         psoftlist = malloc(sizeof (crypto_get_soft_list_t) + len);
575         if (psoftlist == NULL) {
576             cryptodebug("out of memory.");
577             (void) close(fd);
578             return (FAILURE);
579         }
580         psoftlist->sl_soft_names = (caddr_t)(psoftlist + 1);
581         psoftlist->sl_soft_count = count;
582         psoftlist->sl_soft_len = len;

583         if (ioctl(fd, CRYPTO_GET_SOFT_LIST, psoftlist) == -1) {
584             cryptodebug("CRYPTO_GET_SOFT_LIST ioctl failed:"
585                 "%s", strerror(errno));

```

```

590         free(psoftlist);
591         (void) close(fd);
592         return (FAILURE);
593     }
594 }

596     if (psoftlist->sl_return_value != CRYPTO_SUCCESS) {
597         cryptodebug("CRYPTO_GET_SOFT_LIST ioctl failed, "
598             "return_value = %d", psoftlist->sl_return_value);
599         free(psoftlist);
600         (void) close(fd);
601         return (FAILURE);
602     }

603     *ppsoftlist = psoftlist;
604     (void) close(fd);
605     return (SUCCESS);
606 }
607 }

```

unchanged_portion_omitted

```

*****
31046 Tue Oct 28 16:45:29 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/adm_kef_util.c
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #include <errno.h>
27 #include <fcntl.h>
28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <strings.h>
31 #include <time.h>
32 #include <unistd.h>
33 #include <locale.h>
34 #include <sys/types.h>
35 #include <zone.h>
36 #include <sys/stat.h>
37 #include "cryptoadm.h"

39 static int err; /* To store errno which may be overwritten by gettext() */
40 static int build_entrylist(entry_t *, entrylist_t **);
41 static entry_t *dup_entry(entry_t *);
42 static mechlist_t *dup_mechlist(mechlist_t *);
43 static entry_t *getent(char *, entrylist_t *);
44 static int interpret(char *, entry_t **);
45 static int parse_sup_dis_list(char *, entry_t *);
46 static int parse_dislist(char *, entry_t *);

48 /*
49  * Duplicate the mechanism list. A null pointer is returned if the storage
50  * space available is insufficient or the input argument is NULL.
51  */
52 static mechlist_t *
53 dup_mechlist(mechlist_t *plist)
54 {
55     mechlist_t     *pres = NULL;
56     mechlist_t     *pcur;
57     mechlist_t     *ptmp;
58     int             rc = SUCCESS;

60     while (plist != NULL) {

```

```

61         if (!(ptmp = create_mech(plist->name))) {
62             rc = FAILURE;
63             break;
64         }

66         if (pres == NULL) {
67             pres = pcur = ptmp;
68         } else {
69             pcur->next = ptmp;
70             pcur = pcur->next;
71         }
72         plist = plist->next;
73     }

75     if (rc != SUCCESS) {
76         free_mechlist(pres);
77         return (NULL);
78     }

80     return (pres);
81 }

_____unchanged_portion_omitted_____

99 /*
100  * Create one item of type entry_t with the provider name.
101  * Return NULL if there's not enough memory or provname is NULL.
102  */
103 entry_t *
104 create_entry(char *provname)
105 {
106     entry_t     *pent = NULL;

108     if (provname == NULL) {
109         return (NULL);
110     }

112     pent = calloc(1, sizeof (entry_t));
113     if (pent == NULL) {
114         cryptodebug("out of memory.");
115         return (NULL);
116     }

118     (void) strncpy(pent->name, provname, MAXNAMELEN);
119     pent->suplist = NULL;
120     pent->sup_count = 0;
121     pent->dislist = NULL;
122     pent->dis_count = 0;
123     pent->load = B_TRUE;

125     return (pent);
126 }

128 /*
129  * Duplicate an entry for a provider from kcf.conf.
130  * Return NULL if memory is insufficient or the input argument is NULL.
131  * Called by getent().
132  * Duplicate an entry. A null pointer is returned if the storage space
133  * available is insufficient or the input argument is NULL.
134  */
135 static entry_t *
136 dup_entry(entry_t *pent1)
137 {
138     entry_t *pent2 = NULL;

139     if (pent1 == NULL) {
140         return (NULL);

```

```

140     }
142     if ((pent2 = create_entry(pent1->name)) == NULL) {
143     if ((pent2 = malloc(sizeof (entry_t))) == NULL) {
144         cryptodebug("out of memory.");
145         return (NULL);
146     }
147     (void) strcpy(pent2->name, pent1->name, sizeof (pent2->name));
148     pent2->sup_count = pent1->sup_count;
149     pent2->dis_count = pent1->dis_count;
150     pent2->load = pent1->load;
151     pent2->suplist = NULL;
152     pent2->dislist = NULL;
153     if (pent1->suplist != NULL) {
154         pent2->suplist = dup_mechlist(pent1->suplist);
155         if (pent2->suplist == NULL) {
156             free_entry(pent2);
157             return (NULL);
158         }
159     }
160     if (pent1->dislist != NULL) {
161         pent2->dislist = dup_mechlist(pent1->dislist);
162         if (pent2->dislist == NULL) {
163             free_entry(pent2);
164             return (NULL);
165         }
166     }
167     return (pent2);
168 }
169 /*
170 * This routine parses the disabledlist or the supportedlist of an entry
171 * in the kcf.conf configuration file.
172 *
173 * Arguments:
174 *   buf: an input argument which is a char string with the format of
175 *       "disabledlist=m1,m2,..." or "supportedlist=m1,m2,..."
176 *   pent: the entry for the disabledlist. This is an IN/OUT argument.
177 *
178 * Return value: SUCCESS or FAILURE.
179 */
180 static int
181 parse_sup_dis_list(char *buf, entry_t *pent)
182 parse_dislist(char *buf, entry_t *pent)
183 {
184     mechlist_t    *pmech = NULL;
185     mechlist_t    *phead = NULL;
186     mechlist_t    *pmech;
187     mechlist_t    *phead;
188     char          *next_token;
189     char          *value;
190     int           count;
191     int           supflag = B_FALSE;
192     int           disflag = B_FALSE;
193     int           rc = SUCCESS;
194
195     if (strncmp(buf, EF_SUPPORTED, strlen(EF_SUPPORTED)) == 0) {
196         supflag = B_TRUE;
197     } else if (strncmp(buf, EF_DISABLED, strlen(EF_DISABLED)) == 0) {
198         disflag = B_TRUE;
199     } else {
200         /* should not come here */
201         return (FAILURE);
202     }

```

```

199     }
201     if (value = strpbrk(buf, SEP_EQUAL)) {
202         value++; /* get rid of = */
203     } else {
204         cryptodebug("failed to parse the kcf.conf file.");
205         return (FAILURE);
206     }
207
208     if ((next_token = strtok(value, SEP_COMMA)) == NULL) {
209         cryptodebug("failed to parse the kcf.conf file.");
210         return (FAILURE);
211     }
212
213     if ((pmech = create_mech(next_token)) == NULL) {
214         return (FAILURE);
215     }
216
217     if (supflag) {
218         pent->suplist = phead = pmech;
219     } else if (disflag) {
220         pent->dislist = phead = pmech;
221     }
222
223     count = 1;
224     while (next_token) {
225         if (next_token = strtok(NULL, SEP_COMMA)) {
226             if ((pmech = create_mech(next_token)) == NULL) {
227                 rc = FAILURE;
228                 break;
229             }
230             count++;
231             phead->next = pmech;
232             phead = phead->next;
233         }
234     }
235
236     if (rc == SUCCESS) {
237         if (supflag) {
238             pent->sup_count = count;
239         } else if (disflag) {
240             pent->dis_count = count;
241         }
242     } else {
243         free_mechlist(phead);
244     }
245
246     return (rc);
247 }
248
249
250 /*
251 * Convert a char string containing a line about a provider
252 * from kcf.conf into an entry_t structure.
253 *
254 * See ent2str(), the reverse of this function, for the format of
255 * kcf.conf lines.
256 * This routine converts a char string into an entry_t structure
257 */
258 static int
259 interpret(char *buf, entry_t **ppent)
260 {
261     entry_t *pent = NULL;
262     entry_t *pent;
263     char *token1;

```

```

262     char    *token2;
263     char    *token3;
264     int     rc;

266     /* Get provider name */
267     if ((token1 = strtok(buf, SEP_COLON)) == NULL) { /* buf is NULL */
268         return (FAILURE);
269     };

271     pent = create_entry(token1);
239     pent = malloc(sizeof (entry_t));
272     if (pent == NULL) {
273         cryptodebug("out of memory.");
274         return (FAILURE);
275     }
244     (void) strncpy(pent->name, token1, sizeof (pent->name));
245     pent->suplist = NULL;
246     pent->dislist = NULL;
247     pent->sup_count = 0;
248     pent->dis_count = 0;

277     if ((token2 = strtok(NULL, SEP_SEMICOLON)) == NULL) {
278         /* The entry contains a provider name only */
279         free_entry(pent);
280         return (FAILURE);
281     }

283     if (strncmp(token2, EF_UNLOAD, strlen(EF_UNLOAD)) == 0) {
284         pent->load = B_FALSE; /* cryptoadm unload */
285         if ((token2 = strtok(NULL, SEP_SEMICOLON)) == NULL) {
286             /* The entry contains a provider name:unload only */
287             free_entry(pent);
288             return (FAILURE);
289         }
290     }

292     /* need to get token3 first to satisfy nested strtok invocations */
293     token3 = strtok(NULL, SEP_SEMICOLON); /* optional */
257     token3 = strtok(NULL, SEP_SEMICOLON);

295     /* parse supportedlist (or disabledlist if no supportedlist) */
296     if ((token2 != NULL) && ((rc = parse_sup_dis_list(token2, pent)) !=
297         SUCCESS)) {
298         if (token2 && ((rc = parse_dislist(token2, pent)) != SUCCESS)) {
299             free_entry(pent);
300             return (rc);
301         }
302     }
303     /* parse disabledlist (if there's a supportedlist) */
304     if ((token3 != NULL) && ((rc = parse_sup_dis_list(token3, pent)) !=
305         SUCCESS)) {
306         if (token3 && ((rc = parse_dislist(token3, pent)) != SUCCESS)) {
307             free_entry(pent);
308             return (rc);
309         }
310     }
311     *ppent = pent;
312     return (SUCCESS);
313 }

314 /*
315 * Add an entry about a provider from kef.conf to the end of an entry list.
316 * If the entry list pplist is NULL, create the linked list with pent as the
317 * first element.
318 * Add an entry to the end of an entry list. If the entry list is NULL, will

```

```

276     * create an entry list with the pent.
277     */
278     static int
279     build_entrylist(entry_t *pent, entrylist_t **pplist)
280     {
281         entrylist_t    *pentlist;
282         entrylist_t    *pcur = NULL;
283         entrylist_t    *pcur;

285         pentlist = malloc(sizeof (entrylist_t));
286         if (pentlist == NULL) {
287             cryptodebug("out of memory.");
288             return (FAILURE);
289         }
290         pentlist->pent = pent;
291         pentlist->next = NULL;

293         if (*pplist) {
294             pcur = *pplist;
295             while (pcur->next != NULL)
296                 pcur = pcur->next;
297             pcur->next = pentlist;
298         } else { /* empty list */
299             *pplist = pentlist;
300         }

302         return (SUCCESS);
303     }

305 /*
306 * Find the entry with the "provname" name from the entry list and duplicate
307 * it. Called by getent_kef().
308 * it.
309 */
310 static entry_t *
311 getent(char *provname, entrylist_t *entrylist)
312 {
313     boolean_t    found = B_FALSE;
314     entry_t      *pent1 = NULL;

316     if ((provname == NULL) || (entrylist == NULL)) {
317         return (NULL);
318     }

320     while (!found && entrylist) {
321         if (strcmp(entrylist->pent->name, provname) == 0) {
322             found = B_TRUE;
323             pent1 = entrylist->pent;
324         } else {
325             entrylist = entrylist->next;
326         }
327     }

329     if (!found) {
330         return (NULL);
331     }

333     /* duplicate the entry to be returned */
334     return (dup_entry(pent1));
335 }

337 /*
338 * Free memory in entry_t.

```

```

381 * That is, the supported and disabled lists for a provider
382 * from kcf.conf.
383 */
384 void
385 free_entry(entry_t *pent)
386 {
387     if (pent == NULL) {
388         return;
389     } else {
390         free_mechlist(pent->suplist);
391         free_mechlist(pent->dislist);
392         free(pent);
393     }
394 }

397 /*
398 * Free elements in a entrylist_t linked list,
399 * which lists providers in kcf.conf.
400 */
401 void
402 free_entrylist(entrylist_t *entrylist)
403 {
404     entrylist_t *pNext;

406     while (entrylist != NULL) {
407         pNext = entrylist->next;
408         free_entry(entrylist->pent);
409         entrylist = pNext;
410     }
411 }

414 /*
415 * Convert an entry to a string. This routine builds a string for the entry
416 * to be inserted in the kcf.conf file. Based on the content of each entry,
417 * the result string can be one of these 6 forms:
418 * to be inserted in the config file. Based on the content of each entry,
419 * the result string can be one of the 4 forms:
420 * - name
421 * - name:supportedlist=m1,m2,...,mj
422 * - name:disabledlist=m1,m2,...,mj
423 * - name:supportedlist=m1,...,mj;disabledlist=m1,m2,...,mk
424 * - name:unload;supportedlist=m1,m2,...,mj
425 * - name:unload;disabledlist=m1,m2,...,mj
426 * - name:unload;supportedlist=m1,...,mj;disabledlist=m1,m2,...,mk
427 * Note that the caller is responsible for freeing the returned string
428 * (with free_entry()).
429 * See interpret() for the reverse of this function: converting a string
430 * to an entry_t.
431 * Note that the caller is responsible for freeing the returned string.
432 */
433 char *
434 ent2str(entry_t *pent)
435 {
436     char *buf;
437     mechlist_t *pcur = NULL;
438     boolean_t semicolon_separator = B_FALSE;
439     mechlist_t *phead;
440     boolean_t supflag = B_FALSE;

```

```

440         return (NULL);
441     }

443     if ((buf = malloc(BUFSIZ)) == NULL) {
444         return (NULL);
445     }

447     /* convert the provider name */
448     if (strncpy(buf, pent->name, BUFSIZ) >= BUFSIZ) {
449         free(buf);
450         return (NULL);
451     }

453     if (!pent->load) { /* add "unload" keyword */
454         /* convert the supported list if any */
455         phead = pent->suplist;
456         if (phead != NULL) {
457             supflag = B_TRUE;
458         }

459         if (strcat(buf, SEP_COLON, BUFSIZ) >= BUFSIZ) {
460             free(buf);
461             return (NULL);
462         }

463         if (strcat(buf, EF_UNLOAD, BUFSIZ) >= BUFSIZ) {
464             free(buf);
465             return (NULL);
466         }

467         semicolon_separator = B_TRUE;
468     }

469     /* convert the supported list if any */
470     pcur = pent->suplist;
471     if (pcur != NULL) {
472         if (strcat(buf,
473             semicolon_separator ? SEP_SEMICOLON : SEP_COLON,
474             BUFSIZ) >= BUFSIZ) {
475             free(buf);
476             return (NULL);
477         }

478         if (strcat(buf, EF_SUPPORTED, BUFSIZ) >= BUFSIZ) {
479             free(buf);
480             return (NULL);
481         }

482         while (pcur != NULL) {
483             if (strcat(buf, pcur->name, BUFSIZ) >= BUFSIZ) {
484                 while (phead != NULL) {
485                     if (strcat(buf, phead->name, BUFSIZ) >= BUFSIZ) {
486                         free(buf);
487                         return (NULL);
488                     }
489                 }
490             }

491             pcur = pcur->next;
492             if (pcur != NULL) {
493                 phead = phead->next;
494                 if (phead != NULL) {
495                     if (strcat(buf, SEP_COMMA, BUFSIZ)
496                         >= BUFSIZ) {
497                         free(buf);
498                         return (NULL);
499                     }
500                 }
501             }
502         }
503     }

```

```

497     semicolon_separator = B_TRUE;
498 }

500 /* convert the disabled list if any */
501 pcur = pent->dislist;
502 if (pcur != NULL) {
503     if (strlcat(buf,
504               semicolon_separator ? SEP_SEMICOLON : SEP_COLON,
505               BUFSIZ) >= BUFSIZ) {
431     phead = pent->dislist;
432     if (phead != NULL) {
433         if (supflag) {
434             if (strlcat(buf, ";disabledlist=", BUFSIZ) >= BUFSIZ) {
506                 free(buf);
507                 return (NULL);
508             }
510             if (strlcat(buf, EF_DISABLED, BUFSIZ) >= BUFSIZ) {
438             } else {
439                 if (strlcat(buf, ":disabledlist=", BUFSIZ) >= BUFSIZ) {
511                     free(buf);
512                     return (NULL);
513                 }
443             }
515         while (pcur != NULL) {
516             if (strlcat(buf, pcur->name, BUFSIZ) >= BUFSIZ) {
445             while (phead != NULL) {
446                 if (strlcat(buf, phead->name, BUFSIZ) >= BUFSIZ) {
517                     free(buf);
518                     return (NULL);
519                 }
521             pcur = pcur->next;
522             if (pcur != NULL) {
451                 phead = phead->next;
452                 if (phead != NULL) {
523                     if (strlcat(buf, SEP_COMMA, BUFSIZ)
524                         >= BUFSIZ) {
525                         free(buf);
526                         return (NULL);
527                     }
528                 }
529             }
530         } semicolon_separator = B_TRUE;
531     }

533     if (strlcat(buf, "\n", BUFSIZ) >= BUFSIZ) {
534         free(buf);
535         return (NULL);
536     }

538     return (buf);
539 }

542 /*
543 * Enable the mechanisms for the provider pointed by *ppent. If allflag is
544 * TRUE, enable all. Otherwise, enable the mechanisms specified in the 3rd
545 * argument "mlist". The result will be stored in ppent also.
546 */
547 int
548 enable_mechs(entry_t **ppent, boolean_t allflag, mechlist_t *mlist)
549 {
550     entry_t     *pent;
551     mechlist_t  *phead; /* the current and resulting disabled list */

```

```

552     mechlist_t  *ptr = NULL;
553     mechlist_t  *pcur = NULL;
481     mechlist_t *ptr;
482     mechlist_t *pcur;
554     boolean_t   found;

556     pent = *ppent;
557     if (pent == NULL) {
558         return (FAILURE);
559     }

561     if (allflag) {
562         free_mechlist(pent->dislist);
563         pent->dis_count = 0;
564         pent->dislist = NULL;
565         return (SUCCESS);
566     }

568     /*
569     * for each mechanism in the to-be-enabled mechanism list,
570     * - check if it is in the current disabled list
571     * - if found, delete it from the disabled list
572     * otherwise, give a warning.
573     */
574     ptr = mlist;
575     while (ptr != NULL) {
576         found = B_FALSE;
577         phead = pcur = pent->dislist;
578         while (!found && pcur) {
579             if (strcmp(pcur->name, ptr->name) == 0) {
580                 found = B_TRUE;
581             } else {
582                 phead = pcur;
583                 pcur = pcur->next;
584             }
585         }

587         if (found) {
588             if (phead == pcur) {
589                 pent->dislist = pent->dislist->next;
590                 free(pcur);
591             } else {
592                 phead->next = pcur->next;
593                 free(pcur);
594             }
595             pent->dis_count--;
596         } else {
597             cryptoerror(LOG_STDERR, gettext(
598                 "(Warning) %1$s is either enabled already or not "
599                 "a valid mechanism for %2$s"), ptr->name,
600                 pent->name);
601         }
602         ptr = ptr->next;
603     }

605     if (pent->dis_count == 0) {
606         pent->dislist = NULL;
607     }

609     return (SUCCESS);

611 }

614 /*
615 * Determine if the kernel provider name, path, is a device

```

```

616 * (that is, it contains a slash character (e.g., "mca/0").
617 * If so, it is a hardware provider; otherwise it is a software provider.
618 */
619 boolean_t
620 is_device(char *path)
621 {
622     if (strchr(path, SEP_SLASH) != NULL) {
623         return (B_TRUE);
624     } else {
625         return (B_FALSE);
626     }
627 }

629 /*
630 * Split a hardware provider name with the "name/inst_num" format into
631 * a name and a number (e.g., split "mca/0" into "mca" instance 0).
632 * a name and a number.
633 */
634 int
635 split_hw_provname(char *provname, char *pname, int *inst_num)
636 {
637     char    name[MAXNAMELEN];
638     char    *inst_str;

639     if (provname == NULL) {
640         return (FAILURE);
641     }

642     (void) strncpy(name, provname, MAXNAMELEN);
643     if (strtok(name, "/") == NULL) {
644         return (FAILURE);
645     }

646     if ((inst_str = strtok(NULL, "/")) == NULL) {
647         return (FAILURE);
648     }

649     (void) strncpy(pname, name, MAXNAMELEN);
650     *inst_num = atoi(inst_str);

651     return (SUCCESS);
652 }

653 /*
654 * Retrieve information from kcf.conf and build a hardware device entry list
655 * and a software entry list of kernel crypto providers.
656 * This list is usually incomplete, as kernel crypto providers only have to
657 * be listed in kcf.conf if a mechanism is disabled (by cryptoadm) or
658 * if the kernel provider module is not one of the default kernel providers.
659 * The kcf.conf file is available only in the global zone.
660 * Retrieve information from kcf.conf and build a device entry list and
661 * a software entry list
662 */
663 int
664 get_kcfconf_info(entrylist_t **ppdevlist, entrylist_t **ppssoftlist)
665 {
666     FILE    *pfile = NULL;
667     FILE    *pfile;
668     char    buffer[BUFSIZ];
669     int     len;
670     entry_t *pent = NULL;
671     int     rc = SUCCESS;

```

```

672     if ((pfile = fopen(_PATH_KCF_CONF, "r")) == NULL) {
673         cryptodebug("failed to open the kcf.conf file for read only");
674         return (FAILURE);
675     }

676     *ppdevlist = NULL;
677     *ppssoftlist = NULL;
678     while (fgets(buffer, BUFSIZ, pfile) != NULL) {
679         if (buffer[0] == '#' || buffer[0] == ' ' ||
680             buffer[0] == '\n' || buffer[0] == '\t') {
681             continue; /* ignore comment lines */
682         }

683         len = strlen(buffer);
684         if (buffer[len - 1] == '\n') { /* get rid of trailing '\n' */
685             if (buffer[len - 2] == '\n') { /* get rid of trailing '\n' */
686                 len--;
687             }
688             buffer[len] = '\0';
689         }

690         if ((rc = interpret(buffer, &pent)) == SUCCESS) {
691             if (is_device(pent->name)) {
692                 rc = build_entrylist(pent, ppdevlist);
693             } else {
694                 rc = build_entrylist(pent, ppssoftlist);
695             }
696         } else {
697             cryptoerror(LOG_STDERR, gettext(
698                 "failed to parse configuration."));
699         }

700         if (rc != SUCCESS) {
701             free_entrylist(*ppdevlist);
702             free_entrylist(*ppssoftlist);
703             free_entry(pent);
704             break;
705         }
706     }

707     (void) fclose(pfile);
708     return (rc);
709 }

710 /*
711 * Retrieve information from admin device and build a device entry list and
712 * a software entry list. This is used where there is no kcf.conf, e.g., the
713 * non-global zone.
714 */
715 int
716 get_admindev_info(entrylist_t **ppdevlist, entrylist_t **ppssoftlist)
717 {
718     crypto_get_dev_list_t *pdevlist_kernel = NULL;
719     crypto_get_soft_list_t *pssoftlist_kernel = NULL;
720     char    *devname;
721     int     inst_num;
722     int     mcount;
723     mechlist_t *pmech = NULL;
724     entry_t *pent_dev = NULL, *pent_soft = NULL;
725     mechlist_t *pmech;
726     entry_t *pent = NULL;
727     int     i;
728     char    *pssoftname;
729     entrylist_t *tmp_pdev = NULL;
730     entrylist_t *tmp_psoft = NULL;
731     entrylist_t *phardlist = NULL, *ppssoftlist = NULL;

```



```

741  /*
742  * Get hardware providers
743  */
744  if (get_dev_list(&pdevlist_kernel) != SUCCESS) {
745      cryptodebug("failed to get hardware provider list from kernel");
746      return (FAILURE);
747  }

749  for (i = 0; i < pdevlist_kernel->dl_dev_count; i++) {
750      devname = pdevlist_kernel->dl_devs[i].le_dev_name;
751      inst_num = pdevlist_kernel->dl_devs[i].le_dev_instance;
752      mcount = pdevlist_kernel->dl_devs[i].le_mechanism_count;

754      pmech = NULL;
755      if (get_dev_info(devname, inst_num, mcount, &pmech) !=
756          SUCCESS) {
757          cryptodebug(
758              "failed to retrieve the mechanism list for %s/%d.",
759              devname, inst_num);
760          goto fail_out;
761      }

763      if ((pent_dev = create_entry(devname)) == NULL) {
764          if ((pent = malloc(sizeof (entry_t))) == NULL) {
765              cryptodebug("out of memory.");
766              free_mechlist(pmech);
767              goto fail_out;
768          }
769          pent_dev->suplist = pmech;
770          pent_dev->sup_count = mcount;

771          if (build_entrylist(pent_dev, &tmp_pdev) != SUCCESS) {
772              (void) strcpy(pent->name, devname, MAXNAMELEN);
773              pent->suplist = pmech;
774              pent->sup_count = mcount;
775              pent->dislist = NULL;
776              pent->dis_count = 0;

777              if (build_entrylist(pent, &tmp_pdev) != SUCCESS) {
778                  goto fail_out;
779              }

781              /* because incorporated in tmp_pdev */
782              pent = NULL;
783          }

785          free(pdevlist_kernel);
786          pdevlist_kernel = NULL;

789          /*
790          * Get software providers
791          */
792          if (getzoneid() == GLOBAL_ZONEID) {
793              if (get_kcfconf_info(&phardlist, &psoftlist) != SUCCESS) {
794                  goto fail_out;
795              }
796          }

798          if (get_soft_list(&psoftlist_kernel) != SUCCESS) {
799              cryptodebug("failed to get software provider list from kernel");
800              goto fail_out;
801          }

803          for (i = 0, psoftname = psoftlist_kernel->sl_soft_names;
804              i < psoftlist_kernel->sl_soft_count;

```

```

805          i++, psoftname = psoftname + strlen(psoftname) + 1) {
806              pmech = NULL;
807              if (get_soft_info(psoftname, &pmech, phardlist, psoftlist) !=
808                  SUCCESS) {
809                  if (get_soft_info(psoftname, &pmech) != SUCCESS) {
810                      cryptodebug(
811                          "failed to retrieve the mechanism list for %s.",
812                          psoftname);
813                      goto fail_out;
814                  }
815              }

817              if ((pent_soft = create_entry(psoftname)) == NULL) {
818                  if ((pent = malloc(sizeof (entry_t))) == NULL) {
819                      cryptodebug("out of memory.");
820                      free_mechlist(pmech);
821                      goto fail_out;
822                  }
823                  pent_soft->suplist = pmech;
824                  pent_soft->sup_count = get_mech_count(pmech);
825                  pent_soft->dislist = NULL;
826                  pent_soft->dis_count = 0;

828                  if (build_entrylist(pent_soft, &tmp_psoft) != SUCCESS) {
829                      (void) strcpy(pent->name, psoftname, MAXNAMELEN);
830                      pent->suplist = pmech;
831                      pent->sup_count = get_mech_count(pmech);
832                      pent->dislist = NULL;
833                      pent->dis_count = 0;

835                  if (build_entrylist(pent, &tmp_psoft) != SUCCESS) {
836                      goto fail_out;
837                  }
838              }

840              free(psoftlist_kernel);
841              psoftlist_kernel = NULL;

843              *ppdevlist = tmp_pdev;
844              *ppsoftlist = tmp_psoft;

846              return (SUCCESS);

848 fail_out:
849         if (pent_dev != NULL)
850             free_entry(pent_dev);
851         if (pent_soft != NULL)
852             free_entry(pent_soft);
853         if (pent != NULL)
854             free_entry(pent);

856         free_entrylist(tmp_pdev);
857         free_entrylist(tmp_psoft);

859         if (pdevlist_kernel != NULL)
860             free(pdevlist_kernel);
861         if (psoftlist_kernel != NULL)
862             free(psoftlist_kernel);

864         return (FAILURE);
865     }

867 /*
868 * Return configuration information for a kernel provider from kcf.conf.
869 * For kernel software providers return a enabled list and disabled list.
870 * For kernel hardware providers return just a disabled list.
871 *
872 * Parameters phardlist and psoftlist are supplied by get_kcfconf_info().
873 * If NULL, this function calls get_kcfconf_info() internally.

```

```

757 * Find the entry in the "kcf.conf" file with "provname" as the provider name.
758 * Return the entry if found, otherwise return NULL.
850 */
851 entry_t *
852 getent_kef(char *provname, entrylist_t *phardlist, entrylist_t *psoftlist)
853 {
854     entrylist_t *pdevlist = NULL;
855     entrylist_t *psoftlist = NULL;
856     entry_t *pent = NULL;
857     boolean_t memory_allocated = B_FALSE;
858
859     if ((phardlist == NULL) || (psoftlist == NULL)) {
860         if (get_kcfconf_info(&phardlist, &psoftlist) != SUCCESS) {
861             if (get_kcfconf_info(&pdevlist, &psoftlist) != SUCCESS) {
862                 return (NULL);
863             }
864             memory_allocated = B_TRUE;
865         }
866     }
867
868     if (is_device(provname)) {
869         pent = getent(provname, phardlist);
870     } else {
871         pent = getent(provname, pdevlist);
872     }
873
874     if (memory_allocated) {
875         free_entrylist(phardlist);
876         free_entrylist(pdevlist);
877         free_entrylist(psoftlist);
878     }
879
880     return (pent);
881 }
882
883 /*
884 * Print out the provider name and the mechanism list.
885 */
886 void
887 print_mechlist(char *provname, mechlist_t *pmechlist)
888 {
889     mechlist_t *ptr = NULL;
890     mechlist_t *ptr;
891
892     if (provname == NULL) {
893         return;
894     }
895
896     (void) printf("%s: ", provname);
897     if (pmechlist == NULL) {
898         (void) printf(gettext("No mechanisms presented.\n"));
899         return;
900     }
901
902     ptr = pmechlist;
903     while (ptr != NULL) {
904         (void) printf("%s", ptr->name);
905         ptr = ptr->next;
906         if (ptr == NULL) {
907             (void) printf("\n");
908         } else {
909             (void) printf(",");
910         }
911     }
912 }

```

```

909 /*
910 * Update the kcf.conf file based on the update mode:
911 * - If update_mode is MODIFY_MODE, modify the entry with the same name.
912 * - If not found, append a new entry to the kcf.conf file.
913 * - If update_mode is DELETE_MODE, delete the entry with the same name.
914 * - If update_mode is ADD_MODE, append a new entry to the kcf.conf file.
915 * Update the kcf.conf file based on the specified entry and the update mode.
916 * - If update_mode is MODIFY_MODE or DELETE_MODE, the entry with the same
917 * provider name will be modified or deleted.
918 * - If update_mode is ADD_MODE, this must be a hardware provider without
919 * an entry in the kcf.conf file yet. Need to locate its driver package
920 * bracket and insert an entry into the bracket.
921 */
922 int
923 update_kcfconf(entry_t *pent, int update_mode)
924 {
925     boolean_t add_it = B_FALSE;
926     boolean_t delete_it = B_FALSE;
927     boolean_t found_package = B_FALSE;
928     boolean_t found_entry = B_FALSE;
929     FILE *pfile = NULL;
930     FILE *pfile_tmp = NULL;
931     FILE *pfile;
932     FILE *pfile_tmp;
933     char buffer[BUFSIZ];
934     char buffer2[BUFSIZ];
935     char devname[MAXNAMELEN];
936     char tmpfile_name[MAXPATHLEN];
937     char *name;
938     char *str;
939     char *new_str = NULL;
940     int inst_num;
941     int rc = SUCCESS;
942
943     if (pent == NULL) {
944         cryptoerror(LOG_STDERR, gettext("internal error.));
945         return (FAILURE);
946     }
947
948     /* Check the update_mode */
949     switch (update_mode) {
950     case ADD_MODE:
951         if (update_mode == ADD_MODE) {
952             add_it = B_TRUE;
953             /* FALLTHROUGH */
954         case MODIFY_MODE:
955             /* Convert the entry a string to add to kcf.conf */
956             /* Get the hardware provider name first */
957             if (split_hw_provname(pent->name, devname, &inst_num) ==
958                 FAILURE) {
959                 return (FAILURE);
960             }
961
962             /* Convert the entry to be a string */
963             if ((new_str = ent2str(pent)) == NULL) {
964                 return (FAILURE);
965             }
966             break;
967         case DELETE_MODE:
968             } else if (update_mode == DELETE_MODE) {
969                 delete_it = B_TRUE;
970             break;
971         default:

```

```

862     } else if (update_mode != MODIFY_MODE) {
951         cryptoerror(LOG_STDERR, gettext("internal error.));
952         return (FAILURE);
953     }

955     /* Open the kcf.conf file */
956     if ((pfile = fopen(_PATH_KCF_CONF, "r+")) == NULL) {
957         err = errno;
958         cryptoerror(LOG_STDERR,
959             gettext("failed to update the configuration - %s"),
960             strerror(err));
961         cryptodebug("failed to open %s for write.", _PATH_KCF_CONF);
962         return (FAILURE);
963     }

965     /* Lock the kcf.conf file */
966     if (lockf(fileno(pfile), F_TLOCK, 0) == -1) {
967         err = errno;
968         cryptoerror(LOG_STDERR,
969             gettext("failed to update the configuration - %s"),
970             strerror(err));
971         (void) fclose(pfile);
972         return (FAILURE);
973     }

975     /*
976     * Create a temporary file in the /etc/crypto directory to save
977     * updated configuration file first.
978     */
979     (void) strncpy(tmpfile_name, TMPFILE_TEMPLATE, sizeof(tmpfile_name));
980     if (mkstemp(tmpfile_name) == -1) {
981         err = errno;
982         cryptoerror(LOG_STDERR,
983             gettext("failed to create a temporary file - %s"),
984             strerror(err));
985         (void) fclose(pfile);
986         return (FAILURE);
987     }

989     if ((pfile_tmp = fopen(tmpfile_name, "w")) == NULL) {
990         err = errno;
991         cryptoerror(LOG_STDERR, gettext("failed to open %s - %s"),
992             tmpfile_name, strerror(err));
993         (void) fclose(pfile);
994         return (FAILURE);
995     }

997     /*
998     * Loop thru the entire kcf.conf file, insert, modify or delete
999     * an entry.
1000     */
1001     while (fgets(buffer, BUFSIZ, pfile) != NULL) {
1002         if (add_it) {
1003             /* always keep the current line */
1004             if (fputs(buffer, pfile_tmp) == EOF) {
1005                 err = errno;
1006                 cryptoerror(LOG_STDERR, gettext(
1007                     "failed to write to a temp file: %s."),
1008                     strerror(err));
1009                 rc = FAILURE;
1010                 break;
1011             }

1012             /*
1013             * If the current position is the beginning of a driver

```

```

928     * package and if the driver name matches the hardware
929     * provider name, then we want to insert the entry
930     * here.
931     */
932     if ((strstr(buffer, HW_DRIVER_STRING) != NULL) &&
933         (strstr(buffer, devname) != NULL)) {
934         found_package = B_TRUE;
935         if (fputs(new_str, pfile_tmp) == EOF) {
936             err = errno;
937             cryptoerror(LOG_STDERR, gettext(
938                 "failed to write to a temp file: "
939                 "%s."), strerror(err));
940             rc = FAILURE;
941             break;
942         }
943     }
1012     } else { /* modify or delete */
1013         found_entry = B_FALSE;

1015         if (!(buffer[0] == '#' || buffer[0] == ' ' ||
1016             buffer[0] == '\n' || buffer[0] == '\t')) {
1017             /*
1018             * Get the provider name from this line and
1019             * check if this is the entry to be updated
1020             * or deleted. Note: can not use "buffer"
1021             * directly because strtok will change its
1022             * value.
1023             */
1024             (void) strncpy(buffer2, buffer, BUFSIZ);
1025             if ((name = strtok(buffer2, SEP_COLON)) ==
1026                 NULL) {
1027                 rc = FAILURE;
1028                 break;
1029             }

1031             if (strcmp(pent->name, name) == 0) {
1032                 found_entry = B_TRUE;
1033             }
1034         }

1036         if (found_entry && !delete_it) {
1037             /*
1038             * This is the entry to be updated; get the
1039             * updated string and place into buffer.
1040             */
1041             (void) strncpy(buffer, new_str, BUFSIZ);
1042             free(new_str);
1043             if ((str = ent2str(pent)) == NULL) {
1044                 rc = FAILURE;
1045                 break;
1046             } else {
1047                 (void) strncpy(buffer, str, BUFSIZ);
1048                 free(str);
1049             }
1050         }

1051         if (!(found_entry && delete_it)) {
1052             /* This is the entry to be updated/reserved */
1053             if (fputs(buffer, pfile_tmp) == EOF) {
1054                 err = errno;
1055                 cryptoerror(LOG_STDERR, gettext(
1056                     "failed to write to a temp file: "
1057                     "%s."), strerror(err));
1058                 rc = FAILURE;
1059                 break;
1060             }

```

```

1055     }
1056     }
1057 }

1059 if ((!delete_it) && (rc != FAILURE)) {
1060     if (add_it || !found_entry) {
1061         /* append new entry to end of file */
1062         if (fputs(new_str, pfile_tmp) == EOF) {
1063             if (add_it) {
1064                 free(new_str);
1065             }
1066
1069             if ((add_it && !found_package) || (rc == FAILURE)) {
1070                 if (add_it && !found_package) {
1071                     cryptoerror(LOG_STDERR,
1072                                gettext("failed to update configuration - no "
1073                                         "driver package information."));
1074                 }
1075
1076                 (void) fclose(pfile);
1077                 (void) fclose(pfile_tmp);
1078                 if (unlink(tmpfile_name) != 0) {
1079                     err = errno;
1080                     cryptoerror(LOG_STDERR, gettext(
1081                                "failed to write to a temp file: %s."),
1082                                strerror(err));
1083                     rc = FAILURE;
1084                     gettext("Warning) failed to remove %s: %s"),
1085                             tmpfile_name, strerror(err));
1086                 }
1087                 free(new_str);
1088                 return (FAILURE);
1089             }
1090         }
1091     }
1092
1093     (void) fclose(pfile);
1094     if (fclose(pfile_tmp) != 0) {
1095         err = errno;
1096         cryptoerror(LOG_STDERR,
1097                    gettext("failed to close %s: %s"), tmpfile_name,
1098                    strerror(err));
1099         return (FAILURE);
1100     }
1101 }

1102 /* Copy the temporary file to the kcf.conf file */
1103 if (rename(tmpfile_name, _PATH_KCF_CONF) == -1) {
1104     err = errno;
1105     cryptoerror(LOG_STDERR,
1106                gettext("failed to update the configuration - %s"),
1107                strerror(err));
1108     cryptodebug("failed to rename %s to %s: %s", tmpfile,
1109                _PATH_KCF_CONF, strerror(err));
1110     rc = FAILURE;
1111 } else if (chmod(_PATH_KCF_CONF,
1112                S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) == -1) {
1113     err = errno;
1114     cryptoerror(LOG_STDERR,
1115                gettext("failed to update the configuration - %s"),
1116                strerror(err));
1117     cryptodebug("failed to chmod to %s: %s", _PATH_KCF_CONF,
1118                strerror(err));
1119     rc = FAILURE;
1120 } else {
1121     rc = SUCCESS;
1122 }

```

```

1104     if ((rc == FAILURE) && (unlink(tmpfile_name) != 0)) {
1105         err = errno;
1106         cryptoerror(LOG_STDERR, gettext(
1107                    "(Warning) failed to remove %s: %s"),
1108                    tmpfile_name, strerror(err));
1109     }
1110
1111     return (rc);
1112 }

1115 /*
1116  * Disable the mechanisms for the provider pointed by *ppent. If allflag is
1117  * TRUE, disable all. Otherwise, disable the mechanisms specified in the
1118  * dislist argument. The "infolist" argument contains the mechanism list
1119  * supported by this provider.
1120  */
1121 int
1122 disable_mechs(entry_t **ppent, mechlist_t *infolist, boolean_t allflag,
1123 mechlist_t *dislist)
1124 {
1125     entry_t *pent;
1126     mechlist_t *plist = NULL;
1127     mechlist_t *phead = NULL;
1128     mechlist_t *pmech = NULL;
1129     mechlist_t *plist;
1130     mechlist_t *phead;
1131     mechlist_t *pmech;
1132     int rc = SUCCESS;
1133
1134     pent = *ppent;
1135     if (pent == NULL) {
1136         return (FAILURE);
1137     }
1138
1139     if (allflag) {
1140         free_mechlist(pent->dislist);
1141         pent->dis_count = get_mech_count(infolist);
1142         if (!(pent->dislist = dup_mechlist(infolist))) {
1143             return (FAILURE);
1144         } else {
1145             return (SUCCESS);
1146         }
1147     }
1148
1149     /*
1150      * Not disable all. Now loop thru the mechanisms specified in the
1151      * dislist. If the mechanism is not supported by the provider,
1152      * ignore it with a warning. If the mechanism is disabled already,
1153      * do nothing. Otherwise, prepend it to the beginning of the disabled
1154      * list of the provider.
1155      */
1156     plist = dislist;
1157     while (plist != NULL) {
1158         if (!is_in_list(plist->name, infolist)) {
1159             cryptoerror(LOG_STDERR, gettext("(Warning) "
1160                                            "%1$s is not a valid mechanism for %2$s."),
1161                        plist->name, pent->name);
1162         } else if (!is_in_list(plist->name, pent->dislist)) {
1163             /* Add this mechanism into the disabled list */
1164             if ((pmech = create_mech(plist->name)) == NULL) {
1165                 rc = FAILURE;
1166                 break;
1167             }
1168         }
1169     }
1170
1171     if (pent->dislist == NULL) {

```

```

1167         pent->dislist = pmech;
1168     } else {
1169         phead = pent->dislist;
1170         pent->dislist = pmech;
1171         pmech->next = phead;
1172     }
1173     pent->dis_count++;
1174 }
1175     plist = plist->next;
1176 }

1178     return (rc);
1179 }
    unchanged_portion_omitted

1224 /*
1225  * Print out the mechanism policy for a kernel provider that has an entry
1226  * in the kcf.conf file.
1227  *
1228  * The flag has_random is set to B_TRUE if the provider does random
1229  * numbers. The flag has_mechs is set by the caller to B_TRUE if the provider
1230  * has some mechanisms.
1231  *
1232  * If pent is NULL, the provider doesn't have a kcf.conf entry.
1233  */
1234 void
1235 print_kef_policy(char *provname, entry_t *pent, boolean_t has_random,
1236                boolean_t has_mechs)
1237 print_kef_policy(entry_t *pent, boolean_t has_random, boolean_t has_mechs)
1238 {
1239     mechlist_t *ptr = NULL;
1240     mechlist_t *ptr;
1241     boolean_t rnd_disabled = B_FALSE;

1242     if (pent != NULL) {
1243         if (pent == NULL) {
1244             return;
1245         }

1246         rnd_disabled = filter_mechlist(&pent->dislist, RANDOM);
1247         ptr = pent->dislist;
1248     }

1249     (void) printf("%s:", provname);
1250     (void) printf("%s:", pent->name);

1251     if (has_mechs == B_TRUE) {
1252         /*
1253          * TRANSLATION_NOTE
1254          * This code block may need to be modified a bit to avoid
1255          * constructing the text message on the fly.
1256          */
1257         (void) printf(gettext(" all mechanisms are enabled"));
1258         if (ptr != NULL)
1259             (void) printf(gettext(", except "));
1260         while (ptr != NULL) {
1261             (void) printf("%s", ptr->name);
1262             ptr = ptr->next;
1263             if (ptr != NULL)
1264                 (void) printf(",");
1265         }
1266         if (ptr == NULL)
1267             (void) printf(".");
1268     }

```

```

1267     /*
1268      * TRANSLATION_NOTE
1269      * "random" is a keyword and not to be translated.
1270      */
1271     if (rnd_disabled)
1272         (void) printf(gettext(" %s is disabled."), "random");
1273     else if (has_random)
1274         (void) printf(gettext(" %s is enabled."), "random");
1275     (void) printf("\n");
1276 }

1279 /*
1280  * Check if a kernel software provider is in the kernel.
1281  *
1282  * Parameters:
1283  * provname      Provider name
1284  * psoftlist_kernel  Optional software provider list. If NULL, it will be
1285  *                  obtained from get_soft_list().
1286  * in_kernel     Set to B_TRUE if device is in the kernel, else B_FALSE
1287  */
1288 int
1289 check_kernel_for_soft(char *provname, crypto_get_soft_list_t *psoftlist_kernel,
1290                      boolean_t *in_kernel)
1291 check_active_for_soft(char *provname, boolean_t *is_active)
1292 {
1293     crypto_get_soft_list_t *psoftlist_kernel = NULL;
1294     char *ptr;
1295     int i;
1296     boolean_t psoftlist_allocated = B_FALSE;

1297     if (provname == NULL) {
1298         cryptoerror(LOG_STDERR, gettext("internal error."));
1299         return (FAILURE);
1300     }

1301     if (psoftlist_kernel == NULL) {
1302         if (get_soft_list(&psoftlist_kernel) == FAILURE) {
1303             cryptodebug("failed to get the software provider list"
1304                       " from kernel.");
1305             cryptodebug("failed to get the software provider list from"
1306                       "kernel.");
1307             return (FAILURE);
1308         }
1309         psoftlist_allocated = B_TRUE;
1310     }

1311     *in_kernel = B_FALSE;
1312     *is_active = B_FALSE;
1313     ptr = psoftlist_kernel->sl_soft_names;
1314     for (i = 0; i < psoftlist_kernel->sl_soft_count; i++) {
1315         if (strcmp(provname, ptr) == 0) {
1316             *in_kernel = B_TRUE;
1317             *is_active = B_TRUE;
1318             break;
1319         }
1320         ptr = ptr + strlen(ptr) + 1;
1321     }

1322     if (psoftlist_allocated)
1323         free(psoftlist_kernel);

1324     return (SUCCESS);
1325 }

```

```

1327 /*
1328  * Check if a kernel hardware provider is in the kernel.
1329  *
1330  * Parameters:
1331  * provname    Provider name
1332  * pdevlist   Optional Hardware Crypto Device List.  If NULL, it will be
1333  *            obtained from get_dev_list().
1334  * in_kernel   Set to B_TRUE if device is in the kernel, otherwise B_FALSE
1335  */
1336 int
1337 check_kernel_for_hard(char *provname,
1338                      crypto_get_dev_list_t *pdevlist, boolean_t *in_kernel)
1261 check_active_for_hard(char *provname, boolean_t *is_active)
1339 {
1263     crypto_get_dev_list_t *pdevlist = NULL;
1340     char devname[MAXNAMELEN];
1341     int inst_num;
1342     int i;
1343     boolean_t dev_list_allocated = B_FALSE;

1345     if (provname == NULL) {
1346         cryptoerror(LOG_STDERR, gettext("internal error.));
1347         return (FAILURE);
1348     }

1350     if (split_hw_provname(provname, devname, &inst_num) == FAILURE) {
1351         return (FAILURE);
1352     }

1354     if (pdevlist == NULL) {
1355         if (get_dev_list(&pdevlist) == FAILURE) {
1356             cryptoerror(LOG_STDERR, gettext("internal error.));
1357             return (FAILURE);
1358         }
1359         dev_list_allocated = B_TRUE;
1360     }

1362     *in_kernel = B_FALSE;
1282     *is_active = B_FALSE;
1363     for (i = 0; i < pdevlist->dl_dev_count; i++) {
1364         if ((strcmp(pdevlist->dl_devs[i].le_dev_name, devname) == 0) &&
1365             (pdevlist->dl_devs[i].le_dev_instance == inst_num)) {
1366             *in_kernel = B_TRUE;
1286             *is_active = B_TRUE;
1367             break;
1368         }
1369     }

1371     if (dev_list_allocated)
1372         free(pdevlist);

1374     return (SUCCESS);
1375 }
unchanged_portion_omitted

```

```

*****
42338 Tue Oct 28 16:45:32 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/cryptoadm.c
6414175 kcf.conf's supportedlist not providing much usefulness
*****
_____unchanged_portion_omitted_____

```

```

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 (e.g., "aes").
252 * - PROV_KEF_HARD if provname is a base name only
253 * - PROV_KEF_HARD if provname contains one slash only and the slash is not
254 *   the 1st character (e.g., "mca/0").
255 *   the 1st character.
256 * - PROV_BADNAME otherwise.
257 */
258 static int
259 get_provider_type(char *provname)
260 {
261     char *pslash1;
262     char *pslash2;
263
264     if (provname == NULL) {
265         return (FAILURE);
266     }
267
268     if (provname[0] == '/') {
269         return (PROV_UEF_LIB);
270     } else if ((pslash1 = strchr(provname, SEP_SLASH)) == NULL) {
271         /* no slash */
272         return (PROV_KEF_SOFT);
273     } else {
274         pslash2 = strrchr(provname, SEP_SLASH);
275         if (pslash1 == pslash2) {
276             return (PROV_KEF_HARD);
277         } else {
278             return (PROV_BADNAME);
279         }
280     }
281 }
_____unchanged_portion_omitted_____

```

```

532 /*
533 * The top level function for the "cryptoadm list" subcommand and options.
534 * The top level function for the list subcommand and options.
535 */
536 static int
537 do_list(int argc, char **argv)
538 {
539     boolean_t mflag = B_FALSE;
540     boolean_t pflag = B_FALSE;
541     boolean_t vflag = B_FALSE;
542     char ch;
543     cryptoadm_provider_t *prov = NULL;
544     int rc = SUCCESS;
545
546     argc --;
547     argv ++;
548
549     if (argc == 1) {
550         rc = list_simple_for_all(B_FALSE);
551         goto out;
552     }
553
554     /*
555      * cryptoadm list [-v] [-m] [-p] [provider=<>] [mechanism=<>]
556      * [-v] [-m] [-p] [provider=<>] [mechanism=<>]
557      */
558 }

```

```

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     char *provname = prov->cp_name;

636     if (mflag || vflag) {
637         if (vflag)
638             (void) printf(gettext("Provider: %s\n"),
639                 provname);
640         switch (prov->cp_type) {
641             case PROV_UEF_LIB:
642                 rc = list_mechlist_for_lib(provname,
643                     mecharglist, NULL, B_FALSE, vflag, mflag);
644                 rc = list_mechlist_for_lib(prov->cp_name,
645                     mecharglist, NULL, B_FALSE,
646                     vflag, mflag);
647                 break;
648             case PROV_KEF_SOFT:
649                 rc = list_mechlist_for_soft(provname,
650                     NULL, NULL);
651                 rc = list_mechlist_for_soft(prov->cp_name);
652                 break;
653             case PROV_KEF_HARD:
654                 rc = list_mechlist_for_hard(provname);
655                 rc = list_mechlist_for_hard(prov->cp_name);
656                 break;
657             default: /* should not come here */
658                 rc = FAILURE;
659                 break;
660         }
661     } else if (pflag) {
662         switch (prov->cp_type) {
663             case PROV_UEF_LIB:
664                 rc = list_policy_for_lib(provname);
665                 rc = list_policy_for_lib(prov->cp_name);
666                 break;
667             case PROV_KEF_SOFT:
668                 if (getzoneid() == GLOBAL_ZONEID) {
669                     rc = list_policy_for_soft(provname,
670                         NULL, NULL);
671                     rc = list_policy_for_soft(
672                         prov->cp_name);
673                 } else {
674                     /*
675                      * TRANSLATION_NOTE
676                      * "global" is keyword and not to
677                      * be translated.
678                      */
679                     cryptoerror(LOG_STDERR, gettext(
680                         "policy information for kernel "
681                         "providers is available "
682                         "in the %s zone only"), "global");
683                     rc = FAILURE;
684                 }
685             }
686         }
687     }
688     break;

```

```

678         case PROV_KEF_HARD:
679             if (getzoneid() == GLOBAL_ZONEID) {
680                 rc = list_policy_for_hard(
681                     provname, NULL, NULL, NULL);
682                 prov->cp_name;
683             } else {
684                 /*
685                  * TRANSLATION_NOTE
686                  * "global" is keyword and not to
687                  * be translated.
688                  */
689                 cryptoerror(LOG_STDERR, gettext(
690                     "policy information for kernel "
691                     "providers is available "
692                     "in the %s zone only"), "global");
693                 rc = FAILURE;
694             }
695         }
696         break;
697     default: /* should not come here */
698         rc = FAILURE;
699         break;
700     } else {
701         /* error message */
702         usage();
703         rc = ERROR_USAGE;
704     }
705 }

707 out:
708     if (prov != NULL)
709         free(prov);

711     if (mecharglist != NULL)
712         free_mechlist(mecharglist);
713     return (rc);
714 }

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

727     if ((argc < 3) || (argc > 5)) {
728         usage();
729         return (ERROR_USAGE);
730     }

732     prov = get_provider(argc, argv);
733     if (prov == NULL) {
734         usage();
735         return (ERROR_USAGE);
736     }
737     if (prov->cp_type == PROV_BADNAME) {
738         return (FAILURE);
739     }

741     if ((rc = process_feature_operands(argc, argv)) != SUCCESS) {

```



```

742         goto out;
743     }

745     /*
746     * If allflag or rndflag has already been set there is no reason to
747     * process mech=
748     */
749     if (prov->cp_type == METASLOT) {
750         if ((argc > 3) &&
751             (rc = process_metaslot_operands(argc, argv,
752             NULL, NULL, NULL, &auto_key_migrate_flag)) != SUCCESS) {
753             usage();
754             return (rc);
755         }
756     } else if (!allflag && !rndflag &&
757         (rc = process_mech_operands(argc, argv, B_FALSE)) != SUCCESS) {
758         return (rc);
759     }

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

```

```

808         * "disable" could be either a literal keyword
809         * and hence not to be translated, or a verb and
810         * translatable. A choice was made to view it as
811         * a literal keyword. "global" is keyword and not
812         * to be translated.
813         */
814         cryptoerror(LOG_STDERR, gettext("%!1$s for kernel "
815         "providers is supported in the %2$s zone only"),
816             "disable", "global");
817         rc = FAILURE;
818     }
819     break;
820     default: /* should not come here */
821         rc = FAILURE;
822         break;
823     }

825     out:
826     free(prov);
827     if (mecharglist != NULL) {
828         free_mechlist(mecharglist);
829     }
830     return (rc);
831 }

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

847     if ((argc < 3) || (argc > 6)) {
848         usage();
849         return (ERROR_USAGE);
850     }

851     prov = get_provider(argc, argv);
852     if (prov == NULL) {
853         usage();
854         return (ERROR_USAGE);
855     }
856     if ((prov->cp_type != METASLOT) && (argc != 4)) {
857         usage();
858         return (ERROR_USAGE);
859     }
860     if (prov->cp_type == PROV_BADNAME) {
861         rc = FAILURE;
862         goto out;
863     }

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

```

```

872     }
873     if ((alt_slot || alt_token) && use_default) {
874         usage();
875         rc = FAILURE;
876         goto out;
877     }
878 } else {
879     if ((rc = process_feature_operands(argc, argv)) != SUCCESS) {
880         goto out;
881     }
882
883     /*
884     * If allflag or rndflag has already been set there is
885     * no reason to process mech=
886     */
887     if (!allflag && !rndflag &&
888         (rc = process_mech_operands(argc, argv, B_FALSE))
889         != SUCCESS) {
890         goto out;
891     }
892 }
893
894 switch (prov->cp_type) {
895 case METASLOT:
896     rc = enable_metaslot(alt_token, alt_slot, use_default,
897         mecharglist, allflag, auto_key_migrate_flag);
898     break;
899 case PROV_UEF_LIB:
900     rc = enable_uef_lib(prov->cp_name, rndflag, allflag,
901         mecharglist);
902     break;
903 case PROV_KEF_SOFT:
904 case PROV_KEF_HARD:
905     if (rndflag && !allflag) {
906         if ((mecharglist = create_mech(RANDOM)) == NULL) {
907             rc = FAILURE;
908             break;
909         }
910     }
911     if (getzoneid() == GLOBAL_ZONEID) {
912         rc = enable_kef(prov->cp_name, rndflag, allflag,
913             mecharglist);
914     } else {
915         /*
916         * TRANSLATION_NOTE
917         * "enable" could be either a literal keyword
918         * and hence not to be translated, or a verb and
919         * translatable. A choice was made to view it as
920         * a literal keyword. "global" is keyword and not
921         * to be translated.
922         */
923         cryptoerror(LOG_STDERR, gettext("%1$s for kernel "
924             "providers is supported in the %2$s zone only"),
925             "enable", "global");
926         rc = FAILURE;
927     }
928     break;
929 default: /* should not come here */
930     rc = FAILURE;
931     break;
932 }
933 out:
934 free(prov);
935 if (mecharglist != NULL) {
936     free_mechlist(mecharglist);
937 }

```

```

938     if (alt_token != NULL) {
939         free(alt_token);
940     }
941     if (alt_slot != NULL) {
942         free(alt_slot);
943     }
944     return (rc);
945 }
946
947 /*
948 * The top level function for the "cryptoadm install" subcommand.
949 * The top level function fo the install subcommand.
950 */
951 static int
952 do_install(int argc, char **argv)
953 {
954     cryptoadm_provider_t *prov = NULL;
955     int rc;
956
957     if (argc < 3) {
958         usage();
959         return (ERROR_USAGE);
960     }
961
962     prov = get_provider(argc, argv);
963     if (prov == NULL ||
964         prov->cp_type == PROV_BADNAME || prov->cp_type == PROV_KEF_HARD) {
965         /*
966         * TRANSLATION_NOTE
967         * "install" could be either a literal keyword and hence
968         * not to be translated, or a verb and translatable. A
969         * choice was made to view it as a literal keyword.
970         */
971         cryptoerror(LOG_STDERR,
972             gettext("bad provider name for %s."), "install");
973         rc = FAILURE;
974         goto out;
975     }
976
977     if (prov->cp_type == PROV_UEF_LIB) {
978         rc = install_uef_lib(prov->cp_name);
979         goto out;
980     }
981
982     /* It is the PROV_KEF_SOFT type now */
983
984     /* check if there are mechanism operands */
985     if (argc < 4) {
986         /*
987         * TRANSLATION_NOTE
988         * "mechanism" could be either a literal keyword and hence
989         * not to be translated, or a descriptive word and
990         * translatable. A choice was made to view it as a literal
991         * keyword.
992         */
993         cryptoerror(LOG_STDERR,
994             gettext("need %s operands for installing a "
995                 "kernel software provider."), "mechanism");
996         rc = ERROR_USAGE;
997         goto out;
998     }
999
1000     if ((rc = process_mech_operands(argc, argv, B_FALSE)) != SUCCESS) {
1001         goto out;
1002     }

```

```

1003     }
1004
1005     if (allflag == B_TRUE) {
1006         /*
1007          * TRANSLATION_NOTE
1008          * "all", "mechanism", and "install" are all keywords and
1009          * not to be translated.
1010          */
1011         cryptoerror(LOG_STDERR,
1012             gettext("can not use the %1$s keyword for %2$s "
1013                 "in the %3$s subcommand."), "all", "mechanism", "install");
1014         rc = ERROR_USAGE;
1015         goto out;
1016     }
1017
1018     if (getzoneid() == GLOBAL_ZONEID) {
1019         rc = install_kef(prov->cp_name, mecharglist);
1020     } else {
1021         /*
1022          * TRANSLATION_NOTE
1023          * "install" could be either a literal keyword and hence
1024          * not to be translated, or a verb and translatable. A
1025          * choice was made to view it as a literal keyword.
1026          * "global" is keyword and not to be translated.
1027          */
1028         cryptoerror(LOG_STDERR, gettext("%1$s for kernel providers "
1029             "is supported in the %2$s zone only"), "install", "global");
1030         rc = FAILURE;
1031     }
1032 out:
1033     free(prov);
1034     return (rc);
1035 }

```

```

1039 /*
1040 * The top level function for the "cryptoadm uninstall" subcommand.
1041 * The top level function for the uninstall subcommand.
1042 */
1043 static int
1044 do_uninstall(int argc, char **argv)
1045 {
1046     cryptoadm_provider_t *prov = NULL;
1047     int rc = SUCCESS;
1048
1049     if (argc != 3) {
1050         usage();
1051         return (ERROR_USAGE);
1052     }
1053
1054     prov = get_provider(argc, argv);
1055     if (prov == NULL ||
1056         prov->cp_type == PROV_BADNAME || prov->cp_type == PROV_KEF_HARD) {
1057         /*
1058          * TRANSLATION_NOTE
1059          * "uninstall" could be either a literal keyword and hence
1060          * not to be translated, or a verb and translatable. A
1061          * choice was made to view it as a literal keyword.
1062          */
1063         cryptoerror(LOG_STDERR,
1064             gettext("bad provider name for %s."), "uninstall");
1065         free(prov);
1066         return (FAILURE);
1067     }

```

```

1068     if (prov->cp_type == PROV_UEF_LIB) {
1069         rc = uninstall_uef_lib(prov->cp_name);
1070     } else if (prov->cp_type == PROV_KEF_SOFT) {
1071         if (getzoneid() == GLOBAL_ZONEID) {
1072             /* unload and remove from kcf.conf */
1073             rc = uninstall_kef(prov->cp_name);
1074         } else {
1075             /*
1076              * TRANSLATION_NOTE
1077              * "uninstall" could be either a literal keyword and
1078              * hence not to be translated, or a verb and
1079              * translatable. A choice was made to view it as a
1080              * literal keyword. "global" is keyword and not to
1081              * be translated.
1082              */
1083             cryptoerror(LOG_STDERR, gettext("%1$s for kernel "
1084                 "providers is supported in the %2$s zone only"),
1085                 "uninstall", "global");
1086             rc = FAILURE;
1087         }
1088     }
1089 }
1090
1091 free(prov);
1092 return (rc);
1093 }

```

```

1096 /*
1097 * The top level function for the "cryptoadm unload" subcommand.
1098 * The top level function for the unload subcommand.
1099 */
1100 static int
1101 do_unload(int argc, char **argv)
1102 {
1103     cryptoadm_provider_t *prov = NULL;
1104     entry_t *pent = NULL;
1105     boolean_t in_kernel = B_FALSE;
1106     entry_t *pent;
1107     boolean_t is_active;
1108     int rc = SUCCESS;
1109     char *provname = NULL;
1110
1111     if (argc != 3) {
1112         usage();
1113         return (ERROR_USAGE);
1114     }
1115
1116     /* check if it is a kernel software provider */
1117     prov = get_provider(argc, argv);
1118     if (prov == NULL) {
1119         cryptoerror(LOG_STDERR,
1120             gettext("unable to determine provider name."));
1121         goto out;
1122     }
1123     provname = prov->cp_name;
1124     if (prov->cp_type != PROV_KEF_SOFT) {
1125         cryptoerror(LOG_STDERR,
1126             gettext("%s is not a valid kernel software provider."),
1127             provname);
1128         free(prov);
1129         rc = FAILURE;
1130         goto out;
1131     }
1132
1133     if (getzoneid() != GLOBAL_ZONEID) {

```

```

1130      /*
1131       * TRANSLATION_NOTE
1132       * "unload" could be either a literal keyword and hence
1133       * not to be translated, or a verb and translatable.
1134       * A choice was made to view it as a literal keyword.
1135       * "global" is keyword and not to be translated.
1136       */
1137      cryptoerror(LOG_STDERR, gettext("%1$s for kernel providers "
1138      "is supported in the %2$s zone only"), "unload", "global");
1139      rc = FAILURE;
1140      goto out;
1141  }

1143  if (check_kernel_for_soft(provname, NULL, &in_kernel) == FAILURE) {
1144      cryptodebug("internal error");
1145      rc = FAILURE;
1146      goto out;
1147  } else if (in_kernel == B_FALSE) {
1148      /* Check if it is in the kcf.conf file first */
1149      if ((pent = getent_kef(prov->cp_name)) == NULL) {
1150          cryptoerror(LOG_STDERR,
1151          gettext("provider %s is not loaded or does not exist.",
1152          provname);
1153          gettext("provider %s does not exist."), prov->cp_name);
1154          rc = FAILURE;
1155          goto out;
1156      }
1157      free_entry(pent);
1158  }

1159  /* Get kcf.conf entry. If none, build a new entry */
1160  if ((pent = getent_kef(provname, NULL, NULL)) == NULL) {
1161      if ((pent = create_entry(provname)) == NULL) {
1162          cryptoerror(LOG_STDERR, gettext("out of memory."));
1163          /* If it is unloaded already, return */
1164          if (check_active_for_soft(prov->cp_name, &is_active) == FAILURE) {
1165              cryptodebug("internal error");
1166              cryptoerror(LOG_STDERR,
1167              gettext("failed to unload %s."), prov->cp_name);
1168              rc = FAILURE;
1169              goto out;
1170          }
1171      }
1172  }

1173  /* If it is unloaded already, return */
1174  if (!pent->load) { /* unloaded already */
1175      cryptoerror(LOG_STDERR,
1176      gettext("failed to unload %s."), provname);
1177      rc = FAILURE;
1178  }

1179  if (is_active == B_FALSE) { /* unloaded already */
1180      rc = SUCCESS;
1181      goto out;
1182  } else if (unload_kef_soft(provname) != FAILURE) {
1183      /* Mark as unloaded in kcf.conf */
1184      pent->load = B_FALSE;
1185      rc = update_kcfconf(pent, MODIFY_MODE);
1186  } else {
1187      if (unload_kef_soft(prov->cp_name, B_TRUE) == FAILURE) {
1188          cryptoerror(LOG_STDERR,
1189          gettext("failed to unload %s."), provname);
1190          gettext("failed to unload %s."), prov->cp_name);
1191          rc = FAILURE;
1192      } else {
1193          rc = SUCCESS;
1194      }
1195  }
1196  out:
1197  free(prov);

```

```

1181      free_entry(pent);
1182      return (rc);
1183  }

1187  /*
1188  * The top level function for the "cryptoadm refresh" subcommand.
1189  * The top level function for the refresh subcommand.
1190  */
1191  static int
1192  do_refresh(int argc)
1193  {
1194      if (argc != 2) {
1195          usage();
1196          return (ERROR_USAGE);
1197      }

1198      if (getzoneid() == GLOBAL_ZONEID) {
1199          return (refresh());
1200      } else { /* non-global zone */
1201          /*
1202           * Note: in non-global zone, this must silently return SUCCESS
1203           * due to integration with SMF, for "svcadm refresh cryptosvc"
1204           */
1205          if (getzoneid() != GLOBAL_ZONEID)
1206              return (SUCCESS);
1207      }

1208      return (refresh());
1209  }

1210  /*
1211  * The top level function for the "cryptoadm start" subcommand.
1212  * The top level function for the start subcommand.
1213  */
1214  static int
1215  do_start(int argc)
1216  {
1217      int ret;

1218      if (argc != 2) {
1219          usage();
1220          return (ERROR_USAGE);
1221      }

1222      ret = do_refresh(argc);
1223      if (ret != SUCCESS)
1224          return (ret);
1225  }

1226      return (start_daemon());
1227  }

1228  /*
1229  * The top level function for the "cryptoadm stop" subcommand.
1230  * The top level function for the stop subcommand.
1231  */
1232  static int
1233  do_stop(int argc)
1234  {
1235      if (argc != 2) {
1236          usage();
1237          return (ERROR_USAGE);
1238      }
1239  }

```

```

1241     return (stop_daemon());
1242 }

1246 /*
1247 * Print a list all the the providers.
1248 * Called for "cryptoadm list" or "cryptoadm list -v" (no -m or -p).
1249 * List all the providers.
1250 */
1251 static int
1252 list_simple_for_all(boolean_t verbose)
1253 {
1254     uentrylist_t      *pliblist = NULL;
1255     uentrylist_t      *plibptr = NULL;
1256     entry_t           *pent = NULL;
1257     uentrylist_t      *pliblist;
1258     uentrylist_t      *plibptr;
1259     entrylist_t       *pdevlist_conf;
1260     entrylist_t       *psoftlist_conf;
1261     entrylist_t       *pdevlist_zone;
1262     entrylist_t       *psoftlist_zone;
1263     entrylist_t       *ptr;
1264     crypto_get_dev_list_t *pdevlist_kernel = NULL;
1265     int                rc = SUCCESS;
1266     boolean_t         is_active;
1267     int                ru = SUCCESS;
1268     int                rs = SUCCESS;
1269     int                rd = SUCCESS;
1270     int                i;

1271     /* get user-level providers */
1272     (void) printf(gettext("\nUser-level providers:\n"));
1273     if (get_pkcs11conf_info(&pliblist) != SUCCESS) {
1274         cryptoerror(LOG_STDERR, gettext(
1275             "failed to retrieve the list of user-level providers."));
1276         rc = FAILURE;
1277         ru = FAILURE;
1278     }

1279     for (plibptr = pliblist; plibptr != NULL; plibptr = plibptr->next) {
1280         plibptr = pliblist;
1281         while (plibptr != NULL) {
1282             if (strcmp(plibptr->puent->name, METASLOT_KEYWORD) != 0) {
1283                 (void) printf(gettext("Provider: %s\n"),
1284                     plibptr->puent->name);
1285                 if (verbose) {
1286                     (void) list_mechlist_for_lib(
1287                         plibptr->puent->name, mecharglist, NULL,
1288                         B_FALSE, verbose, B_FALSE);
1289                     (void) printf("\n");
1290                 }
1291             }
1292             plibptr = plibptr->next;
1293         }
1294         free_uentrylist(pliblist);
1295     }

1296     /* get kernel software providers */
1297     (void) printf(gettext("\nKernel software providers:\n"));

1298     if (getzoneid() == GLOBAL_ZONEID) {
1299         /* get kernel software providers from kernel ioctl */
1300         crypto_get_soft_list_t
1301             *psoftlist_kernel = NULL;
1302         uint_t
1303             sl_soft_count;
1304         char
1305             *psoftname;
1306         entrylist_t
1307             *pdevlist_conf = NULL;

```

```

1308     entrylist_t
1309         *psoftlist_conf = NULL;
1310     /* use kcf.conf for kernel software providers in global zone */
1311     pdevlist_conf = NULL;
1312     psoftlist_conf = NULL;

1313     if (get_soft_list(&psoftlist_kernel) == FAILURE) {
1314         cryptoerror(LOG_ERR, gettext("Failed to retrieve the "
1315             "software provider list from kernel."));
1316         rc = FAILURE;
1317     } else {
1318         sl_soft_count = psoftlist_kernel->sl_soft_count;
1319         if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf) !=
1320             SUCCESS) {
1321             cryptoerror(LOG_STDERR,
1322                 gettext("failed to retrieve the "
1323                     "list of kernel software providers.\n"));
1324             rs = FAILURE;
1325         }

1326         if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf)
1327             ptr = psoftlist_conf;
1328             while (ptr != NULL) {
1329                 if (check_active_for_soft(ptr->pent->name, &is_active)
1330                     == FAILURE) {
1331                     cryptoerror(LOG_ERR,
1332                         "failed to retrieve the providers' "
1333                         "information from file kcf.conf - %s.",
1334                         PATH_KCF_CONF);
1335                     free(psoftlist_kernel);
1336                     rc = FAILURE;
1337                 } else {
1338                     rs = FAILURE;
1339                     cryptoerror(LOG_STDERR, gettext("failed to "
1340                         "get the state of a kernel software "
1341                         "providers.\n"));
1342                     break;
1343                 }

1344                 for (i = 0,
1345                     psoftname = psoftlist_kernel->sl_soft_names;
1346                     i < sl_soft_count;
1347                     ++i, psoftname += strlen(psoftname) + 1) {
1348                     pent = getent_kef(psoftname,
1349                         pdevlist_conf, psoftlist_conf);
1350                     (void) printf("\t%s\n", psoftname,
1351                         (pent == NULL) || (pent->load) ?
1352                         "" : gettext(" (inactive)"));
1353                     (void) printf("\t%s", ptr->pent->name);
1354                     if (is_active == B_FALSE) {
1355                         (void) printf(gettext(" (inactive)\n"));
1356                     } else {
1357                         (void) printf("\n");
1358                     }
1359                     free_entrylist(pdevlist_conf);
1360                     free_entrylist(psoftlist_conf);
1361                     ptr = ptr->next;
1362                 }
1363                 free(psoftlist_kernel);
1364             }

1365         free_entrylist(pdevlist_conf);
1366         free_entrylist(psoftlist_conf);
1367     } else {
1368         /* kcf.conf not there in non-global zone, use /dev/cryptoadm */
1369         entrylist_t
1370             *pdevlist_zone = NULL;
1371         entrylist_t
1372             *psoftlist_zone = NULL;

```

```

1330     entrylist_t    *ptr;
1312     pdevlist_zone = NULL;
1313     psoftlist_zone = NULL;

1332     if (get_admindev_info(&pdevlist_zone, &psoftlist_zone) !=
1333         SUCCESS) {
1334         cryptoerror(LOG_STDERR,
1335             gettext("failed to retrieve the "
1336                 "list of kernel software providers.\n"));
1337         rc = FAILURE;
1320         rs = FAILURE;
1338     }

1340     ptr = psoftlist_zone;
1341     while (ptr != NULL) {
1342         (void) printf("\t%s\n", ptr->pent->name);
1343         ptr = ptr->next;
1344     }

1346     free_entrylist(pdevlist_zone);
1347     free_entrylist(psoftlist_zone);
1348 }

1350 /* get kernel hardware providers */
1351 (void) printf(gettext("\nKernel hardware providers:\n"));
1352 if (get_dev_list(&pdevlist_kernel) == FAILURE) {
1353     cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1354         "the list of kernel hardware providers.\n"));
1355     rc = FAILURE;
1338     rd = FAILURE;
1356 } else {
1357     for (i = 0; i < pdevlist_kernel->dl_dev_count; i++) {
1358         (void) printf("\t%s/%d\n",
1359             pdevlist_kernel->dl_devs[i].le_dev_name,
1360             pdevlist_kernel->dl_devs[i].le_dev_instance);
1361     }
1362 }
1363 free(pdevlist_kernel);

1365 return (rc);
1348 if (ru == FAILURE || rs == FAILURE || rd == FAILURE) {
1349     return (FAILURE);
1350 } else {
1351     return (SUCCESS);
1352 }
1366 }

1370 /*
1371 * List all the providers. And for each provider, list the mechanism list.
1372 * Called for "cryptoadm list -m" or "cryptoadm list -mv" .
1373 */
1374 static int
1375 list_mechlist_for_all(boolean_t verbose)
1376 {
1377     crypto_get_dev_list_t    *pdevlist_kernel = NULL;
1378     uentrylist_t            *pliblist = NULL;
1379     uentrylist_t            *plibptr = NULL;
1380     entry_t                  *pent = NULL;
1381     mechlist_t               *pmechlist = NULL;
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;

1382     char            provname[MAXNAMELEN];
1383     char            devname[MAXNAMELEN];
1384     int             inst_num;
1385     int             count;
1386     int             i;
1387     int             rv;
1388     int             rc = SUCCESS;

1390     /* get user-level providers */
1391     (void) printf(gettext("\nUser-level providers:\n"));
1392     /*
1393     * TRANSLATION_NOTE
1394     * Strictly for appearance's sake, this line should be as long as
1395     * the length of the translated text above.
1396     */
1397     (void) printf(gettext("=====\n"));
1398     if (get_pkcs11conf_info(&pliblist) != SUCCESS) {
1399         cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1400             "the list of user-level providers.\n"));
1401         rc = FAILURE;
1402     }

1404     plibptr = pliblist;
1405     while (plibptr != NULL) {
1406         /* skip metaslot entry */
1407         if (strcmp(plibptr->puent->name, METASLOT_KEYWORD) != 0) {
1408             (void) printf(gettext("\nProvider: %s\n"),
1409                 plibptr->puent->name);
1410             rv = list_mechlist_for_lib(plibptr->puent->name,
1411                 mecharglist, NULL, B_FALSE, verbose, B_TRUE);
1412             if (rv == FAILURE) {
1413                 rc = FAILURE;
1414             }
1415         }
1416         plibptr = plibptr->next;
1417     }
1418     free_uentrylist(pliblist);

1420     /* get kernel software providers */
1421     (void) printf(gettext("\nKernel software providers:\n"));

1423     /*
1424     * TRANSLATION_NOTE
1425     * Strictly for appearance's sake, this line should be as long as
1426     * the length of the translated text above.
1427     */
1428     (void) printf(gettext("=====\n"));
1429     if (getzoneid() == GLOBAL_ZONEID) {
1430         /* get kernel software providers from kernel ioctl */
1431         crypto_get_soft_list_t    *psoftlist_kernel = NULL;
1432         uint_t                     sl_soft_count;
1433         char                       *psoftname;
1434         int                         i;
1435         entrylist_t                *pdevlist_conf = NULL;
1436         entrylist_t                *psoftlist_conf = NULL;
1420     /* use kcf.conf for kernel software providers in global zone */
1421     pdevlist_conf = NULL;
1422     psoftlist_conf = NULL;

1438     if (get_soft_list(&psoftlist_kernel) == FAILURE) {
1439         cryptoerror(LOG_ERR, gettext("Failed to retrieve the "

```

```

1440     "software provider list from kernel.");
1441     return (FAILURE);
1442     if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf) !=
1443         SUCCESS) {
1444         cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1445             "the list of kernel software providers.\n"));
1446         rc = FAILURE;
1447     }
1448     sl_soft_count = psoftlist_kernel->sl_soft_count;
1449
1450     if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf)
1451         == FAILURE) {
1452         cryptoerror(LOG_ERR,
1453             "failed to retrieve the providers' "
1454             "information from file kcf.conf - %s.",
1455             _PATH_KCF_CONF);
1456         free(psoftlist_kernel);
1457         return (FAILURE);
1458     }
1459
1460     for (i = 0, psoftname = psoftlist_kernel->sl_soft_names;
1461         i < sl_soft_count;
1462         ++i, psoftname += strlen(psoftname) + 1) {
1463         pent = getent_kef(psoftname, pdevlist_conf,
1464             psoftlist_conf);
1465         if ((pent == NULL) || (pent->load)) {
1466             rv = list_mechlist_for_soft(psoftname,
1467                 NULL, NULL);
1468             ptr = psoftlist_conf;
1469             while (ptr != NULL) {
1470                 if (check_active_for_soft(ptr->pent->name, &is_active)
1471                     == SUCCESS) {
1472                     if (is_active) {
1473                         rv = list_mechlist_for_soft(
1474                             ptr->pent->name);
1475                     } else {
1476                         (void) printf(gettext("%s: (inactive)\n"),
1477                             psoftname);
1478                         (void) printf(gettext(
1479                             "%s: (inactive)\n"),
1480                             ptr->pent->name);
1481                     }
1482                 } else {
1483                     /* should not happen */
1484                     (void) printf(gettext(
1485                         "%s: failed to get the mechanism list.\n"),
1486                         ptr->pent->name);
1487                     rc = FAILURE;
1488                 }
1489                 ptr = ptr->next;
1490             }
1491         } else {
1492             free(psoftlist_kernel);
1493             free_entrylist(pdevlist_conf);
1494             free_entrylist(psoftlist_conf);
1495         }
1496     } else {
1497         /* kcf.conf not there in non-global zone, use /dev/cryptoadm */
1498         entrylist_t *pdevlist_zone = NULL;
1499         entrylist_t *psoftlist_zone = NULL;
1500         entrylist_t *ptr;
1501         pdevlist_zone = NULL;
1502         psoftlist_zone = NULL;

```

```

1503         if (get_admindev_info(&pdevlist_zone, &psoftlist_zone) !=
1504             SUCCESS) {
1505             cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1506                 "the list of kernel software providers.\n"));
1507             rc = FAILURE;
1508         }
1509
1510         for (ptr = psoftlist_zone; ptr != NULL; ptr = ptr->next) {
1511             rv = list_mechlist_for_soft(ptr->pent->name,
1512                 pdevlist_zone, psoftlist_zone);
1513             ptr = psoftlist_zone;
1514             while (ptr != NULL) {
1515                 rv = list_mechlist_for_soft(ptr->pent->name);
1516                 if (rv == FAILURE) {
1517                     (void) printf(gettext(
1518                         "%s: failed to get the mechanism list.\n"),
1519                         ptr->pent->name);
1520                     rc = FAILURE;
1521                 }
1522                 ptr = ptr->next;
1523             }
1524             free_entrylist(pdevlist_zone);
1525             free_entrylist(psoftlist_zone);
1526         }
1527
1528         /* Get kernel hardware providers and their mechanism lists */
1529         (void) printf(gettext("\nKernel hardware providers:\n"));
1530         /*
1531          * TRANSLATION_NOTE
1532          * Strictly for appearance's sake, this line should be as long as
1533          * the length of the translated text above.
1534          */
1535         (void) printf(gettext("=====\n"));
1536         if (get_dev_list(&pdevlist_kernel) != SUCCESS) {
1537             cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1538                 "the list of hardware providers.\n"));
1539             return (FAILURE);
1540         }
1541
1542         for (i = 0; i < pdevlist_kernel->dl_dev_count; i++) {
1543             (void) strcpy(devname,
1544                 pdevlist_kernel->dl_devs[i].le_dev_name, MAXNAMELEN);
1545             inst_num = pdevlist_kernel->dl_devs[i].le_dev_instance;
1546             count = pdevlist_kernel->dl_devs[i].le_mechanism_count;
1547             (void) snprintf(provname, sizeof (provname), "%s/%d", devname,
1548                 inst_num);
1549             if (get_dev_info(devname, inst_num, count, &pmechlist) ==
1550                 SUCCESS) {
1551                 (void) filter_mechlist(&pmechlist, RANDOM);
1552                 print_mechlist(provname, pmechlist);
1553                 free_mechlist(pmechlist);
1554             } else {
1555                 (void) printf(gettext("%s: failed to get the mechanism"
1556                     " list.\n"), provname);
1557                 rc = FAILURE;
1558             }
1559         }
1560         free(pdevlist_kernel);
1561         return (rc);
1562     }
1563
1564     /*
1565      * List all the providers. And for each provider, list the policy information.

```

```

1543 * Called for "cryptoadm list -p".
1544 */
1545 static int
1546 list_policy_for_all(void)
1547 {
1548     crypto_get_dev_list_t *pdevlist_kernel = NULL;
1549     uentrylist_t          *pliblist = NULL;
1550     entrylist_t           *pdevlist_conf = NULL;
1551     entrylist_t           *psoftlist_conf = NULL;
1552     entrylist_t           *ptr = NULL;
1553     entrylist_t           *phead = NULL;
1554     boolean_t             found = B_FALSE;
1555     crypto_get_dev_list_t *pdevlist_kernel;
1556     uentrylist_t          *pliblist;
1557     entrylist_t           *plibptr;
1558     entrylist_t           *pdevlist_conf;
1559     entrylist_t           *psoftlist_conf;
1560     entrylist_t           *ptr;
1561     entrylist_t           *phead;
1562     boolean_t             found;
1563     char                  provname[MAXNAMELEN];
1564     int                   i;
1565     int                   rc = SUCCESS;
1566
1567     /* Get user-level providers */
1568     (void) printf(gettext("\nUser-level providers:\n"));
1569     /*
1570      * TRANSLATION_NOTE
1571      * Strictly for appearance's sake, this line should be as long as
1572      * the length of the translated text above.
1573      */
1574     (void) printf(gettext("=====\n"));
1575     if (get_pkcs11conf_info(&pliblist) == FAILURE) {
1576         cryptoerror(LOG_STDERR, gettext("failed to retrieve "
1577             "the list of user-level providers.\n"));
1578         rc = FAILURE;
1579     } else {
1580         uentrylist_t *plibptr = pliblist;
1581
1582         plibptr = pliblist;
1583         while (plibptr != NULL) {
1584             /* skip metaslot entry */
1585             if (strcmp(plibptr->puent->name,
1586                 METASLOT_KEYWORD) != 0) {
1587                 if (print_uf_policy(plibptr->puent)
1588                     == FAILURE) {
1589                     rc = FAILURE;
1590                 }
1591             }
1592             plibptr = plibptr->next;
1593         }
1594         free_uentrylist(pliblist);
1595     }
1596
1597     /* kernel software providers */
1598     (void) printf(gettext("\nKernel software providers:\n"));
1599     /*
1600      * TRANSLATION_NOTE
1601      * Strictly for appearance's sake, this line should be as long as
1602      * the length of the translated text above.
1603      */
1604     (void) printf(gettext("=====\n"));
1605
1606     /* Get all entries from the kernel */
1607     /* Get all entries from the kcf.conf file */
1608     pdevlist_conf = NULL;

```

```

1598     if (getzoneid() == GLOBAL_ZONEID) {
1599         /* get kernel software providers from kernel ioctl */
1600         crypto_get_soft_list_t *psoftlist_kernel = NULL;
1601         uint_t                  sl_soft_count;
1602         char                    *psoftname;
1603         int                     i;
1604         /* use kcf.conf for kernel software providers in global zone */
1605         psoftlist_conf = NULL;
1606
1607         if (get_soft_list(&psoftlist_kernel) == FAILURE) {
1608             cryptoerror(LOG_ERR, gettext("Failed to retrieve the "
1609                 "software provider list from kernel."));
1610             rc = FAILURE;
1611         } else {
1612             sl_soft_count = psoftlist_kernel->sl_soft_count;
1613             if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf) ==
1614                 FAILURE) {
1615                 cryptoerror(LOG_STDERR, gettext(
1616                     "failed to retrieve the list of kernel "
1617                     "providers.\n"));
1618                 return (FAILURE);
1619             }
1620
1621             for (i = 0, psoftname = psoftlist_kernel->sl_soft_names;
1622                 i < sl_soft_count;
1623                 ++i, psoftname += strlen(psoftname) + 1) {
1624                 (void) list_policy_for_soft(psoftname,
1625                     pdevlist_conf, psoftlist_conf);
1626                 ptr = psoftlist_conf;
1627                 while (ptr != NULL) {
1628                     (void) list_policy_for_soft(ptr->pent->name);
1629                     ptr = ptr->next;
1630                 }
1631                 free(psoftlist_kernel);
1632             }
1633         }
1634     } else {
1635         free_entrylist(psoftlist_conf);
1636         /* kcf.conf not there in non-global zone, no policy info */
1637
1638         /*
1639          * TRANSLATION_NOTE
1640          * "global" is keyword and not to be translated.
1641          */
1642         cryptoerror(LOG_STDERR, gettext(
1643             "policy information for kernel software providers is "
1644             "available in the %s zone only"), "global");
1645     }
1646
1647     /* Kernel hardware providers */
1648     (void) printf(gettext("\nKernel hardware providers:\n"));
1649     /*
1650      * TRANSLATION_NOTE
1651      * Strictly for appearance's sake, this line should be as long as
1652      * the length of the translated text above.
1653      */
1654     (void) printf(gettext("=====\n"));
1655
1656     if (getzoneid() != GLOBAL_ZONEID) {
1657         /*
1658          * TRANSLATION_NOTE
1659          * "global" is keyword and not to be translated.
1660          */
1661         cryptoerror(LOG_STDERR, gettext(
1662             "policy information for kernel hardware providers is "
1663             "available in the %s zone only"), "global");

```



```

1650         return (FAILURE);
1651     }

1653     /* Get the hardware provider list from kernel */
1654     if (get_dev_list(&pdevlist_kernel) != SUCCESS) {
1655         cryptoerror(LOG_STDERR, gettext(
1656             "failed to retrieve the list of hardware providers.\n"));
1657         free_entrylist(pdevlist_conf);
1658     }

1660     if (get_kcfconf_info(&pdevlist_conf, &psoftlist_conf) == FAILURE) {
1661         cryptoerror(LOG_ERR, "failed to retrieve the providers' "
1662             "information from file kcf.conf - %s.",
1663             _PATH_KCF_CONF);
1664         return (FAILURE);
1665     }

1668     /*
1669     * For each hardware provider from kernel, check if it has an entry
1670     * in the config file.  If it has an entry, print out the policy from
1671     * config file and remove the entry from the hardware provider list
1672     * of the config file.  If it does not have an entry in the config
1673     * file, no mechanisms of it have been disabled. But, we still call
1674     * list_policy_for_hard() to account for the "random" feature.
1675     */
1676     for (i = 0; i < pdevlist_kernel->dl_dev_count; i++) {
1677         (void) snprintf(provname, sizeof (provname), "%s/%d",
1678             pdevlist_kernel->dl_devs[i].le_dev_name,
1679             pdevlist_kernel->dl_devs[i].le_dev_instance);

1681         found = B_FALSE;
1682         phead = ptr = pdevlist_conf;
1683         while (!found && ptr) {
1684             if (strcmp(ptr->pent->name, provname) == 0) {
1685                 found = B_TRUE;
1686             } else {
1687                 phead = ptr;
1688                 ptr = ptr->next;
1689             }
1690         }

1692         if (found) {
1693             (void) list_policy_for_hard(ptr->pent->name,
1694                 pdevlist_conf, psoftlist_conf, pdevlist_kernel);
1695             (void) list_policy_for_hard(ptr->pent->name);
1696             if (phead == ptr) {
1697                 pdevlist_conf = pdevlist_conf->next;
1698             } else {
1699                 phead->next = ptr->next;
1700             }
1701             free_entry(ptr->pent);
1702             free(ptr);
1703         } else {
1704             (void) list_policy_for_hard(provname, pdevlist_conf,
1705                 psoftlist_conf, pdevlist_kernel);
1706             (void) list_policy_for_hard(provname);
1707         }
1708     }

1708     /*
1709     * If there are still entries left in the pdevlist_conf list from
1710     * the config file, these providers must have been detached.
1711     * Should print out their policy information also.
1712     */

```

```

1713     for (ptr = pdevlist_conf; ptr != NULL; ptr = ptr->next) {
1714         print_kef_policy(ptr->pent->name, ptr->pent, B_FALSE, B_TRUE);
1715     }

1717     free_entrylist(pdevlist_conf);
1718     free_entrylist(psoftlist_conf);
1719     free(pdevlist_kernel);

1721     return (rc);
1722 }
_____unchanged_portion_omitted_

```

```

*****
6141 Tue Oct 28 16:45:36 2008
new/usr/src/cmd/cmd-crypto/cryptoadm/cryptoadm.h
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #ifndef _CRYPTOADM_H
27 #define _CRYPTOADM_H

29 #include <sys/types.h>
29 #pragma ident "%Z%M% %I% %E% SMI"

30 #include <sys/crypto/ioctladm.h>
31 #include <cryptoutil.h>
32 #include <security/cryptoki.h>

34 #ifdef __cplusplus
35 extern "C" {
36 #endif

38 #define _PATH_KCF_CONF "/etc/crypto/kcf.conf"
39 #define _PATH_KCFD "/usr/lib/crypto/kcfd"
40 #define TMPFILE_TEMPLATE "/etc/crypto/admXXXXXX"

42 #define ERROR_USAGE 2

44 /*
45 * Common keywords and delimiters for pkcs11.conf and kcf.conf files are
46 * defined in usr/lib/libcryptoutil/common/cryptoutil.h. The following is
47 * the extra keywords and delimiters used in kcf.conf file.
48 */
49 #define SEP_SLASH '/'
50 #define EF_SUPPORTED "supportedlist="
51 #define EF_UNLOAD "unload"
52 #define HW_DRIVER_STRING "driver_names"
52 #define RANDOM "random"
53 #define UEF_FRAME_LIB "/usr/lib/libpkcs11.so"

55 #define ADD_MODE 1
56 #define DELETE_MODE 2
57 #define MODIFY_MODE 3

```

```

59 typedef char prov_name_t[MAXNAMELEN];
60 typedef char mech_name_t[CRYPTO_MAX_MECH_NAME];

62 typedef struct mechlist {
63     mech_name_t name;
64     struct mechlist *next;
65 } mechlist_t;

68 typedef struct entry {
69     prov_name_t name;
70     mechlist_t *suplist; /* supported list */
71     uint_t sup_count;
72     mechlist_t *dislist; /* disabled list */
73     uint_t dis_count;
74     boolean_t load; /* B_FALSE after cryptoadm unload */
75 } entry_t;
unchanged_portion_omitted

88 extern int errno;

90 /* adm_util */
91 extern boolean_t is_in_list(char *, mechlist_t *);
92 extern mechlist_t *create_mech(char *);
93 extern void free_mechlist(mechlist_t *);

95 /* adm_kef_util */
96 extern boolean_t is_device(char *);
97 extern char *ent2str(entry_t *);
98 extern entry_t *getent_kef(char *provname,
99     entrylist_t *pdevlist, entrylist_t *psoftlist);
100 extern int check_kernel_for_soft(char *provname,
101     crypto_get_soft_list_t *psoftlist, boolean_t *in_kernel);
102 extern int check_kernel_for_hard(char *provname,
103     crypto_get_dev_list_t *pdevlist, boolean_t *in_kernel);
98 extern entry_t *getent_kef(char *);
99 extern int check_active_for_soft(char *, boolean_t *);
100 extern int check_active_for_hard(char *, boolean_t *);
104 extern int disable_mechs(entry_t **, mechlist_t *, boolean_t, mechlist_t *);
105 extern int enable_mechs(entry_t **, boolean_t, mechlist_t *);
106 extern int get_kcfconf_info(entrylist_t **, entrylist_t **);
107 extern int get_admindev_info(entrylist_t **, entrylist_t **);
108 extern int get_mech_count(mechlist_t *);
109 extern entry_t *create_entry(char *provname);
110 extern int insert_kcfconf(entry_t *);
111 extern int split_hw_provname(char *, char *, int *);
112 extern int update_kcfconf(entry_t *, int);
113 extern void free_entry(entry_t *);
114 extern void free_entrylist(entrylist_t *);
115 extern void print_mechlist(char *, mechlist_t *);
116 extern void print_kef_policy(char *provname, entry_t *pent,
117     boolean_t has_random, boolean_t has_mechs);
112 extern void print_kef_policy(entry_t *, boolean_t, boolean_t);
118 extern boolean_t filter_mechlist(mechlist_t **, const char *);
119 extern uentry_t *getent_uef(char *);

122 /* adm_uef */
123 extern int list_mechlist_for_lib(char *, mechlist_t *, flag_val_t *,
124     boolean_t, boolean_t, boolean_t);
125 extern int list_policy_for_lib(char *);
126 extern int disable_uef_lib(char *, boolean_t, boolean_t, mechlist_t *);
127 extern int enable_uef_lib(char *, boolean_t, boolean_t, mechlist_t *);
128 extern int install_uef_lib(char *);
129 extern int uninstall_uef_lib(char *);

```

```
130 extern int print_uf_policy(uentry_t *);
131 extern void display_token_flags(CK_FLAGS flags);
132 extern int convert_mechlist(CK_MECHANISM_TYPE **, CK_ULONG *, mechlist_t *);
133 extern void display_verbose_mech_header();
134 extern void display_mech_info(CK_MECHANISM_INFO *);
135 extern int display_policy(uentry_t *);
136 extern int update_pkcs11conf(uentry_t *);
137 extern int update_policylist(uentry_t *, mechlist_t *, int);

139 /* adm_kef */
140 extern int list_mechlist_for_soft(char *provname,
141     entrylist_t *phardlist, entrylist_t *psoftlist);
142 extern int list_mechlist_for_hard(char *);
143 extern int list_policy_for_soft(char *provname,
144     entrylist_t *phardlist, entrylist_t *psoftlist);
145 extern int list_policy_for_hard(char *provname,
146     entrylist_t *phardlist, entrylist_t *psoftlist,
147     crypto_get_dev_list_t *pdevlist);
148 extern int list_policy_for_soft(char *);
149 extern int list_policy_for_hard(char *);
150 extern int disable_kef_software(char *, boolean_t, boolean_t, mechlist_t *);
151 extern int disable_kef_hardware(char *, boolean_t, boolean_t, mechlist_t *);
152 extern int enable_kef(char *, boolean_t, boolean_t, mechlist_t *);
153 extern int install_kef(char *, mechlist_t *);
154 extern int uninstall_kef(char *);
155 extern int unload_kef_soft(char *provname);
156 extern int unload_kef_soft(char *, boolean_t);
157 extern int refresh(void);
158 extern int start_daemon(void);
159 extern int stop_daemon(void);

161 /* adm_ioctl */
162 extern crypto_load_soft_config_t *setup_soft_conf(entry_t *);
163 extern crypto_load_soft_disabled_t *setup_soft_dis(entry_t *);
164 extern crypto_load_dev_disabled_t *setup_dev_dis(entry_t *);
165 extern crypto_unload_soft_module_t *setup_unload_soft(entry_t *);
166 extern int get_dev_info(char *, int, int, mechlist_t **);
167 extern int get_dev_list(crypto_get_dev_list_t **);
168 extern int get_soft_info(char *provname, mechlist_t **ppmechlist,
169     entrylist_t *phardlist, entrylist_t *psoftlist);
170 extern int get_soft_info(char *, mechlist_t **);
171 extern int get_soft_list(crypto_get_soft_list_t **);

173 /* adm metaslot */
174 extern int list_metaslot_info(boolean_t, boolean_t, mechlist_t *);
175 extern int list_metaslot_policy();
176 extern int disable_metaslot(mechlist_t *, boolean_t, boolean_t);
177 extern int enable_metaslot(char *, char *, boolean_t, mechlist_t *, boolean_t,
178     boolean_t);

179 #ifdef __cplusplus
180 }
181 unchanged portion omitted

```

new/usr/src/cmd/cmd-crypto/etc/kcf.conf

1

1332 Tue Oct 28 16:45:40 2008

new/usr/src/cmd/cmd-crypto/etc/kcf.conf

6414175 kcf.conf's supportedlist not providing much usefulness

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
22 #
23 # Copyright 2007 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
26 # ident "%Z%M% %I% %E% SMI"
27 #
28 # /etc/crypto/kcf.conf
29 #
30 # Do NOT edit this file by hand. An administrator should use cryptoadm(1m)
31 # to administer the cryptographic framework.
32 # to administer the cryptographic framework. A developer for a kernel software
33 # provider package or a cryptographic provider device driver(s) package should
34 # provide an input file and use the {i,r}.kcfconf class action scripts to
35 # update this file during the installation and removal of the package.
36 #
37 # This document does not constitute an API. The /etc/crypto/kcf.conf file may
38 # not exist or may have a different content or interpretation in a future
39 # release. The existence of this notice does not imply that any other
40 # documentation that lacks this notice constitutes an API.
41 #
42 #
43 # Start SUNWcsr
44 des:supportedlist=CKM_DES_CBC,CKM_DES_ECB,CKM_DES3_CBC,CKM_DES3_ECB
45 aes:supportedlist=CKM_AES_ECB,CKM_AES_CBC,CKM_AES_CTR
46 arcfour:supportedlist=CKM_RC4
47 blowfish:supportedlist=CKM_BLOWFISH_ECB,CKM_BLOWFISH_CBC
48 ecc:supportedlist=CKM_EC_KEY_PAIR_GEN,CKM_ECDH1_DERIVE,CKM_ECDSA,CKM_ECDSA_SHA1
49 sha1:supportedlist=CKM_SHA_1,CKM_SHA_1_HMAC_GENERAL,CKM_SHA_1_HMAC
50 sha2:supportedlist=CKM_SHA256,CKM_SHA256_HMAC,CKM_SHA256_HMAC_GENERAL,CKM_SHA384
51 md4:supportedlist=CKM_MD4
52 md5:supportedlist=CKM_MD5,CKM_MD5_HMAC_GENERAL,CKM_MD5_HMAC
53 rsa:supportedlist=CKM_RSA_PKCS,CKM_RSA_X_509,CKM_MD5_RSA_PKCS,CKM_SHA1_RSA_PKCS,
54 swrand:supportedlist=random
55 # End SUNWcsr
```

new/usr/src/pkgdefs/SUNWcryptoint/prototype_com

1

2168 Tue Oct 28 16:45:46 2008

new/usr/src/pkgdefs/SUNWcryptoint/prototype_com

6414175 kcf.conf's supportedlist not providing much usefulness

```
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # This required package information file contains a list of package contents.
26 # The 'pkgmk' command uses this file to identify the contents of a package
27 # and their location on the development machine when building the package.
28 # Can be created via a text editor or through use of the 'pkgproto' command.
29 #
30 #!search <pathname pathname ...>      # where to find pkg objects
31 #!include <filename>                   # include another 'prototype' file
32 #!default <mode> <owner> <group>      # default used if not specified on entry
33 #!<param>=<value>                      # puts parameter in pkg environment
34 #
35 # packaging files
36 i pkginfo
37 i copyright
38 i depend
39 i postinstall
40 i preremove
41 #
42 # source locations relative to the prototype file
43 #
44 # SUNWcryptoint
45 #
46 # CRYPT DELETE START
47 d none etc 755 root sys
48 d none etc/certs 755 root sys
49 f none etc/certs/SUNWosnetSolaris 644 root sys
50 f none etc/certs/SUNWosnetSE 644 root sys
51 d none etc/crypto 755 root sys
52 d none etc/crypto/certs 755 root sys
53 f none etc/crypto/certs/SUNWosnet 644 root sys
54 f none etc/crypto/certs/SUNWosnetLimited 644 root sys
55 f none etc/crypto/certs/SUNWosnetCF 644 root sys
56 f none etc/crypto/certs/SUNWosnetCFLimited 644 root sys
57 # CRYPT DELETE END
58 d none kernel 755 root sys
59 d none kernel/crypto 755 root sys
60 d none kernel/drv 755 root sys
61 f none kernel/drv/dprov.conf 644 root sys
```

new/usr/src/pkgdefs/SUNWdcar/postinstall

1

```
*****
1763 Tue Oct 28 16:45:53 2008
new/usr/src/pkgdefs/SUNWdcar/postinstall
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 #!/bin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 #
24 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
25 # Use is subject to license terms.
26 #
27 #pragma ident "%Z%M% %I% %E% SMI"
28 #
29 #
30 #
31 #
32 PATH="/usr/bin:/usr/sbin:${PATH}"
33 export PATH
34 #
35 #
36 #
37 # Add hardware provider section for the dca driver
38 # to /etc/crypto/kcf.conf
39 #
40 pkg_start="# Start $PKGINST"
41 pkg_end="# End $PKGINST"
42 kcfconf=${BASEDIR}/etc/crypto/kcf.conf
43 tmpfile=/tmp/${kcfconf}
44 error=no
45 #
46 #
47 # If /etc/crypto/kcf.conf doesn't exist, bail immediately
48 #
49 if [ ! -f "$kcfconf" ]
50 then
51     echo "$0: ERROR - $kcfconf doesn't exist"
52     exit 2
53 fi
54 #
55 #
56 # If the package has been already installed, remove old entries
57 #
58 start=0
59 end=0
60 egrep -s "$pkg_start" $kcfconf && start=1
```

new/usr/src/pkgdefs/SUNWdcar/postinstall

2

```
61 egrep -s "$pkg_end" $kcfconf && end=1
62 #
63 if [ $start -ne $end ] ; then
64     echo "$0: missing Start or End delimiters for $PKGINST in $kcfconf."
65     echo "$0: $kcfconf may be corrupted and was not updated."
66     error=yes
67     exit 2
68 fi
69 #
70 # to preserve the gid
71 cp -p $kcfconf $tmpfile || error=yes
72 if [ $start -eq 1 ]
73 then
74     sed -e "/$pkg_start/,/$pkg_end/d" $kcfconf > $tmpfile || error=yes
75 fi
76 #
77 #
78 # Append the delimiters for this package
79 #
80 echo "$pkg_start driver_names=dca" >> $tmpfile || error=yes
81 echo "$pkg_end" >> $tmpfile || error=yes
82 #
83 #
84 # Install the updated config file and clean up the tmp file
85 #
86 if [ "$error" = no ]
87 then
88     mv $tmpfile $kcfconf || error=yes
89 fi
90 rm -f $tmpfile
91 #
92 #
93 # All done, if any of the steps above fail, report the error
94 #
95 if [ "$error" = yes ]
96 then
97     echo "$0: ERROR - failed to update $kcfconf."
98     exit 2
99 fi
100 #
101 #
102 #
103 #
104 #
105 #
106 #
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 NAMEMAJOR="${BASEDIR}/etc/name_to_major"
135 #
136 #
137 # Is the hardware there?
138 #
139 checkHardware()
140 {
141     for i in "pci14e4,5820" "pci14e4,5821" "pci14e4,5822" "pci14e4,5825" \
142             "pci108e,5454" "pci108e,5455" "pci108e,5456" "pci108e,5457"
143     do
144         prtconf -pv | egrep -s "$i"
145         if [ $? -eq 0 ]
146         then
147             return 1
148         fi
149     done
150     return 0
151 }
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
979 #
980 #
981 #
982 #
983 #
984 #
985 #
986 #
987 #
988 #
989 #
990 #
991 #
992 #
993 #
994 #
995 #
996 #
997 #
998 #
999 #
1000 #
```

```

*****
1426 Tue Oct 28 16:45:58 2008
new/usr/src/pkgdefs/SUNWdcar/preremove
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 #! /bin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 #
24 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
25 # Use is subject to license terms.
26 #
27 #pragma ident "%Z%M% %I% %E% SMI"
28 #
29 #
29 # Preremove script for SUNWdcar.
30 #
31 # This script removes the driver with rem_drv(1M) if necessary.
32 # This script removes the hardware provider section for the dca
33 # driver from /etc/crypto/kcf.conf
34 #
33 DRV=dca
34 NAMEMAJOR="${BASEDIR}/etc/name_to_major"
35 #
36 #
37 # Determine if we are on an alternate BASEDIR
38 #
39 if [ "${BASEDIR:=/}" = "/" ]
40 then
41     REM_DRV="/usr/sbin/rem_drv"
42 else
43     REM_DRV="/usr/sbin/rem_drv -b ${BASEDIR}"
44 fi
45 #
46 #
47 # Remove the driver, but only if this has not already been done.
48 #
49 grep -w "${DRV}" ${NAMEMAJOR} > /dev/null 2>&1
50 if [ $? -eq 0 ]; then
51     ${REM_DRV} ${DRV} || exit 1
52 fi
53 #
54 #
55 #
56 #
57 pkg_start="# Start $PKGINST"
58 pkg_end="# End $PKGINST"

```

```

59 kcfconf=${BASEDIR}/etc/crypto/kcf.conf
60 tmpfile=/tmp/${kcfconf}
61 error=no
62 #
63 #
64 # If /etc/crypto/kcf.conf doesn't exist, bail immediately
65 #
66 if [ ! -f "$kcfconf" ]
67 then
68     echo "$0: ERROR - $kcfconf doesn't exist"
69     exit 2
70 fi
71 #
72 #
73 # Strip all entries belonging to this package
74 #
75 start=0
76 end=0
77 egrep -s "$pkg_start" $kcfconf && start=1
78 egrep -s "$pkg_end" $kcfconf && end=1
79 #
80 if [ $start -ne $end ] ; then
81     echo "$0: missing Start or End delimiters for $PKGINST in $kcfconf."
82     echo "$0: $kcfconf may be corrupted and was not updated."
83     error=yes
84     exit 2
85 fi
86 #
87 if [ $start -eq 1 ]
88 then
89     # To preserve the gid
90     cp -p $kcfconf $tmpfile
91     sed -e "/$pkg_start/,/$pkg_end/d" $kcfconf > $tmpfile || error=yes
92     if [ "$error" = no ]
93     then
94         mv $tmpfile $kcfconf || error=yes
95     fi
96     rm -f $tmpfile
97 else
98     echo "$0: WARNING - no entries to be removed from $kcfconf"
99     exit 0
100 fi
101 #
102 if [ "$error" = yes ]
103 then
104     echo "$0: ERROR - failed to update $kcfconf."
105     exit 2
106 fi
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
979 #
980 #
981 #
982 #
983 #
984 #
985 #
986 #
987 #
988 #
989 #
990 #
991 #
992 #
993 #
994 #
995 #
996 #
997 #
998 #
999 #
1000 #

```

new/usr/src/pkgdefs/SUNWn2cp.v/postinstall

1

1315 Tue Oct 28 16:46:02 2008

new/usr/src/pkgdefs/SUNWn2cp.v/postinstall

6414175 kcf.conf's supportedlist not providing much usefulness

```
1 #!/bin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 #
24 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
25 # Copyright 2007 Sun Microsystems, Inc. All rights reserved.
26 # Use is subject to license terms.
27 # ident "%Z%M% %I% %E% SMI"
28 #
29 #
30 #
31 #
32 PATH="/usr/bin:/usr/sbin:${PATH}"
33 export PATH
34 #
35 #
36 #
37 # Add hardware provider section for the n2cp driver
38 # to /etc/crypto/kcf.conf
39 #
40 pkg_start="# Start $PKGINST"
41 pkg_end="# End $PKGINST"
42 kcfconf=${BASEDIR}/etc/crypto/kcf.conf
43 tmpfile=/tmp/${kcfconf}
44 error=no
45 #
46 #
47 # If /etc/crypto/kcf.conf doesn't exist, bail immediately
48 #
49 if [ ! -f "$kcfconf" ]
50 then
51     echo "$0: ERROR - $kcfconf doesn't exist"
52     exit 2
53 fi
54 #
55 #
56 # If the package has been already installed, remove old entries
57 #
58 start=0
59 end=0
60 egrep -s "$pkg_start" $kcfconf && start=1
```

new/usr/src/pkgdefs/SUNWn2cp.v/postinstall

2

```
61 egrep -s "$pkg_end" $kcfconf && end=1
62 #
63 if [ $start -ne $end ] ; then
64     echo "$0: missing Start or End delimiters for $PKGINST in $kcfconf."
65     echo "$0: $kcfconf may be corrupted and was not updated."
66     error=yes
67     exit 2
68 fi
69 #
70 if [ $start -eq 1 ]
71 then
72     cp -p $kcfconf $tmpfile || error=yes
73     sed -e "/$pkg_start/,/$pkg_end/d" $kcfconf > $tmpfile || error=yes
74 else
75     cp -p $kcfconf $tmpfile || error=yes
76 fi
77 #
78 #
79 # Append the delimiters for this package
80 #
81 echo "$pkg_start driver_names=n2cp" >> $tmpfile || error=yes
82 echo "$pkg_end" >> $tmpfile || error=yes
83 #
84 #
85 # Install the updated config file and clean up the tmp file
86 #
87 if [ "$error" = no ]
88 then
89     mv $tmpfile $kcfconf || error=yes
90 fi
91 rm -f $tmpfile
92 #
93 #
94 # All done, if any of the steps above fail, report the error
95 #
96 if [ "$error" = yes ]
97 then
98     echo "$0: ERROR - failed to update $kcfconf."
99     exit 2
100 fi
101 #
102 #
103 #
104 NAMEMAJOR=${BASEDIR}/etc/name_to_major"
105 #
106 if [ "${BASEDIR:=/}" = "/" ]
107 then
108     ADD_DRV="/usr/sbin/add_drv"
109 else
110     ADD_DRV="/usr/sbin/add_drv -b ${BASEDIR}"
111 fi
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
979 #
980 #
981 #
982 #
983 #
984 #
985 #
986 #
987 #
988 #
989 #
990 #
991 #
992 #
993 #
994 #
995 #
996 #
997 #
998 #
999 #
1000 #
```


new/usr/src/pkgdefs/SUNWn2cp.v/preremove

1

```
*****
1425 Tue Oct 28 16:46:07 2008
new/usr/src/pkgdefs/SUNWn2cp.v/preremove
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 #! /bin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 #
24 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
25 # Copyright 2006 Sun Microsystems, Inc. All rights reserved.
26 # Use is subject to license terms.
27 # ident "%Z%M% %I% %E% SMI"
28 #
29 #
30 # Preremove script for SUNWn2cp.v.
31 # This script removes driver with rem_drv(1M) if necessary.
32 # This script removes the hardware provider section for the n2cp
33 # driver from /etc/crypto/kcf.conf
34 #
35 #
36 #
37 # Determine if we are on an alternate BASEDIR
38 #
39 if [ "${BASEDIR:=/}" = "/" ]
40 then
41     REM_DRV="/usr/sbin/rem_drv"
42 else
43     REM_DRV="/usr/sbin/rem_drv -b ${BASEDIR}"
44 fi
45 #
46 #
47 # Remove the driver, but only if this has not already been done.
48 #
49 grep -w "${DRV}" ${NAMEMAJOR} > /dev/null 2>&1
50 if [ $? -eq 0 ]; then
51     ${REM_DRV} ${DRV} || exit 1
52 fi
53 #
54 #
55 #
56 #
57 pkg_start="# Start $PKGINST"
58 pkg_end="# End $PKGINST"
```

new/usr/src/pkgdefs/SUNWn2cp.v/preremove

2

```
59 kcfconf=${BASEDIR}/etc/crypto/kcf.conf
60 tmpfile=/tmp/${kcfconf}
61 error=no
62 #
63 #
64 # If /etc/crypto/kcf.conf doesn't exist, bail immediately
65 #
66 if [ ! -f "$kcfconf" ]
67 then
68     echo "$0: ERROR - $kcfconf doesn't exist"
69     exit 2
70 fi
71 #
72 #
73 # Strip all entries belonging to this package
74 #
75 start=0
76 end=0
77 egrep -s "$pkg_start" $kcfconf && start=1
78 egrep -s "$pkg_end" $kcfconf && end=1
79 #
80 if [ $start -ne $end ] ; then
81     echo "$0: missing Start or End delimiters for $PKGINST in $kcfconf."
82     echo "$0: $kcfconf may be corrupted and was not updated."
83     error=yes
84     exit 2
85 fi
86 #
87 if [ $start -eq 1 ]
88 then
89     cp -p $kcfconf $tmpfile || error=yes
90     sed -e "/$pkg_start/,/$pkg_end/d" $kcfconf > $tmpfile || error=yes
91     if [ "$error" = no ]
92     then
93         mv $tmpfile $kcfconf || error=yes
94     fi
95     rm -f $tmpfile
96 else
97     exit 0
98 fi
99 #
100 if [ "$error" = yes ]
101 then
102     echo "$0: ERROR - failed to update $kcfconf."
103     exit 2
104 fi
105 #
106 #
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
979 #
980 #
981 #
982 #
983 #
984 #
985 #
986 #
987 #
988 #
989 #
990 #
991 #
992 #
993 #
994 #
995 #
996 #
997 #
998 #
999 #
1000 #
```

```

*****
31872 Tue Oct 28 16:46:18 2008
new/usr/src/uts/common/crypto/core/kcf_cryptoadm.c
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

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

26 /*
27  * Core KCF (Kernel Cryptographic Framework). This file implements
28  * the cryptoadm entry points.
29  */

31 #include <sys/systm.h>
32 #include <sys/errno.h>
33 #include <sys/cmn_err.h>
34 #include <sys/rwlock.h>
35 #include <sys/kmem.h>
36 #include <sys/modctl.h>
37 #include <sys/sunddi.h>
38 #include <sys/door.h>
39 #include <sys/crypto/common.h>
40 #include <sys/crypto/api.h>
41 #include <sys/crypto/spi.h>
42 #include <sys/crypto/impl.h>
43 #include <sys/crypto/sched_impl.h>

45 /* protects the the soft_config_list. */
46 kmutex_t soft_config_mutex;

48 /*
49  * This linked list contains software configuration entries.
50  * The initial list is just software providers loaded by kcf_soft_config_init().
51  * Additional entries may appear for both hardware and software providers
52  * from kcf.conf. These come from "cryptoadm start", which reads file kcf.conf
53  * and updates this table using the CRYPTO_LOAD_SOFT_CONFIG ioctl.
54  * Further cryptoadm commands modify this file and update this table with ioctl.
55  * This list is protected by the soft_config_mutex.
56  * This linked list contains software configuration entries that
57  * are loaded into the kernel by the CRYPTO_LOAD_SOFT_CONFIG ioctl.
58  * It is protected by the soft_config_mutex.
59  */

```

```

57 kcf_soft_conf_entry_t *soft_config_list;

59 static int add_soft_config(char *, uint_t, crypto_mech_name_t *);
60 static int dup_mech_names(kcf_provider_desc_t *, crypto_mech_name_t **,
61     uint_t *, int);
62 static void free_soft_config_entry(kcf_soft_conf_entry_t *);

64 #define KCF_MAX_CONFIG_ENTRIES 512 /* maximum entries in soft_config_list */

66 #if DEBUG
67 extern int kcf_frmwrk_debug;
68 static void kcf_soft_config_dump(char *message);
69 #endif /* DEBUG */

71 /*
72  * Count and return the number of mechanisms in an array of crypto_mech_name_t
73  * (excluding final NUL-character string element).
74  */
75 static int
76 count_mechanisms(crypto_mech_name_t mechs[]) {
77     int count;
78     for (count = 0; mechs[count][0] != '\0'; ++count);
79     return (count);
80 }

82 /*
83  * Initialize a mutex and populate soft_config_list with default entries
84  * of kernel software providers.
85  * Called from kcf module _init().
86  */
87 void
88 kcf_soft_config_init(void)
89 {
90     typedef struct {
91         char *name;
92         crypto_mech_name_t *mechs;
93     } initial_soft_config_entry_t;

95     /*
96      * This provides initial default values to soft_config_list.
97      * It is equivalent to these lines in /etc/crypto/kcf.conf
98      * (without line breaks and indenting):
99      *
100     * # /etc/crypto/kcf.conf
101     * des:supportedlist=CKM_DES_CBC,CKM_DES_ECB,CKM_DES3_CBC,CKM_DES3_ECB
102     * aes:supportedlist=CKM_AES_ECB,CKM_AES_CBC,CKM_AES_CTR,CKM_AES_CCM
103     * arcfour:supportedlist=CKM_RC4
104     * blowfish:supportedlist=CKM_BLOWFISH_ECB,CKM_BLOWFISH_CBC
105     * ecc:supportedlist=CKM_EC_KEY_PAIR_GEN,CKM_ECDH1_DERIVE,CKM_ECDSA,\
106     * CKM_ECDSA_SHA1
107     * sha1:supportedlist=CKM_SHA_1,CKM_SHA_1_HMAC_GENERAL,CKM_SHA_1_HMAC
108     * sha2:supportedlist=CKM_SHA256,CKM_SHA256_HMAC,
109     * CKM_SHA256_HMAC_GENERAL,CKM_SHA384,CKM_SHA384_HMAC,\
110     * CKM_SHA384_HMAC_GENERAL,CKM_SHA512,CKM_SHA512_HMAC,\
111     * CKM_SHA512_HMAC_GENERAL
112     * md4:supportedlist=CKM_MD4
113     * md5:supportedlist=CKM_MD5,CKM_MD5_HMAC_GENERAL,CKM_MD5_HMAC
114     * rsa:supportedlist=CKM_RSA_PKCS,CKM_RSA_X_509,CKM_MD5_RSA_PKCS,\
115     * CKM_SHA1_RSA_PKCS,CKM_SHA256_RSA_PKCS,CKM_SHA384_RSA_PKCS,\
116     * CKM_SHA512_RSA_PKCS
117     * swrand:supportedlist=random
118     *
119     * WARNING: If you add a new kernel crypto provider or mechanism,
120     * you must update these constants.
121     *
122     * 1. To add a new mechanism to a provider add the string to the

```

```

123     * appropriate array below.
124     *
125     * 2. To add a new provider, create a new *_mechs array listing the
126     * provider's mechanism(s). For example:
127     *   sha3_mechs[SHA3_MECH_COUNT] = {"CKM_SHA_3"};
128     * Add the new *_mechs array to initial_soft_config_entry[.
129     */
130     static crypto_mech_name_t   des_mechs[] = {
131         "CKM_DES_CBC", "CKM_DES_ECB", "CKM_DES3_CBC", "CKM_DES3_ECB", ""};
132     static crypto_mech_name_t   aes_mechs[] = {
133         "CKM_AES_ECB", "CKM_AES_CBC", "CKM_AES_CTR", "CKM_AES_CCM", ""};
134     static crypto_mech_name_t   arcfour_mechs[] = {
135         "CKM_RC4", ""};
136     static crypto_mech_name_t   blowfish_mechs[] = {
137         "CKM_BLOWFISH_ECB", "CKM_BLOWFISH_CBC", ""};
138     static crypto_mech_name_t   ecc_mechs[] = {
139         "CKM_EC_KEY_PAIR_GEN", "CKM_ECDH1_DERIVE", "CKM_ECDSA",
140         "CKM_ECDSA_SHA1", ""};
141     static crypto_mech_name_t   sha1_mechs[] = {
142         "CKM_SHA_1", "CKM_SHA_1_HMAC_GENERAL", "CKM_SHA_1_HMAC", ""};
143     static crypto_mech_name_t   sha2_mechs[] = {
144         "CKM_SHA256", "CKM_SHA256_HMAC", "CKM_SHA256_HMAC_GENERAL",
145         "CKM_SHA384", "CKM_SHA384_HMAC", "CKM_SHA384_HMAC_GENERAL",
146         "CKM_SHA512", "CKM_SHA512_HMAC", "CKM_SHA512_HMAC_GENERAL", ""};
147     static crypto_mech_name_t   md4_mechs[] = {
148         "CKM_MD4", ""};
149     static crypto_mech_name_t   md5_mechs[] = {
150         "CKM_MD5", "CKM_MD5_HMAC_GENERAL", "CKM_MD5_HMAC", ""};
151     static crypto_mech_name_t   rsa_mechs[] = {
152         "CKM_RSA_PKCS", "CKM_RSA_X_509", "CKM_MD5_RSA_PKCS",
153         "CKM_SHA1_RSA_PKCS", "CKM_SHA256_RSA_PKCS", "CKM_SHA384_RSA_PKCS",
154         "CKM_SHA512_RSA_PKCS", ""};
155     static crypto_mech_name_t   swrand_mechs[] = {
156         "random", NULL};
157     static initial_soft_config_entry_t
158     initial_soft_config_entry[] = {
159         "des", des_mechs,
160         "aes", aes_mechs,
161         "arcfour", arcfour_mechs,
162         "blowfish", blowfish_mechs,
163         "ecc", ecc_mechs,
164         "sha1", sha1_mechs,
165         "sha2", sha2_mechs,
166         "md4", md4_mechs,
167         "md5", md5_mechs,
168         "rsa", rsa_mechs,
169         "swrand", swrand_mechs
170     };
171     const int   initial_soft_config_entries =
172         sizeof (initial_soft_config_entry)
173         / sizeof (initial_soft_config_entry_t);
174     int   i;
175
176     mutex_init(&soft_config_mutex, NULL, MUTEX_DRIVER, NULL);
177
178     /*
179     * Initialize soft_config_list with default providers.
180     * Populate the linked list backwards so the first entry appears first.
181     */
182     for (i = initial_soft_config_entries - 1; i >= 0; --i) {
183         initial_soft_config_entry_t *p = &initial_soft_config_entry[i];
184         crypto_mech_name_t           *mechsp;
185         char                          *namep;
186         uint_t                        namelen, alloc_size;
187         int                            mech_count, r;

```

```

189         /* allocate/initialize memory for name and mechanism list */
190         namelen = strlen(p->name) + 1;
191         namep = kmem_alloc(namelen, KM_SLEEP);
192         (void) strncpy(namep, p->name, namelen);
193         mech_count = count_mechanisms(p->mechs);
194         alloc_size = mech_count * CRYPTO_MAX_MECH_NAME;
195         mechsp = kmem_alloc(alloc_size, KM_SLEEP);
196         bcopy(p->mechs, mechsp, alloc_size);
197
198         r = add_soft_config(namep, mech_count, mechsp);
199         if (r != 0)
200             cmn_err(CE_WARN,
201                 "add_soft_config(%s) failed; returned %d\n",
202                 namep, r);
203     }
204     #if DEBUG
205         if (kcf_frmwrk_debug >= 1)
206             kcf_soft_config_dump("kcf_soft_config_init");
207     #endif /* DEBUG */
208 }
209
210
211 #if DEBUG
212 /*
213  * Dump soft_config_list, containing a list of kernel software providers
214  * and (optionally) hardware providers, with updates from kcf.conf.
215  * Dump mechanism lists too if kcf_frmwrk_debug is >= 2.
216  */
217 static void
218 kcf_soft_config_dump(char *message)
219 {
220     kcf_soft_conf_entry_t   *p;
221     uint_t                  i;
222
223     mutex_enter(&soft_config_mutex);
224     printf("Soft provider config list soft_config_list: %s\n",
225         message != NULL ? message : "");
226
227     for (p = soft_config_list; p != NULL; p = p->ce_next) {
228         printf("ce_name: %s, %d ce_mechs\n", p->ce_name, p->ce_count);
229         if (kcf_frmwrk_debug >= 2) {
230             printf("\tce_mechs: ");
231             for (i = 0; i < p->ce_count; i++) {
232                 printf("%s ", p->ce_mechs[i]);
233             }
234             printf("\n");
235         }
236     }
237     printf("(end of soft_config_list)\n");
238
239     mutex_exit(&soft_config_mutex);
240 }
241 #endif /* DEBUG */
242
243
244 /*
245  * Utility routine to identify the providers to filter out and
246  * present only one provider. This happens when a hardware provider
247  * registers multiple units of the same device instance.
248  * Called from crypto_get_dev_list().
249  */
250 static void
251 filter_providers(uint_t count, kcf_provider_desc_t **provider_array,
252     char *skip_providers, int *mech_counts, int *new_count)
253 {
254 }

```

```

255     int i, j;
256     kcf_provider_desc_t *prov1, *prov2;
257     int n = 0;

259     for (i = 0; i < count; i++) {
260         if (skip_providers[i] == 1)
261             continue;

263         prov1 = provider_array[i];
264         mech_counts[i] = prov1->pd_mech_list_count;
265         for (j = i + 1; j < count; j++) {
266             prov2 = provider_array[j];
267             if (strncmp(prov1->pd_name, prov2->pd_name,
268                 MAXNAMELEN) == 0 &&
269                 prov1->pd_instance == prov2->pd_instance) {
270                 skip_providers[j] = 1;
271                 mech_counts[i] += prov2->pd_mech_list_count;
272             }
273         }
274         n++;
275     }

277     *new_count = n;
278 }

281 /*
282  * Return a list of kernel hardware providers and a count of each
283  * provider's supported mechanisms.
284  * Called from the CRYPTO_GET_DEV_LIST ioctl.
285  */
286 int
287 crypto_get_dev_list(uint_t *count, crypto_dev_list_entry_t **array)
288 {
289     kcf_provider_desc_t **provider_array;
290     kcf_provider_desc_t *pd;
291     crypto_dev_list_entry_t *p;
292     size_t skip_providers_size, mech_counts_size;
293     char *skip_providers;
294     uint_t provider_count;
295     int rval, i, j, new_count, *mech_counts;

297     /*
298      * Take snapshot of provider table returning only hardware providers
299      * that are in a usable state. Logical providers not included.
300      */
301     rval = kcf_get_hw_prov_tab(&provider_count, &provider_array, KM_SLEEP,
302         NULL, 0, B_FALSE);
303     if (rval != CRYPTO_SUCCESS)
304         return (rval);

306     if (provider_count == 0) {
307         *array = NULL;
308         *count = 0;
309         return (CRYPTO_SUCCESS);
310     }

312     skip_providers_size = provider_count * sizeof (char);
313     mech_counts_size = provider_count * sizeof (int);

315     skip_providers = kmem_zalloc(skip_providers_size, KM_SLEEP);
316     mech_counts = kmem_zalloc(mech_counts_size, KM_SLEEP);
317     filter_providers(provider_count, provider_array, skip_providers,
318         mech_counts, &new_count);

```

```

320     p = kmem_alloc(new_count * sizeof (crypto_dev_list_entry_t), KM_SLEEP);
321     for (i = 0, j = 0; i < provider_count; i++) {
322         if (skip_providers[i] == 1) {
323             ASSERT(mech_counts[i] == 0);
324             continue;
325         }
326         pd = provider_array[i];
327         p[j].le_mechanism_count = mech_counts[i];
328         p[j].le_dev_instance = pd->pd_instance;
329         (void) strncpy(p[j].le_dev_name, pd->pd_name, MAXNAMELEN);
330         j++;
331     }

333     kcf_free_provider_tab(provider_count, provider_array);
334     kmem_free(skip_providers, skip_providers_size);
335     kmem_free(mech_counts, mech_counts_size);

337     *array = p;
338     *count = new_count;
339     return (CRYPTO_SUCCESS);
340 }

342 /*
343  * Return a buffer containing the null terminated names of software providers
344  * Called from the CRYPTO_GET_SOFT_LIST ioctl, this routine returns
345  * a buffer containing the null terminated names of software providers
346  * loaded by CRYPTO_LOAD_SOFT_CONFIG.
347  * Called from the CRYPTO_GET_SOFT_LIST ioctl.
348  */
349 int
350 crypto_get_soft_list(uint_t *count, char **array, size_t *len)
351 {
352     char *names = NULL, *namep, *end;
353     kcf_soft_conf_entry_t *p;
354     uint_t n = 0, cnt = 0, final_count = 0;
355     size_t name_len, final_size = 0;

357     /* first estimate */
358     mutex_enter(&soft_config_mutex);
359     for (p = soft_config_list; p != NULL; p = p->ce_next) {
360         n += strlen(p->ce_name) + 1;
361         cnt++;
362     }
363     mutex_exit(&soft_config_mutex);

365     if (cnt == 0)
366         goto out;

368     again:
369     namep = names = kmem_alloc(n, KM_SLEEP);
370     end = names + n;
371     final_size = 0;
372     final_count = 0;

374     mutex_enter(&soft_config_mutex);
375     for (p = soft_config_list; p != NULL; p = p->ce_next) {
376         name_len = strlen(p->ce_name) + 1;
377         /* check for enough space */
378         if ((namep + name_len) > end) {
379             mutex_exit(&soft_config_mutex);
380             kmem_free(names, n);
381             n = n << 1;
382             goto again;
383         }
384         (void) strcpy(namep, p->ce_name);
385         namep += name_len;

```

```

384         final_size += name_len;
385         final_count++;
386     }
387     mutex_exit(&soft_config_mutex);
389     ASSERT(final_size <= n);
391     /* check if buffer we allocated is too large */
392     if (final_size < n) {
393         char *final_buffer;
395         final_buffer = kmem_alloc(final_size, KM_SLEEP);
396         bcopy(names, final_buffer, final_size);
397         kmem_free(names, n);
398         names = final_buffer;
399     }
400 out:
401     *array = names;
402     *count = final_count;
403     *len = final_size;
404     return (CRYPTO_SUCCESS);
405 }
407 /*
408  * Check if a mechanism name is already in a mechanism name array
409  * Called by crypto_get_dev_info().
410  */
411 static boolean_t
412 duplicate(char *name, crypto_mech_name_t *array, int count)
413 {
414     int i;
416     for (i = 0; i < count; i++) {
417         if (strncmp(name, &array[i][0],
418             sizeof (crypto_mech_name_t)) == 0)
419             return (B_TRUE);
420     }
421     return (B_FALSE);
422 }
424 /*
425  * Return a list of kernel hardware providers for a given name and instance.
426  * For each entry, also return a list of their supported mechanisms.
427  * Called from the CRYPTO_GET_DEV_INFO ioctl.
428  */
429 int
430 crypto_get_dev_info(char *name, uint_t instance, uint_t *count,
431     crypto_mech_name_t **array)
432 {
433     int rv;
434     crypto_mech_name_t *mech_names, *resized_array;
435     int i, j, k = 0, max_count;
436     uint_t provider_count;
437     kcf_provider_desc_t **provider_array;
438     kcf_provider_desc_t *pd;
440     /*
441     * Get provider table entries matching name and instance
442     * for hardware providers that are in a usable state.
443     * Logical providers not included. NULL name matches
444     * all hardware providers.
445     */
446     rv = kcf_get_hw_prov_tab(&provider_count, &provider_array, KM_SLEEP,
447         name, instance, B_FALSE);
448     if (rv != CRYPTO_SUCCESS)

```

```

449         return (rv);
451     if (provider_count == 0)
452         return (CRYPTO_ARGUMENTS_BAD);
454     /* Count all mechanisms supported by all providers */
455     max_count = 0;
456     for (i = 0; i < provider_count; i++)
457         max_count += provider_array[i]->pd_mech_list_count;
459     if (max_count == 0) {
460         mech_names = NULL;
461         goto out;
462     }
464     /* Allocate space and copy mech names */
465     mech_names = kmem_alloc(max_count * sizeof (crypto_mech_name_t),
466         KM_SLEEP);
468     k = 0;
469     for (i = 0; i < provider_count; i++) {
470         pd = provider_array[i];
471         for (j = 0; j < pd->pd_mech_list_count; j++) {
472             /* check for duplicate */
473             if (duplicate(&pd->pd_mechanisms[j].cm_mech_name[0],
474                 mech_names, k))
475                 continue;
476             bcopy(&pd->pd_mechanisms[j].cm_mech_name[0],
477                 &mech_names[k][0], sizeof (crypto_mech_name_t));
478             k++;
479         }
480     }
482     /* resize */
483     if (k != max_count) {
484         resized_array =
485             kmem_alloc(k * sizeof (crypto_mech_name_t), KM_SLEEP);
486         bcopy(mech_names, resized_array,
487             k * sizeof (crypto_mech_name_t));
488         kmem_free(mech_names,
489             max_count * sizeof (crypto_mech_name_t));
490         mech_names = resized_array;
491     }
493 out:
494     kcf_free_provider_tab(provider_count, provider_array);
495     *count = k;
496     *array = mech_names;
498     return (CRYPTO_SUCCESS);
499 }
501 /*
502  * Given a kernel software provider name, return a list of mechanisms
503  * it supports.
504  * Called from the CRYPTO_GET_SOFT_INFO ioctl.
505  */
506 int
507 crypto_get_soft_info(caddr_t name, uint_t *count, crypto_mech_name_t **array)
508 {
509     ddi_modhandle_t modh = NULL;
510     kcf_provider_desc_t *provider;
511     int rv;
513     provider = kcf_prov_tab_lookup_by_name(name);

```

```

514     if (provider == NULL) {
515         if (in_soft_config_list(name)) {
516             char *tmp;
517             int name_len;

518             /* strlen("crypto/") + NULL terminator == 8 */
519             name_len = strlen(name);
520             tmp = kmem_alloc(name_len + 8, KM_SLEEP);
521             bcopy("crypto/", tmp, 7);
522             bcopy(name, &tmp[7], name_len);
523             tmp[name_len + 7] = '\0';

524             modh = ddi_modopen(tmp, KRTLD_MODE_FIRST, NULL);
525             kmem_free(tmp, name_len + 8);

526             if (modh == NULL) {
527                 return (CRYPTO_ARGUMENTS_BAD);
528             }

529             provider = kcf_prov_tab_lookup_by_name(name);
530             if (provider == NULL) {
531                 return (CRYPTO_ARGUMENTS_BAD);
532             }
533             } else {
534                 return (CRYPTO_ARGUMENTS_BAD);
535             }
536         }
537     }

538     rv = dup_mech_names(provider, array, count, KM_SLEEP);
539     KCF_PROV_REFRELE(provider);
540     if (modh != NULL)
541         (void) ddi_modclose(modh);
542     return (rv);
543 }

544 /*
545  * Change the mechanism list for a provider.
546  * If "direction" is CRYPTO_MECH_ADDED, add new mechanisms.
547  * If "direction" is CRYPTO_MECH_REMOVED, remove the mechanism list.
548  * Called from crypto_load_dev_disabled().
549  */
550 static void
551 kcf_change_mechs(kcf_provider_desc_t *provider, uint_t count,
552                 crypto_mech_name_t *array, crypto_event_change_t direction)
553 {
554     crypto_notify_event_change_t ec;
555     crypto_mech_info_t *mi;
556     kcf_prov_mech_desc_t *pmd;
557     char *mech;
558     int i, j, n;

559     ASSERT(direction == CRYPTO_MECH_ADDED ||
560            direction == CRYPTO_MECH_REMOVED);

561     if (provider == NULL) {
562         /*
563          * Nothing to add or remove from the tables since
564          * the provider isn't registered.
565          */
566         return;
567     }

568     for (i = 0; i < count; i++) {
569         if (array[i][0] == '\0')
570             continue;

```

```

571         mech = &array[i][0];

572         n = provider->pd_mech_list_count;
573         for (j = 0; j < n; j++) {
574             mi = &provider->pd_mechanisms[j];
575             if (strncmp(mi->cm_mech_name, mech,
576                       CRYPTO_MAX_MECH_NAME) == 0)
577                 break;
578         }
579         if (j == n)
580             continue;

581         switch (direction) {
582         case CRYPTO_MECH_ADDED:
583             (void) kcf_add_mech_provider(j, provider, &pmd);
584             break;

585         case CRYPTO_MECH_REMOVED:
586             kcf_remove_mech_provider(mech, provider);
587             break;
588         }

589         /* Inform interested clients of the event */
590         ec.ec_provider_type = provider->pd_prov_type;
591         ec.ec_change = direction;

592         (void) strncpy(ec.ec_mech_name, mech, CRYPTO_MAX_MECH_NAME);
593         kcf_walk_ntfylist(CRYPTO_EVENT_MECHS_CHANGED, &ec);
594     }
595 }

596 unchanged portion omitted

597 /*
598  * Called from CRYPTO_LOAD_SOFT_DISABLED ioctl.
599  * If new_count is 0, then completely remove the entry.
600  */
601 int
602 crypto_load_soft_disabled(char *name, uint_t new_count,
603                           crypto_mech_name_t *new_array)
604 {
605     kcf_provider_desc_t *provider = NULL;
606     crypto_mech_name_t *prev_array;
607     uint_t prev_count = 0;
608     int rv;

609     provider = kcf_prov_tab_lookup_by_name(name);
610     if (provider != NULL) {
611         mutex_enter(&provider->pd_lock);
612         /*
613          * Check if any other thread is disabling or removing
614          * this provider. We return if this is the case.
615          */
616         if (provider->pd_state >= KCF_PROV_DISABLED) {
617             mutex_exit(&provider->pd_lock);
618             KCF_PROV_REFRELE(provider);
619             return (CRYPTO_BUSY);
620         }
621         provider->pd_state = KCF_PROV_DISABLED;
622         mutex_exit(&provider->pd_lock);

623         undo_register_provider(provider, B_TRUE);
624         KCF_PROV_REFRELE(provider);
625         if (provider->pd_kstat != NULL)
626             KCF_PROV_REFRELE(provider);

```

```

718     mutex_enter(&provider->pd_lock);
719     /* Wait till the existing requests complete. */
720     while (provider->pd_state != KCF_PROV_FREED) {
721         cv_wait(&provider->pd_remove_cv, &provider->pd_lock);
722     }
723     mutex_exit(&provider->pd_lock);
724 }

726 if (new_count == 0) {
727     kcf_policy_remove_by_name(name, &prev_count, &prev_array);
728     crypto_free_mech_list(prev_array, prev_count);
729     rv = CRYPTO_SUCCESS;
730     goto out;
731 }

733 /* put disabled mechanisms into policy table */
734 if ((rv = kcf_policy_load_soft_disabled(name, new_count, new_array,
735     &prev_count, &prev_array)) == CRYPTO_SUCCESS) {
736     crypto_free_mech_list(prev_array, prev_count);
737 }

739 out:
740 if (provider != NULL) {
741     redo_register_provider(provider);
742     if (provider->pd_kstat != NULL)
743         KCF_PROV_REFHOLD(provider);
744     mutex_enter(&provider->pd_lock);
745     provider->pd_state = KCF_PROV_READY;
746     mutex_exit(&provider->pd_lock);
747 } else if (rv == CRYPTO_SUCCESS) {
748     /*
749      * There are some cases where it is useful to kcf clients
750      * to have a provider whose mechanism is enabled now to be
751      * available. So, we attempt to load it here.
752      *
753      * The check, new_count < prev_count, ensures that we do this
754      * only in the case where a mechanism(s) is now enabled.
755      * This check assumes that enable and disable are separate
756      * administrative actions and are not done in a single action.
757      */
758     if ((new_count < prev_count) &&
759         if (new_count < prev_count && (in_soft_config_list(name)) &&
760             (modload("crypto", name) != -1)) {
761         struct modctl *mcp;
762         boolean_t load_again = B_FALSE;

763         if ((mcp = mod_hold_by_name(name)) != NULL) {
764             mcp->mod_loadflags |= MOD_NOAUTOUNLOAD;

766             /* memory pressure may have unloaded module */
767             if (!mcp->mod_installed)
768                 load_again = B_TRUE;
769             mod_release_mod(mcp);

771             if (load_again)
772                 (void) modload("crypto", name);
773         }
774     }
775 }

777     return (rv);
778 }

```

unchanged portion omitted

```

787 /*
788 * Unload a kernel software crypto module.

```

```

789 * Called from the CRYPTO_UNLOAD_SOFT_MODULE ioctl.
790 */
791 /* called from the CRYPTO_UNLOAD_SOFT_MODULE ioctl */
792 int
793 crypto_unload_soft_module(caddr_t name)
794 {
795     int error;
796     modid_t id;
797     kcf_provider_desc_t *provider;
798     struct modctl *mcp;

799     /* verify that 'name' refers to a registered crypto provider */
800     if ((provider = kcf_prov_tab_lookup_by_name(name)) == NULL)
801         return (CRYPTO_UNKNOWN_PROVIDER);

803     /*
804      * We save the module id and release the reference. We need to
805      * do this as modunload() calls unregister which waits for the
806      * refcnt to drop to zero.
807      */
808     id = provider->pd_module_id;
809     KCF_PROV_REFRELE(provider);

811     if ((mcp = mod_hold_by_name(name)) != NULL) {
812         mcp->mod_loadflags &= ~(MOD_NOAUTOUNLOAD);
813         mod_release_mod(mcp);
814     }

816     if ((error = modunload(id)) != 0) {
817         return (error == EBUSY ? CRYPTO_BUSY : CRYPTO_FAILED);
818     }

820     return (CRYPTO_SUCCESS);
821 }

823 /*
824 * Free the list of kernel hardware crypto providers.
825 * Called by get_dev_list() for the CRYPTO_GET_DEV_LIST ioctl.
826 */
827 /* called from CRYPTO_GET_DEV_LIST ioctl */
828 void
829 crypto_free_dev_list(crypto_dev_list_entry_t *array, uint_t count)
830 {
831     if (count == 0 || array == NULL)
832         return;

833     kmem_free(array, count * sizeof (crypto_dev_list_entry_t));
834 }

```

unchanged portion omitted

```

1016 /*
1017 * Free memory for elements in a kcf_soft_config_entry_t. This entry must
1018 * have been previously removed from the soft_config_list linked list.
1019 */
1020 static void
1021 free_soft_config_entry(kcf_soft_conf_entry_t *p)
1022 {
1023     kmem_free(p->ce_name, strlen(p->ce_name) + 1);
1024     crypto_free_mech_list(p->ce_mechs, p->ce_count);
1025     kmem_free(p, sizeof (kcf_soft_conf_entry_t));
1026 }

1028 /*
1029 * Store configuration information for software providers in a linked list.
1030 * Called from the CRYPTO_LOAD_SOFT_CONFIG ioctl, this routine stores
1031 * configuration information for software providers in a linked list.

```

```

1030 * If the list already contains an entry for the specified provider
1031 * and the specified mechanism list has at least one mechanism, then
1032 * the mechanism list for the provider is updated. If the mechanism list
1033 * is empty, the entry for the provider is removed.
1034 *
1035 * Called from kcf_soft_config_init() (to initially populate the list
1036 * with default kernel providers) and from crypto_load_soft_config() for
1037 * the CRYPTO_LOAD_SOFT_CONFIG ioctl (for third-party kernel modules).
1038 *
1039 * Important note: the name and array arguments must be allocated memory
1040 * and are consumed in soft_config_list.
1041 *
1042 * Important note: the array argument is consumed.
1041 */
1042 static int
1043 add_soft_config(char *name, uint_t count, crypto_mech_name_t *array)
1044 {
1045     static uint_t soft_config_count = 0;
1046     kcf_soft_conf_entry_t *prev = NULL, *entry = NULL, *new_entry, *p;
1047     size_t name_len;
1048
1049     /*
1050      * Allocate storage for a new entry.
1051      * Free later if an entry already exists.
1052      */
1053     name_len = strlen(name) + 1;
1054     new_entry = kmem_zalloc(sizeof(kcf_soft_conf_entry_t), KM_SLEEP);
1055     new_entry->ce_name = kmem_alloc(name_len, KM_SLEEP);
1056     (void) strcpy(new_entry->ce_name, name);
1057
1058     mutex_enter(&soft_config_mutex);
1059     p = soft_config_list;
1060     if (p != NULL) {
1061         do {
1062             if (strncmp(name, p->ce_name, MAXNAMELEN) == 0) {
1063                 entry = p;
1064                 break;
1065             }
1066             prev = p;
1067         } while ((p = p->ce_next) != NULL);
1068
1069     }
1070
1071     if (entry == NULL) {
1072         if (count == 0) {
1073             mutex_exit(&soft_config_mutex);
1074             kmem_free(new_entry->ce_name, name_len);
1075             kmem_free(new_entry, sizeof(kcf_soft_conf_entry_t));
1076             return (CRYPTO_SUCCESS);
1077         }
1078
1079         if (soft_config_count > KCF_MAX_CONFIG_ENTRIES) {
1080             mutex_exit(&soft_config_mutex);
1081             kmem_free(new_entry->ce_name, name_len);
1082             kmem_free(new_entry, sizeof(kcf_soft_conf_entry_t));
1083             cmn_err(CE_WARN, "out of soft_config_list entries");
1084             return (CRYPTO_FAILED);
1085         }
1086
1087         /* add to head of list */
1088         new_entry->ce_next = soft_config_list;
1089         soft_config_list = new_entry;
1090         soft_config_count++;
1091         entry = new_entry;
1092     } else { /* mechanism already in list */
1093         } else {
1094             kmem_free(new_entry->ce_name, name_len);

```

```

1094         kmem_free(new_entry, sizeof(kcf_soft_conf_entry_t));
1095     }
1096
1097     /* mechanism count == 0 means remove entry from list */
1098     if (count == 0) {
1099         if (prev == NULL) {
1100             /* remove first in list */
1101             soft_config_list = entry->ce_next;
1102         } else {
1103             prev->ce_next = entry->ce_next;
1104         }
1105         soft_config_count--;
1106         mutex_exit(&soft_config_mutex);
1107
1108         /* free entry */
1109         free_soft_config_entry(entry);
1110
1111         return (CRYPTO_SUCCESS);
1112     }
1113
1114     /* replace mechanisms */
1115     if (entry->ce_mechs != NULL)
1116         crypto_free_mech_list(entry->ce_mechs, entry->ce_count);
1117
1118     entry->ce_mechs = array;
1119     entry->ce_count = count;
1120     mutex_exit(&soft_config_mutex);
1121
1122     return (CRYPTO_SUCCESS);
1123 }
1124
1125 /*
1126 * This routine searches the soft_config_list for the first entry that
1127 * has the specified mechanism in its mechanism list. If found,
1128 * a buffer containing the name of the software module that implements
1129 * the mechanism is allocated and stored in 'name'.
1130 */
1131 int
1132 get_sw_provider_for_mech(crypto_mech_name_t mech, char **name)
1133 {
1134     kcf_soft_conf_entry_t *p, *next;
1135     char tmp_name[MAXNAMELEN];
1136     size_t name_len = 0;
1137     int i;
1138
1139     mutex_enter(&soft_config_mutex);
1140     p = soft_config_list;
1141     while (p != NULL) {
1142         next = p->ce_next;
1143         for (i = 0; i < p->ce_count; i++) {
1144             if (strcmp(mech, &p->ce_mechs[i][0]) == 0) {
1145                 name_len = strlen(p->ce_name) + 1;
1146                 bcopy(p->ce_name, tmp_name, name_len);
1147                 break;
1148             }
1149         }
1150         p = next;
1151     }
1152     mutex_exit(&soft_config_mutex);
1153
1154     if (name_len == 0)
1155         return (CRYPTO_FAILED);
1156
1157     *name = kmem_alloc(name_len, KM_SLEEP);
1158     bcopy(tmp_name, *name, name_len);

```



```
1160         return (CRYPTO_SUCCESS);
953     }

955     /*
956     * This routine searches the soft_config_list for the specified
957     * software provider, returning B_TRUE if it is in the list.
958     */
959     boolean_t
960     in_soft_config_list(char *provider_name)
961     {
962         kcf_soft_conf_entry_t *p;
963         boolean_t rv = B_FALSE;

965         mutex_enter(&soft_config_mutex);
966         for (p = soft_config_list; p != NULL; p = p->ce_next) {
967             if (strcmp(provider_name, p->ce_name) == 0) {
968                 rv = B_TRUE;
969                 break;
970             }
971         }
972         mutex_exit(&soft_config_mutex);
973         return (rv);
1161     }
unchanged_portion_omitted
```

```

*****
25851 Tue Oct 28 16:46:23 2008
new/usr/src/uts/common/crypto/core/kcf_prov_tabs.c
6414175 kcf.conf's supportedlist not providing much usefulness
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 * Copyright 2007 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */
26 #pragma ident      "%Z%M% %I%      %E% SMI"
27
28 /*
29 * This file is part of the core Kernel Cryptographic Framework.
30 * It implements the management of tables of Providers. Entries to
31 * added and removed when cryptographic providers register with
32 * and unregister from the framework, respectively. The KCF scheduler
33 * and ioctl pseudo driver call this function to obtain the list
34 * of available providers.
35 *
36 * The provider table is indexed by crypto_provider_id_t. Each
37 * element of the table contains a pointer to a provider descriptor,
38 * or NULL if the entry is free.
39 *
40 * This file also implements helper functions to allocate and free
41 * provider descriptors.
42 */
43
44 #include <sys/types.h>
45 #include <sys/kmem.h>
46 #include <sys/cmn_err.h>
47 #include <sys/ddi.h>
48 #include <sys/sunddi.h>
49 #include <sys/ksynch.h>
50 #include <sys/crypto/common.h>
51 #include <sys/crypto/impl.h>
52 #include <sys/crypto/sched_impl.h>
53 #include <sys/crypto/spi.h>
54
55 #define KCF_MAX_PROVIDERS      512      /* max number of providers */
56
57 /*
58 * Prov_tab is an array of providers which is updated when
59 * a crypto provider registers with kcf. The provider calls the
60 * SPI routine, crypto_register_provider(), which in turn calls

```

```

59 * kcf_prov_tab_add_provider().
60 *
61 * A provider unregisters by calling crypto_unregister_provider()
62 * which triggers the removal of the prov_tab entry.
63 * It also calls kcf_remove_mech_provider().
64 *
65 * prov_tab entries are not updated from kcf.conf or by cryptoadm(1M).
66 */
67 static kcf_provider_desc_t **prov_tab = NULL;
68 static kmutex_t prov_tab_mutex; /* ensure exclusive access to the table */
69 static kcf_provider_desc_t **prov_tab = NULL;
70 static uint_t prov_tab_num = 0; /* number of providers in table */
71 static uint_t prov_tab_max = KCF_MAX_PROVIDERS;
72
73 #if DEBUG
74 extern int kcf_frmwrk_debug;
75 static void kcf_prov_tab_dump(char *message);
76 static void kcf_prov_tab_dump(void);
77 #endif /* DEBUG */
78
79 /*
80 * Initialize a mutex and the KCF providers table, prov_tab.
81 * The providers table is dynamically allocated with prov_tab_max entries.
82 * Called from kcf module _init().
83 * Initialize the providers table. The providers table is dynamically
84 * allocated with prov_tab_max entries.
85 */
86 void
87 kcf_prov_tab_init(void)
88 {
89     mutex_init(&prov_tab_mutex, NULL, MUTEX_DRIVER, NULL);
90
91     prov_tab = kmem_zalloc(prov_tab_max * sizeof(kcf_provider_desc_t *),
92                             KM_SLEEP);
93 }
94
95 /*
96 * Add a provider to the provider table. If no free entry can be found
97 * for the new provider, returns CRYPTO_HOST_MEMORY. Otherwise, add
98 * the provider to the table, initialize the pd_prov_id field
99 * of the specified provider descriptor to the index in that table,
100 * and return CRYPTO_SUCCESS. Note that a REFHOLD is done on the
101 * provider when pointed to by a table entry.
102 */
103 int
104 kcf_prov_tab_add_provider(kcf_provider_desc_t *prov_desc)
105 {
106     uint_t i;
107
108     ASSERT(prov_tab != NULL);
109
110     mutex_enter(&prov_tab_mutex);
111
112     /* find free slot in providers table */
113     for (i = 0; i < KCF_MAX_PROVIDERS && prov_tab[i] != NULL; i++)
114         ;
115     if (i == KCF_MAX_PROVIDERS) {
116         /* ran out of providers entries */
117         mutex_exit(&prov_tab_mutex);
118         cmn_err(CE_WARN, "out of providers entries");
119         return (CRYPTO_HOST_MEMORY);
120     }
121
122     /* initialize entry */
123     prov_tab[i] = prov_desc;

```

```

121     KCF_PROV_REFHOLD prov_desc;
122     KCF_PROV_IREFHOLD prov_desc;
123     prov_tab_num++;

125     mutex_exit(&prov_tab_mutex);

127     /* update provider descriptor */
128     prov_desc->pd_prov_id = i;

130     /*
131     * The KCF-private provider handle is defined as the internal
132     * provider id.
133     */
134     prov_desc->pd_kcf_prov_handle =
135     (crypto_kcf_provider_handle_t)prov_desc->pd_prov_id;

137 #if DEBUG
138     if (kcf_frmwrk_debug >= 1)
139         kcf_prov_tab_dump("kcf_prov_tab_add_provider");
140 #endif /* DEBUG */

142     return (CRYPTO_SUCCESS);
143 }

145 /*
146 * Remove the provider specified by its id. A REFRELE is done on the
147 * corresponding provider descriptor before this function returns.
148 * Returns CRYPTO_UNKNOWN_PROVIDER if the provider id is not valid.
149 */
150 int
151 kcf_prov_tab_rem_provider(crypto_provider_id_t prov_id)
152 {
153     kcf_provider_desc_t *prov_desc;

155     ASSERT(prov_tab != NULL);
156     ASSERT(prov_tab_num >= 0);

158     /*
159     * Validate provider id, since it can be specified by a 3rd-party
160     * provider.
161     */

163     mutex_enter(&prov_tab_mutex);
164     if (prov_id >= KCF_MAX_PROVIDERS ||
165         ((prov_desc = prov_tab[prov_id]) == NULL)) {
166         mutex_exit(&prov_tab_mutex);
167         return (CRYPTO_INVALID_PROVIDER_ID);
168     }
169     mutex_exit(&prov_tab_mutex);

171     /*
172     * The provider id must remain valid until the associated provider
173     * descriptor is freed. For this reason, we simply release our
174     * reference to the descriptor here. When the reference count
175     * reaches zero, kcf_free_provider_desc() will be invoked and
176     * the associated entry in the providers table will be released
177     * at that time.
178     */

180     KCF_PROV_REFRELE(prov_desc);
181     KCF_PROV_IREFRELE(prov_desc);

183 #if DEBUG
184     if (kcf_frmwrk_debug >= 1)
185         kcf_prov_tab_dump("kcf_prov_tab_rem_provider");

```

```

173         kcf_prov_tab_dump();
186 #endif /* DEBUG */

188     return (CRYPTO_SUCCESS);
189 }

    unchanged_portion_omitted

831 #if DEBUG
832 /*
833 * Dump the Kernel crypto providers table, prov_tab.
834 * If kcf_frmwrk_debug is >=2, also dump the mechanism lists.
835 */

836 static void
837 kcf_prov_tab_dump(char *message)
822 kcf_prov_tab_dump(void)
838 {
839     uint_t i, j;
824     uint_t i;

841     mutex_enter(&prov_tab_mutex);
842     printf("Providers table prov_tab at %s:\n",
843         message != NULL ? message : "");

828     printf("Providers table:\n");
845     for (i = 0; i < KCF_MAX_PROVIDERS; i++) {
846         kcf_provider_desc_t *p = prov_tab[i];
847         if (p != NULL) {
848             printf("[%d]: (%s) %d mechanisms, %s\n", i,
849                 (p->pd_prov_type == CRYPTO_HW_PROVIDER) ?
850                 "HW" : "SW",
851                 p->pd_mech_list_count, p->pd_description);
852             if (kcf_frmwrk_debug >= 2) {
853                 printf("\tprovider mechanisms: ");
854                 for (j = 0; j < p->pd_mech_list_count; ++j) {
855                     printf("%s \n",
856                         p->pd_mechanisms[j].cm_mech_name);
830                 if (prov_tab[i] != NULL) {
831                     printf("[%d]: (%s) %s\n",
832                         i, (prov_tab[i]->pd_prov_type ==
833                             CRYPTO_HW_PROVIDER) ? "HW" : "SW",
834                         prov_tab[i]->pd_description);
857                 }
858                 printf("\n");
859             }
860         }
861     }
862     printf("(end of providers table)\n");

864     mutex_exit(&prov_tab_mutex);
865 }

    unchanged_portion_omitted

```