```
**********************************************************
    1758 Fri Jun 13 16:44:29 2008
new/usr/src/common/crypto/THIRDPARTYLICENSE.cryptogams
6704653 THIRDPARTYLICENSE fixes for open source crypto source
**********************************************************
```

```
Copyright (c) 2006, CRYPTOGAMS by <appro@openssl.org>
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

      * Redistributions of source code must retain copyright notices,
        this list of conditions and the following disclaimer.

      * Redistributions in binary form must reproduce the above
        copyright notice, this list of conditions and the following
        disclaimer in the documentation and/or other materials
        provided with the distribution.

      * Neither the name of the CRYPTOGAMS nor the names of its
        copyright holder and contributors may be used to endorse or
        promote products derived from this software without specific
        prior written permission.

ALTERNATIVELY, provided that this notice is retained in full, this
product may be distributed under the terms of the GNU General Public
License (GPL), in which case the provisions of the GPL apply INSTEAD OF
those given above.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
**********************************************************
      50 Fri Jun 13 16:44:30 2008
new/usr/src/common/crypto/THIRDPARTYLICENSE.cryptogams.descrip
6704653 THIRDPARTYLICENSE fixes for open source crypto source
**********************************************************
   1 PORTIONS OF ARCFOUR, SHA1, AND SHA2 FUNCTIONALITY
```

```
**********************************************************
   45541 Fri Jun 13 16:44:31 2008
new/usr/src/common/crypto/aes/aes_cbc_crypt.c
5072963 Need an optimized AES implementation for amd64
6699938 CCM max payload computation is off by one
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "@(#)aes_cbc_crypt.c    1.10    08/06/13 SMI"
#pragma ident   "@(#)aes_cbc_crypt.c    1.9     08/05/09 SMI"


#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/crypto/common.h>
#include <sys/strsun.h>
#include "aes_cbc_crypt.h"
#include "aes_impl.h"
#ifndef _KERNEL
#include <limits.h>
#include <strings.h>
#endif  /* !_KERNEL */

static int aes_ctr_ccm_mode_contiguous_blocks(aes_ctx_t *, char *, size_t,
    crypto_data_t *);
static void
encode_adata_len(ulong_t auth_data_len, uint8_t *encoded, size_t *encoded_len);
static void
aes_ccm_format_initial_blocks(uchar_t *nonce, ulong_t nonceSize,
    ulong_t authDataSize, uint8_t *b0, aes_ctx_t *aes_ctx);
static int
aes_ccm_decrypt_contiguous_blocks(aes_ctx_t *ctx, char *data, size_t length,
    crypto_data_t *out);

/*
 * Initialize by setting iov_or_mp to point to the current iovec or mp,
 * and by setting current_offset to an offset within the current iovec or mp.
 * and by setting current_offset to an offset within the current iovec or mp .
 */
static void
aes_init_ptrs(crypto_data_t *out, void **iov_or_mp, offset_t *current_offset)
{
        offset_t offset;
```



```
        switch (out->cd_format) {
        case CRYPTO_DATA_RAW:
                *current_offset = out->cd_offset;
                break;

        case CRYPTO_DATA_UIO: {
                uio_t *uiop = out->cd_uio;
                uintptr_t vec_idx;

                offset = out->cd_offset;
                for (vec_idx = 0; vec_idx < uiop->uio_iovcnt &&
                    offset >= uiop->uio_iov[vec_idx].iov_len;
                    offset -= uiop->uio_iov[vec_idx++].iov_len)
                        ;

                *current_offset = offset;
                *iov_or_mp = (void *)vec_idx;
                break;
        }

        case CRYPTO_DATA_MBLK: {
                mblk_t *mp;

                offset = out->cd_offset;
                for (mp = out->cd_mp; mp != NULL && offset >= MBLKL(mp);
                    offset -= MBLKL(mp), mp = mp->b_cont)
                        ;

                *current_offset = offset;
                *iov_or_mp = mp;
                break;

        }
        } /* end switch */
}
```

*_____unchanged_portion_omitted_*

```
static int
aes_cbc_encrypt_contiguous_blocks(aes_ctx_t *ctx, char *data, size_t length,
    crypto_data_t *out)
{

/* EXPORT DELETE START */

        size_t remainder = length;
        size_t need;
        uint8_t *datap = (uint8_t *)data;
        uint8_t *blockp;
        uint8_t *lastp;
        uint32_t tmp[4];
        void *iov_or_mp;
        offset_t offset;
        uint8_t *out_data_1;
        uint8_t *out_data_2;
        size_t out_data_1_len;

        if (length + ctx->ac_remainder_len < AES_BLOCK_LEN) {
                /* accumulate bytes here and return */
                bcopy(datap,
                    (uint8_t *)ctx->ac_remainder + ctx->ac_remainder_len,
                    length);
                ctx->ac_remainder_len += length;
                ctx->ac_copy_to = datap;
                return (0);
        }
```

```
212         lastp = (uint8_t *)ctx->ac_iv;
213         if (out != NULL)
214                 aes_init_ptrs(out, &iov_or_mp, &offset);
215
216         do {
217                 /* Unprocessed data from last call. */
218                 if (ctx->ac_remainder_len > 0) {
219                         need = AES_BLOCK_LEN - ctx->ac_remainder_len;
220
221                         if (need > remainder)
222                                 return (1);
223
224                         bcopy(datap, &((uint8_t *)ctx->ac_remainder)
225                             [ctx->ac_remainder_len], need);
226
227                         blockp = (uint8_t *)ctx->ac_remainder;
228                 } else {
229                         blockp = datap;
230                 }
231
232                 /* don't write on the plaintext */
233                 if (out != NULL) {
234                         if (IS_P2ALIGNED(blockp, sizeof (uint32_t))) {
235                                 /* LINTED: pointer alignment */
236                                 tmp[0] = *(uint32_t *)blockp;
237                                 /* LINTED: pointer alignment */
238                                 tmp[1] = *(uint32_t *)&blockp[4];
239                                 /* LINTED: pointer alignment */
240                                 tmp[2] = *(uint32_t *)&blockp[8];
241                                 /* LINTED: pointer alignment */
242                                 tmp[3] = *(uint32_t *)&blockp[12];
243                         } else {
244                                 uint8_t *tmp8 = (uint8_t *)tmp;
245
246                                 AES_COPY_BLOCK(blockp, tmp8);
247                         }
248                         blockp = (uint8_t *)tmp;
249                 }
250
251                 if (ctx->ac_flags & AES_CBC_MODE) {
252                         /*
253                          * XOR the previous cipher block or IV with the
254                          * current clear block. Check for alignment.
255                          */
256                         if (IS_P2ALIGNED2(blockp, lastp, sizeof (uint32_t))) {
256                         if (IS_P2ALIGNED(blockp, sizeof (uint32_t)) &&
257                             IS_P2ALIGNED(lastp, sizeof (uint32_t))) {
257                                 /* LINTED: pointer alignment */
258                                 *(uint32_t *)&blockp[0] ^=
259                                 /* LINTED: pointer alignment */
260                                     *(uint32_t *)&lastp[0];
261                                 /* LINTED: pointer alignment */
262                                 *(uint32_t *)&blockp[4] ^=
263                                 /* LINTED: pointer alignment */
264                                     *(uint32_t *)&lastp[4];
265                                 /* LINTED: pointer alignment */
266                                 *(uint32_t *)&blockp[8] ^=
267                                 /* LINTED: pointer alignment */
268                                     *(uint32_t *)&lastp[8];
269                                 /* LINTED: pointer alignment */
270                                 *(uint32_t *)&blockp[12] ^=
271                                 /* LINTED: pointer alignment */
272                                     *(uint32_t *)&lastp[12];
273                         } else {
274                                 AES_XOR_BLOCK(lastp, blockp);
```



```
275                         }
276                 }
277
278                 if (out == NULL) {
279                         aes_encrypt_block(ctx->ac_keysched, blockp, blockp);
280
281                         ctx->ac_lastp = blockp;
282                         lastp = blockp;
283
284                         if (ctx->ac_remainder_len > 0) {
285                                 bcopy(blockp, ctx->ac_copy_to,
286                                     ctx->ac_remainder_len);
287                                 bcopy(blockp + ctx->ac_remainder_len, datap,
288                                     need);
289                         }
290                 } else {
291                         aes_encrypt_block(ctx->ac_keysched, blockp, lastp);
292                         aes_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
293                             &out_data_1_len, &out_data_2, AES_BLOCK_LEN);
294
295                         /* copy block to where it belongs */
296                         if ((out_data_1_len == AES_BLOCK_LEN) &&
297                             (IS_P2ALIGNED2(lastp, out_data_1,
298                             sizeof (uint32_t)))) {
299                                 /* LINTED: pointer alignment */
300                                 uint32_t *d = (uint32_t *)out_data_1;
301                                 /* LINTED: pointer alignment */
302                                 d[0] = *(uint32_t *)lastp;
303                                 /* LINTED: pointer alignment */
304                                 d[1] = *(uint32_t *)&lastp[4];
305                                 /* LINTED: pointer alignment */
306                                 d[2] = *(uint32_t *)&lastp[8];
307                                 /* LINTED: pointer alignment */
308                                 d[3] = *(uint32_t *)&lastp[12];
309                         } else {
310                                 bcopy(lastp, out_data_1, out_data_1_len);
311                         }
312                         if (out_data_2 != NULL) {
313                                 bcopy(lastp + out_data_1_len, out_data_2,
314                                     AES_BLOCK_LEN - out_data_1_len);
315                         }
316
317                         /* update offset */
318                         out->cd_offset += AES_BLOCK_LEN;
319                 }
320
321                 /* Update pointer to next block of data to be processed. */
322                 if (ctx->ac_remainder_len != 0) {
323                         datap += need;
324                         ctx->ac_remainder_len = 0;
325                 } else {
326                         datap += AES_BLOCK_LEN;
327                 }
328
329                 remainder = (size_t)&data[length] - (size_t)datap;
330
331                 /* Incomplete last block. */
332                 if (remainder > 0 && remainder < AES_BLOCK_LEN) {
333                         bcopy(datap, ctx->ac_remainder, remainder);
334                         ctx->ac_remainder_len = remainder;
335                         ctx->ac_copy_to = datap;
336                         goto out;
337                 }
338                 ctx->ac_copy_to = NULL;
339
340         } while (remainder > 0);
```

```
342 out:
343         /*
344          * Save the last encrypted block in the context - but only for
345          * the CBC mode of operation.
346          */
347         if ((ctx->ac_flags & AES_CBC_MODE) && (ctx->ac_lastp != NULL)) {
348                 uint8_t *iv8 = (uint8_t *)ctx->ac_iv;
349                 uint8_t *last8 = (uint8_t *)ctx->ac_lastp;

351                 if (IS_P2ALIGNED(ctx->ac_lastp, sizeof (uint32_t))) {
352                         /* LINTED: pointer alignment */
353                         *(uint32_t *)iv8 = *(uint32_t *)last8;
354                         /* LINTED: pointer alignment */
355                         *(uint32_t *)&iv8[4] = *(uint32_t *)&last8[4];
356                         /* LINTED: pointer alignment */
357                         *(uint32_t *)&iv8[8] = *(uint32_t *)&last8[8];
358                         /* LINTED: pointer alignment */
359                         *(uint32_t *)&iv8[12] = *(uint32_t *)&last8[12];
360                 } else {
361                         AES_COPY_BLOCK(last8, iv8);
362                 }
363                 ctx->ac_lastp = (uint8_t *)ctx->ac_iv;
364         }

366 /* EXPORT DELETE END */

368         return (0);
369 }
```

unchanged_portion_omitted_

```
391 /* ARGSUSED */
392 static int
393 aes_cbc_decrypt_contiguous_blocks(aes_ctx_t *ctx, char *data, size_t length,
394     crypto_data_t *out)
395 {

397 /* EXPORT DELETE START */

399         size_t remainder = length;
400         size_t need;
401         uint8_t *datap = (uint8_t *)data;
402         uint8_t *blockp;
403         uint8_t *lastp;
404         uint32_t tmp[4];
405         void *iov_or_mp;
406         offset_t offset;
407         uint8_t *out_data_1;
408         uint8_t *out_data_2;
409         size_t out_data_1_len;

411         if (length + ctx->ac_remainder_len < AES_BLOCK_LEN) {
412                 /* accumulate bytes here and return */
413                 bcopy(datap,
414                     (uint8_t *)ctx->ac_remainder + ctx->ac_remainder_len,
415                     length);
416                 ctx->ac_remainder_len += length;
417                 ctx->ac_copy_to = datap;
418                 return (0);
419         }

421         lastp = ctx->ac_lastp;
422         if (out != NULL)
423                 aes_init_ptrs(out, &iov_or_mp, &offset);

425         do {
```



```
426                 /* Unprocessed data from last call. */
427                 if (ctx->ac_remainder_len > 0) {
428                         need = AES_BLOCK_LEN - ctx->ac_remainder_len;

430                         if (need > remainder)
431                                 return (1);

433                         bcopy(datap, &((uint8_t *)ctx->ac_remainder)
434                             [ctx->ac_remainder_len], need);

436                         blockp = (uint8_t *)ctx->ac_remainder;
437                 } else {
438                         blockp = datap;
439                 }

441                 if (ctx->ac_flags & AES_CBC_MODE) {

443                         /* Save current ciphertext block */
444                         if (IS_P2ALIGNED(blockp, sizeof (uint32_t))) {
445                                 uint32_t *tmp32;

447                                 /* LINTED: pointer alignment */
448                                 tmp32 = (uint32_t *)OTHER((uint64_t *)lastp,
449                                     ctx);

451                                 /* LINTED: pointer alignment */
452                                 *tmp32++ = *(uint32_t *)blockp;
453                                 /* LINTED: pointer alignment */
454                                 *tmp32++ = *(uint32_t *)&blockp[4];
455                                 /* LINTED: pointer alignment */
456                                 *tmp32++ = *(uint32_t *)&blockp[8];
457                                 /* LINTED: pointer alignment */
458                                 *tmp32++ = *(uint32_t *)&blockp[12];
459                         } else {
460                                 uint8_t *tmp8;
461                                 /* LINTED: pointer alignment */
462                                 tmp8 = (uint8_t *)OTHER((uint64_t *)lastp, ctx);

464                                 AES_COPY_BLOCK(blockp, tmp8);
465                         }
466                 }

468                 if (out != NULL) {
469                         aes_decrypt_block(ctx->ac_keysched, blockp,
470                             (uint8_t *)tmp);
471                         blockp = (uint8_t *)tmp;
472                 } else {
473                         aes_decrypt_block(ctx->ac_keysched, blockp, blockp);
474                 }

476                 if (ctx->ac_flags & AES_CBC_MODE) {
477                         /*
478                          * XOR the previous cipher block or IV with the
479                          * currently decrypted block.  Check for alignment.
480                          */
481                         if (IS_P2ALIGNED2(blockp, lastp, sizeof (uint32_t))) {
466                         if (IS_P2ALIGNED(blockp, sizeof (uint32_t)) &&
467                             IS_P2ALIGNED(lastp, sizeof (uint32_t))) {
482                                 /* LINTED: pointer alignment */
483                                 *(uint32_t *)blockp ^= *(uint32_t *)lastp;
484                                 /* LINTED: pointer alignment */
485                                 *(uint32_t *)&blockp[4] ^=
486                                 /* LINTED: pointer alignment */
487                                     *(uint32_t *)&lastp[4];
488                                 /* LINTED: pointer alignment */
489                                 *(uint32_t *)&blockp[8] ^=
```

```
490                                 /* LINTED: pointer alignment */
491                                     *(uint32_t *)&lastp[8];
492                                 /* LINTED: pointer alignment */
493                                 *(uint32_t *)&blockp[12] ^=
494                                 /* LINTED: pointer alignment */
495                                     *(uint32_t *)&lastp[12];
496                         } else {
497                                 AES_XOR_BLOCK(lastp, blockp);
498                         }

500                         /* LINTED: pointer alignment */
501                         lastp = (uint8_t *)OTHER((uint64_t *)lastp, ctx);
502                 }

504                 if (out != NULL) {
505                         aes_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
506                             &out_data_1_len, &out_data_2, AES_BLOCK_LEN);

508                         /* copy temporary block to where it belongs */
509                         if ((out_data_1_len == AES_BLOCK_LEN) &&
510                             (IS_P2ALIGNED(out_data_1, sizeof (uint32_t)))) {
511                                 /* LINTED: pointer alignment */
512                                 uint32_t *d = (uint32_t *)out_data_1;
513                                 d[0] = tmp[0];
514                                 d[1] = tmp[1];
515                                 d[2] = tmp[2];
516                                 d[3] = tmp[3];
517                         } else {
518                                 bcopy(&tmp, out_data_1, out_data_1_len);
519                         }
520                         if (out_data_2 != NULL) {
521                                 bcopy((uint8_t *)&tmp + out_data_1_len,
522                                     out_data_2, AES_BLOCK_LEN - out_data_1_len);
523                         }

525                         /* update offset */
526                         out->cd_offset += AES_BLOCK_LEN;

528                 } else if (ctx->ac_remainder_len > 0) {
529                         /* copy temporary block to where it belongs */
530                         bcopy(blockp, ctx->ac_copy_to, ctx->ac_remainder_len);
531                         bcopy(blockp + ctx->ac_remainder_len, datap, need);
532                 }

534                 /* Update pointer to next block of data to be processed. */
535                 if (ctx->ac_remainder_len != 0) {
536                         datap += need;
537                         ctx->ac_remainder_len = 0;
538                 } else {
539                         datap += AES_BLOCK_LEN;
540                 }

542                 remainder = (size_t)&data[length] - (size_t)datap;

544                 /* Incomplete last block. */
545                 if (remainder > 0 && remainder < AES_BLOCK_LEN) {
546                         bcopy(datap, ctx->ac_remainder, remainder);
547                         ctx->ac_remainder_len = remainder;
548                         ctx->ac_lastp = lastp;
549                         ctx->ac_copy_to = datap;
550                         return (0);
551                 }
552                 ctx->ac_copy_to = NULL;

554         } while (remainder > 0);
```



```
556         ctx->ac_lastp = lastp;

558 /* EXPORT DELETE END */

560         return (0);
561 }
_____unchanged_portion_omitted_

630 /*
631  * Encrypt and decrypt multiple blocks of data in counter mode.
632  * Encrypt multiple blocks of data in CCM mode.  Decrypt for CCM mode
633  * is done in another function.
634  */
635 /* ARGSUSED */
636 int
637 aes_ctr_ccm_mode_contiguous_blocks(aes_ctx_t *ctx, char *data, size_t length,
638     crypto_data_t *out)
639 {

641 /* EXPORT DELETE START */

643         size_t remainder = length;
644         size_t need;
645         uint8_t *datap = (uint8_t *)data;
646         uint8_t *blockp;
647         uint8_t *lastp;
648         uint32_t tmp[4];
649         uint32_t counter_block[4];
650         void *iov_or_mp;
651         offset_t offset;
652         uint8_t *out_data_1;
653         uint8_t *out_data_2;
654         size_t out_data_1_len;
655         uint64_t counter;
656         uint8_t *mac_buf;
657 #ifdef _LITTLE_ENDIAN
658         uint8_t *p;
659 #endif

661         if (length + ctx->ac_remainder_len < AES_BLOCK_LEN) {
662                 /* accumulate bytes here and return */
663                 bcopy(datap,
664                     (uint8_t *)ctx->ac_remainder + ctx->ac_remainder_len,
665                     length);
666                 ctx->ac_remainder_len += length;
667                 ctx->ac_copy_to = datap;
668                 return (0);
669         }

671         lastp = (uint8_t *)ctx->ac_cb;
672         if (out != NULL)
673                 aes_init_ptrs(out, &iov_or_mp, &offset);

675         if (ctx->ac_flags & AES_CCM_MODE) {
676                 mac_buf = (uint8_t *)ctx->ac_ccm_mac_buf;
677         }

679         do {
680                 /* Unprocessed data from last call. */
681                 if (ctx->ac_remainder_len > 0) {
682                         need = AES_BLOCK_LEN - ctx->ac_remainder_len;

684                         if (need > remainder)
685                                 return (1);

687                         bcopy(datap, &((uint8_t *)ctx->ac_remainder)
```

```
688                             [ctx->ac_remainder_len], need);

690                     blockp = (uint8_t *)ctx->ac_remainder;
691             } else {
692                     blockp = datap;
693             }

695             /* don't write on the plaintext */
696             if (out != NULL) {
697                     if (IS_P2ALIGNED(blockp, sizeof (uint32_t))) {
698                             /* LINTED: pointer alignment */
699                             tmp[0] = *(uint32_t *)blockp;
700                             /* LINTED: pointer alignment */
701                             tmp[1] = *(uint32_t *)&blockp[4];
702                             /* LINTED: pointer alignment */
703                             tmp[2] = *(uint32_t *)&blockp[8];
704                             /* LINTED: pointer alignment */
705                             tmp[3] = *(uint32_t *)&blockp[12];
706                     } else {
707                             uint8_t *tmp8 = (uint8_t *)tmp;

709                             AES_COPY_BLOCK(blockp, tmp8);
710                     }
711                     blockp = (uint8_t *)tmp;
712             }

714             if (ctx->ac_flags & AES_CCM_MODE) {
715                     /*
716                      * do CBC MAC
717                      *
718                      * XOR the previous cipher block current clear block.
719                      * mac_buf always contain previous cipher block.
720                      */
721                     if (IS_P2ALIGNED2(blockp, mac_buf, sizeof (uint32_t))) {
697                     if (IS_P2ALIGNED(blockp, sizeof (uint32_t)) &&
698                         IS_P2ALIGNED(mac_buf, sizeof (uint32_t))) {
722                             /* LINTED: pointer alignment */
723                             *(uint32_t *)&mac_buf[0] ^=
724                             /* LINTED: pointer alignment */
725                                 *(uint32_t *)&blockp[0];
726                             /* LINTED: pointer alignment */
727                             *(uint32_t *)&mac_buf[4] ^=
728                             /* LINTED: pointer alignment */
729                                 *(uint32_t *)&blockp[4];
730                             /* LINTED: pointer alignment */
731                             *(uint32_t *)&mac_buf[8] ^=
732                             /* LINTED: pointer alignment */
733                                 *(uint32_t *)&blockp[8];
734                             /* LINTED: pointer alignment */
735                             *(uint32_t *)&mac_buf[12] ^=
736                             /* LINTED: pointer alignment */
737                                 *(uint32_t *)&blockp[12];
738                     } else {
739                             AES_XOR_BLOCK(blockp, mac_buf);
740                     }
741                     aes_encrypt_block(ctx->ac_keysched, mac_buf, mac_buf);
742             }


745             /* ac_cb is the counter block */
746             aes_encrypt_block(ctx->ac_keysched, (uint8_t *)ctx->ac_cb,
747                 (uint8_t *)counter_block);

749             lastp = (uint8_t *)counter_block;

751             /*
```



```
752              * Increment counter. Counter bits are confined
753              * to the bottom 64 bits of the counter block.
754              */
755             counter = ctx->ac_cb[1] & ctx->ac_counter_mask;
756 #ifdef _LITTLE_ENDIAN
757             p = (uint8_t *)&counter;
758             counter = (((uint64_t)p[0] << 56) |
759                 ((uint64_t)p[1] << 48) |
760                 ((uint64_t)p[2] << 40) |
761                 ((uint64_t)p[3] << 32) |
762                 ((uint64_t)p[4] << 24) |
763                 ((uint64_t)p[5] << 16) |
764                 ((uint64_t)p[6] << 8) |
765                 (uint64_t)p[7]);
766 #endif
767             counter++;
768 #ifdef _LITTLE_ENDIAN
769             counter = (((uint64_t)p[0] << 56) |
770                 ((uint64_t)p[1] << 48) |
771                 ((uint64_t)p[2] << 40) |
772                 ((uint64_t)p[3] << 32) |
773                 ((uint64_t)p[4] << 24) |
774                 ((uint64_t)p[5] << 16) |
775                 ((uint64_t)p[6] << 8) |
776                 (uint64_t)p[7]);
777 #endif
778             counter &= ctx->ac_counter_mask;
779             ctx->ac_cb[1] =
780                 (ctx->ac_cb[1] & ~(ctx->ac_counter_mask)) | counter;

782             /*
783              * XOR the previous cipher block or IV with the
784              * current clear block. Check for alignment.
785              */
786             if (IS_P2ALIGNED2(blockp, lastp, sizeof (uint32_t))) {
763             if (IS_P2ALIGNED(blockp, sizeof (uint32_t)) &&
764                 IS_P2ALIGNED(lastp, sizeof (uint32_t))) {
787                     /* LINTED: pointer alignment */
788                     *(uint32_t *)&blockp[0] ^=
789                     /* LINTED: pointer alignment */
790                         *(uint32_t *)&lastp[0];
791                     /* LINTED: pointer alignment */
792                     *(uint32_t *)&blockp[4] ^=
793                     /* LINTED: pointer alignment */
794                         *(uint32_t *)&lastp[4];
795                     /* LINTED: pointer alignment */
796                     *(uint32_t *)&blockp[8] ^=
797                     /* LINTED: pointer alignment */
798                         *(uint32_t *)&lastp[8];
799                     /* LINTED: pointer alignment */
800                     *(uint32_t *)&blockp[12] ^=
801                     /* LINTED: pointer alignment */
802                         *(uint32_t *)&lastp[12];
803             } else {
804                     AES_XOR_BLOCK(lastp, blockp);
805             }

807             ctx->ac_lastp = blockp;
808             lastp = blockp;
809             if (ctx->ac_flags & AES_CCM_MODE) {
810                     ctx->ac_ccm_processed_data_len += AES_BLOCK_LEN;
811             }

813             if (out == NULL) {
814                     if (ctx->ac_remainder_len > 0) {
815                             bcopy(blockp, ctx->ac_copy_to,
```

```
816                                     ctx->ac_remainder_len);
817                                 bcopy(blockp + ctx->ac_remainder_len, datap,
818                                     need);
819                         }
820                 } else {
821                         aes_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
822                             &out_data_1_len, &out_data_2, AES_BLOCK_LEN);

824                         /* copy block to where it belongs */
825                         if ((out_data_1_len == AES_BLOCK_LEN) &&
826                             (IS_P2ALIGNED2(lastp, out_data_1,
827                             sizeof (uint32_t)))) {
828                                 /* LINTED: pointer alignment */
829                                 uint32_t *d = (uint32_t *)out_data_1;
830                                 /* LINTED: pointer alignment */
831                                 d[0] = *(uint32_t *)lastp;
832                                 /* LINTED: pointer alignment */
833                                 d[1] = *(uint32_t *)&lastp[4];
834                                 /* LINTED: pointer alignment */
835                                 d[2] = *(uint32_t *)&lastp[8];
836                                 /* LINTED: pointer alignment */
837                                 d[3] = *(uint32_t *)&lastp[12];
838                         } else {
839                                 bcopy(lastp, out_data_1, out_data_1_len);
840                         }
841                         if (out_data_2 != NULL) {
842                                 bcopy(lastp + out_data_1_len, out_data_2,
843                                     AES_BLOCK_LEN - out_data_1_len);
844                         }

846                         /* update offset */
847                         out->cd_offset += AES_BLOCK_LEN;
848                 }

850                 /* Update pointer to next block of data to be processed. */
851                 if (ctx->ac_remainder_len != 0) {
852                         datap += need;
853                         ctx->ac_remainder_len = 0;
854                 } else {
855                         datap += AES_BLOCK_LEN;
856                 }

858                 remainder = (size_t)&data[length] - (size_t)datap;

860                 /* Incomplete last block. */
861                 if (remainder > 0 && remainder < AES_BLOCK_LEN) {
862                         bcopy(datap, ctx->ac_remainder, remainder);
863                         ctx->ac_remainder_len = remainder;
864                         ctx->ac_copy_to = datap;
865                         goto out;
866                 }
867                 ctx->ac_copy_to = NULL;

869         } while (remainder > 0);

871 out:

873 /* EXPORT DELETE END */

875         return (0);
876 }

878 /*
879  * The following function should be call at encrypt or decrypt init time
880  * for AES CCM mode.
881  */
```



```
882 int
883 aes_ccm_init(aes_ctx_t *ctx, unsigned char *nonce, size_t nonce_len,
884     unsigned char *auth_data, size_t auth_data_len)
885 {
886 /* EXPORT DELETE START */
887         uint8_t *mac_buf, *datap, *ivp, *authp;
888         uint32_t iv[4], tmp[4];
889         size_t remainder, processed;
890         uint8_t encoded_a[10]; /* max encoded auth data length is 10 octets */
891         size_t encoded_a_len = 0;

893         mac_buf = (uint8_t *)&(ctx->ac_ccm_mac_buf);

895         /*
896          * Format the 1st block for CBC-MAC and construct the
897          * 1st counter block.
898          *
899          * aes_ctx->ac_iv is used for storing the counter block
900          * mac_buf will store b0 at this time.
901          */
902         aes_ccm_format_initial_blocks(nonce, nonce_len,
903             auth_data_len, mac_buf, ctx);

905         /* The IV for CBC MAC for AES CCM mode is always zero */
906         bzero(iv, AES_BLOCK_LEN);
907         ivp = (uint8_t *)iv;

909         if (IS_P2ALIGNED2(ivp, mac_buf, sizeof (uint32_t))) {
871         if (IS_P2ALIGNED(ivp, sizeof (uint32_t)) &&
872             IS_P2ALIGNED(mac_buf, sizeof (uint32_t))) {
910                 /* LINTED: pointer alignment */
911                 *(uint32_t *)&mac_buf[0] ^= *(uint32_t *)&ivp[0];
912                 /* LINTED: pointer alignment */
913                 *(uint32_t *)&mac_buf[4] ^= *(uint32_t *)&ivp[4];
914                 /* LINTED: pointer alignment */
915                 *(uint32_t *)&mac_buf[8] ^= *(uint32_t *)&ivp[8];
916                 /* LINTED: pointer alignment */
917                 *(uint32_t *)&mac_buf[12] ^= *(uint32_t *)&ivp[12];
918         } else {
919                 AES_XOR_BLOCK(ivp, mac_buf);
920         }

922         /* encrypt the nonce */
923         aes_encrypt_block(ctx->ac_keysched, mac_buf, mac_buf);

925         /* take care of the associated data, if any */
926         if (auth_data_len == 0) {
927                 return (0);
928         }

930         encode_adata_len(auth_data_len, encoded_a, &encoded_a_len);

932         remainder = auth_data_len;

934         /* 1st block: it contains encoded associated data, and some data */
935         authp = (uint8_t *)tmp;
936         bzero(authp, AES_BLOCK_LEN);
937         bcopy(encoded_a, authp, encoded_a_len);
938         processed = AES_BLOCK_LEN - encoded_a_len;
939         if (processed > auth_data_len) {
940                 /* in case auth_data is very small */
941                 processed = auth_data_len;
942         }
943         bcopy(auth_data, authp+encoded_a_len, processed);
944         /* xor with previous buffer */
945         if (IS_P2ALIGNED2(authp, mac_buf, sizeof (uint32_t))) {
```

```
 908         if (IS_P2ALIGNED(authp, sizeof (uint32_t)) &&
 909             IS_P2ALIGNED(mac_buf, sizeof (uint32_t))) {
 946                 /* LINTED: pointer alignment */
 947                 *(uint32_t *)&mac_buf[0] ^= *(uint32_t *)&authp[0];
 948                 /* LINTED: pointer alignment */
 949                 *(uint32_t *)&mac_buf[4] ^= *(uint32_t *)&authp[4];
 950                 /* LINTED: pointer alignment */
 951                 *(uint32_t *)&mac_buf[8] ^= *(uint32_t *)&authp[8];
 952                 /* LINTED: pointer alignment */
 953                 *(uint32_t *)&mac_buf[12] ^= *(uint32_t *)&authp[12];
 954         } else {
 955                 AES_XOR_BLOCK(authp, mac_buf);
 956         }
 957         aes_encrypt_block(ctx->ac_keysched, mac_buf, mac_buf);
 958         remainder -= processed;
 959         if (remainder == 0) {
 960                 /* a small amount of associated data, it's all done now */
 961                 return (0);
 962         }

 964         do {
 965                 if (remainder < AES_BLOCK_LEN) {
 966                         /*
 967                          * There's not a block full of data, pad rest of
 968                          * buffer with zero
 969                          */
 970                         bzero(authp, AES_BLOCK_LEN);
 971                         bcopy(&(auth_data[processed]), authp, remainder);
 972                         datap = (uint8_t *)authp;
 973                         remainder = 0;
 974                 } else {
 975                         datap = (uint8_t *)(&(auth_data[processed]));
 976                         processed += AES_BLOCK_LEN;
 977                         remainder -= AES_BLOCK_LEN;
 978                 }

 980                 /* xor with previous buffer */
 981                 if (IS_P2ALIGNED2(datap, mac_buf, sizeof (uint32_t))) {
 945                 if (IS_P2ALIGNED(datap, sizeof (uint32_t)) &&
 946                     IS_P2ALIGNED(mac_buf, sizeof (uint32_t))) {
 982                         /* LINTED: pointer alignment */
 983                         *(uint32_t *)&mac_buf[0] ^= *(uint32_t *)&datap[0];
 984                         /* LINTED: pointer alignment */
 985                         *(uint32_t *)&mac_buf[4] ^= *(uint32_t *)&datap[4];
 986                         /* LINTED: pointer alignment */
 987                         *(uint32_t *)&mac_buf[8] ^= *(uint32_t *)&datap[8];
 988                         /* LINTED: pointer alignment */
 989                         *(uint32_t *)&mac_buf[12] ^= *(uint32_t *)&datap[12];
 990                 } else {
 991                         AES_XOR_BLOCK(datap, mac_buf);
 992                 }

 994                 aes_encrypt_block(ctx->ac_keysched, mac_buf, mac_buf);

 996         } while (remainder > 0);

 998 /* EXPORT DELETE END */
 999         return (0);
1000 }
______unchanged_portion_omitted_

1029 /* ARGSUSED */
1030 int
1031 aes_ccm_encrypt_final(aes_ctx_t *ctx, crypto_data_t *out)
1032 {
1033 /* EXPORT DELETE START */
```



```
1035         uint8_t *lastp, *mac_buf, *ccm_mac_p, *macp;
1036         uint32_t counter_block[4];
1037         uint32_t tmp[4];
1038         uint8_t ccm_mac[AES_BLOCK_LEN];
1039         void *iov_or_mp;
1040         offset_t offset;
1041         uint8_t *out_data_1;
1042         uint8_t *out_data_2;
1043         size_t out_data_1_len;
1044         int i;

1046         if (out->cd_length < (ctx->ac_remainder_len + ctx->ac_ccm_mac_len)) {
1047                 return (CRYPTO_ARGUMENTS_BAD);
1048         }

1050         /*
1051          * When we get here, the number of bytes of payload processed
1052          * plus whatever data remains, if any,
1053          * should be the same as the number of bytes that's being
1054          * passed in the argument during init time.
1055          */
1056         if ((ctx->ac_ccm_processed_data_len + ctx->ac_remainder_len)
1057             != (ctx->ac_ccm_data_len)) {
1058                 return (CRYPTO_DATA_LEN_RANGE);
1059         }

1061         mac_buf = (uint8_t *)ctx->ac_ccm_mac_buf;

1063         if (ctx->ac_remainder_len > 0) {

1065                 macp = (uint8_t *)tmp;
1066                 bzero(macp, AES_BLOCK_LEN);

1068                 /* copy remainder to temporary buffer */
1069                 bcopy(ctx->ac_remainder, macp, ctx->ac_remainder_len);

1071                 /* calculate the CBC MAC */
1072                 if (IS_P2ALIGNED2(macp, mac_buf, sizeof (uint32_t))) {
1037                 if (IS_P2ALIGNED(macp, sizeof (uint32_t)) &&
1038                     IS_P2ALIGNED(mac_buf, sizeof (uint32_t))) {
1073                         /* LINTED: pointer alignment */
1074                         *(uint32_t *)&mac_buf[0] ^= *(uint32_t *)&macp[0];
1075                         /* LINTED: pointer alignment */
1076                         *(uint32_t *)&mac_buf[4] ^= *(uint32_t *)&macp[4];
1077                         /* LINTED: pointer alignment */
1078                         *(uint32_t *)&mac_buf[8] ^= *(uint32_t *)&macp[8];
1079                         /* LINTED: pointer alignment */
1080                         *(uint32_t *)&mac_buf[12] ^= *(uint32_t *)&macp[12];
1081                 } else {
1082                         AES_XOR_BLOCK(macp, mac_buf);
1083                 }
1084                 aes_encrypt_block(ctx->ac_keysched, mac_buf, mac_buf);

1086                 /* calculate the counter mode */
1087                 aes_encrypt_block(ctx->ac_keysched, (uint8_t *)ctx->ac_cb,
1088                     (uint8_t *)counter_block);

1090                 lastp = (uint8_t *)counter_block;

1092                 /* copy remainder to temporary buffer */
1093                 bcopy(ctx->ac_remainder, macp, ctx->ac_remainder_len);

1095                 /* XOR with counter block */
1096                 for (i = 0; i < ctx->ac_remainder_len; i++) {
1097                         macp[i] ^= lastp[i];
```

```
1098                }
1099                ctx->ac_ccm_processed_data_len += ctx->ac_remainder_len;
1100        }

1102        /* Calculate the CCM MAC */
1103        ccm_mac_p = ccm_mac;
1104        calculate_ccm_mac(ctx, &ccm_mac_p);

1106        aes_init_ptrs(out, &iov_or_mp, &offset);
1107        aes_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
1108            &out_data_1_len, &out_data_2,
1109            ctx->ac_remainder_len + ctx->ac_ccm_mac_len);

1111        if (ctx->ac_remainder_len > 0) {

1113                /* copy temporary block to where it belongs */
1114                if (out_data_2 == NULL) {
1115                        /* everything will fit in out_data_1 */
1116                        bcopy(macp, out_data_1, ctx->ac_remainder_len);
1117                        bcopy(ccm_mac, out_data_1 + ctx->ac_remainder_len,
1118                            ctx->ac_ccm_mac_len);
1119                } else {

1121                        if (out_data_1_len < ctx->ac_remainder_len) {

1123                                size_t data_2_len_used;

1125                                bcopy(macp, out_data_1, out_data_1_len);

1127                                data_2_len_used = ctx->ac_remainder_len
1128                                    - out_data_1_len;

1130                                bcopy((uint8_t *)macp + out_data_1_len,
1131                                    out_data_2, data_2_len_used);
1132                                bcopy(ccm_mac, out_data_2 + data_2_len_used,
1133                                    ctx->ac_ccm_mac_len);
1134                        } else {
1135                                bcopy(macp, out_data_1, out_data_1_len);
1136                                if (out_data_1_len == ctx->ac_remainder_len) {
1137                                        /* mac will be in out_data_2 */
1138                                        bcopy(ccm_mac, out_data_2,
1139                                            ctx->ac_ccm_mac_len);
1140                                } else {
1141                                        size_t len_not_used
1142                                            = out_data_1_len -
1143                                            ctx->ac_remainder_len;
1144                                        /*
1145                                         * part of mac in will be in
1146                                         * out_data_1, part of the mac will be
1147                                         * in out_data_2
1148                                         */
1149                                        bcopy(ccm_mac,
1150                                            out_data_1 + ctx->ac_remainder_len,
1151                                            len_not_used);
1152                                        bcopy(ccm_mac+len_not_used, out_data_2,
1153                                            ctx->ac_ccm_mac_len - len_not_used);

1155                                }
1156                        }
1157                }
1158        } else {
1159                /* copy block to where it belongs */
1160                bcopy(ccm_mac, out_data_1, out_data_1_len);
1161                if (out_data_2 != NULL) {
1162                        bcopy(ccm_mac + out_data_1_len, out_data_2,
1163                            AES_BLOCK_LEN - out_data_1_len);
```



```
1164                }
1165        }
1166        out->cd_offset += ctx->ac_remainder_len + ctx->ac_ccm_mac_len;
1167        ctx->ac_remainder_len = 0;

1169 /* EXPORT DELETE END */

1171        return (0);
1172 }

1174 int
1175 aes_ccm_validate_args(CK_AES_CCM_PARAMS *ccm_param, boolean_t is_encrypt_init)
1176 {

1178 /* EXPORT DELETE START */
1179        size_t macSize, nonceSize;
1180        uint8_t q;
1181        uint64_t maxValue;

1183        /*
1184         * Check the byte length of the MAC.  The only valid
1150         * Check the length of the MAC.  Only valid length
1185         * lengths for the MAC are: 4, 6, 8, 10, 12, 14, 16
1186         */
1187        macSize = ccm_param->ulMACSize;
1188        if ((macSize < 4) || (macSize > 16) || ((macSize % 2) != 0)) {
1189                return (CRYPTO_MECHANISM_PARAM_INVALID);
1190        }

1192        /* Check the nonce length.  Valid values are 7, 8, 9, 10, 11, 12, 13 */
1158        /* Check the nonce value.  Valid values are 7, 8, 9, 10, 11, 12, 13 */
1193        nonceSize = ccm_param->ulNonceSize;
1194        if ((nonceSize < 7) || (nonceSize > 13)) {
1195                return (CRYPTO_MECHANISM_PARAM_INVALID);
1196        }

1198        /* q is the length of the field storing the length, in bytes */
1199        q = (uint8_t)((15 - nonceSize) & 0xFF);


1202        /*
1203         * If it is decrypt, need to make sure size of ciphertext is at least
1204         * bigger than MAC len
1205         */
1206        if ((!is_encrypt_init) && (ccm_param->ulDataSize < macSize)) {
1207                return (CRYPTO_MECHANISM_PARAM_INVALID);
1208        }

1210        /*
1211         * Check to make sure the length of the payload is within the
1212         * range of values allowed by q
1213         */
1214        if (q < 8) {
1215                maxValue = (1ULL << (q * 8)) - 1;
1180                maxValue = 1ULL << (q * 8);
1216        } else {
1217                maxValue = ULONG_MAX;
1218        }

1220        if (ccm_param->ulDataSize > maxValue) {
1221                return (CRYPTO_MECHANISM_PARAM_INVALID);
1222        }

1224 /* EXPORT DELETE END */
1225        return (0);
1226 }
```

```
/*
 * Format the first block used in CBC-MAC (B0) and the initial counter
 * block based on formatting functions and counter generation functions
 * block based on formating functions and counter generation functions
 * specified in RFC 3610 and NIST publication 800-38C, appendix A
 *
 * b0 is the first block used in CBC-MAC
 * cb0 is the first counter block
 *
 * It's assumed that the arguments b0 and cb0 are preallocated AES blocks
 *
 */
static void
aes_ccm_format_initial_blocks(uchar_t *nonce, ulong_t nonceSize,
    ulong_t authDataSize, uint8_t *b0, aes_ctx_t *aes_ctx)
{
/* EXPORT DELETE START */
        uint64_t payloadSize;
        uint8_t t, q, have_adata = 0;
        size_t limit;
        int i, j, k;
        uint64_t mask = 0;
        uint8_t *cb;
#ifdef _LITTLE_ENDIAN
        uint8_t *p8;
#endif  /* _LITTLE_ENDIAN */

        q = (uint8_t)((15 - nonceSize) & 0xFF);
        t = (uint8_t)((aes_ctx->ac_ccm_mac_len) & 0xFF);

        /* Construct the first octet of b0 */
        /* Construct the first octect of b0 */
        if (authDataSize > 0) {
                have_adata = 1;
        }
        b0[0] = (have_adata << 6) | (((t - 2)  / 2) << 3) | (q - 1);

        /* copy the nonce value into b0 */
        bcopy(nonce, &(b0[1]), nonceSize);

        /* store the length of the payload into b0 */
        bzero(&(b0[1+nonceSize]), q);

        payloadSize = aes_ctx->ac_ccm_data_len;
        limit = 8 < q ? 8 : q;

        for (i = 0, j = 0, k = 15; i < limit; i++, j += 8, k--) {
                b0[k] = (uint8_t)((payloadSize >> j) & 0xFF);
        }

        /* format the counter block */

        cb = (uint8_t *)aes_ctx->ac_cb;

        cb[0] = 0x07 & (q-1); /* first byte */

        /* copy the nonce value into the counter block */
        bcopy(nonce, &(cb[1]), nonceSize);

        bzero(&(cb[1+nonceSize]), q);

        /* Create the mask for the counter field based on the size of nonce */
        q <<= 3;
        while (q-- > 0) {
                mask |= (1ULL << q);
```



```
        }

#ifdef _LITTLE_ENDIAN
        p8 = (uint8_t *)&mask;
        mask = (((uint64_t)p8[0] << 56) |
            ((uint64_t)p8[1] << 48) |
            ((uint64_t)p8[2] << 40) |
            ((uint64_t)p8[3] << 32) |
            ((uint64_t)p8[4] << 24) |
            ((uint64_t)p8[5] << 16) |
            ((uint64_t)p8[6] << 8) |
            (uint64_t)p8[7]);
#endif
        aes_ctx->ac_counter_mask = mask;

        /*
         * During calculation, we start using counter block 1, we will
         * set it up right here.
         * We can just set the last byte to have the value 1, because
         * even with the biggest nonce of 13, the last byte of the
         * even with the bigest nonce of 13, the last byte of the
         * counter block will be used for the counter value.
         */
        cb[15] = 0x01;

/* EXPORT DELETE END */

}
______unchanged_portion_omitted_

/*
 * This will decrypt the cipher text.  However, the plaintext won't be
 * returned to the caller.  It will be returned when decrypt_final() is
 * called if the MAC matches
 */
/* ARGSUSED */
static int
aes_ccm_decrypt_contiguous_blocks(aes_ctx_t *ctx, char *data, size_t length,
    crypto_data_t *out)
{

/* EXPORT DELETE START */

        size_t remainder = length;
        size_t need;
        uint8_t *datap = (uint8_t *)data;
        uint8_t *blockp;
        uint32_t counter_block[4];
        uint8_t *cbp;
        uint64_t counter;
        size_t pt_len, total_decrypted_len, mac_len, pm_len, pd_len;
        uint32_t tmp[4];
        uint8_t *resultp;
#ifdef _LITTLE_ENDIAN
        uint8_t *p;
#endif  /* _LITTLE_ENDIAN */


        pm_len = ctx->ac_ccm_processed_mac_len;

        if (pm_len > 0) {
                uint8_t *tmp;
                /*
                 * all ciphertext has been processed, just waiting for
                 * part of the value of the mac
                 */
```

```
1427                if ((pm_len + length) > ctx->ac_ccm_mac_len) {
1428                        return (CRYPTO_DATA_LEN_RANGE);
1429                }
1430                tmp = (uint8_t *)ctx->ac_ccm_mac_input_buf;

1432                bcopy(datap, tmp + pm_len, length);

1434                ctx->ac_ccm_processed_mac_len += length;
1435                return (0);
1436        }

1438        /*
1439         * If we decrypt the given data, what total amount of data would
1440         * have been decrypted?
1441         */
1442        pd_len = ctx->ac_ccm_processed_data_len;
1443        total_decrypted_len = pd_len + length + ctx->ac_remainder_len;

1445        if (total_decrypted_len >
1446            (ctx->ac_ccm_data_len + ctx->ac_ccm_mac_len)) {
1447                return (CRYPTO_DATA_LEN_RANGE);
1448        }

1450        pt_len = ctx->ac_ccm_data_len;

1452        if (total_decrypted_len > pt_len) {
1453                /*
1454                 * part of the input will be the MAC, need to isolate that
1455                 * to be dealt with later.  The left-over data in
1456                 * ac_remainder_len from last time will not be part of the
1457                 * MAC.  Otherwise, it would have already been taken out
1458                 * when this call is made last time.
1459                 */
1460                size_t pt_part = pt_len - pd_len - ctx->ac_remainder_len;

1462                mac_len = length - pt_part;

1464                ctx->ac_ccm_processed_mac_len = mac_len;
1465                bcopy(data + pt_part, ctx->ac_ccm_mac_input_buf, mac_len);

1467                if (pt_part + ctx->ac_remainder_len < AES_BLOCK_LEN) {
1468                        /*
1469                         * since this is last of the ciphertext, will
1470                         * just decrypt with it here
1471                         */
1472                        bcopy(datap, &((uint8_t *)ctx->ac_remainder)
1473                            [ctx->ac_remainder_len], pt_part);
1474                        ctx->ac_remainder_len += pt_part;
1475                        aes_ccm_decrypt_incomplete_block(ctx);
1476                        ctx->ac_remainder_len = 0;
1477                        ctx->ac_ccm_processed_data_len += pt_part;
1478                        return (0);
1479                } else {
1480                        /* let rest of the code handle this */
1481                        length = pt_part;
1482                }
1483        } else if (length + ctx->ac_remainder_len < AES_BLOCK_LEN) {
1484                        /* accumulate bytes here and return */
1485                bcopy(datap,
1486                    (uint8_t *)ctx->ac_remainder + ctx->ac_remainder_len,
1487                    length);
1488                ctx->ac_remainder_len += length;
1489                ctx->ac_copy_to = datap;
1490                return (0);
1491        }
```



```
1493        do {
1494                /* Unprocessed data from last call. */
1495                if (ctx->ac_remainder_len > 0) {
1496                        need = AES_BLOCK_LEN - ctx->ac_remainder_len;

1498                        if (need > remainder)
1499                                return (1);

1501                        bcopy(datap, &((uint8_t *)ctx->ac_remainder)
1502                            [ctx->ac_remainder_len], need);

1504                        blockp = (uint8_t *)ctx->ac_remainder;
1505                } else {
1506                        blockp = datap;
1507                }

1509                /* don't write on the plaintext */
1510                if (IS_P2ALIGNED(blockp, sizeof (uint32_t))) {
1511                        /* LINTED: pointer alignment */
1512                        tmp[0] = *(uint32_t *)blockp;
1513                        /* LINTED: pointer alignment */
1514                        tmp[1] = *(uint32_t *)&blockp[4];
1515                        /* LINTED: pointer alignment */
1516                        tmp[2] = *(uint32_t *)&blockp[8];
1517                        /* LINTED: pointer alignment */
1518                        tmp[3] = *(uint32_t *)&blockp[12];
1519                } else {
1520                        uint8_t *tmp8 = (uint8_t *)tmp;

1522                        AES_COPY_BLOCK(blockp, tmp8);
1523                }
1524                blockp = (uint8_t *)tmp;

1526                /* Calculate the counter mode, ac_cb is the counter block */
1527                aes_encrypt_block(ctx->ac_keysched, (uint8_t *)ctx->ac_cb,
1528                    (uint8_t *)counter_block);
1529                cbp = (uint8_t *)counter_block;

1531                /*
1532                 * Increment counter.
1533                 * Counter bits are confined to the bottom 64 bits
1534                 */
1535                counter = ctx->ac_cb[1] & ctx->ac_counter_mask;
1536 #ifdef _LITTLE_ENDIAN
1537                p = (uint8_t *)&counter;
1538                counter = (((uint64_t)p[0] << 56) |
1539                    ((uint64_t)p[1] << 48) |
1540                    ((uint64_t)p[2] << 40) |
1541                    ((uint64_t)p[3] << 32) |
1542                    ((uint64_t)p[4] << 24) |
1543                    ((uint64_t)p[5] << 16) |
1544                    ((uint64_t)p[6] << 8) |
1545                    (uint64_t)p[7]);
1546 #endif
1547                counter++;
1548 #ifdef _LITTLE_ENDIAN
1549                counter = (((uint64_t)p[0] << 56) |
1550                    ((uint64_t)p[1] << 48) |
1551                    ((uint64_t)p[2] << 40) |
1552                    ((uint64_t)p[3] << 32) |
1553                    ((uint64_t)p[4] << 24) |
1554                    ((uint64_t)p[5] << 16) |
1555                    ((uint64_t)p[6] << 8) |
1556                    (uint64_t)p[7]);
1557 #endif
1558                counter &= ctx->ac_counter_mask;
```

```
1559                ctx->ac_cb[1] =
1560                    (ctx->ac_cb[1] & ~(ctx->ac_counter_mask)) | counter;

1562                /* XOR with the ciphertext */
1563                if (IS_P2ALIGNED2(blockp, cbp, sizeof (uint32_t))) {
1528                if (IS_P2ALIGNED(blockp, sizeof (uint32_t)) &&
1529                    IS_P2ALIGNED(cbp, sizeof (uint32_t))) {
1564                        /* LINTED: pointer alignment */
1565                        *(uint32_t *)&blockp[0] ^= *(uint32_t *)&cbp[0];
1566                        /* LINTED: pointer alignment */
1567                        *(uint32_t *)&blockp[4] ^= *(uint32_t *)&cbp[4];
1568                        /* LINTED: pointer alignment */
1569                        *(uint32_t *)&blockp[8] ^= *(uint32_t *)&cbp[8];
1570                        /* LINTED: pointer alignment */
1571                        *(uint32_t *)&blockp[12] ^= *(uint32_t *)&cbp[12];
1572                } else {
1573                        AES_XOR_BLOCK(cbp, blockp);
1574                }

1576                /* Copy the plaintext to the "holding buffer" */
1577                resultp = (uint8_t *)ctx->ac_ccm_pt_buf +
1578                    ctx->ac_ccm_processed_data_len;
1579                if (IS_P2ALIGNED2(blockp, resultp, sizeof (uint32_t))) {
1545                if (IS_P2ALIGNED(blockp, sizeof (uint32_t)) &&
1546                    IS_P2ALIGNED(resultp, sizeof (uint32_t))) {
1580                        /* LINTED: pointer alignment */
1581                        *(uint32_t *)&resultp[0] = *(uint32_t *)blockp;
1582                        /* LINTED: pointer alignment */
1583                        *(uint32_t *)&resultp[4] = *(uint32_t *)&blockp[4];
1584                        /* LINTED: pointer alignment */
1585                        *(uint32_t *)&resultp[8] = *(uint32_t *)&blockp[8];
1586                        /* LINTED: pointer alignment */
1587                        *(uint32_t *)&resultp[12] = *(uint32_t *)&blockp[12];
1588                } else {
1589                        AES_COPY_BLOCK(blockp, resultp);
1590                }

1592                ctx->ac_ccm_processed_data_len += AES_BLOCK_LEN;

1594                ctx->ac_lastp = blockp;

1596                /* Update pointer to next block of data to be processed. */
1597                if (ctx->ac_remainder_len != 0) {
1598                        datap += need;
1599                        ctx->ac_remainder_len = 0;
1600                } else {
1601                        datap += AES_BLOCK_LEN;
1602                }

1604                remainder = (size_t)&data[length] - (size_t)datap;

1606                /* Incomplete last block */
1607                if (remainder > 0 && remainder < AES_BLOCK_LEN) {
1608                        bcopy(datap, ctx->ac_remainder, remainder);
1609                        ctx->ac_remainder_len = remainder;
1610                        ctx->ac_copy_to = datap;
1611                        if (ctx->ac_ccm_processed_mac_len > 0) {
1612                                /*
1613                                 * not expecting anymore ciphertext, just
1614                                 * compute plaintext for the remaining input
1615                                 */
1616                                aes_ccm_decrypt_incomplete_block(ctx);
1617                                ctx->ac_ccm_processed_data_len += remainder;
1618                                ctx->ac_remainder_len = 0;
1619                        }
1620                        goto out;
```



```
1621                }
1622                ctx->ac_copy_to = NULL;

1624        } while (remainder > 0);

1626 out:
1627 /* EXPORT DELETE END */

1629        return (0);
1630 }

1632 int
1633 aes_ccm_decrypt_final(aes_ctx_t *ctx, crypto_data_t *out)
1634 {
1635 /* EXPORT DELETE START */
1636        size_t mac_remain, pt_len;
1637        uint8_t *pt, *mac_buf, *macp, *ccm_mac_p;
1638        uint8_t ccm_mac[AES_BLOCK_LEN];
1639        void *iov_or_mp;
1640        offset_t offset;
1641        uint8_t *out_data_1, *out_data_2;
1642        size_t out_data_1_len;
1643        uint32_t tmp[4];

1645        pt_len = ctx->ac_ccm_data_len;

1647        /* Make sure output buffer can fit all of the plaintext */
1648        if (out->cd_length < pt_len) {
1649                return (CRYPTO_ARGUMENTS_BAD);
1650        }

1652        pt = ctx->ac_ccm_pt_buf;
1653        mac_remain = ctx->ac_ccm_processed_data_len;
1654        mac_buf = (uint8_t *)ctx->ac_ccm_mac_buf;

1656        macp = (uint8_t *)tmp;

1658        while (mac_remain > 0) {

1660                if (mac_remain < AES_BLOCK_LEN) {
1661                        bzero(tmp, AES_BLOCK_LEN);
1662                        bcopy(pt, tmp, mac_remain);
1663                        mac_remain = 0;
1664                } else {
1665                        if (IS_P2ALIGNED2(pt, macp, sizeof (uint32_t))) {
1632                        if (IS_P2ALIGNED(pt, sizeof (uint32_t)) &&
1633                            IS_P2ALIGNED(macp, sizeof (uint32_t))) {
1666                                /* LINTED: pointer alignment */
1667                                *(uint32_t *)&macp[0] = *(uint32_t *)pt;
1668                                /* LINTED: pointer alignment */
1669                                *(uint32_t *)&macp[4] = *(uint32_t *)&pt[4];
1670                                /* LINTED: pointer alignment */
1671                                *(uint32_t *)&macp[8] = *(uint32_t *)&pt[8];
1672                                /* LINTED: pointer alignment */
1673                                *(uint32_t *)&macp[12] = *(uint32_t *)&pt[12];
1674                        } else {
1675                                AES_COPY_BLOCK(pt, macp);
1676                        }
1677                        mac_remain -= AES_BLOCK_LEN;
1678                        pt += AES_BLOCK_LEN;
1679                }

1681                /* calculate the CBC MAC */
1682                if (IS_P2ALIGNED2(macp, mac_buf, sizeof (uint32_t))) {
1650                if (IS_P2ALIGNED(macp, sizeof (uint32_t)) &&
1651                    IS_P2ALIGNED(mac_buf, sizeof (uint32_t))) {
```

```
1683                        /* LINTED: pointer alignment */
1684                        *(uint32_t *)&mac_buf[0] ^= *(uint32_t *)&macp[0];
1685                        /* LINTED: pointer alignment */
1686                        *(uint32_t *)&mac_buf[4] ^= *(uint32_t *)&macp[4];
1687                        /* LINTED: pointer alignment */
1688                        *(uint32_t *)&mac_buf[8] ^= *(uint32_t *)&macp[8];
1689                        /* LINTED: pointer alignment */
1690                        *(uint32_t *)&mac_buf[12] ^= *(uint32_t *)&macp[12];
1691                } else {
1692                        AES_XOR_BLOCK(macp, mac_buf);
1693                }
1694                aes_encrypt_block(ctx->ac_keysched, mac_buf, mac_buf);
1695        }

1697        /* Calculate the CCM MAC */
1698        ccm_mac_p = ccm_mac;
1699        calculate_ccm_mac(ctx, &ccm_mac_p);

1701        /* compare the input CCM MAC value with what we calculated */
1702        if (bcmp(ctx->ac_ccm_mac_input_buf, ccm_mac, ctx->ac_ccm_mac_len)) {
1703                /* They don't match */
1704                return (CRYPTO_DATA_LEN_RANGE);
1705        } else {
1706                aes_init_ptrs(out, &iov_or_mp, &offset);
1707                aes_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
1708                    &out_data_1_len, &out_data_2, pt_len);
1709                bcopy(ctx->ac_ccm_pt_buf, out_data_1, out_data_1_len);
1710                if (out_data_2 != NULL) {
1711                        bcopy((ctx->ac_ccm_pt_buf) + out_data_1_len,
1712                            out_data_2, pt_len - out_data_1_len);
1713                }
1714                out->cd_offset += pt_len;
1715        }

1717 /* EXPORT DELETE END */
1718        return (0);
1719 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4207 Fri Jun 13 16:44:33 2008
new/usr/src/common/crypto/aes/aes_cbc_crypt.h
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _AES_CBC_CRYPT_H
#define _AES_CBC_CRYPT_H

#pragma ident   "@(#)aes_cbc_crypt.h    1.7     08/05/21 SMI"
#pragma ident   "@(#)aes_cbc_crypt.h    1.6     07/09/11 SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/crypto/common.h>
#include "aes_impl.h"

/*
 * ac_keysched:         Pointer to key schedule.
 *
 * ac_keysched_len:     Length of the key schedule.
 *
 * ac_remainder:        This is for residual data, i.e. data that can't
 *                      be processed because there are too few bytes.
 *                      Must wait until more data arrives.
 *
 * ac_remainder_len:    Number of bytes in ac_remainder.
 *
 * ac_iv:               Scratch buffer that sometimes contains the IV.
 *
 * ac_lastblock:        Scratch buffer.
 *
 * ac_lastp:            Pointer to previous block of ciphertext.
 *
 * ac_copy_to:          Pointer to where encrypted residual data needs
 *                      to be copied.
 *
 * ac_flags:            AES_PROVIDER_OWNS_KEY_SCHEDULE
 *                      When a context is freed, it is necessary
```



```
 *                      to know whether the key schedule was allocated
 *                      by the caller, or by aes_encrypt_init() or
 *                      aes_decrypt_init().  If allocated by the latter,
 *                      then it needs to be freed.
 *
 *                      AES_ECB_MODE, AES_CBC_MODE, or AES_CTR_MODE
 *                      AES_CCM_MODE
 *
 * ac_ccm_mac_len:      Stores length of the MAC in CCM mode.
 * ac_ccm_mac_buf:      Stores the intermediate value for MAC in CCM encrypt.
 *                      In CCM decrypt, stores the input MAC value.
 * ac_ccm_data_len:     Length of the plaintext for CCM mode encrypt, or
 *                      length of the ciphertext for CCM mode decrypt.
 * ac_ccm_processed_data_len:
 *                      Length of processed plaintext in CCM mode encrypt,
 *                      or length of processed ciphertex for CCM mode decrypt.
 * ac_ccm_processed_mac_len:
 *                      Length of MAC data accumulated in CCM mode decrypt.
 *
 * ac_ccm_pt_buf:       Only used in CCM mode decrypt.  It stores the
 *                      decrypted plaintext to be returned when
 *                      MAC verification succeeds in decrypt_final.
 *                      Memory for this should be allocated in the AES module.
 *
 */
typedef struct aes_ctx {
        void *ac_keysched;
        size_t ac_keysched_len;
        uint64_t ac_iv[2];
        uint64_t ac_lastblock[2];
        uint64_t ac_remainder[2];
        size_t ac_remainder_len;
        uint8_t *ac_lastp;
        uint8_t *ac_copy_to;
        uint32_t ac_flags;
        size_t ac_ccm_mac_len;
        uint64_t ac_ccm_mac_buf[2];
        size_t ac_ccm_data_len;
        size_t ac_ccm_processed_data_len;
        size_t ac_ccm_processed_mac_len;
        uint8_t *ac_ccm_pt_buf;
        uint64_t ac_ccm_mac_input_buf[2];
} aes_ctx_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   59286 Fri Jun 13 16:44:36 2008
new/usr/src/common/crypto/aes/aes_impl.c
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "@(#)aes_impl.c 1.3     08/06/12 SMI"

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/ddi.h>
#include <sys/sysmacros.h>
#include <sys/strsun.h>
#include <netinet/in.h>
#include "aes_impl.h"
#ifndef _KERNEL
#include <strings.h>
#include <stdlib.h>
#endif  /* !_KERNEL */

#pragma ident   "@(#)aes_impl.c 1.2     05/06/08 SMI"

/*
 * This file is derived from the file  rijndael-alg-fst.c  taken from the
 * "optimized C code v3.0" on the "rijndael home page"
 * http://www.iaik.tu-graz.ac.at/research/krypto/AES/old/~rijmen/rijndael/
 * pointed by the NIST web-site http://csrc.nist.gov/archive/aes/
 * (http://www.esat.kuleuven.ac.be/~rijmen/rijndael/)
 * pointed by the NIST web-site (http://nist.gov)
 *
 * The following note is from the original file:
 */

/*
 * rijndael-alg-fst.c
 *
 * @version 3.0 (December 2000)
 *
```



```
 * Optimised ANSI C code for the Rijndael cipher (now AES)
 *
 * @author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>
 * @author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>
 * @author Paulo Barreto <paulo.barreto@terra.com.br>
 *
 * This code is hereby placed in the public domain.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
 * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/* EXPORT DELETE START */

#if defined(sun4u) || defined(__amd64)
/* External assembly functions: */
extern void aes_encrypt_impl(const uint32_t rk[], int Nr, const uint32_t pt[4],
        uint32_t ct[4]);
extern void aes_decrypt_impl(const uint32_t rk[], int Nr, const uint32_t ct[4],
        uint32_t pt[4]);
#define AES_ENCRYPT_IMPL                aes_encrypt_impl
#define AES_DECRYPT_IMPL                aes_decrypt_impl
#ifndef _RIJNDAEL_TBL_H
#define _RIJNDAEL_TBL_H

#ifdef  __amd64
extern int rijndael_key_setup_enc(uint32_t rk[], const uint32_t cipherKey[],
        int keyBits);
extern int rijndael_key_setup_dec(uint32_t rk[], const uint32_t cipherKey[],
        int keyBits);
#endif

#else
#define AES_ENCRYPT_IMPL                rijndael_encrypt
#define AES_DECRYPT_IMPL                rijndael_decrypt
#define rijndael_key_setup_enc_raw      rijndael_key_setup_enc
#endif  /* sun4u || __amd64 */

#if defined(_LITTLE_ENDIAN) && !defined(__amd64)
#define AES_BYTE_SWAP
#endif

#ifndef __amd64
/*
 *  Constant tables
 */

/*
 * Te0[x] = S [x].[02, 01, 01, 03];
 * Te1[x] = S [x].[03, 02, 01, 01];
 * Te2[x] = S [x].[01, 03, 02, 01];
 * Te3[x] = S [x].[01, 01, 03, 02];
 * Te4[x] = S [x].[01, 01, 01, 01];
 *
 * Td0[x] = Si[x].[0e, 09, 0d, 0b];
 * Td1[x] = Si[x].[0b, 0e, 09, 0d];
 * Td2[x] = Si[x].[0d, 0b, 0e, 09];
```

```
 119  * Td3[x] = Si[x].[09, 0d, 0b, 0e];
 120  * Td4[x] = Si[x].[01, 01, 01, 01];
 121  */

 123 /* Encrypt Sbox constants (for the substitute bytes operation) */

 125 static const uint32_t Te0[256] =
 126 {
 127         0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,
 128         0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,
 129         0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,
 130         0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,
 131         0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,
 132         0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfbf0f00bU,
 133         0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afafeaU,
 134         0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,
 135         0x75b7b7c2U, 0xe1fdfd1cU, 0x3d9393aeU, 0x4c26266aU,
 136         0x6c36365aU, 0x7e3f3f41U, 0xf5f7f702U, 0x83cccc4fU,
 137         0x6834345cU, 0x51a5a5f4U, 0xd1e5e534U, 0xf9f1f108U,
 138         0xe2717193U, 0xabd8d873U, 0x62313153U, 0x2a15153fU,
 139         0x0804040cU, 0x95c7c752U, 0x46232365U, 0x9dc3c35eU,
 140         0x30181828U, 0x379696a1U, 0x0a05050fU, 0x2f9a9ab5U,
 141         0x0e070709U, 0x24121236U, 0x1b80809bU, 0xdfe2e23dU,
 142         0xcdebeb26U, 0x4e272769U, 0x7fb2b2cdU, 0xea75759fU,
 143         0x1209091bU, 0x1d83839eU, 0x582c2c74U, 0x341a1a2eU,
 144         0x361b1b2dU, 0xdc6e6eb2U, 0xb45a5aeeU, 0x5ba0a0fbU,
 145         0xa45252f6U, 0x763b3b4dU, 0xb7d6d661U, 0x7db3b3ceU,
 146         0x5229297bU, 0xdde3e33eU, 0x5e2f2f71U, 0x13848497U,
 147         0xa65353f5U, 0xb9d1d168U, 0x00000000U, 0xc1eded2cU,
 148         0x40202060U, 0xe3fcfc1fU, 0x79b1b1c8U, 0xb65b5bedU,
 149         0xd46a6abeU, 0x8dcbcb46U, 0x67bebed9U, 0x7239394bU,
 150         0x944a4adeU, 0x984c4cd4U, 0xb05858e8U, 0x85cfcf4aU,
 151         0xbbd0d06bU, 0xc5efef2aU, 0x4faaaae5U, 0xedfbfb16U,
 152         0x864343c5U, 0x9a4d4dd7U, 0x66333355U, 0x11858594U,
 153         0x8a4545cfU, 0xe9f9f910U, 0x04020206U, 0xfe7f7f81U,
 154         0xa05050f0U, 0x783c3c44U, 0x259f9fbaU, 0x4ba8a8e3U,
 155         0xa25151f3U, 0x5da3a3feU, 0x804040c0U, 0x058f8f8aU,
 156         0x3f9292adU, 0x219d9dbcU, 0x70383848U, 0xf1f5f504U,
 157         0x63bcbcdfU, 0x77b6b6c1U, 0xafdada75U, 0x42212163U,
 158         0x20101030U, 0xe5ffff1aU, 0xfdf3f30eU, 0xbfd2d26dU,
 159         0x81cdcd4cU, 0x180c0c14U, 0x26131335U, 0xc3ecec2fU,
 160         0xbe5f5fe1U, 0x359797a2U, 0x884444ccU, 0x2e171739U,
 161         0x93c4c457U, 0x55a7a7f2U, 0xfc7e7e82U, 0x7a3d3d47U,
 162         0xc86464acU, 0xba5d5de7U, 0x3219192bU, 0xe6737395U,
 163         0xc06060a0U, 0x19818198U, 0x9e4f4fd1U, 0xa3dcdc7fU,
 164         0x44222266U, 0x542a2a7eU, 0x3b9090abU, 0x0b888883U,
 165         0x8c4646caU, 0xc7eeee29U, 0x6bb8b8d3U, 0x2814143cU,
 166         0xa7dede79U, 0xbc5e5ee2U, 0x160b0b1dU, 0xaddbdb76U,
 167         0xdbe0e03bU, 0x64323256U, 0x743a3a4eU, 0x140a0a1eU,
 168         0x924949dbU, 0x0c06060aU, 0x4824246cU, 0xb85c5ce4U,
 169         0x9fc2c25dU, 0xbdd3d36eU, 0x43acacefU, 0xc46262a6U,
 170         0x399191a8U, 0x319595a4U, 0xd3e4e437U, 0xf279798bU,
 171         0xd5e7e732U, 0x8bc8c843U, 0x6e373759U, 0xda6d6db7U,
 172         0x018d8d8cU, 0xb1d5d564U, 0x9c4e4ed2U, 0x49a9a9e0U,
 173         0xd86c6cb4U, 0xac5656faU, 0xf3f4f407U, 0xcfeaea25U,
 174         0xca6565afU, 0xf47a7a8eU, 0x47aeaee9U, 0x10080818U,
 175         0x6fbabad5U, 0xf0787888U, 0x4a25256fU, 0x5c2e2e72U,
 176         0x381c1c24U, 0x57a6a6f1U, 0x73b4b4c7U, 0x97c6c651U,
 177         0xcbe8e823U, 0xa1dddd7cU, 0xe874749cU, 0x3e1f1f21U,
 178         0x964b4bddU, 0x61bdbddcU, 0x0d8b8b86U, 0x0f8a8a85U,
 179         0xe0707090U, 0x7c3e3e42U, 0x71b5b5c4U, 0xcc6666aaU,
 180         0x904848d8U, 0x06030305U, 0xf7f6f601U, 0x1c0e0e12U,
 181         0xc26161a3U, 0x6a35355fU, 0xae5757f9U, 0x69b9b9d0U,
 182         0x17868691U, 0x99c1c158U, 0x3a1d1d27U, 0x279e9eb9U,
 183         0xd9e1e138U, 0xebf8f813U, 0x2b9898b3U, 0x22111133U,
 184         0xd26969bbU, 0xa9d9d970U, 0x078e8e89U, 0x339494a7U,
```



```
 185         0x2d9b9bb6U, 0x3c1e1e22U, 0x15878792U, 0xc9e9e920U,
 186         0x87cece49U, 0xaa5555ffU, 0x50282878U, 0xa5dfdf7aU,
 187         0x038c8c8fU, 0x59a1a1f8U, 0x09898980U, 0x1a0d0d17U,
 188         0x65bfbfdaU, 0xd7e6e631U, 0x844242c6U, 0xd06868b8U,
 189         0x824141c3U, 0x299999b0U, 0x5a2d2d77U, 0x1e0f0f11U,
 190         0x7bb0b0cbU, 0xa85454fcU, 0x6dbbbbd6U, 0x2c16163aU
 191 };
_____*unchanged_portion_omitted_*

 468 /* Decrypt Sbox constants (for the substitute bytes operation) */

 470 static const uint32_t Td0[256] =
 471 {
 472         0x51f4a750U, 0x7e416553U, 0x1a17a4c3U, 0x3a275e96U,
 473         0x3bab6bcbU, 0x1f9d45f1U, 0xacfa58abU, 0x4be30393U,
 474         0x2030fa55U, 0xad766df6U, 0x88cc7691U, 0xf5024c25U,
 475         0x4fe5d7fcU, 0xc52acbd7U, 0x26354480U, 0xb562a38fU,
 476         0xdeb15a49U, 0x25ba1b67U, 0x45ea0e98U, 0x5dfec0e1U,
 477         0xc32f7502U, 0x814cf012U, 0x8d4697a3U, 0x6bd3f9c6U,
 478         0x038f5fe7U, 0x15929c95U, 0xbf6d7aebU, 0x955259daU,
 479         0xd4be832dU, 0x587421d3U, 0x49e06929U, 0x8ec9c844U,
 480         0x75c2896aU, 0xf48e7978U, 0x99583e6bU, 0x27b971ddU,
 481         0xbee14fb6U, 0xf088ad17U, 0xc920ac66U, 0x7dce3ab4U,
 482         0x63df4a18U, 0xe51a3182U, 0x97513360U, 0x62537f45U,
 483         0xb16477e0U, 0xbb6bae84U, 0xfe81a01cU, 0xf9082b94U,
 484         0x70486858U, 0x8f45fd19U, 0x94de6c87U, 0x527bf8b7U,
 485         0xab73d323U, 0x724b02e2U, 0xe31f8f57U, 0x6655ab2aU,
 486         0xb2eb2807U, 0x2fb5c203U, 0x86c57b9aU, 0xd33708a5U,
 487         0x302887f2U, 0x23bfa5b2U, 0x02036abaU, 0xed16825cU,
 488         0x8acf1c2bU, 0xa779b492U, 0xf307f2f0U, 0x4e69e2a1U,
 489         0x65daf4cdU, 0x0605bed5U, 0xd134621fU, 0xc4a6fe8aU,
 490         0x342e539dU, 0xa2f355a0U, 0x058ae132U, 0xa4f6eb75U,
 491         0x0b83ec39U, 0x4060efaaU, 0x5e719f06U, 0xbd6e1051U,
 492         0x3e218af9U, 0x96dd063dU, 0xdd3e05aeU, 0x4de6bd46U,
 493         0x91548db5U, 0x71c45d05U, 0x0406d46fU, 0x605015ffU,
 494         0x1998fb24U, 0xd6bde997U, 0x894043ccU, 0x67d99e77U,
 495         0xb0e842bdU, 0x07898b88U, 0xe7195b38U, 0x79c8eedbU,
 496         0xa17c0a47U, 0x7c420fe9U, 0xf8841ec9U, 0x00000000U,
 497         0x09808683U, 0x322bed48U, 0x1e1170acU, 0x6c5a724eU,
 498         0xfd0efffbU, 0x0f853856U, 0x3daed51eU, 0x362d3927U,
 499         0x0a0fd964U, 0x685ca621U, 0x9b5b54d1U, 0x24362e3aU,
 500         0x0c0a67b1U, 0x9357e70fU, 0xb4ee96d2U, 0x1b9b919eU,
 501         0x80c0c54fU, 0x61dc20a2U, 0x5a774b69U, 0x1c121a16U,
 502         0xe293ba0aU, 0xc0a02ae5U, 0x3c22e043U, 0x121b171dU,
 503         0x0e090d0bU, 0xf28bc7adU, 0x2db6a8b9U, 0x141ea9c8U,
 504         0x57f11985U, 0xaf75074cU, 0xee99ddbbU, 0xa37f60fdU,
 505         0xf701269fU, 0x5c72f5bcU, 0x44663bc5U, 0x5bfb7e34U,
 506         0x8b432976U, 0xcb23c6dcU, 0xb6edfc68U, 0xb8e4f163U,
 507         0xd731dccaU, 0x42638510U, 0x13972240U, 0x84c61120U,
 508         0x854a247dU, 0xd2bb3df8U, 0xaef93211U, 0xc729a16dU,
 509         0x1d9e2f4bU, 0xdcb230f3U, 0x0d8652ecU, 0x77c1e3d0U,
 510         0x2bb3166cU, 0xa970b999U, 0x119448faU, 0x47e96422U,
 511         0xa8fc8cc4U, 0xa0f03f1aU, 0x567d2cd8U, 0x223390efU,
 512         0x87494ec7U, 0xd938d1c1U, 0x8ccaa2feU, 0x98d40b36U,
 513         0xa6f581cfU, 0xa57ade28U, 0xdab78e26U, 0x3fadbfa4U,
 514         0x2c3a9de4U, 0x5078920dU, 0x6a5fcc9bU, 0x547e4662U,
 515         0xf68d13c2U, 0x90d8b8e8U, 0x2e39f75eU, 0x82c3aff5U,
 516         0x9f5d80beU, 0x69d0937cU, 0x6fd52da9U, 0xcf2512b3U,
 517         0xc8ac993bU, 0x10187da7U, 0xe89c636eU, 0xdb3bbb7bU,
 518         0xcd267809U, 0x6e5918f4U, 0xec9ab701U, 0x834f9aa8U,
 519         0xe6956e65U, 0xaaffe67eU, 0x21bccf08U, 0xef15e8e6U,
 520         0xbae79bd9U, 0x4a6f36ceU, 0xea9f09d4U, 0x29b07cd6U,
 521         0x31a4b2afU, 0x2a3f2331U, 0xc6a59430U, 0x35a266c0U,
 522         0x744ebc37U, 0xfc82caa6U, 0xe090d0b0U, 0x33a7d815U,
 523         0xf104984aU, 0x41ecdaf7U, 0x7fcd500eU, 0x1791f62fU,
 524         0x764dd68dU, 0x43efb04dU, 0xccaa4d54U, 0xe49604dfU,
```

```
525         0x9ed1b5e3U, 0x4c6a881bU, 0xc12c1fb8U, 0x4665517fU,
526         0x9d5eea04U, 0x018c355dU, 0xfa877473U, 0xfb0b412eU,
527         0xb3671d5aU, 0x92dbd252U, 0xe9105633U, 0x6dd64713U,
528         0x9ad7618cU, 0x37a10c7aU, 0x59f8148eU, 0xeb133c89U,
529         0xcea927eeU, 0xb761c935U, 0xe11ce5edU, 0x7a47b13cU,
530         0x9cd2df59U, 0x55f2733fU, 0x1814ce79U, 0x73c737bfU,
531         0x53f7cdeaU, 0x5ffdaa5bU, 0xdf3d6f14U, 0x7844db86U,
532         0xcaaff381U, 0xb968c43eU, 0x3824342cU, 0xc2a3405fU,
533         0x161dc372U, 0xbce2250cU, 0x283c498bU, 0xff0d9541U,
534         0x39a80171U, 0x080cb3deU, 0xd8b4e49cU, 0x6456c190U,
535         0x7bcb8461U, 0xd532b670U, 0x486c5c74U, 0xd0b85742U
536 };
_____unchanged_portion_omitted_

811 /* Rcon is Round Constant; used for encryption key expansion */
812 static const uint32_t rcon[RC_LENGTH] =
784 static const uint32_t rcon[] =
813 {
814         /* for 128-bit blocks, Rijndael never uses more than 10 rcon values */
815         0x01000000, 0x02000000, 0x04000000, 0x08000000,
816         0x10000000, 0x20000000, 0x40000000, 0x80000000,
817         0x1B000000, 0x36000000
818 };
791 #endif


821 /*
822  * Expand the cipher key into the encryption key schedule.
823  *
824  * Return the number of rounds for the given cipher key size.
795  *  return  the number of rounds for the given cipher key size.
825  * The size of the key schedule depends on the number of rounds
826  * (which can be computed from the size of the key), i.e. 4*(Nr + 1).
827  *
828  * Parameters:
829  * rk           AES key schedule 32-bit array to be initialized
830  * cipherKey    User key
831  * keyBits      AES key size (128, 192, or 256 bits)
832  */
833 static int
834 rijndael_key_setup_enc_raw(uint32_t rk[], const uint32_t cipherKey[],
835     int keyBits)
836 {
837         int             i = 0;
838         uint32_t        temp;

840         rk[0] = cipherKey[0];
841         rk[1] = cipherKey[1];
842         rk[2] = cipherKey[2];
843         rk[3] = cipherKey[3];

845         if (keyBits == 128) {
846                 for (;;) {
847                         temp  = rk[3];
848                         rk[4] = rk[0] ^
849                             (Te4[(temp >> 16) & 0xff] & 0xff000000) ^
850                             (Te4[(temp >> 8) & 0xff] & 0x00ff0000) ^
851                             (Te4[temp & 0xff] & 0x0000ff00) ^
852                             (Te4[temp >> 24] & 0x000000ff) ^
853                             rcon[i];
854                         rk[5] = rk[1] ^ rk[4];
855                         rk[6] = rk[2] ^ rk[5];
856                         rk[7] = rk[3] ^ rk[6];

858                         if (++i == 10) {
859                                 return (10);
```



```
860                         }
861                         rk += 4;
862                 }
863         }

865         rk[4] = cipherKey[4];
866         rk[5] = cipherKey[5];

868         if (keyBits == 192) {
869                 for (;;) {
870                         temp = rk[5];
871                         rk[6] = rk[0] ^
872                             (Te4[(temp >> 16) & 0xff] & 0xff000000) ^
873                             (Te4[(temp >> 8) & 0xff] & 0x00ff0000) ^
874                             (Te4[temp & 0xff] & 0x0000ff00) ^
875                             (Te4[temp >> 24] & 0x000000ff) ^
876                             rcon[i];
877                         rk[7] = rk[1] ^ rk[6];
878                         rk[8] = rk[2] ^ rk[7];
879                         rk[9] = rk[3] ^ rk[8];

881                         if (++i == 8) {
882                                 return (12);
883                         }

885                         rk[10] = rk[4] ^ rk[9];
886                         rk[11] = rk[5] ^ rk[10];
887                         rk += 6;
888                 }
889         }

891         rk[6] = cipherKey[6];
892         rk[7] = cipherKey[7];

894         if (keyBits == 256) {
895                 for (;;) {
896                         temp = rk[7];
897                         rk[8] = rk[0] ^
898                             (Te4[(temp >> 16) & 0xff] & 0xff000000) ^
899                             (Te4[(temp >> 8) & 0xff] & 0x00ff0000) ^
900                             (Te4[temp & 0xff] & 0x0000ff00) ^
901                             (Te4[temp >> 24] & 0x000000ff) ^
902                             rcon[i];
903                         rk[9] = rk[1] ^ rk[8];
904                         rk[10] = rk[2] ^ rk[9];
905                         rk[11] = rk[3] ^ rk[10];

907                         if (++i == 7) {
908                                 return (14);
909                         }
910                         temp = rk[11];
911                         rk[12] = rk[4] ^
912                             (Te4[temp >> 24] & 0xff000000) ^
913                             (Te4[(temp >> 16) & 0xff] & 0x00ff0000) ^
914                             (Te4[(temp >> 8) & 0xff] & 0x0000ff00) ^
915                             (Te4[temp & 0xff] & 0x000000ff);
916                         rk[13] = rk[5] ^ rk[12];
917                         rk[14] = rk[6] ^ rk[13];
918                         rk[15] = rk[7] ^ rk[14];

920                         rk += 8;
921                 }
922         }

924         return (0);
925 }
```

```
926 #endif  /* !__amd64 */


929 #ifdef  sun4u

931 /*
932  * Expand the cipher key into the encryption key schedule.
897  *  Expand the cipher key into the enryption key schedule as used
933  * by the sun4u optimized assembly implementation.
934  *
935  * Return the number of rounds for the given cipher key size.
899  *  return  the number of rounds for the given cipher key size.
936  * The size of the key schedule depends on the number of rounds
937  * (which can be computed from the size of the key), i.e. 4*(Nr + 1).
938  *
939  * Parameters:
940  * rk           AES key schedule 64-bit array to be initialized
941  * cipherKey    User key
942  * keyBits      AES key size (128, 192, or 256 bits)
943  */
944 static int
945 rijndael_key_setup_enc(uint64_t rk[], const uint32_t cipherKey[], int keyBits)
946 {
947         uint32_t        rk1[4 * (MAX_AES_NR + 1)];
948         uint64_t        *rk64 = (uint64_t *)rk;
949         uint32_t        *rkt;
950         uint64_t        t;
951         int             i, Nr;

953         Nr = rijndael_key_setup_enc_raw(rk1, cipherKey, keyBits);

955         for (i = 0; i < 4 * Nr; i++) {
956                 t = (uint64_t)(rk1[i]);
957                 rk64[i] = ((t & 0xff000000) << 11) |
958                     ((t & 0xff0000) << 8) |
959                     ((t & 0xffff) << 3);
960         }

962         rkt = (uint32_t *)(&(rk64[4 * Nr]));

964         for (i = 0; i < 4; i++) {
965                 rkt[i] = rk1[4 * Nr+i];
966         }

968         return (Nr);
969 }


972 /*
973  * Expand the cipher key into the decryption key schedule as used
974  * by the sun4u optimized assembly implementation.
975  *
976  * Return the number of rounds for the given cipher key size.
934  *  return  the number of rounds for the given cipher key size.
977  * The size of the key schedule depends on the number of rounds
978  * (which can be computed from the size of the key), i.e. 4*(Nr + 1).
979  *
980  * Parameters:
981  * rk           AES key schedule 32-bit array to be initialized
982  * cipherKey    User key
983  * keyBits      AES key size (128, 192, or 256 bits)
984  */
985 static int
986 rijndael_key_setup_dec_raw(uint32_t rk[], const uint32_t cipherKey[],
987     int keyBits)
988 {
```



```
 989         int             Nr, i;
 990         uint32_t        temp;

 992         /* expand the cipher key: */
 993         Nr = rijndael_key_setup_enc_raw(rk, cipherKey, keyBits);

 995         /* invert the order of the round keys: */

 997         for (i = 0; i < 2 * Nr + 2; i++) {
 998                 temp = rk[i];
 999                 rk[i] = rk[4 * Nr - i + 3];
1000                 rk[4 * Nr - i + 3] = temp;
1001         }

1003         /*
1004          * apply the inverse MixColumn transform to all
1005          * round keys but the first and the last:
1006          */
1007         for (i = 1; i < Nr; i++) {
1008                 rk += 4;
1009                 rk[0] = Td0[Te4[rk[0] >> 24] & 0xff] ^
1010                     Td1[Te4[(rk[0] >> 16) & 0xff] & 0xff] ^
1011                     Td2[Te4[(rk[0] >>  8) & 0xff] & 0xff] ^
1012                     Td3[Te4[rk[0] & 0xff] & 0xff];
1013                 rk[1] = Td0[Te4[rk[1] >> 24] & 0xff] ^
1014                     Td1[Te4[(rk[1] >> 16) & 0xff] & 0xff] ^
1015                     Td2[Te4[(rk[1] >> 8) & 0xff] & 0xff] ^
1016                     Td3[Te4[rk[1] & 0xff] & 0xff];
1017                 rk[2] = Td0[Te4[rk[2] >> 24] & 0xff] ^
1018                     Td1[Te4[(rk[2] >> 16) & 0xff] & 0xff] ^
1019                     Td2[Te4[(rk[2] >> 8) & 0xff] & 0xff] ^
1020                     Td3[Te4[rk[2] & 0xff] & 0xff];
1021                 rk[3] = Td0[Te4[rk[3] >> 24] & 0xff] ^
1022                     Td1[Te4[(rk[3] >> 16) & 0xff] & 0xff] ^
1023                     Td2[Te4[(rk[3] >> 8) & 0xff] & 0xff] ^
1024                     Td3[Te4[rk[3] & 0xff] & 0xff];
1025         }

1027         return (Nr);
1028 }


1031 /*
1032  * The size of the key schedule depends on the number of rounds
1033  * (which can be computed from the size of the key), i.e. 4*(Nr + 1).
1034  *
1035  * Parameters:
1036  * rk           AES key schedule 64-bit array to be initialized
1037  * cipherKey    User key
1038  * keyBits      AES key size (128, 192, or 256 bits)
1039  */
1040 static int
 988 int
1041 rijndael_key_setup_dec(uint64_t rk[], const uint32_t cipherKey[], int keyBits)
1042 {
1043         uint32_t        rk1[4 * (MAX_AES_NR + 1)];
1044         uint64_t        *rk64 = (uint64_t *)rk;
1045         uint32_t        *rkt;
1046         uint64_t        t;
1047         int             i, Nr;

1049         Nr = rijndael_key_setup_dec_raw(rk1, cipherKey, keyBits);
1050         for (i = 0; i < 4 * Nr; i++) {
1051                 t = (uint64_t)(rk1[i]);
1052                 rk64[i] = ((t & 0xff000000) << 11) |
1053                     ((t & 0xff0000) << 8) |
```

```
                    ((t & 0xffff) << 3);
        }

        rkt = (uint32_t *)(&(rk64[4 * Nr]));

        for (i = 0; i < 4; i++) {
                rkt[i] = rk1[4 * Nr + i];
        }

        return (Nr);
}


/*
 * Expand the 64-bit AES cipher key array into the encryption and decryption
 * key schedules.
 *
 * Parameters:
 * key          AES key schedule to be initialized
 * keyarr32     User key
 * keyBits      AES key size (128, 192, or 256 bits)
 */
static void
aes_setupkeys(aes_key_t *key, const uint32_t *keyarr32, int keybits)
aes_setupkeys(aes_key_t *key, uint32_t *keyarr32, int keybits)
{
        key->nr = rijndael_key_setup_enc(&(key->encr_ks.ks64[0]), keyarr32,
            keybits);
        key->nr = rijndael_key_setup_dec(&(key->decr_ks.ks64[0]), keyarr32,
            keybits);
        key->type = AES_64BIT_KS;
}

#else /* !sun4u */

#elif !defined(__amd64)
/*
 *  Expand the cipher key into the encryption key schedule.
 *  return  the number of rounds for the given cipher key size.
 *  The size of the key schedule depends on the number of rounds
 *  (which can be computed from the size of the key), i.e. 4*(Nr + 1).
 */
static int
rijndael_key_setup_enc(uint32_t rk[], const uint32_t cipherKey[], int keyBits)
{
        return (rijndael_key_setup_enc_raw(rk, cipherKey, keyBits));
}


/*
 *  Expand the cipher key into the decryption key schedule.
 *  Return the number of rounds for the given cipher key size.
 *  @return  the number of rounds for the given cipher key size.
 *  The size of the key schedule depends on the number of rounds
 *  (which can be computed from the size of the key), i.e. 4*(Nr + 1).
 *
 * Parameters:
 * rk           AES key schedule 32-bit array to be initialized
 * cipherKey    User key
 * keyBits      AES key size (128, 192, or 256 bits)
 */
static int
rijndael_key_setup_dec(uint32_t rk[], const uint32_t cipherKey[], int keyBits)
{
        int      Nr, i, j;
        uint32_t temp;
```



```
        /* expand the cipher key: */
        Nr = rijndael_key_setup_enc_raw(rk, cipherKey, keyBits);

        /* invert the order of the round keys: */
        for (i = 0, j = 4 * Nr; i < j; i += 4, j -= 4) {
                temp = rk[i];
                rk[i] = rk[j];
                rk[j] = temp;
                temp = rk[i + 1];
                rk[i + 1] = rk[j + 1];
                rk[j + 1] = temp;
                temp = rk[i + 2];
                rk[i + 2] = rk[j + 2];
                rk[j + 2] = temp;
                temp = rk[i + 3];
                rk[i + 3] = rk[j + 3];
                rk[j + 3] = temp;
        }

        /*
         * apply the inverse MixColumn transform to all
         * round keys but the first and the last:
         */
        for (i = 1; i < Nr; i++) {
                rk += 4;
                rk[0] = Td0[Te4[rk[0] >> 24] & 0xff] ^
                    Td1[Te4[(rk[0] >> 16) & 0xff] & 0xff] ^
                    Td2[Te4[(rk[0] >> 8) & 0xff] & 0xff] ^
                    Td3[Te4[rk[0] & 0xff] & 0xff];
                rk[1] = Td0[Te4[rk[1] >> 24] & 0xff] ^
                    Td1[Te4[(rk[1] >> 16) & 0xff] & 0xff] ^
                    Td2[Te4[(rk[1] >> 8) & 0xff] & 0xff] ^
                    Td3[Te4[rk[1] & 0xff] & 0xff];
                rk[2] = Td0[Te4[rk[2] >> 24] & 0xff] ^
                    Td1[Te4[(rk[2] >> 16) & 0xff] & 0xff] ^
                    Td2[Te4[(rk[2] >> 8) & 0xff] & 0xff] ^
                    Td3[Te4[rk[2] & 0xff] & 0xff];
                rk[3] = Td0[Te4[rk[3] >> 24] & 0xff] ^
                    Td1[Te4[(rk[3] >> 16) & 0xff] & 0xff] ^
                    Td2[Te4[(rk[3] >> 8) & 0xff] & 0xff] ^
                    Td3[Te4[rk[3] & 0xff] & 0xff];
        }

        return (Nr);
}

static void
aes_setupkeys(aes_key_t *key, uint32_t *keyarr32, int keybits)
{
        key->nr = rijndael_key_setup_enc(&(key->encr_ks.ks32[0]), keyarr32,
            keybits);
        key->nr = rijndael_key_setup_dec(&(key->decr_ks.ks32[0]), keyarr32,
            keybits);
        key->type = AES_32BIT_KS;
}

/*
 * Encrypt one block of data. The block is assumed to be an array
 * of four uint32_t values, so copy for alignment (and byte-order
 * reversal for little endian systems might be necessary on the
 * input and output byte streams.
 * The size of the key schedule depends on the number of rounds
 * (which can be computed from the size of the key), i.e. 4*(Nr + 1).
 *
```

```
 * Parameters:
 * rk   Key schedule, of aes_ks_t (60 32-bit integers)
 * Nr   Number of rounds
 * pt   Input block (plain text)
 * ct   Output block (crypto text).  Can overlap with pt
 */
static void
void
rijndael_encrypt(const uint32_t rk[], int Nr, const uint32_t pt[4],
    uint32_t ct[4])
{
        uint32_t        s0, s1, s2, s3, t0, t1, t2, t3;
        int             r;

        /*
         * map byte array block to cipher state
         * and add initial round key:
         */

        s0 = pt[0] ^ rk[0];
        s1 = pt[1] ^ rk[1];
        s2 = pt[2] ^ rk[2];
        s3 = pt[3] ^ rk[3];

        /*
         * Nr - 1 full rounds:
         */

        r = Nr >> 1;

        for (;;) {
                t0 = Te0[s0 >> 24] ^
                    Te1[(s1 >> 16) & 0xff] ^
                    Te2[(s2 >>  8) & 0xff] ^
                    Te3[s3 & 0xff] ^
                    rk[4];

                t1 = Te0[s1 >> 24] ^
                    Te1[(s2 >> 16) & 0xff] ^
                    Te2[(s3 >>  8) & 0xff] ^
                    Te3[s0 & 0xff] ^
                    rk[5];

                t2 = Te0[s2 >> 24] ^
                    Te1[(s3 >> 16) & 0xff] ^
                    Te2[(s0 >>  8) & 0xff] ^
                    Te3[s1 & 0xff] ^
                    rk[6];

                t3 = Te0[s3 >> 24] ^
                    Te1[(s0 >> 16) & 0xff] ^
                    Te2[(s1 >>  8) & 0xff] ^
                    Te3[s2 & 0xff] ^
                    rk[7];

                rk += 8;

                if (--r == 0) {
                        break;
                }

                s0 = Te0[t0 >> 24] ^
                    Te1[(t1 >> 16) & 0xff] ^
                    Te2[(t2 >>  8) & 0xff] ^
                    Te3[t3 & 0xff] ^
                    rk[0];
```



```
                s1 = Te0[t1 >> 24] ^
                    Te1[(t2 >> 16) & 0xff] ^
                    Te2[(t3 >>  8) & 0xff] ^
                    Te3[t0 & 0xff] ^
                    rk[1];

                s2 = Te0[t2 >> 24] ^
                    Te1[(t3 >> 16) & 0xff] ^
                    Te2[(t0 >>  8) & 0xff] ^
                    Te3[t1 & 0xff] ^
                    rk[2];

                s3 = Te0[t3 >> 24] ^
                    Te1[(t0 >> 16) & 0xff] ^
                    Te2[(t1 >>  8) & 0xff] ^
                    Te3[t2 & 0xff] ^
                    rk[3];
        }

        /*
         * apply last round and
         * map cipher state to byte array block:
         */

        s0 = (Te4[(t0 >> 24)] & 0xff000000) ^
            (Te4[(t1 >> 16) & 0xff] & 0x00ff0000) ^
            (Te4[(t2 >>  8) & 0xff] & 0x0000ff00) ^
            (Te4[t3 & 0xff] & 0x000000ff) ^
            rk[0];
        ct[0] = s0;

        s1 = (Te4[(t1 >> 24)] & 0xff000000) ^
            (Te4[(t2 >> 16) & 0xff] & 0x00ff0000) ^
            (Te4[(t3 >>  8) & 0xff] & 0x0000ff00) ^
            (Te4[t0 & 0xff] & 0x000000ff) ^
            rk[1];
        ct[1] = s1;

        s2 = (Te4[(t2 >> 24)] & 0xff000000) ^
            (Te4[(t3 >> 16) & 0xff] & 0x00ff0000) ^
            (Te4[(t0 >>  8) & 0xff] & 0x0000ff00) ^
            (Te4[t1 & 0xff] & 0x000000ff) ^
            rk[2];
        ct[2] = s2;

        s3 = (Te4[(t3 >> 24)] & 0xff000000) ^
            (Te4[(t0 >> 16) & 0xff] & 0x00ff0000) ^
            (Te4[(t1 >>  8) & 0xff] & 0x0000ff00) ^
            (Te4[t2 & 0xff] & 0x000000ff) ^
            rk[3];
        ct[3] = s3;
}


/*
 * Decrypt one block of data. The block is assumed to be an array
 * of four uint32_t values, so copy for alignment (and byte-order
 * reversal for little endian systems might be necessary on the
 * input and output byte streams.
 * The size of the key schedule depends on the number of rounds
 * (which can be computed from the size of the key), i.e. 4*(Nr + 1).
 *
 * Parameters:
 * rk   Key schedule, of aes_ks_t (60 32-bit integers)
 * Nr   Number of rounds
```

```
 * ct   Input block (crypto text)
 * pt   Output block (plain text). Can overlap with pt
 */
static void
void
rijndael_decrypt(const uint32_t rk[], int Nr, const uint32_t ct[4],
    uint32_t pt[4])
{
        uint32_t s0, s1, s2, s3, t0, t1, t2, t3;
        int      r;

        /*
         * map byte array block to cipher state
         * and add initial round key:
         */
        s0 = ct[0] ^ rk[0];
        s1 = ct[1] ^ rk[1];
        s2 = ct[2] ^ rk[2];
        s3 = ct[3] ^ rk[3];

        /*
         * Nr - 1 full rounds:
         */

        r = Nr >> 1;

        for (;;) {
                t0 = Td0[s0 >> 24] ^
                    Td1[(s3 >> 16) & 0xff] ^
                    Td2[(s2 >> 8) & 0xff] ^
                    Td3[s1 & 0xff] ^
                    rk[4];

                t1 = Td0[s1 >> 24] ^
                    Td1[(s0 >> 16) & 0xff] ^
                    Td2[(s3 >>  8) & 0xff] ^
                    Td3[s2 & 0xff] ^
                    rk[5];

                t2 = Td0[s2 >> 24] ^
                    Td1[(s1 >> 16) & 0xff] ^
                    Td2[(s0 >>  8) & 0xff] ^
                    Td3[s3 & 0xff] ^
                    rk[6];

                t3 = Td0[s3 >> 24] ^
                    Td1[(s2 >> 16) & 0xff] ^
                    Td2[(s1 >> 8) & 0xff] ^
                    Td3[s0 & 0xff] ^
                    rk[7];

                rk += 8;

                if (--r == 0) {
                        break;
                }

                s0 = Td0[t0 >> 24] ^
                    Td1[(t3 >> 16) & 0xff] ^
                    Td2[(t2 >> 8) & 0xff] ^
                    Td3[t1 & 0xff] ^
                    rk[0];

                s1 = Td0[t1 >> 24] ^
                    Td1[(t0 >> 16) & 0xff] ^
                    Td2[(t3 >> 8) & 0xff] ^
```



```
                    Td3[t2 & 0xff] ^
                    rk[1];

                s2 = Td0[t2 >> 24] ^
                    Td1[(t1 >> 16) & 0xff] ^
                    Td2[(t0 >> 8) & 0xff] ^
                    Td3[t3 & 0xff] ^
                    rk[2];

                s3 = Td0[t3 >> 24] ^
                    Td1[(t2 >> 16) & 0xff] ^
                    Td2[(t1 >> 8) & 0xff] ^
                    Td3[t0 & 0xff] ^
                    rk[3];
        }

        /*
         * apply last round and
         * map cipher state to byte array block:
         */

        s0 = (Td4[t0 >> 24] & 0xff000000) ^
            (Td4[(t3 >> 16) & 0xff] & 0x00ff0000) ^
            (Td4[(t2 >> 8) & 0xff] & 0x0000ff00) ^
            (Td4[t1 & 0xff] & 0x000000ff) ^
            rk[0];
        pt[0] = s0;

        s1 = (Td4[t1 >> 24] & 0xff000000) ^
            (Td4[(t0 >> 16) & 0xff] & 0x00ff0000) ^
            (Td4[(t3 >>  8) & 0xff] & 0x0000ff00) ^
            (Td4[t2 & 0xff] & 0x000000ff) ^
            rk[1];
        pt[1] = s1;

        s2 = (Td4[t2 >> 24] & 0xff000000) ^
            (Td4[(t1 >> 16) & 0xff] & 0x00ff0000) ^
            (Td4[(t0 >> 8) & 0xff] & 0x0000ff00) ^
            (Td4[t3 & 0xff] & 0x000000ff) ^
            rk[2];
        pt[2] = s2;

        s3 = (Td4[t3 >> 24] & 0xff000000) ^
            (Td4[(t2 >> 16) & 0xff] & 0x00ff0000) ^
            (Td4[(t1 >>  8) & 0xff] & 0x0000ff00) ^
            (Td4[t0 & 0xff] & 0x000000ff) ^
            rk[3];
        pt[3] = s3;
}
#endif  /* sun4u, !__amd64 */

#ifndef sun4u
/*
 * Expand the 32-bit AES cipher key array into the encryption and decryption
 * key schedules.
 *
 * Parameters:
 * key          AES key schedule to be initialized
 * keyarr32     User key
 * keyBits      AES key size (128, 192, or 256 bits)
 */
static void
aes_setupkeys(aes_key_t *key, const uint32_t *keyarr32, int keybits)
void
aes_encrypt_impl(const aes_ks_t *ks, int Nr, const uint32_t pt[4],
    uint32_t ct[4])
```

```
1420 {
1421         key->nr = rijndael_key_setup_enc(&(key->encr_ks.ks32[0]), keyarr32,
1422             keybits);
1423         key->nr = rijndael_key_setup_dec(&(key->decr_ks.ks32[0]), keyarr32,
1424             keybits);
1425         key->type = AES_32BIT_KS;
1355         rijndael_encrypt(&(ks->ks32[0]), Nr, pt, ct);
1426 }

1358 void
1359 aes_decrypt_impl(const aes_ks_t *ks, int Nr, const uint32_t ct[4],
1360     uint32_t pt[4])
1361 {
1362         rijndael_decrypt(&(ks->ks32[0]), Nr, ct, pt);
1363 }

1427 #endif  /* sun4u */
1428 /* EXPORT DELETE END */


1431 /*
1432  * Initialize key schedules for AES
1433  *
1434  * Parameters:
1435  * cipherKey    User key
1436  * keyBits      AES key size (128, 192, or 256 bits)
1437  * keysched     AES key schedule to be initialized, of type aes_key_t.
1438  *              Allocated by aes_alloc_keysched().
1439  */
1440 void
1441 aes_init_keysched(const uint8_t *cipherKey, uint_t keyBits, void *keysched)
1373 aes_init_keysched(uint8_t *cipherKey, uint_t keyBits, void *keysched)
1442 {
1443 /* EXPORT DELETE START */
1444         aes_key_t       *newbie = keysched;
1377         uint64_t keyarr64[4];
1378         uint32_t *keyarr32 = (uint32_t *)keyarr64;
1445         uint_t          keysize, i, j;
1446         union {
1447                 uint64_t        ka64[4];
1448                 uint32_t        ka32[8];
1449                 } keyarr;

1451         switch (keyBits) {
1452         case 128:
1453                 newbie->nr = 10;
1454                 break;

1456         case 192:
1457                 newbie->nr = 12;
1458                 break;

1460         case 256:
1461                 newbie->nr = 14;
1462                 break;

1464         default:
1465                 /* should never get here */
1466                 return;
1467         }
1468         keysize = keyBits >> 3;

1470         /*
1471          * For _LITTLE_ENDIAN machines (except AMD64), reverse every
1472          * 4 bytes in the key.  On _BIG_ENDIAN and AMD64, copy the key
1473          * without reversing bytes.
```



```
1474          * For AMD64, do not byte swap for aes_setupkeys().
1475          *
1476          * SPARCv8/v9 uses a key schedule array with 64-bit elements.
1477          * X86/AMD64  uses a key schedule array with 32-bit elements.
1401          * The code below, that is always executed on LITTLE_ENDIAN machines,
1402          * reverses every 4 bytes in the key.  On BIG_ENDIAN, the same code
1403          * copies the key without reversing bytes.
1478          */
1479 #ifndef AES_BYTE_SWAP
1405 #ifdef _BIG_ENDIAN
1480         if (IS_P2ALIGNED(cipherKey, sizeof (uint64_t))) {
1481                 for (i = 0, j = 0; j < keysize; i++, j += 8) {
1482                         /* LINTED: pointer alignment */
1483                         keyarr.ka64[i] = *((uint64_t *)&cipherKey[j]);
1409                         keyarr64[i] = *((uint64_t *)&cipherKey[j]);
1484                 }
1485         } else {
1486                 bcopy(cipherKey, keyarr.ka32, keysize);
1487         }

1489 #else   /* byte swap */
1412 #endif
1490         for (i = 0, j = 0; j < keysize; i++, j += 4) {
1491                 keyarr.ka32[i] = (((uint32_t)cipherKey[j] << 24) |
1414                         keyarr32[i] = (((uint32_t)cipherKey[j] << 24) |
1492                     ((uint32_t)cipherKey[j + 1] << 16) |
1493                     ((uint32_t)cipherKey[j + 2] << 8) |
1494                     (uint32_t)cipherKey[j + 3]);
1495         }
1419 #ifdef _BIG_ENDIAN
1420         }
1496 #endif
1422         aes_setupkeys(newbie, keyarr32, keyBits);

1498         aes_setupkeys(newbie, keyarr.ka32, keyBits);
1499 /* EXPORT DELETE END */
1500 }


1503 /*
1504  * Encrypt one block using AES.
1505  * Align if needed and (for x86 32-bit only) byte-swap.
1506  *
1507  * Parameters:
1508  * ks   Key schedule, of type aes_key_t
1509  * pt   Input block (plain text)
1510  * ct   Output block (crypto text).  Can overlap with pt
1511  */
1512 void
1513 aes_encrypt_block(const void *ks, const uint8_t *pt, uint8_t *ct)
1428 aes_encrypt_block(void *ks, uint8_t *pt, uint8_t *ct)
1514 {
1515 /* EXPORT DELETE START */

1516         aes_key_t       *ksch = (aes_key_t *)ks;

1518 #ifndef AES_BYTE_SWAP
1519         if (IS_P2ALIGNED2(pt, ct, sizeof (uint32_t))) {
1520                 AES_ENCRYPT_IMPL(&ksch->encr_ks.ks32[0], ksch->nr,
1434 #ifdef _BIG_ENDIAN
1435         if (IS_P2ALIGNED(pt, sizeof (uint32_t)) &&
1436             IS_P2ALIGNED(ct, sizeof (uint32_t))) {
1521                         /* LINTED:  pointer alignment */
1522                         (uint32_t *)pt, (uint32_t *)ct);
1438                 aes_encrypt_impl(&ksch->encr_ks, ksch->nr, (uint32_t *)pt,
1439                         /* LINTED:  pointer alignment */
```

```
1440                    (uint32_t *)ct);
1523        } else {
1524 #endif
1525                uint32_t buffer[AES_BLOCK_LEN / sizeof (uint32_t)];

1527                /* Copy input block into buffer */
1528 #ifndef AES_BYTE_SWAP
1529                bcopy(pt, &buffer, AES_BLOCK_LEN);

1531 #else   /* byte swap */
1532                buffer[0] = (((uint32_t)pt[0] << 24) | ((uint32_t)pt[1] << 16) |
1533                    ((uint32_t)pt[2] << 8) | (uint32_t)pt[3]);

1534                buffer[1] = (((uint32_t)pt[4] << 24) | ((uint32_t)pt[5] << 16) |
1535                    ((uint32_t)pt[6] << 8) | (uint32_t)pt[7]);

1536                buffer[2] = (((uint32_t)pt[8] << 24) | ((uint32_t)pt[9] << 16) |
1537                    ((uint32_t)pt[10] << 8) | (uint32_t)pt[11]);

1538                buffer[3] = (((uint32_t)pt[12] << 24) |
1539                    ((uint32_t)pt[13] << 16) | ((uint32_t)pt[14] << 8) |
1540                    (uint32_t)pt[15]);
1541 #endif

1543                AES_ENCRYPT_IMPL(&ksch->encr_ks.ks32[0], ksch->nr,
1544                    buffer, buffer);
1458                aes_encrypt_impl(&ksch->encr_ks, ksch->nr, buffer, buffer);

1546                /* Copy result from buffer to output block */
1547 #ifndef AES_BYTE_SWAP
1548                bcopy(&buffer, ct, AES_BLOCK_LEN);
1549        }

1551 #else   /* byte swap */
1552                ct[0] = buffer[0] >> 24;
1553                ct[1] = buffer[0] >> 16;
1554                ct[2] = buffer[0] >> 8;
1555                ct[3] = (uint8_t)buffer[0];
1556                ct[4] = buffer[1] >> 24;
1557                ct[5] = buffer[1] >> 16;
1558                ct[6] = buffer[1] >> 8;
1559                ct[7] = (uint8_t)buffer[1];
1560                ct[8] = buffer[2] >> 24;
1561                ct[9] = buffer[2] >> 16;
1562                ct[10] = buffer[2] >> 8;
1563                ct[11] = (uint8_t)buffer[2];
1564                ct[12] = buffer[3] >> 24;
1565                ct[13] = buffer[3] >> 16;
1566                ct[14] = buffer[3] >> 8;
1567                ct[15] = (uint8_t)buffer[3];
1476 #ifdef _BIG_ENDIAN
1477        }
1568 #endif

1569 /* EXPORT DELETE END */
1570 }

1573 /*
1574  * Decrypt one block using AES.
1575  * Align and byte-swap if needed.
1576  *
1577  * Parameters:
1578  * ks   Key schedule, of type aes_key_t
1579  * ct   Input block (crypto text)
1580  * pt   Output block (plain text). Can overlap with pt
```



```
1581  */
1582 void
1583 aes_decrypt_block(const void *ks, const uint8_t *ct, uint8_t *pt)
1484 aes_decrypt_block(void *ks, uint8_t *ct, uint8_t *pt)
1584 {
1585 /* EXPORT DELETE START */

1586        aes_key_t       *ksch = (aes_key_t *)ks;

1588 #ifndef AES_BYTE_SWAP
1589        if (IS_P2ALIGNED2(ct, pt, sizeof (uint32_t))) {
1590                AES_DECRYPT_IMPL(&ksch->decr_ks.ks32[0], ksch->nr,
1490 #ifdef _BIG_ENDIAN
1491        if (IS_P2ALIGNED(ct, sizeof (uint32_t)) &&
1492            IS_P2ALIGNED(pt, sizeof (uint32_t))) {
1591                    /* LINTED:  pointer alignment */
1592                    (uint32_t *)ct, (uint32_t *)pt);
1494                aes_decrypt_impl(&ksch->decr_ks, ksch->nr, (uint32_t *)ct,
1495                    /* LINTED:  pointer alignment */
1496                    (uint32_t *)pt);
1593        } else {
1594 #endif
1595                uint32_t buffer[AES_BLOCK_LEN / sizeof (uint32_t)];

1597                /* Copy input block into buffer */
1598 #ifndef AES_BYTE_SWAP
1599                bcopy(ct, &buffer, AES_BLOCK_LEN);

1601 #else   /* byte swap */
1602                buffer[0] = (((uint32_t)ct[0] << 24) | ((uint32_t)ct[1] << 16) |
1603                    ((uint32_t)ct[2] << 8) | (uint32_t)ct[3]);

1605                buffer[1] = (((uint32_t)ct[4] << 24) | ((uint32_t)ct[5] << 16) |
1606                    ((uint32_t)ct[6] << 8) | (uint32_t)ct[7]);

1608                buffer[2] = (((uint32_t)ct[8] << 24) | ((uint32_t)ct[9] << 16) |
1609                    ((uint32_t)ct[10] << 8) | (uint32_t)ct[11]);

1611                buffer[3] = (((uint32_t)ct[12] << 24) |
1612                    ((uint32_t)ct[13] << 16) | ((uint32_t)ct[14] << 8) |
1613                    (uint32_t)ct[15]);
1614 #endif

1616                AES_DECRYPT_IMPL(&ksch->decr_ks.ks32[0], ksch->nr,
1617                    buffer, buffer);
1514                aes_decrypt_impl(&ksch->decr_ks, ksch->nr, buffer, buffer);

1619                /* Copy result from buffer to output block */
1620 #ifndef AES_BYTE_SWAP
1621                bcopy(&buffer, pt, AES_BLOCK_LEN);
1622        }

1624 #else   /* byte swap */
1625                pt[0] = buffer[0] >> 24;
1626                pt[1] = buffer[0] >> 16;
1627                pt[2] = buffer[0] >> 8;
1628                pt[3] = (uint8_t)buffer[0];
1629                pt[4] = buffer[1] >> 24;
1630                pt[5] = buffer[1] >> 16;
1631                pt[6] = buffer[1] >> 8;
1632                pt[7] = (uint8_t)buffer[1];
1633                pt[8] = buffer[2] >> 24;
1634                pt[9] = buffer[2] >> 16;
1635                pt[10] = buffer[2] >> 8;
1636                pt[11] = (uint8_t)buffer[2];
1637                pt[12] = buffer[3] >> 24;
```

```
1638                 pt[13] = buffer[3] >> 16;
1639                 pt[14] = buffer[3] >> 8;
1640                 pt[15] = (uint8_t)buffer[3];
1532 #ifdef _BIG_ENDIAN
1533         }
1641 #endif

1643 /* EXPORT DELETE END */
1644 }


1647 /*
1648  * Allocate key schedule for AES.
1649  *
1650  * Return the pointer and set size to the number of bytes allocated.
1651  * Memory allocated must be freed by the caller when done.
1652  *
1653  * Parameters:
1654  * size         Size of key schedule allocated, in bytes
1655  * kmflag       Flag passed to kmem_alloc(9F); ignored in userland.
1540  * Allocate key schedule for AES
1656  */
1657 /* ARGSUSED */
1658 void *
1659 aes_alloc_keysched(size_t *size, int kmflag)
1660 {

1661 /* EXPORT DELETE START */

1662         aes_key_t *keysched;

1664 #ifdef  _KERNEL
1665         keysched = (aes_key_t *)kmem_alloc(sizeof (aes_key_t), kmflag);
1666 #else   /* !_KERNEL */
1667         keysched = (aes_key_t *)malloc(sizeof (aes_key_t));
1668 #endif  /* _KERNEL */

1670         if (keysched != NULL) {
1671                 *size = sizeof (aes_key_t);
1672                 return (keysched);
1673         }

1674 /* EXPORT DELETE END */

1675         return (NULL);
1676 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3499 Fri Jun 13 16:44:38 2008
new/usr/src/common/crypto/aes/aes_impl.h
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _AES_IMPL_H
#define _AES_IMPL_H

#pragma ident   "@(#)aes_impl.h 1.4     08/05/19 SMI"
#pragma ident   "@(#)aes_impl.h 1.3     08/02/26 SMI"

/*
 * Common definitions used by AES.
 */

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#define AES_BLOCK_LEN 16

/* Similar to sysmacros.h IS_P2ALIGNED, but checks two pointers: */
#define IS_P2ALIGNED2(v, w, a) \
        ((((uintptr_t)(v) | (uintptr_t)(w)) & ((uintptr_t)(a) - 1)) == 0)

#define AES_BLOCK_LEN   16      /* bytes */
/* Round constant length, in number of 32-bit elements: */
#define RC_LENGTH       (5 * ((AES_BLOCK_LEN) / 4 - 2))

#define AES_COPY_BLOCK(src, dst) \
        (dst)[0] = (src)[0]; \
        (dst)[1] = (src)[1]; \
        (dst)[2] = (src)[2]; \
        (dst)[3] = (src)[3]; \
        (dst)[4] = (src)[4]; \
        (dst)[5] = (src)[5]; \
        (dst)[6] = (src)[6]; \
        (dst)[7] = (src)[7]; \
        (dst)[8] = (src)[8]; \
        (dst)[9] = (src)[9]; \
```



```
        (dst)[10] = (src)[10]; \
        (dst)[11] = (src)[11]; \
        (dst)[12] = (src)[12]; \
        (dst)[13] = (src)[13]; \
        (dst)[14] = (src)[14]; \
        (dst)[15] = (src)[15]

#define AES_XOR_BLOCK(src, dst) \
        (dst)[0] ^= (src)[0]; \
        (dst)[1] ^= (src)[1]; \
        (dst)[2] ^= (src)[2]; \
        (dst)[3] ^= (src)[3]; \
        (dst)[4] ^= (src)[4]; \
        (dst)[5] ^= (src)[5]; \
        (dst)[6] ^= (src)[6]; \
        (dst)[7] ^= (src)[7]; \
        (dst)[8] ^= (src)[8]; \
        (dst)[9] ^= (src)[9]; \
        (dst)[10] ^= (src)[10]; \
        (dst)[11] ^= (src)[11]; \
        (dst)[12] ^= (src)[12]; \
        (dst)[13] ^= (src)[13]; \
        (dst)[14] ^= (src)[14]; \
        (dst)[15] ^= (src)[15]

/* AES key size definitions */
#define AES_MINBITS             128
#define AES_MINBYTES            ((AES_MINBITS) >> 3)
#define AES_MAXBITS             256
#define AES_MAXBYTES            ((AES_MAXBITS) >> 3)

#define AES_MIN_KEY_BYTES       ((AES_MINBITS) >> 3)
#define AES_MAX_KEY_BYTES       ((AES_MAXBITS) >> 3)
#define AES_192_KEY_BYTES       24
#define AES_IV_LEN              16

/* AES key schedule may be implemented with 32- or 64-bit elements: */
#define AES_32BIT_KS            32
#define AES_64BIT_KS            64

#define MAX_AES_NR              14 /* Maximum number of rounds */
#define MAX_AES_NB              4  /* Number of columns comprising a state */

typedef union {
#ifdef  sun4u
        uint64_t        ks64[((MAX_AES_NR) + 1) * (MAX_AES_NB)];
#endif
        uint32_t        ks32[((MAX_AES_NR) + 1) * (MAX_AES_NB)];
} aes_ks_t;
```

_____*unchanged_portion_omitted_*

```
extern void aes_encrypt_block(const void *ks, const uint8_t *pt, uint8_t *ct);
extern void aes_decrypt_block(const void *ks, const uint8_t *ct, uint8_t *pt);
extern void aes_init_keysched(const uint8_t *cipherKey, uint_t keyBits,
        void *keysched);
extern void *aes_alloc_keysched(size_t *size, int kmflag);
```

```
108 extern void *aes_alloc_keysched(size_t *, int);
109 extern void aes_encrypt_impl(const aes_ks_t *ks, int Nr, const uint32_t pt[4],
110     uint32_t ct[4]);
111 extern void aes_decrypt_impl(const aes_ks_t *ks, int Nr, const uint32_t ct[4],
112     uint32_t pt[4]);

124 #ifdef  __cplusplus
125 }
```

_____***unchanged_portion_omitted_***

****
951 Fri Jun 13 16:44:41 2008
new/usr/src/common/crypto/aes/amd64/THIRDPARTYLICENSE
5072963 Need an optimized AES implementation for amd64
****

```
------------------------------------------------------------------------
Copyright (c) 1998-2007, Brian Gladman, Worcester, UK. All rights reserved.

LICENSE TERMS

The free distribution and use of this software is allowed (with or without
changes) provided that:

 1. source code distributions include the above copyright notice, this
    list of conditions and the following disclaimer;

 2. binary distributions include the above copyright notice, this list
    of conditions and the following disclaimer in their documentation;

 3. the name of the copyright holder is not used to endorse products
    built using this software without specific written permission.

DISCLAIMER

This software is provided 'as is' with no explicit or implied warranties
in respect of its properties, including, but not limited to, correctness
and/or fitness for purpose.
------------------------------------------------------------------------
```

****

30 Fri Jun 13 16:44:42 2008
new/usr/src/common/crypto/aes/amd64/THIRDPARTYLICENSE.descrip
5072963 Need an optimized AES implementation for amd64

****

```
   1 PORTIONS OF AES FUNCTIONALITY
```

```
**********************************************************
   27316 Fri Jun 13 16:44:43 2008
new/usr/src/common/crypto/aes/amd64/aes_amd64.s
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * ---------------------------------------------------------------------------
 * Copyright (c) 1998-2007, Brian Gladman, Worcester, UK. All rights reserved.
 *
 * LICENSE TERMS
 *
 * The free distribution and use of this software is allowed (with or without
 * changes) provided that:
 *
 *  1. source code distributions include the above copyright notice, this
 *     list of conditions and the following disclaimer;
 *
 *  2. binary distributions include the above copyright notice, this list
 *     of conditions and the following disclaimer in their documentation;
 *
 *  3. the name of the copyright holder is not used to endorse products
 *     built using this software without specific written permission.
 *
 * DISCLAIMER
 *
 * This software is provided 'as is' with no explicit or implied warranties
 * in respect of its properties, including, but not limited to, correctness
 * and/or fitness for purpose.
 * ---------------------------------------------------------------------------
 * Issue 20/12/2007
 *
 * I am grateful to Dag Arne Osvik for many discussions of the techniques that
 * can be used to optimise AES assembler code on AMD64/EM64T architectures.
 * Some of the techniques used in this implementation are the result of
 * suggestions made by him for which I am most grateful.
 *
 * An AES implementation for AMD64 processors using the YASM assembler.  This
 * implementation provides only encryption, decryption and hence requires key
 * scheduling support in C. It uses 8k bytes of tables but its encryption and
 * decryption performance is very close to that obtained using large tables.
 * It can use either MS Windows or Gnu/Linux/OpenSolaris OS calling conventions,
 * which are as follows:
 *                ms windows  gnu/linux/opensolaris os
 *
 *   in_blk          rcx     rdi
 *   out_blk         rdx     rsi
 *   context (cx)     r8     rdx
 *
 *   preserved       rsi      -    + rbx, rbp, rsp, r12, r13, r14 & r15
 *   registers       rdi      -      on both
 *
 *   destroyed        -      rsi   + rax, rcx, rdx, r8, r9, r10 & r11
 *   registers        -      rdi     on both
 *
 * The convention used here is that for gnu/linux/opensolaris os.
 *
 * This code provides the standard AES block size (128 bits, 16 bytes) and the
 * three standard AES key sizes (128, 192 and 256 bits). It has the same call
 * interface as my C implementation.  It uses the Microsoft C AMD64 calling
 * conventions in which the three parameters are placed in  rcx, rdx and r8
 * respectively.  The rbx, rsi, rdi, rbp and r12..r15 registers are preserved.
 *
 * OpenSolaris Note:
 * Modified to use GNU/Linux/Solaris calling conventions.
 * That is parameters are placed in rdi, rsi, rdx, and rcx, respectively.
 *
```



```
 *     AES_RETURN aes_encrypt(const unsigned char in_blk[],
 *                    unsigned char out_blk[], const aes_encrypt_ctx cx[1])/
 *
 *     AES_RETURN aes_decrypt(const unsigned char in_blk[],
 *                    unsigned char out_blk[], const aes_decrypt_ctx cx[1])/
 *
 *     AES_RETURN aes_encrypt_key<NNN>(const unsigned char key[],
 *                                           const aes_encrypt_ctx cx[1])/
 *
 *     AES_RETURN aes_decrypt_key<NNN>(const unsigned char key[],
 *                                           const aes_decrypt_ctx cx[1])/
 *
 *     AES_RETURN aes_encrypt_key(const unsigned char key[],
 *                                unsigned int len, const aes_decrypt_ctx cx[1])/
 *
 *     AES_RETURN aes_decrypt_key(const unsigned char key[],
 *                                unsigned int len, const aes_decrypt_ctx cx[1])/
 *
 * where <NNN> is 128, 102 or 256.  In the last two calls the length can be in
 * either bits or bytes.
 *
 * Comment in/out the following lines to obtain the desired subroutines. These
 * selections MUST match those in the C header file aesopt.h
 */
#define AES_REV_DKS       /* define if key decryption schedule is reversed */

#define LAST_ROUND_TABLES /* define for the faster version using extra tables */

/*
 * The encryption key schedule has the following in memory layout where N is the
 * number of rounds (10, 12 or 14):
 *
 * lo: | input key (round 0)  |  / each round is four 32-bit words
 *     | encryption round 1   |
 *     | encryption round 2   |
 *     ....
 *     | encryption round N-1 |
 * hi: | encryption round N   |
 *
 * The decryption key schedule is normally set up so that it has the same
 * layout as above by actually reversing the order of the encryption key
 * schedule in memory (this happens when AES_REV_DKS is set):
 *
 * lo: | decryption round 0   | =             | encryption round N   |
 *     | decryption round 1   | = INV_MIX_COL[ | encryption round N-1 | ]
 *     | decryption round 2   | = INV_MIX_COL[ | encryption round N-2 | ]
 *     ....                       ....
 *     | decryption round N-1 | = INV_MIX_COL[ | encryption round 1   | ]
 * hi: | decryption round N   | =             | input key (round 0)  |
 *
 * with rounds except the first and last modified using inv_mix_column()
 * But if AES_REV_DKS is NOT set the order of keys is left as it is for
 * encryption so that it has to be accessed in reverse when used for
 * decryption (although the inverse mix column modifications are done)
 *
 * lo: | decryption round 0   | =             | input key (round 0)  |
 *     | decryption round 1   | = INV_MIX_COL[ | encryption round 1   | ]
 *     | decryption round 2   | = INV_MIX_COL[ | encryption round 2   | ]
 *     ....                       ....
 *     | decryption round N-1 | = INV_MIX_COL[ | encryption round N-1 | ]
 * hi: | decryption round N   | =             | encryption round N   |
 *
 * This layout is faster when the assembler key scheduling provided here
 * is used.
 *
 * End of user defines
```

```
 */

/*
 * ---------------------------------------------------------------------------
 * OpenSolaris OS modifications
 *
 * This source originates from Brian Gladman file aes_amd64.asm
 * in http://fp.gladman.plus.com/AES/aes-src-04-03-08.zip
 * with these changes:
 *
 * 1. Removed MS Windows-specific code within DLL_EXPORT, _SEH_, and
 * !__GNUC__ ifdefs.  Also removed ENCRYPTION, DECRYPTION,
 * AES_128, AES_192, AES_256, AES_VAR ifdefs.
 *
 * 2. Translate yasm/nasm %define and .macro definitions to cpp(1) #define
 *
 * 3. Translate yasm/nasm %ifdef/%ifndef to cpp(1) #ifdef
 *
 * 4. Translate Intel/yasm/nasm syntax to ATT/OpenSolaris as(1) syntax
 * (operands reversed, literals prefixed with "$", registers prefixed with "%",
 * and "[register+offset]", addressing changed to "offset(register)",
 * parenthesis in constant expressions "()" changed to square brackets "[]",
 * "." removed from  local (numeric) labels, and other changes.
 * Examples:
 * Intel/yasm/nasm Syntax       ATT/OpenSolaris Syntax
 * mov  rax,(4*20h)             mov     $[4*0x20],%rax
 * mov  rax,[ebx+20h]           mov     0x20(%ebx),%rax
 * lea  rax,[ebx+ecx]           lea     (%ebx,%ecx),%rax
 * sub  rax,[ebx+ecx*4-20h]     sub     -0x20(%ebx,%ecx,4),%rax
 *
 * 5. Added OpenSolaris ENTRY_NP/SET_SIZE macros from
 * /usr/include/sys/asm_linkage.h, .ident keywords, and lint(1B) guards.
 *
 * 6. Renamed functions and reordered parameters to match OpenSolaris:
 * Original Gladman interface:
 *      int aes_encrypt(const unsigned char *in,
 *              unsigned char *out, const aes_encrypt_ctx cx[1])/
 *      int aes_decrypt(const unsigned char *in,
 *              unsigned char *out, const aes_encrypt_ctx cx[1])/
 * Note: aes_encrypt_ctx contains ks, a 60 element array of uint32_t,
 * and a union type, inf., containing inf.l, a uint32_t and
 * inf.b, a 4-element array of uint32_t.  Only b[0] in the array (aka "l") is
 * used and contains the key schedule length * 16 where key schedule length is
 * 10, 12, or 14 bytes.
 *
 * OpenSolaris OS interface:
 *      void aes_encrypt_impl(const aes_ks_t *ks, int Nr,
 *              const uint32_t pt[4], uint32_t ct[4])/
 *      void aes_decrypt_impl(const aes_ks_t *ks, int Nr,
 *              const uint32_t pt[4], uint32_t ct[4])/
 *      typedef union {uint64_t ks64[(MAX_AES_NR + 1) * 4]/
 *               uint32_t ks32[(MAX_AES_NR + 1) * 4]/ } aes_ks_t/
 * Note: ks is the AES key schedule, Nr is number of rounds, pt is plain text,
 * ct is crypto text, and MAX_AES_NR is 14.
 * For the x86 64-bit architecture, OpenSolaris OS uses ks32 instead of ks64.
 */

#if !defined(lint) && !defined(__lint)
        .ident  "@(#)aes_amd64.s        1.1     08/06/11 SMI"
#include <sys/asm_linkage.h>

#define KS_LENGTH       60

#define raxd            eax
#define rdxd            edx
#define rcxd            ecx
```



```
#define rbxd            ebx
#define rsid            esi
#define rdid            edi

#define raxb            al
#define rdxb            dl
#define rcxb            cl
#define rbxb            bl
#define rsib            sil
#define rdib            dil

/ finite field multiplies by {02}, {04} and {08}

#define f2(x) [[x<<1]^[[[x>>7]&1]*0x11b]]
#define f4(x) [[x<<2]^[[[x>>6]&1]*0x11b]^[[[x>>6]&2]*0x11b]]
#define f8(x) [[x<<3]^[[[x>>5]&1]*0x11b]^[[[x>>5]&2]*0x11b]^[[[x>>5]&4]*0x11b]]

/ finite field multiplies required in table generation

#define f3(x) [[f2(x)] ^ [x]]
#define f9(x) [[f8(x)] ^ [x]]
#define fb(x) [[f8(x)] ^ [f2(x)] ^ [x]]
#define fd(x) [[f8(x)] ^ [f4(x)] ^ [x]]
#define fe(x) [[f8(x)] ^ [f4(x)] ^ [f2(x)]]

/ macros for expanding S-box data

#define u8(x) [f2(x)], [x], [x], [f3(x)], [f2(x)], [x], [x], [f3(x)]
#define v8(x) [fe(x)], [f9(x)], [fd(x)], [fb(x)], [fe(x)], [f9(x)], [fd(x)], [x]
#define w8(x) [x], 0, 0, 0, [x], 0, 0, 0

#define enc_vals(x)     \
   .byte x(0x63),x(0x7c),x(0x77),x(0x7b),x(0xf2),x(0x6b),x(0x6f),x(0xc5); \
   .byte x(0x30),x(0x01),x(0x67),x(0x2b),x(0xfe),x(0xd7),x(0xab),x(0x76); \
   .byte x(0xca),x(0x82),x(0xc9),x(0x7d),x(0xfa),x(0x59),x(0x47),x(0xf0); \
   .byte x(0xad),x(0xd4),x(0xa2),x(0xaf),x(0x9c),x(0xa4),x(0x72),x(0xc0); \
   .byte x(0xb7),x(0xfd),x(0x93),x(0x26),x(0x36),x(0x3f),x(0xf7),x(0xcc); \
   .byte x(0x34),x(0xa5),x(0xe5),x(0xf1),x(0x71),x(0xd8),x(0x31),x(0x15); \
   .byte x(0x04),x(0xc7),x(0x23),x(0xc3),x(0x18),x(0x96),x(0x05),x(0x9a); \
   .byte x(0x07),x(0x12),x(0x80),x(0xe2),x(0xeb),x(0x27),x(0xb2),x(0x75); \
   .byte x(0x09),x(0x83),x(0x2c),x(0x1a),x(0x1b),x(0x6e),x(0x5a),x(0xa0); \
   .byte x(0x52),x(0x3b),x(0xd6),x(0xb3),x(0x29),x(0xe3),x(0x2f),x(0x84); \
   .byte x(0x53),x(0xd1),x(0x00),x(0xed),x(0x20),x(0xfc),x(0xb1),x(0x5b); \
   .byte x(0x6a),x(0xcb),x(0xbe),x(0x39),x(0x4a),x(0x4c),x(0x58),x(0xcf); \
   .byte x(0xd0),x(0xef),x(0xaa),x(0xfb),x(0x43),x(0x4d),x(0x33),x(0x85); \
   .byte x(0x45),x(0xf9),x(0x02),x(0x7f),x(0x50),x(0x3c),x(0x9f),x(0xa8); \
   .byte x(0x51),x(0xa3),x(0x40),x(0x8f),x(0x92),x(0x9d),x(0x38),x(0xf5); \
   .byte x(0xbc),x(0xb6),x(0xda),x(0x21),x(0x10),x(0xff),x(0xf3),x(0xd2); \
   .byte x(0xcd),x(0x0c),x(0x13),x(0xec),x(0x5f),x(0x97),x(0x44),x(0x17); \
   .byte x(0xc4),x(0xa7),x(0x7e),x(0x3d),x(0x64),x(0x5d),x(0x19),x(0x73); \
   .byte x(0x60),x(0x81),x(0x4f),x(0xdc),x(0x22),x(0x2a),x(0x90),x(0x88); \
   .byte x(0x46),x(0xee),x(0xb8),x(0x14),x(0xde),x(0x5e),x(0x0b),x(0xdb); \
   .byte x(0xe0),x(0x32),x(0x3a),x(0x0a),x(0x49),x(0x06),x(0x24),x(0x5c); \
   .byte x(0xc2),x(0xd3),x(0xac),x(0x62),x(0x91),x(0x95),x(0xe4),x(0x79); \
   .byte x(0xe7),x(0xc8),x(0x37),x(0x6d),x(0x8d),x(0xd5),x(0x4e),x(0xa9); \
   .byte x(0x6c),x(0x56),x(0xf4),x(0xea),x(0x65),x(0x7a),x(0xae),x(0x08); \
   .byte x(0xba),x(0x78),x(0x25),x(0x2e),x(0x1c),x(0xa6),x(0xb4),x(0xc6); \
   .byte x(0xe8),x(0xdd),x(0x74),x(0x1f),x(0x4b),x(0xbd),x(0x8b),x(0x8a); \
   .byte x(0x70),x(0x3e),x(0xb5),x(0x66),x(0x48),x(0x03),x(0xf6),x(0x0e); \
   .byte x(0x61),x(0x35),x(0x57),x(0xb9),x(0x86),x(0xc1),x(0x1d),x(0x9e); \
   .byte x(0xe1),x(0xf8),x(0x98),x(0x11),x(0x69),x(0xd9),x(0x8e),x(0x94); \
   .byte x(0x9b),x(0x1e),x(0x87),x(0xe9),x(0xce),x(0x55),x(0x28),x(0xdf); \
   .byte x(0x8c),x(0xa1),x(0x89),x(0x0d),x(0xbf),x(0xe6),x(0x42),x(0x68); \
   .byte x(0x41),x(0x99),x(0x2d),x(0x0f),x(0xb0),x(0x54),x(0xbb),x(0x16)

#define dec_vals(x) \
```

```
260     .byte x(0x52),x(0x09),x(0x6a),x(0xd5),x(0x30),x(0x36),x(0xa5),x(0x38); \
261     .byte x(0xbf),x(0x40),x(0xa3),x(0x9e),x(0x81),x(0xf3),x(0xd7),x(0xfb); \
262     .byte x(0x7c),x(0xe3),x(0x39),x(0x82),x(0x9b),x(0x2f),x(0xff),x(0x87); \
263     .byte x(0x34),x(0x8e),x(0x43),x(0x44),x(0xc4),x(0xde),x(0xe9),x(0xcb); \
264     .byte x(0x54),x(0x7b),x(0x94),x(0x32),x(0xa6),x(0xc2),x(0x23),x(0x3d); \
265     .byte x(0xee),x(0x4c),x(0x95),x(0x0b),x(0x42),x(0xfa),x(0xc3),x(0x4e); \
266     .byte x(0x08),x(0x2e),x(0xa1),x(0x66),x(0x28),x(0xd9),x(0x24),x(0xb2); \
267     .byte x(0x76),x(0x5b),x(0xa2),x(0x49),x(0x6d),x(0x8b),x(0xd1),x(0x25); \
268     .byte x(0x72),x(0xf8),x(0xf6),x(0x64),x(0x86),x(0x68),x(0x98),x(0x16); \
269     .byte x(0xd4),x(0xa4),x(0x5c),x(0xcc),x(0x5d),x(0x65),x(0xb6),x(0x92); \
270     .byte x(0x6c),x(0x70),x(0x48),x(0x50),x(0xfd),x(0xed),x(0xb9),x(0xda); \
271     .byte x(0x5e),x(0x15),x(0x46),x(0x57),x(0xa7),x(0x8d),x(0x9d),x(0x84); \
272     .byte x(0x90),x(0xd8),x(0xab),x(0x00),x(0x8c),x(0xbc),x(0xd3),x(0x0a); \
273     .byte x(0xf7),x(0xe4),x(0x58),x(0x05),x(0xb8),x(0xb3),x(0x45),x(0x06); \
274     .byte x(0xd0),x(0x2c),x(0x1e),x(0x8f),x(0xca),x(0x3f),x(0x0f),x(0x02); \
275     .byte x(0xc1),x(0xaf),x(0xbd),x(0x03),x(0x01),x(0x13),x(0x8a),x(0x6b); \
276     .byte x(0x3a),x(0x91),x(0x11),x(0x41),x(0x4f),x(0x67),x(0xdc),x(0xea); \
277     .byte x(0x97),x(0xf2),x(0xcf),x(0xce),x(0xf0),x(0xb4),x(0xe6),x(0x73); \
278     .byte x(0x96),x(0xac),x(0x74),x(0x22),x(0xe7),x(0xad),x(0x35),x(0x85); \
279     .byte x(0xe2),x(0xf9),x(0x37),x(0xe8),x(0x1c),x(0x75),x(0xdf),x(0x6e); \
280     .byte x(0x47),x(0xf1),x(0x1a),x(0x71),x(0x1d),x(0x29),x(0xc5),x(0x89); \
281     .byte x(0x6f),x(0xb7),x(0x62),x(0x0e),x(0xaa),x(0x18),x(0xbe),x(0x1b); \
282     .byte x(0xfc),x(0x56),x(0x3e),x(0x4b),x(0xc6),x(0xd2),x(0x79),x(0x20); \
283     .byte x(0x9a),x(0xdb),x(0xc0),x(0xfe),x(0x78),x(0xcd),x(0x5a),x(0xf4); \
284     .byte x(0x1f),x(0xdd),x(0xa8),x(0x33),x(0x88),x(0x07),x(0xc7),x(0x31); \
285     .byte x(0xb1),x(0x12),x(0x10),x(0x59),x(0x27),x(0x80),x(0xec),x(0x5f); \
286     .byte x(0x60),x(0x51),x(0x7f),x(0xa9),x(0x19),x(0xb5),x(0x4a),x(0x0d); \
287     .byte x(0x2d),x(0xe5),x(0x7a),x(0x9f),x(0x93),x(0xc9),x(0x9c),x(0xef); \
288     .byte x(0xa0),x(0xe0),x(0x3b),x(0x4d),x(0xae),x(0x2a),x(0xf5),x(0xb0); \
289     .byte x(0xc8),x(0xeb),x(0xbb),x(0x3c),x(0x83),x(0x53),x(0x99),x(0x61); \
290     .byte x(0x17),x(0x2b),x(0x04),x(0x7e),x(0xba),x(0x77),x(0xd6),x(0x26); \
291     .byte x(0xe1),x(0x69),x(0x14),x(0x63),x(0x55),x(0x21),x(0x0c),x(0x7d)

293 #define tptr    %rbp    /* table pointer */
294 #define kptr    %r8     /* key schedule pointer */
295 #define fofs    128     /* adjust offset in key schedule to keep |disp| < 128 */
296 #define fk_ref(x, y)    -16*x+fofs+4*y(kptr)

298 #ifdef  AES_REV_DKS
299 #define rofs            128
300 #define ik_ref(x, y)    -16*x+rofs+4*y(kptr)

302 #else
303 #define rofs            -128
304 #define ik_ref(x, y)    16*x+rofs+4*y(kptr)
305 #endif  /* AES_REV_DKS */

307 #define tab_0(x)        (tptr,x,8)
308 #define tab_1(x)        3(tptr,x,8)
309 #define tab_2(x)        2(tptr,x,8)
310 #define tab_3(x)        1(tptr,x,8)
311 #define tab_f(x)        1(tptr,x,8)
312 #define tab_i(x)        7(tptr,x,8)

314         /* EXPORT DELETE START */
315 #define ff_rnd(p1, p2, p3, p4, round)   /* normal forward round */ \
316         mov     fk_ref(round,0), p1; \
317         mov     fk_ref(round,1), p2; \
318         mov     fk_ref(round,2), p3; \
319         mov     fk_ref(round,3), p4; \
320  \
321         movzx   %al, %esi; \
322         movzx   %ah, %edi; \
323         shr     $16, %eax; \
324         xor     tab_0(%rsi), p1; \
325         xor     tab_1(%rdi), p4; \
```



```
326         movzx   %al, %esi; \
327         movzx   %ah, %edi; \
328         xor     tab_2(%rsi), p3; \
329         xor     tab_3(%rdi), p2; \
330  \
331         movzx   %bl, %esi; \
332         movzx   %bh, %edi; \
333         shr     $16, %ebx; \
334         xor     tab_0(%rsi), p2; \
335         xor     tab_1(%rdi), p1; \
336         movzx   %bl, %esi; \
337         movzx   %bh, %edi; \
338         xor     tab_2(%rsi), p4; \
339         xor     tab_3(%rdi), p3; \
340  \
341         movzx   %cl, %esi; \
342         movzx   %ch, %edi; \
343         shr     $16, %ecx; \
344         xor     tab_0(%rsi), p3; \
345         xor     tab_1(%rdi), p2; \
346         movzx   %cl, %esi; \
347         movzx   %ch, %edi; \
348         xor     tab_2(%rsi), p1; \
349         xor     tab_3(%rdi), p4; \
350  \
351         movzx   %dl, %esi; \
352         movzx   %dh, %edi; \
353         shr     $16, %edx; \
354         xor     tab_0(%rsi), p4; \
355         xor     tab_1(%rdi), p3; \
356         movzx   %dl, %esi; \
357         movzx   %dh, %edi; \
358         xor     tab_2(%rsi), p2; \
359         xor     tab_3(%rdi), p1; \
360  \
361         mov     p1, %eax; \
362         mov     p2, %ebx; \
363         mov     p3, %ecx; \
364         mov     p4, %edx

366 #ifdef  LAST_ROUND_TABLES

368 #define fl_rnd(p1, p2, p3, p4, round)   /* last forward round */ \
369         add     $2048, tptr; \
370         mov     fk_ref(round,0), p1; \
371         mov     fk_ref(round,1), p2; \
372         mov     fk_ref(round,2), p3; \
373         mov     fk_ref(round,3), p4; \
374  \
375         movzx   %al, %esi; \
376         movzx   %ah, %edi; \
377         shr     $16, %eax; \
378         xor     tab_0(%rsi), p1; \
379         xor     tab_1(%rdi), p4; \
380         movzx   %al, %esi; \
381         movzx   %ah, %edi; \
382         xor     tab_2(%rsi), p3; \
383         xor     tab_3(%rdi), p2; \
384  \
385         movzx   %bl, %esi; \
386         movzx   %bh, %edi; \
387         shr     $16, %ebx; \
388         xor     tab_0(%rsi), p2; \
389         xor     tab_1(%rdi), p1; \
390         movzx   %bl, %esi; \
391         movzx   %bh, %edi; \
```

```
392         xor     tab_2(%rsi), p4; \
393         xor     tab_3(%rdi), p3; \
394  \
395         movzx   %cl, %esi; \
396         movzx   %ch, %edi; \
397         shr     $16, %ecx; \
398         xor     tab_0(%rsi), p3; \
399         xor     tab_1(%rdi), p2; \
400         movzx   %cl, %esi; \
401         movzx   %ch, %edi; \
402         xor     tab_2(%rsi), p1; \
403         xor     tab_3(%rdi), p4; \
404  \
405         movzx   %dl, %esi; \
406         movzx   %dh, %edi; \
407         shr     $16, %edx; \
408         xor     tab_0(%rsi), p4; \
409         xor     tab_1(%rdi), p3; \
410         movzx   %dl, %esi; \
411         movzx   %dh, %edi; \
412         xor     tab_2(%rsi), p2; \
413         xor     tab_3(%rdi), p1

415 #else

417 #define fl_rnd(p1, p2, p3, p4, round)   /* last forward round */ \
418         mov     fk_ref(round,0), p1; \
419         mov     fk_ref(round,1), p2; \
420         mov     fk_ref(round,2), p3; \
421         mov     fk_ref(round,3), p4; \
422  \
423         movzx   %al, %esi; \
424         movzx   %ah, %edi; \
425         shr     $16, %eax; \
426         movzx   tab_f(%rsi), %esi; \
427         movzx   tab_f(%rdi), %edi; \
428         xor     %esi, p1; \
429         rol     $8, %edi; \
430         xor     %edi, p4; \
431         movzx   %al, %esi; \
432         movzx   %ah, %edi; \
433         movzx   tab_f(%rsi), %esi; \
434         movzx   tab_f(%rdi), %edi; \
435         rol     $16, %esi; \
436         rol     $24, %edi; \
437         xor     %esi, p3; \
438         xor     %edi, p2; \
439  \
440         movzx   %bl, %esi; \
441         movzx   %bh, %edi; \
442         shr     $16, %ebx; \
443         movzx   tab_f(%rsi), %esi; \
444         movzx   tab_f(%rdi), %edi; \
445         xor     %esi, p2; \
446         rol     $8, %edi; \
447         xor     %edi, p1; \
448         movzx   %bl, %esi; \
449         movzx   %bh, %edi; \
450         movzx   tab_f(%rsi), %esi; \
451         movzx   tab_f(%rdi), %edi; \
452         rol     $16, %esi; \
453         rol     $24, %edi; \
454         xor     %esi, p4; \
455         xor     %edi, p3; \
456  \
457         movzx   %cl, %esi; \
```



```
458         movzx   %ch, %edi; \
459         movzx   tab_f(%rsi), %esi; \
460         movzx   tab_f(%rdi), %edi; \
461         shr     $16, %ecx; \
462         xor     %esi, p3; \
463         rol     $8, %edi; \
464         xor     %edi, p2; \
465         movzx   %cl, %esi; \
466         movzx   %ch, %edi; \
467         movzx   tab_f(%rsi), %esi; \
468         movzx   tab_f(%rdi), %edi; \
469         rol     $16, %esi; \
470         rol     $24, %edi; \
471         xor     %esi, p1; \
472         xor     %edi, p4; \
473  \
474         movzx   %dl, %esi; \
475         movzx   %dh, %edi; \
476         movzx   tab_f(%rsi), %esi; \
477         movzx   tab_f(%rdi), %edi; \
478         shr     $16, %edx; \
479         xor     %esi, p4; \
480         rol     $8, %edi; \
481         xor     %edi, p3; \
482         movzx   %dl, %esi; \
483         movzx   %dh, %edi; \
484         movzx   tab_f(%rsi), %esi; \
485         movzx   tab_f(%rdi), %edi; \
486         rol     $16, %esi; \
487         rol     $24, %edi; \
488         xor     %esi, p2; \
489         xor     %edi, p1

491 #endif  /* LAST_ROUND_TABLES */

493 #define ii_rnd(p1, p2, p3, p4, round)   /* normal inverse round */ \
494         mov     ik_ref(round,0), p1; \
495         mov     ik_ref(round,1), p2; \
496         mov     ik_ref(round,2), p3; \
497         mov     ik_ref(round,3), p4; \
498  \
499         movzx   %al, %esi; \
500         movzx   %ah, %edi; \
501         shr     $16, %eax; \
502         xor     tab_0(%rsi), p1; \
503         xor     tab_1(%rdi), p2; \
504         movzx   %al, %esi; \
505         movzx   %ah, %edi; \
506         xor     tab_2(%rsi), p3; \
507         xor     tab_3(%rdi), p4; \
508  \
509         movzx   %bl, %esi; \
510         movzx   %bh, %edi; \
511         shr     $16, %ebx; \
512         xor     tab_0(%rsi), p2; \
513         xor     tab_1(%rdi), p3; \
514         movzx   %bl, %esi; \
515         movzx   %bh, %edi; \
516         xor     tab_2(%rsi), p4; \
517         xor     tab_3(%rdi), p1; \
518  \
519         movzx   %cl, %esi; \
520         movzx   %ch, %edi; \
521         shr     $16, %ecx; \
522         xor     tab_0(%rsi), p3; \
523         xor     tab_1(%rdi), p4; \
```

```
524         movzx   %cl, %esi; \
525         movzx   %ch, %edi; \
526         xor     tab_2(%rsi), p1; \
527         xor     tab_3(%rdi), p2; \
528  \
529         movzx   %dl, %esi; \
530         movzx   %dh, %edi; \
531         shr     $16, %edx; \
532         xor     tab_0(%rsi), p4; \
533         xor     tab_1(%rdi), p1; \
534         movzx   %dl, %esi; \
535         movzx   %dh, %edi; \
536         xor     tab_2(%rsi), p2; \
537         xor     tab_3(%rdi), p3; \
538  \
539         mov     p1, %eax; \
540         mov     p2, %ebx; \
541         mov     p3, %ecx; \
542         mov     p4, %edx

544 #ifdef  LAST_ROUND_TABLES

546 #define il_rnd(p1, p2, p3, p4, round)   /* last inverse round */ \
547         add     $2048, tptr; \
548         mov     ik_ref(round,0), p1; \
549         mov     ik_ref(round,1), p2; \
550         mov     ik_ref(round,2), p3; \
551         mov     ik_ref(round,3), p4; \
552  \
553         movzx   %al, %esi; \
554         movzx   %ah, %edi; \
555         shr     $16, %eax; \
556         xor     tab_0(%rsi), p1; \
557         xor     tab_1(%rdi), p2; \
558         movzx   %al, %esi; \
559         movzx   %ah, %edi; \
560         xor     tab_2(%rsi), p3; \
561         xor     tab_3(%rdi), p4; \
562  \
563         movzx   %bl, %esi; \
564         movzx   %bh, %edi; \
565         shr     $16, %ebx; \
566         xor     tab_0(%rsi), p2; \
567         xor     tab_1(%rdi), p3; \
568         movzx   %bl, %esi; \
569         movzx   %bh, %edi; \
570         xor     tab_2(%rsi), p4; \
571         xor     tab_3(%rdi), p1; \
572  \
573         movzx   %cl, %esi; \
574         movzx   %ch, %edi; \
575         shr     $16, %ecx; \
576         xor     tab_0(%rsi), p3; \
577         xor     tab_1(%rdi), p4; \
578         movzx   %cl, %esi; \
579         movzx   %ch, %edi; \
580         xor     tab_2(%rsi), p1; \
581         xor     tab_3(%rdi), p2; \
582  \
583         movzx   %dl, %esi; \
584         movzx   %dh, %edi; \
585         shr     $16, %edx; \
586         xor     tab_0(%rsi), p4; \
587         xor     tab_1(%rdi), p1; \
588         movzx   %dl, %esi; \
589         movzx   %dh, %edi; \
```



```
590         xor     tab_2(%rsi), p2; \
591         xor     tab_3(%rdi), p3

593 #else

595 #define il_rnd(p1, p2, p3, p4, round)   /* last inverse round */ \
596         mov     ik_ref(round,0), p1; \
597         mov     ik_ref(round,1), p2; \
598         mov     ik_ref(round,2), p3; \
599         mov     ik_ref(round,3), p4; \
600  \
601         movzx   %al, %esi; \
602         movzx   %ah, %edi; \
603         movzx   tab_i(%rsi), %esi; \
604         movzx   tab_i(%rdi), %edi; \
605         shr     $16, %eax; \
606         xor     %esi, p1; \
607         rol     $8, %edi; \
608         xor     %edi, p2; \
609         movzx   %al, %esi; \
610         movzx   %ah, %edi; \
611         movzx   tab_i(%rsi), %esi; \
612         movzx   tab_i(%rdi), %edi; \
613         rol     $16, %esi; \
614         rol     $24, %edi; \
615         xor     %esi, p3; \
616         xor     %edi, p4; \
617  \
618         movzx   %bl, %esi; \
619         movzx   %bh, %edi; \
620         movzx   tab_i(%rsi), %esi; \
621         movzx   tab_i(%rdi), %edi; \
622         shr     $16, %ebx; \
623         xor     %esi, p2; \
624         rol     $8, %edi; \
625         xor     %edi, p3; \
626         movzx   %bl, %esi; \
627         movzx   %bh, %edi; \
628         movzx   tab_i(%rsi), %esi; \
629         movzx   tab_i(%rdi), %edi; \
630         rol     $16, %esi; \
631         rol     $24, %edi; \
632         xor     %esi, p4; \
633         xor     %edi, p1; \
634  \
635         movzx   %cl, %esi; \
636         movzx   %ch, %edi; \
637         movzx   tab_i(%rsi), %esi; \
638         movzx   tab_i(%rdi), %edi; \
639         shr     $16, %ecx; \
640         xor     %esi, p3; \
641         rol     $8, %edi; \
642         xor     %edi, p4; \
643         movzx   %cl, %esi; \
644         movzx   %ch, %edi; \
645         movzx   tab_i(%rsi), %esi; \
646         movzx   tab_i(%rdi), %edi; \
647         rol     $16, %esi; \
648         rol     $24, %edi; \
649         xor     %esi, p1; \
650         xor     %edi, p2; \
651  \
652         movzx   %dl, %esi; \
653         movzx   %dh, %edi; \
654         movzx   tab_i(%rsi), %esi; \
655         movzx   tab_i(%rdi), %edi; \
```

```
656         shr     $16, %edx; \
657         xor     %esi, p4; \
658         rol     $8, %edi; \
659         xor     %edi, p1; \
660         movzx   %dl, %esi; \
661         movzx   %dh, %edi; \
662         movzx   tab_i(%rsi), %esi; \
663         movzx   tab_i(%rdi), %edi; \
664         rol     $16, %esi; \
665         rol     $24, %edi; \
666         xor     %esi, p2; \
667         xor     %edi, p3

669 #endif  /* LAST_ROUND_TABLES */
670         /* EXPORT DELETE END */

672 /*
673  * OpenSolaris OS:
674  * void aes_encrypt_impl(const aes_ks_t *ks, int Nr,
675  *      const uint32_t pt[4], uint32_t ct[4])/
676  *
677  * Original interface:
678  * int aes_encrypt(const unsigned char *in,
679  *      unsigned char *out, const aes_encrypt_ctx cx[1])/
680  */
681         .align  64
682 enc_tab:
683         enc_vals(u8)
684 #ifdef  LAST_ROUND_TABLES
685         / Last Round Tables:
686         enc_vals(w8)
687 #endif


690         ENTRY_NP(aes_encrypt_impl)
691         /* EXPORT DELETE START */
692 #ifdef  GLADMAN_INTERFACE
693         / Original interface
694         sub     $[4*8], %rsp    / gnu/linux/opensolaris binary interface
695         mov     %rsi, (%rsp)    / output pointer (P2)
696         mov     %rdx, %r8       / context (P3)

698         mov     %rbx, 1*8(%rsp) / P1: input pointer in rdi
699         mov     %rbp, 2*8(%rsp) / P2: output pointer in (rsp)
700         mov     %r12, 3*8(%rsp) / P3: context in r8
701         movzx   4*KS_LENGTH(kptr), %esi / Get byte key length * 16

703 #else
704         / OpenSolaris OS interface
705         sub     $[4*8], %rsp    / Make room on stack to save registers
706         mov     %rcx, (%rsp)    / Save output pointer (P4) on stack
707         mov     %rdi, %r8       / context (P1)
708         mov     %rdx, %rdi      / P3: save input pointer
709         shl     $4, %esi        / P2: esi byte key length * 16

711         mov     %rbx, 1*8(%rsp) / Save registers
712         mov     %rbp, 2*8(%rsp)
713         mov     %r12, 3*8(%rsp)
714         / P1: context in r8
715         / P2: byte key length * 16 in esi
716         / P3: input pointer in rdi
717         / P4: output pointer in (rsp)
718 #endif  /* GLADMAN_INTERFACE */

720         lea     enc_tab(%rip), tptr
721         sub     $fofs, kptr
```



```
723         / Load input block into registers
724         mov     (%rdi), %eax
725         mov     1*4(%rdi), %ebx
726         mov     2*4(%rdi), %ecx
727         mov     3*4(%rdi), %edx

729         xor     fofs(kptr), %eax
730         xor     fofs+4(kptr), %ebx
731         xor     fofs+8(kptr), %ecx
732         xor     fofs+12(kptr), %edx

734         lea     (kptr,%rsi), kptr
735         / Jump based on byte key length * 16:
736         cmp     $[10*16], %esi
737         je      3f
738         cmp     $[12*16], %esi
739         je      2f
740         cmp     $[14*16], %esi
741         je      1f
742         mov     $-1, %rax       / error
743         jmp     4f

745         / Perform normal forward rounds
746 1:      ff_rnd(%r9d, %r10d, %r11d, %r12d, 13)
747         ff_rnd(%r9d, %r10d, %r11d, %r12d, 12)
748 2:      ff_rnd(%r9d, %r10d, %r11d, %r12d, 11)
749         ff_rnd(%r9d, %r10d, %r11d, %r12d, 10)
750 3:      ff_rnd(%r9d, %r10d, %r11d, %r12d,  9)
751         ff_rnd(%r9d, %r10d, %r11d, %r12d,  8)
752         ff_rnd(%r9d, %r10d, %r11d, %r12d,  7)
753         ff_rnd(%r9d, %r10d, %r11d, %r12d,  6)
754         ff_rnd(%r9d, %r10d, %r11d, %r12d,  5)
755         ff_rnd(%r9d, %r10d, %r11d, %r12d,  4)
756         ff_rnd(%r9d, %r10d, %r11d, %r12d,  3)
757         ff_rnd(%r9d, %r10d, %r11d, %r12d,  2)
758         ff_rnd(%r9d, %r10d, %r11d, %r12d,  1)
759         fl_rnd(%r9d, %r10d, %r11d, %r12d,  0)

761         / Copy results
762         mov     (%rsp), %rbx
763         mov     %r9d, (%rbx)
764         mov     %r10d, 4(%rbx)
765         mov     %r11d, 8(%rbx)
766         mov     %r12d, 12(%rbx)
767         xor     %rax, %rax
768 4:      / Restore registers
769         mov     1*8(%rsp), %rbx
770         mov     2*8(%rsp), %rbp
771         mov     3*8(%rsp), %r12
772         add     $[4*8], %rsp
773         /* EXPORT DELETE END */
774         ret

776         SET_SIZE(aes_encrypt_impl)

778 /*
779  * OpenSolaris OS:
780  * void aes_decrypt_impl(const aes_ks_t *ks, int Nr,
781  *      const uint32_t pt[4], uint32_t ct[4])/
782  *
783  * Original interface:
784  * int aes_decrypt(const unsigned char *in,
785  *      unsigned char *out, const aes_encrypt_ctx cx[1])/
786  */
787         .align  64
```

```
dec_tab:
        dec_vals(v8)
#ifdef  LAST_ROUND_TABLES
        / Last Round Tables:
        dec_vals(w8)
#endif


        ENTRY_NP(aes_decrypt_impl)
        /* EXPORT DELETE START */
#ifdef  GLADMAN_INTERFACE
        / Original interface
        sub     $[4*8], %rsp    / gnu/linux/opensolaris binary interface
        mov     %rsi, (%rsp)    / output pointer (P2)
        mov     %rdx, %r8       / context (P3)

        mov     %rbx, 1*8(%rsp) / P1: input pointer in rdi
        mov     %rbp, 2*8(%rsp) / P2: output pointer in (rsp)
        mov     %r12, 3*8(%rsp) / P3: context in r8
        movzx   4*KS_LENGTH(kptr), %esi / Get byte key length * 16

#else
        / OpenSolaris OS interface
        sub     $[4*8], %rsp    / Make room on stack to save registers
        mov     %rcx, (%rsp)    / Save output pointer (P4) on stack
        mov     %rdi, %r8       / context (P1)
        mov     %rdx, %rdi      / P3: save input pointer
        shl     $4, %esi        / P2: esi byte key length * 16

        mov     %rbx, 1*8(%rsp) / Save registers
        mov     %rbp, 2*8(%rsp)
        mov     %r12, 3*8(%rsp)
        / P1: context in r8
        / P2: byte key length * 16 in esi
        / P3: input pointer in rdi
        / P4: output pointer in (rsp)
#endif  /* GLADMAN_INTERFACE */

        lea     dec_tab(%rip), tptr
        sub     $rofs, kptr

        / Load input block into registers
        mov     (%rdi), %eax
        mov     1*4(%rdi), %ebx
        mov     2*4(%rdi), %ecx
        mov     3*4(%rdi), %edx

#ifdef AES_REV_DKS
        mov     kptr, %rdi
        lea     (kptr,%rsi), kptr
#else
        lea     (kptr,%rsi), %rdi
#endif

        xor     rofs(%rdi), %eax
        xor     rofs+4(%rdi), %ebx
        xor     rofs+8(%rdi), %ecx
        xor     rofs+12(%rdi), %edx

        / Jump based on byte key length * 16:
        cmp     $[10*16], %esi
        je      3f
        cmp     $[12*16], %esi
        je      2f
        cmp     $[14*16], %esi
        je      1f
```



```
        mov     $-1, %rax       / error
        jmp     4f

        / Perform normal inverse rounds
1:      ii_rnd(%r9d, %r10d, %r11d, %r12d, 13)
        ii_rnd(%r9d, %r10d, %r11d, %r12d, 12)
2:      ii_rnd(%r9d, %r10d, %r11d, %r12d, 11)
        ii_rnd(%r9d, %r10d, %r11d, %r12d, 10)
3:      ii_rnd(%r9d, %r10d, %r11d, %r12d,  9)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  8)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  7)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  6)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  5)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  4)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  3)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  2)
        ii_rnd(%r9d, %r10d, %r11d, %r12d,  1)
        il_rnd(%r9d, %r10d, %r11d, %r12d,  0)

        / Copy results
        mov     (%rsp), %rbx
        mov     %r9d, (%rbx)
        mov     %r10d, 4(%rbx)
        mov     %r11d, 8(%rbx)
        mov     %r12d, 12(%rbx)
        xor     %rax, %rax
4:      / Restore registers
        mov     1*8(%rsp), %rbx
        mov     2*8(%rsp), %rbp
        mov     3*8(%rsp), %r12
        add     $[4*8], %rsp
        /* EXPORT DELETE END */
        ret

        SET_SIZE(aes_decrypt_impl)

#else
        /* LINTED */
        /* Nothing to be linted in this file--it's pure assembly source. */
#endif  /* !lint && !__lint */
```

```
**********************************************************
   15807 Fri Jun 13 16:44:44 2008
new/usr/src/common/crypto/aes/amd64/aeskey.c
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * ---------------------------------------------------------------------------
 * Copyright (c) 1998-2007, Brian Gladman, Worcester, UK. All rights reserved.
 *
 * LICENSE TERMS
 *
 * The free distribution and use of this software is allowed (with or without
 * changes) provided that:
 *
 *  1. source code distributions include the above copyright notice, this
 *       list of conditions and the following disclaimer;
 *
 *  2. binary distributions include the above copyright notice, this list
 *       of conditions and the following disclaimer in their documentation;
 *
 *  3. the name of the copyright holder is not used to endorse products
 *       built using this software without specific written permission.
 *
 * DISCLAIMER
 *
 * This software is provided 'as is' with no explicit or implied warranties
 * in respect of its properties, including, but not limited to, correctness
 * and/or fitness for purpose.
 * ---------------------------------------------------------------------------
 * Issue Date: 20/12/2007
 */

#pragma ident   "@(#)aeskey.c   1.1     08/05/21 SMI"

#include "aes_impl.h"
#include "aesopt.h"
#include "aestab.h"
#include "aestab2.h"

/*
 *      Initialise the key schedule from the user supplied key. The key
 *      length can be specified in bytes, with legal values of 16, 24
 *      and 32, or in bits, with legal values of 128, 192 and 256. These
 *      values correspond with Nk values of 4, 6 and 8 respectively.
 *
 *      The following macros implement a single cycle in the key
 *      schedule generation process. The number of cycles needed
 *      for each cx->n_col and nk value is:
 *
 *      nk =            4  5  6  7  8
 *      ------------------------------
 *      cx->n_col = 4   10  9  8  7  7
 *      cx->n_col = 5   14 11 10  9  9
 *      cx->n_col = 6   19 15 12 11 11
 *      cx->n_col = 7   21 19 16 13 14
 *      cx->n_col = 8   29 23 19 17 14
 */

/*
 * OpenSolaris changes
 * 1. Added header files aes_impl.h and aestab2.h
 * 2. Changed uint_8t and uint_32t to uint8_t and uint32_t
 * 3. Remove code under ifdef USE_VIA_ACE_IF_PRESENT (always undefined)
 * 4. Removed always-defined ifdefs FUNCS_IN_C, ENC_KEYING_IN_C,
 *      AES_128, AES_192, AES_256, AES_VAR defines
 * 5. Changed aes_encrypt_key* aes_decrypt_key* functions to "static void"
```



```
 * 6. Changed N_COLS to MAX_AES_NB
 * 7. Replaced functions aes_encrypt_key and aes_decrypt_key with
 *      OpenSolaris-compatible functions rijndael_key_setup_enc and
 *      rijndael_key_setup_dec
 * 8. cstyled code and removed lint warnings
 *
 */

#if defined(REDUCE_CODE_SIZE)
#define ls_box ls_sub
        uint32_t        ls_sub(const uint32_t t, const uint32_t n);
#define inv_mcol im_sub
        uint32_t        im_sub(const uint32_t x);
#ifdef ENC_KS_UNROLL
#undef ENC_KS_UNROLL
#endif
#ifdef DEC_KS_UNROLL
#undef DEC_KS_UNROLL
#endif
#endif  /* REDUCE_CODE_SIZE */


#define ke4(k, i) \
{       k[4 * (i) + 4] = ss[0] ^= ls_box(ss[3], 3) ^ t_use(r, c)[i]; \
        k[4 * (i) + 5] = ss[1] ^= ss[0]; \
        k[4 * (i) + 6] = ss[2] ^= ss[1]; \
        k[4 * (i) + 7] = ss[3] ^= ss[2]; \
}

static void
aes_encrypt_key128(const unsigned char *key, uint32_t rk[])
{
        uint32_t        ss[4];

        rk[0] = ss[0] = word_in(key, 0);
        rk[1] = ss[1] = word_in(key, 1);
        rk[2] = ss[2] = word_in(key, 2);
        rk[3] = ss[3] = word_in(key, 3);

#ifdef ENC_KS_UNROLL
        ke4(rk, 0);  ke4(rk, 1);
        ke4(rk, 2);  ke4(rk, 3);
        ke4(rk, 4);  ke4(rk, 5);
        ke4(rk, 6);  ke4(rk, 7);
        ke4(rk, 8);
#else
        {
                uint32_t        i;
                for (i = 0; i < 9; ++i)
                        ke4(rk, i);
        }
#endif  /* ENC_KS_UNROLL */
        ke4(rk, 9);
}


#define kef6(k, i) \
{       k[6 * (i) + 6] = ss[0] ^= ls_box(ss[5], 3) ^ t_use(r, c)[i]; \
        k[6 * (i) + 7] = ss[1] ^= ss[0]; \
        k[6 * (i) + 8] = ss[2] ^= ss[1]; \
        k[6 * (i) + 9] = ss[3] ^= ss[2]; \
}

#define ke6(k, i) \
{       kef6(k, i); \
        k[6 * (i) + 10] = ss[4] ^= ss[3]; \
```

```
128         k[6 * (i) + 11] = ss[5] ^= ss[4]; \
129 }

131 static void
132 aes_encrypt_key192(const unsigned char *key, uint32_t rk[])
133 {
134         uint32_t        ss[6];

136         rk[0] = ss[0] = word_in(key, 0);
137         rk[1] = ss[1] = word_in(key, 1);
138         rk[2] = ss[2] = word_in(key, 2);
139         rk[3] = ss[3] = word_in(key, 3);
140         rk[4] = ss[4] = word_in(key, 4);
141         rk[5] = ss[5] = word_in(key, 5);

143 #ifdef ENC_KS_UNROLL
144         ke6(rk, 0);  ke6(rk, 1);
145         ke6(rk, 2);  ke6(rk, 3);
146         ke6(rk, 4);  ke6(rk, 5);
147         ke6(rk, 6);
148 #else
149         {
150                 uint32_t        i;
151                 for (i = 0; i < 7; ++i)
152                         ke6(rk, i);
153         }
154 #endif  /* ENC_KS_UNROLL */
155         kef6(rk, 7);
156 }



160 #define kef8(k, i) \
161 {       k[8 * (i) + 8] = ss[0] ^= ls_box(ss[7], 3) ^ t_use(r, c)[i]; \
162         k[8 * (i) + 9] = ss[1] ^= ss[0]; \
163         k[8 * (i) + 10] = ss[2] ^= ss[1]; \
164         k[8 * (i) + 11] = ss[3] ^= ss[2]; \
165 }

167 #define ke8(k, i) \
168 {   kef8(k, i); \
169         k[8 * (i) + 12] = ss[4] ^= ls_box(ss[3], 0); \
170         k[8 * (i) + 13] = ss[5] ^= ss[4]; \
171         k[8 * (i) + 14] = ss[6] ^= ss[5]; \
172         k[8 * (i) + 15] = ss[7] ^= ss[6]; \
173 }

175 static void
176 aes_encrypt_key256(const unsigned char *key, uint32_t rk[])
177 {
178         uint32_t        ss[8];

180         rk[0] = ss[0] = word_in(key, 0);
181         rk[1] = ss[1] = word_in(key, 1);
182         rk[2] = ss[2] = word_in(key, 2);
183         rk[3] = ss[3] = word_in(key, 3);
184         rk[4] = ss[4] = word_in(key, 4);
185         rk[5] = ss[5] = word_in(key, 5);
186         rk[6] = ss[6] = word_in(key, 6);
187         rk[7] = ss[7] = word_in(key, 7);

189 #ifdef ENC_KS_UNROLL
190         ke8(rk, 0); ke8(rk, 1);
191         ke8(rk, 2); ke8(rk, 3);
192         ke8(rk, 4); ke8(rk, 5);
193 #else
```



```
194         {
195                 uint32_t        i;
196                 for (i = 0; i < 6; ++i)
197                         ke8(rk,  i);
198         }
199 #endif  /* ENC_KS_UNROLL */
200         kef8(rk, 6);
201 }


204 /*
205  * Expand the cipher key into the encryption key schedule.
206  *
207  * Return the number of rounds for the given cipher key size.
208  * The size of the key schedule depends on the number of rounds
209  * (which can be computed from the size of the key), i.e. 4 * (Nr + 1).
210  *
211  * Parameters:
212  * rk           AES key schedule 32-bit array to be initialized
213  * cipherKey    User key
214  * keyBits      AES key size (128, 192, or 256 bits)
215  */
216 int
217 rijndael_key_setup_enc(uint32_t rk[], const uint32_t cipherKey[], int keyBits)
218 {
219         switch (keyBits) {
220         case 128:
221                 aes_encrypt_key128((unsigned char *)&cipherKey[0], rk);
222                 return (10);
223         case 192:
224                 aes_encrypt_key192((unsigned char *)&cipherKey[0], rk);
225                 return (12);
226         case 256:
227                 aes_encrypt_key256((unsigned char *)&cipherKey[0], rk);
228                 return (14);
229         default: /* should never get here */
230                 break;
231         }

233         return (0);
234 }


237 /* this is used to store the decryption round keys  */
238 /* in forward or reverse order */

240 #ifdef AES_REV_DKS
241 #define v(n, i)  ((n) - (i) + 2 * ((i) & 3))
242 #else
243 #define v(n, i)  (i)
244 #endif

246 #if DEC_ROUND == NO_TABLES
247 #define ff(x)   (x)
248 #else
249 #define ff(x)   inv_mcol(x)
250 #if defined(dec_imvars)
251 #define d_vars  dec_imvars
252 #endif
253 #endif  /* FUNCS_IN_C & DEC_KEYING_IN_C */


256 #define k4e(k, i) \
257 {       k[v(40, (4 * (i)) + 4)] = ss[0] ^= ls_box(ss[3], 3) ^ t_use(r, c)[i]; \
258         k[v(40, (4 * (i)) + 5)] = ss[1] ^= ss[0]; \
259         k[v(40, (4 * (i)) + 6)] = ss[2] ^= ss[1]; \
```

```
260         k[v(40, (4 * (i)) + 7)] = ss[3] ^= ss[2]; \
261 }

263 #if 1

265 #define kdf4(k, i) \
266 {       ss[0] = ss[0] ^ ss[2] ^ ss[1] ^ ss[3]; \
267         ss[1] = ss[1] ^ ss[3]; \
268         ss[2] = ss[2] ^ ss[3]; \
269         ss[4] = ls_box(ss[(i + 3) % 4], 3) ^ t_use(r, c)[i]; \
270         ss[i % 4] ^= ss[4]; \
271         ss[4] ^= k[v(40, (4 * (i)))];   k[v(40, (4 * (i)) + 4)] = ff(ss[4]); \
272         ss[4] ^= k[v(40, (4 * (i)) + 1)]; k[v(40, (4 * (i)) + 5)] = ff(ss[4]); \
273         ss[4] ^= k[v(40, (4 * (i)) + 2)]; k[v(40, (4 * (i)) + 6)] = ff(ss[4]); \
274         ss[4] ^= k[v(40, (4 * (i)) + 3)]; k[v(40, (4 * (i)) + 7)] = ff(ss[4]); \
275 }

277 #define kd4(k, i) \
278 {       ss[4] = ls_box(ss[(i + 3) % 4], 3) ^ t_use(r, c)[i]; \
279         ss[i % 4] ^= ss[4]; ss[4] = ff(ss[4]); \
280         k[v(40, (4 * (i)) + 4)] = ss[4] ^= k[v(40, (4 * (i)))]; \
281         k[v(40, (4 * (i)) + 5)] = ss[4] ^= k[v(40, (4 * (i)) + 1)]; \
282         k[v(40, (4 * (i)) + 6)] = ss[4] ^= k[v(40, (4 * (i)) + 2)]; \
283         k[v(40, (4 * (i)) + 7)] = ss[4] ^= k[v(40, (4 * (i)) + 3)]; \
284 }

286 #define kdl4(k, i) \
287 {       ss[4] = ls_box(ss[(i + 3) % 4], 3) ^ t_use(r, c)[i]; \
288         ss[i % 4] ^= ss[4]; \
289         k[v(40, (4 * (i)) + 4)] = (ss[0] ^= ss[1]) ^ ss[2] ^ ss[3]; \
290         k[v(40, (4 * (i)) + 5)] = ss[1] ^ ss[3]; \
291         k[v(40, (4 * (i)) + 6)] = ss[0]; \
292         k[v(40, (4 * (i)) + 7)] = ss[1]; \
293 }

295 #else

297 #define kdf4(k, i) \
298 {       ss[0] ^= ls_box(ss[3], 3) ^ t_use(r, c)[i]; \
299         k[v(40, (4 * (i)) + 4)] = ff(ss[0]); \
300         ss[1] ^= ss[0]; k[v(40, (4 * (i)) + 5)] = ff(ss[1]); \
301         ss[2] ^= ss[1]; k[v(40, (4 * (i)) + 6)] = ff(ss[2]); \
302         ss[3] ^= ss[2]; k[v(40, (4 * (i)) + 7)] = ff(ss[3]); \
303 }

305 #define kd4(k, i) \
306 {       ss[4] = ls_box(ss[3], 3) ^ t_use(r, c)[i]; \
307         ss[0] ^= ss[4]; \
308         ss[4] = ff(ss[4]); \
309         k[v(40, (4 * (i)) + 4)] = ss[4] ^= k[v(40, (4 * (i)))]; \
310         ss[1] ^= ss[0]; \
311         k[v(40, (4 * (i)) + 5)] = ss[4] ^= k[v(40, (4 * (i)) + 1)]; \
312         ss[2] ^= ss[1]; \
313         k[v(40, (4 * (i)) + 6)] = ss[4] ^= k[v(40, (4 * (i)) + 2)]; \
314         ss[3] ^= ss[2]; \
315         k[v(40, (4 * (i)) + 7)] = ss[4] ^= k[v(40, (4 * (i)) + 3)]; \
316 }

318 #define kdl4(k, i) \
319 {       ss[0] ^= ls_box(ss[3], 3) ^ t_use(r, c)[i]; \
320         k[v(40, (4 * (i)) + 4)] = ss[0]; \
321         ss[1] ^= ss[0]; k[v(40, (4 * (i)) + 5)] = ss[1]; \
322         ss[2] ^= ss[1]; k[v(40, (4 * (i)) + 6)] = ss[2]; \
323         ss[3] ^= ss[2]; k[v(40, (4 * (i)) + 7)] = ss[3]; \
324 }
```



```
326 #endif

328 static void
329 aes_decrypt_key128(const unsigned char *key, uint32_t rk[])
330 {
331         uint32_t        ss[5];
332 #if defined(d_vars)
333         d_vars;
334 #endif
335         rk[v(40, (0))] = ss[0] = word_in(key, 0);
336         rk[v(40, (1))] = ss[1] = word_in(key, 1);
337         rk[v(40, (2))] = ss[2] = word_in(key, 2);
338         rk[v(40, (3))] = ss[3] = word_in(key, 3);

340 #ifdef DEC_KS_UNROLL
341         kdf4(rk, 0); kd4(rk, 1);
342         kd4(rk, 2);  kd4(rk, 3);
343         kd4(rk, 4);  kd4(rk, 5);
344         kd4(rk, 6);  kd4(rk, 7);
345         kd4(rk, 8);  kdl4(rk, 9);
346 #else
347         {
348                 uint32_t        i;
349                 for (i = 0; i < 10; ++i)
350                         k4e(rk, i);
351 #if !(DEC_ROUND == NO_TABLES)
352                 for (i = MAX_AES_NB; i < 10 * MAX_AES_NB; ++i)
353                         rk[i] = inv_mcol(rk[i]);
354 #endif
355         }
356 #endif  /* DEC_KS_UNROLL */
357 }



361 #define k6ef(k, i) \
362 {       k[v(48, (6 * (i)) + 6)] = ss[0] ^= ls_box(ss[5], 3) ^ t_use(r, c)[i]; \
363         k[v(48, (6 * (i)) + 7)] = ss[1] ^= ss[0]; \
364         k[v(48, (6 * (i)) + 8)] = ss[2] ^= ss[1]; \
365         k[v(48, (6 * (i)) + 9)] = ss[3] ^= ss[2]; \
366 }

368 #define k6e(k, i) \
369 {       k6ef(k, i); \
370         k[v(48, (6 * (i)) + 10)] = ss[4] ^= ss[3]; \
371         k[v(48, (6 * (i)) + 11)] = ss[5] ^= ss[4]; \
372 }

374 #define kdf6(k, i) \
375 {       ss[0] ^= ls_box(ss[5], 3) ^ t_use(r, c)[i]; \
376         k[v(48, (6 * (i)) + 6)] = ff(ss[0]); \
377         ss[1] ^= ss[0]; k[v(48, (6 * (i)) + 7)] = ff(ss[1]); \
378         ss[2] ^= ss[1]; k[v(48, (6 * (i)) + 8)] = ff(ss[2]); \
379         ss[3] ^= ss[2]; k[v(48, (6 * (i)) + 9)] = ff(ss[3]); \
380         ss[4] ^= ss[3]; k[v(48, (6 * (i)) + 10)] = ff(ss[4]); \
381         ss[5] ^= ss[4]; k[v(48, (6 * (i)) + 11)] = ff(ss[5]); \
382 }

384 #define kd6(k, i) \
385 {       ss[6] = ls_box(ss[5], 3) ^ t_use(r, c)[i]; \
386         ss[0] ^= ss[6]; ss[6] = ff(ss[6]); \
387         k[v(48, (6 * (i)) + 6)] = ss[6] ^= k[v(48, (6 * (i)))]; \
388         ss[1] ^= ss[0]; \
389         k[v(48, (6 * (i)) + 7)] = ss[6] ^= k[v(48, (6 * (i)) + 1)]; \
390         ss[2] ^= ss[1]; \
391         k[v(48, (6 * (i)) + 8)] = ss[6] ^= k[v(48, (6 * (i)) + 2)]; \
```

```
392         ss[3] ^= ss[2]; \
393         k[v(48, (6 * (i)) + 9)] = ss[6] ^= k[v(48, (6 * (i)) + 3)]; \
394         ss[4] ^= ss[3]; \
395         k[v(48, (6 * (i)) + 10)] = ss[6] ^= k[v(48, (6 * (i)) + 4)]; \
396         ss[5] ^= ss[4]; \
397         k[v(48, (6 * (i)) + 11)] = ss[6] ^= k[v(48, (6 * (i)) + 5)]; \
398 }

400 #define kdl6(k, i) \
401 {       ss[0] ^= ls_box(ss[5], 3) ^ t_use(r, c)[i]; \
402         k[v(48, (6 * (i)) + 6)] = ss[0]; \
403         ss[1] ^= ss[0]; k[v(48, (6 * (i)) + 7)] = ss[1]; \
404         ss[2] ^= ss[1]; k[v(48, (6 * (i)) + 8)] = ss[2]; \
405         ss[3] ^= ss[2]; k[v(48, (6 * (i)) + 9)] = ss[3]; \
406 }

408 static void
409 aes_decrypt_key192(const unsigned char *key, uint32_t rk[])
410 {
411         uint32_t        ss[7];
412 #if defined(d_vars)
413         d_vars;
414 #endif
415         rk[v(48, (0))] = ss[0] = word_in(key, 0);
416         rk[v(48, (1))] = ss[1] = word_in(key, 1);
417         rk[v(48, (2))] = ss[2] = word_in(key, 2);
418         rk[v(48, (3))] = ss[3] = word_in(key, 3);

420 #ifdef DEC_KS_UNROLL
421         ss[4] = word_in(key, 4);
422         rk[v(48, (4))] = ff(ss[4]);
423         ss[5] = word_in(key, 5);
424         rk[v(48, (5))] = ff(ss[5]);
425         kdf6(rk, 0); kd6(rk, 1);
426         kd6(rk, 2);  kd6(rk, 3);
427         kd6(rk, 4);  kd6(rk, 5);
428         kd6(rk, 6);  kdl6(rk, 7);
429 #else
430         rk[v(48, (4))] = ss[4] = word_in(key, 4);
431         rk[v(48, (5))] = ss[5] = word_in(key, 5);
432         {
433                 uint32_t        i;

435                 for (i = 0; i < 7; ++i)
436                         k6e(rk, i);
437                 k6ef(rk, 7);
438 #if !(DEC_ROUND == NO_TABLES)
439                 for (i = MAX_AES_NB; i < 12 * MAX_AES_NB; ++i)
440                         rk[i] = inv_mcol(rk[i]);
441 #endif
442         }
443 #endif
444 }



448 #define k8ef(k, i) \
449 {       k[v(56, (8 * (i)) + 8)] = ss[0] ^= ls_box(ss[7], 3) ^ t_use(r, c)[i]; \
450         k[v(56, (8 * (i)) + 9)] = ss[1] ^= ss[0]; \
451         k[v(56, (8 * (i)) + 10)] = ss[2] ^= ss[1]; \
452         k[v(56, (8 * (i)) + 11)] = ss[3] ^= ss[2]; \
453 }

455 #define k8e(k, i) \
456 {       k8ef(k, i); \
457         k[v(56, (8 * (i)) + 12)] = ss[4] ^= ls_box(ss[3], 0); \
```



```
458         k[v(56, (8 * (i)) + 13)] = ss[5] ^= ss[4]; \
459         k[v(56, (8 * (i)) + 14)] = ss[6] ^= ss[5]; \
460         k[v(56, (8 * (i)) + 15)] = ss[7] ^= ss[6]; \
461 }

463 #define kdf8(k, i) \
464 {       ss[0] ^= ls_box(ss[7], 3) ^ t_use(r, c)[i]; \
465         k[v(56, (8 * (i)) + 8)] = ff(ss[0]); \
466         ss[1] ^= ss[0]; k[v(56, (8 * (i)) + 9)] = ff(ss[1]); \
467         ss[2] ^= ss[1]; k[v(56, (8 * (i)) + 10)] = ff(ss[2]); \
468         ss[3] ^= ss[2]; k[v(56, (8 * (i)) + 11)] = ff(ss[3]); \
469         ss[4] ^= ls_box(ss[3], 0); k[v(56, (8 * (i)) + 12)] = ff(ss[4]); \
470         ss[5] ^= ss[4]; k[v(56, (8 * (i)) + 13)] = ff(ss[5]); \
471         ss[6] ^= ss[5]; k[v(56, (8 * (i)) + 14)] = ff(ss[6]); \
472         ss[7] ^= ss[6]; k[v(56, (8 * (i)) + 15)] = ff(ss[7]); \
473 }

475 #define kd8(k, i) \
476 {       ss[8] = ls_box(ss[7], 3) ^ t_use(r, c)[i]; \
477         ss[0] ^= ss[8]; \
478         ss[8] = ff(ss[8]); \
479         k[v(56, (8 * (i)) + 8)] = ss[8] ^= k[v(56, (8 * (i)))]; \
480         ss[1] ^= ss[0]; \
481         k[v(56, (8 * (i)) + 9)] = ss[8] ^= k[v(56, (8 * (i)) + 1)]; \
482         ss[2] ^= ss[1]; \
483         k[v(56, (8 * (i)) + 10)] = ss[8] ^= k[v(56, (8 * (i)) + 2)]; \
484         ss[3] ^= ss[2]; \
485         k[v(56, (8 * (i)) + 11)] = ss[8] ^= k[v(56, (8 * (i)) + 3)]; \
486         ss[8] = ls_box(ss[3], 0); \
487         ss[4] ^= ss[8]; \
488         ss[8] = ff(ss[8]); \
489         k[v(56, (8 * (i)) + 12)] = ss[8] ^= k[v(56, (8 * (i)) + 4)]; \
490         ss[5] ^= ss[4]; \
491         k[v(56, (8 * (i)) + 13)] = ss[8] ^= k[v(56, (8 * (i)) + 5)]; \
492         ss[6] ^= ss[5]; \
493         k[v(56, (8 * (i)) + 14)] = ss[8] ^= k[v(56, (8 * (i)) + 6)]; \
494         ss[7] ^= ss[6]; \
495         k[v(56, (8 * (i)) + 15)] = ss[8] ^= k[v(56, (8 * (i)) + 7)]; \
496 }

498 #define kdl8(k, i) \
499 {       ss[0] ^= ls_box(ss[7], 3) ^ t_use(r, c)[i]; \
500         k[v(56, (8 * (i)) + 8)] = ss[0]; \
501         ss[1] ^= ss[0]; k[v(56, (8 * (i)) + 9)] = ss[1]; \
502         ss[2] ^= ss[1]; k[v(56, (8 * (i)) + 10)] = ss[2]; \
503         ss[3] ^= ss[2]; k[v(56, (8 * (i)) + 11)] = ss[3]; \
504 }

506 static void
507 aes_decrypt_key256(const unsigned char *key, uint32_t rk[])
508 {
509         uint32_t        ss[9];
510 #if defined(d_vars)
511         d_vars;
512 #endif
513         rk[v(56, (0))] = ss[0] = word_in(key, 0);
514         rk[v(56, (1))] = ss[1] = word_in(key, 1);
515         rk[v(56, (2))] = ss[2] = word_in(key, 2);
516         rk[v(56, (3))] = ss[3] = word_in(key, 3);

518 #ifdef DEC_KS_UNROLL
519         ss[4] = word_in(key, 4);
520         rk[v(56, (4))] = ff(ss[4]);
521         ss[5] = word_in(key, 5);
522         rk[v(56, (5))] = ff(ss[5]);
523         ss[6] = word_in(key, 6);
```

```
524         rk[v(56, (6))] = ff(ss[6]);
525         ss[7] = word_in(key, 7);
526         rk[v(56, (7))] = ff(ss[7]);
527         kdf8(rk, 0); kd8(rk, 1);
528         kd8(rk, 2);  kd8(rk, 3);
529         kd8(rk, 4);  kd8(rk, 5);
530         kdl8(rk, 6);
531 #else
532         rk[v(56, (4))] = ss[4] = word_in(key, 4);
533         rk[v(56, (5))] = ss[5] = word_in(key, 5);
534         rk[v(56, (6))] = ss[6] = word_in(key, 6);
535         rk[v(56, (7))] = ss[7] = word_in(key, 7);
536         {
537                 uint32_t        i;

539                 for (i = 0; i < 6; ++i)
540                         k8e(rk,  i);
541                 k8ef(rk,  6);
542 #if !(DEC_ROUND == NO_TABLES)
543                 for (i = MAX_AES_NB; i < 14 * MAX_AES_NB; ++i)
544                         rk[i] = inv_mcol(rk[i]);
545 #endif
546         }
547 #endif  /* DEC_KS_UNROLL */
548 }


551 /*
552  * Expand the cipher key into the decryption key schedule.
553  *
554  * Return the number of rounds for the given cipher key size.
555  * The size of the key schedule depends on the number of rounds
556  * (which can be computed from the size of the key), i.e. 4 * (Nr + 1).
557  *
558  * Parameters:
559  * rk           AES key schedule 32-bit array to be initialized
560  * cipherKey    User key
561  * keyBits      AES key size (128, 192, or 256 bits)
562  */
563 int
564 rijndael_key_setup_dec(uint32_t rk[], const uint32_t cipherKey[], int keyBits)
565 {
566         switch (keyBits) {
567         case 128:
568                 aes_decrypt_key128((unsigned char *)&cipherKey[0], rk);
569                 return (10);
570         case 192:
571                 aes_decrypt_key192((unsigned char *)&cipherKey[0], rk);
572                 return (12);
573         case 256:
574                 aes_decrypt_key256((unsigned char *)&cipherKey[0], rk);
575                 return (14);
576         default: /* should never get here */
577                 break;
578         }

580         return (0);
581 }
```

```
**********************************************************
   24356 Fri Jun 13 16:44:45 2008
new/usr/src/common/crypto/aes/amd64/aesopt.h
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * ---------------------------------------------------------------------------
 * Copyright (c) 1998-2007, Brian Gladman, Worcester, UK. All rights reserved.
 *
 * LICENSE TERMS
 *
 * The free distribution and use of this software is allowed (with or without
 * changes) provided that:
 *
 *  1. source code distributions include the above copyright notice, this
 *      list of conditions and the following disclaimer;
 *
 *  2. binary distributions include the above copyright notice, this list
 *      of conditions and the following disclaimer in their documentation;
 *
 *  3. the name of the copyright holder is not used to endorse products
 *      built using this software without specific written permission.
 *
 * DISCLAIMER
 *
 * This software is provided 'as is' with no explicit or implied warranties
 * in respect of its properties, including, but not limited to, correctness
 * and/or fitness for purpose.
 * ---------------------------------------------------------------------------
 * Issue Date: 20/12/2007
 *
 * This file contains the compilation options for AES (Rijndael) and code
 * that is common across encryption, key scheduling and table generation.
 *
 * OPERATION
 *
 * These source code files implement the AES algorithm Rijndael designed by
 * Joan Daemen and Vincent Rijmen. This version is designed for the standard
 * block size of 16 bytes and for key sizes of 128, 192 and 256 bits (16, 24
 * and 32 bytes).
 *
 * This version is designed for flexibility and speed using operations on
 * 32-bit words rather than operations on bytes.  It can be compiled with
 * either big or little endian internal byte order but is faster when the
 * native byte order for the processor is used.
 *
 * THE CIPHER INTERFACE
 *
 * The cipher interface is implemented as an array of bytes in which lower
 * AES bit sequence indexes map to higher numeric significance within bytes.
 */

/*
 * OpenSolaris changes
 * 1. Added __cplusplus and _AESTAB_H header guards
 * 2. Added header files sys/types.h and aes_impl.h
 * 3. Added defines for AES_ENCRYPT, AES_DECRYPT, AES_REV_DKS, and ASM_AMD64_C
 * 4. Moved defines for IS_BIG_ENDIAN, IS_LITTLE_ENDIAN, PLATFORM_BYTE_ORDER
 *    from brg_endian.h
 * 5. Undefined VIA_ACE_POSSIBLE and ASSUME_VIA_ACE_PRESENT
 * 6. Changed uint_8t and uint_32t to uint8_t and uint32_t
 * 7. cstyled and hdrchk code
 *
 */

#ifndef _AESOPT_H
```



```
#define _AESOPT_H

#pragma ident   "@(#)aesopt.h   1.1     08/05/21 SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <aes_impl.h>

/*  SUPPORT FEATURES */
#define AES_ENCRYPT /* if support for encryption is needed */
#define AES_DECRYPT /* if support for decryption is needed */

/*  PLATFORM-SPECIFIC FEATURES */
#define IS_BIG_ENDIAN           4321 /* byte 0 is most significant (mc68k) */
#define IS_LITTLE_ENDIAN        1234 /* byte 0 is least significant (i386) */
#define PLATFORM_BYTE_ORDER     IS_LITTLE_ENDIAN
#define AES_REV_DKS /* define to reverse decryption key schedule */


/*
 *  CONFIGURATION - THE USE OF DEFINES
 *      Later in this section there are a number of defines that control the
 *      operation of the code.  In each section, the purpose of each define is
 *      explained so that the relevant form can be included or excluded by
 *      setting either 1's or 0's respectively on the branches of the related
 *      #if clauses.  The following local defines should not be changed.
 */

#define ENCRYPTION_IN_C 1
#define DECRYPTION_IN_C 2
#define ENC_KEYING_IN_C 4
#define DEC_KEYING_IN_C 8

#define NO_TABLES       0
#define ONE_TABLE       1
#define FOUR_TABLES     4
#define NONE            0
#define PARTIAL         1
#define FULL            2

/*  --- START OF USER CONFIGURED OPTIONS --- */

/*
 *  1. BYTE ORDER WITHIN 32 BIT WORDS
 *
 *      The fundamental data processing units in Rijndael are 8-bit bytes. The
 *      input, output and key input are all enumerated arrays of bytes in which
 *      bytes are numbered starting at zero and increasing to one less than the
 *      number of bytes in the array in question. This enumeration is only used
 *      for naming bytes and does not imply any adjacency or order relationship
 *      from one byte to another. When these inputs and outputs are considered
 *      as bit sequences, bits 8*n to 8*n+7 of the bit sequence are mapped to
 *      byte[n] with bit 8n+i in the sequence mapped to bit 7-i within the byte.
 *      In this implementation bits are numbered from 0 to 7 starting at the
 *      numerically least significant end of each byte.  Bit n represents 2^n.
 *
 *      However, Rijndael can be implemented more efficiently using 32-bit
 *      words by packing bytes into words so that bytes 4*n to 4*n+3 are placed
 *      into word[n]. While in principle these bytes can be assembled into words
 *      in any positions, this implementation only supports the two formats in
 *      which bytes in adjacent positions within words also have adjacent byte
 *      numbers. This order is called big-endian if the lowest numbered bytes
 *      in words have the highest numeric significance and little-endian if the
```

```
 *      opposite applies.
 *
 *      This code can work in either order irrespective of the order used by the
 *      machine on which it runs. Normally the internal byte order will be set
 *      to the order of the processor on which the code is to be run but this
 *      define  can be used to reverse this in special situations
 *
 *      WARNING: Assembler code versions rely on PLATFORM_BYTE_ORDER being set.
 *      This define will hence be redefined later (in section 4) if necessary
 */

#if 1
#define ALGORITHM_BYTE_ORDER PLATFORM_BYTE_ORDER
#elif 0
#define ALGORITHM_BYTE_ORDER IS_LITTLE_ENDIAN
#elif 0
#define ALGORITHM_BYTE_ORDER IS_BIG_ENDIAN
#else
#error The algorithm byte order is not defined
#endif

/*  2. VIA ACE SUPPORT */

#if defined(__GNUC__) && defined(__i386__) || \
        defined(_WIN32) && defined(_M_IX86) && \
        !(defined(_WIN64) || defined(_WIN32_WCE) || \
        defined(_MSC_VER) && (_MSC_VER <= 800))
#define VIA_ACE_POSSIBLE
#endif

/*
 *  Define this option if support for the VIA ACE is required. This uses
 *  inline assembler instructions and is only implemented for the Microsoft,
 *  Intel and GCC compilers.  If VIA ACE is known to be present, then defining
 *  ASSUME_VIA_ACE_PRESENT will remove the ordinary encryption/decryption
 *  code.  If USE_VIA_ACE_IF_PRESENT is defined then VIA ACE will be used if
 *  it is detected (both present and enabled) but the normal AES code will
 *  also be present.
 *
 *  When VIA ACE is to be used, all AES encryption contexts MUST be 16 byte
 *  aligned; other input/output buffers do not need to be 16 byte aligned
 *  but there are very large performance gains if this can be arranged.
 *  VIA ACE also requires the decryption key schedule to be in reverse
 *  order (which later checks below ensure).
 */

/*  VIA ACE is not used here for OpenSolaris: */
#undef  VIA_ACE_POSSIBLE
#undef  ASSUME_VIA_ACE_PRESENT

#if 0 && defined(VIA_ACE_POSSIBLE) && !defined(USE_VIA_ACE_IF_PRESENT)
#define USE_VIA_ACE_IF_PRESENT
#endif

#if 0 && defined(VIA_ACE_POSSIBLE) && !defined(ASSUME_VIA_ACE_PRESENT)
#define ASSUME_VIA_ACE_PRESENT
#endif


/*
 *  3. ASSEMBLER SUPPORT
 *
 *      This define (which can be on the command line) enables the use of the
 *      assembler code routines for encryption, decryption and key scheduling
 *      as follows:
 *
```



```
 *      ASM_X86_V1C uses the assembler (aes_x86_v1.asm) with large tables for
 *              encryption and decryption and but with key scheduling in C
 *      ASM_X86_V2  uses assembler (aes_x86_v2.asm) with compressed tables for
 *              encryption, decryption and key scheduling
 *      ASM_X86_V2C uses assembler (aes_x86_v2.asm) with compressed tables for
 *              encryption and decryption and but with key scheduling in C
 *      ASM_AMD64_C uses assembler (aes_amd64.asm) with compressed tables for
 *              encryption and decryption and but with key scheduling in C
 *
 *      Change one 'if 0' below to 'if 1' to select the version or define
 *      as a compilation option.
 */

#if 0 && !defined(ASM_X86_V1C)
#define ASM_X86_V1C
#elif 0 && !defined(ASM_X86_V2)
#define ASM_X86_V2
#elif 0 && !defined(ASM_X86_V2C)
#define ASM_X86_V2C
#elif 1 && !defined(ASM_AMD64_C)
#define ASM_AMD64_C
#endif

#if (defined(ASM_X86_V1C) || defined(ASM_X86_V2) || defined(ASM_X86_V2C)) && \
        !defined(_M_IX86) || defined(ASM_AMD64_C) && !defined(_M_X64) && \
        !defined(__amd64)
#error Assembler code is only available for x86 and AMD64 systems
#endif

/*
 *  4. FAST INPUT/OUTPUT OPERATIONS.
 *
 *      On some machines it is possible to improve speed by transferring the
 *      bytes in the input and output arrays to and from the internal 32-bit
 *      variables by addressing these arrays as if they are arrays of 32-bit
 *      words.  On some machines this will always be possible but there may
 *      be a large performance penalty if the byte arrays are not aligned on
 *      the normal word boundaries. On other machines this technique will
 *      lead to memory access errors when such 32-bit word accesses are not
 *      properly aligned. The option SAFE_IO avoids such problems but will
 *      often be slower on those machines that support misaligned access
 *      (especially so if care is taken to align the input  and output byte
 *      arrays on 32-bit word boundaries). If SAFE_IO is not defined it is
 *      assumed that access to byte arrays as if they are arrays of 32-bit
 *      words will not cause problems when such accesses are misaligned.
 */
#if 1 && !defined(_MSC_VER)
#define SAFE_IO
#endif

/*
 *  5. LOOP UNROLLING
 *
 *      The code for encryption and decryption cycles through a number of rounds
 *      that can be implemented either in a loop or by expanding the code into a
 *      long sequence of instructions, the latter producing a larger program but
 *      one that will often be much faster. The latter is called loop unrolling.
 *      There are also potential speed advantages in expanding two iterations in
 *      a loop with half the number of iterations, which is called partial loop
 *      unrolling.  The following options allow partial or full loop unrolling
 *      to be set independently for encryption and decryption
 */
#if 1
#define ENC_UNROLL  FULL
#elif 0
#define ENC_UNROLL  PARTIAL
```

```
#else
#define ENC_UNROLL  NONE
#endif

#if 1
#define DEC_UNROLL  FULL
#elif 0
#define DEC_UNROLL  PARTIAL
#else
#define DEC_UNROLL  NONE
#endif

#if 1
#define ENC_KS_UNROLL
#endif

#if 1
#define DEC_KS_UNROLL
#endif

/*
 *  6. FAST FINITE FIELD OPERATIONS
 *
 *      If this section is included, tables are used to provide faster finite
 *      field arithmetic.  This has no effect if FIXED_TABLES is defined.
 */
#if 1
#define FF_TABLES
#endif

/*
 *  7. INTERNAL STATE VARIABLE FORMAT
 *
 *      The internal state of Rijndael is stored in a number of local 32-bit
 *      word variables which can be defined either as an array or as individual
 *      names variables. Include this section if you want to store these local
 *      variables in arrays. Otherwise individual local variables will be used.
 */
#if 1
#define ARRAYS
#endif

/*
 *  8. FIXED OR DYNAMIC TABLES
 *
 *      When this section is included the tables used by the code are compiled
 *      statically into the binary file.  Otherwise the subroutine aes_init()
 *      must be called to compute them before the code is first used.
 */
#if 1 && !(defined(_MSC_VER) && (_MSC_VER <= 800))
#define FIXED_TABLES
#endif

/*
 *  9. MASKING OR CASTING FROM LONGER VALUES TO BYTES
 *
 *      In some systems it is better to mask longer values to extract bytes
 *      rather than using a cast. This option allows this choice.
 */
#if 0
#define to_byte(x)  ((uint8_t)(x))
#else
#define to_byte(x)  ((x) & 0xff)
#endif

/*
```



```
 *  10. TABLE ALIGNMENT
 *
 *      On some systems speed will be improved by aligning the AES large lookup
 *      tables on particular boundaries. This define should be set to a power of
 *      two giving the desired alignment. It can be left undefined if alignment
 *      is not needed.  This option is specific to the Micrsoft VC++ compiler -
 *      it seems to sometimes cause trouble for the VC++ version 6 compiler.
 */

#if 1 && defined(_MSC_VER) && (_MSC_VER >= 1300)
#define TABLE_ALIGN 32
#endif

/*
 *  11.  REDUCE CODE AND TABLE SIZE
 *
 *      This replaces some expanded macros with function calls if AES_ASM_V2 or
 *      AES_ASM_V2C are defined
 */

#if 1 && (defined(ASM_X86_V2) || defined(ASM_X86_V2C))
#define REDUCE_CODE_SIZE
#endif

/*
 *  12. TABLE OPTIONS
 *
 *      This cipher proceeds by repeating in a number of cycles known as rounds
 *      which are implemented by a round function which is optionally be speeded
 *      up using tables.  The basic tables are 256 32-bit words, with either
 *      one or four tables being required for each round function depending on
 *      how much speed is required. Encryption and decryption round functions
 *      are different and the last encryption and decryption round functions are
 *      different again making four different round functions in all.
 *
 *      This means that:
 *      1. Normal encryption and decryption rounds can each use either 0, 1
 *              or 4 tables and table spaces of 0, 1024 or 4096 bytes each.
 *      2. The last encryption and decryption rounds can also use either 0, 1
 *              or 4 tables and table spaces of 0, 1024 or 4096 bytes each.
 *
 *      Include or exclude the appropriate definitions below to set the number
 *      of tables used by this implementation.
 */

#if 1   /* set tables for the normal encryption round */
#define ENC_ROUND   FOUR_TABLES
#elif 0
#define ENC_ROUND   ONE_TABLE
#else
#define ENC_ROUND   NO_TABLES
#endif

#if 1   /* set tables for the last encryption round */
#define LAST_ENC_ROUND  FOUR_TABLES
#elif 0
#define LAST_ENC_ROUND  ONE_TABLE
#else
#define LAST_ENC_ROUND  NO_TABLES
#endif

#if 1   /* set tables for the normal decryption round */
#define DEC_ROUND   FOUR_TABLES
#elif 0
#define DEC_ROUND   ONE_TABLE
#else
```

```
#define DEC_ROUND   NO_TABLES
#endif

#if 1   /* set tables for the last decryption round */
#define LAST_DEC_ROUND  FOUR_TABLES
#elif 0
#define LAST_DEC_ROUND  ONE_TABLE
#else
#define LAST_DEC_ROUND  NO_TABLES
#endif

/*
 *  The decryption key schedule can be speeded up with tables in the same
 *      way that the round functions can.  Include or exclude the following
 *      defines to set this requirement.
 */
#if 1
#define KEY_SCHED   FOUR_TABLES
#elif 0
#define KEY_SCHED   ONE_TABLE
#else
#define KEY_SCHED   NO_TABLES
#endif

/*  ---- END OF USER CONFIGURED OPTIONS ---- */

/* VIA ACE support is only available for VC++ and GCC */

#if !defined(_MSC_VER) && !defined(__GNUC__)
#if defined(ASSUME_VIA_ACE_PRESENT)
#undef ASSUME_VIA_ACE_PRESENT
#endif
#if defined(USE_VIA_ACE_IF_PRESENT)
#undef USE_VIA_ACE_IF_PRESENT
#endif
#endif

#if defined(ASSUME_VIA_ACE_PRESENT) && !defined(USE_VIA_ACE_IF_PRESENT)
#define USE_VIA_ACE_IF_PRESENT
#endif

#if defined(USE_VIA_ACE_IF_PRESENT) && !defined(AES_REV_DKS)
#define AES_REV_DKS
#endif

/* Assembler support requires the use of platform byte order */

#if (defined(ASM_X86_V1C) || defined(ASM_X86_V2C) || defined(ASM_AMD64_C)) && \
        (ALGORITHM_BYTE_ORDER != PLATFORM_BYTE_ORDER)
#undef  ALGORITHM_BYTE_ORDER
#define ALGORITHM_BYTE_ORDER PLATFORM_BYTE_ORDER
#endif

/*
 * In this implementation the columns of the state array are each held in
 *      32-bit words. The state array can be held in various ways: in an array
 *      of words, in a number of individual word variables or in a number of
 *      processor registers. The following define maps a variable name x and
 *      a column number c to the way the state array variable is to be held.
 *      The first define below maps the state into an array x[c] whereas the
 *      second form maps the state into a number of individual variables x0,
 *      x1, etc.  Another form could map individual state columns to machine
 *      register names.
 */

#if defined(ARRAYS)
```



```
#define s(x, c) x[c]
#else
#define s(x, c) x##c
#endif

/*
 *  This implementation provides subroutines for encryption, decryption
 *      and for setting the three key lengths (separately) for encryption
 *      and decryption. Since not all functions are needed, masks are set
 *      up here to determine which will be implemented in C
 */

#if !defined(AES_ENCRYPT)
#define EFUNCS_IN_C   0
#elif defined(ASSUME_VIA_ACE_PRESENT) || defined(ASM_X86_V1C) || \
        defined(ASM_X86_V2C) || defined(ASM_AMD64_C)
#define EFUNCS_IN_C   ENC_KEYING_IN_C
#elif !defined(ASM_X86_V2)
#define EFUNCS_IN_C   (ENCRYPTION_IN_C | ENC_KEYING_IN_C)
#else
#define EFUNCS_IN_C   0
#endif

#if !defined(AES_DECRYPT)
#define DFUNCS_IN_C   0
#elif defined(ASSUME_VIA_ACE_PRESENT) || defined(ASM_X86_V1C) || \
        defined(ASM_X86_V2C) || defined(ASM_AMD64_C)
#define DFUNCS_IN_C   DEC_KEYING_IN_C
#elif !defined(ASM_X86_V2)
#define DFUNCS_IN_C   (DECRYPTION_IN_C | DEC_KEYING_IN_C)
#else
#define DFUNCS_IN_C   0
#endif

#define FUNCS_IN_C  (EFUNCS_IN_C | DFUNCS_IN_C)

/* END OF CONFIGURATION OPTIONS */

/* Disable or report errors on some combinations of options */

#if ENC_ROUND == NO_TABLES && LAST_ENC_ROUND != NO_TABLES
#undef  LAST_ENC_ROUND
#define LAST_ENC_ROUND  NO_TABLES
#elif ENC_ROUND == ONE_TABLE && LAST_ENC_ROUND == FOUR_TABLES
#undef  LAST_ENC_ROUND
#define LAST_ENC_ROUND  ONE_TABLE
#endif

#if ENC_ROUND == NO_TABLES && ENC_UNROLL != NONE
#undef  ENC_UNROLL
#define ENC_UNROLL  NONE
#endif

#if DEC_ROUND == NO_TABLES && LAST_DEC_ROUND != NO_TABLES
#undef  LAST_DEC_ROUND
#define LAST_DEC_ROUND  NO_TABLES
#elif DEC_ROUND == ONE_TABLE && LAST_DEC_ROUND == FOUR_TABLES
#undef  LAST_DEC_ROUND
#define LAST_DEC_ROUND  ONE_TABLE
#endif

#if DEC_ROUND == NO_TABLES && DEC_UNROLL != NONE
#undef  DEC_UNROLL
#define DEC_UNROLL  NONE
#endif
```

```
524 #if defined(bswap32)
525 #define aes_sw32        bswap32
526 #elif defined(bswap_32)
527 #define aes_sw32        bswap_32
528 #else
529 #define brot(x, n)   (((uint32_t)(x) <<  n) | ((uint32_t)(x) >> (32 - n)))
530 #define aes_sw32(x) ((brot((x), 8) & 0x00ff00ff) | (brot((x), 24) & 0xff00ff00))
531 #endif

533 /*
534  *  upr(x, n):  rotates bytes within words by n positions, moving bytes to
535  *              higher index positions with wrap around into low positions
536  *      ups(x, n):  moves bytes by n positions to higher index positions in
537  *              words but without wrap around
538  *      bval(x, n): extracts a byte from a word
539  *
540  *      WARNING:   The definitions given here are intended only for use with
541  *              unsigned variables and with shift counts that are compile
542  *              time constants
543  */

545 #if (ALGORITHM_BYTE_ORDER == IS_LITTLE_ENDIAN)
546 #define upr(x, n)       (((uint32_t)(x) << (8 * (n))) | \
547                          ((uint32_t)(x) >> (32 - 8 * (n))))
548 #define ups(x, n)       ((uint32_t)(x) << (8 * (n)))
549 #define bval(x, n)      to_byte((x) >> (8 * (n)))
550 #define bytes2word(b0, b1, b2, b3)  \
551                 (((uint32_t)(b3) << 24) | ((uint32_t)(b2) << 16) | \
552                 ((uint32_t)(b1) << 8) | (b0))
553 #endif

555 #if (ALGORITHM_BYTE_ORDER == IS_BIG_ENDIAN)
556 #define upr(x, n)       (((uint32_t)(x) >> (8 * (n))) | \
557                          ((uint32_t)(x) << (32 - 8 * (n))))
558 #define ups(x, n)       ((uint32_t)(x) >> (8 * (n)))
559 #define bval(x, n)      to_byte((x) >> (24 - 8 * (n)))
560 #define bytes2word(b0, b1, b2, b3)  \
561                 (((uint32_t)(b0) << 24) | ((uint32_t)(b1) << 16) | \
562                 ((uint32_t)(b2) << 8) | (b3))
563 #endif

565 #if defined(SAFE_IO)
566 #define word_in(x, c)   bytes2word(((const uint8_t *)(x) + 4 * c)[0], \
567                                    ((const uint8_t *)(x) + 4 * c)[1], \
568                                    ((const uint8_t *)(x) + 4 * c)[2], \
569                                    ((const uint8_t *)(x) + 4 * c)[3])
570 #define word_out(x, c, v) { ((uint8_t *)(x) + 4 * c)[0] = bval(v, 0); \
571                         ((uint8_t *)(x) + 4 * c)[1] = bval(v, 1); \
572                         ((uint8_t *)(x) + 4 * c)[2] = bval(v, 2); \
573                         ((uint8_t *)(x) + 4 * c)[3] = bval(v, 3); }
574 #elif (ALGORITHM_BYTE_ORDER == PLATFORM_BYTE_ORDER)
575 #define word_in(x, c)   (*((uint32_t *)(x) + (c)))
576 #define word_out(x, c, v) (*((uint32_t *)(x) + (c)) = (v))
577 #else
578 #define word_in(x, c)   aes_sw32(*((uint32_t *)(x) + (c)))
579 #define word_out(x, c, v) (*((uint32_t *)(x) + (c)) = aes_sw32(v))
580 #endif

582 /* the finite field modular polynomial and elements */

584 #define WPOLY   0x011b
585 #define BPOLY   0x1b

587 /* multiply four bytes in GF(2^8) by 'x' {02} in parallel */

589 #define m1  0x80808080
```



```
590 #define m2  0x7f7f7f7f
591 #define gf_mulx(x)  ((((x) & m2) << 1) ^ ((((x) & m1) >> 7) * BPOLY))

593 /*
594  * The following defines provide alternative definitions of gf_mulx that might
595  * give improved performance if a fast 32-bit multiply is not available. Note
596  * that a temporary variable u needs to be defined where gf_mulx is used.
597  *
598  * #define      gf_mulx(x) (u = (x) & m1, u |= (u >> 1), ((x) & m2) << 1) ^ \
599  *                         ((u >> 3) | (u >> 6))
600  * #define      m4  (0x01010101 * BPOLY)
601  * #define      gf_mulx(x) (u = (x) & m1, ((x) & m2) << 1) ^ ((u - (u >> 7)) \
602  *                         & m4)
603  */

605 /* Work out which tables are needed for the different options   */

607 #if defined(ASM_X86_V1C)
608 #if defined(ENC_ROUND)
609 #undef  ENC_ROUND
610 #endif
611 #define ENC_ROUND   FOUR_TABLES
612 #if defined(LAST_ENC_ROUND)
613 #undef  LAST_ENC_ROUND
614 #endif
615 #define LAST_ENC_ROUND  FOUR_TABLES
616 #if defined(DEC_ROUND)
617 #undef  DEC_ROUND
618 #endif
619 #define DEC_ROUND   FOUR_TABLES
620 #if defined(LAST_DEC_ROUND)
621 #undef  LAST_DEC_ROUND
622 #endif
623 #define LAST_DEC_ROUND  FOUR_TABLES
624 #if defined(KEY_SCHED)
625 #undef  KEY_SCHED
626 #define KEY_SCHED   FOUR_TABLES
627 #endif
628 #endif

630 #if (FUNCS_IN_C & ENCRYPTION_IN_C) || defined(ASM_X86_V1C)
631 #if ENC_ROUND == ONE_TABLE
632 #define FT1_SET
633 #elif ENC_ROUND == FOUR_TABLES
634 #define FT4_SET
635 #else
636 #define SBX_SET
637 #endif
638 #if LAST_ENC_ROUND == ONE_TABLE
639 #define FL1_SET
640 #elif LAST_ENC_ROUND == FOUR_TABLES
641 #define FL4_SET
642 #elif !defined(SBX_SET)
643 #define SBX_SET
644 #endif
645 #endif

647 #if (FUNCS_IN_C & DECRYPTION_IN_C) || defined(ASM_X86_V1C)
648 #if DEC_ROUND == ONE_TABLE
649 #define IT1_SET
650 #elif DEC_ROUND == FOUR_TABLES
651 #define IT4_SET
652 #else
653 #define ISB_SET
654 #endif
655 #if LAST_DEC_ROUND == ONE_TABLE
```

```
#define IL1_SET
#elif LAST_DEC_ROUND == FOUR_TABLES
#define IL4_SET
#elif !defined(ISB_SET)
#define ISB_SET
#endif
#endif


#if !(defined(REDUCE_CODE_SIZE) && (defined(ASM_X86_V2) || \
        defined(ASM_X86_V2C)))
#if ((FUNCS_IN_C & ENC_KEYING_IN_C) || (FUNCS_IN_C & DEC_KEYING_IN_C))
#if KEY_SCHED == ONE_TABLE
#if !defined(FL1_SET) && !defined(FL4_SET)
#define LS1_SET
#endif
#elif KEY_SCHED == FOUR_TABLES
#if !defined(FL4_SET)
#define LS4_SET
#endif
#elif !defined(SBX_SET)
#define SBX_SET
#endif
#endif
#if (FUNCS_IN_C & DEC_KEYING_IN_C)
#if KEY_SCHED == ONE_TABLE
#define IM1_SET
#elif KEY_SCHED == FOUR_TABLES
#define IM4_SET
#elif !defined(SBX_SET)
#define SBX_SET
#endif
#endif
#endif

/* generic definitions of Rijndael macros that use tables */

#define no_table(x, box, vf, rf, c) bytes2word(\
        box[bval(vf(x, 0, c), rf(0, c))], \
        box[bval(vf(x, 1, c), rf(1, c))], \
        box[bval(vf(x, 2, c), rf(2, c))], \
        box[bval(vf(x, 3, c), rf(3, c))])

#define one_table(x, op, tab, vf, rf, c) \
        (tab[bval(vf(x, 0, c), rf(0, c))] \
        ^ op(tab[bval(vf(x, 1, c), rf(1, c))], 1) \
        ^ op(tab[bval(vf(x, 2, c), rf(2, c))], 2) \
        ^ op(tab[bval(vf(x, 3, c), rf(3, c))], 3))

#define four_tables(x, tab, vf, rf, c) \
        (tab[0][bval(vf(x, 0, c), rf(0, c))] \
        ^ tab[1][bval(vf(x, 1, c), rf(1, c))] \
        ^ tab[2][bval(vf(x, 2, c), rf(2, c))] \
        ^ tab[3][bval(vf(x, 3, c), rf(3, c))])

#define vf1(x, r, c)    (x)
#define rf1(r, c)       (r)
#define rf2(r, c)       ((8+r-c)&3)

/*
 * Perform forward and inverse column mix operation on four bytes in long word
 * x in parallel. NOTE: x must be a simple variable, NOT an expression in
 * these macros.
 */

#if !(defined(REDUCE_CODE_SIZE) && (defined(ASM_X86_V2) || \
```



```
        defined(ASM_X86_V2C)))

#if defined(FM4_SET)    /* not currently used */
#define fwd_mcol(x)     four_tables(x, t_use(f, m), vf1, rf1, 0)
#elif defined(FM1_SET)  /* not currently used */
#define fwd_mcol(x)     one_table(x, upr, t_use(f, m), vf1, rf1, 0)
#else
#define dec_fmvars      uint32_t g2
#define fwd_mcol(x)     (g2 = gf_mulx(x), g2 ^ upr((x) ^ g2, 3) ^ \
                                upr((x), 2) ^ upr((x), 1))
#endif

#if defined(IM4_SET)
#define inv_mcol(x)     four_tables(x, t_use(i, m), vf1, rf1, 0)
#elif defined(IM1_SET)
#define inv_mcol(x)     one_table(x, upr, t_use(i, m), vf1, rf1, 0)
#else
#define dec_imvars      uint32_t g2, g4, g9
#define inv_mcol(x)     (g2 = gf_mulx(x), g4 = gf_mulx(g2), g9 = \
                                (x) ^ gf_mulx(g4), g4 ^= g9, \
                                (x) ^ g2 ^ g4 ^ upr(g2 ^ g9, 3) ^ \
                                upr(g4, 2) ^ upr(g9, 1))
#endif

#if defined(FL4_SET)
#define ls_box(x, c)    four_tables(x, t_use(f, l), vf1, rf2, c)
#elif defined(LS4_SET)
#define ls_box(x, c)    four_tables(x, t_use(l, s), vf1, rf2, c)
#elif defined(FL1_SET)
#define ls_box(x, c)    one_table(x, upr, t_use(f, l), vf1, rf2, c)
#elif defined(LS1_SET)
#define ls_box(x, c)    one_table(x, upr, t_use(l, s), vf1, rf2, c)
#else
#define ls_box(x, c)    no_table(x, t_use(s, box), vf1, rf2, c)
#endif

#endif

#if defined(ASM_X86_V1C) && defined(AES_DECRYPT) && !defined(ISB_SET)
#define ISB_SET
#endif

#ifdef  __cplusplus
}
#endif

#endif  /* _AESOPT_H */
```

```
**********************************************************
    5141 Fri Jun 13 16:44:46 2008
new/usr/src/common/crypto/aes/amd64/aestab.h
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * ---------------------------------------------------------------------------
 * Copyright (c) 1998-2007, Brian Gladman, Worcester, UK. All rights reserved.
 *
 * LICENSE TERMS
 *
 * The free distribution and use of this software is allowed (with or without
 * changes) provided that:
 *
 *  1. source code distributions include the above copyright notice, this
 *     list of conditions and the following disclaimer;
 *
 *  2. binary distributions include the above copyright notice, this list
 *     of conditions and the following disclaimer in their documentation;
 *
 *  3. the name of the copyright holder is not used to endorse products
 *     built using this software without specific written permission.
 *
 * DISCLAIMER
 *
 * This software is provided 'as is' with no explicit or implied warranties
 * in respect of its properties, including, but not limited to, correctness
 * and/or fitness for purpose.
 * ---------------------------------------------------------------------------
 * Issue Date: 20/12/2007
 *
 * This file contains the code for declaring the tables needed to implement
 * AES. The file aesopt.h is assumed to be included before this header file.
 * If there are no global variables, the definitions here can be used to put
 * the AES tables in a structure so that a pointer can then be added to the
 * AES context to pass them to the AES routines that need them.   If this
 * facility is used, the calling program has to ensure that this pointer is
 * managed appropriately.  In particular, the value of the t_dec(in, it) item
 * in the table structure must be set to zero in order to ensure that the
 * tables are initialised. In practice the three code sequences in aeskey.c
 * that control the calls to aes_init() and the aes_init() routine itself will
 * have to be changed for a specific implementation. If global variables are
 * available it will generally be preferable to use them with the precomputed
 * FIXED_TABLES option that uses static global tables.
 *
 * The following defines can be used to control the way the tables
 * are defined, initialised and used in embedded environments that
 * require special features for these purposes
 *
 *    the 't_dec' construction is used to declare fixed table arrays
 *    the 't_set' construction is used to set fixed table values
 *    the 't_use' construction is used to access fixed table values
 *
 *    256 byte tables:
 *
 *        t_xxx(s, box)    => forward S box
 *        t_xxx(i, box)    => inverse S box
 *
 *    256 32-bit word OR 4 x 256 32-bit word tables:
 *
 *        t_xxx(f, n)      => forward normal round
 *        t_xxx(f, l)      => forward last round
 *        t_xxx(i, n)      => inverse normal round
 *        t_xxx(i, l)      => inverse last round
 *        t_xxx(l, s)      => key schedule table
 *        t_xxx(i, m)      => key schedule table
```



```
 *
 *    Other variables and tables:
 *
 *        t_xxx(r, c)      => the rcon table
 */

/*
 * OpenSolaris OS modifications
 *
 * 1. Added __cplusplus and _AESTAB_H header guards
 * 2. Added header file sys/types.h
 * 3. Remove code defined for _MSC_VER
 * 4. Changed all variables to "static const"
 * 5. Changed uint_8t and uint_32t to uint8_t and uint32_t
 * 6. Cstyled and hdrchk code
 */

#ifndef _AESTAB_H
#define _AESTAB_H

#pragma ident   "@(#)aestab.h   1.1     08/05/21 SMI"

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>

#define t_dec(m, n) t_##m##n
#define t_set(m, n) t_##m##n
#define t_use(m, n) t_##m##n

#if defined(DO_TABLES) && defined(FIXED_TABLES)
#define d_1(t, n, b, e)          static const t n[256]    =   b(e)
#define d_4(t, n, b, e, f, g, h) static const t n[4][256] = \
                                         {b(e), b(f), b(g), b(h)}
static const uint32_t t_dec(r, c)[RC_LENGTH] = rc_data(w0);
#else
#define d_1(t, n, b, e)                  static const t n[256]
#define d_4(t, n, b, e, f, g, h)         static const t n[4][256]
static const uint32_t t_dec(r, c)[RC_LENGTH];
#endif

#if defined(SBX_SET)
        d_1(uint8_t, t_dec(s, box), sb_data, h0);
#endif
#if defined(ISB_SET)
        d_1(uint8_t, t_dec(i, box), isb_data, h0);
#endif

#if defined(FT1_SET)
        d_1(uint32_t, t_dec(f, n), sb_data, u0);
#endif
#if defined(FT4_SET)
        d_4(uint32_t, t_dec(f, n), sb_data, u0, u1, u2, u3);
#endif

#if defined(FL1_SET)
        d_1(uint32_t, t_dec(f, l), sb_data, w0);
#endif
#if defined(FL4_SET)
        d_4(uint32_t, t_dec(f, l), sb_data, w0, w1, w2, w3);
#endif

#if defined(IT1_SET)
        d_1(uint32_t, t_dec(i, n), isb_data, v0);
```

```
#endif
#if defined(IT4_SET)
        d_4(uint32_t, t_dec(i, n), isb_data, v0, v1, v2, v3);
#endif

#if defined(IL1_SET)
        d_1(uint32_t, t_dec(i, l), isb_data, w0);
#endif
#if defined(IL4_SET)
        d_4(uint32_t, t_dec(i, l), isb_data, w0, w1, w2, w3);
#endif

#if defined(LS1_SET)
#if defined(FL1_SET)
#undef  LS1_SET
#else
        d_1(uint32_t, t_dec(l, s), sb_data, w0);
#endif
#endif

#if defined(LS4_SET)
#if defined(FL4_SET)
#undef  LS4_SET
#else
        d_4(uint32_t, t_dec(l, s), sb_data, w0, w1, w2, w3);
#endif
#endif

#if defined(IM1_SET)
        d_1(uint32_t, t_dec(i, m), mm_data, v0);
#endif
#if defined(IM4_SET)
        d_4(uint32_t, t_dec(i, m), mm_data, v0, v1, v2, v3);
#endif

#ifdef  __cplusplus
}
#endif

#endif  /* _AESTAB_H */
```

```
**********************************************************
   26886 Fri Jun 13 16:44:47 2008
new/usr/src/common/crypto/aes/amd64/aestab2.h
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _AESTAB2_H
#define _AESTAB2_H

#pragma ident   "@(#)aestab2.h  1.1     08/05/21 SMI"

#ifdef __cplusplus
extern "C" {
#endif

/*
 * To create this file for OpenSolaris:
 * 1. Compile and run tablegen.c, from aes-src-04-03-08.zip,
 *      after defining ASM_AMD64_C
 * 2. mv aestab2.c aestab2.h
 * 3. Add __cplusplus and _AESTAB2_H header guards
 * 3. Add #include <aes_impl.h>
 * 4. Change "uint_32t" to "uint32_t"
 * 5. Change all variables to "static const"
 * 6. Cstyle and hdrchk this file
 */

#include <aes_impl.h>

static const uint32_t t_rc[RC_LENGTH] =
{
        0x00000001, 0x00000002, 0x00000004, 0x00000008,
        0x00000010, 0x00000020, 0x00000040, 0x00000080,
        0x0000001b, 0x00000036
};

static const uint32_t t_ls[4][256] =
{
        {
        0x00000063, 0x0000007c, 0x00000077, 0x0000007b,
        0x000000f2, 0x0000006b, 0x0000006f, 0x000000c5,
        0x00000030, 0x00000001, 0x00000067, 0x0000002b,
```



```
        0x000000fe, 0x000000d7, 0x000000ab, 0x00000076,
        0x000000ca, 0x00000082, 0x000000c9, 0x0000007d,
        0x000000fa, 0x00000059, 0x00000047, 0x000000f0,
        0x000000ad, 0x000000d4, 0x000000a2, 0x000000af,
        0x0000009c, 0x000000a4, 0x00000072, 0x000000c0,
        0x000000b7, 0x000000fd, 0x00000093, 0x00000026,
        0x00000036, 0x0000003f, 0x000000f7, 0x000000cc,
        0x00000034, 0x000000a5, 0x000000e5, 0x000000f1,
        0x00000071, 0x000000d8, 0x00000031, 0x00000015,
        0x00000004, 0x000000c7, 0x00000023, 0x000000c3,
        0x00000018, 0x00000096, 0x00000005, 0x0000009a,
        0x00000007, 0x00000012, 0x00000080, 0x000000e2,
        0x000000eb, 0x00000027, 0x000000b2, 0x00000075,
        0x00000009, 0x00000083, 0x0000002c, 0x0000001a,
        0x0000001b, 0x0000006e, 0x0000005a, 0x000000a0,
        0x00000052, 0x0000003b, 0x000000d6, 0x000000b3,
        0x00000029, 0x000000e3, 0x0000002f, 0x00000084,
        0x00000053, 0x000000d1, 0x00000000, 0x000000ed,
        0x00000020, 0x000000fc, 0x000000b1, 0x0000005b,
        0x0000006a, 0x000000cb, 0x000000be, 0x00000039,
        0x0000004a, 0x0000004c, 0x00000058, 0x000000cf,
        0x000000d0, 0x000000ef, 0x000000aa, 0x000000fb,
        0x00000043, 0x0000004d, 0x00000033, 0x00000085,
        0x00000045, 0x000000f9, 0x00000002, 0x0000007f,
        0x00000050, 0x0000003c, 0x0000009f, 0x000000a8,
        0x00000051, 0x000000a3, 0x00000040, 0x0000008f,
        0x00000092, 0x0000009d, 0x00000038, 0x000000f5,
        0x000000bc, 0x000000b6, 0x000000da, 0x00000021,
        0x00000010, 0x000000ff, 0x000000f3, 0x000000d2,
        0x000000cd, 0x0000000c, 0x00000013, 0x000000ec,
        0x0000005f, 0x00000097, 0x00000044, 0x00000017,
        0x000000c4, 0x000000a7, 0x0000007e, 0x0000003d,
        0x00000064, 0x0000005d, 0x00000019, 0x00000073,
        0x00000060, 0x00000081, 0x0000004f, 0x000000dc,
        0x00000022, 0x0000002a, 0x00000090, 0x00000088,
        0x00000046, 0x000000ee, 0x000000b8, 0x00000014,
        0x000000de, 0x0000005e, 0x0000000b, 0x000000db,
        0x000000e0, 0x00000032, 0x0000003a, 0x0000000a,
        0x00000049, 0x00000006, 0x00000024, 0x0000005c,
        0x000000c2, 0x000000d3, 0x000000ac, 0x00000062,
        0x00000091, 0x00000095, 0x000000e4, 0x00000079,
        0x000000e7, 0x000000c8, 0x00000037, 0x0000006d,
        0x0000008d, 0x000000d5, 0x0000004e, 0x000000a9,
        0x0000006c, 0x00000056, 0x000000f4, 0x000000ea,
        0x00000065, 0x0000007a, 0x000000ae, 0x00000008,
        0x000000ba, 0x00000078, 0x00000025, 0x0000002e,
        0x0000001c, 0x000000a6, 0x000000b4, 0x000000c6,
        0x000000e8, 0x000000dd, 0x00000074, 0x0000001f,
        0x0000004b, 0x000000bd, 0x0000008b, 0x0000008a,
        0x00000070, 0x0000003e, 0x000000b5, 0x00000066,
        0x00000048, 0x00000003, 0x000000f6, 0x0000000e,
        0x00000061, 0x00000035, 0x00000057, 0x000000b9,
        0x00000086, 0x000000c1, 0x0000001d, 0x0000009e,
        0x000000e1, 0x000000f8, 0x00000098, 0x00000011,
        0x00000069, 0x000000d9, 0x0000008e, 0x00000094,
        0x0000009b, 0x0000001e, 0x00000087, 0x000000e9,
        0x000000ce, 0x00000055, 0x00000028, 0x000000df,
        0x0000008c, 0x000000a1, 0x00000089, 0x0000000d,
        0x000000bf, 0x000000e6, 0x00000042, 0x00000068,
        0x00000041, 0x00000099, 0x0000002d, 0x0000000f,
        0x000000b0, 0x00000054, 0x000000bb, 0x00000016
        },
        {
        0x00006300, 0x00007c00, 0x00007700, 0x00007b00,
        0x0000f200, 0x00006b00, 0x00006f00, 0x0000c500,
        0x00003000, 0x00000100, 0x00006700, 0x00002b00,
```

```
128         0x0000fe00, 0x0000d700, 0x0000ab00, 0x00007600,
129         0x0000ca00, 0x00008200, 0x0000c900, 0x00007d00,
130         0x0000fa00, 0x00005900, 0x00004700, 0x0000f000,
131         0x0000ad00, 0x0000d400, 0x0000a200, 0x0000af00,
132         0x00009c00, 0x0000a400, 0x00007200, 0x0000c000,
133         0x0000b700, 0x0000fd00, 0x00009300, 0x00002600,
134         0x00003600, 0x00003f00, 0x0000f700, 0x0000cc00,
135         0x00003400, 0x0000a500, 0x0000e500, 0x0000f100,
136         0x00007100, 0x0000d800, 0x00003100, 0x00001500,
137         0x00000400, 0x0000c700, 0x00002300, 0x0000c300,
138         0x00001800, 0x00009600, 0x00000500, 0x00009a00,
139         0x00000700, 0x00001200, 0x00008000, 0x0000e200,
140         0x0000eb00, 0x00002700, 0x0000b200, 0x00007500,
141         0x00000900, 0x00008300, 0x00002c00, 0x00001a00,
142         0x00001b00, 0x00006e00, 0x00005a00, 0x0000a000,
143         0x00005200, 0x00003b00, 0x0000d600, 0x0000b300,
144         0x00002900, 0x0000e300, 0x00002f00, 0x00008400,
145         0x00005300, 0x0000d100, 0x00000000, 0x0000ed00,
146         0x00002000, 0x0000fc00, 0x0000b100, 0x00005b00,
147         0x00006a00, 0x0000cb00, 0x0000be00, 0x00003900,
148         0x00004a00, 0x00004c00, 0x00005800, 0x0000cf00,
149         0x0000d000, 0x0000ef00, 0x0000aa00, 0x0000fb00,
150         0x00004300, 0x00004d00, 0x00003300, 0x00008500,
151         0x00004500, 0x0000f900, 0x00000200, 0x00007f00,
152         0x00005000, 0x00003c00, 0x00009f00, 0x0000a800,
153         0x00005100, 0x0000a300, 0x00004000, 0x00008f00,
154         0x00009200, 0x00009d00, 0x00003800, 0x0000f500,
155         0x0000bc00, 0x0000b600, 0x0000da00, 0x00002100,
156         0x00001000, 0x0000ff00, 0x0000f300, 0x0000d200,
157         0x0000cd00, 0x00000c00, 0x00001300, 0x0000ec00,
158         0x00005f00, 0x00009700, 0x00004400, 0x00001700,
159         0x0000c400, 0x0000a700, 0x00007e00, 0x00003d00,
160         0x00006400, 0x00005d00, 0x00001900, 0x00007300,
161         0x00006000, 0x00008100, 0x00004f00, 0x0000dc00,
162         0x00002200, 0x00002a00, 0x00009000, 0x00008800,
163         0x00004600, 0x0000ee00, 0x0000b800, 0x00001400,
164         0x0000de00, 0x00005e00, 0x00000b00, 0x0000db00,
165         0x0000e000, 0x00003200, 0x00003a00, 0x00000a00,
166         0x00004900, 0x00000600, 0x00002400, 0x00005c00,
167         0x0000c200, 0x0000d300, 0x0000ac00, 0x00006200,
168         0x00009100, 0x00009500, 0x0000e400, 0x00007900,
169         0x0000e700, 0x0000c800, 0x00003700, 0x00006d00,
170         0x00008d00, 0x0000d500, 0x00004e00, 0x0000a900,
171         0x00006c00, 0x00005600, 0x0000f400, 0x0000ea00,
172         0x00006500, 0x00007a00, 0x0000ae00, 0x00000800,
173         0x0000ba00, 0x00007800, 0x00002500, 0x00002e00,
174         0x00001c00, 0x0000a600, 0x0000b400, 0x0000c600,
175         0x0000e800, 0x0000dd00, 0x00007400, 0x00001f00,
176         0x00004b00, 0x0000bd00, 0x00008b00, 0x00008a00,
177         0x00007000, 0x00003e00, 0x0000b500, 0x00006600,
178         0x00004800, 0x00000300, 0x0000f600, 0x00000e00,
179         0x00006100, 0x00003500, 0x00005700, 0x0000b900,
180         0x00008600, 0x0000c100, 0x00001d00, 0x00009e00,
181         0x0000e100, 0x0000f800, 0x00009800, 0x00001100,
182         0x00006900, 0x0000d900, 0x00008e00, 0x00009400,
183         0x00009b00, 0x00001e00, 0x00008700, 0x0000e900,
184         0x0000ce00, 0x00005500, 0x00002800, 0x0000df00,
185         0x00008c00, 0x0000a100, 0x00008900, 0x00000d00,
186         0x0000bf00, 0x0000e600, 0x00004200, 0x00006800,
187         0x00004100, 0x00009900, 0x00002d00, 0x00000f00,
188         0x0000b000, 0x00005400, 0x0000bb00, 0x00001600
189         },
190         {
191         0x00630000, 0x007c0000, 0x00770000, 0x007b0000,
192         0x00f20000, 0x006b0000, 0x006f0000, 0x00c50000,
193         0x00300000, 0x00010000, 0x00670000, 0x002b0000,
```



```
194         0x00fe0000, 0x00d70000, 0x00ab0000, 0x00760000,
195         0x00ca0000, 0x00820000, 0x00c90000, 0x007d0000,
196         0x00fa0000, 0x00590000, 0x00470000, 0x00f00000,
197         0x00ad0000, 0x00d40000, 0x00a20000, 0x00af0000,
198         0x009c0000, 0x00a40000, 0x00720000, 0x00c00000,
199         0x00b70000, 0x00fd0000, 0x00930000, 0x00260000,
200         0x00360000, 0x003f0000, 0x00f70000, 0x00cc0000,
201         0x00340000, 0x00a50000, 0x00e50000, 0x00f10000,
202         0x00710000, 0x00d80000, 0x00310000, 0x00150000,
203         0x00040000, 0x00c70000, 0x00230000, 0x00c30000,
204         0x00180000, 0x00960000, 0x00050000, 0x009a0000,
205         0x00070000, 0x00120000, 0x00800000, 0x00e20000,
206         0x00eb0000, 0x00270000, 0x00b20000, 0x00750000,
207         0x00090000, 0x00830000, 0x002c0000, 0x001a0000,
208         0x001b0000, 0x006e0000, 0x005a0000, 0x00a00000,
209         0x00520000, 0x003b0000, 0x00d60000, 0x00b30000,
210         0x00290000, 0x00e30000, 0x002f0000, 0x00840000,
211         0x00530000, 0x00d10000, 0x00000000, 0x00ed0000,
212         0x00200000, 0x00fc0000, 0x00b10000, 0x005b0000,
213         0x006a0000, 0x00cb0000, 0x00be0000, 0x00390000,
214         0x004a0000, 0x004c0000, 0x00580000, 0x00cf0000,
215         0x00d00000, 0x00ef0000, 0x00aa0000, 0x00fb0000,
216         0x00430000, 0x004d0000, 0x00330000, 0x00850000,
217         0x00450000, 0x00f90000, 0x00020000, 0x007f0000,
218         0x00500000, 0x003c0000, 0x009f0000, 0x00a80000,
219         0x00510000, 0x00a30000, 0x00400000, 0x008f0000,
220         0x00920000, 0x009d0000, 0x00380000, 0x00f50000,
221         0x00bc0000, 0x00b60000, 0x00da0000, 0x00210000,
222         0x00100000, 0x00ff0000, 0x00f30000, 0x00d20000,
223         0x00cd0000, 0x000c0000, 0x00130000, 0x00ec0000,
224         0x005f0000, 0x00970000, 0x00440000, 0x00170000,
225         0x00c40000, 0x00a70000, 0x007e0000, 0x003d0000,
226         0x00640000, 0x005d0000, 0x00190000, 0x00730000,
227         0x00600000, 0x00810000, 0x004f0000, 0x00dc0000,
228         0x00220000, 0x002a0000, 0x00900000, 0x00880000,
229         0x00460000, 0x00ee0000, 0x00b80000, 0x00140000,
230         0x00de0000, 0x005e0000, 0x000b0000, 0x00db0000,
231         0x00e00000, 0x00320000, 0x003a0000, 0x000a0000,
232         0x00490000, 0x00060000, 0x00240000, 0x005c0000,
233         0x00c20000, 0x00d30000, 0x00ac0000, 0x00620000,
234         0x00910000, 0x00950000, 0x00e40000, 0x00790000,
235         0x00e70000, 0x00c80000, 0x00370000, 0x006d0000,
236         0x008d0000, 0x00d50000, 0x004e0000, 0x00a90000,
237         0x006c0000, 0x00560000, 0x00f40000, 0x00ea0000,
238         0x00650000, 0x007a0000, 0x00ae0000, 0x00080000,
239         0x00ba0000, 0x00780000, 0x00250000, 0x002e0000,
240         0x001c0000, 0x00a60000, 0x00b40000, 0x00c60000,
241         0x00e80000, 0x00dd0000, 0x00740000, 0x001f0000,
242         0x004b0000, 0x00bd0000, 0x008b0000, 0x008a0000,
243         0x00700000, 0x003e0000, 0x00b50000, 0x00660000,
244         0x00480000, 0x00030000, 0x00f60000, 0x000e0000,
245         0x00610000, 0x00350000, 0x00570000, 0x00b90000,
246         0x00860000, 0x00c10000, 0x001d0000, 0x009e0000,
247         0x00e10000, 0x00f80000, 0x00980000, 0x00110000,
248         0x00690000, 0x00d90000, 0x008e0000, 0x00940000,
249         0x009b0000, 0x001e0000, 0x00870000, 0x00e90000,
250         0x00ce0000, 0x00550000, 0x00280000, 0x00df0000,
251         0x008c0000, 0x00a10000, 0x00890000, 0x000d0000,
252         0x00bf0000, 0x00e60000, 0x00420000, 0x00680000,
253         0x00410000, 0x00990000, 0x002d0000, 0x000f0000,
254         0x00b00000, 0x00540000, 0x00bb0000, 0x00160000
255         },
256         {
257         0x63000000, 0x7c000000, 0x77000000, 0x7b000000,
258         0xf2000000, 0x6b000000, 0x6f000000, 0xc5000000,
259         0x30000000, 0x01000000, 0x67000000, 0x2b000000,
```

```
        0xfe000000, 0xd7000000, 0xab000000, 0x76000000,
        0xca000000, 0x82000000, 0xc9000000, 0x7d000000,
        0xfa000000, 0x59000000, 0x47000000, 0xf0000000,
        0xad000000, 0xd4000000, 0xa2000000, 0xaf000000,
        0x9c000000, 0xa4000000, 0x72000000, 0xc0000000,
        0xb7000000, 0xfd000000, 0x93000000, 0x26000000,
        0x36000000, 0x3f000000, 0xf7000000, 0xcc000000,
        0x34000000, 0xa5000000, 0xe5000000, 0xf1000000,
        0x71000000, 0xd8000000, 0x31000000, 0x15000000,
        0x04000000, 0xc7000000, 0x23000000, 0xc3000000,
        0x18000000, 0x96000000, 0x05000000, 0x9a000000,
        0x07000000, 0x12000000, 0x80000000, 0xe2000000,
        0xeb000000, 0x27000000, 0xb2000000, 0x75000000,
        0x09000000, 0x83000000, 0x2c000000, 0x1a000000,
        0x1b000000, 0x6e000000, 0x5a000000, 0xa0000000,
        0x52000000, 0x3b000000, 0xd6000000, 0xb3000000,
        0x29000000, 0xe3000000, 0x2f000000, 0x84000000,
        0x53000000, 0xd1000000, 0x00000000, 0xed000000,
        0x20000000, 0xfc000000, 0xb1000000, 0x5b000000,
        0x6a000000, 0xcb000000, 0xbe000000, 0x39000000,
        0x4a000000, 0x4c000000, 0x58000000, 0xcf000000,
        0xd0000000, 0xef000000, 0xaa000000, 0xfb000000,
        0x43000000, 0x4d000000, 0x33000000, 0x85000000,
        0x45000000, 0xf9000000, 0x02000000, 0x7f000000,
        0x50000000, 0x3c000000, 0x9f000000, 0xa8000000,
        0x51000000, 0xa3000000, 0x40000000, 0x8f000000,
        0x92000000, 0x9d000000, 0x38000000, 0xf5000000,
        0xbc000000, 0xb6000000, 0xda000000, 0x21000000,
        0x10000000, 0xff000000, 0xf3000000, 0xd2000000,
        0xcd000000, 0x0c000000, 0x13000000, 0xec000000,
        0x5f000000, 0x97000000, 0x44000000, 0x17000000,
        0xc4000000, 0xa7000000, 0x7e000000, 0x3d000000,
        0x64000000, 0x5d000000, 0x19000000, 0x73000000,
        0x60000000, 0x81000000, 0x4f000000, 0xdc000000,
        0x22000000, 0x2a000000, 0x90000000, 0x88000000,
        0x46000000, 0xee000000, 0xb8000000, 0x14000000,
        0xde000000, 0x5e000000, 0x0b000000, 0xdb000000,
        0xe0000000, 0x32000000, 0x3a000000, 0x0a000000,
        0x49000000, 0x06000000, 0x24000000, 0x5c000000,
        0xc2000000, 0xd3000000, 0xac000000, 0x62000000,
        0x91000000, 0x95000000, 0xe4000000, 0x79000000,
        0xe7000000, 0xc8000000, 0x37000000, 0x6d000000,
        0x8d000000, 0xd5000000, 0x4e000000, 0xa9000000,
        0x6c000000, 0x56000000, 0xf4000000, 0xea000000,
        0x65000000, 0x7a000000, 0xae000000, 0x08000000,
        0xba000000, 0x78000000, 0x25000000, 0x2e000000,
        0x1c000000, 0xa6000000, 0xb4000000, 0xc6000000,
        0xe8000000, 0xdd000000, 0x74000000, 0x1f000000,
        0x4b000000, 0xbd000000, 0x8b000000, 0x8a000000,
        0x70000000, 0x3e000000, 0xb5000000, 0x66000000,
        0x48000000, 0x03000000, 0xf6000000, 0x0e000000,
        0x61000000, 0x35000000, 0x57000000, 0xb9000000,
        0x86000000, 0xc1000000, 0x1d000000, 0x9e000000,
        0xe1000000, 0xf8000000, 0x98000000, 0x11000000,
        0x69000000, 0xd9000000, 0x8e000000, 0x94000000,
        0x9b000000, 0x1e000000, 0x87000000, 0xe9000000,
        0xce000000, 0x55000000, 0x28000000, 0xdf000000,
        0x8c000000, 0xa1000000, 0x89000000, 0x0d000000,
        0xbf000000, 0xe6000000, 0x42000000, 0x68000000,
        0x41000000, 0x99000000, 0x2d000000, 0x0f000000,
        0xb0000000, 0x54000000, 0xbb000000, 0x16000000
        }
};

static const uint32_t t_im[4][256] =
{
```



```
        {
        0x00000000, 0x0b0d090e, 0x161a121c, 0x1d171b12,
        0x2c342438, 0x27392d36, 0x3a2e3624, 0x31233f2a,
        0x58684870, 0x5365417e, 0x4e725a6c, 0x457f5362,
        0x745c6c48, 0x7f516546, 0x62467e54, 0x694b775a,
        0xb0d090e0, 0xbbdd99ee, 0xa6ca82fc, 0xadc78bf2,
        0x9ce4b4d8, 0x97e9bdd6, 0x8afea6c4, 0x81f3afca,
        0xe8b8d890, 0xe3b5d19e, 0xfea2ca8c, 0xf5afc382,
        0xc48cfca8, 0xcf81f5a6, 0xd296eeb4, 0xd99be7ba,
        0x7bbb3bdb, 0x70b632d5, 0x6da129c7, 0x66ac20c9,
        0x578f1fe3, 0x5c8216ed, 0x41950dff, 0x4a9804f1,
        0x23d373ab, 0x28de7aa5, 0x35c961b7, 0x3ec468b9,
        0x0fe75793, 0x04ea5e9d, 0x19fd458f, 0x12f04c81,
        0xcb6bab3b, 0xc066a235, 0xdd71b927, 0xd67cb029,
        0xe75f8f03, 0xec52860d, 0xf1459d1f, 0xfa489411,
        0x9303e34b, 0x980eea45, 0x8519f157, 0x8e14f859,
        0xbf37c773, 0xb43ace7d, 0xa92dd56f, 0xa220dc61,
        0xf66d76ad, 0xfd607fa3, 0xe07764b1, 0xeb7a6dbf,
        0xda595295, 0xd1545b9b, 0xcc434089, 0xc74e4987,
        0xae053edd, 0xa50837d3, 0xb81f2cc1, 0xb31225cf,
        0x82311ae5, 0x893c13eb, 0x942b08f9, 0x9f2601f7,
        0x46bde64d, 0x4db0ef43, 0x50a7f451, 0x5baafd5f,
        0x6a89c275, 0x6184cb7b, 0x7c93d069, 0x779ed967,
        0x1ed5ae3d, 0x15d8a733, 0x08cfbc21, 0x03c2b52f,
        0x32e18a05, 0x39ec830b, 0x24fb9819, 0x2ff69117,
        0x8dd64d76, 0x86db4478, 0x9bcc5f6a, 0x90c15664,
        0xa1e2694e, 0xaaef6040, 0xb7f87b52, 0xbcf5725c,
        0xd5be0506, 0xdeb30c08, 0xc3a4171a, 0xc8a91e14,
        0xf98a213e, 0xf2872830, 0xef903322, 0xe49d3a2c,
        0x3d06dd96, 0x360bd498, 0x2b1ccf8a, 0x2011c684,
        0x1132f9ae, 0x1a3ff0a0, 0x0728ebb2, 0x0c25e2bc,
        0x656e95e6, 0x6e639ce8, 0x737487fa, 0x78798ef4,
        0x495ab1de, 0x4257b8d0, 0x5f40a3c2, 0x544daacc,
        0xf7daec41, 0xfcd7e54f, 0xe1c0fe5d, 0xeacdf753,
        0xdbeec879, 0xd0e3c177, 0xcdf4da65, 0xc6f9d36b,
        0xafb2a431, 0xa4bfad3f, 0xb9a8b62d, 0xb2a5bf23,
        0x83868009, 0x888b8907, 0x959c9215, 0x9e919b1b,
        0x470a7ca1, 0x4c0775af, 0x51106ebd, 0x5a1d67b3,
        0x6b3e5899, 0x60335197, 0x7d244a85, 0x7629438b,
        0x1f6234d1, 0x146f3ddf, 0x097826cd, 0x02752fc3,
        0x335610e9, 0x385b19e7, 0x254c02f5, 0x2e410bfb,
        0x8c61d79a, 0x876cde94, 0x9a7bc586, 0x9176cc88,
        0xa055f3a2, 0xab58faac, 0xb64fe1be, 0xbd42e8b0,
        0xd4099fea, 0xdf0496e4, 0xc2138df6, 0xc91e84f8,
        0xf83dbbd2, 0xf330b2dc, 0xee27a9ce, 0xe52aa0c0,
        0x3cb1477a, 0x37bc4e74, 0x2aab5566, 0x21a65c68,
        0x10856342, 0x1b886a4c, 0x069f715e, 0x0d927850,
        0x64d90f0a, 0x6fd40604, 0x72c31d16, 0x79ce1418,
        0x48ed2b32, 0x43e0223c, 0x5ef7392e, 0x55fa3020,
        0x01b79aec, 0x0aba93e2, 0x17ad88f0, 0x1ca081fe,
        0x2d83bed4, 0x268eb7da, 0x3b99acc8, 0x3094a5c6,
        0x59dfd29c, 0x52d2db92, 0x4fc5c080, 0x44c8c98e,
        0x75ebf6a4, 0x7ee6ffaa, 0x63f1e4b8, 0x68fcedb6,
        0xb1670a0c, 0xba6a0302, 0xa77d1810, 0xac70111e,
        0x9d532e34, 0x965e273a, 0x8b493c28, 0x80443526,
        0xe90f427c, 0xe2024b72, 0xff155060, 0xf418596e,
        0xc53b6644, 0xce366f4a, 0xd3217458, 0xd82c7d56,
        0x7a0ca137, 0x7101a839, 0x6c16b32b, 0x671bba25,
        0x5638850f, 0x5d358c01, 0x40229713, 0x4b2f9e1d,
        0x2264e947, 0x2969e049, 0x347efb5b, 0x3f73f255,
        0x0e50cd7f, 0x055dc471, 0x184adf63, 0x1347d66d,
        0xcadc31d7, 0xc1d138d9, 0xdcc623cb, 0xd7cb2ac5,
        0xe6e815ef, 0xede51ce1, 0xf0f207f3, 0xfbff0efd,
        0x92b479a7, 0x99b970a9, 0x84ae6bbb, 0x8fa362b5,
        0xbe805d9f, 0xb58d5491, 0xa89a4f83, 0xa397468d
        },
```

```
392         {
393         0x00000000, 0x0d090e0b, 0x1a121c16, 0x171b121d,
394         0x3424382c, 0x392d3627, 0x2e36243a, 0x233f2a31,
395         0x68487058, 0x65417e53, 0x725a6c4e, 0x7f536245,
396         0x5c6c4874, 0x5165467f, 0x467e5462, 0x4b775a69,
397         0xd090e0b0, 0xdd99eebb, 0xca82fca6, 0xc78bf2ad,
398         0xe4b4d89c, 0xe9bdd697, 0xfea6c48a, 0xf3afca81,
399         0xb8d890e8, 0xb5d19ee3, 0xa2ca8cfe, 0xafc382f5,
400         0x8cfca8c4, 0x81f5a6cf, 0x96eeb4d2, 0x9be7bad9,
401         0xbb3bdb7b, 0xb632d570, 0xa129c76d, 0xac20c966,
402         0x8f1fe357, 0x8216ed5c, 0x950dff41, 0x9804f14a,
403         0xd373ab23, 0xde7aa528, 0xc961b735, 0xc468b93e,
404         0xe757930f, 0xea5e9d04, 0xfd458f19, 0xf04c8112,
405         0x6bab3bcb, 0x66a235c0, 0x71b927dd, 0x7cb029d6,
406         0x5f8f03e7, 0x52860dec, 0x459d1ff1, 0x489411fa,
407         0x03e34b93, 0x0eea4598, 0x19f15785, 0x14f8598e,
408         0x37c773bf, 0x3ace7db4, 0x2dd56fa9, 0x20dc61a2,
409         0x6d76adf6, 0x607fa3fd, 0x7764b1e0, 0x7a6dbfeb,
410         0x595295da, 0x545b9bd1, 0x434089cc, 0x4e4987c7,
411         0x053eddae, 0x0837d3a5, 0x1f2cc1b8, 0x1225cfb3,
412         0x311ae582, 0x3c13eb89, 0x2b08f994, 0x2601f79f,
413         0xbde64d46, 0xb0ef434d, 0xa7f45150, 0xaafd5f5b,
414         0x89c2756a, 0x84cb7b61, 0x93d0697c, 0x9ed96777,
415         0xd5ae3d1e, 0xd8a73315, 0xcfbc2108, 0xc2b52f03,
416         0xe18a0532, 0xec830b39, 0xfb981924, 0xf691172f,
417         0xd64d768d, 0xdb447886, 0xcc5f6a9b, 0xc1566490,
418         0xe2694ea1, 0xef6040aa, 0xf87b52b7, 0xf5725cbc,
419         0xbe0506d5, 0xb30c08de, 0xa4171ac3, 0xa91e14c8,
420         0x8a213ef9, 0x872830f2, 0x903322ef, 0x9d3a2ce4,
421         0x06dd963d, 0x0bd49836, 0x1ccf8a2b, 0x11c68420,
422         0x32f9ae11, 0x3ff0a01a, 0x28ebb207, 0x25e2bc0c,
423         0x6e95e665, 0x639ce86e, 0x7487fa73, 0x798ef478,
424         0x5ab1de49, 0x57b8d042, 0x40a3c25f, 0x4daacc54,
425         0xdaec41f7, 0xd7e54ffc, 0xc0fe5de1, 0xcdf753ea,
426         0xeec879db, 0xe3c177d0, 0xf4da65cd, 0xf9d36bc6,
427         0xb2a431af, 0xbfad3fa4, 0xa8b62db9, 0xa5bf23b2,
428         0x86800983, 0x8b890788, 0x9c921595, 0x919b1b9e,
429         0x0a7ca147, 0x0775af4c, 0x106ebd51, 0x1d67b35a,
430         0x3e58996b, 0x33519760, 0x244a857d, 0x29438b76,
431         0x6234d11f, 0x6f3ddf14, 0x7826cd09, 0x752fc302,
432         0x5610e933, 0x5b19e738, 0x4c02f525, 0x410bfb2e,
433         0x61d79a8c, 0x6cde9487, 0x7bc5869a, 0x76cc8891,
434         0x55f3a2a0, 0x58faacab, 0x4fe1beb6, 0x42e8b0bd,
435         0x099fead4, 0x0496e4df, 0x138df6c2, 0x1e84f8c9,
436         0x3dbbd2f8, 0x30b2dcf3, 0x27a9ceee, 0x2aa0c0e5,
437         0xb1477a3c, 0xbc4e7437, 0xab55662a, 0xa65c6821,
438         0x85634210, 0x886a4c1b, 0x9f715e06, 0x9278500d,
439         0xd90f0a64, 0xd406046f, 0xc31d1672, 0xce141879,
440         0xed2b3248, 0xe0223c43, 0xf7392e5e, 0xfa302055,
441         0xb79aec01, 0xba93e20a, 0xad88f017, 0xa081fe1c,
442         0x83bed42d, 0x8eb7da26, 0x99acc83b, 0x94a5c630,
443         0xdfd29c59, 0xd2db9252, 0xc5c0804f, 0xc8c98e44,
444         0xebf6a475, 0xe6ffaa7e, 0xf1e4b863, 0xfcedb668,
445         0x670a0cb1, 0x6a0302ba, 0x7d1810a7, 0x70111eac,
446         0x532e349d, 0x5e273a96, 0x493c288b, 0x44352680,
447         0x0f427ce9, 0x024b72e2, 0x155060ff, 0x18596ef4,
448         0x3b6644c5, 0x366f4ace, 0x217458d3, 0x2c7d56d8,
449         0x0ca1377a, 0x01a83971, 0x16b32b6c, 0x1bba2567,
450         0x38850f56, 0x358c015d, 0x22971340, 0x2f9e1d4b,
451         0x64e94722, 0x69e04929, 0x7efb5b34, 0x73f2553f,
452         0x50cd7f0e, 0x5dc47105, 0x4adf6318, 0x47d66d13,
453         0xdc31d7ca, 0xd138d9c1, 0xc623cbdc, 0xcb2ac5d7,
454         0xe815efe6, 0xe51ce1ed, 0xf207f3f0, 0xff0efdfb,
455         0xb479a792, 0xb970a999, 0xae6bbb84, 0xa362b58f,
456         0x805d9fbe, 0x8d5491b5, 0x9a4f83a8, 0x97468da3
457         },
```



```
458         {
459         0x00000000, 0x090e0b0d, 0x121c161a, 0x1b121d17,
460         0x24382c34, 0x2d362739, 0x36243a2e, 0x3f2a3123,
461         0x48705868, 0x417e5365, 0x5a6c4e72, 0x5362457f,
462         0x6c48745c, 0x65467f51, 0x7e546246, 0x775a694b,
463         0x90e0b0d0, 0x99eebbdd, 0x82fca6ca, 0x8bf2adc7,
464         0xb4d89ce4, 0xbdd697e9, 0xa6c48afe, 0xafca81f3,
465         0xd890e8b8, 0xd19ee3b5, 0xca8cfea2, 0xc382f5af,
466         0xfca8c48c, 0xf5a6cf81, 0xeeb4d296, 0xe7bad99b,
467         0x3bdb7bbb, 0x32d570b6, 0x29c76da1, 0x20c966ac,
468         0x1fe3578f, 0x16ed5c82, 0x0dff4195, 0x04f14a98,
469         0x73ab23d3, 0x7aa528de, 0x61b735c9, 0x68b93ec4,
470         0x57930fe7, 0x5e9d04ea, 0x458f19fd, 0x4c8112f0,
471         0xab3bcb6b, 0xa235c066, 0xb927dd71, 0xb029d67c,
472         0x8f03e75f, 0x860dec52, 0x9d1ff145, 0x9411fa48,
473         0xe34b9303, 0xea45980e, 0xf1578519, 0xf8598e14,
474         0xc773bf37, 0xce7db43a, 0xd56fa92d, 0xdc61a220,
475         0x76adf66d, 0x7fa3fd60, 0x64b1e077, 0x6dbfeb7a,
476         0x5295da59, 0x5b9bd154, 0x4089cc43, 0x4987c74e,
477         0x3eddae05, 0x37d3a508, 0x2cc1b81f, 0x25cfb312,
478         0x1ae58231, 0x13eb893c, 0x08f9942b, 0x01f79f26,
479         0xe64d46bd, 0xef434db0, 0xf45150a7, 0xfd5f5baa,
480         0xc2756a89, 0xcb7b6184, 0xd0697c93, 0xd967779e,
481         0xae3d1ed5, 0xa73315d8, 0xbc2108cf, 0xb52f03c2,
482         0x8a0532e1, 0x830b39ec, 0x981924fb, 0x91172ff6,
483         0x4d768dd6, 0x447886db, 0x5f6a9bcc, 0x566490c1,
484         0x694ea1e2, 0x6040aaef, 0x7b52b7f8, 0x725cbcf5,
485         0x0506d5be, 0x0c08deb3, 0x171ac3a4, 0x1e14c8a9,
486         0x213ef98a, 0x2830f287, 0x3322ef90, 0x3a2ce49d,
487         0xdd963d06, 0xd498360b, 0xcf8a2b1c, 0xc6842011,
488         0xf9ae1132, 0xf0a01a3f, 0xebb20728, 0xe2bc0c25,
489         0x95e6656e, 0x9ce86e63, 0x87fa7374, 0x8ef47879,
490         0xb1de495a, 0xb8d04257, 0xa3c25f40, 0xaacc544d,
491         0xec41f7da, 0xe54ffcd7, 0xfe5de1c0, 0xf753eacd,
492         0xc879dbee, 0xc177d0e3, 0xda65cdf4, 0xd36bc6f9,
493         0xa431afb2, 0xad3fa4bf, 0xb62db9a8, 0xbf23b2a5,
494         0x80098386, 0x8907888b, 0x9215959c, 0x9b1b9e91,
495         0x7ca1470a, 0x75af4c07, 0x6ebd5110, 0x67b35a1d,
496         0x58996b3e, 0x51976033, 0x4a857d24, 0x438b7629,
497         0x34d11f62, 0x3ddf146f, 0x26cd0978, 0x2fc30275,
498         0x10e93356, 0x19e7385b, 0x02f5254c, 0x0bfb2e41,
499         0xd79a8c61, 0xde94876c, 0xc5869a7b, 0xcc889176,
500         0xf3a2a055, 0xfaacab58, 0xe1beb64f, 0xe8b0bd42,
501         0x9fead409, 0x96e4df04, 0x8df6c213, 0x84f8c91e,
502         0xbbd2f83d, 0xb2dcf330, 0xa9ceee27, 0xa0c0e52a,
503         0x477a3cb1, 0x4e7437bc, 0x55662aab, 0x5c6821a6,
504         0x63421085, 0x6a4c1b88, 0x715e069f, 0x78500d92,
505         0x0f0a64d9, 0x06046fd4, 0x1d1672c3, 0x141879ce,
506         0x2b3248ed, 0x223c43e0, 0x392e5ef7, 0x302055fa,
507         0x9aec01b7, 0x93e20aba, 0x88f017ad, 0x81fe1ca0,
508         0xbed42d83, 0xb7da268e, 0xacc83b99, 0xa5c63094,
509         0xd29c59df, 0xdb9252d2, 0xc0804fc5, 0xc98e44c8,
510         0xf6a475eb, 0xffaa7ee6, 0xe4b863f1, 0xedb668fc,
511         0x0a0cb167, 0x0302ba6a, 0x1810a77d, 0x111eac70,
512         0x2e349d53, 0x273a965e, 0x3c288b49, 0x35268044,
513         0x427ce90f, 0x4b72e202, 0x5060ff15, 0x596ef418,
514         0x6644c53b, 0x6f4ace36, 0x7458d321, 0x7d56d82c,
515         0xa1377a0c, 0xa8397101, 0xb32b6c16, 0xba25671b,
516         0x850f5638, 0x8c015d35, 0x97134022, 0x9e1d4b2f,
517         0xe9472264, 0xe0492969, 0xfb5b347e, 0xf2553f73,
518         0xcd7f0e50, 0xc471055d, 0xdf63184a, 0xd66d1347,
519         0x31d7cadc, 0x38d9c1d1, 0x23cbdcc6, 0x2ac5d7cb,
520         0x15efe6e8, 0x1ce1ede5, 0x07f3f0f2, 0x0efdfbff,
521         0x79a792b4, 0x70a999b9, 0x6bbb84ae, 0x62b58fa3,
522         0x5d9fbe80, 0x5491b58d, 0x4f83a89a, 0x468da397
523         },
```

```
524         {
525         0x00000000, 0x0e0b0d09, 0x1c161a12, 0x121d171b,
526         0x382c3424, 0x3627392d, 0x243a2e36, 0x2a31233f,
527         0x70586848, 0x7e536541, 0x6c4e725a, 0x62457f53,
528         0x48745c6c, 0x467f5165, 0x5462467e, 0x5a694b77,
529         0xe0b0d090, 0xeebbdd99, 0xfca6ca82, 0xf2adc78b,
530         0xd89ce4b4, 0xd697e9bd, 0xc48afea6, 0xca81f3af,
531         0x90e8b8d8, 0x9ee3b5d1, 0x8cfea2ca, 0x82f5afc3,
532         0xa8c48cfc, 0xa6cf81f5, 0xb4d296ee, 0xbad99be7,
533         0xdb7bbb3b, 0xd570b632, 0xc76da129, 0xc966ac20,
534         0xe3578f1f, 0xed5c8216, 0xff41950d, 0xf14a9804,
535         0xab23d373, 0xa528de7a, 0xb735c961, 0xb93ec468,
536         0x930fe757, 0x9d04ea5e, 0x8f19fd45, 0x8112f04c,
537         0x3bcb6bab, 0x35c066a2, 0x27dd71b9, 0x29d67cb0,
538         0x03e75f8f, 0x0dec5286, 0x1ff1459d, 0x11fa4894,
539         0x4b9303e3, 0x45980eea, 0x578519f1, 0x598e14f8,
540         0x73bf37c7, 0x7db43ace, 0x6fa92dd5, 0x61a220dc,
541         0xadf66d76, 0xa3fd607f, 0xb1e07764, 0xbfeb7a6d,
542         0x95da5952, 0x9bd1545b, 0x89cc4340, 0x87c74e49,
543         0xddae053e, 0xd3a50837, 0xc1b81f2c, 0xcfb31225,
544         0xe582311a, 0xeb893c13, 0xf9942b08, 0xf79f2601,
545         0x4d46bde6, 0x434db0ef, 0x5150a7f4, 0x5f5baafd,
546         0x756a89c2, 0x7b6184cb, 0x697c93d0, 0x67779ed9,
547         0x3d1ed5ae, 0x3315d8a7, 0x2108cfbc, 0x2f03c2b5,
548         0x0532e18a, 0x0b39ec83, 0x1924fb98, 0x172ff691,
549         0x768dd64d, 0x7886db44, 0x6a9bcc5f, 0x6490c156,
550         0x4ea1e269, 0x40aaef60, 0x52b7f87b, 0x5cbcf572,
551         0x06d5be05, 0x08deb30c, 0x1ac3a417, 0x14c8a91e,
552         0x3ef98a21, 0x30f28728, 0x22ef9033, 0x2ce49d3a,
553         0x963d06dd, 0x98360bd4, 0x8a2b1ccf, 0x842011c6,
554         0xae1132f9, 0xa01a3ff0, 0xb20728eb, 0xbc0c25e2,
555         0xe6656e95, 0xe86e639c, 0xfa737487, 0xf478798e,
556         0xde495ab1, 0xd04257b8, 0xc25f40a3, 0xcc544daa,
557         0x41f7daec, 0x4ffcd7e5, 0x5de1c0fe, 0x53eacdf7,
558         0x79dbeec8, 0x77d0e3c1, 0x65cdf4da, 0x6bc6f9d3,
559         0x31afb2a4, 0x3fa4bfad, 0x2db9a8b6, 0x23b2a5bf,
560         0x09838680, 0x07888b89, 0x15959c92, 0x1b9e919b,
561         0xa1470a7c, 0xaf4c0775, 0xbd51106e, 0xb35a1d67,
562         0x996b3e58, 0x97603351, 0x857d244a, 0x8b762943,
563         0xd11f6234, 0xdf146f3d, 0xcd097826, 0xc302752f,
564         0xe9335610, 0xe7385b19, 0xf5254c02, 0xfb2e410b,
565         0x9a8c61d7, 0x94876cde, 0x869a7bc5, 0x889176cc,
566         0xa2a055f3, 0xacab58fa, 0xbeb64fe1, 0xb0bd42e8,
567         0xead4099f, 0xe4df0496, 0xf6c2138d, 0xf8c91e84,
568         0xd2f83dbb, 0xdcf330b2, 0xceee27a9, 0xc0e52aa0,
569         0x7a3cb147, 0x7437bc4e, 0x662aab55, 0x6821a65c,
570         0x42108563, 0x4c1b886a, 0x5e069f71, 0x500d9278,
571         0x0a64d90f, 0x046fd406, 0x1672c31d, 0x1879ce14,
572         0x3248ed2b, 0x3c43e022, 0x2e5ef739, 0x2055fa30,
573         0xec01b79a, 0xe20aba93, 0xf017ad88, 0xfe1ca081,
574         0xd42d83be, 0xda268eb7, 0xc83b99ac, 0xc63094a5,
575         0x9c59dfd2, 0x9252d2db, 0x804fc5c0, 0x8e44c8c9,
576         0xa475ebf6, 0xaa7ee6ff, 0xb863f1e4, 0xb668fced,
577         0x0cb1670a, 0x02ba6a03, 0x10a77d18, 0x1eac7011,
578         0x349d532e, 0x3a965e27, 0x288b493c, 0x26804435,
579         0x7ce90f42, 0x72e2024b, 0x60ff1550, 0x6ef41859,
580         0x44c53b66, 0x4ace366f, 0x58d32174, 0x56d82c7d,
581         0x377a0ca1, 0x397101a8, 0x2b6c16b3, 0x25671bba,
582         0x0f563885, 0x015d358c, 0x13402297, 0x1d4b2f9e,
583         0x472264e9, 0x492969e0, 0x5b347efb, 0x553f73f2,
584         0x7f0e50cd, 0x71055dc4, 0x63184adf, 0x6d1347d6,
585         0xd7cadc31, 0xd9c1d138, 0xcbdcc623, 0xc5d7cb2a,
586         0xefe6e815, 0xe1ede51c, 0xf3f0f207, 0xfdfbff0e,
587         0xa792b479, 0xa999b970, 0xbb84ae6b, 0xb58fa362,
588         0x9fbe805d, 0x91b58d54, 0x83a89a4f, 0x8da39746
589         }
```



```
590 };

592 #ifdef  __cplusplus
593 }
594 #endif

596 #endif  /* _AESTAB2_H */
```

****

134 Fri Jun 13 16:44:48 2008
new/usr/src/common/crypto/md5/amd64/THIRDPARTYLICENSE
6704653 THIRDPARTYLICENSE fixes for open source crypto source
****

```
1 Author: Marc Bevand <bevand_m (at) epita.fr>
2 Licence: I hereby disclaim the copyright on this code and place it
3 in the public domain.
```

```
**********************************************************
      30 Fri Jun 13 16:44:49 2008
new/usr/src/common/crypto/md5/amd64/THIRDPARTYLICENSE.descrip
6704653 THIRDPARTYLICENSE fixes for open source crypto source
**********************************************************
```

```
   1 PORTIONS OF MD5 FUNCTIONALITY
```

```
**********************************************************
    2161 Fri Jun 13 16:44:50 2008
new/usr/src/lib/pkcs11/pkcs11_softtoken/amd64/Makefile
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)Makefile   1.6     08/05/23 SMI"
# ident "@(#)Makefile   1.5     08/03/20 SMI"
#
# lib/pkcs11/pkcs11_softtoken/amd64/Makefile

AES_PSR_OBJECTS = aes_amd64.o aeskey.o
AES_PSR_OBJECTS =
ARCFOUR_PSR_OBJECTS = arcfour-x86_64.o
DES_PSR_OBJECTS =
RSA_PSR_OBJECTS =
BIGNUM_PSR_OBJECTS = bignum_amd64.o bignum_amd64_asm.o
BIGNUM_PSR_PICS = $(BIGNUM_PSR_OBJECTS:%=pics/%)
BIGNUM_CFG = -DPSR_MUL
BIGNUM_PSR_SRCS = \
        $(BIGNUMDIR)/$(MACH64)/bignum_amd64.c \
        $(BIGNUMDIR)/$(MACH64)/bignum_amd64_asm.s
        $(BIGNUMDIR)/amd64/bignum_amd64.c \
        $(BIGNUMDIR)/amd64/bignum_amd64_asm.s

pics/bignum_amd64.o := amd64_COPTFLAG = -xO3

include ../Makefile.com
include ../../../Makefile.lib.64

#
# Overrides
#
CLEANFILES      += arcfour-x86_64.s

install: all $(ROOTLIBS64) $(ROOTLINKS64)

$(BIGNUM_PSR_PICS) := CFLAGS += $(C_BIGPICFLAGS) $(BIGNUM_CFG)

LINTFLAGS64 += $(BIGNUM_CFG)

pics/%.o: %.s
        $(COMPILE.s) -o $@ $<
```



```
pics/arcfour-x86_64.o: arcfour-x86_64.s
        $(COMPILE.s) -o $@ $(AS_BIGPICFLAGS) ${@F:.o=.s}
        $(POST_PROCESS_O)

arcfour-x86_64.s: $(ARCFOURDIR)/$(MACH64)/arcfour-x86_64.pl
arcfour-x86_64.s: $(ARCFOURDIR)/amd64/arcfour-x86_64.pl
        $(PERL) $? $@

pics/%.o: $(BIGNUMDIR)/$(MACH64)/%.c
        $(COMPILE.c) -o $@ $(C_BIGPICFLAGS) $(BIGNUM_CFG) $<
        $(POST_PROCESS_O)

pics/%.o: $(BIGNUMDIR)/$(MACH64)/%.s
        $(COMPILE.s) -o $@ $(AS_BIGPICFLAGS) $(BIGNUM_CFG) $<
        $(POST_PROCESS_O)

pics/%.o: $(AESDIR)/$(MACH64)/%.c
        $(COMPILE.c) -o $@ $(AESDIR)/$(MACH64)/${@F:.o=.c}
        $(POST_PROCESS_O)

pics/%.o: $(AESDIR)/$(MACH64)/%.s
        $(COMPILE.s) -o $@ $<
        $(POST_PROCESS_O)
```

```
**********************************************************
    1844 Fri Jun 13 16:44:52 2008
new/usr/src/pkgdefs/SUNWckr/Makefile
5072963 Need an optimized AES implementation for amd64
6704653 THIRDPARTYLICENSE fixes for open source crypto source
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)Makefile   1.10    08/05/21 SMI"
# ident "@(#)Makefile   1.9     08/04/10 SMI"
#

include ../Makefile.com

DATAFILES +=    i.etcsystem i.kcfconfbase i.manifest i.preserve i.renameold \
                i.scsivhciconf i.mptconf

MACHDATAFILES += i.sdconf
CLOBBERFILES += $(MACHDATAFILES)

LICENSEFILES_i386 = \
        ../../common/crypto/aes/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/md5/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/THIRDPARTYLICENSE.cryptogams \
        ../../uts/intel/io/acpica/THIRDPARTYLICENSE
        ../../uts/intel/io/acpica/THIRDPARTYLICENSE \
        ../../common/crypto/arcfour/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/sha1/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/sha2/amd64/THIRDPARTYLICENSE

LICENSEFILES += \
        ../../common/crypto/ecc/THIRDPARTYLICENSE \
        ../../common/mpi/THIRDPARTYLICENSE \
        ../../uts/common/inet/ip/THIRDPARTYLICENSE.rts \
        ../../uts/common/inet/tcp/THIRDPARTYLICENSE \
        ../../uts/common/io/THIRDPARTYLICENSE.etheraddr \
        ../../uts/common/sys/THIRDPARTYLICENSE.icu \
        ../../uts/common/sys/THIRDPARTYLICENSE.unicode \
       $(LICENSEFILES_$(MACH))

.KEEP_STATE:

all: $(FILES) $(MACHDATAFILES) depend preinstall postinstall
```



```
install: all pkg

include ../Makefile.targ
```

****
1590 Fri Jun 13 16:44:55 2008
new/usr/src/pkgdefs/SUNWcslr/Makefile
6704653 THIRDPARTYLICENSE fixes for open source crypto source
****

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)Makefile   1.4     08/05/23 SMI"
# ident "@(#)Makefile   1.3     08/04/10 SMI"
#

include ../Makefile.com

DATAFILES += depend

LICENSEFILES_i386 = \
        ../../common/crypto/aes/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/md5/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/THIRDPARTYLICENSE.cryptogams
        ../../common/crypto/arcfour/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/sha1/amd64/THIRDPARTYLICENSE \
        ../../common/crypto/sha2/amd64/THIRDPARTYLICENSE

LICENSEFILES += \
        ../../common/crypto/ecc/THIRDPARTYLICENSE \
        ../../lib/libcmd/THIRDPARTYLICENSE \
        ../../lib/libinetutil/common/THIRDPARTYLICENSE \
        ../../lib/libmp/common/THIRDPARTYLICENSE \
        ../../lib/libresolv/THIRDPARTYLICENSE \
        ../../lib/libresolv2/THIRDPARTYLICENSE \
        ../../uts/common/sys/THIRDPARTYLICENSE.unicode \
        $(LICENSEFILES_$(MACH))

.KEEP_STATE:

all: $(FILES)

install: all pkg

include ../Makefile.targ
```

```
**********************************************************
    6978 Fri Jun 13 16:44:57 2008
new/usr/src/tools/opensolaris/license-list
5072963 Need an optimized AES implementation for amd64
6704653 THIRDPARTYLICENSE fixes for open source crypto source
**********************************************************
```

```
  1 usr/closed/cmd/man/src/util/solbookv1/THIRDPARTYLICENSE
  2 usr/closed/cmd/ndmpd/LICENSE
  3 usr/closed/lib/smartcard/ocfserv/opencard/core/event/THIRDPARTYLICENSE
  4 usr/src/cmd/agents/snmp/THIRDPARTYLICENSE
  5 usr/src/cmd/ast/THIRDPARTYLICENSE
  6 usr/src/cmd/backup/dump/THIRDPARTYLICENSE
  7 usr/src/cmd/bnu/THIRDPARTYLICENSE
  8 usr/src/cmd/checkeq/THIRDPARTYLICENSE
  9 usr/src/cmd/checknr/THIRDPARTYLICENSE
 10 usr/src/cmd/cmd-inet/THIRDPARTYLICENSE.kcmd
 11 usr/src/cmd/cmd-inet/sbin/ifparse/THIRDPARTYLICENSE
 12 usr/src/cmd/cmd-inet/usr.bin/THIRDPARTYLICENSE.rcp
 13 usr/src/cmd/cmd-inet/usr.bin/THIRDPARTYLICENSE.rsh
 14 usr/src/cmd/cmd-inet/usr.bin/ftp/THIRDPARTYLICENSE
 15 usr/src/cmd/cmd-inet/usr.bin/nc/THIRDPARTYLICENSE
 16 usr/src/cmd/cmd-inet/usr.bin/pppd/THIRDPARTYLICENSE
 17 usr/src/cmd/cmd-inet/usr.bin/pppd/plugins/THIRDPARTYLICENSE.minconnect
 18 usr/src/cmd/cmd-inet/usr.bin/pppd/plugins/THIRDPARTYLICENSE.passwd
 19 usr/src/cmd/cmd-inet/usr.bin/pppdump/LICENSE.top
 20 usr/src/cmd/cmd-inet/usr.bin/rdist/THIRDPARTYLICENSE
 21 usr/src/cmd/cmd-inet/usr.bin/telnet/THIRDPARTYLICENSE
 22 usr/src/cmd/cmd-inet/usr.lib/in.mpathd/THIRDPARTYLICENSE
 23 usr/src/cmd/cmd-inet/usr.lib/mdnsd/THIRDPARTYLICENSE
 24 usr/src/cmd/cmd-inet/usr.lib/wpad/THIRDPARTYLICENSE
 25 usr/src/cmd/cmd-inet/usr.sbin/THIRDPARTYLICENSE.arp
 26 usr/src/cmd/cmd-inet/usr.sbin/THIRDPARTYLICENSE.comsat
 27 usr/src/cmd/cmd-inet/usr.sbin/THIRDPARTYLICENSE.rlogind
 28 usr/src/cmd/cmd-inet/usr.sbin/THIRDPARTYLICENSE.route
 29 usr/src/cmd/cmd-inet/usr.sbin/ifconfig/THIRDPARTYLICENSE
 30 usr/src/cmd/cmd-inet/usr.sbin/in.ftpd/LICENSE
 31 usr/src/cmd/cmd-inet/usr.sbin/in.rdisc/THIRDPARTYLICENSE
 32 usr/src/cmd/cmd-inet/usr.sbin/in.routed/THIRDPARTYLICENSE
 33 usr/src/cmd/cmd-inet/usr.sbin/traceroute/THIRDPARTYLICENSE
 34 usr/src/cmd/compress/THIRDPARTYLICENSE
 35 usr/src/cmd/csh/THIRDPARTYLICENSE
 36 usr/src/cmd/eeprom/THIRDPARTYLICENSE
 37 usr/src/cmd/eqn/THIRDPARTYLICENSE
 38 usr/src/cmd/fs.d/udfs/fsck/THIRDPARTYLICENSE
 39 usr/src/cmd/fs.d/ufs/THIRDPARTYLICENSE
 40 usr/src/cmd/hal/LICENSE
 41 usr/src/cmd/ipf/tools/IPFILTER.LICENCE
 42 usr/src/cmd/lastcomm/THIRDPARTYLICENSE
 43 usr/src/cmd/ldap/THIRDPARTYLICENSE
 44 usr/src/cmd/look/THIRDPARTYLICENSE
 45 usr/src/cmd/lp/cmd/lptest/THIRDPARTYLICENSE
 46 usr/src/cmd/man/src/THIRDPARTYLICENSE
 47 usr/src/cmd/man/src/util/THIRDPARTYLICENSE
 48 usr/src/cmd/man/src/util/instant.src/THIRDPARTYLICENSE
 49 usr/src/cmd/man/src/util/nsgmls.src/COPYING
 50 usr/src/cmd/man/src/util/solbookv2/THIRDPARTYLICENSE
 51 usr/src/cmd/mdb/common/libstand/THIRDPARTYLICENSE
 52 usr/src/cmd/mt/THIRDPARTYLICENSE
 53 usr/src/cmd/perl/5.8.4/distrib/ext/Cwd/THIRDPARTYLICENSE
 54 usr/src/cmd/perl/THIRDPARTYLICENSE
 55 usr/src/cmd/refer/THIRDPARTYLICENSE
 56 usr/src/cmd/script/THIRDPARTYLICENSE
 57 usr/src/cmd/sendmail/THIRDPARTYLICENSE
 58 usr/src/cmd/soelim/THIRDPARTYLICENSE
 59 usr/src/cmd/ssh/THIRDPARTYLICENSE
 60 usr/src/cmd/stat/vmstat/THIRDPARTYLICENSE
```



```
 61 usr/src/cmd/tbl/THIRDPARTYLICENSE
 62 usr/src/cmd/tcpd/THIRDPARTYLICENSE
 63 usr/src/cmd/terminfo/THIRDPARTYLICENSE
 64 usr/src/cmd/tip/THIRDPARTYLICENSE
 65 usr/src/cmd/ul/THIRDPARTYLICENSE
 66 usr/src/cmd/units/THIRDPARTYLICENSE
 67 usr/src/cmd/vgrind/THIRDPARTYLICENSE
 68 usr/src/cmd/vi/THIRDPARTYLICENSE
 69 usr/src/cmd/which/THIRDPARTYLICENSE
 70 usr/src/cmd/xntpd/THIRDPARTYLICENSE
 71 usr/src/cmd/xstr/THIRDPARTYLICENSE
 72 usr/src/common/crypto/THIRDPARTYLICENSE.cryptogams
 73 usr/src/common/crypto/aes/amd64/THIRDPARTYLICENSE
 74 usr/src/common/crypto/ecc/THIRDPARTYLICENSE
 75 usr/src/common/crypto/md5/amd64/THIRDPARTYLICENSE
 72 usr/src/common/crypto/arcfour/amd64/THIRDPARTYLICENSE
 73 usr/src/common/crypto/sha1/amd64/THIRDPARTYLICENSE
 74 usr/src/common/crypto/sha2/amd64/THIRDPARTYLICENSE
 76 usr/src/common/openssl/LICENSE
 77 usr/src/grub/grub-0.95/COPYING
 78 usr/src/lib/gss_mechs/mech_krb5/THIRDPARTYLICENSE
 79 usr/src/lib/krb5/THIRDPARTYLICENSE
 80 usr/src/lib/libast/THIRDPARTYLICENSE
 81 usr/src/lib/libbc/THIRDPARTYLICENSE
 82 usr/src/lib/libbsdmalloc/THIRDPARTYLICENSE
 83 usr/src/lib/libcmd/THIRDPARTYLICENSE
 84 usr/src/lib/libdll/THIRDPARTYLICENSE
 85 usr/src/lib/libdns_sd/THIRDPARTYLICENSE
 86 usr/src/lib/libgss/THIRDPARTYLICENSE
 87 usr/src/lib/libinetutil/common/THIRDPARTYLICENSE
 88 usr/src/lib/libkmf/THIRDPARTYLICENSE
 89 usr/src/lib/libldap5/THIRDPARTYLICENSE
 90 usr/src/lib/libmp/common/THIRDPARTYLICENSE
 91 usr/src/lib/libpp/THIRDPARTYLICENSE
 92 usr/src/lib/libresolv/THIRDPARTYLICENSE
 93 usr/src/lib/libresolv2/THIRDPARTYLICENSE
 94 usr/src/lib/libsasl/THIRDPARTYLICENSE
 95 usr/src/lib/libshell/THIRDPARTYLICENSE
 96 usr/src/lib/libtecla/THIRDPARTYLICENSE
 97 usr/src/lib/libwrap/THIRDPARTYLICENSE
 98 usr/src/lib/pam_modules/authtok_check/THIRDPARTYLICENSE
 99 usr/src/lib/passwdutil/THIRDPARTYLICENSE
100 usr/src/lib/pkcs11/include/THIRDPARTYLICENSE
101 usr/src/lib/print/libhttp-core/common/LICENSE.txt
102 usr/src/stand/lib/tcp/THIRDPARTYLICENSE
103 usr/src/tools/ctf/dwarf/THIRDPARTYLICENSE
104 usr/src/ucbcmd/basename/THIRDPARTYLICENSE
105 usr/src/ucbcmd/echo/THIRDPARTYLICENSE
106 usr/src/ucbcmd/from/THIRDPARTYLICENSE
107 usr/src/ucbcmd/groups/THIRDPARTYLICENSE
108 usr/src/ucbcmd/ln/THIRDPARTYLICENSE
109 usr/src/ucbcmd/ls/THIRDPARTYLICENSE
110 usr/src/ucbcmd/plot/THIRDPARTYLICENSE
111 usr/src/ucbcmd/sum/THIRDPARTYLICENSE
112 usr/src/ucbcmd/test/THIRDPARTYLICENSE
113 usr/src/ucbcmd/tset/THIRDPARTYLICENSE
114 usr/src/ucbcmd/users/THIRDPARTYLICENSE
115 usr/src/ucbcmd/whereis/THIRDPARTYLICENSE
116 usr/src/ucbcmd/whoami/THIRDPARTYLICENSE
117 usr/src/ucblib/libcurses/THIRDPARTYLICENSE
118 usr/src/ucblib/libtermcap/THIRDPARTYLICENSE
119 usr/src/ucblib/libucb/THIRDPARTYLICENSE
120 usr/src/uts/common/gssapi/mechs/krb5/THIRDPARTYLICENSE
121 usr/src/uts/common/inet/ip/THIRDPARTYLICENSE.rts
122 usr/src/uts/common/inet/tcp/THIRDPARTYLICENSE
123 usr/src/uts/common/io/THIRDPARTYLICENSE.etheraddr
```

```
 124 usr/src/uts/common/io/aac/THIRDPARTYLICENSE
 125 usr/src/uts/common/io/afe/THIRDPARTYLICENSE
 126 usr/src/uts/common/io/ath/LICENSE
 127 usr/src/uts/common/io/chxge/com/THIRDPARTYLICENSE
 128 usr/src/uts/common/io/drm/THIRDPARTYLICENSE
 129 usr/src/uts/common/io/ib/clients/rds/THIRDPARTYLICENSE
 130 usr/src/uts/common/io/iwk/THIRDPARTYLICENSE
 131 usr/src/uts/common/io/iwk/fw-iw/LICENSE
 132 usr/src/uts/common/io/ipw/THIRDPARTYLICENSE
 133 usr/src/uts/common/io/ipw/fw-ipw2100/LICENSE
 134 usr/src/uts/common/io/iwi/THIRDPARTYLICENSE
 135 usr/src/uts/common/io/iwi/fw-ipw2200/LICENSE
 136 usr/src/uts/common/io/ixgbe/THIRDPARTYLICENSE
 137 usr/src/uts/common/io/mega_sas/THIRDPARTYLICENSE
 138 usr/src/uts/common/io/mxfe/THIRDPARTYLICENSE
 139 usr/src/uts/common/io/pcan/THIRDPARTYLICENSE
 140 usr/src/uts/common/io/pcwl/THIRDPARTYLICENSE
 141 usr/src/uts/common/io/ppp/THIRDPARTYLICENSE
 142 usr/src/uts/common/io/ral/THIRDPARTYLICENSE
 143 usr/src/uts/common/io/rtw/THIRDPARTYLICENSE
 144 usr/src/uts/common/io/sfe/THIRDPARTYLICENSE
 145 usr/src/uts/common/io/ural/THIRDPARTYLICENSE
 146 usr/src/uts/common/io/wpi/THIRDPARTYLICENSE
 147 usr/src/uts/common/io/wpi/fw-wpi/LICENSE
 148 usr/src/uts/common/sys/THIRDPARTYLICENSE.agpgart
 149 usr/src/uts/common/sys/THIRDPARTYLICENSE.icu
 150 usr/src/uts/common/sys/THIRDPARTYLICENSE.unicode
 151 usr/src/uts/common/zmod/THIRDPARTYLICENSE
 152 usr/src/uts/intel/THIRDPARTYLICENSE
 153 usr/src/uts/intel/io/acpica/THIRDPARTYLICENSE
 154 usr/src/uts/intel/io/amd8111s/THIRDPARTYLICENSE.amd8111s
 155 usr/src/uts/intel/io/amr/THIRDPARTYLICENSE
 155 usr/src/common/crypto/ecc/THIRDPARTYLICENSE
 156 usr/src/common/mpi/THIRDPARTYLICENSE
 157 usr/src/lib/libsmbfs/smb/THIRDPARTYLICENSE.apple
 158 usr/src/lib/libsmbfs/smb/THIRDPARTYLICENSE.boris_popov
 159 usr/src/lib/libsmbfs/smb/THIRDPARTYLICENSE.bsd4
 160 usr/src/lib/libsmbfs/smb/THIRDPARTYLICENSE.microsoft
```

```
**********************************************************
    2891 Fri Jun 13 16:45:00 2008
new/usr/src/uts/intel/aes/Makefile
5072963 Need an optimized AES implementation for amd64
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#ident  "@(#)Makefile   1.9     08/05/21 SMI"
#ident  "@(#)Makefile   1.8     08/01/02 SMI"
#
#       This makefile drives the production of the AES KEF provider.
#
#       intel implementation architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..
COM_DIR = $(COMMONBASE)/crypto/aes

#
#       Define the module and object file sets.
#
MODULE          = aes
LINTS           = $(AESPROV_OBJS:%.o=$(LINTS_DIR)/%.ln)
AESPROV_OBJS_32 =
AESPROV_OBJS_64 = aes_amd64.o aeskey.o
AESPROV_OBJS    += $(AESPROV_OBJS_$(CLASS))
OBJECTS         = $(AESPROV_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(AESPROV_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_CRYPTO_DIR)/$(MODULE)

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

#       set signing mode
ELFSIGN_MOD     = $(ELFSIGN_CRYPTO)

#
#       Define targets
#
```



```
ALL_TARGET      = $(BINARY)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

#
#       Linkage dependencies
#
LDFLAGS += -dy

CPPFLAGS        += -I$(COM_DIR) -DCRYPTO_PROVIDER_NAME=\"$(MODULE)\"

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

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

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

$(OBJS_DIR)/%.ln: $(COM_DIR)/amd64/%.c
        @($(LHEAD) $(LINT.c) $(COM_DIR)/amd64/${@F:.ln=.c} $(LTAIL))

$(OBJS_DIR)/%.o:  $(COM_DIR)/amd64/%.c
        $(COMPILE.c) -o $@ $(COM_DIR)/amd64/${@F:.o=.c}
        $(POST_PROCESS_O)

$(OBJS_DIR)/aes_amd64.o: $(COM_DIR)/amd64/aes_amd64.s
        $(COMPILE.s) -o $@ $(COM_DIR)/amd64/${@F:.o=.s}
        $(POST_PROCESS_O)

$(OBJS_DIR)/aes_amd64.ln: $(COM_DIR)/amd64/aes_amd64.s
        @($(LHEAD) $(LINT.s) $(COM_DIR)/amd64/${@F:.ln=.s} $(LTAIL))
```