Twofish を利用した暗号ソフト

（C）宇山　靖政

2012.04.12

Twofish 暗号について

「Blowfish」暗号の考案者として知られる Bruce Shcnier 氏が考案した、秘密鍵暗号方式の一つです。鍵長は 128、192、256 ビットの 3 種類で、128 ビットのブロックごとに暗号化を行ないます。高速な暗号化と復号化が特徴です。アルゴリズムはライセンスフリーで公開されていて、誰でも自由に使うことができます。

米国商務省標準技術局(NIST)の新世代暗号標準「AES」の候補として最終選考まで残っていましたが、ベルギーの研究者が考案した別の方式（Rijndael）が採用され、惜しくも標準の座は逃しました。

ソースコードについては、参考文献の最後や、つぎの場所にあります。

Twofish source code

参考文献

The Twofish Encryption Algorithm: A 128-Bit Block Cipher (ハードカバー)
John Kelsey

比較しやすいように、3種類のソースコードを掲載いたします。

1．最適化された、Twofish のソースコード
2．TwofishEC のソースコード
3．TwofishDC のソースコード

このうち、2，3についての二次著作権の主張はいたしません。ご自由に変更して下さい。ただし、自作のソフトの中にソースコードを組み込む場合や、鍵の長さが56ビットを超える場合には、貿易管理令についてもご確認下さい。

これらは、コンソールタイプのソフトであり、独立して動きます。

1．Twofish のソースコード

最初に、4つのヘッダーファイル、AES.H　DEBUG.H　PLATFORM.H　TABLE.H です。

Aes.h

```
////////////////////////////////////////////////////////////////////////
/* aes.h */

/* ---------- See examples at end of this file for typical usage -------- */

/* AES Cipher header file for ANSI C Submissions
        Lawrence E. Bassham III
        Computer Security Division
        National Institute of Standards and Technology

        This sample is to assist implementers developing to the
Cryptographic API Profile for AES Candidate Algorithm Submissions.
Please consult this document as a cross-reference.
```

```
        ANY CHANGES, WHERE APPROPRIATE, TO INFORMATION PROVIDED IN THIS FILE
MUST BE DOCUMENTED. CHANGES ARE ONLY APPROPRIATE WHERE SPECIFIED WITH
THE STRING "CHANGE POSSIBLE". FUNCTION CALLS AND THEIR PARAMETERS
CANNOT BE CHANGED. STRUCTURES CAN BE ALTERED TO ALLOW IMPLEMENTERS TO
INCLUDE IMPLEMENTATION SPECIFIC INFORMATION.
*/

/* Includes:
        Standard include files
*/

#include<stdio.h>
#include"platform.h"                    /* platform-specific defines */

/*      Defines:
                Add any additional defines you need
*/

#define DIR_ENCRYPT     0               /* Are we encrpyting? */
#define DIR_DECRYPT     1               /* Are we decrpyting? */
#define MODE_ECB                1               /* Are we ciphering in ECB mode? */
#define MODE_CBC                2               /* Are we ciphering in CBC mode? */
#define MODE_CFB1               3               /* Are we ciphering in 1-bit CFB mode? */

#define TRUE                    1
#define FALSE                   0

#define BAD_KEY_DIR             -1      /* Key direction is invalid (unknown value) */
#define BAD_KEY_MAT             -2      /* Key material not of correct length */
#define BAD_KEY_INSTANCE        -3      /* Key passed is not valid */
#define BAD_CIPHER_MODE  -4     /* Params struct passed to cipherInit invalid */
#define BAD_CIPHER_STATE        -5      /* Cipher in wrong state (e.g., not initialized) */

/* CHANGE POSSIBLE: inclusion of algorithm specific defines */
/* TWOFISH specific definitions */
#define         MAX_KEY_SIZE            64      /* # of ASCII chars needed to represent a key */
#define         MAX_IV_SIZE                     16      /* # of bytes needed to represent an IV */
#define         BAD_INPUT_LEN           -6      /* inputLen not a multiple of block size */
#define         BAD_PARAMS                      -7      /* invalid parameters */
#define         BAD_IV_MAT                      -8      /* invalid IV text */
#define         BAD_ENDIAN                      -9      /* incorrect endianness define */
#define         BAD_ALIGN32                     -10     /* incorrect 32-bit alignment */

#define         BLOCK_SIZE                      128     /* number of bits per block */
#define         MAX_ROUNDS                       16     /* max # rounds (for allocating subkey array)
*/
#define         ROUNDS_128                       16     /* default number of rounds for 128-bit keys*/
#define         ROUNDS_192                       16     /* default number of rounds for 192-bit keys*/
#define         ROUNDS_256                       16     /* default number of rounds for 256-bit keys*/
#define         MAX_KEY_BITS            256     /* max number of bits of key */
#define         MIN_KEY_BITS            128     /* min number of bits of key (zero pad) */
```

```
#define          VALID_SIG        0x48534946      /* initialization signature ('FISH') */
#define          MCT_OUTER                        400      /* MCT outer loop */
#define          MCT_INNER                  10000 /* MCT inner loop */
#define          REENTRANT                        1      /* nonzero forces reentrant code (slightly
slower) */

#define          INPUT_WHITEN             0       /* subkey array indices */
#define          OUTPUT_WHITEN            ( INPUT_WHITEN + BLOCK_SIZE/32)
#define          ROUND_SUBKEYS            (OUTPUT_WHITEN + BLOCK_SIZE/32)   /* use 2 * (# rounds) */
#define          TOTAL_SUBKEYS            (ROUND_SUBKEYS + 2*MAX_ROUNDS)

/* Typedefs:
        Typedef'ed data storage elements. Add any algorithm specific
        parameters at the bottom of the structs as appropriate.
*/

typedef unsigned char BYTE;
typedef  unsigned long DWORD;             /* 32-bit unsigned quantity */
typedef DWORD fullSbox[4][256];

/* The structure for key information */
typedef struct
        {
        BYTE direction;                                 /* Key used for encrypting or decrypting? */
#if ALIGN32
        BYTE dummyAlign[3];                             /* keep 32-bit alignment */
#endif
        int  keyLen;                                    /* Length of the key */
        char keyMaterial[MAX_KEY_SIZE+4];/* Raw key data in ASCII */

        /* Twofish-specific parameters: */
        DWORD keySig;                                   /* set to VALID_SIG by makeKey() */
        int       numRounds;                            /* number of rounds in cipher */
        DWORD key32[MAX_KEY_BITS/32];     /* actual key bits, in dwords */
        DWORD sboxKeys[MAX_KEY_BITS/64];/* key bits used for S-boxes */
        DWORD subKeys[TOTAL_SUBKEYS];     /* round subkeys, input/output whitening bits */
#if REENTRANT
        fullSbox sBox8x32;                              /* fully expanded S-box */
  #if defined(COMPILE_KEY) && defined(USE_ASM)
#undef  VALID_SIG
#define VALID_SIG        0x504D4F43              /* 'COMP':  C is compiled with -DCOMPILE_KEY */
        DWORD cSig1;                                    /* set after first "compile" (zero at "init")
*/
        void *encryptFuncPtr;                   /* ptr to asm encrypt function */
        void *decryptFuncPtr;                   /* ptr to asm decrypt function */
        DWORD codeSize;                                 /* size of compiledCode */
        DWORD cSig2;                                    /* set after first "compile" */
        BYTE  compiledCode[5000];         /* make room for the code itself */
  #endif
#endif
        } keyInstance;
```

```
/* The structure for cipher information */
typedef struct
        {
        BYTE  mode;                                             /* MODE_ECB, MODE_CBC, or MODE_CFB1
*/
#if ALIGN32
        BYTE dummyAlign[3];                                     /* keep 32-bit alignment */
#endif
        BYTE  IV[MAX_IV_SIZE];                      /* CFB1 iv bytes  (CBC uses iv32) */

        /* Twofish-specific parameters: */
        DWORD cipherSig;                            /* set to VALID_SIG by cipherInit() */
        DWORD iv32[BLOCK_SIZE/32];                  /* CBC IV bytes arranged as dwords */
        } cipherInstance;

/* Function protoypes */
int makeKey(keyInstance *key, BYTE direction, int keyLen, char *keyMaterial);

int cipherInit(cipherInstance *cipher, BYTE mode, char *IV);

int blockEncrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                                int inputLen, BYTE *outBuffer);

int blockDecrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                                int inputLen, BYTE *outBuffer);

int     reKey(keyInstance *key); /* do key schedule using modified key.keyDwords */

/* API to check table usage, for use in ECB_TBL KAT */
#define         TAB_DISABLE                     0
#define         TAB_ENABLE                      1
#define         TAB_RESET                       2
#define         TAB_QUERY                       3
#define         TAB_MIN_QUERY           50
int TableOp(int op);


#define         CONST                           /* helpful C++ syntax sugar, NOP for ANSI C */

#if BLOCK_SIZE == 128                   /* optimize block copies */
#define         Copy1(d,s,N)    ((DWORD *)(d))[N] = ((DWORD *)(s))[N]
#define         BlockCopy(d,s)  { Copy1(d,s,0);Copy1(d,s,1);Copy1(d,s,2);Copy1(d,s,3); }
#else
#define         BlockCopy(d,s)  { memcpy(d,s,BLOCK_SIZE/8); }
#endif


#ifdef TEST_2FISH
/*                                              ----- EXAMPLES -----

Unfortunately, the AES API is somewhat clumsy, and it is not entirely
obvious how to use the above functions.  In particular, note that
```

makeKey() takes an ASCII hex nibble key string (e.g., 32 characters
for a 128-bit key), which is rarely the way that keys are internally
represented. The reKey() function uses instead the keyInstance.key32
array of key bits and is the preferred method. In fact, makeKey()
initializes some internal keyInstance state, then parse the ASCII
string into the binary key32, and calls reKey(). To initialize the
keyInstance state, use a 'dummy' call to makeKey(); i.e., set the
keyMaterial parameter to NULL. Then use reKey() for all key changes.
Similarly, cipherInit takes an IV string in ASCII hex, so a dummy setup
call with a null IV string will skip the ASCII parse.

Note that CFB mode is not well tested nor defined by AES, so using the
Twofish MODE_CFB it not recommended. If you wish to implement a CFB mode,
build it external to the Twofish code, using the Twofish functions only
in ECB mode.

Below is a sample piece of code showing how the code is typically used
to set up a key, encrypt, and decrypt. Error checking is somewhat limited
in this example. Pseudorandom bytes are used for all key and text.

If you compile TWOFISH2.C or TWOFISH.C as a DOS (or Windows Console) app
with this code enabled, the test will be run. For example, using
Borland C, you would compile using:
  BCC32 -DTEST_2FISH twofish2.c
to run the test on the optimized code, or
  BCC32 -DTEST_2FISH twofish.c
to run the test on the pedagogical code.

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define MAX_BLK_CNT              4               /* max # blocks per call in TestTwofish */
int TestTwofish(int mode,int keySize) /* keySize must be 128, 192, or 256 */
        {                                                       /* return 0 iff test passes */
        keyInstance    ki;                      /* key information, including tables */
        cipherInstance ci;                      /* keeps mode (ECB, CBC) and IV */
        BYTE  plainText[MAX_BLK_CNT*(BLOCK_SIZE/8)];
        BYTE cipherText[MAX_BLK_CNT*(BLOCK_SIZE/8)];
        BYTE decryptOut[MAX_BLK_CNT*(BLOCK_SIZE/8)];
        BYTE iv[BLOCK_SIZE/8];
        int  i,byteCnt;

        if (makeKey(&ki,DIR_ENCRYPT,keySize,NULL) != TRUE)
                return 1;                               /* 'dummy' setup for a 128-bit key */
        if (cipherInit(&ci,mode,NULL) != TRUE)
                return 1;                               /* 'dummy' setup for cipher */

        for (i=0;i<keySize/32;i++)        /* select key bits */
```

```
            ki.key32[i]=0x10003 * rand();
        reKey(&ki);                                     /* run the key schedule */

        if (mode != MODE_ECB)           /* set up random iv (if needed)*/
                {
                for (i=0;i<sizeof(iv);i++)
                        iv[i]=(BYTE) rand();
                memcpy(ci.iv32,iv,sizeof(ci.iv32));     /* copy the IV to ci */
                }

        /* select number of bytes to encrypt (multiple of block) */
        /* e.g., byteCnt = 16, 32, 48, 64 */
        byteCnt = (BLOCK_SIZE/8) * (1 + (rand() % MAX_BLK_CNT));

        for (i=0;i<byteCnt;i++)         /* generate test data */
                plainText[i]=(BYTE) rand();

        /* encrypt the bytes */
        if (blockEncrypt(&ci,&ki, plainText,byteCnt*8,cipherText) != byteCnt*8)
                return 1;

        /* decrypt the bytes */
        if (mode != MODE_ECB)           /* first re-init the IV (if needed) */
                memcpy(ci.iv32,iv,sizeof(ci.iv32));

        if (blockDecrypt(&ci,&ki,cipherText,byteCnt*8,decryptOut) != byteCnt*8)
                return 1;

        /* make sure the decrypt output matches original plaintext */
        if (memcmp(plainText,decryptOut,byteCnt))
                return 1;

        return 0;                                       /* tests passed! */
        }

void main(void)
        {
        int testCnt,keySize;

        srand((unsigned) time(NULL));     /* randomize */

        for (keySize=128;keySize<=256;keySize+=64)
                for (testCnt=0;testCnt<10;testCnt++)
                        {
                        if (TestTwofish(MODE_ECB,keySize))
                                { printf("ECB Failure at keySize=%d",keySize); return; }
                        if (TestTwofish(MODE_CBC,keySize))
                                { printf("CBC Failure at keySize=%d",keySize); return; }
                        }
        printf("Tests passed");
        }
#endif /* TEST_2FISH */
```

Debug.h

```
////////////////////////////////////////////////////////////////////////
#ifdef DEBUG     /* keep these macros common so they are same for both versions */
CONST int debugCompile   =       1;
extern  int      debug;
extern  void DebugIO(CONST char *s);      /* display the debug output */

#define DebugDump(x,s,R,XOR,doRot,showT,needBswap)¥
        { if (debug) _Dump(x,s,R,XOR,doRot,showT,needBswap,t0,t1); }
#define DebugDumpKey(key) { if (debug) _DumpKey(key); }
#define IV_ROUND-100

void _Dump(CONST void *p,CONST char *s,int R,int XOR,int doRot,int showT,int needBswap,
                    DWORD t0,DWORD t1)
        {
        char line[512];  /* build output here */
        int  i,n;
        DWORD q[4];

        if (R == IV_ROUND)
                sprintf(line,"%sIV:    ",s);
        else
                sprintf(line,"%sR[%2d]: ",s,R);
        for (n=0;line[n];n++) ;

        for (i=0;i<4;i++)
                {
                q[i]=((CONST DWORD *)p)[i^(XOR)];
                if (needBswap) q[i]=Bswap(q[i]);
                }

        sprintf(line+n,"x= %08lX  %08lX  %08lX  %08lX.",
                        ROR(q[0],doRot*(R  )/2),
                        ROL(q[1],doRot*(R  )/2),
                        ROR(q[2],doRot*(R+1)/2),
                        ROL(q[3],doRot*(R+1)/2));
        for (;line[n];n++) ;

        if (showT)
                sprintf(line+n,"    t0=%08lX. t1=%08lX.",t0,t1);
        for (;line[n];n++) ;

        sprintf(line+n,"¥n");
        DebugIO(line);
        }

void _DumpKey(CONST keyInstance *key)
        {
```

```
        char    line[512];
        int             i;
        int             k64Cnt=(key->keyLen+63)/64;     /* round up to next multiple of 64 bits */
        int             subkeyCnt = ROUND_SUBKEYS + 2*key->numRounds;

        sprintf(line,";¥n;makeKey:   Input key          -->  S-box key     [%s]¥n",
                (key->direction == DIR_ENCRYPT) ? "Encrypt" : "Decrypt");
        DebugIO(line);
        for (i=0;i<k64Cnt;i++)    /* display in RS format */
                {
                sprintf(line,";%12s %08lX %08lX  -->  %08lX¥n","",
                        key->key32[2*i+1],key->key32[2*i],key->sboxKeys[k64Cnt-1-i]);
                DebugIO(line);
                }
        sprintf(line,";%11sSubkeys¥n","");
        DebugIO(line);
        for (i=0;i<subkeyCnt/2;i++)
                {
                sprintf(line,";%12s %08lX %08lX%s¥n","",key->subKeys[2*i],key->subKeys[2*i+1],
                        (2*i ==  INPUT_WHITEN) ? "   Input whiten" :
                        (2*i == OUTPUT_WHITEN) ? "  Output whiten" :
                      (2*i == ROUND_SUBKEYS) ? "  Round subkeys" : "");
                DebugIO(line);
                }
        DebugIO(";¥n");
        }
#else
CONST int debugCompile    =       0;
#define DebugDump(x,s,R,XOR,doRot,showT,needBswap)
#define DebugDumpKey(key)
#endif
```

Platform.h

```
/*****************************************************************************
        PLATFORM.H      -- Platform-specific defines for TWOFISH code

        Submitters:
                Bruce Schneier, Counterpane Systems
                Doug Whiting,   Hi/fn
                John Kelsey,    Counterpane Systems
                Chris Hall,             Counterpane Systems
                David Wagner,   UC Berkeley

        Code Author:            Doug Whiting,   Hi/fn

        Version  1.00           April 1998

        Copyright 1998, Hi/fn and Counterpane Systems.  All rights reserved.
```

```
        Notes:
                *       Tab size is set to 4 characters in this file

*********************************************************************/

/* use intrinsic rotate if possible */
#define ROL(x,n) (((x) << ((n) & 0x1F)) | ((x) >> (32-((n) & 0x1F))))
#define ROR(x,n) (((x) >> ((n) & 0x1F)) | ((x) << (32-((n) & 0x1F))))

#if (0) && defined(__BORLANDC__) && (__BORLANDC__ >= 0x462)
#error "!!!This does not work for some reason!!!"
#include<stdlib.h>                                      /* get prototype for _lrotl() , _lrotr() */
#pragma inline __lrotl__
#pragma inline __lrotr__
#undef  ROL                                                     /* get rid of inefficient
definitions */
#undef  ROR
#define ROL(x,n) __lrotl__(x,n)           /* use compiler intrinsic rotations */
#define ROR(x,n) __lrotr__(x,n)
#endif

#ifdef _MSC_VER
#include<stdlib.h>                                      /* get prototypes for rotation functions */
#undef  ROL
#undef  ROR
#pragma intrinsic(_lrotl,_lrotr)          /* use intrinsic compiler rotations */
#define ROL(x,n) _lrotl(x,n)
#define ROR(x,n) _lrotr(x,n)
#endif

#ifndef _M_IX86
#ifdef  __BORLANDC__
#define _M_IX86                                 300             /*make sure this is defined for Intel
CPUs */
#endif
#endif

#ifdef _M_IX86
#define         LittleEndian            1               /* e.g., 1 for Pentium, 0 for 68K */
#define         ALIGN32                         0               /* need dword alignment? (no for
Pentium) */
#else   /* non-Intel platforms */
#define         LittleEndian            0               /* (assume big endian */
#define         ALIGN32                         1               /* (assume need alignment for
non-Intel) */
#endif

#if LittleEndian
#define         Bswap(x)                (x)             /* NOP for little-endian machines */
#define         ADDR_XOR                0               /* NOP for little-endian machines */
#else
```

```
#define         Bswap(x)                ((ROR(x,8) & 0xFF00FF00) | (ROL(x,8) & 0x00FF00FF))
#define         ADDR_XOR                3               /* convert byte address in dword */
#endif

/*      Macros for extracting bytes from dwords (correct for endianness) */
#define _b(x,N)  (((BYTE *)&x)[((N) & 3) ^ ADDR_XOR]) /* pick bytes out of a dword */

#define         b0(x)                   _b(x,0)         /* extract LSB of DWORD */
#define         b1(x)                   _b(x,1)
#define         b2(x)                   _b(x,2)
#define         b3(x)                   _b(x,3)         /* extract MSB of DWORD */
```

Teble.h

```
/*****************************************************************************
        TABLE.H -- Tables, macros, constants for Twofish S-boxes and MDS matrix

        Submitters:
                Bruce Schneier, Counterpane Systems
                Doug Whiting,   Hi/fn
                John Kelsey,    Counterpane Systems
                Chris Hall,             Counterpane Systems
                David Wagner,   UC Berkeley

        Code Author:            Doug Whiting,   Hi/fn

        Version  1.00           April 1998

        Copyright 1998, Hi/fn and Counterpane Systems.  All rights reserved.

        Notes:
                *       Tab size is set to 4 characters in this file
                *       These definitions should be used in optimized and unoptimized
                        versions to insure consistency.

*****************************************************************************/

/* for computing subkeys */
#define SK_STEP                 0x02020202u
#define SK_BUMP                 0x01010101u
#define SK_ROTL                 9

/* Reed-Solomon code parameters: (12,8) reversible code
        g(x) = x**4 + (a + 1/a) x**3 + a x**2 + (a + 1/a) x + 1
   where a = primitive root of field generator 0x14D */
#define RS_GF_FDBK              0x14D           /* field generator */
#define RS_rem(x)               ¥
        { BYTE  b  = (BYTE) (x >> 24);
                        ¥
```

```
          DWORD g2 = ((b << 1) ^ ((b & 0x80) ? RS_GF_FDBK : 0 )) & 0xFF;            ¥
          DWORD g3 = ((b >> 1) & 0x7F) ^ ((b & 1) ? RS_GF_FDBK >> 1 : 0 ) ^ g2 ; ¥
          x = (x << 8) ^ (g3 << 24) ^ (g2 << 16) ^ (g3 << 8) ^ b;                   ¥
        }

/*      Macros for the MDS matrix
*       The MDS matrix is (using primitive polynomial 169):
*      01  EF  5B  5B
*      5B  EF  EF  01
*      EF  5B  01  EF
*      EF  01  EF  5B
*-----------------------------------------------------------------
* More statistical properties of this matrix (from MDS.EXE output):
*
* Min Hamming weight (one byte difference) =  8. Max=26.  Total =  1020.
* Prob[8]:      7    23    42    20    52    95    88    94   121   128    91
*             102    76    41    24     8     4     1     3     0     0     0
* Runs[8]:      2     4     5     6     7     8     9    11
* MSBs[8]:      1     4    15     8    18    38    40    43
* HW= 8: 05040705 0A080E0A 14101C14 28203828 50407050 01499101 A080E0A0
* HW= 9: 04050707 080A0E0E 10141C1C 20283838 40507070 80A0E0E0 C6432020 07070504
*        0E0E0A08 1C1C1410 38382820 70705040 E0E0A080 202043C6 05070407 0A0E080E
*        141C101C 28382038 50704070 A0E080E0 4320C620 02924B02 089A4508
* Min Hamming weight (two byte difference) =  3. Max=28.  Total = 390150.
* Prob[3]:      7    18    55   149   270   914  2185  5761 11363 20719 32079
*           43492 51612 53851 52098 42015 31117 20854 11538  6223  2492  1033
* MDS OK, ROR:   6+  7+  8+  9+ 10+ 11+ 12+ 13+ 14+ 15+ 16+
*               17+ 18+ 19+ 20+ 21+ 22+ 23+ 24+ 25+ 26+
*/
#define MDS_GF_FDBK              0x169    /* primitive polynomial for GF(256)*/
#define LFSR1(x) ( ((x) >> 1)  ^ (((x) & 0x01) ?   MDS_GF_FDBK/2 : 0))
#define LFSR2(x) ( ((x) >> 2)  ^ (((x) & 0x02) ?   MDS_GF_FDBK/2 : 0)  ¥
                                                    ^ (((x) & 0x01) ?   MDS_GF_FDBK/4 : 0))

#define Mx_1(x) ((DWORD)  (x))           /* force result to dword so << will work */
#define Mx_X(x) ((DWORD) ((x) ^           LFSR2(x)))      /* 5B */
#define Mx_Y(x) ((DWORD) ((x) ^ LFSR1(x) ^ LFSR2(x)))     /* EF */

#define M00             Mul_1
#define M01             Mul_Y
#define M02             Mul_X
#define M03             Mul_X

#define M10             Mul_X
#define M11             Mul_Y
#define M12             Mul_Y
#define M13             Mul_1

#define M20             Mul_Y
#define M21             Mul_X
#define M22             Mul_1
#define M23             Mul_Y
```

```
#define M30             Mul_Y
#define M31             Mul_1
#define M32             Mul_Y
#define M33             Mul_X

#define Mul_1   Mx_1
#define Mul_X   Mx_X
#define Mul_Y   Mx_Y

/*      Define the fixed p0/p1 permutations used in keyed S-box lookup.
        By changing the following constant definitions for P_ij, the S-boxes will
        automatically get changed in all the Twofish source code. Note that P_i0 is
        the "outermost" 8x8 permutation applied.  See the f32() function to see
        how these constants are to be  used.
*/
#define P_00    1                                       /* "outermost" permutation */
#define P_01    0
#define P_02    0
#define P_03    (P_01^1)                /* "extend" to larger key sizes */
#define P_04    1

#define P_10    0
#define P_11    0
#define P_12    1
#define P_13    (P_11^1)
#define P_14    0

#define P_20    1
#define P_21    1
#define P_22    0
#define P_23    (P_21^1)
#define P_24    0

#define P_30    0
#define P_31    1
#define P_32    1
#define P_33    (P_31^1)
#define P_34    1

#define p8(N)   P8x8[P_##N]                     /* some syntax shorthand */

/* fixed 8x8 permutation S-boxes */

/***********************************************************************
*  07:07:14  05/30/98  [4x4]  TestCnt=256. keySize=128. CRC=4BD14D9E.
* maxKeyed:  dpMax = 18. lpMax =100. fixPt =  8. skXor =  0. skDup =  6.
* log2(dpMax[ 6..18])=   --- 15.42  1.33  0.89  4.05  7.98 12.05
* log2(lpMax[ 7..12])=  9.32  1.01  1.16  4.23  8.02 12.45
* log2(fixPt[ 0.. 8])=  1.44  1.44  2.44  4.06  6.01  8.21 11.07 14.09 17.00
* log2(skXor[ 0.. 0])
* log2(skDup[ 0.. 6])=   ---  2.37  0.44  3.94  8.36 13.04 17.99
```

```
*********************************************************************/
CONST BYTE P8x8[2][256]=
        {
/*  p0:   */
/*  dpMax      = 10.   lpMax      = 64.   cycleCnt=   1  1  1  0.          */
/* 817D6F320B59ECA4.ECB81235F4A6709D.BA5E6D90C8F32471.D7F4126E9B3085CA. */
/* Karnaugh maps:
*  0111 0001 0011 1010. 0001 1001 1100 1111. 1001 1110 0011 1110. 1101 0101 1111 1001.
*  0101 1111 1100 0100. 1011 0101 0010 0000. 0101 1000 1100 0101. 1000 0111 0011 0010.
*  0000 1001 1110 1101. 1011 1000 1010 0011. 0011 1001 0101 0000. 0100 0010 0101 1011.
*  0111 0100 0001 0110. 1000 1011 1110 1001. 0011 0011 1001 1101. 1101 0101 0000 1100.
*/
        {
        0xA9, 0x67, 0xB3, 0xE8, 0x04, 0xFD, 0xA3, 0x76,
        0x9A, 0x92, 0x80, 0x78, 0xE4, 0xDD, 0xD1, 0x38,
        0x0D, 0xC6, 0x35, 0x98, 0x18, 0xF7, 0xEC, 0x6C,
        0x43, 0x75, 0x37, 0x26, 0xFA, 0x13, 0x94, 0x48,
        0xF2, 0xD0, 0x8B, 0x30, 0x84, 0x54, 0xDF, 0x23,
        0x19, 0x5B, 0x3D, 0x59, 0xF3, 0xAE, 0xA2, 0x82,
        0x63, 0x01, 0x83, 0x2E, 0xD9, 0x51, 0x9B, 0x7C,
        0xA6, 0xEB, 0xA5, 0xBE, 0x16, 0x0C, 0xE3, 0x61,
        0xC0, 0x8C, 0x3A, 0xF5, 0x73, 0x2C, 0x25, 0x0B,
        0xBB, 0x4E, 0x89, 0x6B, 0x53, 0x6A, 0xB4, 0xF1,
        0xE1, 0xE6, 0xBD, 0x45, 0xE2, 0xF4, 0xB6, 0x66,
        0xCC, 0x95, 0x03, 0x56, 0xD4, 0x1C, 0x1E, 0xD7,
        0xFB, 0xC3, 0x8E, 0xB5, 0xE9, 0xCF, 0xBF, 0xBA,
        0xEA, 0x77, 0x39, 0xAF, 0x33, 0xC9, 0x62, 0x71,
        0x81, 0x79, 0x09, 0xAD, 0x24, 0xCD, 0xF9, 0xD8,
        0xE5, 0xC5, 0xB9, 0x4D, 0x44, 0x08, 0x86, 0xE7,
        0xA1, 0x1D, 0xAA, 0xED, 0x06, 0x70, 0xB2, 0xD2,
        0x41, 0x7B, 0xA0, 0x11, 0x31, 0xC2, 0x27, 0x90,
        0x20, 0xF6, 0x60, 0xFF, 0x96, 0x5C, 0xB1, 0xAB,
        0x9E, 0x9C, 0x52, 0x1B, 0x5F, 0x93, 0x0A, 0xEF,
        0x91, 0x85, 0x49, 0xEE, 0x2D, 0x4F, 0x8F, 0x3B,
        0x47, 0x87, 0x6D, 0x46, 0xD6, 0x3E, 0x69, 0x64,
        0x2A, 0xCE, 0xCB, 0x2F, 0xFC, 0x97, 0x05, 0x7A,
        0xAC, 0x7F, 0xD5, 0x1A, 0x4B, 0x0E, 0xA7, 0x5A,
        0x28, 0x14, 0x3F, 0x29, 0x88, 0x3C, 0x4C, 0x02,
        0xB8, 0xDA, 0xB0, 0x17, 0x55, 0x1F, 0x8A, 0x7D,
        0x57, 0xC7, 0x8D, 0x74, 0xB7, 0xC4, 0x9F, 0x72,
        0x7E, 0x15, 0x22, 0x12, 0x58, 0x07, 0x99, 0x34,
        0x6E, 0x50, 0xDE, 0x68, 0x65, 0xBC, 0xDB, 0xF8,
        0xC8, 0xA8, 0x2B, 0x40, 0xDC, 0xFE, 0x32, 0xA4,
        0xCA, 0x10, 0x21, 0xF0, 0xD3, 0x5D, 0x0F, 0x00,
        0x6F, 0x9D, 0x36, 0x42, 0x4A, 0x5E, 0xC1, 0xE0
        },
/*  p1:   */
/*  dpMax      = 10.   lpMax      = 64.   cycleCnt=   2  0  0  1.          */
/* 28BDF76E31940AC5.1E2B4C376DA5F908.4C75169A0ED82B3F.B951C3DE647F208A. */
/* Karnaugh maps:
*  0011 1001 0010 0111. 1010 0111 0100 0110. 0011 0001 1111 0100. 1111 1000 0001 1100.
*  1100 1111 1111 1010. 0011 0011 1110 0100. 1001 0110 0100 0011. 0101 0110 1011 1011.
```

```
*  0010 0100 0011 0101. 1100 1000 1000 1110. 0111 1111 0010 0110. 0000 1010 0000 0011.
*  1101 1000 0010 0001. 0110 1001 1110 0101. 0001 0100 0101 0111. 0011 1011 1111 0010.
*/
        {
        0x75, 0xF3, 0xC6, 0xF4, 0xDB, 0x7B, 0xFB, 0xC8,
        0x4A, 0xD3, 0xE6, 0x6B, 0x45, 0x7D, 0xE8, 0x4B,
        0xD6, 0x32, 0xD8, 0xFD, 0x37, 0x71, 0xF1, 0xE1,
        0x30, 0x0F, 0xF8, 0x1B, 0x87, 0xFA, 0x06, 0x3F,
        0x5E, 0xBA, 0xAE, 0x5B, 0x8A, 0x00, 0xBC, 0x9D,
        0x6D, 0xC1, 0xB1, 0x0E, 0x80, 0x5D, 0xD2, 0xD5,
        0xA0, 0x84, 0x07, 0x14, 0xB5, 0x90, 0x2C, 0xA3,
        0xB2, 0x73, 0x4C, 0x54, 0x92, 0x74, 0x36, 0x51,
        0x38, 0xB0, 0xBD, 0x5A, 0xFC, 0x60, 0x62, 0x96,
        0x6C, 0x42, 0xF7, 0x10, 0x7C, 0x28, 0x27, 0x8C,
        0x13, 0x95, 0x9C, 0xC7, 0x24, 0x46, 0x3B, 0x70,
        0xCA, 0xE3, 0x85, 0xCB, 0x11, 0xD0, 0x93, 0xB8,
        0xA6, 0x83, 0x20, 0xFF, 0x9F, 0x77, 0xC3, 0xCC,
        0x03, 0x6F, 0x08, 0xBF, 0x40, 0xE7, 0x2B, 0xE2,
        0x79, 0x0C, 0xAA, 0x82, 0x41, 0x3A, 0xEA, 0xB9,
        0xE4, 0x9A, 0xA4, 0x97, 0x7E, 0xDA, 0x7A, 0x17,
        0x66, 0x94, 0xA1, 0x1D, 0x3D, 0xF0, 0xDE, 0xB3,
        0x0B, 0x72, 0xA7, 0x1C, 0xEF, 0xD1, 0x53, 0x3E,
        0x8F, 0x33, 0x26, 0x5F, 0xEC, 0x76, 0x2A, 0x49,
        0x81, 0x88, 0xEE, 0x21, 0xC4, 0x1A, 0xEB, 0xD9,
        0xC5, 0x39, 0x99, 0xCD, 0xAD, 0x31, 0x8B, 0x01,
        0x18, 0x23, 0xDD, 0x1F, 0x4E, 0x2D, 0xF9, 0x48,
        0x4F, 0xF2, 0x65, 0x8E, 0x78, 0x5C, 0x58, 0x19,
        0x8D, 0xE5, 0x98, 0x57, 0x67, 0x7F, 0x05, 0x64,
        0xAF, 0x63, 0xB6, 0xFE, 0xF5, 0xB7, 0x3C, 0xA5,
        0xCE, 0xE9, 0x68, 0x44, 0xE0, 0x4D, 0x43, 0x69,
        0x29, 0x2E, 0xAC, 0x15, 0x59, 0xA8, 0x0A, 0x9E,
        0x6E, 0x47, 0xDF, 0x34, 0x35, 0x6A, 0xCF, 0xDC,
        0x22, 0xC9, 0xC0, 0x9B, 0x89, 0xD4, 0xED, 0xAB,
        0x12, 0xA2, 0x0D, 0x52, 0xBB, 0x02, 0x2F, 0xA9,
        0xD7, 0x61, 0x1E, 0xB4, 0x50, 0x04, 0xF6, 0xC2,
        0x16, 0x25, 0x86, 0x56, 0x55, 0x09, 0xBE, 0x91
        }
        };
```

次は、2 つの C のファイル

Test2fish.c

```
/*****************************************************************************
        TST2FISH.C      -- Test code for Twofish encryption

        Submitters:
                Bruce Schneier, Counterpane Systems
                Doug Whiting,   Hi/fn
                John Kelsey,    Counterpane Systems
```

```
                Chris Hall,             Counterpane Systems
                David Wagner,   UC Berkeley

        Code Author:            Doug Whiting,   Hi/fn

        Version  1.00           April 1998

        Copyright 1998, Hi/fn and Counterpane Systems.  All rights reserved.

        Notes:
                *       Tab size is set to 4 characters in this file
                *       A random number generator is generated and used here, so that
                        the same results can be generated on different platforms/compilers.
                *       Command line arguments:
                                -h or ? ==>     give help message
                                -lNN    ==> set sanity count test loop count to NN
                                -m              ==> do full MCT generation
                                -pPath  ==> set file base path
                                -r      ==> set initial random seed based on time
                                -tNN    ==> perform timings with iteration count NN
                                -rNN    ==> set initial random seed to NN
                                -v              ==> read & verify files instead of creating them

*************************************************************************/

#include "aes.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

extern  CONST char *moduleDescription;  /* which module is running */
extern  CONST char *modeString;                 /* which key schedule mode */
extern  CONST int  debugCompile;        /* is external module compiled with debug? */

char CompilerName[8]=
        #if   defined(__BORLANDC__)
                "BCC";
        #elif defined(_MSC_VER)
                "MSC";
        #elif defined(__WATCOMC__)
                "WAT";
        #else
                "???";
        #endif

#if defined(__WATCOMC__) && defined(_M_IX86) && !defined(NO_TIMER)
    DWORD ReadTimeStampCounter(void);
        #pragma aux ReadTimeStampCounter = " db 0Fh,031h" value [eax] modify exact [eax edx] // RDTSC opcode
#endif
```

```
/*
+*****************************************************************************
*                         Constants/Macros/Tables
-*****************************************************************************/

typedef struct
        {
        FILE *f;                        /* the file being written/read */
        int  I;                                 /* test number */
        int      keySize;                       /* key size in bits */
        int      gotDebugIO;            /* got any debug IO? */
        BYTE pt[BLOCK_SIZE/8];   /* plaintext */
        BYTE ct[BLOCK_SIZE/8];   /* ciphertext */

        keyInstance    ki;              /* use ki.keyDwords as key bits */
        cipherInstance ci;              /* use ci.iv as iv bits */
        } testData;


static char hexTab[]     =      "0123456789ABCDEF";
char            filePath[80]=   "";

int                     useAsm          =       0;      /* use assembly language */
int                     mctInner=       MCT_INNER/100;
int                     mctOuter=       MCT_OUTER/10;
int                     verify          =       0;      /* set to nonzero to read&verify files */
int                     debug           =       0;      /* debugging mode */
int                     verbose         =       0;      /* verbose output */
int                     quietVerify     =       0;      /* quiet during verify */
int                     timeIterCnt     =       0;      /* how many times to iterate for timing */
DWORD           randBits[64]= {1};      /* use Knuth's additive generator */
int                     randPtr;
testData *      debugTD         = NULL; /* for use with debugIO */
int                     CLKS_BYTE       =       0;      /* use clks/byte? (vs. clks/block) */
int                     FMT_LOG         =       0;      /* format for log file */
int                     CLK_MHZ         =       200;/* default clock speed */

#define         KEY_BITS_0                      128                     /* first key bit setting to
test */
#define         STEP_KEY_BITS           ((MAX_KEY_BITS-KEY_BITS_0)/2)

static char  hexString[]=
                "0123456789ABCDEFFEDCBA98765432100011223344556677889AABBCCDDEEFF";

/*
+*****************************************************************************
*                         Functions
-*****************************************************************************/
DWORD Here(DWORD x)
        {
        unsigned int mask=~0U;
```

```
        return (* (((DWORD *)&x)-1)) & mask;
        }
extern DWORD TwofishCodeSize(void);

#ifdef USE_ASM
int cdecl get_cpu_type(void);                       /* return CPU type */
#endif

/*
+*****************************************************************************
*
* Function Name: Rand
*
* Function:                     Generate random number
*
* Arguments:            None.
*
* Return:                       New random number.
*
* Notes:                Uses Knuth's additive generator, other magic
*
-*****************************************************************************/
DWORD Rand(void)
        {
        if (randPtr >= 57)
                randPtr = 0;                        /* handle the ptr wrap */

        randBits[randPtr] += randBits[(randPtr < 7) ? randPtr-7+57 : randPtr-7];

        randBits[62]+= randBits[61];
        randBits[63] = ROL(randBits[63],9) + 0x6F4ED7D0;   /* very long period! */

        return (randBits[randPtr++] ^ randBits[63]) + randBits[62];
        }


/*
+*****************************************************************************
*
* Function Name: SetRand
*
* Function:                     Initialize random number seed
*
* Arguments:            seed    =       new seed value
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void SetRand(DWORD seed)
```

```
        {
        int i;
        DWORD x;

        randPtr=0;
        for (i=x=0;i<64;i++)
                {
                randBits[i]=seed;
                x |= seed;              /* keep track of lsb of all entries */
                seed = ROL(seed,11) + 0x12345678;
                }

        if ((x & 1) == 0)       /* insure maximal period by having at least one odd value */
                randBits[0]++;

        for (i=0;i<1000;i++)
                Rand();                 /* run it for a while */

        randBits[63] = Rand();
        randBits[62] = Rand();
        randBits[61] = Rand() | 1;      /* make it odd */
        }


/*
+*****************************************************************************
*
* Function Name: ClearTestData
*
* Function:                     Initialize test data to all zeroes
*
* Arguments:            t               =       pointer to testData structure
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void ClearTestData(testData *t)
        {
        t->gotDebugIO=0;
        memset(t->pt,0,BLOCK_SIZE/8);
        memset(t->ct,0,BLOCK_SIZE/8);
        memset(t->ci.iv32,0,BLOCK_SIZE/8);
        memset(t->ki.key32,0,MAX_KEY_BITS/8);
        memset(t->ki.keyMaterial,'0',sizeof(t->ki.keyMaterial));
#if defined(COMPILE_KEY) && defined(USE_ASM)
        t->ki.cSig1=t->ki.cSig2=0;
#endif
        }

/*
```

```
+--------------------------------------------------------------------------
*
* Function Name: FatalError
*
* Function:                     Output a fatal error message and exit
*
* Arguments:            msg             =       fatal error description (printf string)
*                                       msg2    =       2nd parameter to printf msg
*
* Return:                       None.
*
* Notes:
*
-*************************************************************************/
void FatalError(CONST char *msg,CONST char *msg2)
        {
        printf("¥nFATAL ERROR: ");
        printf(msg,msg2);
        exit(2);
        }


/*
+--------------------------------------------------------------------------
*
* Function Name: GetTimer
*
* Function:                     Return a hi-frequency timer value
*
* Arguments:            None
*
* Return:                       None.
*
* Notes:
*
-*************************************************************************/
DWORD GetTimer(void)
        {
        DWORD x;

#if defined(__BORLANDC__) && defined(__WIN32__) && !defined(NO_TIMER)
#define HI_RES_CLK      1
        x=0;
#pragma option -Od                                      /* disable optimizations (it's a REAL hack!)
*/
        __emit__(0x0F); __emit__(0x31); /* RDTSC opcode */
#pragma option -O.                                      /* restore optimization setting */
#elif defined(_MSC_VER) && defined(_M_IX86) && !defined(NO_TIMER)
#define HI_RES_CLK      1
        _asm
                {
                _emit 0x0F
```

```
                _emit 0x31
                mov     x,eax
                };
#elif defined(__WATCOMC__) && defined(_M_IX86) && !defined(NO_TIMER)
        #define HI_RES_CLK      1
        x = ReadTimeStampCounter();
#elif defined(CLOCKS_PER_SEC)
        x=clock();
#else
#define CLOCKS_PER_SEC   1                      /* very low resolution timer */
        x=time(NULL);
#endif

        return x;
        }


/*
+*************************************************************************
*
* Function Name: TimeOps
*
* Function:                     Time encryption/decryption and print results
*
* Arguments:            iterCnt  = how many calls to make
*
* Return:                       None.
*
* Notes:                None.
*
-*************************************************************************/
void TimeOps(int iterCnt)
        {
        enum { TEST_CNT  = 3, BLOCK_CNT=64 };
        int   i,j,k,n,q;
        DWORD t0,t1,dt[8],minT;
        DWORD testTime[3][TEST_CNT];
        testData t;
        BYTE text[BLOCK_CNT*(BLOCK_SIZE/8)];
        static char *testName[TEST_CNT]={"BlockEncrypt:","BlockDecrypt:","reKeyEncrypt:"};
        static char *atomName[TEST_CNT]={"block","block","call "};
        static char *format  [TEST_CNT]={"%10.1f/%s ","%10.1f/%s ","%10.1f/%s "};
        static int        denom   [TEST_CNT]={BLOCK_CNT,BLOCK_CNT,1};
        static int        needSet [TEST_CNT]={1,1,0};

        ClearTestData(&t);
        for (i=0;i<TEST_CNT;i++)
                {
                if (needSet[i] & 1)
                        {
                        denom[i]=sizeof(text)/((CLKS_BYTE) ? 1 : (BLOCK_SIZE/8) );
                        atomName[i] = (CLKS_BYTE) ? "byte "      : "block";
```

```
                        }
                format  [i] = (CLKS_BYTE) ? "%10.1f/%s " : "%10.0f/%s ";
                }

        for (i=0;i<MAX_KEY_SIZE;i++)                /* generate random key material */
                t.ki.keyMaterial[i]=hexTab[Rand() & 0xF];
        for (j=0;j<sizeof(text);j++)
                text[j]=(BYTE) Rand();
        memset(dt,0,sizeof(dt));
        dt[0]++;                                                        /* make sure it's in the cache */

        /* calibrate our timing code */
        t0=GetTimer();   t1=GetTimer();   t0++; t1++;       /* force cache line fill */
        for (i=0;i<sizeof(dt)/sizeof(dt[0]);i++)
                {
                t0=GetTimer();
                t1=GetTimer();
                dt[i]=t1-t0;
                }

        for (n=0;n<TEST_CNT;n++)                    /* gather all data into testTime[][] */
                {
                for (t.keySize=KEY_BITS_0,q=0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS,q++)
                        {
                        cipherInit(&t.ci,MODE_ECB,NULL);
                        makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial);

#if defined(HI_RES_CLK)
#define         CALL_N                                          /* just call once */
#define         ICNT    1.0
#define         TSCALE  1.0
                        for (k=0,minT=~0lu;k<iterCnt;k++)
#else
#define         CALL_N  for (j=0;j<iterCnt;j++)
#define         ICNT    ((double)iterCnt)
#define         TSCALE  ((1.0E6*CLK_MHZ)/CLOCKS_PER_SEC)
                        for (k=0,minT=~0lu;k<4;k++)
#endif
                                {       /* run a few times to get "best" time */
                                switch (n)
                                        {
                                        case 0:
                                                blockEncrypt(&t.ci,&t.ki,text,sizeof(text)*8,text);
                                                GetTimer();
                                                t0=GetTimer();
                                                CALL_N
blockEncrypt(&t.ci,&t.ki,text,sizeof(text)*8,text);
                                                t1=GetTimer();
                                                break;
                                        case 1:
                                                blockDecrypt(&t.ci,&t.ki,text,sizeof(text)*8,text);
                                                GetTimer();
```

```
                                                t0=GetTimer();
                                                CALL_N
blockDecrypt(&t.ci,&t.ki,text,sizeof(text)*8,text);
                                                t1=GetTimer();
                                                break;
                                        case 2:
                                                reKey(&t.ki);
                                                GetTimer();
                                                t0=GetTimer();
                                                CALL_N { t.ki.key32[0]+=0x87654321;       /* change
key bytes to force cache misses */
                                                                t.ki.key32[1]+=0x9ABCDEF3;
                                                                reKey(&t.ki); }
                                                t1=GetTimer();
                                                break;
                                        default:
                                                FatalError("Unknown test","");
                                                break;
                                        }
                                if (minT > t1-t0)
                                        minT = t1-t0;
                                }
                        testTime[q][n]=minT;
                        }
                }
        /* now print all the results */
#ifdef HI_RES_CLK
        if (!FMT_LOG)
                {
                printf("¥nCalibrate GetTimer(): ",t1-t0);
                for (i=0;i<sizeof(dt)/sizeof(dt[0]);i++)
                        printf("%6ld",dt[i]);
                printf("¥n¥n");
                }
#else
        printf("All times in clks, assuming %d MHz CPU (use -fmNN switch to set)¥n",CLK_MHZ);
        printf("CLOCKS_PER_SEC = %8.1f¥n",(double)CLOCKS_PER_SEC);
#endif

        printf("%-13s","keySize=");
        for (t.keySize=KEY_BITS_0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS)
                printf("%10d bits  ",t.keySize);
        printf("¥n");

        for (n=0;n<TEST_CNT;n++)
                {
                printf("%-13s",testName[n]);
                for (q=0;q<3;q++)
                        {
                        printf(format[n],TSCALE*testTime[q][n]/(ICNT*(double)denom[n]),atomName[n]);
                        }
                printf("¥n");
```

```
			}
		if (FMT_LOG)
			printf(";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;¥n");
		}


/*
+*****************************************************************************
*
* Function Name:	AES_Sanity_Check
*
* Function:			Make sure things work to the interface spec and
*					that encryption and decryption are inverse functions
*
* Arguments:		None.
*
* Return:			None.
*
* Notes:		Will FatalError if any problems found
*
-****************************************************************************/
void AES_Sanity_Check(int testCnt)
	{
	static DWORD hexVal[] =
			{0x67452301,0xEFCDAB89,0x98BADCFE,0x10325476,
			 0x33221100,0x77665544,0xBBAA9988,0xFFEEDDCC};
	static char *modeNames[]={"(null)","MODE_ECB","MODE_CBC","MODE_CFB1"};

	int i,j,q,n,testNum,lim,saveDebug=debug;
	testData t;
	keyInstance k2;
	BYTE pt[128];
	BYTE ct[128];
	char ivString[BLOCK_SIZE/4];
	char *mName;
	BYTE mode;
#if ALIGN32
	BYTE alignDummy[3];	/* keep dword alignment on stack after BYTE mode */
#endif

	if (!quietVerify) printf("¥nTwofish code sanity check...");
#if (MODE_CFB1 != MODE_ECB  + 2)
#error Need to change mode loop constants
#endif
	if (testCnt)
	for (mode=MODE_ECB;mode<=MODE_CFB1;mode++)
		{
		debug=(mode == saveDebug);
		mName=modeNames[mode];
		if (cipherInit(&t.ci,mode,hexString) != TRUE)
			FatalError("cipherInit error during sanity check %s",mName);
		if (t.ci.mode != mode)
```

```
                        FatalError("Cipher mode not set properly during sanity check %s",mName);
                if (mode != MODE_ECB)
                        for (i=0;i<BLOCK_SIZE/32;i++)
                                if (t.ci.iv32[i] != hexVal[i])
                                        FatalError("Invalid IV parse during sanity check %s",mName);
                lim = (mode == MODE_CFB1) ? (testCnt+31)/32 : testCnt;
                for (t.keySize=KEY_BITS_0;t.keySize <= MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS)
                        {
                        /* printf("Running %-9s sanity check on keySize = %3d.¥n",mName,t.keySize); */
                        if (!quietVerify) printf(".");    /* show some progress */
                        ClearTestData(&t);
                        debug=0;                /* don't show debug info in this makeKey call */
                        if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,hexString) != TRUE)
                                FatalError("Error parsing key during sanity check %s",mName);
                        debug=(mode == saveDebug);
                        for (i=0;i<t.keySize/32;i++)
                                if (t.ki.key32[i]!=hexVal[i])
                                        FatalError("Invalid key parse during sanity check %s",mName);
                        for (testNum=0;testNum<lim;testNum++)
                                {                                               /* run a bunch of
encode/decode tests */
                                if ((testNum&0x1F)==0)    /* periodic re-key time? */
                                        {
                                        for (j=0;j<t.keySize/4;j++)
                                                t.ki.keyMaterial[j]=hexTab[Rand() & 0xF];
                                        if (testNum==0)
                                                ClearTestData(&t);       /* give "easy" test data the
first time */
                                        if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) !=
TRUE)
                                                FatalError("Encrypt makeKey during sanity
check %s",mName);
                                        debug=0;
                                        if (makeKey(&k2  ,DIR_DECRYPT,t.keySize,t.ki.keyMaterial) !=
TRUE)
                                                FatalError("Decrypt makeKey during sanity
check %s",mName);
                                        debug=(mode == saveDebug);
                                        }
                                if (mode != MODE_ECB)                           /* set IV  if
needed*/
                                        for (j=0;j<BLOCK_SIZE/4;j++)
                                                ivString[j]=hexTab[(testNum)? Rand() & 0xF : 0];
                                if ((debug) || (testNum == 0))
                                        n = (BLOCK_SIZE/8);                             /* do only
one block if debugging */
                                else
                                        n = (BLOCK_SIZE/8)*(1 + (Rand() %
(sizeof(pt)/(BLOCK_SIZE/8))));

                                for (j=0;j<n;j++)                               /* set
random plaintext */
```

```
                                        pt[j]=(testNum) ? (BYTE) Rand() : 0;
                                if (mode == MODE_CBC)
                                        {       /* check that CBC works as advertised */
                                        cipherInit(&t.ci,mode,ivString);
                                        t.ci.mode=MODE_ECB;
                                        for (q=0;q<BLOCK_SIZE/8;q++)      /* copy over the iv */
                                                t.pt[q] = (BYTE) (t.ci.iv32[q/4] >> (8*(q&3)));
        /* auto-Bswap! */
                                        for (j=0;j<n;j+=BLOCK_SIZE/8)
                                                {
                                                for (q=0;q<BLOCK_SIZE/8;q++)      /* xor in next block
*/
                                                        t.pt[q] ^= pt[j+q];
                                                debug=0;
                                                if (BLOCK_SIZE !=
blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.pt))
                                                        FatalError("blockEncrypt return value during
sanity check %s",mName);
                                                debug=(mode == saveDebug);
                                                }
                                        t.ci.mode=MODE_CBC;                       /* restore mode */
                                        }
                                /* encrypt */
                                cipherInit(&t.ci,mode,ivString);
                                if ((testNum < 4) || (Rand() & 1))
                                        {
                                        if (n*8 != blockEncrypt(&t.ci,&t.ki,pt,n*8,ct))
                                                FatalError("blockEncrypt return value during sanity
check %s",mName);
                                        }
                                else    /* do it in pieces */
                                        for (j=0;j<n;j+=BLOCK_SIZE/8)
                                                if (BLOCK_SIZE !=
blockEncrypt(&t.ci,&t.ki,pt+j,BLOCK_SIZE,ct+j))
                                                FatalError("blockEncrypt return value during sanity
check %s",mName);

                                if (mode == MODE_CBC)                     /* validate CBC "hash" */
                                        for (q=0;q<BLOCK_SIZE/8;q++)
                                                if (t.pt[q] != ct[n-BLOCK_SIZE/8+q])
                                                        FatalError("CBC doesn't work during sanity
check %s",mName);
                                /* decrypt */
                                cipherInit(&t.ci,mode,ivString);
                                if ((testNum < 4) || (Rand() & 1))
                                        {
                                        if (n*8 != blockDecrypt(&t.ci,&t.ki,ct,n*8,ct))
                                                FatalError("blockDecrypt return value during sanity
check %s",mName);
                                        }
                                else    /* do it in pieces */
                                        for (j=0;j<n;j+=BLOCK_SIZE/8)
```

```
                                        if (BLOCK_SIZE !=
blockDecrypt(&t.ci,&t.ki,ct+j,BLOCK_SIZE,ct+j))
                                        FatalError("blockEncrypt return value during sanity
check %s",mName);

                                /* compare */
                                for (j=0;j<n;j++)
                                        if (pt[j] != ct[j])
                                                {
                                                char s[128];
                                                sprintf(s,"Sanity check: encrypt/decrypt miscompare
(mode=%s,keySize=%d)",
                                                                mName,t.keySize);
                                                FatalError(s,"");
                                                }
                                if (debug)
                                        {
                                        if (testNum >= debug)
                                                exit(1);

        printf(";----------------------------------------------¥n");
                                        }
                                }
                        }
                }
        debug=saveDebug;
        if (!quietVerify) printf("  OK¥n");
        }


/*
+*************************************************************************
*
* Function Name: AES_FileIO
*
* Function:                     Output to file or verify file contents vs. string
*
* Arguments:            f               =       opened file
*                                       s               =       string to output/compare
(NULL-->reset, return)
*                                       errOK   =       do not fatalError on miscompare
*
* Return:                       Zero --> compare ok
*
* Notes:                  On miscompare, FatalError (unless errOK)
*
-*************************************************************************/
int AES_FileIO(FILE *f,CONST char *s,int errOK)
        {
        int  i;
        static int  lineNum=0;
        static int  j=0;
```

```
        static char line[516]="";

        if (s == NULL)   /* starting new file */
                {
                line[0]=j=lineNum=0;
                return 0;
                }

        if (!verify)
                {
                fprintf(f,s);
                return 0;
                }

        /* here to verify the file against the string */
        for (i=0;s[i];i++)
                {
                while (line[j] == 0)
                        {
                        lineNum++;
                        if (fgets(line,sizeof(line)-4,f) == NULL)
                                {
                                if ((s[i]=='¥n') && (s[i+1]==0))
                                        {
                                        line[0]=j=0;     /* missing final eol is ok */
                                        return 0;
                                        }
                                FatalError("Unexpected EOF looking for %s",s);
                                }
                        if (verbose) printf(line);
                        j=0;
                        }
                if (s[i] != line[j])
                        {
                        if ((s[i] == '¥n') && ((i==0) || (s[i-1] == '¥n'))) continue; /* blank line skip
*/
                        if (line[j] == '¥n') {j++; continue; }
                        if (!errOK)
                                {
                                char tmp[1024];
                                sprintf(tmp,"Miscompare at line #%d:¥n%s¥nlooking
for¥n¥n%%s",lineNum,line);
                                FatalError(tmp,s);
                                }
                        line[0]=j=0;     /* let caller re-synch if desired */
                        return 1;                /* return error flag */
                        }
                j++;
                }

        return 0;
        }
```

```
/*
+*****************************************************************************
*
* Function Name: AES_PutFileHeader
*
* Function:                     Output a text header for AES test file
*
* Arguments:            fileName=       name of file to create
*                                       testName=       name of the specific test
*
* Return:                       Open FILE pointer
*
* Notes:                        If unable to create, gives FatalError
*
-*****************************************************************************/
FILE *AES_PutFileHeader(CONST char *fileName,CONST char *testName)
        {
        char s[512];
        FILE *f;

        sprintf(s,"%s%s",filePath,fileName);
        if (verify)
                {
                if (!quietVerify) printf("Verifying file %s",s);
                f=fopen(s,"rt");
                AES_FileIO(NULL,NULL,0);          /* reset file read state */
                }
        else
                {
                printf("Creating file %s.¥n",s);
                f=fopen(s,"wt");
                }
        if (f == NULL) FatalError("Unable to open file '%s'",s);

        sprintf(s,
                        "¥n=========================¥n"
                        "¥n"
                        "FILENAME:  ¥"%s¥"¥n"
                        "¥n"
                        "%s¥n"
                        "¥n"
                        "Algorithm Name:        TWOFISH¥n"
                        "Principal Submitter:  Bruce Schneier, Counterpane Systems¥n"
                        "¥n"
                        "==========¥n"
                        "¥n",
                        fileName,testName);

        if (AES_FileIO(f,s,1))
```

```
                {                                                       /* header mismatch */
                if (!verify)
                        FatalError("Miscompare while not verifying??","");
                printf("  ¥tWARNING:  header mismatch!");
                fgets(s,sizeof(s)-4,f);
                do      {                                               /* skip rest of "bad" header */
                        if (fgets(s,sizeof(s)-4,f) == NULL)
                                break;                  /* end of file? */
                        }
                while ((s[0] != '=') || (s[1] != '='));
                fgets(s,sizeof(s)-4,f);  /* skip trailing blank line */
                }

        if (verify)
                if (!quietVerify) printf("¥n");

        return f;
        }

/*
+*****************************************************************************
*
* Function Name: AES_PutTestResult
*
* Function:                     Output a test result
*
* Arguments:            f               =       output file
*                                       name    =       name of field
*                                       p               =       pointer to bytes/dwords
*                                       cnt             =       # bytes to output
*                                       fmt32   =       nonzero --> p points to dwords, else bytes
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void AES_PutBytes(FILE *f,CONST char *name,CONST void *p,int cnt,int fmt32)
        {
        char s[128];
        int i,j,a;
        if (p == NULL) return;

        a = (fmt32) ? ADDR_XOR : 0;       /* handle big/little endian on dword i/o */

        sprintf(s,"%s=",name);
        for (j=0;s[j];j++) ;
        for (i=0;i<cnt;i++)
                {
                s[j++]=hexTab[((BYTE *)p)[i ^ a] >> 4 ];
                s[j++]=hexTab[((BYTE *)p)[i ^ a] & 0xF];
                }
        s[j++]='¥n';
```

```
        s[j  ]=0;       /* terminate the string */

        AES_FileIO(f,s,0);
        }


/*
+*****************************************************************************
*
* Function Name: AES_printf
*
* Function:                     Output a test result
*
* Arguments:            t               =       testData (includes output file)
*                                       fmt             =       format list (string of chars, see
notes)
*
* Return:                       None.
*
* Notes:
*       The fmt string specifies what is output. The following characters are
*       treated specially (S,K,P,C,v,V,I).  See the code in the switch statement
*       to see how they are handled.  All other characters (e.g., '¥n') are
*       simply output to the file.
*
-*****************************************************************************/
void AES_printf(testData *t,CONST char *fmt)
        {
        char s[40];

        for (s[1]=0;*fmt;fmt++)
                switch (*fmt)
                        {
                        case 'I': sprintf(s,"I=%d¥n",t->I);
        AES_FileIO(t->f,s,0);    break;
                        case 'S': sprintf(s,"KEYSIZE=%d¥n",t->keySize); AES_FileIO(t->f,s,0);
        break;
                        case 'P': AES_PutBytes(t->f,"PT" ,t->pt             ,BLOCK_SIZE/8,0);
        break;
                        case 'C': AES_PutBytes(t->f,"CT" ,t->ct         ,BLOCK_SIZE/8,0);   break;
                        case 'v': AES_PutBytes(t->f,"IV" ,t->ci.IV    ,BLOCK_SIZE/8,0);     break;
                        case 'V': AES_PutBytes(t->f,"IV" ,t->ci.iv32 ,BLOCK_SIZE/8,1);      break;
                        case 'K': AES_PutBytes(t->f,"KEY",t->ki.key32,t->keySize/8,1);      break;
                        default:  s[0]=*fmt; s[1]=0; AES_FileIO(t->f,s,0);
        break;
                        }
        }

/*
+*****************************************************************************
*
* Function Name: AES_EndSection
```

```
*
* Function:                     Insert a separator between sections
*
* Arguments:            t               =       ptr to testData, contains file
*
* Return:                       None.
*
* Notes:
*
-*************************************************************************/
void AES_EndSection(testData *t)
        {
        AES_FileIO(t->f,"==========¥n¥n",0);
        }

/*
+*************************************************************************
*
* Function Name: AES_Close
*
* Function:                     Close an AES text file
*
* Arguments:            t       =       testData ptr (contains f)
*
* Return:                       None.
*
* Notes:
*
-*************************************************************************/
void AES_Close(testData *t)
        {
        fclose(t->f);
        }

/*
+*************************************************************************
*
* Function Name: DebugIO
*
* Function:                     Output debug string
*
* Arguments:            s       =       string to output
*
* Return:                       None.
*
* Notes:
*
-*************************************************************************/
void DebugIO(CONST char *s)
        {
        if (debugTD)
                {
```

```
                AES_FileIO(debugTD->f,s,0);
                debugTD->gotDebugIO=1;
                }
        else
                printf(s);
        }

/*
+*************************************************************************
*
* Function Name: AES_Test_VK
*
* Function:                     Run variable key test
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-*************************************************************************/
void AES_Test_VK(CONST char *fname)
        {
        testData t;

        memset(t.ki.keyMaterial,'0',MAX_KEY_SIZE);

        t.f=AES_PutFileHeader(fname,
                        "Electronic Codebook (ECB) Mode¥nVariable Key Known Answer Tests");

        if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                FatalError("cipherInit error during %s test",fname);

        for (t.keySize=KEY_BITS_0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS)
                {
                ClearTestData(&t);
                AES_printf(&t,"S¥nP¥n"); /* output key size, plaintext */
                for (t.I=1;t.I<=t.keySize;t.I++)
                        {
                        t.ki.keyMaterial[(t.I-1)/4]='0' + (8 >> ((t.I-1) & 3));
                        if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
                                FatalError("Error parsing key during %s test",fname);
                        if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
                                FatalError("blockEncrypt return during %s test",fname);
                        AES_printf(&t,"IKC¥n");  /* output I, KEY, CT, newline */

                        t.ki.keyMaterial[(t.I-1)/4]='0'; /* rezero the key bit */
                        }
                AES_EndSection(&t);
                }

        AES_Close(&t);
```

```
        }

/*
+***************************************************************************
*
* Function Name: AES_Test_Intermediate
*
* Function:                     Run intermediate value test
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-***************************************************************************/
void AES_Test_Intermediate(CONST char *fname)
        {
        testData t;

        if ((useAsm) || (!debugCompile))
                {
                if (!quietVerify) printf("WARNING: Skipping %s test¥n",fname);
                return;
                }

        memset(t.ki.keyMaterial,'0',MAX_KEY_SIZE);

        t.f=AES_PutFileHeader(fname,
                        "Electronic Codebook (ECB) Mode¥nIntermediate Value Tests");

        if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                FatalError("cipherInit error during %s test",fname);

        for (t.keySize=KEY_BITS_0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS)
                {
                ClearTestData(&t);
                debugTD=&t;
                debug=1;

                if (t.keySize > KEY_BITS_0)
                        memcpy(t.ki.keyMaterial,hexString,sizeof(t.ki.keyMaterial));
                debug=0;
                if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
                        FatalError("Error parsing key during %s test",fname);
                debug=1;
                AES_printf(&t,"S¥nK¥n"); /* output key size, key */
                if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
                        FatalError("Error parsing key during %s test",fname);

                AES_printf(&t,"P¥n");           /* output plaintext */
                AES_FileIO(t.f,"Encrypt()¥n",0);
```

```
                if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
                        FatalError("blockEncrypt return during %s test",fname);
                AES_printf(&t,"¥nC¥n");  /* output CT, newline */

                AES_FileIO(t.f,"Decrypt()¥n",0);
                AES_printf(&t,"¥nC¥n");  /* output CT, newline */
                if (blockDecrypt(&t.ci,&t.ki,t.ct,BLOCK_SIZE,t.pt) != BLOCK_SIZE)
                        FatalError("blockDecrypt return during %s test",fname);
                AES_printf(&t,"¥nP¥n");  /* output PT, newline */

                AES_EndSection(&t);
                if (!t.gotDebugIO)
                        FatalError("Need to run DEBUG version to test %s",fname);
                debug=0;
                debugTD=NULL;
                }

        AES_Close(&t);
        }


/*
+*****************************************************************************
*
* Function Name: AES_Test_VT
*
* Function:                     Run variable text test
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void AES_Test_VT(CONST char *fname)
        {
        testData t;

        memset( t.ki.keyMaterial,'0',MAX_KEY_SIZE);

        t.f=AES_PutFileHeader(fname,
                        "Electronic Codebook (ECB) Mode¥nVariable Text Known Answer Tests");

        if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                FatalError("cipherInit error during %s test",fname);

        for (t.keySize=KEY_BITS_0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS)
                {
                ClearTestData(&t);
                if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
```

```
                        FatalError("Error parsing key during %s test",fname);

                AES_printf(&t,"S¥nK¥n"); /* output key size, key */
                for (t.I=1;t.I<=BLOCK_SIZE;t.I++)
                        {
                        t.pt[(t.I-1)/8] = 0x80 >> ((t.I-1) & 7);
                        if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
                                FatalError("blockEncrypt return during %s test",fname);
                        AES_printf(&t,"IPC¥n");  /* output I, PT, CT, newline */
                        t.pt[(t.I-1)/8] = 0;
                        }
                AES_EndSection(&t);
                }
        AES_Close(&t);
        }

/*
+*****************************************************************************
*
* Function Name: AES_Test_TBL
*
* Function:                     Run tabl test
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void AES_Test_TBL(CONST char *fname)
        {
        int i;
        testData t;

        t.f=AES_PutFileHeader(fname,
                        "Electronic Codebook (ECB) Mode¥nTables Known Answer Test¥n"
                        "Tests permutation tables and MDS matrix multiply tables.");

        for (t.keySize=KEY_BITS_0;t.keySize <= MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS)
                {
                AES_printf(&t,"S¥n");    /* output key size */
                TableOp(TAB_ENABLE);
                TableOp(TAB_RESET);

                ClearTestData(&t);
                if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                        FatalError("Error cipherInit() during %s test",fname);

                for (t.I=1;TableOp(TAB_QUERY) == FALSE;t.I++)
                        {
                        if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
```

```
                        FatalError("Error parsing key during %s test",fname);
                    if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
                        FatalError("blockEncrypt during %s test",fname);
                    AES_printf(&t,"IKPC¥n");          /* show the 'vector' */
                    memcpy(t.ki.keyMaterial+MAX_KEY_SIZE/2,t.ki.keyMaterial,MAX_KEY_SIZE/2);
                    for (i=0;i<MAX_KEY_SIZE/2;i+=2)   /* make new key from old paintext */
                        {
                        t.ki.keyMaterial[i  ]=hexTab[t.pt[i/2] >> 4];
                        t.ki.keyMaterial[i+1]=hexTab[t.pt[i/2] &0xF];
                        }
                    memcpy(t.pt,t.ct,BLOCK_SIZE/8);   /* use ciphertext as new plaintext */
                    }
                TableOp(TAB_DISABLE);
                AES_EndSection(&t);               /* output separator */
                if (!quietVerify) printf("  [%d,%3d]",t.keySize,t.I);
                }
        if (!quietVerify) printf("¥n");
        AES_Close(&t);
        }

/*
+*****************************************************************************
*
* Function Name: AES_Test_ECB_E_MCT
*
* Function:                     Run ECB Monte Carlo test for ECB encryption
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void AES_Test_ECB_E_MCT(CONST char *fname)
        {
        int i,j,q;
        testData t;

        t.f=AES_PutFileHeader(fname,
                        "Electronic Codebook (ECB) Mode - ENCRYPTION¥nMonte Carlo Test");

        if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                FatalError("cipherInit error during %s test",fname);

        for (t.keySize=KEY_BITS_0,q=0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS,q+=2)
                {
                AES_printf(&t,"S¥n");                   /* output key size */
                if (!quietVerify) printf("  keyLen = %3d. ",t.keySize);

                ClearTestData(&t);                              /* start with all zeroes */
                if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
```

```
                        FatalError("Error parsing key during %s test",fname);

                for (t.I=0;t.I<mctOuter;t.I++)
                        {
                        AES_printf(&t,"IKP");
                        if (!quietVerify) printf("%3d¥b¥b¥b",t.I);
                        for (j=0;j<mctInner;j++)
                                {
                                if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
                                        FatalError("blockEncrypt return during %s test",fname);
                                if (j == mctInner-1)      /* xor the key for next outer loop */
                                        for (i=0;i<t.keySize/32;i++)
                                                t.ki.key32[i] ^=
                                                        Bswap((i>=q) ? ((DWORD *)t.ct)[i-q] :
                                                                                ((DWORD
*)t.pt)[BLOCK_SIZE/32-q+i]);
                                BlockCopy(t.pt,t.ct);
                                }
                        AES_printf(&t,"C¥n");
                        if (reKey(&t.ki) != TRUE)
                                FatalError("reKey return during %s test",fname);
                        }
                AES_EndSection(&t);
                }
        if (!quietVerify) printf("   ¥n");
        AES_Close(&t);
        }

/*
+*****************************************************************************
*
* Function Name: AES_Test_ECB_D_MCT
*
* Function:                     Run ECB Monte Carlo test for ECB decryption
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void AES_Test_ECB_D_MCT(CONST char *fname)
        {
        int i,j,q;
        testData t;

        t.f=AES_PutFileHeader(fname,
                        "Electronic Codebook (ECB) Mode - DECRYPTION¥nMonte Carlo Test");

        if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                FatalError("cipherInit error during %s test",fname);
```

```
        for (t.keySize=KEY_BITS_0,q=0;t.keySize <= MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS,q+=2)
                {
                AES_printf(&t,"S¥n");                       /* output key size */
                if (!quietVerify) printf("  keyLen = %3d. ",t.keySize);

                ClearTestData(&t);                                  /* start with all zeroes */
                if (makeKey(&t.ki,DIR_DECRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
                        FatalError("Error parsing key during %s test",fname);

                for (t.I=0;t.I<mctOuter;t.I++)
                        {
                        AES_printf(&t,"IKC");
                        if (!quietVerify) printf("%3d¥b¥b¥b",t.I);
                        for (j=0;j<mctInner;j++)
                                {
                                if (blockDecrypt(&t.ci,&t.ki,t.ct,BLOCK_SIZE,t.pt) != BLOCK_SIZE)
                                        FatalError("blockDecrypt return during %s test",fname);
                                if (j == mctInner-1)      /* xor the key for next outer loop */
                                        for (i=0;i<t.keySize/32;i++)
                                                t.ki.key32[i] ^=
                                                        Bswap((i>=q) ? ((DWORD *)t.pt)[i-q] :
                                                                        ((DWORD
*)t.ct)[BLOCK_SIZE/32-q+i]);
                                BlockCopy(t.ct,t.pt);
                                }
                        AES_printf(&t,"P¥n");
                        if (reKey(&t.ki) != TRUE)
                                FatalError("reKey return during %s test",fname);
                        }
                AES_EndSection(&t);
                }
        if (!quietVerify) printf("   ¥n");
        AES_Close(&t);
        }

/*
+*****************************************************************************
*
* Function Name: AES_Test_CBC_E_MCT
*
* Function:                     Run ECB Monte Carlo test for CBC encryption
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void AES_Test_CBC_E_MCT(CONST char *fname)
        {
```

```
        int i,j,q;
        testData t;
        BYTE ctPrev[BLOCK_SIZE/8];
        BYTE IV[BLOCK_SIZE/8];
#define CV      t.ci.IV                                         /* use t.ci.IV as CV */

        t.f=AES_PutFileHeader(fname,
                        "Cipher Block Chaining (CBC) Mode - ENCRYPTION¥nMonte Carlo Test");

        if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                FatalError("cipherInit error during %s test",fname);

        for (t.keySize=KEY_BITS_0,q=0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS,q+=2)
                {
                AES_printf(&t,"S¥n");                       /* output key size */
                if (!quietVerify) printf("  keyLen = %3d. ",t.keySize);

                ClearTestData(&t);                                  /* start with all zeroes */
                memset(IV,0,sizeof(IV));
                if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial)   != TRUE)
                        FatalError("Error parsing key during %s test",fname);

                BlockCopy(CV,IV);
                for (t.I=0;t.I<mctOuter;t.I++)
                        {
                        AES_printf(&t,"IKvP");
                        if (!quietVerify) printf("%3d¥b¥b¥b",t.I);
                        for (j=0;j<mctInner;j++)
                                {
                                for (i=0;i<BLOCK_SIZE/8;i++)
                                        t.pt[i]  ^= CV[i];                  /* IB = PT ^ CV */
                                BlockCopy(ctPrev,t.ct);           /* save previous ct */

                                if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
                                        FatalError("blockEncrypt return during %s test",fname);
                                BlockCopy(t.pt,(j)? ctPrev : CV);
                                BlockCopy(CV,t.ct);
                                }
                        AES_printf(&t,"C¥n");

                        for (i=0;i<t.keySize/32;i++)
                                t.ki.key32[i] ^=
                                        Bswap((i>=q) ? ((DWORD *)t.ct  )[i-q] :
                                                                ((DWORD
*)ctPrev)[BLOCK_SIZE/32-q+i]);
                        BlockCopy(t.pt,ctPrev);
                        BlockCopy(CV,t.ct);

                        if (reKey(&t.ki) != TRUE)
                                FatalError("reKey return during %s test",fname);
                        }
                AES_EndSection(&t);
```

```
                }
        if (!quietVerify) printf("   ¥n");
        AES_Close(&t);
        }

/*
+***************************************************************************
*
* Function Name: AES_Test_CBC_D_MCT
*
* Function:                     Run ECB Monte Carlo test for CBC decryption
*
* Arguments:            fname   =       file name to produce
*
* Return:                       None.
*
* Notes:
*
-***************************************************************************/
void AES_Test_CBC_D_MCT(CONST char *fname)
        {
        int i,j,q;
        testData t;
        BYTE ptPrev[BLOCK_SIZE/8];
        BYTE IV[BLOCK_SIZE/8];
#define CV      t.ci.IV                                         /* use t.ci.IV as CV */

        t.f=AES_PutFileHeader(fname,
                        "Cipher Block Chaining (CBC) Mode - DECRYPTION¥nMonte Carlo Test");

        if (cipherInit(&t.ci,MODE_ECB,NULL) != TRUE)
                FatalError("cipherInit error during %s test",fname);

        for (t.keySize=KEY_BITS_0,q=0;t.keySize <= MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS,q+=2)
                {
                AES_printf(&t,"S¥n");                   /* output key size */
                if (!quietVerify) printf("  keyLen = %3d. ",t.keySize);

                ClearTestData(&t);                              /* start with all zeroes */
                memset(IV,0,sizeof(IV));
                if (makeKey(&t.ki,DIR_DECRYPT,t.keySize,t.ki.keyMaterial)   != TRUE)
                        FatalError("Error parsing key during %s test",fname);

                BlockCopy(CV,IV);
                for (t.I=0;t.I<mctOuter;t.I++)
                        {
                        AES_printf(&t,"IKvC");
                        if (!quietVerify) printf("%3d¥b¥b¥b",t.I);
                        for (j=0;j<mctInner;j++)
                                {
                                BlockCopy(ptPrev,t.pt);
                                if (blockDecrypt(&t.ci,&t.ki,t.ct,BLOCK_SIZE,t.pt) != BLOCK_SIZE)
```

```
                                        FatalError("blockDecrypt return during %s test",fname);
                                for (i=0;i<BLOCK_SIZE/8;i++)
                                        t.pt[i]  ^= CV[i];                /* PT = OB ^ CV */
                                BlockCopy(CV,t.ct);                       /* CV = CT */
                                BlockCopy(t.ct,t.pt);             /* CT = PT */
                                }
                        AES_printf(&t,"P¥n");

                        for (i=0;i<t.keySize/32;i++)
                                t.ki.key32[i] ^=
                                        Bswap((i>=q) ? ((DWORD *)t.pt  )[i-q] :
                                                                        ((DWORD
*)ptPrev)[BLOCK_SIZE/32-q+i]);
                        if (reKey(&t.ki) != TRUE)
                                FatalError("reKey return during %s test",fname);
                        }
                AES_EndSection(&t);
                }
        if (!quietVerify) printf("   ¥n");
        AES_Close(&t);
        }

/*
+*****************************************************************************
*
* Function Name: ShowParams
*
* Function:                     Print out the settings of settable parameters
*
* Arguments:            None.
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void ShowParams(void)
        {
        int saveDebug=debug;
        testData t;

        debug=0;/* turn off debug output */
        memset(t.ki.keyMaterial,'0',sizeof(t.ki.keyMaterial));

        printf("(keyLen,numRounds): ");
        for (t.keySize=KEY_BITS_0;t.keySize<=MAX_KEY_BITS;t.keySize+=STEP_KEY_BITS)
                {
                if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial) != TRUE)
                        FatalError("Error parsing key during ShowParam","");
                printf("  (%d,%d)",t.keySize,t.ki.numRounds);
                }
        printf("¥n");
```

```
        debug=saveDebug;
        }

/*
+*****************************************************************************
*
* Function Name: ParseArgFile
*
* Function:                     Parse commands from argument file
*
* Arguments:            fName           =       name of file to read
*                                       argList         =       list of ptrs to fill in
*                                       maxArgCnt       =       size of argList
*
* Return:                       None.
*
* Notes:'/' and ';' are comment to end of line characters in the file
*                       This function is used to allow a "custom" set of switches to
*                       be automatically read from a file at startup.
*
-*****************************************************************************/
int ParseArgFile(CONST char *fName,char *argList[],int maxArgCnt)
        {
        static char buf[1024];
        static int  bufCnt=0;           /* current # chars in buf */

        int i,j,k,argCnt;
        char line[256];
        FILE *f=fopen(fName,"rt");

        if (f == NULL) return 0;
        if (debug) printf("Reading args from file %s: ",fName);

        for (argCnt=0;argCnt<maxArgCnt;)
                {       /* read in arg file one line at a time */
                memset(line,0,sizeof(line));
                if (fgets(line,sizeof(line)-4,f) == NULL)
                        break;
                for (i=0;line[i];i++)    /* ignore comments to end of line */
                        if ((line[i]=='/') || (line[i]==';'))
                                { line[i]=line[i+1]=0; break; }
                for (i=0;line[i];)              /* parse out tokens */
                        {
                        for (j=i;line[j];j++)    /* skip leading whitespace */
                                if (line[j] > ' ') break;
                        if (line[j]==0) break;
                        for (k=j;line[k];k++)
                                if (line[k] <= ' ') break;
                        /* now j..k-1 defines a token */
                        if (k-j+1 > (int)(sizeof(buf) - bufCnt))
                                FatalError("Arg file too large: %s",line);
                        if (argCnt >= maxArgCnt)
```

```
                        break;
                    memcpy(buf+bufCnt,line+j,k-j);
                    buf[bufCnt+k-j]=0;        /* terminate the token */
                    if (debug) printf(" %s",buf+bufCnt);
                    argList[argCnt++]=buf+bufCnt;
                    bufCnt+=k-j+1;
                    i=k;                                /* skip to next token */
                    }
                }
        fclose(f);
        if (debug) printf("¥n");
        return argCnt;
        }

/*
+***************************************************************************
*
* Function Name: GiveHelp
*
* Function:                 Print out list of command line switches
*
* Arguments:            None.
*
* Return:                   None.
*
* Notes:
*
-***************************************************************************/
void GiveHelp(void)
        {
        printf("Syntax:   TST2FISH [options]¥n"
                "Purpose:  Generate/validate AES Twofish code and files¥n"
                "Options:  -INN    ==> set sanity check loop to NN¥n"
                "          -m      ==> do full MCT generation¥n"
                "          -pPath  ==> set file path¥n"
                "          -s      ==> set initial random seed based on time¥n"
                "          -sNN    ==> set initial random seed to NN¥n"
                "          -tNN    ==> time performance using NN iterations¥n"
                "          -v      ==> validate files, don't generate them¥n",
                MAX_ROUNDS
               );
        exit(1);
        }

#ifdef TEST_EXTERN
void    Test_Extern(void);
#endif

void ShowHex(FILE *f,CONST void *p,int bCnt,CONST char *name)
        {
        int i;
```

```
        fprintf(f,"    ;%s:",name);
        for (i=0;i<bCnt;i++)
                {
                if ((i % 8) == 0)
                        fprintf(f,"¥n¥t.byte¥t");
                else
                        fprintf(f,",");
                fprintf(f,"0%02Xh",((BYTE *)p)[i]);
                }
        fprintf(f,"¥n");
        }

/* output a formatted 6805 test vector include file */
void Debug6805(void)
        {
        int i,j;
        testData t;
        FILE *f;

        ClearTestData(&t);
        t.keySize=128;

        f=stdout;
        cipherInit(&t.ci,MODE_ECB,NULL);
        makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial);

        for (i=0;i<4;i++)       /* make sure it all fits in 256 bytes */
                {
                reKey(&t.ki);
                blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct);
                fprintf(f,"; Twofish vector #%d¥n",i+1);
                ShowHex(f,&t.keySize,1,"Key Size");
                ShowHex(f,t.ki.key32,16,"Key");
                ShowHex(f,t.pt,BLOCK_SIZE/8,"Plaintext");
                ShowHex(f,t.ct,BLOCK_SIZE/8,"Ciphertext");
                for (j=0;j<16;j++)
                        ((BYTE *)t.ki.key32)[j] = t.pt[j] ^ t.ct[j];
                memcpy(t.pt,t.ct,sizeof(t.pt));
                fprintf(f,";-------------------------------------------------------¥n");
                }
        fprintf(f,"¥n¥t.byte 0¥t;end of list¥n");
        fclose(f);
        }

/*
+*****************************************************************************
*       Main AES test function
-*****************************************************************************/
int main(int argc,char *argv[])
        {
#define MAX_ARGS 40
        int i,j,k,argCnt,testCnt=32;
```

```
        int doTableTest=1;
        int doIntermediate=0;
        DWORD randSeed=0x12345678;
        char *argList[MAX_ARGS];
        char *moduleName=moduleDescription;

        i=1;                    /* make sure LittleEndian is defined correctly */
        if (b0(i) != 1)
                FatalError("LittleEndian defined incorrectly","");
        if ((ALIGN32) && (k == 2))
                FatalError("Cannot enable ALIGN32 in 16-bit mode¥n","");

#if ((MCT_INNER != 10000) || (MCT_OUTER != 400))
#error  MCT loop counts incorrect!
#endif

        argCnt=ParseArgFile("TST2FISH.CFG",argList,MAX_ARGS);      /* read parameter file */
        for (i=1;(i<argc) && (argCnt<MAX_ARGS);i++)                                    /* append command
line */
                argList[argCnt++]=argv[i];

        for (i=0;i<argCnt;i++)    /* parse command line arguments */
                {
                if (argList[i][0] == '-')
                        switch (toupper(argList[i][1]))
                                {
                                case 'D':
                                        if (argList[i][2])
                                                debug=atoi(argList[i]+2);
                                        else
                                                debug=1;
                                        break;
                                case 'F':
                                        switch (toupper(argList[i][2]))
                                                {
                                                case 'L':       FMT_LOG         =       1;
                                                                testCnt         =
        0;      break;
                                                case 'B':       CLKS_BYTE       =
        ~CLKS_BYTE;     break;
                                                case 'M':       CLK_MHZ         =
        atoi(argList[i]+3);     break;
                                                }
                                        break;
                                case '?':
                                case 'H':
                                        GiveHelp();
                                        break;
                                case 'A':
                                        if (argList[i][2])
                                                useAsm=atoi(argList[i]+2);
                                        else
```

```
                                        useAsm=7;               /* enable everything in ASM
*/
                                    break;
                                case 'I':
                                    doIntermediate=1;
                                    break;
                                case 'L':
                                    testCnt = atoi(argList[i]+2);
                                    break;
                                case 'M':                               /* do FULL MCT generation */
                                    mctInner = MCT_INNER;
                                    mctOuter = MCT_OUTER;
                                    break;
                                case 'P':
                                    for (j=0;j<sizeof(filePath)-4;j++)
                                        if ((filePath[j]=argList[i][j+2]) == 0)
                                            break;
                                    filePath[j]=filePath[j+1]=0;
#ifdef _M_IX86   /* DOS/Win specific filePath stuff */
                                    if ((j) && (filePath[j-1] != ':') && (filePath[j-1] != '¥¥'))
                                        filePath[j]='¥¥';       /* append backslash to
filePath */
#endif
                                    break;
                                case 'S':
                                    if (argList[i][2])
                                        randSeed = atol(argList[i]+2);
                                    else
                                        randSeed=(DWORD) time(NULL);      /* randomize */
                                    break;
                                case 'T':
                                    if (argList[i][2])
                                        timeIterCnt = atoi(argList[i]+2);
                                    else
                                        timeIterCnt = 32;
                                    break;
                                case 'V':
                                    verify=1;       /* don't generate files.  Read&verify them */
                                    if (argList[i][2]=='+')
                                        verbose=1;
                                    if (argList[i][2]=='-')
                                        doTableTest=0;
                                    if (toupper(argList[i][2])=='Q')
                                        quietVerify=1;
                                    break;
                                case '6':
                                    Debug6805();
                                    exit(1);
                                    break;
                                }
                        else
                                GiveHelp();
```

```
                }

#ifdef USE_ASM
        if (useAsm & 7)  moduleName="Assembler ";
#endif

        printf("%s%d.%s: %s%s [%ld bytes,%s].¥n",CompilerName,8*sizeof(int),
#ifdef USE_ASM
                    (get_cpu_type() == 5) ? "Pentium" : "Pro/II",
#else
                         (sizeof(int) == 2) ? "x86" : "CPU???",
#endif
                    moduleName,modeString,TwofishCodeSize(),
#if LittleEndian
                    "little-endian"
#else
                    "big-endian"
#endif
                   );

        SetRand(randSeed);                                      /* init pseudorandom generator for
testing */

        if (testCnt)
                AES_Sanity_Check(testCnt);              /* test API compliance, self-consistency */

        if ((timeIterCnt) && (!verify))
                {
                TimeOps(timeIterCnt);
                exit(0);
                }

#ifdef TEST_EXTERN
        Test_Extern();
        exit(0);
#endif

        if (doIntermediate)
                {
                AES_Test_Intermediate("ecb_ival.txt"); /* intermediate value test */
                return 0;
                }


        AES_Test_VK("ecb_vk.txt");                      /* Variable key  KAT */
        AES_Test_VT("ecb_vt.txt");                      /* Variable text KAT */
        AES_Test_Intermediate("ecb_ival.txt"); /* intermediate value test */

        if (!quietVerify)
        printf("%s MCT Generation : %d,%d.¥n",
                    ((MCT_INNER == mctInner) && (MCT_OUTER == mctOuter)) ? "Full" : " *** Partial",
                    mctOuter,mctInner);
```

```
        AES_Test_CBC_E_MCT("cbc_e_m.txt");        /* Monte Carlo test for CBC encryption */
        AES_Test_CBC_D_MCT("cbc_d_m.txt");        /* Monte Carlo test for CBC decryption */
        AES_Test_ECB_E_MCT("ecb_e_m.txt");        /* Monte Carlo test for ECB encryption */
        AES_Test_ECB_D_MCT("ecb_d_m.txt");        /* Monte Carlo test for ECB decryption */

        if (doTableTest)
                AES_Test_TBL("ecb_tbl.txt");              /* Table test */
        else
                if (!quietVerify) printf("WARNING: Skipping ecb_tbl.txt verification¥n");

        if (verify)
                printf("*** All files verified OK ***¥n");

        if (timeIterCnt)
                TimeOps(timeIterCnt);

        return 0;
        }
```

Twofish2.c

```
/**************************************************************************
        TWOFISH2.C      -- Optimized C API calls for TWOFISH AES submission

        Submitters:
                Bruce Schneier, Counterpane Systems
                Doug Whiting,   Hi/fn
                John Kelsey,    Counterpane Systems
                Chris Hall,             Counterpane Systems
                David Wagner,   UC Berkeley

        Code Author:            Doug Whiting,   Hi/fn

        Version  1.00           April 1998

        Copyright 1998, Hi/fn and Counterpane Systems.  All rights reserved.

        Notes:
                *       Optimized version
                *       Tab size is set to 4 characters in this file

**************************************************************************/
#include "aes.h"
#include "table.h"

#include <memory.h>
#include <assert.h>

#if   defined(min_key)  && !defined(MIN_KEY)
```

```
#define MIN_KEY         1                       /* toupper() */
#elif defined(part_key) && !defined(PART_KEY)
#define PART_KEY 1
#elif defined(zero_key) && !defined(ZERO_KEY)
#define ZERO_KEY 1
#endif


#ifdef USE_ASM
extern   int     useAsm;                        /* ok to use ASM code? */

typedef  int cdecl CipherProc
   (cipherInstance *cipher, keyInstance *key,BYTE *input,int inputLen,BYTE *outBuffer);
typedef int      cdecl KeySetupProc(keyInstance *key);

extern CipherProc        *blockEncrypt_86;      /* ptr to ASM functions */
extern CipherProc        *blockDecrypt_86;
extern KeySetupProc      *reKey_86;
extern DWORD             cdecl TwofishAsmCodeSize(void);
#endif

/*
+*****************************************************************************
*                       Constants/Macros/Tables
-*****************************************************************************/

#define         CONST                                   /* help syntax from C++, NOP here */

CONST           fullSbox MDStab;        /* not actually const.  Initialized ONE time */
int                     needToBuildMDS=1;               /* is MDStab initialized yet? */

#define         BIG_TAB         0

#if BIG_TAB
BYTE            bigTab[4][256][256];    /* pre-computed S-box */
#endif

/* number of rounds for various key sizes:  128, 192, 256 */
/* (ignored for now in optimized code!) */
CONST int       numRounds[4]= {0,ROUNDS_128,ROUNDS_192,ROUNDS_256};

#if REENTRANT
#define         _sBox_    key->sBox8x32
#else
static          fullSbox _sBox_;        /* permuted MDStab based on keys */
#endif
#define _sBox8_(N) (((BYTE *) _sBox_) + (N)*256)

/*------- see what level of S-box precomputation we need to do -----*/
#if   defined(ZERO_KEY)
#define MOD_STRING      "(Zero S-box keying)"
#define Fe32_128(x,R)    ¥
```

```
        (       MDStab[0][p8(01)[p8(02)[_b(x,R  )]^b0(SKEY[1])]^b0(SKEY[0])] ^     ¥
                MDStab[1][p8(11)[p8(12)[_b(x,R+1)]^b1(SKEY[1])]^b1(SKEY[0])] ^     ¥
                MDStab[2][p8(21)[p8(22)[_b(x,R+2)]^b2(SKEY[1])]^b2(SKEY[0])] ^     ¥
                MDStab[3][p8(31)[p8(32)[_b(x,R+3)]^b3(SKEY[1])]^b3(SKEY[0])] )
#define Fe32_192(x,R)    ¥
        (       MDStab[0][p8(01)[p8(02)[p8(03)[_b(x,R  )]^b0(SKEY[2])]^b0(SKEY[1])]^b0(SKEY[0])] ^ ¥
                MDStab[1][p8(11)[p8(12)[p8(13)[_b(x,R+1)]^b1(SKEY[2])]^b1(SKEY[1])]^b1(SKEY[0])] ^ ¥
                MDStab[2][p8(21)[p8(22)[p8(23)[_b(x,R+2)]^b2(SKEY[2])]^b2(SKEY[1])]^b2(SKEY[0])] ^ ¥
                MDStab[3][p8(31)[p8(32)[p8(33)[_b(x,R+3)]^b3(SKEY[2])]^b3(SKEY[1])]^b3(SKEY[0])] )
#define Fe32_256(x,R)    ¥
        (
        MDStab[0][p8(01)[p8(02)[p8(03)[p8(04)[_b(x,R  )]^b0(SKEY[3])]^b0(SKEY[2])]^b0(SKEY[1])]^b0(SK
EY[0])] ^ ¥

        MDStab[1][p8(11)[p8(12)[p8(13)[p8(14)[_b(x,R+1)]^b1(SKEY[3])]^b1(SKEY[2])]^b1(SKEY[1])]^b1(SK
EY[0])] ^ ¥

        MDStab[2][p8(21)[p8(22)[p8(23)[p8(24)[_b(x,R+2)]^b2(SKEY[3])]^b2(SKEY[2])]^b2(SKEY[1])]^b2(SK
EY[0])] ^ ¥

        MDStab[3][p8(31)[p8(32)[p8(33)[p8(34)[_b(x,R+3)]^b3(SKEY[3])]^b3(SKEY[2])]^b3(SKEY[1])]^b3(SK
EY[0])] )

#define GetSboxKey       DWORD SKEY[4];   /* local copy */ ¥
                                          memcpy(SKEY,key->sboxKeys,sizeof(SKEY));
/*-------------------------------------------------------------------*/
#elif defined(MIN_KEY)
#define MOD_STRING       "(Minimal keying)"
#define Fe32_(x,R)(MDStab[0][p8(01)[_sBox8_(0)[_b(x,R  )]] ^ b0(SKEY0)] ^ ¥
                                  MDStab[1][p8(11)[_sBox8_(1)[_b(x,R+1)]] ^ b1(SKEY0)] ^ ¥
                                  MDStab[2][p8(21)[_sBox8_(2)[_b(x,R+2)]] ^ b2(SKEY0)] ^ ¥
                                  MDStab[3][p8(31)[_sBox8_(3)[_b(x,R+3)]] ^ b3(SKEY0)])
#define sbSet(N,i,J,v) { _sBox8_(N)[i+J] = v; }
#define GetSboxKey       DWORD SKEY0      = key->sboxKeys[0]               /* local copy */
/*-------------------------------------------------------------------*/
#elif defined(PART_KEY)
#define MOD_STRING       "(Partial keying)"
#define Fe32_(x,R)(MDStab[0][_sBox8_(0)[_b(x,R  )]] ^ ¥
                                  MDStab[1][_sBox8_(1)[_b(x,R+1)]] ^ ¥
                                  MDStab[2][_sBox8_(2)[_b(x,R+2)]] ^ ¥
                                  MDStab[3][_sBox8_(3)[_b(x,R+3)]])
#define sbSet(N,i,J,v) { _sBox8_(N)[i+J] = v; }
#define GetSboxKey
/*-------------------------------------------------------------------*/
#else   /* default is FULL_KEY */
#ifndef FULL_KEY
#define FULL_KEY 1
#endif
#if BIG_TAB
#define TAB_STR          " (Big table)"
#else
#define TAB_STR
```

```
#endif
#ifdef COMPILE_KEY
#define  MOD_STRING       "(Compiled subkeys)" TAB_STR
#else
#define  MOD_STRING       "(Full keying)" TAB_STR
#endif
/* Fe32_ does a full S-box + MDS lookup.  Need to #define _sBox_ before use.
   Note that we "interleave" 0,1, and 2,3 to avoid cache bank collisions
   in optimized assembly language.
*/
#define  Fe32_(x,R) (_sBox_[0][2*_b(x,R  )] ^ _sBox_[0][2*_b(x,R+1)+1] ^     ¥
                                    _sBox_[2][2*_b(x,R+2)] ^ _sBox_[2][2*_b(x,R+3)+1])
                /* set a single S-box value, given the input byte */
#define sbSet(N,i,J,v) { _sBox_[N&2][2*i+(N&1)+2*J]=MDStab[N][v]; }
#define  GetSboxKey
#endif

CONST           char *moduleDescription  ="Optimized C ";
CONST           char *modeString         =MOD_STRING;


/* macro(s) for debugging help */
#define         CHECK_TABLE             0               /* nonzero --> compare against "slow" table */
#define         VALIDATE_PARMS  0               /* disable for full speed */

#include"debug.h"                               /* debug display macros */

/* end of debug macros */

#ifdef GetCodeSize
extern DWORD Here(DWORD x);                     /* return caller's address! */
DWORD TwofishCodeStart(void) { return Here(0); }
#endif

/*
+*****************************************************************************
*
* Function Name: TableOp
*
* Function:                     Handle table use checking
*
* Arguments:            op      =       what to do      (see TAB_* defns in AES.H)
*
* Return:                       TRUE --> done (for TAB_QUERY)
*
* Notes: This routine is for use in generating the tables KAT file.
*               For this optimized version, we don't actually track table usage,
*               since it would make the macros incredibly ugly.  Instead we just
*               run for a fixed number of queries and then say we're done.
*
-*****************************************************************************/
int TableOp(int op)
```

```
        {
        static int queryCnt=0;

        switch (op)
                {
                case TAB_DISABLE:
                        break;
                case TAB_ENABLE:
                        break;
                case TAB_RESET:
                        queryCnt=0;
                        break;
                case TAB_QUERY:
                        queryCnt++;
                        if (queryCnt < TAB_MIN_QUERY)
                                return FALSE;
                }
        return TRUE;
        }


/*
+*************************************************************************
*
* Function Name: ParseHexDword
*
* Function:                     Parse ASCII hex nibbles and fill in key/iv dwords
*
* Arguments:            bit                     =       # bits to read
*                                       srcTxt          =       ASCII source
*                                       d                       =       ptr to dwords to fill in
*                                       dstTxt          =       where to make a copy of ASCII source
*                                                               (NULL ok)
*
* Return:                       Zero if no error.  Nonzero --> invalid hex or length
*
* Notes:  Note that the parameter d is a DWORD array, not a byte array.
*       This routine is coded to work both for little-endian and big-endian
*       architectures.  The character stream is interpreted as a LITTLE-ENDIAN
*       byte stream, since that is how the Pentium works, but the conversion
*       happens automatically below.
*
-*************************************************************************/
int ParseHexDword(int bits,CONST char *srcTxt,DWORD *d,char *dstTxt)
        {
        int i;
        char c;
        DWORD b;

        union   /* make sure LittleEndian is defined correctly */
                {
                BYTE  b[4];
```

```
                DWORD d[1];
                } v;
        v.d[0]=1;
        if (v.b[0 ^ ADDR_XOR] != 1)
                return BAD_ENDIAN;              /* make sure compile-time switch is set ok */

#if VALIDATE_PARMS
  #if ALIGN32
        if (((int)d) & 3)
                return BAD_ALIGN32;
  #endif
#endif

        for (i=0;i*32<bits;i++)
                d[i]=0;                                 /* first, zero the field */

        for (i=0;i*4<bits;i++)          /* parse one nibble at a time */
                {                                               /* case out the hexadecimal
characters */
                c=srcTxt[i];
                if (dstTxt) dstTxt[i]=c;
                if ((c >= '0') && (c <= '9'))
                        b=c-'0';
                else if ((c >= 'a') && (c <= 'f'))
                        b=c-'a'+10;
                else if ((c >= 'A') && (c <= 'F'))
                        b=c-'A'+10;
                else
                        return BAD_KEY_MAT;     /* invalid hex character */
                /* works for big and little endian! */
                d[i/8] |= b << (4*((i^1)&7));
                }

        return 0;                                       /* no error */
        }


#if CHECK_TABLE
/*
+*****************************************************************************
*
* Function Name: f32
*
* Function:                     Run four bytes through keyed S-boxes and apply MDS matrix
*
* Arguments:            x                       =       input to f function
*                                       k32                     =       pointer to key dwords
*                                       keyLen          =       total key length (k32 --> keyLey/2
bits)
*
* Return:                       The output of the keyed permutation applied to x.
*
```

```
* Notes:
*       This function is a keyed 32-bit permutation.  It is the major building
*       block for the Twofish round function, including the four keyed 8x8
*       permutations and the 4x4 MDS matrix multiply.  This function is used
*       both for generating round subkeys and within the round function on the
*       block being encrypted.
*
*       This version is fairly slow and pedagogical, although a smartcard would
*       probably perform the operation exactly this way in firmware.   For
*       ultimate performance, the entire operation can be completed with four
*       lookups into four 256x32-bit tables, with three dword xors.
*
*       The MDS matrix is defined in TABLE.H.  To multiply by Mij, just use the
*       macro Mij(x).
*
-*****************************************************************************/
DWORD f32(DWORD x,CONST DWORD *k32,int keyLen)
        {
        BYTE  b[4];

        /* Run each byte thru 8x8 S-boxes, xoring with key byte at each stage. */
        /* Note that each byte goes through a different combination of S-boxes.*/

        *((DWORD *)b) = Bswap(x);/* make b[0] = LSB, b[3] = MSB */
        switch (((keyLen + 63)/64) & 3)
                {
                case 0:         /* 256 bits of key */
                        b[0] = p8(04)[b[0]] ^ b0(k32[3]);
                        b[1] = p8(14)[b[1]] ^ b1(k32[3]);
                        b[2] = p8(24)[b[2]] ^ b2(k32[3]);
                        b[3] = p8(34)[b[3]] ^ b3(k32[3]);
                        /* fall thru, having pre-processed b[0]..b[3] with k32[3] */
                case 3:         /* 192 bits of key */
                        b[0] = p8(03)[b[0]] ^ b0(k32[2]);
                        b[1] = p8(13)[b[1]] ^ b1(k32[2]);
                        b[2] = p8(23)[b[2]] ^ b2(k32[2]);
                        b[3] = p8(33)[b[3]] ^ b3(k32[2]);
                        /* fall thru, having pre-processed b[0]..b[3] with k32[2] */
                case 2:         /* 128 bits of key */
                        b[0] = p8(00)[p8(01)[p8(02)[b[0]] ^ b0(k32[1])] ^ b0(k32[0])];
                        b[1] = p8(10)[p8(11)[p8(12)[b[1]] ^ b1(k32[1])] ^ b1(k32[0])];
                        b[2] = p8(20)[p8(21)[p8(22)[b[2]] ^ b2(k32[1])] ^ b2(k32[0])];
                        b[3] = p8(30)[p8(31)[p8(32)[b[3]] ^ b3(k32[1])] ^ b3(k32[0])];
                }

        /* Now perform the MDS matrix multiply inline. */
        return   ((M00(b[0]) ^ M01(b[1]) ^ M02(b[2]) ^ M03(b[3]))     ) ^
                        ((M10(b[0]) ^ M11(b[1]) ^ M12(b[2]) ^ M13(b[3])) <<  8) ^
                        ((M20(b[0]) ^ M21(b[1]) ^ M22(b[2]) ^ M23(b[3])) << 16) ^
                        ((M30(b[0]) ^ M31(b[1]) ^ M32(b[2]) ^ M33(b[3])) << 24) ;
        }
#endif  /* CHECK_TABLE */
```

```
/*
+*****************************************************************************
*
* Function Name: RS_MDS_encode
*
* Function:                     Use (12,8) Reed-Solomon code over GF(256) to produce
*                                       a key S-box dword from two key material dwords.
*
* Arguments:            k0      =       1st dword
*                                       k1      =       2nd dword
*
* Return:                       Remainder polynomial generated using RS code
*
* Notes:
*       Since this computation is done only once per reKey per 64 bits of key,
*       the performance impact of this routine is imperceptible. The RS code
*       chosen has "simple" coefficients to allow smartcard/hardware implementation
*       without lookup tables.
*
-*****************************************************************************/
DWORD RS_MDS_Encode(DWORD k0,DWORD k1)
        {
        int i,j;
        DWORD r;

        for (i=r=0;i<2;i++)
                {
                r ^= (i) ? k0 : k1;                     /* merge in 32 more key bits */
                for (j=0;j<4;j++)                       /* shift one byte at a time */
                        RS_rem(r);
                }
        return r;
        }


/*
+*****************************************************************************
*
* Function Name: BuildMDS
*
* Function:                     Initialize the MDStab array
*
* Arguments:            None.
*
* Return:                       None.
*
* Notes:
*       Here we precompute all the fixed MDS table.  This only needs to be done
*       one time at initialization, after which the table is "CONST".
*
```

```
-*************************************************************************/
void BuildMDS(void)
        {
        int i;
        DWORD d;
        BYTE m1[2],mX[2],mY[4];

        for (i=0;i<256;i++)
                {
                m1[0]=P8x8[0][i];                       /* compute all the matrix elements */
                mX[0]=(BYTE) Mul_X(m1[0]);
                mY[0]=(BYTE) Mul_Y(m1[0]);

                m1[1]=P8x8[1][i];
                mX[1]=(BYTE) Mul_X(m1[1]);
                mY[1]=(BYTE) Mul_Y(m1[1]);

#undef  Mul_1                                           /* change what the pre-processor does with Mij */
#undef  Mul_X
#undef  Mul_Y
#define Mul_1   m1                                      /* It will now access m01[], m5B[], and mEF[] */
#define Mul_X   mX
#define Mul_Y   mY

#define SetMDS(N)                                               ¥
                b0(d) = M0##N[P_##N##0]; ¥
                b1(d) = M1##N[P_##N##0]; ¥
                b2(d) = M2##N[P_##N##0]; ¥
                b3(d) = M3##N[P_##N##0]; ¥
                MDStab[N][i] = d;

                SetMDS(0);                                      /* fill in the matrix with elements computed
above */
                SetMDS(1);
                SetMDS(2);
                SetMDS(3);
                }
#undef  Mul_1
#undef  Mul_X
#undef  Mul_Y
#define Mul_1   Mx_1                        /* re-enable true multiply */
#define Mul_X   Mx_X
#define Mul_Y   Mx_Y

#if BIG_TAB
        {
        int j,k;
        BYTE *q0,*q1;

        for (i=0;i<4;i++)
                {
                switch (i)
```

```
                        {
                        case 0: q0=p8(01); q1=p8(02);     break;
                        case 1: q0=p8(11); q1=p8(12);     break;
                        case 2: q0=p8(21); q1=p8(22);     break;
                        case 3: q0=p8(31); q1=p8(32);     break;
                        }
                for (j=0;j<256;j++)
                        for (k=0;k<256;k++)
                                bigTab[i][j][k]=q0[q1[k]^j];
                }
        }
#endif

        needToBuildMDS=0;                       /* NEVER modify the table again! */
        }

/*
+*****************************************************************************
*
* Function Name: ReverseRoundSubkeys
*
* Function:                     Reverse order of round subkeys to switch between encrypt/decrypt
*
* Arguments:            key             =       ptr to keyInstance to be reversed
*                                       newDir  =       new direction value
*
* Return:                       None.
*
* Notes:
*       This optimization allows both blockEncrypt and blockDecrypt to use the same
*       "fallthru" switch statement based on the number of rounds.
*       Note that key->numRounds must be even and >= 2 here.
*
-*****************************************************************************/
void ReverseRoundSubkeys(keyInstance *key,BYTE newDir)
        {
        DWORD t0,t1;
        register DWORD *r0=key->subKeys+ROUND_SUBKEYS;
        register DWORD *r1=r0 + 2*key->numRounds - 2;

        for (;r0 < r1;r0+=2,r1-=2)
                {
                t0=r0[0];                       /* swap the order */
                t1=r0[1];
                r0[0]=r1[0];            /* but keep relative order within pairs */
                r0[1]=r1[1];
                r1[0]=t0;
                r1[1]=t1;
                }

        key->direction=newDir;
        }
```

```
/*
+*****************************************************************************
*
* Function Name: Xor256
*
* Function:                     Copy an 8-bit permutation (256 bytes), xoring with a byte
*
* Arguments:            dst             =       where to put result
*                                       src             =       where to get data (can be same asa
dst)
*                                       b               =       byte to xor
*
* Return:                       None
*
* Notes:
*       BorlandC's optimization is terrible!  When we put the code inline,
*       it generates fairly good code in the *following* segment (not in the Xor256
*       code itself).  If the call is made, the code following the call is awful!
*       The penalty is nearly 50%!  So we take the code size hit for inlining for
*       Borland, while Microsoft happily works with a call.
*
-*****************************************************************************/
#if defined(__BORLANDC__)/* do it inline */
#define Xor32(dst,src,i) { ((DWORD *)dst)[i] = ((DWORD *)src)[i] ^ tmpX; }
#define Xor256(dst,src,b)                                       ¥
        {                                                                               ¥
        register DWORD tmpX=0x01010101u * b;¥
        for (i=0;i<64;i+=4)                                             ¥
                { Xor32(dst,src,i  ); Xor32(dst,src,i+1); Xor32(dst,src,i+2); Xor32(dst,src,i+3); } ¥
        }
#else                                           /* do it as a function call */
void Xor256(void *dst,void *src,BYTE b)
        {
        register DWORD   x=b*0x01010101u; /* replicate byte to all four bytes */
        register DWORD *d=(DWORD *)dst;
        register DWORD *s=(DWORD *)src;
#define X_8(N)    { d[N]=s[N] ^ x; d[N+1]=s[N+1] ^ x; }
#define X_32(N)   { X_8(N); X_8(N+2); X_8(N+4); X_8(N+6); }
        X_32(0 ); X_32( 8); X_32(16); X_32(24);   /* all inline */
        d+=32;  /* keep offsets small! */
        s+=32;
        X_32(0 ); X_32( 8); X_32(16); X_32(24);   /* all inline */
        }
#endif

/*
+*****************************************************************************
*
* Function Name: reKey
*
* Function:                     Initialize the Twofish key schedule from key32
```

```
*
* Arguments:            key                     =       ptr to keyInstance to be initialized
*
* Return:                       TRUE on success
*
* Notes:
*       Here we precompute all the round subkeys, although that is not actually
*       required.  For example, on a smartcard, the round subkeys can
*       be generated on-the-fly  using f32()
*
-**************************************************************************/
int reKey(keyInstance *key)
        {
        int             i,j,k64Cnt,keyLen;
        int             subkeyCnt;
        DWORD   A=0,B=0,q;
        DWORD   sKey[MAX_KEY_BITS/64],k32e[MAX_KEY_BITS/64],k32o[MAX_KEY_BITS/64];
        BYTE    L0[256],L1[256]; /* small local 8-bit permutations */

#if VALIDATE_PARMS
  #if ALIGN32
        if (((int)key) & 3)
                return BAD_ALIGN32;
        if ((key->keyLen % 64) || (key->keyLen < MIN_KEY_BITS))
                return BAD_KEY_INSTANCE;
  #endif
#endif

        if (needToBuildMDS)                     /* do this one time only */
                BuildMDS();

#define F32(res,x,k32)   ¥
        {
                                ¥
        DWORD t=x;
                        ¥
        switch (k64Cnt & 3)
                ¥
            {
                        ¥
                case 0:  /* same as 4 */                                                ¥
                                        b0(t)   = p8(04)[b0(t)] ^ b0(k32[3]);           ¥
                                        b1(t)   = p8(14)[b1(t)] ^ b1(k32[3]);           ¥
                                        b2(t)   = p8(24)[b2(t)] ^ b2(k32[3]);           ¥
                                        b3(t)   = p8(34)[b3(t)] ^ b3(k32[3]);           ¥
                                 /* fall thru, having pre-processed t */          ¥
                case 3:          b0(t)   = p8(03)[b0(t)] ^ b0(k32[2]);           ¥
                                        b1(t)   = p8(13)[b1(t)] ^ b1(k32[2]);           ¥
                                        b2(t)   = p8(23)[b2(t)] ^ b2(k32[2]);           ¥
                                        b3(t)   = p8(33)[b3(t)] ^ b3(k32[2]);           ¥
                                 /* fall thru, having pre-processed t */          ¥
                case 2:  /* 128-bit keys (optimize for this case) */      ¥
```

```
                        res=    MDStab[0][p8(01)[p8(02)[b0(t)] ^ b0(k32[1])] ^ b0(k32[0])] ^       ¥
                                MDStab[1][p8(11)[p8(12)[b1(t)] ^ b1(k32[1])] ^ b1(k32[0])] ^
        ¥
                                MDStab[2][p8(21)[p8(22)[b2(t)] ^ b2(k32[1])] ^ b2(k32[0])] ^
        ¥
                                MDStab[3][p8(31)[p8(32)[b3(t)] ^ b3(k32[1])] ^ b3(k32[0])] ;
        ¥
                }
                        ¥
        }


#if !CHECK_TABLE
#if defined(USE_ASM)                            /* only do this if not using assember */
if (!(useAsm & 4))
#endif
#endif
        {
        subkeyCnt = ROUND_SUBKEYS + 2*key->numRounds;
        keyLen=key->keyLen;
        k64Cnt=(keyLen+63)/64;                  /* number of 64-bit key words */
        for (i=0,j=k64Cnt-1;i<k64Cnt;i++,j--)
                {                                                       /* split into even/odd key
dwords */
                k32e[i]=key->key32[2*i  ];
                k32o[i]=key->key32[2*i+1];
                /* compute S-box keys using (12,8) Reed-Solomon code over GF(256) */
                sKey[j]=key->sboxKeys[j]=RS_MDS_Encode(k32e[i],k32o[i]);   /* reverse order */
                }
        }

#ifdef USE_ASM
if (useAsm & 4)
        {
        #if defined(COMPILE_KEY) && defined(USE_ASM)
                key->keySig             = VALID_SIG;                    /* show that we are
initialized */
                key->codeSize   = sizeof(key->compiledCode);    /* set size */
        #endif
        reKey_86(key);
        }
else
#endif
        {
        for (i=q=0;i<subkeyCnt/2;i++,q+=SK_STEP)
                {                                                       /* compute round subkeys for
PHT */
                F32(A,q        ,k32e);          /* A uses even key dwords */
                F32(B,q+SK_BUMP,k32o);          /* B uses odd  key dwords */
                B = ROL(B,8);
                key->subKeys[2*i  ] = A+B;      /* combine with a PHT */
                B = A + 2*B;
```

```
                key->subKeys[2*i+1] = ROL(B,SK_ROTL);
                }
#if !defined(ZERO_KEY)
        switch (keyLen)  /* case out key length for speed in generating S-boxes */
                {
                case 128:
                #if defined(FULL_KEY) || defined(PART_KEY)
#if BIG_TAB
                        #define  one128(N,J)       sbSet(N,i,J,L0[i+J])
                        #define  sb128(N) {                                                    ¥
                                BYTE *qq=bigTab[N][b##N(sKey[1])];        ¥
                                Xor256(L0,qq,b##N(sKey[0]));              ¥
                                for (i=0;i<256;i+=2) { one128(N,0); one128(N,1); } }
#else
                        #define  one128(N,J)       sbSet(N,i,J,p8(N##1)[L0[i+J]]^k0)
                        #define  sb128(N) {                                              ¥
                                Xor256(L0,p8(N##2),b##N(sKey[1]));        ¥
                                { register DWORD k0=b##N(sKey[0]);        ¥
                                for (i=0;i<256;i+=2) { one128(N,0); one128(N,1); } } }
#endif
                #elif defined(MIN_KEY)
                        #define  sb128(N) Xor256(_sBox8_(N),p8(N##2),b##N(sKey[1]))
                #endif
                        sb128(0); sb128(1); sb128(2); sb128(3);
                        break;
                case 192:
                #if defined(FULL_KEY) || defined(PART_KEY)
                        #define one192(N,J) sbSet(N,i,J,p8(N##1)[p8(N##2)[L0[i+J]]^k1]^k0)
                        #define  sb192(N) {                                                    ¥
                                Xor256(L0,p8(N##3),b##N(sKey[2]));        ¥
                                { register DWORD k0=b##N(sKey[0]);        ¥
                                  register DWORD k1=b##N(sKey[1]);        ¥
                                  for (i=0;i<256;i+=2) { one192(N,0); one192(N,1); } } }
                #elif defined(MIN_KEY)
                        #define one192(N,J) sbSet(N,i,J,p8(N##2)[L0[i+J]]^k1)
                        #define  sb192(N) {                                                    ¥
                                Xor256(L0,p8(N##3),b##N(sKey[2]));        ¥
                                { register DWORD k1=b##N(sKey[1]);        ¥
                                  for (i=0;i<256;i+=2) { one192(N,0); one192(N,1); } } }
                #endif
                        sb192(0); sb192(1); sb192(2); sb192(3);
                        break;
                case 256:
                #if defined(FULL_KEY) || defined(PART_KEY)
                        #define one256(N,J) sbSet(N,i,J,p8(N##1)[p8(N##2)[L0[i+J]]^k1]^k0)
                        #define  sb256(N) {
                        ¥
                                Xor256(L1,p8(N##4),b##N(sKey[3]));
        ¥
                                for (i=0;i<256;i+=2) {L0[i  ]=p8(N##3)[L1[i]];            ¥

L0[i+1]=p8(N##3)[L1[i+1]]; }      ¥
```

```
                                Xor256(L0,L0,b##N(sKey[2]));
        ¥
                                { register DWORD k0=b##N(sKey[0]);
        ¥
                                  register DWORD k1=b##N(sKey[1]);
        ¥
                                  for (i=0;i<256;i+=2) { one256(N,0); one256(N,1); } } }
                #elif defined(MIN_KEY)
                        #define one256(N,J) sbSet(N,i,J,p8(N##2)[L0[i+J]]^k1)
                        #define sb256(N) {
                        ¥
                                Xor256(L1,p8(N##4),b##N(sKey[3]));
        ¥
                                for (i=0;i<256;i+=2) {L0[i  ]=p8(N##3)[L1[i]];              ¥
L0[i+1]=p8(N##3)[L1[i+1]]; }      ¥
                                Xor256(L0,L0,b##N(sKey[2]));
        ¥
                                { register DWORD k1=b##N(sKey[1]);
        ¥
                                  for (i=0;i<256;i+=2) { one256(N,0); one256(N,1); } } }
                #endif
                        sb256(0); sb256(1);       sb256(2); sb256(3);
                        break;
                }
#endif
        }

#if CHECK_TABLE                                         /* sanity check  vs. pedagogical code*/
        {
        GetSboxKey;
        for (i=0;i<subkeyCnt/2;i++)
                {
                A = f32(i*SK_STEP        ,k32e,keyLen);   /* A uses even key dwords */
                B = f32(i*SK_STEP+SK_BUMP,k32o,keyLen);   /* B uses odd  key dwords */
                B = ROL(B,8);
                assert(key->subKeys[2*i  ] == A+  B);
                assert(key->subKeys[2*i+1] == ROL(A+2*B,SK_ROTL));
                }
  #if !defined(ZERO_KEY)                    /* any S-boxes to check? */
        for (i=q=0;i<256;i++,q+=0x01010101)
                assert(f32(q,key->sboxKeys,keyLen) == Fe32_(q,0));
  #endif
        }
#endif /* CHECK_TABLE */

        DebugDumpKey(key);

        if (key->direction == DIR_ENCRYPT)
                ReverseRoundSubkeys(key,DIR_ENCRYPT);     /* reverse the round subkey order */

        return TRUE;
```

```
        }
/*
+*****************************************************************************
*
* Function Name: makeKey
*
* Function:                     Initialize the Twofish key schedule
*
* Arguments:            key                     =       ptr to keyInstance to be initialized
*                                       direction       =       DIR_ENCRYPT or DIR_DECRYPT
*                                       keyLen          =       # bits of key text at *keyMaterial
*                                       keyMaterial     =       ptr to hex ASCII chars representing
key bits
*
* Return:                       TRUE on success
*                                       else error code (e.g., BAD_KEY_DIR)
*
* Notes:This parses the key bits from keyMaterial.  Zeroes out unused key bits
*
-*****************************************************************************/
int makeKey(keyInstance *key, BYTE direction, int keyLen,CONST char *keyMaterial)
        {
#if VALIDATE_PARMS                              /* first, sanity check on parameters */
        if (key == NULL)
                return BAD_KEY_INSTANCE;/* must have a keyInstance to initialize */
        if ((direction != DIR_ENCRYPT) && (direction != DIR_DECRYPT))
                return BAD_KEY_DIR;             /* must have valid direction */
        if ((keyLen > MAX_KEY_BITS) || (keyLen < 8) || (keyLen & 0x3F))
                return BAD_KEY_MAT;             /* length must be valid */
        key->keySig = VALID_SIG; /* show that we are initialized */
  #if ALIGN32
        if ((((int)key) & 3) || (((int)key->key32) & 3))
                return BAD_ALIGN32;
  #endif
#endif

        key->direction   = direction;/* set our cipher direction */
        key->keyLen              = (keyLen+63) & ~63;           /* round up to multiple of 64 */
        key->numRounds   = numRounds[(keyLen-1)/64];
        memset(key->key32,0,sizeof(key->key32));  /* zero unused bits */
        key->keyMaterial[MAX_KEY_SIZE]=0; /* terminate ASCII string */

        if ((keyMaterial == NULL) || (keyMaterial[0]==0))
                return TRUE;                    /* allow a "dummy" call */

        if (ParseHexDword(keyLen,keyMaterial,key->key32,key->keyMaterial))
                return BAD_KEY_MAT;

        return reKey(key);                      /* generate round subkeys */
        }
```

```
/*
+*****************************************************************************
*
* Function Name: cipherInit
*
* Function:                     Initialize the Twofish cipher in a given mode
*
* Arguments:            cipher          =       ptr to cipherInstance to be initialized
*                                       mode            =       MODE_ECB, MODE_CBC, or MODE_CFB1
*                                       IV                      =       ptr to hex ASCII test
representing IV bytes
*
* Return:                       TRUE on success
*                                       else error code (e.g., BAD_CIPHER_MODE)
*
-*****************************************************************************/
int cipherInit(cipherInstance *cipher, BYTE mode,CONST char *IV)
        {
        int i;
#if VALIDATE_PARMS                                      /* first, sanity check on parameters */
        if (cipher == NULL)
                return BAD_PARAMS;                      /* must have a cipherInstance to initialize */
        if ((mode != MODE_ECB) && (mode != MODE_CBC) && (mode != MODE_CFB1))
                return BAD_CIPHER_MODE;  /* must have valid cipher mode */
        cipher->cipherSig       =       VALID_SIG;
  #if ALIGN32
        if ((((int)cipher) & 3) || (((int)cipher->IV) & 3) || (((int)cipher->iv32) & 3))
                return BAD_ALIGN32;
  #endif
#endif

        if ((mode != MODE_ECB) && (IV))   /* parse the IV */
                {
                if (ParseHexDword(BLOCK_SIZE,IV,cipher->iv32,NULL))
                        return BAD_IV_MAT;
                for (i=0;i<BLOCK_SIZE/32;i++)     /* make byte-oriented copy for CFB1 */
                        ((DWORD *)cipher->IV)[i] = Bswap(cipher->iv32[i]);
                }

        cipher->mode            =       mode;

        return TRUE;
        }

/*
+*****************************************************************************
*
* Function Name: blockEncrypt
*
* Function:                     Encrypt block(s) of data using Twofish
*
* Arguments:            cipher          =       ptr to already initialized cipherInstance
```

```
*                                       key                     =       ptr to already initialized
keyInstance
*                                       input           =       ptr to data blocks to be encrypted
*                                       inputLen=       # bits to encrypt (multiple of blockSize)
*                                       outBuffer       =       ptr to where to put encrypted blocks
*
* Return:                       # bits ciphered (>= 0)
*                                       else error code (e.g., BAD_CIPHER_STATE, BAD_KEY_MATERIAL)
*
* Notes: The only supported block size for ECB/CBC modes is BLOCK_SIZE bits.
*                 If inputLen is not a multiple of BLOCK_SIZE bits in those modes,
*                 an error BAD_INPUT_LEN is returned.  In CFB1 mode, all block
*                 sizes can be supported.
*
-*************************************************************************/
int blockEncrypt(cipherInstance *cipher, keyInstance *key,CONST BYTE *input,
                                int inputLen, BYTE *outBuffer)
        {
        int   i,n;                                              /* loop counters */
        DWORD x[BLOCK_SIZE/32];                 /* block being encrypted */
        DWORD t0,t1;                                            /* temp variables */
        int       rounds=key->numRounds; /* number of rounds */
        BYTE  bit,bit0,ctBit,carry;             /* temps for CFB */

        /* make local copies of things for faster access */
        int       mode = cipher->mode;
        DWORD sk[TOTAL_SUBKEYS];
        DWORD IV[BLOCK_SIZE/32];

        GetSboxKey;

#if VALIDATE_PARMS
        if ((cipher == NULL) || (cipher->cipherSig != VALID_SIG))
                return BAD_CIPHER_STATE;
        if ((key == NULL) || (key->keySig != VALID_SIG))
                return BAD_KEY_INSTANCE;
        if ((rounds < 2) || (rounds > MAX_ROUNDS) || (rounds&1))
                return BAD_KEY_INSTANCE;
        if ((mode != MODE_CFB1) && (inputLen % BLOCK_SIZE))
                return BAD_INPUT_LEN;
  #if ALIGN32
        if ( (((int)cipher) & 3) || (((int)key      ) & 3) ||
                 (((int)input ) & 3) || (((int)outBuffer) & 3))
                return BAD_ALIGN32;
  #endif
#endif

        if (mode == MODE_CFB1)
                {       /* use recursion here to handle CFB, one block at a time */
                cipher->mode = MODE_ECB; /* do encryption in ECB */
                for (n=0;n<inputLen;n++)
                        {
```

```
                        blockEncrypt(cipher,key,cipher->IV,BLOCK_SIZE,(BYTE *)x);
                        bit0  = 0x80 >> (n & 7);/* which bit position in byte */
                        ctBit = (input[n/8] & bit0) ^ ((((BYTE *) x)[0] & 0x80) >> (n&7));
                        outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) | ctBit;
                        carry = ctBit >> (7 - (n&7));
                        for (i=BLOCK_SIZE/8-1;i>=0;i--)
                                {
                                bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;
                }

        /* here for ECB, CBC modes */
        if (key->direction != DIR_ENCRYPT)
                ReverseRoundSubkeys(key,DIR_ENCRYPT);     /* reverse the round subkey order */

#ifdef USE_ASM
        if ((useAsm & 1) && (inputLen))
  #ifdef COMPILE_KEY
                if (key->keySig == VALID_SIG)
                        return ((CipherProc
*)(key->encryptFuncPtr))(cipher,key,input,inputLen,outBuffer);
  #else
                return (*blockEncrypt_86)(cipher,key,input,inputLen,outBuffer);
  #endif
#endif
        /* make local copy of subkeys for speed */
        memcpy(sk,key->subKeys,sizeof(DWORD)*(ROUND_SUBKEYS+2*rounds));
        if (mode == MODE_CBC)
                BlockCopy(IV,cipher->iv32)
        else
                IV[0]=IV[1]=IV[2]=IV[3]=0;

        for (n=0;n<inputLen;n+=BLOCK_SIZE,input+=BLOCK_SIZE/8,outBuffer+=BLOCK_SIZE/8)
                {
#ifdef DEBUG
                DebugDump(input,"¥n",-1,0,0,0,1);
                if (cipher->mode == MODE_CBC)
                        DebugDump(cipher->iv32,"",IV_ROUND,0,0,0,0);
#endif
#define LoadBlockE(N)   x[N]=Bswap(((DWORD *)input)[N]) ^ sk[INPUT_WHITEN+N] ^ IV[N]
                LoadBlockE(0);   LoadBlockE(1);   LoadBlockE(2);   LoadBlockE(3);
                DebugDump(x,"",0,0,0,0,0);
#define EncryptRound(K,R,id)      ¥
                        t0          = Fe32##id(x[K  ],0);                                   ¥
                        t1          = Fe32##id(x[K^1],3);                                   ¥
                        x[K^3] = ROL(x[K^3],1);                                                 ¥
                        x[K^2]^= t0 +   t1 + sk[ROUND_SUBKEYS+2*(R)  ];    ¥
```

```
                        x[K^3]^= t0 + 2*t1 + sk[ROUND_SUBKEYS+2*(R)+1];    ¥
                        x[K^2] = ROR(x[K^2],1);                                                    ¥
                        DebugDump(x,"",rounds-(R),0,0,1,0);
#define         Encrypt2(R,id)    { EncryptRound(0,R+1,id); EncryptRound(2,R,id); }

#if defined(ZERO_KEY)
                switch (key->keyLen)
                        {
                        case 128:
                                for (i=rounds-2;i>=0;i-=2)
                                        Encrypt2(i,_128);
                                break;
                        case 192:
                                for (i=rounds-2;i>=0;i-=2)
                                        Encrypt2(i,_192);
                                break;
                        case 256:
                                for (i=rounds-2;i>=0;i-=2)
                                        Encrypt2(i,_256);
                                break;
                        }
#else
                Encrypt2(14,_);
                Encrypt2(12,_);
                Encrypt2(10,_);
                Encrypt2( 8,_);
                Encrypt2( 6,_);
                Encrypt2( 4,_);
                Encrypt2( 2,_);
                Encrypt2( 0,_);
#endif

                /* need to do (or undo, depending on your point of view) final swap */
#if LittleEndian
#define StoreBlockE(N)    ((DWORD *)outBuffer)[N]=x[N^2] ^ sk[OUTPUT_WHITEN+N]
#else
#define StoreBlockE(N)    { t0=x[N^2] ^ sk[OUTPUT_WHITEN+N]; ((DWORD *)outBuffer)[N]=Bswap(t0); }
#endif
                StoreBlockE(0);  StoreBlockE(1);  StoreBlockE(2);  StoreBlockE(3);
                if (mode == MODE_CBC)
                        {
                        IV[0]=Bswap(((DWORD *)outBuffer)[0]);
                        IV[1]=Bswap(((DWORD *)outBuffer)[1]);
                        IV[2]=Bswap(((DWORD *)outBuffer)[2]);
                        IV[3]=Bswap(((DWORD *)outBuffer)[3]);
                        }
#ifdef DEBUG
                DebugDump(outBuffer,"",rounds+1,0,0,0,1);
                if (cipher->mode == MODE_CBC)
                        DebugDump(cipher->iv32,"",IV_ROUND,0,0,0,0);
#endif
                }
```

```
        if (mode == MODE_CBC)
                BlockCopy(cipher->iv32,IV);

        return inputLen;
        }

/*
+*****************************************************************************
*
* Function Name: blockDecrypt
*
* Function:                     Decrypt block(s) of data using Twofish
*
* Arguments:            cipher          =       ptr to already initialized cipherInstance
*                                       key                     =       ptr to already initialized
keyInstance
*                                       input           =       ptr to data blocks to be decrypted
*                                       inputLen=       # bits to encrypt (multiple of blockSize)
*                                       outBuffer       =       ptr to where to put decrypted blocks
*
* Return:                       # bits ciphered (>= 0)
*                                       else error code (e.g., BAD_CIPHER_STATE, BAD_KEY_MATERIAL)
*
* Notes: The only supported block size for ECB/CBC modes is BLOCK_SIZE bits.
*                 If inputLen is not a multiple of BLOCK_SIZE bits in those modes,
*                 an error BAD_INPUT_LEN is returned.  In CFB1 mode, all block
*                 sizes can be supported.
*
-*****************************************************************************/
int blockDecrypt(cipherInstance *cipher, keyInstance *key,CONST BYTE *input,
                                int inputLen, BYTE *outBuffer)
        {
        int   i,n;                                              /* loop counters */
        DWORD x[BLOCK_SIZE/32];                 /* block being encrypted */
        DWORD t0,t1;                                            /* temp variables */
        int       rounds=key->numRounds; /* number of rounds */
        BYTE  bit,bit0,ctBit,carry;             /* temps for CFB */

        /* make local copies of things for faster access */
        int       mode = cipher->mode;
        DWORD sk[TOTAL_SUBKEYS];
        DWORD IV[BLOCK_SIZE/32];

        GetSboxKey;

#if VALIDATE_PARMS
        if ((cipher == NULL) || (cipher->cipherSig != VALID_SIG))
                return BAD_CIPHER_STATE;
        if ((key == NULL) || (key->keySig != VALID_SIG))
                return BAD_KEY_INSTANCE;
        if ((rounds < 2) || (rounds > MAX_ROUNDS) || (rounds&1))
```

```
                return BAD_KEY_INSTANCE;
        if ((cipher->mode != MODE_CFB1) && (inputLen % BLOCK_SIZE))
                return BAD_INPUT_LEN;
  #if ALIGN32
        if ( (((int)cipher) & 3) || (((int)key      ) & 3) ||
                 (((int)input)  & 3) || (((int)outBuffer) & 3))
                return BAD_ALIGN32;
  #endif
#endif

        if (cipher->mode == MODE_CFB1)
                {       /* use blockEncrypt here to handle CFB, one block at a time */
                cipher->mode = MODE_ECB; /* do encryption in ECB */
                for (n=0;n<inputLen;n++)
                        {
                        blockEncrypt(cipher,key,cipher->IV,BLOCK_SIZE,(BYTE *)x);
                        bit0  = 0x80 >> (n & 7);
                        ctBit = input[n/8] & bit0;
                        outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) |
                                                        (ctBit ^ ((((BYTE *) x)[0] & 0x80) >> (n&7)));
                        carry = ctBit >> (7 - (n&7));
                        for (i=BLOCK_SIZE/8-1;i>=0;i--)
                                {
                                bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;
                }

        /* here for ECB, CBC modes */
        if (key->direction != DIR_DECRYPT)
                ReverseRoundSubkeys(key,DIR_DECRYPT);     /* reverse the round subkey order */
#ifdef USE_ASM
        if ((useAsm & 2) && (inputLen))
  #ifdef COMPILE_KEY
                if (key->keySig == VALID_SIG)
                        return ((CipherProc
*)(key->decryptFuncPtr))(cipher,key,input,inputLen,outBuffer);
  #else
                return (*blockDecrypt_86)(cipher,key,input,inputLen,outBuffer);
  #endif
#endif
        /* make local copy of subkeys for speed */
        memcpy(sk,key->subKeys,sizeof(DWORD)*(ROUND_SUBKEYS+2*rounds));
        if (mode == MODE_CBC)
                BlockCopy(IV,cipher->iv32)
        else
                IV[0]=IV[1]=IV[2]=IV[3]=0;
```

```
        for (n=0;n<inputLen;n+=BLOCK_SIZE,input+=BLOCK_SIZE/8,outBuffer+=BLOCK_SIZE/8)
                {
                DebugDump(input,"¥n",rounds+1,0,0,0,1);
#define LoadBlockD(N) x[N^2]=Bswap(((DWORD *)input)[N]) ^ sk[OUTPUT_WHITEN+N]
                LoadBlockD(0);   LoadBlockD(1);   LoadBlockD(2);   LoadBlockD(3);

#define DecryptRound(K,R,id)                                                    ¥
                        t0          = Fe32##id(x[K  ],0);                       ¥
                        t1          = Fe32##id(x[K^1],3);                       ¥
                        DebugDump(x,"",(R)+1,0,0,1,0);                          ¥
                        x[K^2] = ROL (x[K^2],1);                                ¥
                        x[K^2]^= t0 +   t1 + sk[ROUND_SUBKEYS+2*(R)  ];    ¥
                        x[K^3]^= t0 + 2*t1 + sk[ROUND_SUBKEYS+2*(R)+1];    ¥
                        x[K^3] = ROR (x[K^3],1);                                ¥

#define         Decrypt2(R,id)    { DecryptRound(2,R+1,id); DecryptRound(0,R,id); }

#if defined(ZERO_KEY)
                switch (key->keyLen)
                        {
                        case 128:
                                for (i=rounds-2;i>=0;i-=2)
                                        Decrypt2(i,_128);
                                break;
                        case 192:
                                for (i=rounds-2;i>=0;i-=2)
                                        Decrypt2(i,_192);
                                break;
                        case 256:
                                for (i=rounds-2;i>=0;i-=2)
                                        Decrypt2(i,_256);
                                break;
                        }
#else
                {
                Decrypt2(14,_);
                Decrypt2(12,_);
                Decrypt2(10,_);
                Decrypt2( 8,_);
                Decrypt2( 6,_);
                Decrypt2( 4,_);
                Decrypt2( 2,_);
                Decrypt2( 0,_);
                }
#endif
                DebugDump(x,"",0,0,0,0,0);
                if (cipher->mode == MODE_ECB)
                        {
#if LittleEndian
#define StoreBlockD(N)    ((DWORD *)outBuffer)[N] = x[N] ^ sk[INPUT_WHITEN+N]
#else
#define StoreBlockD(N)    { t0=x[N]^sk[INPUT_WHITEN+N]; ((DWORD *)outBuffer)[N] = Bswap(t0); }
```

```
#endif
                        StoreBlockD(0);  StoreBlockD(1);  StoreBlockD(2);  StoreBlockD(3);
#undef  StoreBlockD
                        DebugDump(outBuffer,"",-1,0,0,0,1);
                        continue;
                        }
                else
                        {
#define StoreBlockD(N)  x[N]   ^= sk[INPUT_WHITEN+N] ^ IV[N];     ¥
                                                IV[N]    = Bswap(((DWORD *)input)[N]);     ¥
                                                ((DWORD *)outBuffer)[N] = Bswap(x[N]);
                        StoreBlockD(0);  StoreBlockD(1);  StoreBlockD(2);  StoreBlockD(3);
#undef  StoreBlockD
                        DebugDump(outBuffer,"",-1,0,0,0,1);
                        }
                }
        if (mode == MODE_CBC)     /* restore iv32 to cipher */
                BlockCopy(cipher->iv32,IV)

        return inputLen;
        }

#ifdef GetCodeSize
DWORD TwofishCodeSize(void)
        {
        DWORD x= Here(0);
#ifdef USE_ASM
        if (useAsm & 3)
                return TwofishAsmCodeSize();
#endif
        return x - TwofishCodeStart();
        };
#endif
```

2.　TwofishEC のソースコード

4つのヘッダーファイル、AES.H　DEBUG.H　PLATFORM.H　TABLE.H は、前と同じですので省略します。

Stdafx.h

```
////////////////////////////////////////////////////////////////////////////
// stdafx.h : 標準のシステムインクルードファイルのインクルードファイル、または
// 参照回数が多く、かつあまり変更されない、プロジェクト専用のインクルードファイル
// を記述します。
//

#pragma once
```

```
#ifndef _WIN32_WINNT            // Windows XP 以降のバージョンに固有の機能の使用を許可します。
#define _WIN32_WINNT 0x0501     // これをWindows の他のバージョン向けに適切な値に変更してください。
#endif

#include <stdio.h>
#include <tchar.h>



// TODO: プログラムに必要な追加ヘッダーをここで参照してください。
```

Stdafx.cpp

```
//////////////////////////////////////////////////////////////////////
// stdafx.cpp : 標準インクルードTwofishDC.pch のみを
// 含むソースファイルは、プリコンパイル済みヘッダーになります。
// stdafx.obj にはプリコンパイル済み型情報が含まれます。

#include "stdafx.h"

// TODO: このファイルではなく、STDAFX.H で必要な
// 追加ヘッダーを参照してください。
```

Twofish2.cpp

```
/*****************************************************************************
        TWOFISH2.C      -- Optimized C API calls for TWOFISH AES submission

        Submitters:
                Bruce Schneier, Counterpane Systems
                Doug Whiting,   Hi/fn
                John Kelsey,    Counterpane Systems
                Chris Hall,             Counterpane Systems
                David Wagner,   UC Berkeley

        Code Author:            Doug Whiting,   Hi/fn

        Version  1.00           April 1998

        Copyright 1998, Hi/fn and Counterpane Systems.  All rights reserved.

        Notes:
                *       Optimized version
                *       Tab size is set to 4 characters in this file

*****************************************************************************/
#include "stdafx.h"
```

```
#include"aes.h"
#include"table.h"

#include<memory.h>
#include<assert.h>

#define GetCodeSize //Uyama

#if   defined(min_key)  && !defined(MIN_KEY)
#define  MIN_KEY          1                       /* toupper() */
#elif defined(part_key) && !defined(PART_KEY)
#define  PART_KEY 1
#elif defined(zero_key) && !defined(ZERO_KEY)
#define  ZERO_KEY 1
#endif


#ifdef USE_ASM
extern   int     useAsm;                         /* ok to use ASM code? */

typedef  int cdecl CipherProc
   (cipherInstance *cipher, keyInstance *key,BYTE *input,int inputLen,BYTE *outBuffer);
typedef int      cdecl KeySetupProc(keyInstance *key);

extern CipherProc        *blockEncrypt_86;       /* ptr to ASM functions */
extern CipherProc        *blockDecrypt_86;
extern KeySetupProc      *reKey_86;
extern DWORD             cdecl TwofishAsmCodeSize(void);
#endif

/*
+*****************************************************************************
*                         Constants/Macros/Tables
-*****************************************************************************/

#define          CONST                                  /* help syntax from C++, NOP here */

CONST            fullSbox MDStab;          /* not actually const.  Initialized ONE time */
int                      needToBuildMDS=1;              /* is MDStab initialized yet? */

#define          BIG_TAB          0

#if BIG_TAB
BYTE             bigTab[4][256][256];      /* pre-computed S-box */
#endif

/* number of rounds for various key sizes:  128, 192, 256 */
/* (ignored for now in optimized code!) */
CONST int        numRounds[4]= {0,ROUNDS_128,ROUNDS_192,ROUNDS_256};

#if REENTRANT
#define          _sBox_    key->sBox8x32
```

```
#else
static          fullSbox _sBox_;        /* permuted MDStab based on keys */
#endif
#define _sBox8_(N) (((BYTE *) _sBox_) + (N)*256)

/*------- see what level of S-box precomputation we need to do -----*/
#if   defined(ZERO_KEY)
#define MOD_STRING      "(Zero S-box keying)"
#define Fe32_128(x,R)   ¥
        (       MDStab[0][p8(01)[p8(02)[_b(x,R  )]^b0(SKEY[1])]^b0(SKEY[0])] ^     ¥
                MDStab[1][p8(11)[p8(12)[_b(x,R+1)]^b1(SKEY[1])]^b1(SKEY[0])] ^     ¥
                MDStab[2][p8(21)[p8(22)[_b(x,R+2)]^b2(SKEY[1])]^b2(SKEY[0])] ^     ¥
                MDStab[3][p8(31)[p8(32)[_b(x,R+3)]^b3(SKEY[1])]^b3(SKEY[0])] )
#define Fe32_192(x,R)   ¥
        (       MDStab[0][p8(01)[p8(02)[p8(03)[_b(x,R  )]^b0(SKEY[2])]^b0(SKEY[1])]^b0(SKEY[0])] ^ ¥
                MDStab[1][p8(11)[p8(12)[p8(13)[_b(x,R+1)]^b1(SKEY[2])]^b1(SKEY[1])]^b1(SKEY[0])] ^ ¥
                MDStab[2][p8(21)[p8(22)[p8(23)[_b(x,R+2)]^b2(SKEY[2])]^b2(SKEY[1])]^b2(SKEY[0])] ^ ¥
                MDStab[3][p8(31)[p8(32)[p8(33)[_b(x,R+3)]^b3(SKEY[2])]^b3(SKEY[1])]^b3(SKEY[0])] )
#define Fe32_256(x,R)   ¥
        (
        MDStab[0][p8(01)[p8(02)[p8(03)[p8(04)[_b(x,R  )]^b0(SKEY[3])]^b0(SKEY[2])]^b0(SKEY[1])]^b0(SK
EY[0])] ^ ¥

        MDStab[1][p8(11)[p8(12)[p8(13)[p8(14)[_b(x,R+1)]^b1(SKEY[3])]^b1(SKEY[2])]^b1(SKEY[1])]^b1(SK
EY[0])] ^ ¥

        MDStab[2][p8(21)[p8(22)[p8(23)[p8(24)[_b(x,R+2)]^b2(SKEY[3])]^b2(SKEY[2])]^b2(SKEY[1])]^b2(SK
EY[0])] ^ ¥

        MDStab[3][p8(31)[p8(32)[p8(33)[p8(34)[_b(x,R+3)]^b3(SKEY[3])]^b3(SKEY[2])]^b3(SKEY[1])]^b3(SK
EY[0])] )

#define GetSboxKey      DWORD SKEY[4];  /* local copy */ ¥
                                        memcpy(SKEY,key->sboxKeys,sizeof(SKEY));
/*-----------------------------------------------------------------*/
#elif defined(MIN_KEY)
#define MOD_STRING      "(Minimal keying)"
#define Fe32_(x,R)(MDStab[0][p8(01)[_sBox8_(0)[_b(x,R  )]] ^ b0(SKEY0)] ^ ¥
                                    MDStab[1][p8(11)[_sBox8_(1)[_b(x,R+1)]] ^ b1(SKEY0)] ^ ¥
                                    MDStab[2][p8(21)[_sBox8_(2)[_b(x,R+2)]] ^ b2(SKEY0)] ^ ¥
                                    MDStab[3][p8(31)[_sBox8_(3)[_b(x,R+3)]] ^ b3(SKEY0)])
#define sbSet(N,i,J,v) { _sBox8_(N)[i+J] = v; }
#define GetSboxKey      DWORD SKEY0     = key->sboxKeys[0]              /* local copy */
/*-----------------------------------------------------------------*/
#elif defined(PART_KEY)
#define MOD_STRING      "(Partial keying)"
#define Fe32_(x,R)(MDStab[0][_sBox8_(0)[_b(x,R  )]] ^ ¥
                                    MDStab[1][_sBox8_(1)[_b(x,R+1)]] ^ ¥
                                    MDStab[2][_sBox8_(2)[_b(x,R+2)]] ^ ¥
                                    MDStab[3][_sBox8_(3)[_b(x,R+3)]])
#define sbSet(N,i,J,v) { _sBox8_(N)[i+J] = v; }
#define GetSboxKey
```

```
/*-------------------------------------------------------------------*/
#else   /* default is FULL_KEY */
#ifndef FULL_KEY
#define FULL_KEY 1
#endif
#if BIG_TAB
#define TAB_STR         " (Big table)"
#else
#define TAB_STR
#endif
#ifdef COMPILE_KEY
#define MOD_STRING      "(Compiled subkeys)" TAB_STR
#else
#define MOD_STRING      "(Full keying)" TAB_STR
#endif
/* Fe32_ does a full S-box + MDS lookup.  Need to #define _sBox_ before use.
   Note that we "interleave" 0,1, and 2,3 to avoid cache bank collisions
   in optimized assembly language.
*/
#define Fe32_(x,R) (_sBox_[0][2*_b(x,R  )] ^ _sBox_[0][2*_b(x,R+1)+1] ^     ¥
                                    _sBox_[2][2*_b(x,R+2)] ^ _sBox_[2][2*_b(x,R+3)+1])
                /* set a single S-box value, given the input byte */
#define sbSet(N,i,J,v) { _sBox_[N&2][2*i+(N&1)+2*J]=MDStab[N][v]; }
#define GetSboxKey
#endif

CONST           char *moduleDescription  ="Optimized C ";
CONST           char *modeString         =MOD_STRING;


/* macro(s) for debugging help */
#define         CHECK_TABLE             0               /*nonzero --> compare against "slow" table*/
#define         VALIDATE_PARMS  0               /* disable for full speed */

#include"debug.h"                               /* debug display macros */

/* end of debug macros */

#ifdef GetCodeSize
extern DWORD Here(DWORD x);                     /* return caller's address! */
DWORD TwofishCodeStart(void) { return Here(0); }
#endif

/*
+*****************************************************************************
*
* Function Name: TableOp
*
* Function:                     Handle table use checking
*
* Arguments:            op      =       what to do      (see TAB_* defns in AES.H)
*
```

```
* Return:                      TRUE --> done (for TAB_QUERY)
*
* Notes: This routine is for use in generating the tables KAT file.
*                For this optimized version, we don't actually track table usage,
*                since it would make the macros incredibly ugly.  Instead we just
*                run for a fixed number of queries and then say we're done.
*
-*****************************************************************************/
int TableOp(int op)
        {
        static int queryCnt=0;

        switch (op)
                {
                case TAB_DISABLE:
                        break;
                case TAB_ENABLE:
                        break;
                case TAB_RESET:
                        queryCnt=0;
                        break;
                case TAB_QUERY:
                        queryCnt++;
                        if (queryCnt < TAB_MIN_QUERY)
                                return FALSE;
                }
        return TRUE;
        }


/*
+*****************************************************************************
*
* Function Name: ParseHexDword
*
* Function:                     Parse ASCII hex nibbles and fill in key/iv dwords
*
* Arguments:            bit                     =       # bits to read
*                                       srcTxt          =       ASCII source
*                                       d                       =       ptr to dwords to fill in
*                                       dstTxt          =       where to make a copy of ASCII source
*                                                               (NULL ok)
*
* Return:                       Zero if no error.  Nonzero --> invalid hex or length
*
* Notes:  Note that the parameter d is a DWORD array, not a byte array.
*       This routine is coded to work both for little-endian and big-endian
*       architectures.  The character stream is interpreted as a LITTLE-ENDIAN
*       byte stream, since that is how the Pentium works, but the conversion
*       happens automatically below.
*
-*****************************************************************************/
```

```
int ParseHexDword(int bits,CONST char *srcTxt,DWORD *d,char *dstTxt)
        {
        int i;
        char c;
        DWORD b;

        union    /* make sure LittleEndian is defined correctly */
                {
                BYTE  b[4];
                DWORD d[1];
                } v;
        v.d[0]=1;
        if (v.b[0 ^ ADDR_XOR] != 1)
                return BAD_ENDIAN;               /* make sure compile-time switch is set ok */

#if VALIDATE_PARMS
  #if ALIGN32
        if (((int)d) & 3)
                return BAD_ALIGN32;
  #endif
#endif

        for (i=0;i*32<bits;i++)
                d[i]=0;                                  /* first, zero the field */

        for (i=0;i*4<bits;i++)           /* parse one nibble at a time */
                {                                                /* case out the hexadecimal
characters */
                c=srcTxt[i];
                if (dstTxt) dstTxt[i]=c;
                if ((c >= '0') && (c <= '9'))
                        b=c-'0';
                else if ((c >= 'a') && (c <= 'f'))
                        b=c-'a'+10;
                else if ((c >= 'A') && (c <= 'F'))
                        b=c-'A'+10;
                else
                        return BAD_KEY_MAT;      /* invalid hex character */
                /* works for big and little endian! */
                d[i/8] |= b << (4*((i^1)&7));
                }

        return 0;                                        /* no error */
        }


#if CHECK_TABLE
/*
+*****************************************************************************
*
* Function Name: f32
*
```

```
* Function:                    Run four bytes through keyed S-boxes and apply MDS matrix
*
* Arguments:            x                       =       input to f function
*                                       k32                     =       pointer to key dwords
*                                       keyLen          =       total key length (k32 --> keyLey/2
bits)
*
* Return:                       The output of the keyed permutation applied to x.
*
* Notes:
*       This function is a keyed 32-bit permutation.  It is the major building
*       block for the Twofish round function, including the four keyed 8x8
*       permutations and the 4x4 MDS matrix multiply.  This function is used
*       both for generating round subkeys and within the round function on the
*       block being encrypted.
*
*       This version is fairly slow and pedagogical, although a smartcard would
*       probably perform the operation exactly this way in firmware.   For
*       ultimate performance, the entire operation can be completed with four
*       lookups into four 256x32-bit tables, with three dword xors.
*
*       The MDS matrix is defined in TABLE.H.  To multiply by Mij, just use the
*       macro Mij(x).
*
-*****************************************************************************/
DWORD f32(DWORD x,CONST DWORD *k32,int keyLen)
        {
        BYTE  b[4];

        /* Run each byte thru 8x8 S-boxes, xoring with key byte at each stage. */
        /* Note that each byte goes through a different combination of S-boxes.*/

        *((DWORD *)b) = Bswap(x);/* make b[0] = LSB, b[3] = MSB */
        switch (((keyLen + 63)/64) & 3)
                {
                case 0:         /* 256 bits of key */
                        b[0] = p8(04)[b[0]] ^ b0(k32[3]);
                        b[1] = p8(14)[b[1]] ^ b1(k32[3]);
                        b[2] = p8(24)[b[2]] ^ b2(k32[3]);
                        b[3] = p8(34)[b[3]] ^ b3(k32[3]);
                        /* fall thru, having pre-processed b[0]..b[3] with k32[3] */
                case 3:         /* 192 bits of key */
                        b[0] = p8(03)[b[0]] ^ b0(k32[2]);
                        b[1] = p8(13)[b[1]] ^ b1(k32[2]);
                        b[2] = p8(23)[b[2]] ^ b2(k32[2]);
                        b[3] = p8(33)[b[3]] ^ b3(k32[2]);
                        /* fall thru, having pre-processed b[0]..b[3] with k32[2] */
                case 2:         /* 128 bits of key */
                        b[0] = p8(00)[p8(01)[p8(02)[b[0]] ^ b0(k32[1])] ^ b0(k32[0])];
                        b[1] = p8(10)[p8(11)[p8(12)[b[1]] ^ b1(k32[1])] ^ b1(k32[0])];
                        b[2] = p8(20)[p8(21)[p8(22)[b[2]] ^ b2(k32[1])] ^ b2(k32[0])];
                        b[3] = p8(30)[p8(31)[p8(32)[b[3]] ^ b3(k32[1])] ^ b3(k32[0])];
```

```
                }

        /* Now perform the MDS matrix multiply inline. */
        return  ((M00(b[0]) ^ M01(b[1]) ^ M02(b[2]) ^ M03(b[3]))     ) ^
                        ((M10(b[0]) ^ M11(b[1]) ^ M12(b[2]) ^ M13(b[3])) <<  8) ^
                        ((M20(b[0]) ^ M21(b[1]) ^ M22(b[2]) ^ M23(b[3])) << 16) ^
                        ((M30(b[0]) ^ M31(b[1]) ^ M32(b[2]) ^ M33(b[3])) << 24) ;
        }
#endif  /* CHECK_TABLE */


/*
+*****************************************************************************
*
* Function Name: RS_MDS_encode
*
* Function:                     Use (12,8) Reed-Solomon code over GF(256) to produce
*                                       a key S-box dword from two key material dwords.
*
* Arguments:            k0      =       1st dword
*                                       k1      =       2nd dword
*
* Return:                       Remainder polynomial generated using RS code
*
* Notes:
*       Since this computation is done only once per reKey per 64 bits of key,
*       the performance impact of this routine is imperceptible. The RS code
*       chosen has "simple" coefficients to allow smartcard/hardware implementation
*       without lookup tables.
*
-*****************************************************************************/
DWORD RS_MDS_Encode(DWORD k0,DWORD k1)
        {
        int i,j;
        DWORD r;

        for (i=r=0;i<2;i++)
                {
                r ^= (i) ? k0 : k1;                     /* merge in 32 more key bits */
                for (j=0;j<4;j++)                       /* shift one byte at a time */
                        RS_rem(r);
                }
        return r;
        }


/*
+*****************************************************************************
*
* Function Name: BuildMDS
*
* Function:                     Initialize the MDStab array
```

```
*
* Arguments:             None.
*
* Return:                        None.
*
* Notes:
*       Here we precompute all the fixed MDS table.  This only needs to be done
*       one time at initialization, after which the table is "CONST".
*
-*****************************************************************************/
void BuildMDS(void)
        {
        int i;
        DWORD d;
        BYTE m1[2],mX[2],mY[4];

        for (i=0;i<256;i++)
                {
                m1[0]=P8x8[0][i];               /* compute all the matrix elements */
                mX[0]=(BYTE) Mul_X(m1[0]);
                mY[0]=(BYTE) Mul_Y(m1[0]);

                m1[1]=P8x8[1][i];
                mX[1]=(BYTE) Mul_X(m1[1]);
                mY[1]=(BYTE) Mul_Y(m1[1]);

#undef  Mul_1                                   /* change what the pre-processor does with Mij */
#undef  Mul_X
#undef  Mul_Y
#define Mul_1   m1                              /* It will now access m01[], m5B[], and mEF[] */
#define Mul_X   mX
#define Mul_Y   mY

#define SetMDS(N)                                               ¥
                b0(d) = MO##N[P_##N##0]; ¥
                b1(d) = M1##N[P_##N##0]; ¥
                b2(d) = M2##N[P_##N##0]; ¥
                b3(d) = M3##N[P_##N##0]; ¥
                MDStab[N][i] = d;

                SetMDS(0);                              /* fill in the matrix with elements computed
above */
                SetMDS(1);
                SetMDS(2);
                SetMDS(3);
                }
#undef  Mul_1
#undef  Mul_X
#undef  Mul_Y
#define Mul_1   Mx_1                    /* re-enable true multiply */
#define Mul_X   Mx_X
#define Mul_Y   Mx_Y
```

```
#if BIG_TAB
        {
        int j,k;
        BYTE *q0,*q1;

        for (i=0;i<4;i++)
                {
                switch (i)
                        {
                        case 0: q0=p8(01); q1=p8(02);    break;
                        case 1: q0=p8(11); q1=p8(12);    break;
                        case 2: q0=p8(21); q1=p8(22);    break;
                        case 3: q0=p8(31); q1=p8(32);    break;
                        }
                for (j=0;j<256;j++)
                        for (k=0;k<256;k++)
                                bigTab[i][j][k]=q0[q1[k]^j];
                }
        }
#endif

        needToBuildMDS=0;                       /* NEVER modify the table again! */
        }

/*
+*****************************************************************************
*
* Function Name: ReverseRoundSubkeys
*
* Function:                     Reverse order of round subkeys to switch between encrypt/decrypt
*
* Arguments:            key             =       ptr to keyInstance to be reversed
*                                       newDir  =       new direction value
*
* Return:                       None.
*
* Notes:
*       This optimization allows both blockEncrypt and blockDecrypt to use the same
*       "fallthru" switch statement based on the number of rounds.
*       Note that key->numRounds must be even and >= 2 here.
*
-*****************************************************************************/
void ReverseRoundSubkeys(keyInstance *key,BYTE newDir)
        {
        DWORD t0,t1;
        register DWORD *r0=key->subKeys+ROUND_SUBKEYS;
        register DWORD *r1=r0 + 2*key->numRounds - 2;

        for (;r0 < r1;r0+=2,r1-=2)
                {
                t0=r0[0];                       /* swap the order */
```

```
                t1=r0[1];
                r0[0]=r1[0];            /* but keep relative order within pairs */
                r0[1]=r1[1];
                r1[0]=t0;
                r1[1]=t1;
                }

        key->direction=newDir;
        }

/*
+*****************************************************************************
*
* Function Name: Xor256
*
* Function:                     Copy an 8-bit permutation (256 bytes), xoring with a byte
*
* Arguments:            dst             =       where to put result
*                                       src             =       where to get data (can be same asa
dst)
*                                       b               =       byte to xor
*
* Return:                       None
*
* Notes:
*       BorlandC's optimization is terrible!  When we put the code inline,
*       it generates fairly good code in the *following* segment (not in the Xor256
*       code itself).  If the call is made, the code following the call is awful!
*       The penalty is nearly 50%!  So we take the code size hit for inlining for
*       Borland, while Microsoft happily works with a call.
*
-*****************************************************************************/
#if defined(__BORLANDC__)/* do it inline */
#define Xor32(dst,src,i) { ((DWORD *)dst)[i] = ((DWORD *)src)[i] ^ tmpX; }
#define  Xor256(dst,src,b)                                      ¥
        {                                                                               ¥
        register DWORD tmpX=0x01010101u * b;¥
        for (i=0;i<64;i+=4)                                             ¥
                { Xor32(dst,src,i  ); Xor32(dst,src,i+1); Xor32(dst,src,i+2); Xor32(dst,src,i+3); } ¥
        }
#else                                           /* do it as a function call */
void Xor256(void *dst,void *src,BYTE b)
        {
        register DWORD   x=b*0x01010101u; /* replicate byte to all four bytes */
        register DWORD *d=(DWORD *)dst;
        register DWORD *s=(DWORD *)src;
#define X_8(N)    { d[N]=s[N] ^ x; d[N+1]=s[N+1] ^ x; }
#define X_32(N)   { X_8(N); X_8(N+2); X_8(N+4); X_8(N+6); }
        X_32(0 ); X_32( 8); X_32(16); X_32(24);    /* all inline */
        d+=32;   /* keep offsets small! */
        s+=32;
        X_32(0 ); X_32( 8); X_32(16); X_32(24);    /* all inline */
```

```
        }
#endif

/*
+*****************************************************************************
*
* Function Name: reKey
*
* Function:                     Initialize the Twofish key schedule from key32
*
* Arguments:            key                     =       ptr to keyInstance to be initialized
*
* Return:                       TRUE on success
*
* Notes:
*       Here we precompute all the round subkeys, although that is not actually
*       required.  For example, on a smartcard, the round subkeys can
*       be generated on-the-fly  using f32()
*
-*****************************************************************************/
int reKey(keyInstance *key)
        {
        int             i,j,k64Cnt,keyLen;
        int             subkeyCnt;
        DWORD   A=0,B=0,q;
        DWORD   sKey[MAX_KEY_BITS/64],k32e[MAX_KEY_BITS/64],k32o[MAX_KEY_BITS/64];
        BYTE    L0[256],L1[256]; /* small local 8-bit permutations */

#if VALIDATE_PARMS
  #if ALIGN32
        if (((int)key) & 3)
                return BAD_ALIGN32;
        if ((key->keyLen % 64) || (key->keyLen < MIN_KEY_BITS))
                return BAD_KEY_INSTANCE;
  #endif
#endif

        if (needToBuildMDS)                     /* do this one time only */
                BuildMDS();

#define F32(res,x,k32)   ¥
        {
                                ¥
        DWORD t=x;
                        ¥
        switch (k64Cnt & 3)
                ¥
            {
                        ¥
                case 0:  /* same as 4 */                                        ¥
                                        b0(t)   = p8(04)[b0(t)] ^ b0(k32[3]);           ¥
                                        b1(t)   = p8(14)[b1(t)] ^ b1(k32[3]);           ¥
```

```
                                        b2(t)    = p8(24)[b2(t)] ^ b2(k32[3]);              ¥
                                        b3(t)    = p8(34)[b3(t)] ^ b3(k32[3]);              ¥
                                 /* fall thru, having pre-processed t */           ¥
                case 3:          b0(t)    = p8(03)[b0(t)] ^ b0(k32[2]);              ¥
                                        b1(t)    = p8(13)[b1(t)] ^ b1(k32[2]);              ¥
                                        b2(t)    = p8(23)[b2(t)] ^ b2(k32[2]);              ¥
                                        b3(t)    = p8(33)[b3(t)] ^ b3(k32[2]);              ¥
                                 /* fall thru, having pre-processed t */           ¥
                case 2:  /* 128-bit keys (optimize for this case) */      ¥
                        res=     MDStab[0][p8(01)[p8(02)[b0(t)] ^ b0(k32[1])] ^ b0(k32[0])] ^       ¥
                                        MDStab[1][p8(11)[p8(12)[b1(t)] ^ b1(k32[1])] ^ b1(k32[0])] ^
        ¥
                                        MDStab[2][p8(21)[p8(22)[b2(t)] ^ b2(k32[1])] ^ b2(k32[0])] ^
        ¥
                                        MDStab[3][p8(31)[p8(32)[b3(t)] ^ b3(k32[1])] ^ b3(k32[0])] ;
        ¥
                }
                                ¥
        }


#if !CHECK_TABLE
#if defined(USE_ASM)                            /* only do this if not using assember */
if (!(useAsm & 4))
#endif
#endif
        {
        subkeyCnt = ROUND_SUBKEYS + 2*key->numRounds;
        keyLen=key->keyLen;
        k64Cnt=(keyLen+63)/64;                  /* number of 64-bit key words */
        for (i=0,j=k64Cnt-1;i<k64Cnt;i++,j--)
                {                                                       /* split into even/odd key
dwords */
                k32e[i]=key->key32[2*i  ];
                k32o[i]=key->key32[2*i+1];
                /* compute S-box keys using (12,8) Reed-Solomon code over GF(256) */
                sKey[j]=key->sboxKeys[j]=RS_MDS_Encode(k32e[i],k32o[i]);   /* reverse order */
                }
        }

#ifdef USE_ASM
if (useAsm & 4)
        {
        #if defined(COMPILE_KEY) && defined(USE_ASM)
                key->keySig             = VALID_SIG;                    /* show that we are
initialized */
                key->codeSize    = sizeof(key->compiledCode);      /* set size */
        #endif
        reKey_86(key);
        }
else
#endif
```

```
        {
        for (i=q=0;i<subkeyCnt/2;i++,q+=SK_STEP)
                {                                                       /* compute round subkeys for
PHT */
                F32(A,q        ,k32e);          /* A uses even key dwords */
                F32(B,q+SK_BUMP,k32o);          /* B uses odd  key dwords */
                B = ROL(B,8);
                key->subKeys[2*i  ] = A+B;      /* combine with a PHT */
                B = A + 2*B;
                key->subKeys[2*i+1] = ROL(B,SK_ROTL);
                }
#if !defined(ZERO_KEY)
        switch (keyLen)  /* case out key length for speed in generating S-boxes */
                {
                case 128:
                #if defined(FULL_KEY) || defined(PART_KEY)
#if BIG_TAB
                        #define  one128(N,J)       sbSet(N,i,J,L0[i+J])
                        #define  sb128(N) {                                                    ¥
                                BYTE *qq=bigTab[N][b##N(sKey[1])];      ¥
                                Xor256(L0,qq,b##N(sKey[0]));            ¥
                                for (i=0;i<256;i+=2) { one128(N,0); one128(N,1); } }
#else
                        #define  one128(N,J)       sbSet(N,i,J,p8(N##1)[L0[i+J]]^k0)
                        #define  sb128(N) {                                              ¥
                                Xor256(L0,p8(N##2),b##N(sKey[1]));      ¥
                                { register DWORD k0=b##N(sKey[0]);      ¥
                                for (i=0;i<256;i+=2) { one128(N,0); one128(N,1); } } }
#endif
                #elif defined(MIN_KEY)
                        #define  sb128(N) Xor256(_sBox8_(N),p8(N##2),b##N(sKey[1]))
                #endif
                        sb128(0); sb128(1); sb128(2); sb128(3);
                        break;
                case 192:
                #if defined(FULL_KEY) || defined(PART_KEY)
                        #define one192(N,J) sbSet(N,i,J,p8(N##1)[p8(N##2)[L0[i+J]]^k1]^k0)
                        #define  sb192(N) {                                                    ¥
                                Xor256(L0,p8(N##3),b##N(sKey[2]));      ¥
                                { register DWORD k0=b##N(sKey[0]);      ¥
                                  register DWORD k1=b##N(sKey[1]);      ¥
                                  for (i=0;i<256;i+=2) { one192(N,0); one192(N,1); } } }
                #elif defined(MIN_KEY)
                        #define one192(N,J) sbSet(N,i,J,p8(N##2)[L0[i+J]]^k1)
                        #define  sb192(N) {                                                    ¥
                                Xor256(L0,p8(N##3),b##N(sKey[2]));      ¥
                                { register DWORD k1=b##N(sKey[1]);      ¥
                                  for (i=0;i<256;i+=2) { one192(N,0); one192(N,1); } } }
                #endif
                        sb192(0); sb192(1); sb192(2); sb192(3);
                        break;
                case 256:
```

```
                #if defined(FULL_KEY) || defined(PART_KEY)
                        #define one256(N,J) sbSet(N,i,J,p8(N##1)[p8(N##2)[L0[i+J]]^k1]^k0)
                        #define  sb256(N) {
                        ¥
                                Xor256(L1,p8(N##4),b##N(sKey[3]));
        ¥
                                for (i=0;i<256;i+=2) {L0[i  ]=p8(N##3)[L1[i]];              ¥

LO[i+1]=p8(N##3)[L1[i+1]]; }      ¥
                                Xor256(L0,L0,b##N(sKey[2]));
        ¥
                                { register DWORD k0=b##N(sKey[0]);
        ¥
                                  register DWORD k1=b##N(sKey[1]);
        ¥
                                  for (i=0;i<256;i+=2) { one256(N,0); one256(N,1); } } }
                #elif defined(MIN_KEY)
                        #define one256(N,J) sbSet(N,i,J,p8(N##2)[L0[i+J]]^k1)
                        #define  sb256(N) {
                        ¥
                                Xor256(L1,p8(N##4),b##N(sKey[3]));
        ¥
                                for (i=0;i<256;i+=2) {L0[i  ]=p8(N##3)[L1[i]];              ¥

LO[i+1]=p8(N##3)[L1[i+1]]; }      ¥
                                Xor256(L0,L0,b##N(sKey[2]));
        ¥
                                { register DWORD k1=b##N(sKey[1]);
        ¥
                                  for (i=0;i<256;i+=2) { one256(N,0); one256(N,1); } } }
                #endif
                        sb256(0); sb256(1);       sb256(2); sb256(3);
                        break;
                }
#endif
        }

#if CHECK_TABLE                                         /* sanity check  vs. pedagogical code*/
        {
        GetSboxKey;
        for (i=0;i<subkeyCnt/2;i++)
                {
                A = f32(i*SK_STEP        ,k32e,keyLen);   /* A uses even key dwords */
                B = f32(i*SK_STEP+SK_BUMP,k32o,keyLen);   /* B uses odd  key dwords */
                B = ROL(B,8);
                assert(key->subKeys[2*i  ] == A+  B);
                assert(key->subKeys[2*i+1] == ROL(A+2*B,SK_ROTL));
                }
  #if !defined(ZERO_KEY)                    /* any S-boxes to check? */
        for (i=q=0;i<256;i++,q+=0x01010101)
                assert(f32(q,key->sboxKeys,keyLen) == Fe32_(q,0));
  #endif
```

```
        }
#endif /* CHECK_TABLE */

        DebugDumpKey(key);

        if (key->direction == DIR_ENCRYPT)
                ReverseRoundSubkeys(key,DIR_ENCRYPT);     /* reverse the round subkey order */

        return TRUE;
        }
/*
+*****************************************************************************
*
* Function Name: makeKey
*
* Function:                     Initialize the Twofish key schedule
*
* Arguments:            key                     =       ptr to keyInstance to be initialized
*                                       direction       =       DIR_ENCRYPT or DIR_DECRYPT
*                                       keyLen          =       # bits of key text at *keyMaterial
*                                       keyMaterial     =       ptr to hex ASCII chars representing
key bits
*
* Return:                       TRUE on success
*                                       else error code (e.g., BAD_KEY_DIR)
*
* Notes:This parses the key bits from keyMaterial.  Zeroes out unused key bits
*
-*****************************************************************************/
int makeKey(keyInstance *key, BYTE direction, int keyLen,CONST char *keyMaterial)
        {
#if VALIDATE_PARMS                              /* first, sanity check on parameters */
        if (key == NULL)
                return BAD_KEY_INSTANCE;/* must have a keyInstance to initialize */
        if ((direction != DIR_ENCRYPT) && (direction != DIR_DECRYPT))
                return BAD_KEY_DIR;             /* must have valid direction */
        if ((keyLen > MAX_KEY_BITS) || (keyLen < 8) || (keyLen & 0x3F))
                return BAD_KEY_MAT;             /* length must be valid */
        key->keySig = VALID_SIG; /* show that we are initialized */
  #if ALIGN32
        if ((((int)key) & 3) || (((int)key->key32) & 3))
                return BAD_ALIGN32;
  #endif
#endif

        key->direction   = direction;/* set our cipher direction */
        key->keyLen              = (keyLen+63) & ~63;             /* round up to multiple of 64 */
        key->numRounds   = numRounds[(keyLen-1)/64];
        memset(key->key32,0,sizeof(key->key32));  /* zero unused bits */
        key->keyMaterial[MAX_KEY_SIZE]=0; /* terminate ASCII string */

        if ((keyMaterial == NULL) || (keyMaterial[0]==0))
```

```
                return TRUE;                    /* allow a "dummy" call */

        if (ParseHexDword(keyLen,keyMaterial,key->key32,key->keyMaterial))
                return BAD_KEY_MAT;

        return reKey(key);                      /* generate round subkeys */
        }


/*
+*****************************************************************************
*
* Function Name: cipherInit
*
* Function:                     Initialize the Twofish cipher in a given mode
*
* Arguments:            cipher          =       ptr to cipherInstance to be initialized
*                                       mode            =       MODE_ECB, MODE_CBC, or MODE_CFB1
*                                       IV                      =       ptr to hex ASCII test
representing IV bytes
*
* Return:                       TRUE on success
*                                       else error code (e.g., BAD_CIPHER_MODE)
*
-*****************************************************************************/
int cipherInit(cipherInstance *cipher, BYTE mode,CONST char *IV)
        {
        int i;
#if VALIDATE_PARMS                              /* first, sanity check on parameters */
        if (cipher == NULL)
                return BAD_PARAMS;              /* must have a cipherInstance to initialize */
        if ((mode != MODE_ECB) && (mode != MODE_CBC) && (mode != MODE_CFB1))
                return BAD_CIPHER_MODE;  /* must have valid cipher mode */
        cipher->cipherSig       =       VALID_SIG;
  #if ALIGN32
        if ((((int)cipher) & 3) || (((int)cipher->IV) & 3) || (((int)cipher->iv32) & 3))
                return BAD_ALIGN32;
  #endif
#endif

        if ((mode != MODE_ECB) && (IV))   /* parse the IV */
                {
                if (ParseHexDword(BLOCK_SIZE,IV,cipher->iv32,NULL))
                        return BAD_IV_MAT;
                for (i=0;i<BLOCK_SIZE/32;i++)     /* make byte-oriented copy for CFB1 */
                        ((DWORD *)cipher->IV)[i] = Bswap(cipher->iv32[i]);
                }

        cipher->mode            =       mode;

        return TRUE;
        }
```

```
/*
+*****************************************************************************
*
* Function Name: blockEncrypt
*
* Function:                     Encrypt block(s) of data using Twofish
*
* Arguments:            cipher          =       ptr to already initialized cipherInstance
*                                       key                     =       ptr to already initialized
keyInstance
*                                       input           =       ptr to data blocks to be encrypted
*                                       inputLen=       # bits to encrypt (multiple of blockSize)
*                                       outBuffer       =       ptr to where to put encrypted blocks
*
* Return:                       # bits ciphered (>= 0)
*                                       else error code (e.g., BAD_CIPHER_STATE, BAD_KEY_MATERIAL)
*
* Notes: The only supported block size for ECB/CBC modes is BLOCK_SIZE bits.
*                 If inputLen is not a multiple of BLOCK_SIZE bits in those modes,
*                 an error BAD_INPUT_LEN is returned.  In CFB1 mode, all block
*                 sizes can be supported.
*
-*****************************************************************************/
int blockEncrypt(cipherInstance *cipher, keyInstance *key,CONST BYTE *input,
                                int inputLen, BYTE *outBuffer)
        {
        int   i,n;                                              /* loop counters */
        DWORD x[BLOCK_SIZE/32];                 /* block being encrypted */
        DWORD t0,t1;                                            /* temp variables */
        int       rounds=key->numRounds; /* number of rounds */
        BYTE  bit,bit0,ctBit,carry;             /* temps for CFB */

        /* make local copies of things for faster access */
        int       mode = cipher->mode;
        DWORD sk[TOTAL_SUBKEYS];
        DWORD IV[BLOCK_SIZE/32];

        GetSboxKey;

#if VALIDATE_PARMS
        if ((cipher == NULL) || (cipher->cipherSig != VALID_SIG))
                return BAD_CIPHER_STATE;
        if ((key == NULL) || (key->keySig != VALID_SIG))
                return BAD_KEY_INSTANCE;
        if ((rounds < 2) || (rounds > MAX_ROUNDS) || (rounds&1))
                return BAD_KEY_INSTANCE;
        if ((mode != MODE_CFB1) && (inputLen % BLOCK_SIZE))
                return BAD_INPUT_LEN;
  #if ALIGN32
        if ( (((int)cipher) & 3) || (((int)key      ) & 3) ||
                 (((int)input ) & 3) || (((int)outBuffer) & 3))
```

```
                return BAD_ALIGN32;
  #endif
#endif

        if (mode == MODE_CFB1)
                {       /* use recursion here to handle CFB, one block at a time */
                cipher->mode = MODE_ECB; /* do encryption in ECB */
                for (n=0;n<inputLen;n++)
                        {
                        blockEncrypt(cipher,key,cipher->IV,BLOCK_SIZE,(BYTE *)x);
                        bit0  = 0x80 >> (n & 7);/* which bit position in byte */
                        ctBit = (input[n/8] & bit0) ^ ((((BYTE *) x)[0] & 0x80) >> (n&7));
                        outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) | ctBit;
                        carry = ctBit >> (7 - (n&7));
                        for (i=BLOCK_SIZE/8-1;i>=0;i--)
                                {
                                bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;
                }

        /* here for ECB, CBC modes */
        if (key->direction != DIR_ENCRYPT)
                ReverseRoundSubkeys(key,DIR_ENCRYPT);     /* reverse the round subkey order */

#ifdef USE_ASM
        if ((useAsm & 1) && (inputLen))
  #ifdef COMPILE_KEY
                if (key->keySig == VALID_SIG)
                        return ((CipherProc
*)(key->encryptFuncPtr))(cipher,key,input,inputLen,outBuffer);
  #else
                return (*blockEncrypt_86)(cipher,key,input,inputLen,outBuffer);
  #endif
#endif
        /* make local copy of subkeys for speed */
        memcpy(sk,key->subKeys,sizeof(DWORD)*(ROUND_SUBKEYS+2*rounds));
        if (mode == MODE_CBC)
                BlockCopy(IV,cipher->iv32)
        else
                IV[0]=IV[1]=IV[2]=IV[3]=0;

        for (n=0;n<inputLen;n+=BLOCK_SIZE,input+=BLOCK_SIZE/8,outBuffer+=BLOCK_SIZE/8)
                {
#ifdef DEBUG
                DebugDump(input,"¥n",-1,0,0,0,1);
                if (cipher->mode == MODE_CBC)
                        DebugDump(cipher->iv32,"",IV_ROUND,0,0,0,0);
```

```
#endif
#define LoadBlockE(N)   x[N]=Bswap(((DWORD *)input)[N]) ^ sk[INPUT_WHITEN+N] ^ IV[N]
                LoadBlockE(0);   LoadBlockE(1);   LoadBlockE(2);   LoadBlockE(3);
                DebugDump(x,"",0,0,0,0,0);
#define EncryptRound(K,R,id)      ¥
                        t0          = Fe32##id(x[K  ],0);                                   ¥
                        t1          = Fe32##id(x[K^1],3);                                   ¥
                        x[K^3] = ROL(x[K^3],1);                                                 ¥
                        x[K^2]^= t0 +   t1 + sk[ROUND_SUBKEYS+2*(R)  ];    ¥
                        x[K^3]^= t0 + 2*t1 + sk[ROUND_SUBKEYS+2*(R)+1];    ¥
                        x[K^2] = ROR(x[K^2],1);                                                 ¥
                        DebugDump(x,"",rounds-(R),0,0,1,0);
#define         Encrypt2(R,id)    { EncryptRound(0,R+1,id); EncryptRound(2,R,id); }

#if defined(ZERO_KEY)
                switch (key->keyLen)
                        {
                        case 128:
                                for (i=rounds-2;i>=0;i-=2)
                                        Encrypt2(i,_128);
                                break;
                        case 192:
                                for (i=rounds-2;i>=0;i-=2)
                                        Encrypt2(i,_192);
                                break;
                        case 256:
                                for (i=rounds-2;i>=0;i-=2)
                                        Encrypt2(i,_256);
                                break;
                        }
#else
                Encrypt2(14,_);
                Encrypt2(12,_);
                Encrypt2(10,_);
                Encrypt2( 8,_);
                Encrypt2( 6,_);
                Encrypt2( 4,_);
                Encrypt2( 2,_);
                Encrypt2( 0,_);
#endif

                /* need to do (or undo, depending on your point of view) final swap */
#if LittleEndian
#define StoreBlockE(N)    ((DWORD *)outBuffer)[N]=x[N^2] ^ sk[OUTPUT_WHITEN+N]
#else
#define StoreBlockE(N)    { tO=x[N^2] ^ sk[OUTPUT_WHITEN+N]; ((DWORD *)outBuffer)[N]=Bswap(tO); }
#endif
                StoreBlockE(0);  StoreBlockE(1);  StoreBlockE(2);  StoreBlockE(3);
                if (mode == MODE_CBC)
                        {
                        IV[0]=Bswap(((DWORD *)outBuffer)[0]);
                        IV[1]=Bswap(((DWORD *)outBuffer)[1]);
```

```
                        IV[2]=Bswap(((DWORD *)outBuffer)[2]);
                        IV[3]=Bswap(((DWORD *)outBuffer)[3]);
                        }
#ifdef DEBUG
                DebugDump(outBuffer,"",rounds+1,0,0,0,1);
                if (cipher->mode == MODE_CBC)
                        DebugDump(cipher->iv32,"",IV_ROUND,0,0,0,0);
#endif
                }

        if (mode == MODE_CBC)
                BlockCopy(cipher->iv32,IV);

        return inputLen;
        }

/*
+*****************************************************************************
*
* Function Name: blockDecrypt
*
* Function:                     Decrypt block(s) of data using Twofish
*
* Arguments:            cipher          =       ptr to already initialized cipherInstance
*                                       key                     =       ptr to already initialized
keyInstance
*                                       input           =       ptr to data blocks to be decrypted
*                                       inputLen=       # bits to encrypt (multiple of blockSize)
*                                       outBuffer       =       ptr to where to put decrypted blocks
*
* Return:                       # bits ciphered (>= 0)
*                                       else error code (e.g., BAD_CIPHER_STATE, BAD_KEY_MATERIAL)
*
* Notes: The only supported block size for ECB/CBC modes is BLOCK_SIZE bits.
*                 If inputLen is not a multiple of BLOCK_SIZE bits in those modes,
*                 an error BAD_INPUT_LEN is returned.  In CFB1 mode, all block
*                 sizes can be supported.
*
-*****************************************************************************/
int blockDecrypt(cipherInstance *cipher, keyInstance *key,CONST BYTE *input,
                                int inputLen, BYTE *outBuffer)
        {
        int   i,n;                                              /* loop counters */
        DWORD x[BLOCK_SIZE/32];                 /* block being encrypted */
        DWORD t0,t1;                                            /* temp variables */
        int       rounds=key->numRounds; /* number of rounds */
        BYTE  bit,bit0,ctBit,carry;             /* temps for CFB */

        /* make local copies of things for faster access */
        int       mode = cipher->mode;
        DWORD sk[TOTAL_SUBKEYS];
        DWORD IV[BLOCK_SIZE/32];
```

```
        GetSboxKey;

#if VALIDATE_PARMS
        if ((cipher == NULL) || (cipher->cipherSig != VALID_SIG))
                return BAD_CIPHER_STATE;
        if ((key == NULL) || (key->keySig != VALID_SIG))
                return BAD_KEY_INSTANCE;
        if ((rounds < 2) || (rounds > MAX_ROUNDS) || (rounds&1))
                return BAD_KEY_INSTANCE;
        if ((cipher->mode != MODE_CFB1) && (inputLen % BLOCK_SIZE))
                return BAD_INPUT_LEN;
  #if ALIGN32
        if ( (((int)cipher) & 3) || (((int)key      ) & 3) ||
                 (((int)input)  & 3) || (((int)outBuffer) & 3))
                return BAD_ALIGN32;
  #endif
#endif

        if (cipher->mode == MODE_CFB1)
                {       /* use blockEncrypt here to handle CFB, one block at a time */
                cipher->mode = MODE_ECB; /* do encryption in ECB */
                for (n=0;n<inputLen;n++)
                        {
                        blockEncrypt(cipher,key,cipher->IV,BLOCK_SIZE,(BYTE *)x);
                        bit0  = 0x80 >> (n & 7);
                        ctBit = input[n/8] & bit0;
                        outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) |
                                                        (ctBit ^ ((((BYTE *) x)[0] & 0x80) >> (n&7)));
                        carry = ctBit >> (7 - (n&7));
                        for (i=BLOCK_SIZE/8-1;i>=0;i--)
                                {
                                bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;
                }

        /* here for ECB, CBC modes */
        if (key->direction != DIR_DECRYPT)
                ReverseRoundSubkeys(key,DIR_DECRYPT);     /* reverse the round subkey order */
#ifdef USE_ASM
        if ((useAsm & 2) && (inputLen))
  #ifdef COMPILE_KEY
                if (key->keySig == VALID_SIG)
                        return ((CipherProc
*)(key->decryptFuncPtr))(cipher,key,input,inputLen,outBuffer);
  #else
                return (*blockDecrypt_86)(cipher,key,input,inputLen,outBuffer);
```

```
  #endif
#endif
        /* make local copy of subkeys for speed */
        memcpy(sk,key->subKeys,sizeof(DWORD)*(ROUND_SUBKEYS+2*rounds));
        if (mode == MODE_CBC)
                BlockCopy(IV,cipher->iv32)
        else
                IV[0]=IV[1]=IV[2]=IV[3]=0;

        for (n=0;n<inputLen;n+=BLOCK_SIZE,input+=BLOCK_SIZE/8,outBuffer+=BLOCK_SIZE/8)
                {
                DebugDump(input,"¥n",rounds+1,0,0,0,1);
#define LoadBlockD(N) x[N^2]=Bswap(((DWORD *)input)[N]) ^ sk[OUTPUT_WHITEN+N]
                LoadBlockD(0);   LoadBlockD(1);   LoadBlockD(2);   LoadBlockD(3);

#define DecryptRound(K,R,id)                                                    ¥
                        t0          = Fe32##id(x[K  ],0);                       ¥
                        t1          = Fe32##id(x[K^1],3);                       ¥
                        DebugDump(x,"",(R)+1,0,0,1,0);                          ¥
                        x[K^2] = ROL (x[K^2],1);                                ¥
                        x[K^2]^= t0 +   t1 + sk[ROUND_SUBKEYS+2*(R)  ];    ¥
                        x[K^3]^= t0 + 2*t1 + sk[ROUND_SUBKEYS+2*(R)+1];    ¥
                        x[K^3] = ROR (x[K^3],1);                                ¥

#define         Decrypt2(R,id)    { DecryptRound(2,R+1,id); DecryptRound(0,R,id); }

#if defined(ZERO_KEY)
                switch (key->keyLen)
                        {
                        case 128:
                                for (i=rounds-2;i>=0;i-=2)
                                        Decrypt2(i,_128);
                                break;
                        case 192:
                                for (i=rounds-2;i>=0;i-=2)
                                        Decrypt2(i,_192);
                                break;
                        case 256:
                                for (i=rounds-2;i>=0;i-=2)
                                        Decrypt2(i,_256);
                                break;
                        }
#else
                {
                Decrypt2(14,_);
                Decrypt2(12,_);
                Decrypt2(10,_);
                Decrypt2( 8,_);
                Decrypt2( 6,_);
                Decrypt2( 4,_);
                Decrypt2( 2,_);
                Decrypt2( 0,_);
```

```
                }
#endif
                DebugDump(x,"",0,0,0,0,0);
                if (cipher->mode == MODE_ECB)
                        {
#if LittleEndian
#define StoreBlockD(N)    ((DWORD *)outBuffer)[N] = x[N] ^ sk[INPUT_WHITEN+N]
#else
#define StoreBlockD(N)    { t0=x[N]^sk[INPUT_WHITEN+N]; ((DWORD *)outBuffer)[N] = Bswap(t0); }
#endif
                        StoreBlockD(0);  StoreBlockD(1);  StoreBlockD(2);  StoreBlockD(3);
#undef  StoreBlockD
                        DebugDump(outBuffer,"",-1,0,0,0,1);
                        continue;
                        }
                else
                        {
#define StoreBlockD(N)    x[N]    ^= sk[INPUT_WHITEN+N] ^ IV[N];      ¥
                                        IV[N]    = Bswap(((DWORD *)input)[N]);      ¥
                                        ((DWORD *)outBuffer)[N] = Bswap(x[N]);
                        StoreBlockD(0);  StoreBlockD(1);  StoreBlockD(2);  StoreBlockD(3);
#undef  StoreBlockD
                        DebugDump(outBuffer,"",-1,0,0,0,1);
                        }
                }
        if (mode == MODE_CBC)     /* restore iv32 to cipher */
                BlockCopy(cipher->iv32,IV)

        return inputLen;
        }

#ifdef GetCodeSize
DWORD TwofishCodeSize(void)
        {
        DWORD x= Here(0);
#ifdef USE_ASM
        if (useAsm & 3)
                return TwofishAsmCodeSize();
#endif
        return x - TwofishCodeStart();
        };
#endif
```

Twofishec.cpp

```
////////////////////////////////////////////////////
// TwofishEC.cpp : コンソールアプリケーションのエントリポイントを定義します。
//

#include "stdafx.h"
```

```
#include"aes.h"
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<ctype.h>

extern  CONST char *moduleDescription;  /* which module is running */
extern  CONST char *modeString;                 /* which key schedule mode */
extern  CONST int  debugCompile;        /* is external module compiled with debug? */

#if defined(__WATCOMC__) && defined(_M_IX86) && !defined(NO_TIMER)
    DWORD ReadTimeStampCounter(void);
        #pragma aux ReadTimeStampCounter = " db 0Fh,031h" value [eax] modify exact [eax edx] // RDTSC opcode
#endif


/*
+*****************************************************************************
*                       Constants/Macros/Tables
-*****************************************************************************/

typedef struct
        {
        FILE *f;                        /* the file being written/read */
        int  I;                                 /* test number */
        int      keySize;                       /* key size in bits */
        int      gotDebugIO;            /* got any debug IO? */
        BYTE pt[BLOCK_SIZE/8];  /* plaintext */
        BYTE ct[BLOCK_SIZE/8];  /* ciphertext */

        keyInstance    ki;              /* use ki.keyDwords as key bits */
        cipherInstance ci;              /* use ci.iv as iv bits */
        } testData;


static char hexTab[]    =       "0123456789ABCDEF";
char            filePath[80]=   "";

int                     useAsm          =       0;      /* use assembly language */
int                     mctInner=       MCT_INNER/100;
int                     mctOuter=       MCT_OUTER/10;
int                     verify          =       0;      /* set to nonzero to read&verify files */
int                     debug           =       0;      /* debugging mode */
int                     verbose         =       0;      /* verbose output */
int                     quietVerify     =       0;      /* quiet during verify */
int                     timeIterCnt     =       0;      /* how many times to iterate for timing */
DWORD           randBits[64]= {1};      /* use Knuth's additive generator */
int                     randPtr;
testData *      debugTD         = NULL; /* for use with debugIO */
int                     CLKS_BYTE       =       0;      /* use clks/byte? (vs. clks/block) */
```

```
int                     FMT_LOG         =       0;      /* format for log file */
int                     CLK_MHZ         =       200;/* default clock speed */

#define         KEY_BITS_0                      128                     /* first key bit setting to
test */
#define         STEP_KEY_BITS           ((MAX_KEY_BITS-KEY_BITS_0)/2)
/*
static char  hexString[]=
                "0123456789ABCDEFFEDCBA98765432100011223344556677889AABBCCDDEEFF";
*/
static char  hex7String[72];
//=             "12345671234567123456712345671234567123456712345671234567123456712345671";
////////////////////////////////////////
/*
+*****************************************************************************
*                       Functions
-*****************************************************************************/

DWORD Here(DWORD x)
        {
        unsigned int mask=~0U;

        return (* (((DWORD *)&x)-1)) & mask;
        }
extern DWORD TwofishCodeSize(void);

#ifdef USE_ASM
int cdecl get_cpu_type(void);                   /* return CPU type */
#endif

/*
+*****************************************************************************
*
* Function Name: Rand
*
* Function:                     Generate random number
*
* Arguments:            None.
*
* Return:                       New random number.
*
* Notes:                Uses Knuth's additive generator, other magic
*
-*****************************************************************************/
DWORD Rand(void)
        {
        if (randPtr >= 57)
                randPtr = 0;                    /* handle the ptr wrap */

        randBits[randPtr] += randBits[(randPtr < 7) ? randPtr-7+57 : randPtr-7];

        randBits[62]+= randBits[61];
```

```
        randBits[63] = ROL(randBits[63],9) + 0x6F4ED7D0;   /* very long period! */

        return (randBits[randPtr++] ^ randBits[63]) + randBits[62];
        }


/*
+*****************************************************************************
*
* Function Name: SetRand
*
* Function:                     Initialize random number seed
*
* Arguments:            seed    =       new seed value
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void SetRand(DWORD seed)
        {
        int i;
        DWORD x;

        randPtr=0;
        for (i=x=0;i<64;i++)
                {
                randBits[i]=seed;
                x |= seed;              /* keep track of lsb of all entries */
                seed = ROL(seed,11) + 0x12345678;
                }

        if ((x & 1) == 0)       /* insure maximal period by having at least one odd value */
                randBits[0]++;

        for (i=0;i<1000;i++)
                Rand();                 /* run it for a while */

        randBits[63] = Rand();
        randBits[62] = Rand();
        randBits[61] = Rand() | 1;      /* make it odd */
        }


/*
+*****************************************************************************
*
* Function Name: ClearTestData
*
* Function:                     Initialize test data to all zeroes
*
```

```
* Arguments:             t               =       pointer to testData structure
*
* Return:                        None.
*
* Notes:
*
-*****************************************************************************/
void ClearTestData(testData *t)
        {
        t->gotDebugIO=0;
        memset(t->pt,0,BLOCK_SIZE/8);
        memset(t->ct,0,BLOCK_SIZE/8);
        memset(t->ci.iv32,0,BLOCK_SIZE/8);
        memset(t->ki.key32,0,MAX_KEY_BITS/8);
        memset(t->ki.keyMaterial,'0',sizeof(t->ki.keyMaterial));
#if defined(COMPILE_KEY) && defined(USE_ASM)
        t->ki.cSig1=t->ki.cSig2=0;
#endif
        }

/*
+*****************************************************************************
*
* Function Name: FatalError
*
* Function:                      Output a fatal error message and exit
*
* Arguments:             msg             =       fatal error description (printf string)
*                                        msg2    =       2nd parameter to printf msg
*
* Return:                        None.
*
* Notes:
*
-*****************************************************************************/
void FatalError(CONST char *msg,CONST char *msg2)
        {
        printf("¥nFATAL ERROR: ");
        printf(msg,msg2);
        exit(2);
        }


/*
+*****************************************************************************
*
* Function Name: AES_FileIO
*
* Function:                      Output to file or verify file contents vs. string
*
* Arguments:             f               =       opened file
*                                        s               =       string to output/compare
```

```
(NULL-->reset, return)
*                                       errOK    =       do not fatalError on miscompare
*
* Return:                          Zero --> compare ok
*
* Notes:                  On miscompare, FatalError (unless errOK)
*
-*************************************************************************/
int AES_FileIO(FILE *f,CONST char *s,int errOK)
        {
        int  i;
        static int  lineNum=0;
        static int  j=0;
        static char line[516]="";

        if (s == NULL)   /* starting new file */
                {
                line[0]=j=lineNum=0;
                return 0;
                }

        if (!verify)
                {
                fprintf(f,s);
                return 0;
                }

        /* here to verify the file against the string */
        for (i=0;s[i];i++)
                {
                while (line[j] == 0)
                        {
                        lineNum++;
                        if (fgets(line,sizeof(line)-4,f) == NULL)
                                {
                                if ((s[i]=='¥n') && (s[i+1]==0))
                                        {
                                        line[0]=j=0;     /* missing final eol is ok */
                                        return 0;
                                        }
                                FatalError("Unexpected EOF looking for %s",s);
                                }
                        if (verbose) printf(line);
                        j=0;
                        }
                if (s[i] != line[j])
                        {
                        if ((s[i] == '¥n') && ((i==0) || (s[i-1] == '¥n'))) continue; /* blank line skip
*/
                        if (line[j] == '¥n') {j++; continue; }
                        if (!errOK)
                                {
```

```
                                char tmp[1024];
                                sprintf(tmp,"Miscompare at line #%d:¥n%s¥nlooking
for¥n¥n%%s",lineNum,line);
                                FatalError(tmp,s);
                                }
                        line[0]=j=0;    /* let caller re-synch if desired */
                        return 1;               /* return error flag */
                        }
                j++;
                }

        return 0;
        }




/*
+*****************************************************************************
*
* Function Name: DebugIO
*
* Function:                     Output debug string
*
* Arguments:            s       =       string to output
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void DebugIO(CONST char *s)
        {
        if (debugTD)
                {
                AES_FileIO(debugTD->f,s,0);
                debugTD->gotDebugIO=1;
                }
        else
                printf(s);
        }


//////////////////////////////////////////////////////////////////
/****************************************************************/
////////////  uyama
void UY_EC_TF(char *keyfn, char *pt, char *ct){
        FILE *fkey;
        FILE *stream1;
        FILE *stream2;
        char c_mode[8], c_klen[8], pass[128];
        int i,mode,klen;
```

```
        int numclosed;
        testData t;
        int c, block;
        long mesLength; // 平文長（バイト）
        char* bufp;
        unsigned char cstr[BLOCK_SIZE/8+10];                                    // 暗号文
格納場所へのポインタ

        printf("TfEC Start!¥n" );
        /* 鍵を読み出すファイルを開く*/
        if( (fkey  = fopen( keyfn, "rt" )) == NULL ){
                printf( "Can not open key file.¥n");
                return;
        }

        fgets( c_mode, 6, fkey );
        fgets( c_klen, 6, fkey );
        fgets( pass, 127, fkey );
        mode = atoi(c_mode);
        klen = atoi(c_klen);

        fclose(fkey);

    /* 鍵*/
        for(i=0; i<klen/4; i++){
                hex7String[i] = pass[i];
        }
        hex7String[klen/4] = NULL;

        /* 平文を読み出すファイルを開く*/
        if( (stream1  = fopen( pt, "rb" )) == NULL )
                printf( "Can not open plane text file.¥n");
   /* 暗号文を書き込むファイルを開く*/
        if( (stream2 = fopen( ct ,"wb" )) == NULL )
                printf( "Can not open file for encrypted data.¥n" );

////////////////////////////////////////////////////////////////
   // 平文
        fseek(stream1, 0, SEEK_END);
        long filelen = ftell(stream1);
        fseek(stream1,0,0);
        int head = sizeof(long);
        mesLength = filelen + head;

        if (cipherInit(&t.ci,mode,hex7String) != TRUE)
                FatalError("cipherInit error during %s test",""/*fname*/);
        t.keySize=klen;
        ClearTestData(&t);                                      /* start with all zeroes */
        if (makeKey(&t.ki,DIR_ENCRYPT,t.keySize,hex7String/*t.ki.keyMaterial*/) != TRUE)
                FatalError("Error parsing key during %s test",""/*fname*/);

        //暗号化
```

```
	if(mesLength <= BLOCK_SIZE/8){//63が暗号化の作業サイズ　*20＝
		bufp = (char*)new(char[BLOCK_SIZE/8+10]);
		if(bufp == NULL){
			printf("No memory");
			return;
		}
// 平文
		*(long *)bufp = filelen;
		int i = head;
		do{
			c = fgetc(stream1);
			bufp[i]=c;
			i=i+1;
		}while(c!=EOF);
		bufp[i-1]=NULL;
		for(int j=0; j+i<BLOCK_SIZE/8+10; j++){
			bufp[j+i] = NULL;
		}
		mesLength = i-1; // 平文長（バイト）+ head
		memcpy(t.pt,bufp,BLOCK_SIZE/8);
  // 暗号化実行
		if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
			FatalError("blockEncrypt return during %s test","ff.bin"/*fname*/);
		memcpy(cstr,t.ct,BLOCK_SIZE/8);
		fseek(stream2,0,0);
		fwrite( cstr,sizeof(char),BLOCK_SIZE/8, stream2);
	}
	else{
		long rBlen = mesLength;
		bufp = (char*)new(char[BLOCK_SIZE/8]);//63が暗号化の作業サイズ　*20＝
		if(bufp == NULL){
			printf("メモリ不足¥r¥n");
			return;
		}
		(*(long*)(bufp)) = filelen;

		int r = 0;
		do{
		// 平文
			if(r == 0){
				i = head;
				fseek(stream1, r*BLOCK_SIZE/8, 0);//63が暗号化の作業サイズ　*20＝
			}
			if(r > 0 ){
				i = 0;
				fseek(stream1, r*BLOCK_SIZE/8-4, 0);//63が暗号化の作業サイズ　*20＝
			}
			do{
				c = fgetc(stream1);
				bufp[i]=c;
				i=i+1;
			}while((c!=EOF) && (i<=BLOCK_SIZE/8));//63が暗号化の作業サイズ　*20＝
```

```
                        bufp[i-1]=NULL;
                        for(int j=0; j+i<BLOCK_SIZE/8; j++){
                                bufp[j+i] = NULL;
                        }

                        if(rBlen >= BLOCK_SIZE/8){ block = BLOCK_SIZE/8; }//63が暗号化の作業サイズ  *20
=
                        if(rBlen < BLOCK_SIZE/8){ block = rBlen;}

                        memcpy(t.pt,bufp,BLOCK_SIZE/8);
    // 暗号化実行
                        if (blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct) != BLOCK_SIZE)
                                FatalError("blockEncrypt return during %s test","ff.bin"/*fname*/);

        // 暗号文を書き込む
                        memcpy(cstr,t.ct,BLOCK_SIZE/8);
                        fwrite( cstr,sizeof(char),BLOCK_SIZE/8, stream2);
                        r += 1;
                        rBlen -= block;
                }while(rBlen>0);
        }

        fclose(stream1);
        fclose(stream2);

        //numclosed = _fcloseall(); /* 理由は不明だが使えない。*/
        printf("TfEC End!¥n" );
}

////////////////////////////////////////////////////////////////

/*****************************************************************/


/*
+*****************************************************************************
*
* Function Name: GiveHelp
*
* Function:                     Print out list of command line switches
*
* Arguments:            None.
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void GiveHelp(void)
        {
        printf("Syntax:   TST2FISH [options]¥n"
                "Purpose:  Generate/validate AES Twofish code and files¥n"
```

```
                "Options:  -lNN    ==> set sanity check loop to NN¥n"
                "          -m      ==> do full MCT generation¥n"
                "          -pPath  ==> set file path¥n"
                "          -s      ==> set initial random seed based on time¥n"
                "          -sNN    ==> set initial random seed to NN¥n"
                "          -tNN    ==> time performance using NN iterations¥n"
                "          -v      ==> validate files, don't generate them¥n",
                MAX_ROUNDS
                );
        exit(1);
        }

#ifdef TEST_EXTERN
void    Test_Extern(void);
#endif

void ShowHex(FILE *f,CONST void *p,int bCnt,CONST char *name)
        {
        int i;

        fprintf(f,"    ;%s:",name);
        for (i=0;i<bCnt;i++)
                {
                if ((i % 8) == 0)
                        fprintf(f,"¥n¥t.byte¥t");
                else
                        fprintf(f,",");
                fprintf(f,"0%02Xh",((BYTE *)p)[i]);
                }
        fprintf(f,"¥n");
        }

/* output a formatted 6805 test vector include file */
void Debug6805(void)
        {
        int i,j;
        testData t;
        FILE *f;

        ClearTestData(&t);
        t.keySize=128;

        f=stdout;
        cipherInit(&t.ci,MODE_ECB,NULL);
        makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial);

        for (i=0;i<4;i++)       /* make sure it all fits in 256 bytes */
                {
                reKey(&t.ki);
                blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct);
                fprintf(f,"; Twofish vector #%d¥n",i+1);
                ShowHex(f,&t.keySize,1,"Key Size");
```

```
                ShowHex(f,t.ki.key32,16,"Key");
                ShowHex(f,t.pt,BLOCK_SIZE/8,"Plaintext");
                ShowHex(f,t.ct,BLOCK_SIZE/8,"Ciphertext");
                for (j=0;j<16;j++)
                        ((BYTE *)t.ki.key32)[j] = t.pt[j] ^ t.ct[j];
                memcpy(t.pt,t.ct,sizeof(t.pt));
                fprintf(f,";-----------------------------------------------------¥n");
                }
        fprintf(f,"¥n¥t.byte 0¥t;end of list¥n");
        fclose(f);
        }


//////////////////////////////////////////
int main(int argc, char* argv[])
{
#define MAX_ARGS 40
        int i,testCnt=32;
        DWORD randSeed=0x12345678;
        char *moduleName=moduleDescription;

        // 引数チェック
        if( argc != 4 ){                        // 使い方の誤り
                exit(1);
        }

        i=1;                    /* make sure LittleEndian is defined correctly */
        if (b0(i) != 1)
                FatalError("LittleEndian defined incorrectly","");
//      if ((ALIGN32) && (k == 2))
//              FatalError("Cannot enable ALIGN32 in 16-bit mode¥n","");

#if ((MCT_INNER != 10000) || (MCT_OUTER != 400))
#error  MCT loop counts incorrect!
#endif

#ifdef USE_ASM
        if (useAsm & 7)  moduleName="Assembler ";
#endif

        SetRand(randSeed);                                      /* init pseudorandom generator for
testing */

#ifdef TEST_EXTERN
        Test_Extern();
        exit(0);
#endif

        UY_EC_TF(argv[1],argv[2],argv[3]);

        return 0;
        }
```

以上、VC++2005 でソフトを作成するために必要なファイルです。

3.　TwofishDC のソースコード

4つのヘッダーファイル、AES.H　DEBUG.H　PLATFORM.H　TABLE.H は、前と同じですので省略します。

Cpp のファイルのうちで、twofish2.cpp は、TwofishEC のものと同じですので、省略します。

twofishDC.cpp

```
////////////////////////////////////////////////////////////////////
// TwofishDC.cpp : コンソールアプリケーションのエントリポイントを定義します。
//

#include "stdafx.h"

#include"aes.h"
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<ctype.h>

extern   CONST char *moduleDescription;    /* which module is running */
extern   CONST char *modeString;                    /* which key schedule mode */
extern   CONST int  debugCompile;          /* is external module compiled with debug? */

#if defined(__WATCOMC__) && defined(_M_IX86) && !defined(NO_TIMER)
    DWORD ReadTimeStampCounter(void);
        #pragma aux ReadTimeStampCounter = " db 0Fh,031h" value [eax] modify exact [eax edx] // RDTSC opcode
#endif


/*
+*****************************************************************************
*                         Constants/Macros/Tables
-*****************************************************************************/

typedef struct
        {
        FILE *f;                           /* the file being written/read */
        int  I;                                    /* test number */
        int      keySize;                          /* key size in bits */
        int      gotDebugIO;               /* got any debug IO? */
```

```
        BYTE pt[BLOCK_SIZE/8];   /* plaintext */
        BYTE ct[BLOCK_SIZE/8];   /* ciphertext */

        keyInstance    ki;               /* use ki.keyDwords as key bits */
        cipherInstance ci;               /* use ci.iv as iv bits */
        } testData;


static char hexTab[]     =       "0123456789ABCDEF";
char            filePath[80]=    "";

int                     useAsm          =       0;      /* use assembly language */
int                     mctInner=       MCT_INNER/100;
int                     mctOuter=       MCT_OUTER/10;
int                     verify          =       0;      /* set to nonzero to read&verify files */
int                     debug           =       0;      /* debugging mode */
int                     verbose         =       0;      /* verbose output */
int                     quietVerify     =       0;      /* quiet during verify */
int                     timeIterCnt     =       0;      /* how many times to iterate for timing */
DWORD           randBits[64]= {1};       /* use Knuth's additive generator */
int                     randPtr;
testData *      debugTD         = NULL; /* for use with debugIO */
int                     CLKS_BYTE       =       0;      /* use clks/byte? (vs. clks/block) */
int                     FMT_LOG         =       0;      /* format for log file */
int                     CLK_MHZ         =       200;/* default clock speed */

#define         KEY_BITS_0                      128                     /* first key bit setting to
test */
#define         STEP_KEY_BITS           ((MAX_KEY_BITS-KEY_BITS_0)/2)
/*
static char  hexString[]=
                "0123456789ABCDEFFEDCBA98765432100011223344556677889AABBCCDDEEFF";
*/
static char  hex7String[72];// use 64 byte
//=             "12345671234567123456712345671234567123456712345671234567123456712345671";
////////////////////////////////////////
/*
+*****************************************************************************
*                       Functions
-*****************************************************************************/

DWORD Here(DWORD x)
        {
        unsigned int mask=~0U;

        return (* (((DWORD *)&x)-1)) & mask;
        }
extern DWORD TwofishCodeSize(void);

#ifdef USE_ASM
int cdecl get_cpu_type(void);                   /* return CPU type */
#endif
```

```
/*
+--------------------------------------------------------------------------
*
* Function Name: Rand
*
* Function:                     Generate random number
*
* Arguments:            None.
*
* Return:                       New random number.
*
* Notes:                Uses Knuth's additive generator, other magic
*
-*************************************************************************/
DWORD Rand(void)
        {
        if (randPtr >= 57)
                randPtr = 0;                    /* handle the ptr wrap */

        randBits[randPtr] += randBits[(randPtr < 7) ? randPtr-7+57 : randPtr-7];

        randBits[62]+= randBits[61];
        randBits[63] = ROL(randBits[63],9) + 0x6F4ED7D0;  /* very long period! */

        return (randBits[randPtr++] ^ randBits[63]) + randBits[62];
        }


/*
+--------------------------------------------------------------------------
*
* Function Name: SetRand
*
* Function:                     Initialize random number seed
*
* Arguments:            seed    =       new seed value
*
* Return:                       None.
*
* Notes:
*
-*************************************************************************/
void SetRand(DWORD seed)
        {
        int i;
        DWORD x;

        randPtr=0;
        for (i=x=0;i<64;i++)
                {
                randBits[i]=seed;
```

```
                x |= seed;              /* keep track of lsb of all entries */
                seed = ROL(seed,11) + 0x12345678;
                }

        if ((x & 1) == 0)       /* insure maximal period by having at least one odd value */
                randBits[0]++;

        for (i=0;i<1000;i++)
                Rand();                 /* run it for a while */

        randBits[63] = Rand();
        randBits[62] = Rand();
        randBits[61] = Rand() | 1;      /* make it odd */
        }


/*
+*****************************************************************************
*
* Function Name: ClearTestData
*
* Function:                     Initialize test data to all zeroes
*
* Arguments:            t               =       pointer to testData structure
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void ClearTestData(testData *t)
        {
        t->gotDebugIO=0;
        memset(t->pt,0,BLOCK_SIZE/8);
        memset(t->ct,0,BLOCK_SIZE/8);
        memset(t->ci.iv32,0,BLOCK_SIZE/8);
        memset(t->ki.key32,0,MAX_KEY_BITS/8);
        memset(t->ki.keyMaterial,'0',sizeof(t->ki.keyMaterial));
#if defined(COMPILE_KEY) && defined(USE_ASM)
        t->ki.cSig1=t->ki.cSig2=0;
#endif
        }

/*
+*****************************************************************************
*
* Function Name: FatalError
*
* Function:                     Output a fatal error message and exit
*
* Arguments:            msg             =       fatal error description (printf string)
*                                       msg2    =       2nd parameter to printf msg
```

```
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void FatalError(CONST char *msg,CONST char *msg2)
        {
        printf("¥nFATAL ERROR: ");
        printf(msg,msg2);
        exit(2);
        }


/*
+*****************************************************************************
*
* Function Name: AES_FileIO
*
* Function:                     Output to file or verify file contents vs. string
*
* Arguments:            f               =       opened file
*                                       s               =       string to output/compare
(NULL-->reset, return)
*                                       errOK   =       do not fatalError on miscompare
*
* Return:                       Zero --> compare ok
*
* Notes:                  On miscompare, FatalError (unless errOK)
*
-*****************************************************************************/
int AES_FileIO(FILE *f,CONST char *s,int errOK)
        {
        int  i;
        static int  lineNum=0;
        static int  j=0;
        static char line[516]="";

        if (s == NULL)   /* starting new file */
                {
                line[0]=j=lineNum=0;
                return 0;
                }

        if (!verify)
                {
                fprintf(f,s);
                return 0;
                }

        /* here to verify the file against the string */
        for (i=0;s[i];i++)
```

```
                {
                while (line[j] == 0)
                        {
                        lineNum++;
                        if (fgets(line,sizeof(line)-4,f) == NULL)
                                {
                                if ((s[i]=='¥n') && (s[i+1]==0))
                                        {
                                        line[0]=j=0;     /* missing final eol is ok */
                                        return 0;
                                        }
                                FatalError("Unexpected EOF looking for %s",s);
                                }
                        if (verbose) printf(line);
                        j=0;
                        }
                if (s[i] != line[j])
                        {
                        if ((s[i] == '¥n') && ((i==0) || (s[i-1] == '¥n'))) continue; /* blank line skip
*/
                        if (line[j] == '¥n') {j++; continue; }
                        if (!errOK)
                                {
                                char tmp[1024];
                                sprintf(tmp,"Miscompare at line #%d:¥n%s¥nlooking
for¥n¥n%%s",lineNum,line);
                                FatalError(tmp,s);
                                }
                        line[0]=j=0;     /* let caller re-synch if desired */
                        return 1;                /* return error flag */
                        }
                j++;
                }

        return 0;
        }




/*
+*************************************************************************
*
* Function Name: DebugIO
*
* Function:                     Output debug string
*
* Arguments:            s       =       string to output
*
* Return:                       None.
*
* Notes:
```

```
*
-*****************************************************************************/
void DebugIO(CONST char *s)
        {
        if (debugTD)
                {
                AES_FileIO(debugTD->f,s,0);
                debugTD->gotDebugIO=1;
                }
        else
                printf(s);
        }


//////////////////////////////////////
////////////////////////////////////////////////////////////
/*************************************************************/
///////////  uyama
void UY_DC_TF(char *keyfn, char *ct, char *pt){
        FILE *fkey;
        FILE *stream1;
        FILE *stream2;
        char c_mode[8], c_klen[8], pass[128];
        int i,mode,klen;
        testData t;
        int head;
        long lenp;
        int cc, block;
        char* bufc;
        unsigned char ostr[BLOCK_SIZE/8+10];                                    // 暗号文
格納場所へのポインタ

        printf("TfDC Start!¥n" );
        /* 鍵を読み出すファイルを開く*/
        if( (fkey  = fopen( keyfn, "rt" )) == NULL ){
                printf( "Can not open key file.¥n");
                return;
        }
        fgets( c_mode, 6, fkey );
        fgets( c_klen, 6, fkey );
        fgets( pass, 127, fkey );
        mode = atoi(c_mode);
        klen = atoi(c_klen);

        fclose(fkey);

        /* 鍵*/
        for(i=0; i<klen/4; i++){
                hex7String[i] = pass[i];
        }
        hex7String[klen/4] = NULL;
```

```
        /* 平文を書き込むファイルを開く*/
    if( (stream1  = fopen(pt , "wb" )) == NULL )
       printf( "Can not open plane text file.¥n");
    /* 暗号文を読み出すファイルを開く*/
    if( (stream2 = fopen( ct, "rb" )) == NULL )
       printf( "Can not open file for encrypted data.¥n" );

////////////////////////////////////////////////////////////////
    // 暗文
          fseek(stream2, 0, SEEK_END);
          long filelen = ftell(stream2);
          fseek(stream2,0,0);
          head = sizeof(long);

          t.keySize=klen;
          if (cipherInit(&t.ci,mode,hex7String) != TRUE)
                  FatalError("cipherInit error during %s test",""/*fname*/);
          ClearTestData(&t);                                    /* start with all zeroes */
          if (makeKey(&t.ki,DIR_DECRYPT,t.keySize,hex7String/*t.ki.keyMaterial*/) != TRUE)
                  FatalError("Error parsing key during %s test",""/*fname*/);

          //暗号化
          if(filelen <= BLOCK_SIZE/8){//63が暗号化の作業サイズ  *20＝
                  bufc = (char*)new(char[BLOCK_SIZE/8+10]);
                  int i=0;
                  if(bufc == NULL){
                          printf("No memory");
                          return;
                  }
                  do{
                          cc = fgetc(stream2);
                          bufc[i]=cc;
                          i=i+1;
                  }while(cc!=EOF);
                  bufc[i-1]=NULL;
                  for(int j=0; j+i<BLOCK_SIZE/8+10; j++){
                          bufc[j+i] = NULL;
                  }

                  memcpy(t.ct,bufc,BLOCK_SIZE/8);
     // 復号化実行
                  if (blockDecrypt(&t.ci,&t.ki,t.ct,BLOCK_SIZE,t.pt) != BLOCK_SIZE)
                          FatalError("blockDecrypt return during %s test","ff.bin"/*fname*/);
                  memcpy(ostr,t.pt,BLOCK_SIZE/8);
                  lenp = *((long*)ostr);
                  fseek(stream1,0,0);
                  fwrite( ostr+head,sizeof(char),lenp, stream1);
                  fclose(stream1);
          }
          else{
                  long rBlen = filelen;
                  bufc = (char*)new(char[BLOCK_SIZE/8+10]);//63が暗号化の作業サイズ  *20＝
```

```
		if(bufc == NULL){
			printf("メモリ不足¥r¥n");
			return;
		}
		int r = 0;
		do{
			fseek(stream2, r*BLOCK_SIZE/8, 0);//63が暗号化の作業サイズ　*20＝
			i=0;
			do{
				cc = fgetc(stream2);
				bufc[i]=cc;
				i=i+1;
			}while((cc!=EOF) && (i<=BLOCK_SIZE/8));//63が暗号化の作業サイズ　*20＝
			bufc[i-1]=NULL;
			for(int j=0; j+i<BLOCK_SIZE/8+10; j++){
				bufc[j+i] = NULL;
			}

			if(rBlen >= BLOCK_SIZE/8){ block = BLOCK_SIZE/8; }//63が暗号化の作業サイズ　*20
＝
			if(rBlen < BLOCK_SIZE/8){ block = rBlen;}

			memcpy(t.ct,bufc,BLOCK_SIZE/8);
    // 復号化実行
			if (blockDecrypt(&t.ci,&t.ki,t.ct,BLOCK_SIZE,t.pt) != BLOCK_SIZE)
				FatalError("blockDecrypt return during %s test","ff.bin"/*fname*/);
			memcpy(ostr,t.pt,BLOCK_SIZE/8);
        // 復号文を書き込む
			if(r==0){
				int head = sizeof(long);
				lenp = *((long*)ostr);
				int wc = fwrite( ostr+head,sizeof(char),BLOCK_SIZE/8-head, stream1);
				lenp -= BLOCK_SIZE/8-head;
			}
			if(r>=1){
				if(lenp >= BLOCK_SIZE/8){
					fwrite( ostr,sizeof(char),BLOCK_SIZE/8, stream1);
					lenp -= BLOCK_SIZE/8;
				}
				else {
					if(lenp < BLOCK_SIZE/8){
						fwrite( ostr,sizeof(char),lenp, stream1);
						lenp -= lenp;
					}
				}
			}
			r += 1;
			rBlen -= block;
		}while(rBlen>0);
	}

	fclose(stream1);
```

```
        fclose(stream2);

//      _fcloseall();
        printf("TfDC End!¥n" );
}

/*****************************************************************/


/*
+*****************************************************************************
*
* Function Name: GiveHelp
*
* Function:                     Print out list of command line switches
*
* Arguments:            None.
*
* Return:                       None.
*
* Notes:
*
-*****************************************************************************/
void GiveHelp(void)
        {
        printf("Syntax:   TST2FISH [options]¥n"
                    "Purpose:  Generate/validate AES Twofish code and files¥n"
                    "Options:  -INN    ==> set sanity check loop to NN¥n"
                    "          -m      ==> do full MCT generation¥n"
                    "          -pPath  ==> set file path¥n"
                    "          -s      ==> set initial random seed based on time¥n"
                    "          -sNN    ==> set initial random seed to NN¥n"
                    "          -tNN    ==> time performance using NN iterations¥n"
                    "          -v      ==> validate files, don't generate them¥n",
                    MAX_ROUNDS
                   );
        exit(1);
        }

#ifdef TEST_EXTERN
void    Test_Extern(void);
#endif

void ShowHex(FILE *f,CONST void *p,int bCnt,CONST char *name)
        {
        int i;

        fprintf(f,"    ;%s:",name);
        for (i=0;i<bCnt;i++)
                {
                if ((i % 8) == 0)
                        fprintf(f,"¥n¥t.byte¥t");
```

```
                else
                        fprintf(f,",");
                fprintf(f,"0%02Xh",((BYTE *)p)[i]);
                }
        fprintf(f,"¥n");
        }

/* output a formatted 6805 test vector include file */
void Debug6805(void)
        {
        int i,j;
        testData t;
        FILE *f;

        ClearTestData(&t);
        t.keySize=128;

        f=stdout;
        cipherInit(&t.ci,MODE_ECB,NULL);
        makeKey(&t.ki,DIR_ENCRYPT,t.keySize,t.ki.keyMaterial);

        for (i=0;i<4;i++)       /* make sure it all fits in 256 bytes */
                {
                reKey(&t.ki);
                blockEncrypt(&t.ci,&t.ki,t.pt,BLOCK_SIZE,t.ct);
                fprintf(f,"; Twofish vector #%d¥n",i+1);
                ShowHex(f,&t.keySize,1,"Key Size");
                ShowHex(f,t.ki.key32,16,"Key");
                ShowHex(f,t.pt,BLOCK_SIZE/8,"Plaintext");
                ShowHex(f,t.ct,BLOCK_SIZE/8,"Ciphertext");
                for (j=0;j<16;j++)
                        ((BYTE *)t.ki.key32)[j] = t.pt[j] ^ t.ct[j];
                memcpy(t.pt,t.ct,sizeof(t.pt));
                fprintf(f,";-----------------------------------------------------¥n");
                }
        fprintf(f,"¥n¥t.byte 0¥t;end of list¥n");
        fclose(f);
        }


//////////////////////////////////////////
int main(int argc, char* argv[])
{
#define MAX_ARGS 40
        int i,testCnt=32;
        DWORD randSeed=0x12345678;
        char *moduleName=moduleDescription;

        // 引数チェック
        if( argc != 4 ){                        // 使い方の誤り
                exit(1);
        }
```

```
        i=1;                        /* make sure LittleEndian is defined correctly */
        if (b0(i) != 1)
                FatalError("LittleEndian defined incorrectly","");
//      if ((ALIGN32) && (k == 2))
//              FatalError("Cannot enable ALIGN32 in 16-bit mode¥n","");

#if ((MCT_INNER != 10000) || (MCT_OUTER != 400))
#error  MCT loop counts incorrect!
#endif

#ifdef USE_ASM
        if (useAsm & 7)  moduleName="Assembler ";
#endif

        SetRand(randSeed);                                      /* init pseudorandom generator for
testing */

#ifdef TEST_EXTERN
        Test_Extern();
        exit(0);
#endif

        UY_DC_TF(argv[1],argv[2],argv[3]);

        return 0;
        }
```

以上、VC++2005 でソフトを作成するために必要なファイルです。

おわり。　　2012.04.11　　宇山靖政