AES を利用した暗号ソフト

（C）宇山　靖政

2012.04.10

AES 暗号　（Rijndael 暗号）　について

DES が制定されたのは 1977 年でありその後の暗号解読技術の発展、コンピュータの高性能化から、米国商務省標準技術局(NIST)によって新しい暗号技術の選定作業が行われました。米国政府の次世代標準暗号化方式を決めることになり、NIST は DES に代わる次世代の暗号標準として、AES 候補となる暗号方式を全世界から公募しました。世界中から集まった 15 の方式が審査を受けていたが、2000 年 10 月に、ベルギーの暗号開発者 Joan Daemen 氏と Vincent Rijmen 氏が開発した「Rijndael」という方式が選ばれました。このアルゴリズムは、ロイヤリティーフリーで提供されていましたので、利用させていただきました。

この暗号ソフトは、下記の参考文献を使って作りましたので、AES として採用されているものよりも鍵およびブロックの長さの種類が多くなっています。鍵長、ブロック長は、それぞれ 128,160,192,224,256 ビットの 5 通りです。AES のものは、鍵長、ブロック長は、それぞれ、128,192,256 ビットの 3 通りです。

参考文献

The Design of Rijndael: Aes-The Advanced Encryption Standard (Information Security and Cryptography) (ハードカバー)

Joan Daemen

参考文献のミスプリント The Design Of Rijndael ? Errata

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。

1．参考文献にある、AES(Rijndael) のソースコード
2．AesEC のソースコード
3．AesDC 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。

1．参考文献にある、AES(Rijndael)　のソースコード

TestRijn.cpp

```
////////////////////////////////////////////////////////////////////////
// TestRijn.cpp : コンソールアプリケーション用のエントリポイントの定義
//


int BC, KC, ROUNDS;
```

```
#include "stdafx.h"
#include <fstream>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <limits.h>


#define file_len(x) (unsigned long)x

typedef unsigned char word8;
typedef unsigned int word32;

word8 Logtable[256] = {
          0,   0,  25,   1,  50,   2,  26,198,  75,199,  27,104,  51,238,223,   3,
        100,   4,224,  14,  52,141,129,239,  76,113,   8,200,248,105,  28,193,
        125,194,  29,181,249,185,  39,106,  77,228,166,114,154,201,   9,120,
        101,  47,138,   5,  33,  15,225,  36,  18,240,130,  69,  53,147,218,142,
        150,143,219,189,  54,208,206,148,  19,  92,210,241,  64,  70,131,  56,
        102,221,253,  48,191,   6,139,  98,179,  37,226,152,  34,136,145,  16,
        126,110,  72,195,163,182,  30,  66,  58,107,  40,  84,250,133,  61,186,
         43,121,  10,  21,155,159,  94,202,  78,212,172,229,243,115,167,  87,
        175,  88,168,  80,244,234,214,116,  79,174,233,213,231,230,173,232,
         44,215,117,122,235,  22,  11,245,  89,203,  95,176,156,169,  81,160,
        127,  12,246,111,  23,196,  73,236,216,  67,  31,  45,164,118,123,183,
        204,187,  62,  90,251,  96,177,134,  59,  82,161,108,170,  85,  41,157,
        151,178,135,144,  97,190,220,252,188,149,207,205,  55,  63,  91,209,
         83,  57,132,  60,  65,162,109,  71,  20,  42,158,  93,  86,242,211,171,
         68,  17,146,217,  35,  32,  46,137,180,124,184,  38,119,153,227,165,
        103,  74,237,222,197,  49,254,  24,  13,  99,140,128,192,247,112,   7};

word8 Alogtable[256] = {
          1,   3,   5,  15,  17,  51,  85,255,  26,  46,114,150,161,248,  19,  53,
         95,225,  56,  72,216,115,149,164,247,   2,   6,  10,  30,  34,102,170,
        229,  52,  92,228,  55,  89,235,  38,106,190,217,112,144,171,230,  49,
         83,245,   4,  12,  20,  60,  68,204,  79,209,104,184,211,110,178,205,
         76,212,103,169,224,  59,  77,215,  98,166,241,   8,  24,  40,120,136,
        131,158,185,208,107,189,220,127,129,152,179,206,  73,219,118,154,
        181,196,  87,249,  16,  48,  80,240,  11,  29,  39,105,187,214,  97,163,
        254,  25,  43,125,135,146,173,236,  47,113,147,174,233,  32,  96,160,
        251,  22,  58,  78,210,109,183,194,  93,231,  50,  86,250,  21,  63,  65,
        195,  94,226,  61,  71,201,  64,192,  91,237,  44,116,156,191,218,117,
        159,186,213,100,172,239,  42,126,130,157,188,223,122,142,137,128,
        155,182,193,  88,232,  35,101,175,234,  37,111,177,200,  67,197,  84,
        252,  31,  33,  99,165,244,   7,   9,  27,  45,119,153,176,203,  70,202,
         69,207,  74,222,121,139,134,145,168,227,  62,  66,198,  81,243,  14,
         18,  54,  90,238,  41,123,141,140,143,138,133,148,167,242,  13,  23,
         57,  75,221,124,132,151,162,253,  28,  36,108,180,199,  82,246,   1};

word8 S[256] = {
          99,124,119,123,242,107,111,197,  48,   1,103,  43,254,215,171,118,
        202,130,201,125,250,  89,  71,240,173,212,162,175,156,164,114,192,
```

```
        183,253,147, 38, 54, 63,247,204, 52,165,229,241,113,216, 49, 21,
          4,199, 35,195, 24,150,  5,154,  7, 18,128,226,235, 39,178,117,
          9,131, 44, 26, 27,110, 90,160, 82, 59,214,179, 41,227, 47,132,
         83,209,  0,237, 32,252,177, 91,106,203,190, 57, 74, 76, 88,207,
        208,239,170,251, 67, 77, 51,133, 69,249,  2,127, 80, 60,159,168,
         81,163, 64,143,146,157, 56,245,188,182,218, 33, 16,255,243,210,
        205, 12, 19,236, 95,151, 68, 23,196,167,126, 61,100, 93, 25,115,
         96,129, 79,220, 34, 42,144,136, 70,238,184, 20,222, 94, 11,219,
        224, 50, 58, 10, 73,  6, 36, 92,194,211,172, 98,145,149,228,121,
        231,200, 55,109,141,213, 78,169,108, 86,244,234,101,122,174,  8,
        186,120, 37, 46, 28,166,180,198,232,221,116, 31, 75,189,139,138,
        112, 62,181,102, 72,  3,246, 14, 97, 53, 87,185,134,193, 29,158,
        225,248,152, 17,105,217,142,148,155, 30,135,233,206, 85, 40,223,
        140,161,137, 13,191,230, 66,104, 65,153, 45, 15,176, 84,187, 22};

word8 Si[256] = {
         82,  9,106,213, 48, 54,165, 56,191, 64,163,158,129,243,215,251,
        124,227, 57,130,155, 47,255,135, 52,142, 67, 68,196,222,233,203,
         84,123,148, 50,166,194, 35, 61,238, 76,149, 11, 66,250,195, 78,
          8, 46,161,102, 40,217, 36,178,118, 91,162, 73,109,139,209, 37,
        114,248,246,100,134,104,152, 22,212,164, 92,204, 93,101,182,146,
        108,112, 72, 80,253,237,185,218, 94, 21, 70, 87,167,141,157,132,
        144,216,171,  0,140,188,211, 10,247,228, 88,  5,184,179, 69,  6,
        208, 44, 30,143,202, 63, 15,  2,193,175,189,  3,  1, 19,138,107,
         58,145, 17, 65, 79,103,220,234,151,242,207,206,240,180,230,115,
        150,172,116, 34,231,173, 53,133,226,249, 55,232, 28,117,223,110,
         71,241, 26,113, 29, 41,197,137,111,183, 98, 14,170, 24,190, 27,
        252, 86, 62, 75,198,210,121, 32,154,219,192,254,120,205, 90,244,
         31,221,168, 51,136,  7,199, 49,177, 18, 16, 89, 39,128,236, 95,
         96, 81,127,169, 25,181, 74, 13, 45,229,122,159,147,201,156,239,
        160,224, 59, 77,174, 42,245,176,200,235,187, 60,131, 83,153, 97,
         23, 43,  4,126,186,119,214, 38,225,105, 20, 99, 85, 33, 12,125};


word32 RC[30] = {
        0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
        0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,
        0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,
        0xfa,0xef,0xc5};

#define MAXBC 8
#define MAXKC 8
#define MAXROUNDS 14

static word8 shifts[5][4] = {
        0,1,2,3,
        0,1,2,3,
        0,1,2,3,
        0,1,2,4,
        0,1,3,4};

static int numrounds[5][5] = {
```

```
        10,11,12,13,14,
        11,11,12,13,14,
        12,12,12,13,14,
        13,13,13,13,14,
        14,14,14,14,14};

int BC, KC, ROUNDS;

word8 mul(word8 a, word8 b) {
        if(a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
        else return 0;
}

void AddRoundKey(word8 a[4][MAXBC], word8 rk[4][MAXBC]) {
        int i, j;

        for(i = 0; i<4; i++)
                for(j=0;j<BC;j++) a[i][j] ^= rk[i][j];
}

void SubBytes(word8 a[4][MAXBC], word8 box[256]){
        int i,j;

        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = box[a[i][j]] ;
}

void ShiftRows(word8 a[4][MAXBC], word8 d){
        word8 tmp[MAXBC];
        int i,j;

        if(d==0){
                for(i=1;i<4;i++){
                        for(j=0;j<BC;j++)
                                tmp[j] = a[i][(j+shifts[BC-4][i]) % BC];
                        for(j=0;j<BC;j++) a[i][j] = tmp[j];
                }
        }
        else{
                for(i=1;i<4;i++){
                        for(j=0;j<BC;j++)
                                tmp[j] = a[i][(BC+j-shifts[BC-4][i]) % BC];
                        for(j=0;j<BC;j++) a[i][j] = tmp[j];
                }
        }
}

void MixColumns(word8 a[4][MAXBC]){
        word8 b[4][MAXBC];
        int i,j;

        for(j=0;j<BC;j++)
```

```
                for(i=0;i<4;i++)
                        b[i][j] = mul(2,a[i][j])
                                ^ mul(3,a[(i+1) % 4][j])
                                ^ a[(i+2) % 4][j]
                                ^ a[(i+3) % 4][j];
        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = b[i][j];
}

void InvMixColumns(word8 a[4][MAXBC]){
        word8 b[4][MAXBC];
        int i,j;

        for(j=0;j<BC;j++)
        for(i=0;i<4;i++)
                b[i][j] = mul(0xe, a[i][j])
                        ^ mul(0xb, a[(i+1) % 4][j])
                        ^ mul(0xd, a[(i+2) % 4][j])
                        ^ mul(0x9, a[(i+3) % 4][j]);
        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = b[i][j];
}

int KeyExpansion(word8 k[4][MAXKC],
                                word8 W[MAXROUNDS+1][4][MAXBC]){
        int i,j,t,RCpointer = 1;
        word8 tk[4][MAXKC];

        for(j=0;j<KC;j++)
                for(i=0;i<4;i++)
                        tk[i][j] = k[i][j];
        t = 0;

        for(j=0;(j<KC) && (t<(ROUNDS+1)*BC); j++, t++)
                for(i=0;i<4;i++) W[t / BC][i][t % BC] = tk[i][j];

        while(t<(ROUNDS+1)*BC){
                for(i=0;i<4;i++)
                        tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
                tk[0][0] ^= RC[RCpointer++];

                if(KC<=6)
                        for(j=1;j<KC;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                else{
                        for(j=1;j<4;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                        for(i=0;i<4;i++) tk[i][4] ^= S[tk[i][3]];
                        for(j=5;j<KC;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                }
                for(j=0; (j<KC) && (t<(ROUNDS+1)*BC); j++, t++)
```

```
                        for(i=0;i<4;i++) W[t/BC][i][t%BC] = tk[i][j];
        }
        return 0;
}

int Encrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]){
        int r;

        AddRoundKey(a,rk[0]);

        for(r=1;r<ROUNDS;r++){
                SubBytes(a,S);
                ShiftRows(a,0);
                MixColumns(a);
                AddRoundKey(a,rk[r]);
        }

        SubBytes(a,S);
        ShiftRows(a,0);
        AddRoundKey(a,rk[ROUNDS]);

        return 0;
}

int Decrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]){
        int r;

        AddRoundKey(a,rk[ROUNDS]);
        SubBytes(a,Si);
        ShiftRows(a,1);

        for(r=ROUNDS-1;r>0;r--){
                AddRoundKey(a,rk[r]);
                InvMixColumns(a);
                SubBytes(a,Si);
                ShiftRows(a,1);
        }
        AddRoundKey(a,rk[0]);

        return 0;
}

int main(int argc, char* argv[])
{
        int i,j;
        word8 a[4][MAXBC], rk[MAXROUNDS+1][4][MAXBC], sk[4][MAXKC];

        for(KC=4;KC<=8;KC++)
                for(BC=4;BC<=8;BC++){
                        ROUNDS = numrounds[KC-4][BC-4];
                        for(j=0;j<BC;j++)
                                for(i=0;i<4;i++) a[i][j] = 0;
```

```
                        int k = 0;
                        for(j=0;j<KC;j++){
                                for(i=0;i<4;i++){
                                        sk[i][j] = 0;
                                        k += 1;
                                }
                        }

                        KeyExpansion(sk,rk);

                        printf("block length %d  key length %d¥n", 32*BC, 32*KC);
                        for(j=0;j<BC;j++)
                                for(i=0;i<4;i++) printf("%02X", a[i][j]);
                        printf("¥n");

                        Encrypt(a,rk);
                        for(j=0;j<BC;j++)
                                for(i=0;i<4;i++) printf("%02X", a[i][j]);
                        printf("¥n");

                        Encrypt(a,rk);
                        for(j=0;j<BC;j++)
                                for(i=0;i<4;i++) printf("%02X", a[i][j]);
                        printf("¥n");

                        Decrypt(a,rk);
                        for(j=0;j<BC;j++)
                                for(i=0;i<4;i++) printf("%02X", a[i][j]);
                        printf("¥n");

                        Decrypt(a,rk);
                        for(j=0;j<BC;j++)
                                for(i=0;i<4;i++) printf("%02X", a[i][j]);
                        printf("¥n");
                        printf("¥n");

                }

        return 0;
}
```

2.　AesEC のソースコード

AesEC.cpp

```
////////////////////////////////////////////////////////////////////////
// AesEC.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 <stdio.h>
#include <time.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include <malloc.h>

#define file_len(x) (unsigned long)x
#define MAXBC 8
#define MAXKC 8
#define MAXROUNDS 14

typedef unsigned char word8;
typedef unsigned int word32;

int BC, KC, ROUNDS;
int s;

word8 Logtable[256] = {
          0,   0,  25,   1,  50,   2,  26,198,  75,199,  27,104,  51,238,223,   3,
        100,   4,224,  14,  52,141,129,239,  76,113,   8,200,248,105,  28,193,
        125,194,  29,181,249,185,  39,106,  77,228,166,114,154,201,   9,120,
        101,  47,138,   5,  33,  15,225,  36,  18,240,130,  69,  53,147,218,142,
        150,143,219,189,  54,208,206,148,  19,  92,210,241,  64,  70,131,  56,
        102,221,253,  48,191,   6,139,  98,179,  37,226,152,  34,136,145,  16,
        126,110,  72,195,163,182,  30,  66,  58,107,  40,  84,250,133,  61,186,
         43,121,  10,  21,155,159,  94,202,  78,212,172,229,243,115,167,  87,
        175,  88,168,  80,244,234,214,116,  79,174,233,213,231,230,173,232,
         44,215,117,122,235,  22,  11,245,  89,203,  95,176,156,169,  81,160,
        127,  12,246,111,  23,196,  73,236,216,  67,  31,  45,164,118,123,183,
        204,187,  62,  90,251,  96,177,134,  59,  82,161,108,170,  85,  41,157,
        151,178,135,144,  97,190,220,252,188,149,207,205,  55,  63,  91,209,
         83,  57,132,  60,  65,162,109,  71,  20,  42,158,  93,  86,242,211,171,
         68,  17,146,217,  35,  32,  46,137,180,124,184,  38,119,153,227,165,
        103,  74,237,222,197,  49,254,  24,  13,  99,140,128,192,247,112,   7};

word8 Alogtable[256] = {
          1,   3,   5,  15,  17,  51,  85,255,  26,  46,114,150,161,248,  19,  53,
         95,225,  56,  72,216,115,149,164,247,   2,   6,  10,  30,  34,102,170,
        229,  52,  92,228,  55,  89,235,  38,106,190,217,112,144,171,230,  49,
         83,245,   4,  12,  20,  60,  68,204,  79,209,104,184,211,110,178,205,
         76,212,103,169,224,  59,  77,215,  98,166,241,   8,  24,  40,120,136,
        131,158,185,208,107,189,220,127,129,152,179,206,  73,219,118,154,
        181,196,  87,249,  16,  48,  80,240,  11,  29,  39,105,187,214,  97,163,
        254,  25,  43,125,135,146,173,236,  47,113,147,174,233,  32,  96,160,
        251,  22,  58,  78,210,109,183,194,  93,231,  50,  86,250,  21,  63,  65,
        195,  94,226,  61,  71,201,  64,192,  91,237,  44,116,156,191,218,117,
        159,186,213,100,172,239,  42,126,130,157,188,223,122,142,137,128,
        155,182,193,  88,232,  35,101,175,234,  37,111,177,200,  67,197,  84,
```

```
          252,  31,  33,  99, 165, 244,   7,   9,  27,  45, 119, 153, 176, 203,  70, 202,
           69, 207,  74, 222, 121, 139, 134, 145, 168, 227,  62,  66, 198,  81, 243,  14,
           18,  54,  90, 238,  41, 123, 141, 140, 143, 138, 133, 148, 167, 242,  13,  23,
           57,  75, 221, 124, 132, 151, 162, 253,  28,  36, 108, 180, 199,  82, 246,   1};

word8 S[256] = {
           99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,  43, 254, 215, 171, 118,
          202, 130, 201, 125, 250,  89,  71, 240, 173, 212, 162, 175, 156, 164, 114, 192,
          183, 253, 147,  38,  54,  63, 247, 204,  52, 165, 229, 241, 113, 216,  49,  21,
            4, 199,  35, 195,  24, 150,   5, 154,   7,  18, 128, 226, 235,  39, 178, 117,
            9, 131,  44,  26,  27, 110,  90, 160,  82,  59, 214, 179,  41, 227,  47, 132,
           83, 209,   0, 237,  32, 252, 177,  91, 106, 203, 190,  57,  74,  76,  88, 207,
          208, 239, 170, 251,  67,  77,  51, 133,  69, 249,   2, 127,  80,  60, 159, 168,
           81, 163,  64, 143, 146, 157,  56, 245, 188, 182, 218,  33,  16, 255, 243, 210,
          205,  12,  19, 236,  95, 151,  68,  23, 196, 167, 126,  61, 100,  93,  25, 115,
           96, 129,  79, 220,  34,  42, 144, 136,  70, 238, 184,  20, 222,  94,  11, 219,
          224,  50,  58,  10,  73,   6,  36,  92, 194, 211, 172,  98, 145, 149, 228, 121,
          231, 200,  55, 109, 141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,   8,
          186, 120,  37,  46,  28, 166, 180, 198, 232, 221, 116,  31,  75, 189, 139, 138,
          112,  62, 181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,  29, 158,
          225, 248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,  85,  40, 223,
          140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,  15, 176,  84, 187,  22};

word8 Si[256] = {
           82,   9, 106, 213,  48,  54, 165,  56, 191,  64, 163, 158, 129, 243, 215, 251,
          124, 227,  57, 130, 155,  47, 255, 135,  52, 142,  67,  68, 196, 222, 233, 203,
           84, 123, 148,  50, 166, 194,  35,  61, 238,  76, 149,  11,  66, 250, 195,  78,
            8,  46, 161, 102,  40, 217,  36, 178, 118,  91, 162,  73, 109, 139, 209,  37,
          114, 248, 246, 100, 134, 104, 152,  22, 212, 164,  92, 204,  93, 101, 182, 146,
          108, 112,  72,  80, 253, 237, 185, 218,  94,  21,  70,  87, 167, 141, 157, 132,
          144, 216, 171,   0, 140, 188, 211,  10, 247, 228,  88,   5, 184, 179,  69,   6,
          208,  44,  30, 143, 202,  63,  15,   2, 193, 175, 189,   3,   1,  19, 138, 107,
           58, 145,  17,  65,  79, 103, 220, 234, 151, 242, 207, 206, 240, 180, 230, 115,
          150, 172, 116,  34, 231, 173,  53, 133, 226, 249,  55, 232,  28, 117, 223, 110,
           71, 241,  26, 113,  29,  41, 197, 137, 111, 183,  98,  14, 170,  24, 190,  27,
          252,  86,  62,  75, 198, 210, 121,  32, 154, 219, 192, 254, 120, 205,  90, 244,
           31, 221, 168,  51, 136,   7, 199,  49, 177,  18,  16,  89,  39, 128, 236,  95,
           96,  81, 127, 169,  25, 181,  74,  13,  45, 229, 122, 159, 147, 201, 156, 239,
          160, 224,  59,  77, 174,  42, 245, 176, 200, 235, 187,  60, 131,  83, 153,  97,
           23,  43,   4, 126, 186, 119, 214,  38, 225, 105,  20,  99,  85,  33,  12, 125};


word32 RC[30] = {
          0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
          0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
          0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
          0xfa, 0xef, 0xc5};



static word8 shifts[5][4] = {
          0, 1, 2, 3,
```

```
        0,1,2,3,
        0,1,2,3,
        0,1,2,4,
        0,1,3,4};

static int numrounds[5][5] = {
        10,11,12,13,14,
        11,11,12,13,14,
        12,12,12,13,14,
        13,13,13,13,14,
        14,14,14,14,14};



word8 mul(word8 a, word8 b) {
        if(a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
        else return 0;
}

void AddRoundKey(word8 a[4][MAXBC], word8 rk[4][MAXBC]) {
        int i, j;

        for(i = 0; i<4; i++)
                for(j=0;j<BC;j++) a[i][j] ^= rk[i][j];
}

void SubBytes(word8 a[4][MAXBC], word8 box[256]){
        int i,j;

        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = box[a[i][j]] ;
}

void ShiftRows(word8 a[4][MAXBC], word8 d){
        word8 tmp[MAXBC];
        int i,j;

        if(d==0){
                for(i=1;i<4;i++){
                        for(j=0;j<BC;j++)
                                tmp[j] = a[i][(j+shifts[BC-4][i]) % BC];
                        for(j=0;j<BC;j++) a[i][j] = tmp[j];
                }
        }
        else{
                for(i=1;i<4;i++){
                        for(j=0;j<BC;j++)
                                tmp[j] = a[i][(BC+j-shifts[BC-4][i]) % BC];
                        for(j=0;j<BC;j++) a[i][j] = tmp[j];
                }
        }
}
```

```
void MixColumns(word8 a[4][MAXBC]){
        word8 b[4][MAXBC];
        int i,j;

        for(j=0;j<BC;j++)
                for(i=0;i<4;i++)
                        b[i][j] = mul(2,a[i][j])
                                ^ mul(3,a[(i+1) % 4][j])
                                ^ a[(i+2) % 4][j]
                                ^ a[(i+3) % 4][j];
        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = b[i][j];
}

void InvMixColumns(word8 a[4][MAXBC]){
        word8 b[4][MAXBC];
        int i,j;

        for(j=0;j<BC;j++)
        for(i=0;i<4;i++)
                b[i][j] = mul(0xe, a[i][j])
                        ^ mul(0xb, a[(i+1) % 4][j])
                        ^ mul(0xd, a[(i+2) % 4][j])
                        ^ mul(0x9, a[(i+3) % 4][j]);
        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = b[i][j];
}

int KeyExpansion(word8 k[4][MAXKC],
                                word8 W[MAXROUNDS+1][4][MAXBC]){
        int i,j,t,RCpointer = 1;
        word8 tk[4][MAXKC];

        for(j=0;j<KC;j++)
                for(i=0;i<4;i++)
                        tk[i][j] = k[i][j];
        t = 0;

        for(j=0;(j<KC) && (t<(ROUNDS+1)*BC); j++, t++)
                for(i=0;i<4;i++) W[t / BC][i][t % BC] = tk[i][j];

        while(t<(ROUNDS+1)*BC){
                for(i=0;i<4;i++)
                        tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
                tk[0][0] ^= RC[RCpointer++];

                if(KC<=6)
                        for(j=1;j<KC;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                else{
                        for(j=1;j<4;j++)
```

```
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                        for(i=0;i<4;i++) tk[i][4] ^= S[tk[i][3]];
                        for(j=5;j<KC;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                }
                for(j=0; (j<KC) && (t<(ROUNDS+1)*BC); j++, t++)
                        for(i=0;i<4;i++) W[t/BC][i][t%BC] = tk[i][j];
        }
        return 0;
}

int Encrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]){
        int r;

        AddRoundKey(a,rk[0]);

        for(r=1;r<ROUNDS;r++){
                SubBytes(a,S);
                ShiftRows(a,0);
                MixColumns(a);
                AddRoundKey(a,rk[r]);
        }

        SubBytes(a,S);
        ShiftRows(a,0);
        AddRoundKey(a,rk[ROUNDS]);

        return 0;
}
/*
int Decrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]){
        int r;

        AddRoundKey(a,rk[ROUNDS]);
        SubBytes(a,Si);
        ShiftRows(a,1);

        for(r=ROUNDS-1;r>0;r--){
                AddRoundKey(a,rk[r]);
                InvMixColumns(a);
                SubBytes(a,Si);
                ShiftRows(a,1);
        }
        AddRoundKey(a,rk[0]);

        return 0;
}
*/

int main(int argc, char* argv[])
{
        int i,j,klen,blen;
```

```
    word8 a[4][MAXBC], rk[MAXROUNDS+1][4][MAXBC], sk[4][MAXKC];
    char c_klen[8], c_blen[8], pass[128];
    FILE *fkey =0, *fin = 0, *fout = 0;
fpos_t flen;
    char *dbuf;
unsigned long len, rlen, blen4;

    printf( "Start!¥n" );

    if( argc != 4 ){                          // 使い方の誤り
            printf( "argc != 4 ¥n" );
            return -1;
    }

    if((fkey = fopen(argv[1], "rt")) == NULL){
            printf("Can not find key file for encryption. ¥n");
            return (-1);
    }

    fgets( c_klen, 6, fkey );
    fgets( c_blen, 6, fkey );
    fgets( pass, 127, fkey );

    klen = atoi(c_klen)/32;
    blen = atoi(c_blen)/32;
    blen4 = (unsigned int)blen*4;

    KC = klen;
    if(KC<4 || 8<KC){
            printf("Wrong key size. ¥n");
            return (-1);
    }

    BC = blen;
    if(BC<4 || 8<BC){
            printf("Wrong block size. ¥n");
            return (-1);
    }

    dbuf = (char *)malloc(2*MAXBC*4);
    if(dbuf == NULL){
            printf("No memory. ¥n");
            return (-1);
    }

    if((fin = fopen(argv[2], "rb")) == NULL){
            printf("Can not open plane text file. ¥n");
            return (-1);
    }
fseek(fin, 0, SEEK_END);
fgetpos(fin, &flen);
rlen = file_len(flen);
```

```
    // reset to start
    fseek(fin, 0, SEEK_SET);

        if((fout = fopen(argv[3], "wb")) == NULL){
                printf("Can not open encrypted data file. ¥n");
                return (-1);
        }

        ROUNDS = numrounds[KC-4][BC-4];

        char cl,cr;
        int k = 0;
        for(j=0;j<KC;j++){
                for(i=0;i<4;i++){
                        if(pass[k]>=0x30 && pass[k]<=0x39){cl = pass[k]-0x30;}
                        else if(pass[k]>=0x41 && pass[k]<=0x46){cl = pass[k]-0x37;}
                        else if(pass[k]>=0x61 && pass[k]<=0x66){cl = pass[k]-0x57;}
                        else cl = 0;
                        if(pass[k+1]>=0x30 && pass[k+1]<=0x39){cr = pass[k+1]-0x30;}
                        else if(pass[k+1]>=0x41 && pass[k+1]<=0x46){cr = pass[k+1]-0x37;}
                        else if(pass[k+1]>=0x61 && pass[k+1]<=0x66){cr = pass[k+1]-0x57;}
                        else cr = 0;
                        sk[i][j] = ((cl<<4)|(cr));
                        k += 2;
                }
        }

        KeyExpansion(sk,rk);

////////////////////////////////////////////////////////
        s = sizeof(unsigned int);
        // write the bytes of the file
        *((unsigned int*)dbuf) = rlen;
        len = (unsigned long) fread(dbuf+s, 1, blen4-s , fin);
    rlen -= len;
    // pad the file bytes with zeroes
    for(i = len+s; (unsigned int)i < blen4 ; ++i)
        dbuf[i] = 0;
    // encrypt the top 16 bytes of the buffer
        k=0;
        for(j=0;j<BC;j++){
                for(i=0;i<4;i++){
                        a[i][j] = dbuf[k];
                        k++;
                }
        }
        Encrypt(a,rk);
        k=0;
        for(j=0;j<BC;j++){
                for(i=0;i<4;i++){
                        dbuf[k] = a[i][j];
                        k++;
```

```
            }
        }
    if(fwrite(dbuf, 1, blen4, fout) != (unsigned int)blen4)
        return -2;


    if((rlen <= (unsigned int)blen4) && (rlen > 0))
    {   // if the file length is less than or equal to 16 bytes
        // read the bytes of the file into the buffer and verify length
        len = (unsigned long) fread(dbuf, 1, blen4 , fin);
        rlen -= len;

        if(rlen > 0)
            return -1;

        // pad the file bytes with zeroes
        for(i = len; (unsigned int)i < blen4 ; ++i)
            dbuf[i] = 0;

        // encrypt the top 16 bytes of the buffer
                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                a[i][j] = dbuf[k];
                                k++;
                        }
                }

                Encrypt(a,rk);

        // write the IV and the encrypted file bytes
                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                dbuf[k] = a[i][j];
                                k++;
                        }
                }
        if(fwrite(dbuf, 1, blen4, fout) != (unsigned int)blen4)
            return -2;
    }
    else
    {   // if the file length is more 16 bytes
        // read the file a block at a time
        while(rlen > 0 && !feof(fin))
        {
            // read a block and reduce the remaining byte count
            len = (unsigned long)fread(dbuf, 1, blen4 , fin);
            rlen -= len;

            // verify length of block
            if(len != (unsigned int)blen4 ){
```

```
// pad the file bytes with zeroes
                        for(i = len; (unsigned int)i < blen4 ; ++i)
                                dbuf[i] = 0;
                }

// encrypt the block
                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                a[i][j] = dbuf[k];
                                k++;
                        }
                }
                Encrypt(a,rk);
// write the encrypted block
                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                dbuf[k] = a[i][j];
                                k++;
                        }
                }
if(fwrite(dbuf, 1, blen4, fout) != blen4)
   return -2;

// if there is only one more block do ciphertext stealing
if(rlen > 0 && rlen < (unsigned int)blen4 )
{
                len = (unsigned long)fread(dbuf, 1, blen4 , fin);

        // clear the remainder of the bottom half of buffer
for(i = 0; (unsigned int)i < (unsigned int)blen4  - len; ++i)
    dbuf[len +i] = 0;

    // encrypt the final block
                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                a[i][j] = dbuf[k];
                                k++;
                        }
                }
                Encrypt(a,rk);

                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                dbuf[k] = a[i][j];
                                k++;
                        }
                }
if(fwrite(dbuf, 1, blen4, fout) != blen4)
```

```
                return -2;

                        break;
                 // set the length of the final write
             }

         }
     }

         if(fkey)
         {
         fclose(fkey);
         }

         if(fout)
         {
         fclose(fout);
         }

         if(fin)
         {
                 fclose(fin);
         }

         free(dbuf);

  ////////////////////////////////////////////
         printf( "End!¥n" );

         return 0;
}
```

Stdafx.cpp

```
////////////////////////////////////////////////////
// stdafx.cpp : 標準インクルードAesEC.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。
```

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。
```

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. AesDC のソースコード

AesDC.cpp

```
////////////////////////////////////////////////////////////////////////////
// AesDC.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 <stdio.h>
#include <time.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include <malloc.h>

#define file_len(x) (unsigned long)x
#define MAXBC 8
#define MAXKC 8
#define MAXROUNDS 14

typedef unsigned char word8;
typedef unsigned int word32;

int BC, KC, ROUNDS;
int s;

word8 Logtable[256] = {
```

```
          0,   0,  25,   1,  50,   2,  26, 198,  75, 199,  27, 104,  51, 238, 223,   3,
        100,   4, 224,  14,  52, 141, 129, 239,  76, 113,   8, 200, 248, 105,  28, 193,
        125, 194,  29, 181, 249, 185,  39, 106,  77, 228, 166, 114, 154, 201,   9, 120,
        101,  47, 138,   5,  33,  15, 225,  36,  18, 240, 130,  69,  53, 147, 218, 142,
        150, 143, 219, 189,  54, 208, 206, 148,  19,  92, 210, 241,  64,  70, 131,  56,
        102, 221, 253,  48, 191,   6, 139,  98, 179,  37, 226, 152,  34, 136, 145,  16,
        126, 110,  72, 195, 163, 182,  30,  66,  58, 107,  40,  84, 250, 133,  61, 186,
         43, 121,  10,  21, 155, 159,  94, 202,  78, 212, 172, 229, 243, 115, 167,  87,
        175,  88, 168,  80, 244, 234, 214, 116,  79, 174, 233, 213, 231, 230, 173, 232,
         44, 215, 117, 122, 235,  22,  11, 245,  89, 203,  95, 176, 156, 169,  81, 160,
        127,  12, 246, 111,  23, 196,  73, 236, 216,  67,  31,  45, 164, 118, 123, 183,
        204, 187,  62,  90, 251,  96, 177, 134,  59,  82, 161, 108, 170,  85,  41, 157,
        151, 178, 135, 144,  97, 190, 220, 252, 188, 149, 207, 205,  55,  63,  91, 209,
         83,  57, 132,  60,  65, 162, 109,  71,  20,  42, 158,  93,  86, 242, 211, 171,
         68,  17, 146, 217,  35,  32,  46, 137, 180, 124, 184,  38, 119, 153, 227, 165,
        103,  74, 237, 222, 197,  49, 254,  24,  13,  99, 140, 128, 192, 247, 112,   7};

word8 Alogtable[256] = {
          1,   3,   5,  15,  17,  51,  85, 255,  26,  46, 114, 150, 161, 248,  19,  53,
         95, 225,  56,  72, 216, 115, 149, 164, 247,   2,   6,  10,  30,  34, 102, 170,
        229,  52,  92, 228,  55,  89, 235,  38, 106, 190, 217, 112, 144, 171, 230,  49,
         83, 245,   4,  12,  20,  60,  68, 204,  79, 209, 104, 184, 211, 110, 178, 205,
         76, 212, 103, 169, 224,  59,  77, 215,  98, 166, 241,   8,  24,  40, 120, 136,
        131, 158, 185, 208, 107, 189, 220, 127, 129, 152, 179, 206,  73, 219, 118, 154,
        181, 196,  87, 249,  16,  48,  80, 240,  11,  29,  39, 105, 187, 214,  97, 163,
        254,  25,  43, 125, 135, 146, 173, 236,  47, 113, 147, 174, 233,  32,  96, 160,
        251,  22,  58,  78, 210, 109, 183, 194,  93, 231,  50,  86, 250,  21,  63,  65,
        195,  94, 226,  61,  71, 201,  64, 192,  91, 237,  44, 116, 156, 191, 218, 117,
        159, 186, 213, 100, 172, 239,  42, 126, 130, 157, 188, 223, 122, 142, 137, 128,
        155, 182, 193,  88, 232,  35, 101, 175, 234,  37, 111, 177, 200,  67, 197,  84,
        252,  31,  33,  99, 165, 244,   7,   9,  27,  45, 119, 153, 176, 203,  70, 202,
         69, 207,  74, 222, 121, 139, 134, 145, 168, 227,  62,  66, 198,  81, 243,  14,
         18,  54,  90, 238,  41, 123, 141, 140, 143, 138, 133, 148, 167, 242,  13,  23,
         57,  75, 221, 124, 132, 151, 162, 253,  28,  36, 108, 180, 199,  82, 246,   1};

word8 S[256] = {
         99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,  43, 254, 215, 171, 118,
        202, 130, 201, 125, 250,  89,  71, 240, 173, 212, 162, 175, 156, 164, 114, 192,
        183, 253, 147,  38,  54,  63, 247, 204,  52, 165, 229, 241, 113, 216,  49,  21,
          4, 199,  35, 195,  24, 150,   5, 154,   7,  18, 128, 226, 235,  39, 178, 117,
          9, 131,  44,  26,  27, 110,  90, 160,  82,  59, 214, 179,  41, 227,  47, 132,
         83, 209,   0, 237,  32, 252, 177,  91, 106, 203, 190,  57,  74,  76,  88, 207,
        208, 239, 170, 251,  67,  77,  51, 133,  69, 249,   2, 127,  80,  60, 159, 168,
         81, 163,  64, 143, 146, 157,  56, 245, 188, 182, 218,  33,  16, 255, 243, 210,
        205,  12,  19, 236,  95, 151,  68,  23, 196, 167, 126,  61, 100,  93,  25, 115,
         96, 129,  79, 220,  34,  42, 144, 136,  70, 238, 184,  20, 222,  94,  11, 219,
        224,  50,  58,  10,  73,   6,  36,  92, 194, 211, 172,  98, 145, 149, 228, 121,
        231, 200,  55, 109, 141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,   8,
        186, 120,  37,  46,  28, 166, 180, 198, 232, 221, 116,  31,  75, 189, 139, 138,
        112,  62, 181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,  29, 158,
        225, 248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,  85,  40, 223,
        140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,  15, 176,  84, 187,  22};
```

```
word8 Si[256] = {
          82,   9,106,213,  48,  54,165,  56,191,  64,163,158,129,243,215,251,
         124,227,  57,130,155,  47,255,135,  52,142,  67,  68,196,222,233,203,
          84,123,148,  50,166,194,  35,  61,238,  76,149,  11,  66,250,195,  78,
           8,  46,161,102,  40,217,  36,178,118,  91,162,  73,109,139,209,  37,
         114,248,246,100,134,104,152,  22,212,164,  92,204,  93,101,182,146,
         108,112,  72,  80,253,237,185,218,  94,  21,  70,  87,167,141,157,132,
         144,216,171,   0,140,188,211,  10,247,228,  88,   5,184,179,  69,   6,
         208,  44,  30,143,202,  63,  15,   2,193,175,189,   3,   1,  19,138,107,
          58,145,  17,  65,  79,103,220,234,151,242,207,206,240,180,230,115,
         150,172,116,  34,231,173,  53,133,226,249,  55,232,  28,117,223,110,
          71,241,  26,113,  29,  41,197,137,111,183,  98,  14,170,  24,190,  27,
         252,  86,  62,  75,198,210,121,  32,154,219,192,254,120,205,  90,244,
          31,221,168,  51,136,   7,199,  49,177,  18,  16,  89,  39,128,236,  95,
          96,  81,127,169,  25,181,  74,  13,  45,229,122,159,147,201,156,239,
         160,224,  59,  77,174,  42,245,176,200,235,187,  60,131,  83,153,  97,
          23,  43,   4,126,186,119,214,  38,225,105,  20,  99,  85,  33,  12,125};


word32 RC[30] = {
         0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
         0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,
         0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,
         0xfa,0xef,0xc5};



static word8 shifts[5][4] = {
         0,1,2,3,
         0,1,2,3,
         0,1,2,3,
         0,1,2,4,
         0,1,3,4};

static int numrounds[5][5] = {
         10,11,12,13,14,
         11,11,12,13,14,
         12,12,12,13,14,
         13,13,13,13,14,
         14,14,14,14,14};



word8 mul(word8 a, word8 b) {
         if(a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
         else return 0;
}

void AddRoundKey(word8 a[4][MAXBC], word8 rk[4][MAXBC]) {
         int i, j;
```

```
        for(i = 0; i<4; i++)
                for(j=0;j<BC;j++) a[i][j] ^= rk[i][j];
}

void SubBytes(word8 a[4][MAXBC], word8 box[256]){
        int i,j;

        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = box[a[i][j]] ;
}

void ShiftRows(word8 a[4][MAXBC], word8 d){
        word8 tmp[MAXBC];
        int i,j;

        if(d==0){
                for(i=1;i<4;i++){
                        for(j=0;j<BC;j++)
                                tmp[j] = a[i][(j+shifts[BC-4][i]) % BC];
                        for(j=0;j<BC;j++) a[i][j] = tmp[j];
                }
        }
        else{
                for(i=1;i<4;i++){
                        for(j=0;j<BC;j++)
                                tmp[j] = a[i][(BC+j-shifts[BC-4][i]) % BC];
                        for(j=0;j<BC;j++) a[i][j] = tmp[j];
                }
        }
}

void MixColumns(word8 a[4][MAXBC]){
        word8 b[4][MAXBC];
        int i,j;

        for(j=0;j<BC;j++)
                for(i=0;i<4;i++)
                        b[i][j] = mul(2,a[i][j])
                                ^ mul(3,a[(i+1) % 4][j])
                                ^ a[(i+2) % 4][j]
                                ^ a[(i+3) % 4][j];
        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = b[i][j];
}

void InvMixColumns(word8 a[4][MAXBC]){
        word8 b[4][MAXBC];
        int i,j;

        for(j=0;j<BC;j++)
        for(i=0;i<4;i++)
                b[i][j] = mul(0xe, a[i][j])
```

```
                        ^ mul(0xb, a[(i+1) % 4][j])
                        ^ mul(0xd, a[(i+2) % 4][j])
                        ^ mul(0x9, a[(i+3) % 4][j]);
        for(i=0;i<4;i++)
                for(j=0;j<BC;j++) a[i][j] = b[i][j];
}

int KeyExpansion(word8 k[4][MAXKC],
                                word8 W[MAXROUNDS+1][4][MAXBC]){
        int i,j,t,RCpointer = 1;
        word8 tk[4][MAXKC];

        for(j=0;j<KC;j++)
                for(i=0;i<4;i++)
                        tk[i][j] = k[i][j];
        t = 0;

        for(j=0;(j<KC) && (t<(ROUNDS+1)*BC); j++, t++)
                for(i=0;i<4;i++) W[t / BC][i][t % BC] = tk[i][j];

        while(t<(ROUNDS+1)*BC){
                for(i=0;i<4;i++)
                        tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
                tk[0][0] ^= RC[RCpointer++];

                if(KC<=6)
                        for(j=1;j<KC;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                else{
                        for(j=1;j<4;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                        for(i=0;i<4;i++) tk[i][4] ^= S[tk[i][3]];
                        for(j=5;j<KC;j++)
                                for(i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
                }
                for(j=0; (j<KC) && (t<(ROUNDS+1)*BC); j++, t++)
                        for(i=0;i<4;i++) W[t/BC][i][t%BC] = tk[i][j];
        }
        return 0;
}
/*
int Encrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]){
        int r;

        AddRoundKey(a,rk[0]);

        for(r=1;r<ROUNDS;r++){
                SubBytes(a,S);
                ShiftRows(a,0);
                MixColumns(a);
                AddRoundKey(a,rk[r]);
        }
```

```
        SubBytes(a,S);
        ShiftRows(a,0);
        AddRoundKey(a,rk[ROUNDS]);

        return 0;
}
*/
int Decrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]){
        int r;

        AddRoundKey(a,rk[ROUNDS]);
        SubBytes(a,Si);
        ShiftRows(a,1);

        for(r=ROUNDS-1;r>0;r--){
                AddRoundKey(a,rk[r]);
                InvMixColumns(a);
                SubBytes(a,Si);
                ShiftRows(a,1);
        }
        AddRoundKey(a,rk[0]);

        return 0;
}


int main(int argc, char* argv[])
{
        int i,j,klen,blen;
        word8 a[4][MAXBC], rk[MAXROUNDS+1][4][MAXBC], sk[4][MAXKC];
        char c_klen[8], c_blen[8], pass[128];
        FILE *fkey =0, *fin = 0, *fout = 0;
    fpos_t flen;
        char *dbuf;
    unsigned long len, rlen, blen4;

////////////////////////////////////////////////////////////
        printf("Start!¥n");

        if( argc != 4 ){                        // 使い方の誤り
                printf( "argc != 4 ¥n" );
                return -1;
        }

        if((fkey = fopen(argv[1], "rt")) == NULL){
                printf("Can not find key file for decryption. ¥n");
                return (-1);
        }
        fgets( c_klen, 6, fkey );
        fgets( c_blen, 6, fkey );
        fgets( pass, 127, fkey );
```

```
	klen = atoi(c_klen)/32;
	blen = atoi(c_blen)/32;
	blen4 = (unsigned int)blen*4;


	dbuf = (char *)malloc(2*blen4);


	KC = klen;
	if(KC<4 || 8<KC){
		printf("Wrong key size. ¥n");
		return (-1);
	}

	BC = blen;
	if(BC<4 || 8<BC){
		printf("Wrong block size. ¥n");
		return (-1);
	}


	if((fin = fopen(argv[2], "rb")) == NULL){
		printf("Can not open encrypted file. ¥n");
		return (-1);
	}
fseek(fin, 0, SEEK_END);
fgetpos(fin, &flen);
rlen = file_len(flen);
// reset to start
fseek(fin, 0, SEEK_SET);

	if((fout = fopen(argv[3], "wb")) == NULL){
		printf("Can not open decrypted file. ¥n");
		return (-1);
	}

	ROUNDS = numrounds[KC-4][BC-4];

	char cl,cr;
	int k = 0;
	for(j=0;j<KC;j++){
		for(i=0;i<4;i++){
			if(pass[k]>=0x30 && pass[k]<=0x39){cl = pass[k]-0x30;}
			else if(pass[k]>=0x41 && pass[k]<=0x46){cl = pass[k]-0x37;}
			else if(pass[k]>=0x61 && pass[k]<=0x66){cl = pass[k]-0x57;}
			else cl = 0;
			if(pass[k+1]>=0x30 && pass[k+1]<=0x39){cr = pass[k+1]-0x30;}
			else if(pass[k+1]>=0x41 && pass[k+1]<=0x46){cr = pass[k+1]-0x37;}
			else if(pass[k+1]>=0x61 && pass[k+1]<=0x66){cr = pass[k+1]-0x57;}
			else cr = 0;
			sk[i][j] = ((cl<<4)|(cr));
```

```
                        k += 2;
                }
        }

        KeyExpansion(sk,rk);
        s = sizeof(unsigned int);
////////////////////////////////

        len = (unsigned long) fread(dbuf, 1, blen4 , fin);
    rlen -= len;

    // decrypt the top 16 bytes of the buffer
        k=0;
        for(j=0;j<BC;j++){
                for(i=0;i<4;i++){
                        a[i][j] = dbuf[k];
                        k++;
                }
        }

        Decrypt(a,rk);
    len = blen4 ;

        // write the IV and the encrypted file bytes
        k=0;
        for(j=0;j<BC;j++){
                for(i=0;i<4;i++){
                        dbuf[k] = a[i][j];
                        k++;
                }
        }

        unsigned int wlen = *((unsigned int*)dbuf);

        if(wlen <= len-s){
        if(fwrite(dbuf+s, 1, wlen, fout) != wlen)
                return -2;
        }
        else{
                wlen -= len-s;
        if(fwrite(dbuf+s, 1, len-s, fout) != len-s)
                return -2;
        }

    if((rlen <= blen4) && (rlen>0) )
    {   // if the original file length is less than or equal to 16 bytes
        // read the bytes of the file and verify length
        len = (unsigned long)fread(dbuf, 1, blen4 , fin);
        rlen -= len;

                if(rlen > 0)
            return -1;
```

```
        // decrypt from position len to position len + BLOCK_LEN
                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                a[i][j] = dbuf[k];
                                k++;
                        }
                }

                Decrypt(a,rk);

                k=0;
                for(j=0;j<BC;j++){
                        for(i=0;i<4;i++){
                                dbuf[k] = a[i][j];
                                k++;
                        }
                }

        // output decrypted bytes
        if(fwrite(dbuf, 1, wlen, fout) != (unsigned long)wlen)
            return -2;
    }
    else
    {
        // read the encrypted file a block at a time
        while(rlen > 0 && !feof(fin))
        {
            // input a block and reduce the remaining byte count
            len = (unsigned long)fread(dbuf, 1, blen4, fin);
            rlen -= len;

            // verify the length of the read operation
            if(len != blen4 )
                return -1;

            // decrypt input buffer
                        k=0;
                        for(j=0;j<BC;j++){
                                for(i=0;i<4;i++){
                                        a[i][j] = dbuf[k];
                                        k++;
                                }
                        }

                        Decrypt(a,rk);

                        k=0;
                        for(j=0;j<BC;j++){
                                for(i=0;i<4;i++){
                                        dbuf[k] = a[i][j];
```

```
                                        k++;
                                }
                        }

                        if(wlen < len){
                                if(fwrite(dbuf, 1, wlen, fout) != wlen){
                                    return -2;
                                }
                                break;
                        }
                        else{
                                if(fwrite(dbuf, 1, blen4, fout) != blen4)
                                        return -2;
                        }
                        if(wlen > len){wlen -= len;}
                }
    }

        if(fkey)
        {
        fclose(fkey);
        }

        if(fout)
        {
        fclose(fout);
        }

        if(fin)
        {
                fclose(fin);
        }

        free(dbuf);
////////////////////////////////////////////
        printf("End!¥n");
        return 0;
}
```

Stdafx.cpp

```
////////////////////////////////////////////////////
// stdafx.cpp : 標準インクルードAesDC.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。
```

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。
```

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。　　2012.04.10　　宇山靖政