```
#include "stdafx.h" //  holds include files

//  the following defines are for a 16 byte (128 bit) passphrase
#define Nk  4       //  Number of 32-bit words comprising the Cipher Key
#define Nb  4       //  Number of columns (32-bit words) comprising the State block length
#define Nr  10  //  Number of rounds, which is a function of Nk and Nb

//  this is not size efficient as does not use local memory
//  but advantage is that it enables zeroing of memory
struct AESDetails
{
    int i;
    int pos;
    unsigned char result1;
    unsigned result2;
    unsigned int NoTimes;
    int count;
    unsigned char W[4*8*15];   // the expanded key
    unsigned char gf2_8_inv[256];
    // tables for inverses, byte sub
    unsigned char byte_sub[256];
    unsigned char inv_byte_sub[256];
    unsigned long Rcon[60];
    // long tables for encryption stuff
    unsigned long T0[256];
    unsigned long T1[256];
    unsigned long T2[256];
    unsigned long T3[256];
    // long tables for decryption stuff
    unsigned long I0[256];
    unsigned long I1[256];
    unsigned long I2[256];
    unsigned long I3[256];

    // huge tables - todo - ifdef out
    unsigned long T4[256];
    unsigned long T5[256];
    unsigned long T6[256];
    unsigned long T7[256];

    unsigned long I4[256];
    unsigned long I5[256];
    unsigned long I6[256];
    unsigned long I7[256];

    unsigned char *Buffer;
    unsigned long EntryBuffer[4];
    unsigned long ExitBuffer[4];

    unsigned long state[8*2]; // 2 buffers
    unsigned long * dest;  //   = AESitems->state;
    unsigned long * src;   //   = AESitems->state;

    unsigned char * W_ptr;
    unsigned long * WL;
    int col;
    BOOL retval;
    unsigned char a1,a2,a3,b1,b2,b3,b4,b5;
    unsigned char a0,b0;    //  a1,a2,a3 b1,b2,b3

    unsigned long Tmp;
    unsigned char Ri;
};
struct AESDetails *AESitems;
HANDLE hAES;

    //int Nb;   // block length
    //int Nk; // key length 4
    //int Nr; // number of rounds

extern char *errstring;     //  "Not Enough Memory for Program"

// define to mult a byte by x mod the proper poly
```

```
#define xmult(a) ((a)<<1) ^ (((a)&128) ? 0x01B : 0)

// make 4 bytes (LSB first) into a 4 byte vector
#define VEC4(a,b,c,d) (((long)(a)) | (((long)(b))<<8) | (((long)(c))<<16) | (((long)(d))<<2
4))

// get byte 0 to 3 from word a
#define GetByte(a,n) ((unsigned char)((a) >> (n<<3)))

//  bytes (a,b,c,d) -> (b,c,d,a)    so low becomes high
#define RotByte(a) _rotr(a,8)

#define RotByteL(a) _rotl(a,8)

// this define computes one of the round vectors
#define compute_one(dest,src2,j,C1,C2,C3,Nb)    *(dest+j) = \
    AESitems->T0[GetByte(src2[j],0)]^AESitems->T1[GetByte(src2[((j+C1+Nb)%Nb)],1)]^ \
    AESitems->T2[GetByte(src2[((j+C2+Nb)%Nb)],2)]^AESitems->T3[GetByte(src2[((j+C3+Nb)%Nb)]
,3)] \
    ^*r_ptr++

// for another 4K tables, we save 3 clock cycles - sick
#define compute_one_final(dest,src,j,C1,C2,C3,Nb)  *dest++ = \
    (AESitems->T4[GetByte(src[j],0)])^\
    (AESitems->T5[GetByte(src[((j+C1+Nb)%Nb)],1)])^ \
    (AESitems->T6[GetByte(src[((j+C2+Nb)%Nb)],2)])^ \
    (AESitems->T7[GetByte(src[((j+C3+Nb)%Nb)],3)])^*r_ptr++

// key adding for 4,6,8 column cases
#define AddRoundKey4(dest,src)  \
    *dest++ = *r_ptr++ ^ *src++;\
    *dest++ = *r_ptr++ ^ *src++;\
    *dest++ = *r_ptr++ ^ *src++;\
    *dest++ = *r_ptr++ ^ *src++;

#define Round4(d,s) \
        compute_one(d,s,0,1,2,3,4); \
        compute_one(d,s,1,1,2,3,4); \
        compute_one(d,s,2,1,2,3,4); \
        compute_one(d,s,3,1,2,3,4);

// final round defines - this one is for case for 4 columns
#define FinalRound4(d,s) compute_one_final(d,s,0,1,2,3,4); \
                         compute_one_final(d,s,1,1,2,3,4); \
                         compute_one_final(d,s,2,1,2,3,4); \
                         compute_one_final(d,s,3,1,2,3,4);

#define compute_one_inv(dest,src2,j,C1,C2,C3,Nb)    *(dest+j) = \
    AESitems->I0[GetByte(src2[j],0)]^AESitems->I1[GetByte(src2[((j-C1+Nb)%Nb)],1)]^ \
    AESitems->I2[GetByte(src2[((j-C2+Nb)%Nb)],2)]^AESitems->I3[GetByte(src2[((j-C3+Nb)%Nb)]
,3)] \
    ^*r_ptr++

// inverse cipher stuff
#define compute_one_final_inv(dest,src,j,C1,C2,C3,Nb)  *dest++ = \
    (AESitems->I4[GetByte(src[j],0)])^\
    (AESitems->I5[GetByte(src[((j-C1+Nb)%Nb)],1)])^ \
    (AESitems->I6[GetByte(src[((j-C2+Nb)%Nb)],2)])^ \
    (AESitems->I7[GetByte(src[((j-C3+Nb)%Nb)],3)])^*r_ptr++

#define InvRound4(d,s)  \
        compute_one_inv(d,s,0,1,2,3,4); \
        compute_one_inv(d,s,1,1,2,3,4); \
        compute_one_inv(d,s,2,1,2,3,4); \
        compute_one_inv(d,s,3,1,2,3,4);

#define InvFinalRound4(d,s) compute_one_final_inv(d,s,0,1,2,3,4); \
                         compute_one_final_inv(d,s,1,1,2,3,4); \
                         compute_one_final_inv(d,s,2,1,2,3,4); \
                         compute_one_final_inv(d,s,3,1,2,3,4);

int ErrBox(const char* format, ...);    //  gives error message
LPVOID FAR PASCAL HeapAlloc1(HANDLE hHeap,                  // ensures zeroed memory
```

```
                              DWORD dwFlags, DWORD dwBytes);
void * FAR PASCAL HeapDestroy1(HANDLE *hDATA,
                               void *DATA); //  function zeros created memory
                                            //  before destroying
void FAR PASCAL KeyExpansion(const unsigned char * key);
inline FAR PASCAL StartEncryption(const unsigned char * key);
void FAR PASCAL StartDecryption(unsigned char * key);
void FAR PASCAL AESDecrypt(unsigned long * datain, unsigned long * dataout);
BOOL FAR PASCAL CreateAESTables();
BOOL FAR PASCAL CheckInverses();
BOOL FAR PASCAL CheckByteSub();
BOOL FAR PASCAL CheckInvByteSub();
BOOL FAR PASCAL CheckRcon();
void FAR PASCAL CheckLargeTables();
inline  unsigned char FAR PASCAL GF2_8_mult(unsigned char a, unsigned char b);
unsigned char FAR PASCAL BitSum(unsigned char byte);
void FAR PASCAL TextToHex(const char * in, char * data);
void FAR PASCAL AESEncrypt(unsigned long * datain, unsigned long * dataout);
unsigned long FAR PASCAL SubByte(unsigned long data);
inline  void FAR PASCAL swap(unsigned long * _X, unsigned long * _Y);
BOOL FAR PASCAL AES(unsigned char *fileBuffer,
                               unsigned long fileBufferLength,
                               char *passphrase, BOOL ENCRYPT);

void FAR PASCAL AESDecrypt(unsigned long * datain, unsigned long * dataout)
{
    unsigned long * r_ptr = (unsigned long *)(AESitems->W);
    AESitems->dest  = AESitems->state;
    AESitems->src   = AESitems->state;

    AddRoundKey4(AESitems->dest,datain);

    InvRound4(AESitems->dest, AESitems->src);
    InvRound4(AESitems->src, AESitems->dest);
    InvRound4(AESitems->dest, AESitems->src);
    InvRound4(AESitems->src, AESitems->dest);
    InvRound4(AESitems->dest, AESitems->src);
    InvRound4(AESitems->src, AESitems->dest);
    InvRound4(AESitems->dest, AESitems->src);
    InvRound4(AESitems->src, AESitems->dest);
    InvRound4(AESitems->dest, AESitems->src);

    InvFinalRound4(dataout, AESitems->dest);
    r_ptr = 0;
} // Decrypt


void FAR PASCAL StartDecryption(unsigned char * key)
{
    KeyExpansion(key);

    AESitems->W_ptr = AESitems->W;

    for (AESitems->col = Nb; AESitems->col < (Nr)*Nb; AESitems->col++) // do all but first
and last round
    {
        AESitems->a0 = AESitems->W_ptr[4*AESitems->col+0];
        AESitems->a1 = AESitems->W_ptr[4*AESitems->col+1];
        AESitems->a2 = AESitems->W_ptr[4*AESitems->col+2];
        AESitems->a3 = AESitems->W_ptr[4*AESitems->col+3];

        AESitems->b0 = GF2_8_mult(0x0E,AESitems->a0)^GF2_8_mult(0x0B,AESitems->a1)^
             GF2_8_mult(0x0D,AESitems->a2)^GF2_8_mult(0x09,AESitems->a3);
        AESitems->b1 = GF2_8_mult(0x09,AESitems->a0)^GF2_8_mult(0x0E,AESitems->a1)^
             GF2_8_mult(0x0B,AESitems->a2)^GF2_8_mult(0x0D,AESitems->a3);
        AESitems->b2 = GF2_8_mult(0x0D,AESitems->a0)^GF2_8_mult(0x09,AESitems->a1)^
             GF2_8_mult(0x0E,AESitems->a2)^GF2_8_mult(0x0B,AESitems->a3);
        AESitems->b3 = GF2_8_mult(0x0B,AESitems->a0)^GF2_8_mult(0x0D,AESitems->a1)^
             GF2_8_mult(0x09,AESitems->a2)^GF2_8_mult(0x0E,AESitems->a3);

        AESitems->W_ptr[4*AESitems->col+0] = AESitems->b0;
        AESitems->W_ptr[4*AESitems->col+1] = AESitems->b1;
        AESitems->W_ptr[4*AESitems->col+2] = AESitems->b2;
```

```
        AESitems->W_ptr[4*AESitems->col+3] = AESitems->b3;
    }

    // we reverse the rounds to make decryption faster
    AESitems->WL = (unsigned long*)AESitems->W;
    for (AESitems->pos = 0; AESitems->pos < Nr/2; AESitems->pos++)
    {
        for (AESitems->col = 0; AESitems->col < Nb; AESitems->col++)
            swap(&AESitems->WL[AESitems->col+AESitems->pos*Nb],
                &AESitems->WL[AESitems->col+(Nr-AESitems->pos)*Nb]);
    }
} // StartDecryption

inline  void FAR PASCAL swap(unsigned long * X, unsigned long * Y)
{
    AESitems->Tmp = *X;
    *X = *Y, *Y = AESitems->Tmp;
}



BOOL FAR PASCAL CreateAESTables()
{
    AESitems->retval = true;
    if (CheckInverses() == false)
        AESitems->retval = false;
    if (CheckByteSub() == false)
        AESitems->retval = false;
    if (CheckInvByteSub() == false)
        AESitems->retval = false;
    if (CheckRcon() == false)
        return false;
    CheckLargeTables();
    return AESitems->retval;
} // CreateAESTables

BOOL FAR PASCAL CheckInverses()
{
    // we'll brute force the inverse table
//  assert(GF2_8_mult(0x57,0x13) == 0xFE); // test these first
//  assert(GF2_8_mult(0x01,0x01) == 0x01);
//  assert(GF2_8_mult(0xFF,0x55) == 0xF8);
    unsigned int a,b;

    (AESitems->gf2_8_inv)[0] = (unsigned char) 0;

    for (a = 1; a <= 255; a++)
    {
        b = 1;
        while (GF2_8_mult(a,b) != 1)
        {
            b++;
        }
        (AESitems->gf2_8_inv)[a] = (unsigned char) b;
    }
    a = 0;
    b = 0;
    return true;
}

BOOL FAR PASCAL CheckByteSub()
{
    if (CheckInverses() == false)
    {
        return false; // we cannot do this without inverses
    }

    unsigned int x,y; // need ints here to prevent wrap in loop
    for (x = 0; x <= 255; x++)
    {
        y = AESitems->gf2_8_inv[x]; // inverse to start with

        // affine transform
```

```
        y = BitSum(y&0xF1) | (BitSum(y&0xE3)<<1) | (BitSum(y&0xC7)<<2) | (BitSum(y&0x8F)<<3
) |
            (BitSum(y&0x1F)<<4) | (BitSum(y&0x3E)<<5) | (BitSum(y&0x7C)<<6) | (BitSum(y&0xF
8)<<7);
        y = y ^ 0x63;

        const_cast<unsigned char *>(AESitems->byte_sub)[x] = y;
    }
    x = 0;
    y = 0;
    return true;
}

BOOL FAR PASCAL CheckInvByteSub()
{
    if (CheckInverses() == false)
    {
        return false; // we cannot do this without inverses
    }
    if (CheckByteSub() == false)
    {
        return false; // we cannot do this without byte_sub
    }

    unsigned int x,y;
    for (x = 0; x <= 255; x++)
    {
        // we brute force it...
        y = 0;
        while (AESitems->byte_sub[y] != x)
        {
            y++;
        }

        const_cast<unsigned char *>(AESitems->inv_byte_sub)[x] = y;
    }
    x = 0;
    y = 0;
    return true;
}

BOOL FAR PASCAL CheckRcon()
{
    AESitems->Ri = 1; // start here
    AESitems->Rcon[0] = 0;

    for (int i = 1; i < sizeof(AESitems->Rcon)/sizeof(AESitems->Rcon[0])-1; i++)
    {
        AESitems->Rcon[i] = AESitems->Ri;
        AESitems->Ri = GF2_8_mult(AESitems->Ri,0x02); // multiply by x - todo replace with
xmult
    }
    return true;
} // CheckRCon

void FAR PASCAL CheckLargeTables()
{
    unsigned int i;

    for (i = 0; i < 256; i++)
    {
        AESitems->a1 = AESitems->byte_sub[i];
        AESitems->a2 = xmult(AESitems->a1);
        AESitems->a3 = AESitems->a2^AESitems->a1;

        AESitems->b5 = AESitems->inv_byte_sub[i];
        AESitems->b1 = GF2_8_mult(0x0E,AESitems->b5);
        AESitems->b2 = GF2_8_mult(0x09,AESitems->b5);
        AESitems->b3 = GF2_8_mult(0x0D,AESitems->b5);
        AESitems->b4 = GF2_8_mult(0x0B,AESitems->b5);


        AESitems->T0[i] = VEC4(AESitems->a2,AESitems->a1,AESitems->a1,AESitems->a3);
```

```
        AESitems->T1[i] = RotByteL(AESitems->T0[i]);
        AESitems->T2[i] = RotByteL(AESitems->T1[i]);
        AESitems->T3[i] = RotByteL(AESitems->T2[i]);

        AESitems->T4[i] = VEC4(AESitems->a1,0,0,0); // identity
        AESitems->T5[i] = RotByteL(AESitems->T4[i]);
        AESitems->T6[i] = RotByteL(AESitems->T5[i]);
        AESitems->T7[i] = RotByteL(AESitems->T6[i]);

        AESitems->I0[i] = VEC4(AESitems->b1,AESitems->b2,AESitems->b3,AESitems->b4);
        AESitems->I1[i] = RotByteL(AESitems->I0[i]);
        AESitems->I2[i] = RotByteL(AESitems->I1[i]);
        AESitems->I3[i] = RotByteL(AESitems->I2[i]);

        AESitems->I4[i] = VEC4(AESitems->b5,0,0,0); // identity
        AESitems->I5[i] = RotByteL(AESitems->I4[i]);
        AESitems->I6[i] = RotByteL(AESitems->I5[i]);
        AESitems->I7[i] = RotByteL(AESitems->I6[i]);
    }
    i = 0;
}

inline unsigned char FAR PASCAL GF2_8_mult(unsigned char a, unsigned char b)
{ // todo - make 4x4 table for nibbles, use lookup
    //  unsigned char
    AESitems->result1 = 0;
    // should give 0x57 . 0x13 = 0xFE with poly 0x11B
    //
    AESitems->count = 8;
    while (AESitems->count--)
    {
        if (b&1)
        {
            AESitems->result1 ^= a;
        }
        if (a&128)
        {
            a <<= 1;
            a ^= (0x1B);
        }
        else
        {
            a <<= 1;
        }
        b >>= 1;
    }
    return AESitems->result1;
} // GF2_8_mult

unsigned char FAR PASCAL BitSum(unsigned char byte)
{ // return the sum of bits mod 2
    byte = (byte>>4)^(byte&15);
    byte = (byte>>2)^(byte&3);
    return (byte>>1)^(byte&1);
} // BitSum

void FAR PASCAL AESEncrypt(unsigned long * datain, unsigned long * dataout)
{
      // we only encrypt one block from now on
    unsigned long * r_ptr = (unsigned long *)(AESitems->W);

    AESitems->dest  = AESitems->state;
    AESitems->src   = AESitems->state;

    AddRoundKey4(AESitems->dest,datain);

    Round4(AESitems->dest, AESitems->src);
    Round4(AESitems->src, AESitems->dest);
    Round4(AESitems->dest, AESitems->src);
    Round4(AESitems->src, AESitems->dest);
    Round4(AESitems->dest, AESitems->src);
    Round4(AESitems->src, AESitems->dest);
    Round4(AESitems->dest, AESitems->src);
```

```
    Round4(AESitems->src, AESitems->dest);
    Round4(AESitems->dest, AESitems->src);

    FinalRound4(dataout,AESitems->dest);

} // Encryp

inline FAR PASCAL StartEncryption(const unsigned char * key)
{
    KeyExpansion(key);
}

void FAR PASCAL KeyExpansion(const unsigned char * key)
{
    int i;
    unsigned long temp, * Wb = (unsigned long *)AESitems->W; // todo not portable - Endian
problems

    for (i = 0; i < 4*Nk; i++)
    {
        AESitems->W[i] = key[i];
    }
    for (i = Nk; i < Nb*(Nr+1); i++)
    {
        temp = Wb[i-1];
        if ((i%Nk) == 0)
        {
            temp = SubByte(RotByte(temp)) ^ AESitems->Rcon[i/Nk];
        }
        Wb[i] = Wb[i - Nk]^temp;
    }
    i = 0;
} // KeyExpansion

unsigned long FAR PASCAL SubByte(unsigned long data)
{ // does the SBox on this 4 byte data

    AESitems->result2 = 0;
    AESitems->result2 = AESitems->byte_sub[data>>24];
    AESitems->result2 <<= 8;
    AESitems->result2 |= AESitems->byte_sub[(data>>16)&255];
    AESitems->result2 <<= 8;
    AESitems->result2 |= AESitems->byte_sub[(data>>8)&255];
    AESitems->result2 <<= 8;
    AESitems->result2 |= AESitems->byte_sub[data&255];
    data = 0;
    return AESitems->result2;
} // SubByte

BOOL FAR PASCAL AES(unsigned char *fileBuffer,
                            unsigned long fileBufferLength,
                            char *passphrase, BOOL ENCRYPT)
{
    if (*passphrase == 0) return FALSE;
    hAES = HeapCreate(0, sizeof(struct AESDetails), 0);
    if (hAES == NULL)
    {
        ErrBox(errstring);  //  "Insufficient memory available");
        return TRUE;
    }

    AESitems = (struct AESDetails *) HeapAlloc1(hAES,
                                    HEAP_ZERO_MEMORY, sizeof(struct AESDetails));

    if (AESitems == NULL || hAES == NULL)
    {
        AESitems = (struct AESDetails *) HeapDestroy1(&hAES, AESitems);
        ErrBox(errstring);  //  "Insufficient memory available");
        return FALSE;
    }

    //  we do to nearest 16
```

```
    //  keylength = 128 bits
    //  blocklength = 128 bits

    if (!CreateAESTables())
    {
        AESitems = (struct AESDetails *) HeapDestroy1(&hAES, AESitems);
        ZeroMemory(passphrase, 60);
        return FALSE;
    }
    if (ENCRYPT)
    {
        StartEncryption((const unsigned char *) passphrase);
    }
    else
    {
        StartDecryption((unsigned char *) passphrase);
    }
    ZeroMemory(passphrase, 60); //  passphrase string size always 60 bytes

    unsigned int i;
    if (fileBufferLength > 400000)
    {
        fileBufferLength = 400000;
    }
    AESitems->NoTimes = fileBufferLength/16;
    //  this does not go to end of data
    //  this is ok as data is already encrypted

    AESitems->Buffer = fileBuffer+1;    //  preserve 'P'

    for (i = 0; i < AESitems->NoTimes; i++)
    {
        CopyMemory(AESitems->EntryBuffer,
            (unsigned long*) AESitems->Buffer, 16);
        if (ENCRYPT)
        {
            AESEncrypt( (AESitems->EntryBuffer),
                (AESitems->ExitBuffer));
        }
        else
        {
            AESDecrypt((AESitems->EntryBuffer),
                (AESitems->ExitBuffer));
        }
        CopyMemory(AESitems->Buffer, (char *) AESitems->ExitBuffer, 16);
        AESitems->Buffer += 16;
    }
    i = 0;
    AESitems = (struct AESDetails *) HeapDestroy1(&hAES, AESitems);
    return TRUE;
}

/*
Functions present in other files
void * FAR PASCAL HeapDestroy1(HANDLE *hDATA, void *DATA)
{
    HANDLE hD = *hDATA;
    if (hD)
    {
        unsigned int i;
        i = HeapSize(hD, 0, DATA);
        ZeroMemory(DATA, i);
        HeapDestroy(hD);
        *hDATA = 0;
        DATA = 0;
    }
    return DATA;
}

LPVOID FAR PASCAL HeapAlloc1(HANDLE hHeap, DWORD dwFlags, DWORD dwBytes)
{
    char *a;
    a = (char *) HeapAlloc(hHeap, 0, dwBytes);
```

```
    if (a == 0)
    {
        return 0;
    }

    if (dwFlags == HEAP_ZERO_MEMORY)
    {
        ZeroMemory(a, dwBytes);
    }
    return a;
}

int ErrBox(const char* format, ...)
{
    va_list argptr;
    va_start(argptr, format);
    vsprintf(String, format, argptr);
    va_end(argptr);
    MessageBox(NULL, String, "Error", MB_OK | MB_ICONEXCLAMATION);
    ZeroMemory(String, D2000);
    return 0;
}
*/
```