

```

/*****

FileName:
    KDF.h
Version:
    KDF_V1.1
Date:
    Sep 24, 2016
Description:
    This headfile provides KDF function needed in SM2 algorithm
Function List:
    1.SM3_256      //calls SM3_init、SM3_process and SM3_done to calculate hash value
    2.SM3_init     //init the SM3 state
    3.SM3_process  //compress the the first len/64 blocks of the message
    4.SM3_done     //compress the rest message and output the hash value
    5.SM3_compress //called by SM3_process and SM3_done, compress a single block of message
    6.BiToW        //called by SM3_compress,to calculate W from Bi
    7.WToW1        //called by SM3_compress, calculate W' from W
    8.CF           //called by SM3_compress, to calculate CF function.
    9.BigEndian    //called by SM3_compress and SM3_done.GM/T 0004-2012 requires to use
big-endian.
                    //if CPU uses little-endian, BigEndian function is a necessary call to
change the
                    //little-endian format into big-endian format.
    10.SM3_KDF     //calls SM3_init、SM3_process and SM3_done to generate key stream
History:
    1. Date:   Sep 18, 2016
        Author: Mao Yingying, Huo Lili
        Modification: Adding notes to all the functions
*****/

#include <string.h>

#define SM3_len 256
#define SM3_T1  0x79CC4519
#define SM3_T2  0x7A879D8A
#define SM3_IVA 0x7380166f
#define SM3_IVB 0x4914b2b9
#define SM3_IVC 0x172442d7
#define SM3_IVD 0xda8a0600
#define SM3_IVE 0xa96f30bc
#define SM3_IVF 0x163138aa

```

```

#define SM3_IVG 0xe38dee4d
#define SM3_IVH 0xb0fb0e4e

/* Various logical functions */
#define SM3_p1(x)      (x^SM3_rotl32(x, 15)^SM3_rotl32(x, 23))
#define SM3_p0(x)      (x^SM3_rotl32(x, 9)^SM3_rotl32(x, 17))
#define SM3_ff0(a, b, c)  (a^b^c)
#define SM3_ff1(a, b, c)  ((a&b)|(a&c)|(b&c))
#define SM3_gg0(e, f, g)  (e^f^g)
#define SM3_gg1(e, f, g)  ((e&f)|((~e)&g))
#define SM3_rotl32(x, n)  (((x) << n) | ((x) >> (32 - n)))
#define SM3_rotr32(x, n)  (((x) >> n) | ((x) << (32 - n)))

typedef struct {
    unsigned long  state[8];
    unsigned long  length;
    unsigned long  curlen;
    unsigned char  buf[64];
} SM3_STATE;

void BiToWj(unsigned long Bi[], unsigned long Wj[]);
void WjToWj1(unsigned long Wj[], unsigned long Wj1[]);
void CF(unsigned long Wj[], unsigned long Wj1[], unsigned long V[]);
void BigEndian(unsigned char src[], unsigned int bytelen, unsigned char des[]);
void SM3_init(SM3_STATE *md);
void SM3_compress(SM3_STATE * md);
void SM3_process(SM3_STATE * md, unsigned char buf[], int len);
void SM3_done(SM3_STATE *md, unsigned char *hash);
void SM3_256(unsigned char buf[], int len, unsigned char hash[]);
void SM3_KDF(unsigned char *Z ,unsigned short zlen,unsigned short klen,unsigned char *K);

```

```

/*****

```

```

Function:      BiToW
Description:   calculate W from Bi
Calls:
Called By:     SM3_compress
Input:         Bi[16]    //a block of a message
Output:        W[68]
Return:        null
Others:

```

\*\*\*\*\*/

```
void BiToW(unsigned long Bi[], unsigned long W[])
```

```
{
    int i;
    unsigned long tmp;

    for(i=0;i<=15;i++)
    {
        W[i]=Bi[i];
    }
    for(i=16;i<=67;i++)
    {
        tmp=W[i-16]
            ^ W[i-9]
            ^ SM3_rot132(W[i-3],15);
        W[i]=SM3_p1(tmp)
            ^ (SM3_rot132(W[i-13],7))
            ^ W[i-6];
    }
}
```

/\*\*\*\*\*/

Function:        WToW1  
Description:     calculate W1 from W  
Calls:  
Called By:       SM3\_compress  
Input:           W[68]  
Output:          W1[64]  
Return:          null  
Others:

\*\*\*\*\*/

```
void WToW1(unsigned long W[], unsigned long W1[])
```

```
{
    int i;
    for(i=0;i<=63;i++)
    {
        W1[i]=W[i]^W[i+4];
    }
}
```

/\*\*\*\*\*/

Function:        CF

Description: calculate the CF compress function and update V

Calls:

Called By: SM3\_compress

Input: W[68]

W1[64]

V[8]

Output: V[8]

Return: null

Others:

\*\*\*\*\*/

```
void CF(unsigned long W[], unsigned long W1[], unsigned long V[])
```

```
{
```

```
    unsigned long SS1;
```

```
    unsigned long SS2;
```

```
    unsigned long TT1;
```

```
    unsigned long TT2;
```

```
    unsigned long A, B, C, D, E, F, G, H;
```

```
    unsigned long T=SM3_T1;
```

```
    unsigned long FF;
```

```
    unsigned long GG;
```

```
    int j;
```

```
    //reg init,set ABCDEFGH=V0
```

```
    A=V[0];
```

```
    B=V[1];
```

```
    C=V[2];
```

```
    D=V[3];
```

```
    E=V[4];
```

```
    F=V[5];
```

```
    G=V[6];
```

```
    H=V[7];
```

```
    for(j=0; j<=63; j++)
```

```
    {
```

```
        //SS1
```

```
        if(j==0)
```

```
        {
```

```
            T=SM3_T1;
```

```
        }
```

```
        else if(j==16)
```

```
        {
```

```
            T=SM3_rot132(SM3_T2, 16);
```

```
        }
```

```
        else
```

```

{
    T=SM3_rot132(T, 1);
}
SS1=SM3_rot132((SM3_rot132(A, 12)+E+T), 7);

//SS2
SS2=SS1^SM3_rot132(A, 12);

//TT1
if(j<=15)
{
    FF=SM3_ff0(A, B, C);
}

else
{
    FF=SM3_ff1(A, B, C);
}
TT1=FF+D+SS2*W1;
W1++;

//TT2
if(j<=15)
{
    GG=SM3_gg0(E, F, G);
}

else
{
    GG=SM3_gg1(E, F, G);
}
TT2=GG+H+SS1*W;
W++;

//D
D=C;

//C
C=SM3_rot132(B, 9);

//B
B=A;

//A
A=TT1;

```

```

        //H
        H=G;

        //G
        G=SM3_rot132(F,19);

        //F
        F=E;

        //E
        E=SM3_p0(TT2);
    }

    //update V
    V[0]=A^V[0];
    V[1]=B^V[1];
    V[2]=C^V[2];
    V[3]=D^V[3];
    V[4]=E^V[4];
    V[5]=F^V[5];
    V[6]=G^V[6];
    V[7]=H^V[7];
}

/*****
Function:      BigEndian
Description:   unsigned int endian converse.GM/T 0004-2012 requires to use big-endian.
               if CPU uses little-endian, BigEndian function is a necessary
               call to change the little-endian format into big-endian format.

Calls:
Called By:     SM3_compress, SM3_done
Input:         src[bytelen]
               bytelen
Output:        des[bytelen]
Return:        null
Others:        src and des could implies the same address
*****/
void BigEndian(unsigned char src[], unsigned int bytelen, unsigned char des[])
{
    unsigned char tmp = 0;
    unsigned long i = 0;

```

```

    for(i=0; i<bytelen/4; i++)
    {
        tmp = des[4*i];
        des[4*i] = src[4*i+3];
        src[4*i+3] = tmp;

        tmp = des[4*i+1];
        des[4*i+1] = src[4*i+2];
        des[4*i+2] = tmp;
    }
}

```

/\*\*\*\*\*

Function: SM3\_init  
 Description: initiate SM3 state  
 Calls:  
 Called By: SM3\_256  
 Input: SM3\_STATE \*md  
 Output: SM3\_STATE \*md  
 Return: null  
 Others:

\*\*\*\*\*/

```

void SM3_init(SM3_STATE *md)
{
    md->curlen = md->length = 0;
    md->state[0] = SM3_IVA;
    md->state[1] = SM3_IVB;
    md->state[2] = SM3_IVC;
    md->state[3] = SM3_IVD;
    md->state[4] = SM3_IVE;
    md->state[5] = SM3_IVF;
    md->state[6] = SM3_IVG;
    md->state[7] = SM3_IVH;
}

```

/\*\*\*\*\*

Function: SM3\_compress  
 Description: compress a single block of message  
 Calls: BigEndian  
       BiToW  
       WToW1  
       CF

```

Called By:      SM3_256
Input:          SM3_STATE *md
Output:         SM3_STATE *md
Return:         null
Others:

*****/
void SM3_compress(SM3_STATE * md)
{
    unsigned long W[68];
    unsigned long W1[64];

    //if CPU uses little-endian, BigEndian function is a necessary call
    BigEndian(md->buf, 64, md->buf);

    BiToW((unsigned long *)md->buf, W);
    WToW1(W, W1);
    CF(W, W1, md->state);
}

/*****
Function:      SM3_process
Description:   compress the first (len/64) blocks of message
Calls:        SM3_compress
Called By:    SM3_256
Input:        SM3_STATE *md
               unsigned char buf[len] //the input message
               int len                //bytelen of message
Output:       SM3_STATE *md
Return:       null
Others:

*****/
void SM3_process(SM3_STATE * md, unsigned char *buf, int len)
{
    while (len--)
    {
        /* copy byte */
        md->buf[md->curlen] = *buf++;
        md->curlen++;

        /* is 64 bytes full? */
        if (md->curlen == 64)
        {
            SM3_compress(md);

```



```

        md->length += 512;
        md->curlen = 0;
    }
}
}

/*****
Function:      SM3_done
Description:   compress the rest message that the SM3_process has left behind
Calls:        SM3_compress
Called By:    SM3_256
Input:        SM3_STATE *md
Output:       unsigned char *hash
Return:       null
Others:
*****/
void SM3_done(SM3_STATE *md, unsigned char hash[])
{
    int i;
    unsigned char tmp = 0;

    /* increase the bit length of the message */
    md->length += md->curlen <<3;

    /* append the '1' bit */
    md->buf[md->curlen] = 0x80;
    md->curlen++;

    /* if the length is currently above 56 bytes, appends zeros till
       it reaches 64 bytes, compress the current block, creat a new
       block by appending zeros and length,and then compress it
       */
    if (md->curlen >56)
    {
        for (; md->curlen < 64;)
        {
            md->buf[md->curlen] = 0;
            md->curlen++;
        }
        SM3_compress(md);
        md->curlen = 0;
    }
}

```

```

/* if the length is less than 56 bytes, pad upto 56 bytes of zeroes */
for (; md->curlen < 56;)
{
    md->buf[md->curlen] = 0;
    md->curlen++;
}

/* since all messages are under 2^32 bits we mark the top bits zero */
for (i = 56; i < 60; i++)
{
    md->buf[i] = 0;
}

/* append length */
md->buf[63] = md->length & 0xff;
md->buf[62] = (md->length >> 8) & 0xff;
md->buf[61] = (md->length >> 16) & 0xff;
md->buf[60] = (md->length >> 24) & 0xff;

SM3_compress(md);

/* copy output */
memcpy(hash, md->state, SM3_len/8);
BigEndian(hash, SM3_len/8, hash); //if CPU uses little-endian, BigEndian function is a
necessary call
}

```

```

/*****
Function:      SM3_256
Description:   calculate a hash value from a given message
Calls:        SM3_init
              SM3_process
              SM3_done
Called By:
Input:        unsigned char buf[len] //the input message
              int len                //bytelen of the message
Output:       unsigned char hash[32]
Return:       null
Others:
*****/
void SM3_256(unsigned char buf[], int len, unsigned char hash[])
{

```

```

    SM3_STATE md;
    SM3_init(&md);
    SM3_process(&md, buf, len);
    SM3_done(&md, hash);
}

```

/\*\*\*\*\*

```

Function:      SM3_KDF
Description:   key derivation function
Calls:        SM3_init
              SM3_process
              SM3_done

Called By:

Input:         unsigned char Z[zlen]
              unsigned short zlen      //bytelen of Z
              unsigned short klen      //bytelen of K

Output:        unsigned char K[klen]   //shared secret key

Return:        null

Others:

```

\*\*\*\*\*/

```

void SM3_KDF( unsigned char Z[] ,unsigned short zlen,unsigned short klen,unsigned char K[])
{
    unsigned short i,j,t;
    unsigned int bitklen;
    SM3_STATE md;
    unsigned char Ha[SM2_NUMWORD] ;
    unsigned char ct[4]={0,0,0,1};

    bitklen=klen*8;

    if(bitklen%SM2_NUMBITS)
        t=bitklen/SM2_NUMBITS+1;
    else
        t=bitklen/SM2_NUMBITS;

    //s4:   K=Ha1||Ha2||...
    for(i=1;i<t;i++)
    {
    //s2:   Hai=Hv(Z||ct)
        SM3_init(&md);
        SM3_process(&md, Z, zlen);
        SM3_process(&md, ct, 4);
        SM3_done(&md, Ha);
    }
}

```

```

memcpy((K+SM2_NUMWORD*(i-1)), Ha, SM2_NUMWORD);

if(ct[3]==0xff)
{
    ct[3]=0;
    if(ct[2]==0xff)
    {
        ct[2]=0;
        if(ct[1]==0xff)
        {
            ct[1]=0;
            ct[0]++;
        }
        else ct[1]++;
    }
    else ct[2]++;
}
else ct[3]++;
}

//s3: klen/v非整数的处理
SM3_init(&md);
SM3_process(&md, Z, zlen);
SM3_process(&md, ct, 4);
SM3_done(&md, Ha);

if(bitklen%SM2_NUMBITS)
{
    i=(SM2_NUMBITS-bitklen+SM2_NUMBITS*(bitklen/SM2_NUMBITS))/8;
    j=(bitklen-SM2_NUMBITS*(bitklen/SM2_NUMBITS))/8;
    memcpy((K+SM2_NUMWORD*(t-1)), Ha, j);
}
else
{
    memcpy((K+SM2_NUMWORD*(t-1)), Ha, SM2_NUMWORD);
}
}

```