```
/*****************************************************************************
  File name:    SM2_ENC.c
  Version:      SM2_ENC_V1.1
  Date:         Sep 27,2016
  Description:  implementation of SM2 encryption algorithm and decryption algorithm
  Function List:
        1.SM2_init            //initiate SM2 curve
        2.SM2_ENC             //SM2 encryption, calls SM3_KDF
        3.SM2_DEC             //SM2 decryption, calls
SM2_KDF,Test_null,Test_Point,SM3_init,SM3_process,SM3_done
        4.SM2_ENC_SelfTest    //test whether the calculation is correct by comparing the result
with the standard data
        5.Test_Point          //test if the given point is on SM2 curve
        6.Test_Pubkey         //test if the given public key is valid
        7.Test_Null           //test if the geiven array is all zero
        8.SM2_KeyGeneration   //calculate a pubKey out of a given priKey
        9.SM3_init            //init SM3 state
        10.SM3_process        //compress the the message
        11.SM3_done           //compress the rest message and output the hash value
        12.SM3_KDF            //key deviding function base on SM3, generates key stream
 Notes:
    This SM2 implementation source code can be used for academic, non-profit making or
non-commercial use only.
    This SM2 implementation is created on MIRACL. SM2 implementation source code provider does
not provide  MIRACL library, MIRACL license or any permission to use MIRACL library. Any
commercial use of MIRACL requires a license which may be obtained from Shamus Software Ltd.
*****************************************************************************/


#include "miracl.h"
#include "mirdef.h"
#include "SM2_ENC.h"
#include "kdf.h"


/*****************************************************************
  Function:       Test_Point
  Description:    test if the given point is on SM2 curve
  Calls:
  Called By:      SM2_Decrypt, Test_PubKey
  Input:          point
  Output:         null
  Return:         0: sucess
                  3: not a valid point on curve
```

```
  Others:
*************************************************************/
int Test_Point(epoint* point)
{
    big x,y,x_3,tmp;
    x=mirvar(0);
    y=mirvar(0);
    x_3=mirvar(0);
    tmp=mirvar(0);


    //test if y^2=x^3+ax+b
    epoint_get(point,x,y);
    power (x, 3, para_p, x_3);          //x_3=x^3 mod p
    multiply (x, para_a,x);             //x=a*x
    divide (x, para_p, tmp);            //x=a*x mod p  , tmp=a*x/p
    add(x_3,x,x);                       //x=x^3+ax
    add(x,para_b,x);                    //x=x^3+ax+b
    divide(x,para_p,tmp);               //x=x^3+ax+b mod p
    power (y, 2, para_p, y);            //y=y^2 mod p
    if(compare(x,y)!=0)
        return ERR_NOT_VALID_POINT;
    else
        return 0;
}


/*************************************************************
  Function:       SM2_TestPubKey
  Description:    test if the given point is valid
  Calls:
  Called By:      SM2_Decrypt
  Input:          pubKey    //a point
  Output:         null
  Return:         0: sucess
                  1: a point at infinity
                  2: X or Y coordinate is beyond Fq
                  3: not a valid point on curve
                  4: not a point of order n
  Others:
*************************************************************/
int Test_PubKey(epoint *pubKey)
{
```

```
    big x,y,x_3,tmp;
    epoint *nP;
    x=mirvar(0);
    y=mirvar(0);
    x_3=mirvar(0);
    tmp=mirvar(0);

    nP=epoint_init();

    //test if the pubKey is the point at infinity
    if (point_at_infinity(pubKey))// if pubKey is point at infinity, return error;
        return ERR_INFINITY_POINT;

    //test if x<p   and  y<p  both hold
    epoint_get(pubKey,x,y);
    if((compare(x,para_p)!=-1)  || (compare(y,para_p)!=-1))
        return ERR_NOT_VALID_ELEMENT;

    if(Test_Point(pubKey)!=0)
        return ERR_NOT_VALID_POINT;

    //test if the order of pubKey is equal to n
    ecurve_mult(para_n,pubKey,nP);        // nP=[n]P
    if (!point_at_infinity(nP))           // if np is point NOT at infinity, return error;
        return ERR_ORDER;
    return 0;
}


/*************************************************************
  Function:       Test_Null
  Description:    test if the given array is all zero
  Calls:
  Called By:      SM2_Encrypt
  Input:          array[len]
                  len          //byte len of the array
  Output:         null
  Return:         0: the given array is not all zero
                  1: the given array is all zero

  Others:
*************************************************************/
int Test_Null(unsigned char array[],int len)
{
```

```
    int i=0;
    for(i=0;i<len;i++)
    {
        if (array[i]!=0x00)
        return 0;
    }
    return 1;
}


/*****************************************************************
  Function:       SM2_Init
  Description:    Initiate SM2 curve
  Calls:          MIRACL functions
  Called By:
  Input:          null
  Output:         null
  Return:         0: sucess;
                  7: paremeter error;
                  4: the given point G is not a point of order n
  Others:
*****************************************************************/
int SM2_Init()
{
    epoint *nG;
    para_p=mirvar(0);
    para_a=mirvar(0);
    para_b=mirvar(0);
    para_n=mirvar(0);
    para_Gx=mirvar(0);
    para_Gy=mirvar(0);
    para_h=mirvar(0);
    G=epoint_init();
    nG=epoint_init();
    bytes_to_big(SM2_NUMWORD,SM2_p,para_p);
    bytes_to_big(SM2_NUMWORD,SM2_a,para_a);
    bytes_to_big(SM2_NUMWORD,SM2_b,para_b);
    bytes_to_big(SM2_NUMWORD,SM2_n,para_n);
    bytes_to_big(SM2_NUMWORD,SM2_Gx,para_Gx);
    bytes_to_big(SM2_NUMWORD,SM2_Gy,para_Gy);
    bytes_to_big(SM2_NUMWORD,SM2_h,para_h);

    ecurve_init(para_a,para_b,para_p,MR_PROJECTIVE);//Initialises GF(p) elliptic curve.
                                                    //MR_PROJECTIVE specifying  projective
```

```
coordinates

    if (!epoint_set(para_Gx,para_Gy,0,G))//initialise point G
    {
        return ERR_ECURVE_INIT;
    }

    ecurve_mult(para_n,G,nG);
    if (!point_at_infinity(nG))    //test if the order of the point is n
    {
        return ERR_ORDER;
    }
    return 0;
}


/*************************************************************
  Function:       SM2_KeyGeneration
  Description:    calculate a pubKey out of a given priKey
  Calls:          SM2_TestPubKey
  Called By:
  Input:          priKey       // a big number lies in[1,n-2]
  Output:         pubKey       // pubKey=[priKey]G
  Return:         0: sucess
                  1: fail

  Others:
*************************************************************/
int SM2_KeyGeneration(big priKey,epoint *pubKey)
{
    int i=0;
    big x,y;
    x=mirvar(0);
    y=mirvar(0);

    ecurve_mult(priKey,G,pubKey);//通过大数和基点产生公钥
    epoint_get(pubKey,x,y);

    if(Test_PubKey(pubKey)!=0)
        return 1;
    else
        return 0;
}
```

```
/*****************************************************************
  Function:       SM2_Encrypt
  Description:    SM2 encryption
  Calls:          SM2_KDF,Test_null,Test_Point,SM3_init,SM3_process,SM3_done
  Called By:
  Input:          randK[SM2_NUMWORD]    // a random number K lies in [1,n-1]
                  pubKey                // public key of the cipher receiver
                  M[klen]               // original message
                  klen                  // byte len of original message
  Output:         C[klen+SM2_NUMWORD*3] // cipher C1||C3||C2
  Return:         0: sucess
                  1: S is point at infinity
                  5: the KDF output is all zero

  Others:
*****************************************************************/
int SM2_Encrypt(unsigned char* randK,epoint *pubKey,unsigned char M[],int klen,unsigned char
C[])
{

    big C1x,C1y,x2,y2,rand;
    epoint *C1,*kP,*S;
    int i=0;
    unsigned char x2y2[SM2_NUMWORD*2]={0};
    SM3_STATE md;
    C1x=mirvar(0);
    C1y=mirvar(0);
    x2=mirvar(0);
    y2=mirvar(0);
    rand=mirvar(0);
    C1=epoint_init();
    kP=epoint_init();
    S=epoint_init();

    //Step2.  calculate C1=[k]G=(rGx,rGy)
    bytes_to_big(SM2_NUMWORD,randK,rand);
    ecurve_mult(rand,G,C1);             //C1=[k]G
    epoint_get(C1,C1x,C1y);
    big_to_bytes(SM2_NUMWORD,C1x,C,1);
    big_to_bytes(SM2_NUMWORD,C1y,C+SM2_NUMWORD,1);

    //Step3.  test if S=[h]pubKey if the point at infinity
     ecurve_mult(para_h,pubKey,S);
```

```
    if (point_at_infinity(S))// if S is point at infinity, return error;
       return ERR_INFINITY_POINT;

    //Step4.  calculate [k]PB=(x2,y2)
    ecurve_mult(rand,pubKey,kP);     //kP=[k]P
    epoint_get(kP,x2,y2);

    //Step5.  KDF(x2||y2,klen)
   big_to_bytes(SM2_NUMWORD,x2,x2y2,1);
   big_to_bytes(SM2_NUMWORD,y2,x2y2+SM2_NUMWORD,1);
   SM3_KDF(x2y2 ,SM2_NUMWORD*2, klen,C+SM2_NUMWORD*3);
   if(Test_Null(C+SM2_NUMWORD*3,klen)!=0)
       return ERR_ARRAY_NULL;

   //Step6.  C2=M^t
   for(i=0;i<klen;i++)
   {
       C[SM2_NUMWORD*3+i]=M[i]^C[SM2_NUMWORD*3+i];
   }

   //Step7.  C3=hash(x2,M,y2)
   SM3_init(&md);
   SM3_process(&md,x2y2,SM2_NUMWORD);
   SM3_process(&md,M,klen);
   SM3_process(&md,x2y2+SM2_NUMWORD,SM2_NUMWORD);
   SM3_done(&md,C+SM2_NUMWORD*2);

   return 0;
}


/*************************************************************
  Function:       SM2_Decrypt
  Description:    SM2 decryption
  Calls:          SM2_KDF,Test_Point,SM3_init,SM3_process,SM3_done
  Called By:
  Input:          dB                    // a big number lies in [1,n-2]
                  pubKey                // [dB]G
                  C[Clen]               // cipher C1||C3||C2
                  Clen                  // byte len of cipher
  Output:         M[Clen-SM2_NUMWORD*3] // decrypted data
  Return:         0: sucess
                  1: S is a point at finity
                  3: C1 is not a valid point
```

```
                5: KDF output is all zero
                6: C3 does not match

  Others:
*************************************************************************/
int SM2_Decrypt(big dB,unsigned char C[],int Clen,unsigned char M[])
{

    SM3_STATE md;
    int i=0;
    unsigned char x2y2[SM2_NUMWORD*2]={0};
    unsigned char hash[SM2_NUMWORD]={0};
    big C1x,C1y,x2,y2;
    epoint *C1,*S,*dBC1;
    C1x=mirvar(0);
    C1y=mirvar(0);
    x2=mirvar(0);
    y2=mirvar(0);
    C1=epoint_init();
    S=epoint_init();
    dBC1=epoint_init();

    //Step1.  test if C1 fits the curve
    bytes_to_big(SM2_NUMWORD,C,C1x);
    bytes_to_big(SM2_NUMWORD,C+SM2_NUMWORD,C1y);
    epoint_set(C1x,C1y,0,C1);
    i=Test_Point(C1);
    if(i!=0)
        return i;

    //Step2.  S=[h]C1 and test if S is the point at infinity
    ecurve_mult(para_h,C1,S);
    if (point_at_infinity(S))// if S is point at infinity, return error;
        return ERR_INFINITY_POINT;

    //Step3.  [dB]C1=(x2,y2)
    ecurve_mult(dB,C1,dBC1);
    epoint_get(dBC1,x2,y2);
    big_to_bytes(SM2_NUMWORD,x2,x2y2,1);
    big_to_bytes(SM2_NUMWORD,y2,x2y2+SM2_NUMWORD,1);

    //Step4.  t=KDF(x2||y2,klen)
    SM3_KDF(x2y2,SM2_NUMWORD*2,Clen-SM2_NUMWORD*3,M);
    if(Test_Null(M,Clen-SM2_NUMWORD*3)!=0)
```

```
        return ERR_ARRAY_NULL;

    //Step5.  M=C2^t
    for(i=0;i<Clen-SM2_NUMWORD*3;i++)
        M[i]=M[i]^C[SM2_NUMWORD*3+i];

    //Step6. hash(x2,m,y2)
    SM3_init(&md);
    SM3_process(&md,x2y2,SM2_NUMWORD);
    SM3_process(&md,M,Clen-SM2_NUMWORD*3);
    SM3_process(&md,x2y2+SM2_NUMWORD,SM2_NUMWORD);
    SM3_done(&md,hash);
    if(memcmp(hash,C+SM2_NUMWORD*2,SM2_NUMWORD)!=0)
        return ERR_C3_MATCH;
    else
        return 0;

}


/*************************************************************
  Function:       SM2_ENC_SelfTest
  Description:    test whether the SM2 calculation is correct by comparing the result with the
standard data
  Calls:          SM2_init,SM2_ENC,SM2_DEC
  Called By:
  Input:          NULL
  Output:         NULL
  Return:         0: sucess
                  1: S is a point at finity
                  2: X or Y coordinate is beyond Fq
                  3: not a valid point on curve
                  4: the given point G is not a point of order n
                  5: KDF output is all zero
                  6: C3 does not match
                  8: public key generation error
                  9: SM2 encryption error
                  a: SM2 decryption error

  Others:
*************************************************************/
int SM2_ENC_SelfTest()
{
    int tmp=0,i=0;
```

```
    unsigned char Cipher[115]={0};
    unsigned char M[19]={0};
    unsigned char kGxy[SM2_NUMWORD*2]={0};
    big ks,x,y;
    epoint *kG;

    //standard data
    unsigned char
std_priKey[32]={0x39,0x45,0x20,0x8F,0x7B,0x21,0x44,0xB1,0x3F,0x36,0xE3,0x8A,0xC6,0xD3,0x9F,0
x95,

0x88,0x93,0x93,0x69,0x28,0x60,0xB5,0x1A,0x42,0xFB,0x81,0xEF,0x4D,0xF7,0xC5,0xB8};
    unsigned char
std_pubKey[64]={0x09,0xF9,0xDF,0x31,0x1E,0x54,0x21,0xA1,0x50,0xDD,0x7D,0x16,0x1E,0x4B,0xC5,0
xC6,

0x72,0x17,0x9F,0xAD,0x18,0x33,0xFC,0x07,0x6B,0xB0,0x8F,0xF3,0x56,0xF3,0x50,0x20,

0xCC,0xEA,0x49,0x0C,0xE2,0x67,0x75,0xA5,0x2D,0xC6,0xEA,0x71,0x8C,0xC1,0xAA,0x60,

0x0A,0xED,0x05,0xFB,0xF3,0x5E,0x08,0x4A,0x66,0x32,0xF6,0x07,0x2D,0xA9,0xAD,0x13};
    unsigned char
std_rand[32]={0x59,0x27,0x6E,0x27,0xD5,0x06,0x86,0x1A,0x16,0x68,0x0F,0x3A,0xD9,0xC0,0x2D,0xC
C,

0xEF,0x3C,0xC1,0xFA,0x3C,0xDB,0xE4,0xCE,0x6D,0x54,0xB8,0x0D,0xEA,0xC1,0xBC,0x21};
    unsigned char
std_Message[19]={0x65,0x6E,0x63,0x72,0x79,0x70,0x74,0x69,0x6F,0x6E,0x20,0x73,0x74,0x61,0x6E,
                                0x64,0x61,0x72,0x64};
    unsigned char
std_Cipher[115]={0x04,0xEB,0xFC,0x71,0x8E,0x8D,0x17,0x98,0x62,0x04,0x32,0x26,0x8E,0x77,0xFE,
0xB6,

0x41,0x5E,0x2E,0xDE,0x0E,0x07,0x3C,0x0F,0x4F,0x64,0x0E,0xCD,0x2E,0x14,0x9A,0x73,

0xE8,0x58,0xF9,0xD8,0x1E,0x54,0x30,0xA5,0x7B,0x36,0xDA,0xAB,0x8F,0x95,0x0A,0x3C,

0x64,0xE6,0xEE,0x6A,0x63,0x09,0x4D,0x99,0x28,0x3A,0xFF,0x76,0x7E,0x12,0x4D,0xF0,

0x59,0x98,0x3C,0x18,0xF8,0x09,0xE2,0x62,0x92,0x3C,0x53,0xAE,0xC2,0x95,0xD3,0x03,

0x83,0xB5,0x4E,0x39,0xD6,0x09,0xD1,0x60,0xAF,0xCB,0x19,0x08,0xD0,0xBD,0x87,0x66,

0x21,0x88,0x6C,0xA9,0x89,0xCA,0x9C,0x7D,0x58,0x08,0x73,0x07,0xCA,0x93,0x09,0x2D,0x65,0x1E,0x
```

```
FA};

    mip= mirsys(1000, 16);
    mip->IOBASE=16;
    x=mirvar(0);
    y=mirvar(0);
    ks=mirvar(0);
    kG=epoint_init();
    bytes_to_big(32,std_priKey,ks);  //ks is the standard private key

    //initiate SM2 curve
    SM2_Init();

    //generate key pair
    tmp=SM2_KeyGeneration(ks,kG);
    if (tmp!=0)
        return tmp;
    epoint_get(kG,x,y);
    big_to_bytes(SM2_NUMWORD,x,kGxy,1);
    big_to_bytes(SM2_NUMWORD,y,kGxy+SM2_NUMWORD,1);
    if(memcmp(kGxy,std_pubKey,SM2_NUMWORD*2)!=0)
        return ERR_SELFTEST_KG;

    //encrypt data and compare the result with the standard data
    tmp=SM2_Encrypt(std_rand,kG,std_Message,19,Cipher);
    if(tmp!=0)
        return tmp;
    if(memcmp(Cipher,std_Cipher,19+SM2_NUMWORD*3)!=0)
        return ERR_SELFTEST_ENC;

    //decrypt cipher and compare the result with the standard data
    tmp=SM2_Decrypt(ks,Cipher,115,M);
    if(tmp!=0)
        return tmp;
    if(memcmp(M,std_Message,19)!=0)
        return ERR_SELFTEST_DEC;

    return 0;
}
```