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

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); multiply (x, para\_a,x);

divide (x, para\_p, tmp);

add(x\_3,x,x);

add(x,para\_b,x);

divide(x,para\_p,tmp);

power (y, 2, para\_p, y); if(compare(x,y)!=0)

return ERR\_NOT\_VALID\_POINT; else

return 0; }

//x\_3=x^3 mod p //x=a\*x

//x=a\*x mod p , tmp=a\*x/p

//x=x^3+ax

//x=x^3+ax+b

//x=x^3+ax+b mod p //y=y^2 mod p

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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]

pubKey M[klen] klen

// a random number K lies in [1,n-1] // public key of the cipher receiver // original message

// 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

pubKey C[Clen] Clen

// a big number lies in [1,n-2] // [dB]G

// cipher C1 |C3||C2 // 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; }