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

File name: SM2\_sv.c

Version: SM2\_sv\_V1.0

Date: Sep 27,2016

Description: implementation of SM2 signature algorithm and verification algorithm Function List:

|  |  |
| --- | --- |
| 1.SM2\_Init | //initiate SM2 curve |
| 2.Test\_Point | //test if the given point is on SM2 curve |
| 3.Test\_PubKey | //test if the given public key is valid |
| 4.Test\_Zero | //test if the big x equals zero |
| 5.Test\_n | //test if the big x equals n |
| 6.Test\_Range | //test if the big x belong to the range[1,n-1] |
| 7.SM2\_KeyGeneration | //generate public key |
| 8.SM2\_Sign | //SM2 signature algorithm |
| 9.SM2\_Verify | //SM2 verification |
| 10.SM2\_SelfCheck() | //SM2 slef-check |
| 11.SM3\_256() | //this function can be found in SM3.c and SM3.h |

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 "SM2\_sv.h" #include "KDF.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_Init

Description: Initiate SM2 curve

Calls: MIRACL functions

Called By: SM2\_KeyGeneration,SM2\_Sign,SM2\_Verify,SM2\_SelfCheck

Input: null

Output: null

Return: 0: sucess;

1: parameter initialization error;

4: the given point G is not a point of order n

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int SM2\_Init()

{

Gx=mirvar(0); Gy=mirvar(0); p=mirvar(0); a=mirvar(0); b=mirvar(0); n=mirvar(0);

bytes\_to\_big(SM2\_NUMWORD,SM2\_Gx,Gx); bytes\_to\_big(SM2\_NUMWORD,SM2\_Gy,Gy); bytes\_to\_big(SM2\_NUMWORD,SM2\_p,p);

bytes\_to\_big(SM2\_NUMWORD,SM2\_a,a); bytes\_to\_big(SM2\_NUMWORD,SM2\_b,b); bytes\_to\_big(SM2\_NUMWORD,SM2\_n,n);

ecurve\_init(a,b,p,MR\_PROJECTIVE);

G=epoint\_init(); nG=epoint\_init();

if (!epoint\_set(Gx,Gy,0,G))//initialise point G {

return ERR\_ECURVE\_INIT; }

ecurve\_mult(n,G,nG);

if (!point\_at\_infinity(nG)) //test if the order of the point is n

{

return ERR\_ORDER; }

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: Test\_Point

Description: test if the given point is on SM2 curve

Calls:

Called By: SM2\_KeyGeneration

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, p, x\_3); multiply (x, a,x);

divide (x, p, tmp); add(x\_3,x,x);

add(x,b,x);

divide(x,p,tmp);

power (y, 2, p, y); if(compare(x,y)!=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

return ERR\_NOT\_VALID\_POINT; else

return 0; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: Test\_PubKey

Description: test if the given public key is valid

Calls:

Called By: SM2\_KeyGeneration

Input: pubKey //a point

Output: null

Return: 0: sucess

2: a point at infinity

5: 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,p)!=-1) || (compare(y,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(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\_Zero

Description: test if the big x is zero

Calls:

Called By: SM2\_Sign

|  |  |  |
| --- | --- | --- |
| Input: | pubKey | //a point |
| Output: | null |  |
| Return: | 0: x!=0 |  |

1: x==0

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int Test\_Zero(big x)

{

big zero;

zero=mirvar(0);

if(compare(x,zero)==0)

return 1; else return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: Test\_n

Description: test if the big x is order n

Calls:

Called By: SM2\_Sign

Input: big x //a miracl data type

Output: null

Return: 0: sucess

1: x==n,fail

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int Test\_n(big x)

{

// bytes\_to\_big(32,SM2\_n,n); if(compare(x,n)==0)

return 1; else return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: Test\_Range

Description: test if the big x belong to the range[1,n-1]

Calls:

Called By: SM2\_Verify

Input: big x ///a miracl data type

Output: null

Return: 0: sucess

1: fail

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int Test\_Range(big x)

{

big one,decr\_n;

one=mirvar(0);

decr\_n=mirvar(0);

convert(1,one);

decr(n,1,decr\_n);

if( (compare(x,one) < 0) | (compare(x,decr\_n)>0) ) return 1;

return 0; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_KeyGeneration

Description: calculate a pubKey out of a given priKey

Calls: SM2\_SelfCheck()

|  |  |  |
| --- | --- | --- |
| Called By: | SM2\_Init() |  |
| Input: | priKey | // a big number lies in[1,n-2] |
| Output: | pubKey | // pubKey=[priKey]G |
| Return: | 0: sucess |  |

2: a point at infinity

5: X or Y coordinate is beyond Fq 3: not a valid point on curve

4: not a point of order n

Others:

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

int SM2\_KeyGeneration(unsigned char PriKey[],unsigned char Px[],unsigned char Py[]) {

int i=0;

big d,PAx,PAy; epoint \*PA;

SM2\_Init();

PA=epoint\_init();

d=mirvar(0);

PAx=mirvar(0); PAy=mirvar(0);

bytes\_to\_big(SM2\_NUMWORD,PriKey,d);

ecurve\_mult(d,G,PA);

epoint\_get(PA,PAx,PAy);

big\_to\_bytes(SM2\_NUMWORD,PAx,Px,TRUE); big\_to\_bytes(SM2\_NUMWORD,PAy,Py,TRUE);

i=Test\_PubKey(PA); if(i)

return i; else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_Sign

Description: SM2 signature algorithm

Calls: SM2\_Init(),Test\_Zero(),Test\_n(), SM3\_256()

Called By: SM2\_SelfCheck()

Input:

message len

ZA

rand

d

//the message to be signed //the length of message

// ZA=Hash(ENTLA|| IDA| a|| b|| Gx || Gy || xA | yA) //a random number K lies in [1,n-1]

//the private key

Output: R,S //signature result

Return: 0: sucess

1: parameter initialization error;

4: the given point G is not a point of order n 6: the signed r equals 0 or r+rand equals n

7 the signed s equals 0

Others:

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

int SM2\_Sign(unsigned char \*message,int len,unsigned char ZA[],unsigned char rand[],unsigned char d[],unsigned char R[],unsigned char S[])

{

unsigned char hash[SM3\_len/8]; int M\_len=len+SM3\_len/8;

unsigned char \*M=NULL; int i;

big dA,r,s,e,k,KGx,KGy; big rem,rk,z1,z2;

epoint \*KG;

i=SM2\_Init();

if(i) return i;

//initiate

dA=mirvar(0); e=mirvar(0);

k=mirvar(0);

KGx=mirvar(0); KGy=mirvar(0); r=mirvar(0);

s=mirvar(0);

rem=mirvar(0); rk=mirvar(0); z1=mirvar(0); z2=mirvar(0);

bytes\_to\_big(SM2\_NUMWORD,d,dA);//cinstr(dA,d);

KG=epoint\_init();

//step1,set M=ZA||M

M=(char \*)malloc(sizeof(char)\*(M\_len+1)); memcpy(M,ZA,SM3\_len/8);

memcpy(M+SM3\_len/8,message,len);

//step2,generate e=H(M) SM3\_256(M, M\_len, hash);

bytes\_to\_big(SM3\_len/8,hash,e);

//step3:generate k

bytes\_to\_big(SM3\_len/8,rand,k);

//step4:calculate kG ecurve\_mult(k,G,KG);

//step5:calculate r

epoint\_get(KG,KGx,KGy); add(e,KGx,r);

divide(r,n,rem);

//judge r=0 or n+k=n? add(r,k,rk);

if( Test\_Zero(r) | Test\_n(rk))

return ERR\_GENERATE\_R;

//step6:generate s incr(dA,1,z1);

xgcd(z1,n,z1,z1,z1); multiply(r,dA,z2);

divide(z2,n,rem); subtract(k,z2,z2);

add(z2,n,z2);

multiply(z1,z2,s); divide(s,n,rem);

//judge s=0?

if(Test\_Zero(s))

return ERR\_GENERATE\_S ;

big\_to\_bytes(SM2\_NUMWORD,r,R,TRUE); big\_to\_bytes(SM2\_NUMWORD,s,S,TRUE);

free(M); return 0;

}

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

|  |  |  |
| --- | --- | --- |
| Function: | SM2\_Verify |  |
| Description: | SM2 verification algorithm |  |
| Calls: | SM2\_Init(),Test\_Range(), Test\_Zero(),SM3\_256() |  |
| Called By: | SM2\_SelfCheck() |  |
| Input: | message //the message to be signed  len //the length of message  ZA //ZA=Hash(ENTLA|| IDA | a | b|| Gx || Gy || xA|  Px,Py //the public key |  |
| yA) |
|  |

R,S //signature result

Output:

Return: 0: sucess

1: parameter initialization error;

4: the given point G is not a point of order n B: public key error

8: the signed R out of range [1,n-1] 9: the signed S out of range [1,n-1] A: the intermediate data t equals 0 C: verification fail

Others:

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

int SM2\_Verify(unsigned char \*message,int len,unsigned char ZA[],unsigned char Px[],unsigned char Py[],unsigned char R[],unsigned char S[])

{

unsigned char hash[SM3\_len/8]; int M\_len=len+SM3\_len/8;

unsigned char \*M=NULL; int i;

big PAx,PAy,r,s,e,t,rem,x1,y1; big RR;

epoint \*PA,\*sG,\*tPA;

i=SM2\_Init();

if(i) return i;

PAx=mirvar(0); PAy=mirvar(0); r=mirvar(0);

s=mirvar(0);

e=mirvar(0);

t=mirvar(0);

x1=mirvar(0); y1=mirvar(0); rem=mirvar(0); RR=mirvar(0);

PA=epoint\_init(); sG=epoint\_init(); tPA=epoint\_init();

bytes\_to\_big(SM2\_NUMWORD,Px,PAx); bytes\_to\_big(SM2\_NUMWORD,Py,PAy);

bytes\_to\_big(SM2\_NUMWORD,R,r); bytes\_to\_big(SM2\_NUMWORD,S,s);

if (!epoint\_set(PAx,PAy,0,PA))//initialise public key {

return ERR\_PUBKEY\_INIT; }

//step1: test if r belong to [1,n-1] if (Test\_Range(r))

return ERR\_OUTRANGE\_R;

//step2: test if s belong to [1,n-1]

if (Test\_Range(s))

return ERR\_OUTRANGE\_S;

//step3,generate M

M=(char \*)malloc(sizeof(char)\*(M\_len+1)); memcpy(M,ZA,SM3\_len/8);

memcpy(M+SM3\_len/8,message,len);

//step4,generate e=H(M) SM3\_256(M, M\_len, hash);

bytes\_to\_big(SM3\_len/8,hash,e);

//step5:generate t add(r,s,t);

divide(t,n,rem);

if( Test\_Zero(t))

return ERR\_GENERATE\_T;

//step 6: generate(x1,y1) ecurve\_mult(s,G,sG);

ecurve\_mult(t,PA,tPA); ecurve\_add(sG,tPA);

epoint\_get(tPA,x1,y1);

//step7:generate RR add(e,x1,RR);

divide(RR,n,rem);

free(M);

if(compare(RR,r)==0) return 0;

else

return ERR\_DATA\_MEMCMP; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_SelfCheck

Description: SM2 self check

Calls: SM2\_Init(), SM2\_KeyGeneration,SM2\_Sign, SM2\_Verify,SM3\_256()

Called By: Input:

Output:

Return: 0: sucess

1: paremeter initialization error 2: a point at infinity

5: X or Y coordinate is beyond Fq 3: not a valid point on curve

4: not a point of order n

B: public key error

8: the signed R out of range [1,n-1] 9: the signed S out of range [1,n-1] A: the intermediate data t equals 0 C: verification fail

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int SM2\_SelfCheck()

{

//the private key unsigned char

dA[32]={0x39,0x45,0x20,0x8f,0x7b,0x21,0x44,0xb1,0x3f,0x36,0xe3,0x8a,0xc6,0xd3,0x9f,

0x95,0x88,0x93,0x93,0x69,0x28,0x60,0xb5,0x1a,0x42,0xfb,0x81,0xef,0x4d,0xf7,0xc5,0xb8};

unsigned char

rand[32]={0x59,0x27,0x6E,0x27,0xD5,0x06,0x86,0x1A,0x16,0x68,0x0F,0x3A,0xD9,0xC0,0x2D, 0xCC,0xEF,0x3C,0xC1,0xFA,0x3C,0xDB,0xE4,0xCE,0x6D,0x54,0xB8,0x0D,0xEA,0xC1,0xBC,0x21};

//the public key /\* unsigned char

xA[32]={0x09,0xf9,0xdf,0x31,0x1e,0x54,0x21,0xa1,0x50,0xdd,0x7d,0x16,0x1e,0x4b,0xc5,

0xc6,0x72,0x17,0x9f,0xad,0x18,0x33,0xfc,0x07,0x6b,0xb0,0x8f,0xf3,0x56,0xf3,0x50,0x20};

unsigned char

yA[32]={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 xA[32],yA[32];

unsigned char r[32],s[32];// Signature

unsigned char IDA[16]={0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x31,0x32,0x33,

0x34,0x35,0x36,0x37,0x38};//ASCII code of userA's identification int IDA\_len=16;

unsigned char ENTLA[2]={0x00,0x80};//the length of userA's identification,presentation in ASCII code

unsigned char \*message="message digest";//the message to be signed int len=strlen(message);//the length of message

unsigned char ZA[SM3\_len/8];//ZA=Hash(ENTLA| IDA| a|| b| Gx || Gy || xA | yA)

unsigned char Msg[210]; //210=IDA\_len+2+SM2\_NUMWORD\*6

int temp;

miracl \*mip=mirsys(10000,16); mip->IOBASE=16;

temp=SM2\_KeyGeneration(dA,xA,yA); if(temp)

return temp;

// ENTLA|| IDA|| a|| b|| Gx || Gy || xA| yA

memcpy(Msg,ENTLA,2);

memcpy(Msg+2,IDA,IDA\_len);

memcpy(Msg+2+IDA\_len,SM2\_a,SM2\_NUMWORD);

memcpy(Msg+2+IDA\_len+SM2\_NUMWORD,SM2\_b,SM2\_NUMWORD);

memcpy(Msg+2+IDA\_len+SM2\_NUMWORD\*2,SM2\_Gx,SM2\_NUMWORD); memcpy(Msg+2+IDA\_len+SM2\_NUMWORD\*3,SM2\_Gy,SM2\_NUMWORD);

memcpy(Msg+2+IDA\_len+SM2\_NUMWORD\*4,xA,SM2\_NUMWORD); memcpy(Msg+2+IDA\_len+SM2\_NUMWORD\*5,yA,SM2\_NUMWORD); SM3\_256(Msg,210,ZA);

temp=SM2\_Sign(message,len,ZA,rand,dA,r,s); if(temp)

return temp;

temp=SM2\_Verify(message,len,ZA,xA,yA,r,s); if(temp)

return temp;

return 0; }