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

File name: SM2\_KEY\_EX.c

Version: V1.1

Date: Oct 9,2016

Description: implementation of SM2 Key Exchange Protocol Function List:

|  |  |
| --- | --- |
| 1.SM2\_Init | // initiate SM2 curve，should be called before any calculation on curve. |
| 2.SM2\_KeyEx\_Init\_I | // Step A1 to A3,the first host (initiator A) generates a random number |
| ra and |  |
|  | // calculates point RA which the second host(responder B) receives |
| 3.SM2\_KeyEx\_Re\_I | // Step B1 to B9, responder B generates RB, and calculates a secret |
| shared key |  |
|  | // out of RA and RB, RB should be sent the initiator A |
| 4.SM2\_KeyEx\_Init\_II | // Step A4 to A10, initiator A calculates the secret key out of RA and |
| RB, and calculates a hash |  |
|  | // value which responder B might verifies |
| 5.SM2\_KeyEx\_Re\_II | // Step B10 (optional) verifies the hash value received from initiator |
| A |  |
| 6.SM2\_KeyEX\_SelfTest | // test whether the calculation is correct by comparing the result with |
| the standard data |  |
| 7.SM2\_W | // calculation of w |
| 8.SM3\_Z | // calculation of ZA or ZB |
| 9.Test\_Point | // test if the given point is on SM2 curve |
| 10.Test\_Pubkey | // test if the given public key is valid |
| 11.SM2\_KeyGeneration | //calculate a pubKey out of a given priKey |

Declaration:

The SM2 algorithm source code is for academic, non-profit or non-commercial use only. SM2 implementation is based on MIRACL whose copyright belongs to Shamus Software Ltd. We are in no position to provide MIRACL library or any permission to use it. For commercial use, please apply to Shamus Software Ltd for a license.

Notes:

The MIRACL system must be initialized before attempting to use any other MIRACL routines. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <malloc.h>

#include "SM2\_KEY\_EX.h" #include "kdf.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_W

Description: calculation of w

Calls:

Called By: SM2\_KeyEx\_Re\_I, SM2\_KeyEx\_Init\_II

|  |  |  |
| --- | --- | --- |
| Input: | n | // a big number |
| Output: | null |  |
| Return: | w |  |

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int SM2\_W(big n)

{

big n1; int w=0;

n1=mirvar(0);

w=logb2 (para\_n); //approximate integer log to the base 2 of para\_n

expb2 (w, n1); //n1=2^w

if(compare(para\_n,n1)==1) w++;

if((w%2)==0) w=w/2-1;

else

w=(w+1)/2-1;

return w; }

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

Description: test if the given point is on SM2 curve

Calls:

Called By: Test\_PubKey

Input: point

Output: null

Return: 0: sucess

1: 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 1; 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: Test\_PubKey

Description: test if the given public key is valid

Calls:

Called By: SM2\_KeyGeneration

Input: pubKey //a public key

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: SM3\_Z

Description: calculation of ZA or ZB

Calls: SM3\_init, SM3\_process, SM3\_done

Called By: SM2\_KeyEX\_SelfTest

|  |  |  |
| --- | --- | --- |
| Input: | ID[ELAN/8] ELAN  pubKey | // bit len of ID // public key |
| Output: | hash[SM3\_len/8] | // Z=hash(ELAN| ID||a | b |Gx |Gy |Px |Py) |
| Return: | null |  |

Others:

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

void SM3\_Z(unsigned char ID[], unsigned short int ELAN, epoint\* pubKey, unsigned char hash[]) {

unsigned char Px[SM2\_NUMWORD]={0},Py[SM2\_NUMWORD]={0};

unsigned char IDlen[2]={0}; big x,y;

SM3\_STATE md;

x=mirvar(0); y=mirvar(0);

epoint\_get(pubKey,x,y);

big\_to\_bytes(SM2\_NUMWORD,x,Px,1); big\_to\_bytes(SM2\_NUMWORD,y,Py,1);

memcpy(IDlen,&(unsigned char)ELAN+1,1); memcpy(IDlen+1,&(unsigned char)ELAN,1);

SM3\_init(&md);

SM3\_process(&md,IDlen,2);

SM3\_process(&md,ID,ELAN/8);

SM3\_process(&md,SM2\_a,SM2\_NUMWORD); SM3\_process(&md,SM2\_b,SM2\_NUMWORD); SM3\_process(&md,SM2\_Gx,SM2\_NUMWORD); SM3\_process(&md,SM2\_Gy,SM2\_NUMWORD); SM3\_process(&md,Px,SM2\_NUMWORD);

SM3\_process(&md,Py,SM2\_NUMWORD); SM3\_done(&md,hash);

return; }

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

Description: Initiate SM2 curve

Calls: MIRACL functions

Called By: SM2\_KeyEX\_SelfTest

Input: null

Output: null

Return: 0: sucess;

5: parameter 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: 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 SM2\_KeyGeneration(big priKey,epoint \*pubKey)

{

int i=0;

big x,y;

x=mirvar(0); y=mirvar(0);

//mip= mirsys(1000, 16);

//mip->IOBASE=16;

ecurve\_mult(priKey,G,pubKey);//通过大数和基点产生公钥 epoint\_get(pubKey,x,y);

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

return i; else

return 0; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_KeyExchange\_Init\_I

Description: calculate RA

Calls: SM2\_KeyGeneration

Called By:

|  |  |  |
| --- | --- | --- |
| Input: | ra | // a big number lies in[1,n-1] |
| Output: | RA | // RA=[ra]G |
| 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 SM2\_KeyEx\_Init\_I(big ra, epoint\* RA)

{

return SM2\_KeyGeneration(ra,RA); }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_KeyEx\_Re\_I

Description: calculate RB and a secret key

Calls: SM2\_W, SM2\_KeyGeneration, SM3\_init, SM3\_process, SM3\_done

Called By:

Input:

rb dB RA PA ZA ZB

// a big number lies in[1,n-1] // private key of responder B

// temporary public key received from initiator A // public key of initiator A

// Z=hash(ELAN\_A||ID of A||a ||b||Gx||Gy||PAx |PAy) // Z=hash(ELAN\_B||ID of B||a ||b||Gx||Gy||PBx||PBy)

|  |  |  |
| --- | --- | --- |
|  | klen | // byte len of the secret key that A and B wanna share |
| Output: | K  RB  V  hash | // secret key that A and B wanna share // RB=[rb]G  // V=[h\*tB](PA+[x1\_]RA),in function SM2\_KeyEx\_Re\_II it as input // (option) calculates a hash value SB that initiator A might |
| verifies |  |  |
| Output: |  |  |
| Return: | 0: sucess |  |

1: a point at infinity 6: RA is not valid

Others:

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

int SM2\_KeyEx\_Re\_I(big rb, big dB, epoint\* RA, epoint\* PA, unsigned char ZA[],unsigned char ZB[],unsigned char K[],int klen,epoint\* RB, epoint\* V,unsigned char hash[])

{

SM3\_STATE md; int i=0,w=0;

unsigned char Z[SM2\_NUMWORD\*2+SM3\_len/4]={0}; unsigned char x1y1[SM2\_NUMWORD\*2]={0};

unsigned char x2y2[SM2\_NUMWORD\*2]={0}; unsigned char temp=0x02;

big x1,y1,x1\_,x2,y2,x2\_,tmp,Vx,Vy,temp\_x,temp\_y;

//mip= mirsys(1000, 16); //mip->IOBASE=16;

x1=mirvar(0);

y1=mirvar(0);

x1\_=mirvar(0);

x2=mirvar(0);

y2=mirvar(0);

x2\_=mirvar(0);

tmp=mirvar(0);

Vx=mirvar(0);

Vy=mirvar(0);

temp\_x=mirvar(0); temp\_y=mirvar(0);

w=SM2\_W(para\_n);

//--------B2: RB=[rb]G=(x2,y2)-------- SM2\_KeyGeneration(rb,RB);

epoint\_get(RB,x2,y2);

big\_to\_bytes(SM2\_NUMWORD,x2,x2y2,1);

big\_to\_bytes(SM2\_NUMWORD,y2,x2y2+SM2\_NUMWORD,1);

//--------B3: x2\_=2^w+x2 & (2^w-1)--------

expb2 (w,x2\_); // X2\_=2^w

divide(x2,x2\_,tmp); // x2=x2 mod x2\_=x2 & (2^w-1)

add(x2\_,x2,x2\_);

divide(x2\_,para\_n,tmp); // x2\_=n mod q

//--------B4: tB=(dB+x2\_\*rB)mod n-------- multiply(x2\_,rb,x2\_);

add(dB,x2\_,x2\_);

divide(x2\_,para\_n,tmp);

//--------B5: x1\_=2^w+x1 & (2^w-1)-------- if(Test\_Point(RA)!=0)

return ERR\_KEYEX\_RA;

epoint\_get(RA,x1,y1);

big\_to\_bytes(SM2\_NUMWORD,x1,x1y1,1);

big\_to\_bytes(SM2\_NUMWORD,y1,x1y1+SM2\_NUMWORD,1);

expb2 (w,x1\_); // X1\_=2^w

divide(x1,x1\_,tmp); // x1=x1 mod x1\_ =x1 & (2^w-1)

add(x1\_,x1,x1\_);

divide(x1\_,para\_n,tmp); // x1\_=n mod q

//--------B6: V=[h\*tB](PA+[x1\_]RA)-------- ecurve\_mult(x1\_,RA,V); // v=[x1\_]RA

epoint\_get(V,temp\_x,temp\_y);

ecurve\_add(PA,V); // V=PA+V

epoint\_get(V,temp\_x,temp\_y);

multiply(para\_h,x2\_,x2\_); // tB=tB\*h

ecurve\_mult(x2\_,V,V);

if(point\_at\_infinity(V)==1)

return ERR\_INFINITY\_POINT;

epoint\_get(V,Vx,Vy);

big\_to\_bytes(SM2\_NUMWORD,Vx,Z,1);

big\_to\_bytes(SM2\_NUMWORD,Vy,Z+SM2\_NUMWORD,1);

//------------B7:KB=KDF(VX,VY,ZA,ZB,KLEN)----------

memcpy(Z+SM2\_NUMWORD\*2,ZA,SM3\_len/8);

memcpy(Z+SM2\_NUMWORD\*2+SM3\_len/8,ZB,SM3\_len/8); SM3\_KDF(Z,SM2\_NUMWORD\*2+SM3\_len/4,klen/8,K);

//---------------B8:(optional)

SB=hash(0x02||Vy||HASH(Vx||ZA||ZB| x1| y1| x2| y2)-------------

SM3\_init (&md);

SM3\_process(&md,Z,SM2\_NUMWORD);

SM3\_process(&md,ZA,SM3\_len/8);

SM3\_process(&md,ZB,SM3\_len/8);

SM3\_process(&md,x1y1,SM2\_NUMWORD\*2); SM3\_process(&md,x2y2,SM2\_NUMWORD\*2); SM3\_done(&md, hash);

SM3\_init(&md);

SM3\_process(&md,&temp,1);

SM3\_process(&md,Z+SM2\_NUMWORD,SM2\_NUMWORD); SM3\_process(&md,hash,SM3\_len/8);

SM3\_done(&md, hash);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_KeyEx\_Init\_II

Description: initiator A calculates the secret key out of RA and RB, and calculates a hash

value which responder B might verifies

Calls: SM2\_W,SM3\_init, SM3\_process, SM3\_done,KDF\_lib

Called By:

Input:

ra

dA

RA

RB

PB

ZA

ZB

klen SB

// a big number lies in[1,n-1] // private key of initiator A

// temporary public key received from initiator A // temporary public key received from initiator B // public key of initiator A

// Z=hash(ELAN\_A||ID of A |a||b||Gx||Gy||PAx||PAy)

// Z=hash(ELAN\_B||ID of B |a||b||Gx||Gy||PBx||PBy)

// byte len of the secret key that A and B wanna share // a hash value calculated by initiator B

Output: K // secret key that A and B wanna share

SA // (option) calculates a hash value SA that initiator B might

verifies

Return: 0: sucess

1: a point at infinity 7: RB is not valid

8: key validation failed,form B to A,S1!=SB

Others:

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

int SM2\_KeyEx\_Init\_II(big ra, big dA, epoint\* RA,epoint\* RB, epoint\* PB, unsigned char

ZA[],unsigned char ZB[],unsigned char SB[],unsigned char K[],int klen,unsigned char SA[])

{

SM3\_STATE md; int i=0,w=0;

unsigned char Z[SM2\_NUMWORD\*2+SM3\_len/4]={0};

unsigned char x1y1[SM2\_NUMWORD\*2]={0}; unsigned char x2y2[SM2\_NUMWORD\*2]={0};

unsigned char hash[SM2\_NUMWORD],S1[SM2\_NUMWORD]; unsigned char temp[2]={0x02,0x03};

big x1,y1,x1\_,x2,y2,x2\_,tmp,Ux,Uy,temp\_x,temp\_y,tA; epoint\* U;

// mip= mirsys(1000, 16);

// mip->IOBASE=16;

U=epoint\_init(); x1=mirvar(0);

y1=mirvar(0);

x1\_=mirvar(0);

x2=mirvar(0);

y2=mirvar(0);

x2\_=mirvar(0);

tmp=mirvar(0);

Ux=mirvar(0);

Uy=mirvar(0);

temp\_x=mirvar(0); temp\_y=mirvar(0); tA=mirvar(0);

w=SM2\_W(para\_n);

epoint\_get(RA,x1,y1);

big\_to\_bytes(SM2\_NUMWORD,x1,x1y1,TRUE);

big\_to\_bytes(SM2\_NUMWORD,y1,x1y1+SM2\_NUMWORD,TRUE);

//--------A4: x1\_=2^w+x2 & (2^w-1)--------

expb2 (w,x1\_);

divide(x1,x1\_,tmp); add(x1\_,x1,x1\_);

divide(x1\_,para\_n,tmp);

// x1\_=2^w

//x1=x1 mod x1\_ =x1 & (2^w-1)

//-------- A5: tA=(dA+x1\_\*rA)mod n-------- multiply(x1\_,ra,tA);

divide(tA,para\_n,tmp); add(tA,dA,tA);

divide(tA,para\_n,tmp);

//-------- A6:x2\_=2^w+x2 & (2^w-1)-----------------

if(Test\_Point(RB)!=0)

return ERR\_KEYEX\_RB;////////////////////////////////// epoint\_get(RB,x2,y2);

big\_to\_bytes(SM2\_NUMWORD,x2,x2y2,TRUE);

big\_to\_bytes(SM2\_NUMWORD,y2,x2y2+SM2\_NUMWORD,TRUE);

expb2 (w,x2\_); // x2\_=2^w

divide(x2,x2\_,tmp); // x2=x2 mod x2\_=x2 & (2^w-1)

add(x2\_,x2,x2\_);

divide(x2\_,para\_n,tmp);

//--------A7:U=[h\*tA](PB+[x2\_]RB)-----------------

ecurve\_mult(x2\_,RB,U); // U=[x2\_]RB epoint\_get(U,temp\_x,temp\_y);

ecurve\_add(PB,U); // U=PB+U

epoint\_get(U,temp\_x,temp\_y);

multiply(para\_h,tA,tA); // tA=tA\*h

divide(tA,para\_n,tmp);

ecurve\_mult(tA,U,U);

if(point\_at\_infinity(U)==1)

return ERR\_INFINITY\_POINT;

epoint\_get(U,Ux,Uy);

big\_to\_bytes(SM2\_NUMWORD,Ux,Z,1);

big\_to\_bytes(SM2\_NUMWORD,Uy,Z+SM2\_NUMWORD,1);

//------------A8:KA=KDF(UX,UY,ZA,ZB,KLEN)----------

memcpy(Z+SM2\_NUMWORD\*2,ZA,SM3\_len/8);

memcpy(Z+SM2\_NUMWORD\*2+SM3\_len/8,ZB,SM3\_len/8); SM3\_KDF(Z,SM2\_NUMWORD\*2+SM3\_len/4,klen/8,K);

//---------------A9:(optional) S1 =

Hash(0x02||Uy||Hash(Ux||ZA||ZB||x1 |y1 |x2 |y2))-----------

SM3\_init (&md);

SM3\_process(&md,Z,SM2\_NUMWORD);

SM3\_process(&md,ZA,SM3\_len/8);

SM3\_process(&md,ZB,SM3\_len/8);

SM3\_process(&md,x1y1,SM2\_NUMWORD\*2); SM3\_process(&md,x2y2,SM2\_NUMWORD\*2); SM3\_done(&md, hash);

SM3\_init(&md);

SM3\_process(&md,temp,1);

SM3\_process(&md,Z+SM2\_NUMWORD,SM2\_NUMWORD); SM3\_process(&md,hash,SM3\_len/8);

SM3\_done(&md,S1);

//test S1=SB?

if( memcmp(S1,SB,SM2\_NUMWORD)!=0) return ERR\_EQUAL\_S1SB;

//---------------A10 SA = Hash(0x03||yU||Hash(xU| ZA||ZB||x1||y1||x2||y2))-------------

SM3\_init(&md);

SM3\_process(&md,&temp[1],1);

SM3\_process(&md,Z+SM2\_NUMWORD,SM2\_NUMWORD); SM3\_process(&md,hash,SM3\_len/8);

SM3\_done(&md,SA);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_KeyEx\_Re\_II

Description: (optional)Step B10: verifies the hash value received from initiator A

Calls: SM3\_init, SM3\_process, SM3\_done

Called By:

Input: V // calculated in SM2\_KeyEx\_Re\_I

RA // temporary public key received from initiator A

RB // temporary public key received from initiator B

ZA // Z=hash(ELAN\_A||ID of A||a ||b||Gx||Gy||PAx |PAy)

ZB // Z=hash(ELAN\_B||ID of B||a ||b||Gx||Gy||PBx||PBy)

SA // a hash value SA calculated by initiator A,verified in this

function Output:

Return: 0: sucess

9: key validation failed,S2!=SA

Others:

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

int SM2\_KeyEx\_Re\_II(epoint \*V,epoint \*RA,epoint \*RB,unsigned char ZA[],unsigned char ZB[],unsigned char SA[])

{

big x1,y1,x2,y2,Vx,Vy;

unsigned char hash[SM2\_NUMWORD],S2[SM2\_NUMWORD]; unsigned char temp=0x03;

unsigned char xV[SM2\_NUMWORD],yV[SM2\_NUMWORD]; unsigned char x1y1[SM2\_NUMWORD\*2]={0};

unsigned char x2y2[SM2\_NUMWORD\*2]={0}; SM3\_STATE md;

x1=mirvar(0); y1=mirvar(0); x2=mirvar(0); y2=mirvar(0); Vx=mirvar(0); Vy=mirvar(0);

epoint\_get(RA,x1,y1); epoint\_get(RB,x2,y2); epoint\_get(V,Vx,Vy);

big\_to\_bytes(SM2\_NUMWORD,Vx,xV,TRUE);

big\_to\_bytes(SM2\_NUMWORD,Vy,yV,TRUE);

big\_to\_bytes(SM2\_NUMWORD,x1,x1y1,TRUE);

big\_to\_bytes(SM2\_NUMWORD,y1,x1y1+SM2\_NUMWORD,TRUE); big\_to\_bytes(SM2\_NUMWORD,x2,x2y2,TRUE);

big\_to\_bytes(SM2\_NUMWORD,y2,x2y2+SM2\_NUMWORD,TRUE);

//---------------B10:(optional) S2 = Hash(0x03||Vy |Hash(Vx||ZA||ZB||x1||y1| x2| y2))

SM3\_init (&md);

SM3\_process(&md,xV,SM2\_NUMWORD);

SM3\_process(&md,ZA,SM3\_len/8);

SM3\_process(&md,ZB,SM3\_len/8);

SM3\_process(&md,x1y1,SM2\_NUMWORD\*2); SM3\_process(&md,x2y2,SM2\_NUMWORD\*2); SM3\_done(&md, hash);

SM3\_init(&md);

SM3\_process(&md,&temp,1);

SM3\_process(&md,yV,SM2\_NUMWORD); SM3\_process(&md,hash,SM3\_len/8); SM3\_done(&md,S2);

if( memcmp(S2,SA,SM3\_len/8)!=0)

return ERR\_EQUAL\_S2SA;

return 0; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: SM2\_KeyEX\_SelfTest

Description: self check of SM2 key exchange

Calls: SM2\_Init, SM3\_Z, SM2\_KeyEx\_Init\_I, SM2\_KeyEx\_Re\_I, SM2\_KeyEx\_Init\_II,

SM2\_KeyEx\_Re\_II Called By:

Input: Output:

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

6: RA is not valid 7: RB is not valid

8: key validation failed,form B to A,S1!=SB

A: the hash value Z error,Z=hash(ELAN||ID||a ||b||Gx||Gy||Px |Py) B: initialization I failed

C; the shared key KA error,self check failed D; the shared key KB error,self check failed 9: key validation failed,form A to B,S2!=SA

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int SM2\_KeyEX\_SelfTest()

{

//standard data

unsigned char

std\_priKeyA[SM2\_NUMWORD]={0x81,0xEB,0x26,0xE9,0x41,0xBB,0x5A,0xF1,0x6D,0xF1,0x16,0x49,0x5F,0 x90,0x69,0x52,0x72,0xAE,0x2C,0xD6,0x3D,0x6C,0x4A,0xE1,0x67,0x84,0x18,0xBE,0x48,0x23,0x00,0x2 9};

unsigned char

std\_pubKeyA[SM2\_NUMWORD\*2]={0x16,0x0E,0x12,0x89,0x7D,0xF4,0xED,0xB6,0x1D,0xD8,0x12,0xFE,0xB9

,0x67,0x48,0xFB,

0xD3,0xCC,0xF4,0xFF,0xE2,0x6A,0xA6,0xF6,0xDB,0x95,0x40,0xAF,0x49,0xC9,0x42,0x32, 0x4A,0x7D,0xAD,0x08,0xBB,0x9A,0x45,0x95,0x31,0x69,0x4B,0xEB,0x20,0xAA,0x48,0x9D, 0x66,0x49,0x97,0x5E,0x1B,0xFC,0xF8,0xC4,0x74,0x1B,0x78,0xB4,0xB2,0x23,0x00,0x7F};

unsigned char std\_randA[SM2\_NUMWORD]=

{0xD4,0xDE,0x15,0x47,0x4D,0xB7,0x4D,0x06,0x49,0x1C,0x44,0x0D,0x30,0x5E,0x01,0x24, 0x00,0x99,0x0F,0x3E,0x39,0x0C,0x7E,0x87,0x15,0x3C,0x12,0xDB,0x2E,0xA6,0x0B,0xB3};

unsigned char

std\_priKeyB[SM2\_NUMWORD]={0x78,0x51,0x29,0x91,0x7D,0x45,0xA9,0xEA,0x54,0x37,0xA5,0x93,0x56,0 xB8,0x23,0x38,

0xEA,0xAD,0xDA,0x6C,0xEB,0x19,0x90,0x88,0xF1,0x4A,0xE1,0x0D,0xEF,0xA2,0x29,0xB5}; unsigned char

std\_pubKeyB[SM2\_NUMWORD\*2]={0x6A,0xE8,0x48,0xC5,0x7C,0x53,0xC7,0xB1,0xB5,0xFA,0x99,0xEB,0x22

,0x86,0xAF,0x07,

0x8B,0xA6,0x4C,0x64,0x59,0x1B,0x8B,0x56,0x6F,0x73,0x57,0xD5,0x76,0xF1,0x6D,0xFB, 0xEE,0x48,0x9D,0x77,0x16,0x21,0xA2,0x7B,0x36,0xC5,0xC7,0x99,0x20,0x62,0xE9,0xCD, 0x09,0xA9,0x26,0x43,0x86,0xF3,0xFB,0xEA,0x54,0xDF,0xF6,0x93,0x05,0x62,0x1C,0x4D};

unsigned char std\_randB[SM2\_NUMWORD]=

{0x7E,0x07,0x12,0x48,0x14,0xB3,0x09,0x48,0x91,0x25,0xEA,0xED,0x10,0x11,0x13,0x16, 0x4E,0xBF,0x0F,0x34,0x58,0xC5,0xBD,0x88,0x33,0x5C,0x1F,0x9D,0x59,0x62,0x43,0xD6};

unsigned char

std\_IDA[16]={0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38 };

unsigned char

std\_IDB[16]={0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38 };

unsigned short int std\_ENTLA=0x0080; unsigned short int std\_ENTLB=0x0080; unsigned char

std\_ZA[SM3\_len]={0x3B,0x85,0xA5,0x71,0x79,0xE1,0x1E,0x7E,0x51,0x3A,0xA6,0x22,0x99,0x1F,0x2C, 0xA7,0x4D,0x18,0x07,0xA0,0xBD,0x4D,0x4B,0x38,0xF9,0x09,0x87,0xA1,0x7A,0xC2,0x45,0xB1};

unsigned char

std\_ZB[SM3\_len]={0x79,0xC9,0x88,0xD6,0x32,0x29,0xD9,0x7E,0xF1,0x9F,0xE0,0x2C,0xA1,0x05,0x6E, 0x01,0xE6,0xA7,0x41,0x1E,0xD2,0x46,0x94,0xAA,0x8F,0x83,0x4F,0x4A,0x4A,0xB0,0x22,0xF7};

unsigned char

std\_RA[SM2\_NUMWORD\*2]={0x64,0xCE,0xD1,0xBD,0xBC,0x99,0xD5,0x90,0x04,0x9B,0x43,0x4D,0x0F,0xD7 ,0x34,0x28,0xCF,0x60,0x8A,0x5D,0xB8,0xFE,0x5C,0xE0,0x7F,0x15,0x02,0x69,0x40,0xBA,0xE4,0x0E,

0x37,0x66,0x29,0xC7,0xAB,0x21,0xE7,0xDB,0x26,0x09,0x22,0x49,0x9D,0xDB,0x11,0x8F,0x07,0xCE,0x 8E,0xAA,0xE3,0xE7,0x72,0x0A,0xFE,0xF6,0xA5,0xCC,0x06,0x20,0x70,0xC0};

unsigned char

std\_K[16]={0x6C,0x89,0x34,0x73,0x54,0xDE,0x24,0x84,0xC6,0x0B,0x4A,0xB1,0xFD,0xE4,0xC6,0xE5};

unsigned char std\_RB[SM2\_NUMWORD\*2]=

{0xAC,0xC2,0x76,0x88,0xA6,0xF7,0xB7,0x06,0x09,0x8B,0xC9,0x1F,0xF3,0xAD,0x1B,0xFF, 0x7D,0xC2,0x80,0x2C,0xDB,0x14,0xCC,0xCC,0xDB,0x0A,0x90,0x47,0x1F,0x9B,0xD7,0x07, 0x2F,0xED,0xAC,0x04,0x94,0xB2,0xFF,0xC4,0xD6,0x85,0x38,0x76,0xC7,0x9B,0x8F,0x30, 0x1C,0x65,0x73,0xAD,0x0A,0xA5,0x0F,0x39,0xFC,0x87,0x18,0x1E,0x1A,0x1B,0x46,0xFE};

unsigned char

std\_SB[SM3\_len]={0xD3,0xA0,0xFE,0x15,0xDE,0xE1,0x85,0xCE,0xAE,0x90,0x7A,0x6B,0x59,0x5C,0xC3, 0x2A,0x26,0x6E,0xD7,0xB3,0x36,0x7E,0x99,0x83,0xA8,0x96,0xDC,0x32,0xFA,0x20,0xF8,0xEB};

int std\_Klen=128;//bit len int temp;

big x,y,dA,dB,rA,rB;

epoint\* pubKeyA,\*pubKeyB,\*RA,\*RB,\*V;

unsigned unsigned unsigned unsigned unsigned unsigned

char char char char char char

hash[SM3\_len/8]={0};

ZA[SM3\_len/8]={0};

ZB[SM3\_len/8]={0};

xy[SM2\_NUMWORD\*2]={0}; \*KA,\*KB;

SA[SM3\_len/8];

KA=malloc(std\_Klen/8); KB=malloc(std\_Klen/8);

mip= mirsys(1000, 16); mip->IOBASE=16;

x=mirvar(0); y=mirvar(0); dA=mirvar(0); dB=mirvar(0); rA=mirvar(0); rB=mirvar(0);

pubKeyA=epoint\_init(); pubKeyB=epoint\_init(); RA=epoint\_init();

RB=epoint\_init(); V=epoint\_init();

SM2\_Init();

bytes\_to\_big(SM2\_NUMWORD,std\_priKeyA,dA); bytes\_to\_big(SM2\_NUMWORD,std\_priKeyB,dB); bytes\_to\_big(SM2\_NUMWORD,std\_randA,rA);

bytes\_to\_big(SM2\_NUMWORD,std\_randB,rB); bytes\_to\_big(SM2\_NUMWORD,std\_pubKeyA,x);

bytes\_to\_big(SM2\_NUMWORD,std\_pubKeyA+SM2\_NUMWORD,y); epoint\_set(x,y,0,pubKeyA);

bytes\_to\_big(SM2\_NUMWORD,std\_pubKeyB,x);

bytes\_to\_big(SM2\_NUMWORD,std\_pubKeyB+SM2\_NUMWORD,y); epoint\_set(x,y,0,pubKeyB);

SM3\_Z(std\_IDA, std\_ENTLA, pubKeyA, ZA); if(memcmp(ZA,std\_ZA,SM3\_len/8)!=0)

return ERR\_SELFTEST\_Z;

SM3\_Z(std\_IDB, std\_ENTLB, pubKeyB, ZB); if(memcmp(ZB,std\_ZB,SM3\_len/8)!=0)

return ERR\_SELFTEST\_Z;

temp=SM2\_KeyEx\_Init\_I(rA,RA); if(temp) return temp;

epoint\_get(RA,x,y);

big\_to\_bytes(SM2\_NUMWORD,x,xy,1);

big\_to\_bytes(SM2\_NUMWORD,y,xy+SM2\_NUMWORD,1);

if(memcmp(xy,std\_RA,SM2\_NUMWORD\*2)!=0) return ERR\_SELFTEST\_INI\_I;

temp=SM2\_KeyEx\_Re\_I(rB, dB, RA, pubKeyA, ZA,ZB, KA,std\_Klen, RB,V, hash); if(temp) return temp;

if(memcmp(KA, std\_K, std\_Klen/8)!=0) return ERR\_SELFTEST\_RES\_I;

temp=SM2\_KeyEx\_Init\_II(rA, dA,RA, RB, pubKeyB, ZA,ZB,hash, KB,std\_Klen,SA); if(temp) return temp;

if(memcmp(KB, std\_K, std\_Klen/8)!=0) return ERR\_SELFTEST\_INI\_II;

if(SM2\_KeyEx\_Re\_II(V,RA,RB,ZA,ZB,SA)!=0) return ERR\_EQUAL\_S2SA;

free(KA);free(KB);

return 0; }