```
/************************************************************************
  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);            //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 1;
    else
        return 0;
}


/***************************************************************
  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              // bit len of ID
                  pubKey            // 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            // a big number lies in[1,n-1]
                  dB            // private key of responder B
                  RA            // temporary public key received from initiator A
                  PA            // public key of initiator A
                  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)
```

```
                  klen          // byte len of the secret key that A and B wanna share
  Output:         K             // secret key that A and B wanna share
                  RB            // RB=[rb]G
                  V             // V=[h*tB](PA+[x1_]RA),in function SM2_KeyEx_Re_II it as input
                  hash          // (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         // a big number lies in[1,n-1]
                   dA         // private key of initiator A
                   RA         // temporary public key received from initiator A
                   RB         // temporary public key received from initiator B
                   PB         // public key of initiator A
                   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)
                   klen       // byte len of the secret key that A and B wanna share
                   SB         // 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_);            //  x1_=2^w
divide(x1,x1_,tmp);        //x1=x1 mod x1_ =x1 & (2^w-1)
add(x1_,x1,x1_);
divide(x1_,para_n,tmp);


//-------- 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 char hash[SM3_len/8]={0};
    unsigned char ZA[SM3_len/8]={0};
    unsigned char ZB[SM3_len/8]={0};
    unsigned char xy[SM2_NUMWORD*2]={0};
    unsigned char *KA,*KB;
    unsigned char 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;
}
```