```
///************************************************************************
//  File name:    SM9_Key_ex.c
//  Version:      SM9_Key_ex_V1.0
//  Date:         Jan 1,2017
//  Description:  implementation of SM9 Key Exchange Protocol
//                all operations based on BN curve line function
//  Function List:
//        1.bytes128_to_ecn2        //convert char into ecn2
//        2.zzn12_ElementPrint      //print all element of struct zzn12
//        3.LinkCharZzn12           //link two different types(unsigned char and zzn12)to
one(unsigned char)
//        4.Test_Point              //test if the given point is on SM9 curve
//        5.SM9_KeyEx_KDF           //calculate KDF(IDA||IDB||RA||RB||g1||g2||g3)
//        6.SM9_KeyEx_Hash          //calculate
Hash(hashid||g1||Hash(g2||g3||IDA||IDB||RA||RB))
//        7.SM9_H1                  //function H1 in SM9 standard 5.4.2.2
//        8.SM9_Init                //initiate SM9 curve
//        9.SM9_GenerateEncryptKey  //generate encrypted private and public key
//        10.SM9_KeyEx_InitA_I      //calculate RA (Step A1-A4)
//        11.SM9_KeyEx_ReB_I        //calculate RB ,a hash Value SB and a shared key SKB(Step
B1-A7)
//        12.SM9_KeyEx_InitA_II     //initiator A calculate the secret key SKA and a hash
                                    //SA which responder B might verifies(Step A5-A7)
//        13.SM9_KeyEx_ReB_II       //Step B10 (optional) verifies the hash value SA received
from initiator A
//        14.SM9_SelfCheck()        //SM9 slef-check

//
// Notes:
//   This SM9 implementation source code can be used for academic, non-profit making or
non-commercial use only.
//   This SM9 implementation is created on MIRACL. SM9 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 "SM9_Key_ex.h"
#include "kdf.h"


/****************************************************************
  Function:       bytes128_to_ecn2
  Description:    convert 128 bytes into ecn2
  Calls:          MIRACL functions
  Called By:      SM9_Init,SM9_KeyEx_ReB_I, SM9_KeyEx_InitA_II
  Input:          Ppubs[]
  Output:         ecn2 *res
  Return:         FALSE: execution error
                  TRUE: execute correctly
  Others:
****************************************************************/
BOOL bytes128_to_ecn2(unsigned char Ppubs[],ecn2 *res)
{
    zzn2 x, y;
     big a,b;
    ecn2 r;
    r.x.a=mirvar(0);r.x.b=mirvar(0);
    r.y.a=mirvar(0);r.y.b=mirvar(0);
    r.z.a=mirvar(0);r.z.b=mirvar(0);
    r.marker=MR_EPOINT_INFINITY;

    x.a=mirvar(0);x.b=mirvar(0);
    y.a=mirvar(0);y.b=mirvar(0);
    a=mirvar(0);b=mirvar(0);

    bytes_to_big(BNLEN,Ppubs,b);
    bytes_to_big(BNLEN,Ppubs+BNLEN,a);
    zzn2_from_bigs(a,b,&x);
    bytes_to_big(BNLEN,Ppubs+BNLEN*2,b);
    bytes_to_big(BNLEN,Ppubs+BNLEN*3,a);
    zzn2_from_bigs(a,b,&y);

     return ecn2_set( &x,&y,res);
}
/****************************************************************
  Function:       zzn12_ElementPrint
  Description:    print all element of struct zzn12
  Calls:          MIRACL functions
  Called By:      SM9_KeyEx_ReB_I,SM9_KeyEx_InitA_II
  Input:          zzn12 x
  Output:         NULL
  Return:         NULL
  Others:
****************************************************************/
void zzn12_ElementPrint(zzn12 x)
{
    big tmp;
    tmp=mirvar(0);

    redc(x.c.b.b,tmp);cotnum(tmp,stdout);
    redc(x.c.b.a,tmp);cotnum(tmp,stdout);
    redc(x.c.a.b,tmp);cotnum(tmp,stdout);
    redc(x.c.a.a,tmp);cotnum(tmp,stdout);
    redc(x.b.b.b,tmp);cotnum(tmp,stdout);
    redc(x.b.b.a,tmp);cotnum(tmp,stdout);
    redc(x.b.a.b,tmp);cotnum(tmp,stdout);
    redc(x.b.a.a,tmp);cotnum(tmp,stdout);
    redc(x.a.b.b,tmp);cotnum(tmp,stdout);
    redc(x.a.b.a,tmp);cotnum(tmp,stdout);
    redc(x.a.a.b,tmp);cotnum(tmp,stdout);
    redc(x.a.a.a,tmp);cotnum(tmp,stdout);
}


/****************************************************************
  Function:       LinkCharZzn12
  Description:    link two different types(unsigned char and zzn12)to one(unsigned char)
  Calls:          MIRACL functions
  Called By:      SM9_KeyEx_KDF,SM9_KeyEx_Hash
  Input:          message:
                  len:    length of message
                  w:      zzn12 element
  Output:         Z:      the characters array stored message and w
                  Zlen:   length of Z
  Return:         NULL
  Others:
****************************************************************/
void LinkCharZzn12(unsigned char *message,int len,zzn12 w,unsigned char *Z,int Zlen)
     {
       big tmp;

       tmp=mirvar(0);

       memcpy(Z,message,len);
       redc(w.c.b.b,tmp);big_to_bytes(BNLEN,tmp,Z+len,1);
       redc(w.c.b.a,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN,1);
       redc(w.c.a.b,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*2,1);
       redc(w.c.a.a,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*3,1);
       redc(w.b.b.b,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*4,1);
       redc(w.b.b.a,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*5,1);
       redc(w.b.a.b,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*6,1);
       redc(w.b.a.a,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*7,1);
       redc(w.a.b.b,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*8,1);
       redc(w.a.b.a,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*9,1);
       redc(w.a.a.b,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*10,1);
       redc(w.a.a.a,tmp);big_to_bytes(BNLEN,tmp,Z+len+BNLEN*11,1);
     }

/***************************************************************
  Function:       Test_Point
  Description:    test if the given point is on SM9 curve
  Calls:          MIRACL functions
  Called By:      SM9_KeyEx_ReB_I,SM9_KeyEx_InitA_II
  Input:          point
  Output:         null
  Return:         0: success
                  1: not a valid point on curve

  Others:
***************************************************************/
int Test_Point(epoint* point)
{
    big x,y,x_3,tmp;
    epoint *buf;

    x=mirvar(0); y=mirvar(0);
    x_3=mirvar(0);
    tmp=mirvar(0);
    buf=epoint_init();

    //test if y^2=x^3+b
    epoint_get(point,x,y);
    power (x, 3, para_q, x_3);      //x_3=x^3 mod p
    multiply (x, para_a,x);
    divide (x, para_q, tmp);
    add(x_3,x,x);                   //x=x^3+ax+b
    add(x,para_b,x);
    divide(x,para_q,tmp);           //x=x^3+ax+b mod p
    power (y, 2,para_q, y);         //y=y^2 mod p
    if(mr_compare(x,y)!=0)
        return 1;

    //test infinity
    ecurve_mult(N,point,buf);
    if(point_at_infinity(buf)==FALSE)
```

```
        return 1;

    return 0;
}


/****************************************************************
  Function:       SM9_KeyEx_KDF
  Description:    calculate KDF(IDA||IDB||RA||RB||g1||g2||g3)
  Calls:          MIRACL functions,LinkCharZzn12,SM3_KDF
  Called By:      SM9_KeyEx_ReB_I,SM9_KeyEx_InitA_II
  Input:          IDA,IDB:   //identification of user A and B
                  RA,RB      //element of group G1
                  g1,g2,g3   //R-ate pairing
                  klen       //bytelen of K
  Output:         K          //shared secret key
  Return:         0: success
                  1: asking for memory error
  Others:
****************************************************************/
int SM9_KeyEx_KDF(unsigned char *IDA,unsigned char *IDB,epoint *RA,epoint *RB,zzn12 g1,zzn12
g2,zzn12 g3,int klen,unsigned char K[])
{
    unsigned char *Z=NULL;
    int Zlen;
    int IDALen=strlen(IDA),IDBLen=strlen(IDB);
    big x1,y1,x2,y2;

    x1=mirvar(0);y1=mirvar(0);
    x2=mirvar(0);y2=mirvar(0);
    epoint_get(RA,x1,y1);
    epoint_get(RB,x2,y2);

    Zlen=IDALen+IDBLen+BNLEN*40;
    Z=(char *)malloc(sizeof(char)*(Zlen+1));
    if(Z==NULL) return SM9_ASK_MEMORY_ERR;

    memcpy(Z,IDA,IDALen);
    memcpy(Z+IDALen,IDB,IDBLen);
    big_to_bytes(BNLEN,x1,Z+IDALen+IDBLen,1);
    big_to_bytes(BNLEN,y1,Z+IDALen+IDBLen+BNLEN,1);
    big_to_bytes(BNLEN,x2,Z+IDALen+IDBLen+BNLEN*2,1);
    big_to_bytes(BNLEN,y2,Z+IDALen+IDBLen+BNLEN*3,1);
    LinkCharZzn12(Z,0,g1,Z+IDALen+IDBLen+BNLEN*4,BNLEN*12);
    LinkCharZzn12(Z,0,g2,Z+IDALen+IDBLen+BNLEN*16,BNLEN*12);
    LinkCharZzn12(Z,0,g3,Z+IDALen+IDBLen+BNLEN*28,BNLEN*12);

    SM3_KDF(Z,Zlen,klen,K);
    free(Z);
    return 0;
 }


/****************************************************************
  Function:       SM9_KeyEx_Hash
  Description:    calculate Hash(hashid||g1||Hash(g2||g3||IDA||IDB||RA||RB))
  Calls:          MIRACL functions,LinkCharZzn12,SM3_256
  Called By:      SM9_KeyEx_ReB_I,SM9_KeyEx_InitA_II
  Input:
                  hashid     //0x82,0x83
                  IDA,IDB:   //identification of user A and B
                  RA,RB      //element of group G1
                  g1,g2,g3   //R-ate pairing
  Output:
                  hash       //hash=Hash(hashid||g1||Hash(g2||g3||IDA||IDB||RA||RB))
  Return:         0: success
                  1: asking for memory error
  Others:
****************************************************************/
int SM9_KeyEx_Hash(unsigned char hashid[],unsigned char *IDA,unsigned char *IDB,epoint
*RA,epoint *RB,zzn12 g1,zzn12 g2,zzn12 g3,unsigned char hash[])
{
    int Zlen;
    int IDALen=strlen(IDA),IDBLen=strlen(IDB);
    unsigned char *Z=NULL;
    big x1,y1,x2,y2;

    x1=mirvar(0);y1=mirvar(0);
    x2=mirvar(0);y2=mirvar(0);
    epoint_get(RA,x1,y1); epoint_get(RB,x2,y2);

    Zlen=IDALen+IDBLen+BNLEN*28;
    Z=(char *)malloc(sizeof(char)*(Zlen+1));
    if(Z==NULL) return SM9_ASK_MEMORY_ERR;

    LinkCharZzn12(Z,0,g2,Z,BNLEN*12);
    LinkCharZzn12(Z,0,g3,Z+BNLEN*12,BNLEN*12);
    memcpy(Z+BNLEN*24,IDA,IDALen);
    memcpy(Z+BNLEN*24+IDALen,IDB,IDBLen);
    big_to_bytes(BNLEN,x1,Z+BNLEN*24+IDALen+IDBLen,1);
    big_to_bytes(BNLEN,y1,Z+BNLEN*25+IDALen+IDBLen,1);
    big_to_bytes(BNLEN,x2,Z+BNLEN*26+IDALen+IDBLen,1);
    big_to_bytes(BNLEN,y2,Z+BNLEN*27+IDALen+IDBLen,1);

    SM3_256(Z,Zlen,hash);

    Zlen=1+BNLEN*12+SM3_len/8;
    memcpy(Z,hashid,1);
    LinkCharZzn12(Z,1,g1,Z,1+BNLEN*12);
    memcpy(Z+1+BNLEN*12,hash,SM3_len/8);

    SM3_256(Z,Zlen,hash);
    free(Z);
    return 0;
}
/****************************************************************
  Function:       SM9_H1
  Description:    function H1 in SM9 standard 5.4.2.2
  Calls:          MIRACL functions,SM3_KDF
  Called By:      SM9_GenerateEncryptKey,SM9_KeyEx_InitA_I
  Input:          Z:
                  Zlen:the length of Z
                  n:Frobniues constant X
  Output:         h1=H1(Z,Zlen)
  Return:         0: success;
                  1: asking for memory error
  Others:
****************************************************************/
int SM9_H1(unsigned char Z[],int Zlen,big n,big h1)
{
     int hlen,i,ZHlen;
     big hh,i256,tmp,n1;
     unsigned char *ZH=NULL,*ha=NULL;

     hh=mirvar(0);i256=mirvar(0);
     tmp=mirvar(0);n1=mirvar(0);
     convert(1,i256);
     ZHlen=Zlen+1;

     hlen=(int)ceil((5.0*logb2(n))/32.0);
     decr(n,1,n1);
     ZH=(char *)malloc(sizeof(char)*(ZHlen+1));
     if(ZH==NULL) return SM9_ASK_MEMORY_ERR;
     memcpy(ZH+1,Z,Zlen);
     ZH[0]=0x01;
     ha=(char *)malloc(sizeof(char)*(hlen+1));
     if(ha==NULL) return SM9_ASK_MEMORY_ERR;
     SM3_KDF(ZH,ZHlen,hlen,ha);

     for(i=hlen-1;i>=0;i--)//key[从大到小]
     {
          premult(i256,ha[i],tmp);
        add(hh,tmp,hh);
        premult(i256,256,i256);
        divide(i256,n1,tmp);
        divide(hh,n1,tmp);
     }
    incr(hh,1,h1);
    free(ZH);free(ha);
    return 0;
}



/*****************************************************************
  Function:       SM9_Init
  Description:    Initiate SM9 curve
  Calls:          MIRACL functions
  Called By:      SM9_SelfCheck
  Input:          null
  Output:         null
  Return:         0: success;
                  5: base point P1 error
                  6: base point P2 error
  Others:
*****************************************************************/
int SM9_Init()
{
    big P1_x, P1_y;

    para_q=mirvar(0);N=mirvar(0);
    P1_x=mirvar(0);  P1_y=mirvar(0);
    para_a=mirvar(0);
    para_b=mirvar(0);para_t=mirvar(0);
    X.a=mirvar(0);  X.b=mirvar(0);
    P2.x.a=mirvar(0);P2.x.b=mirvar(0);
    P2.y.a=mirvar(0);P2.y.b=mirvar(0);
    P2.z.a=mirvar(0);P2.z.b=mirvar(0);
    P2.marker=MR_EPOINT_INFINITY;

     P1=epoint_init();
    bytes_to_big(BNLEN,SM9_q,para_q);
    bytes_to_big(BNLEN,SM9_P1x,P1_x);
    bytes_to_big(BNLEN,SM9_P1y,P1_y);
    bytes_to_big(BNLEN,SM9_a,para_a);
     bytes_to_big(BNLEN,SM9_b,para_b);
    bytes_to_big(BNLEN,SM9_N,N);
    bytes_to_big(BNLEN,SM9_t,para_t);

    mip->TWIST=MR_SEXTIC_M;
    ecurve_init(para_a,para_b,para_q,MR_PROJECTIVE);  //Initialises GF(q) elliptic curve
                                       //MR_PROJECTIVE specifying  projective coordinates

    if(!epoint_set(P1_x,P1_y,0,P1)) return SM9_G1BASEPOINT_SET_ERR;



    if(!(bytes128_to_ecn2(SM9_P2,&P2)))   return SM9_G2BASEPOINT_SET_ERR;
     set_frobenius_constant(&X);

    return 0;
}

/****************************************************************
  Function:       SM9_GenerateEncryptKey
  Description:    Generate encryption keys(public key and private key)
  Calls:          MIRACL functions,SM9_H1,xgcd
  Called By:      SM9_SelfCheck
  Input:          hid:0x02
                  ID:identification
                  IDlen:the length of ID
                  ke:master private key used to generate encryption public key and private key
  Output:         Ppubs:encryption public key
                  deB: encryption private key
  Return:         0: success;
                  1: asking for memory error
  Others:
****************************************************************/
int SM9_GenerateEncryptKey(unsigned char hid[],unsigned char *ID,int IDlen,big ke,unsigned char
Ppubs[],unsigned char deB[])
{
   big h1,t1,t2,rem,xPpub,yPpub,tmp;
   unsigned char *Z=NULL;
   int Zlen=IDlen+1,buf;
   ecn2 dEB;
   epoint *Ppub;

   h1=mirvar(0);t1=mirvar(0);
   t2=mirvar(0);rem=mirvar(0);tmp=mirvar(0);
   xPpub=mirvar(0);yPpub=mirvar(0);
   Ppub=epoint_init();
   dEB.x.a=mirvar(0);dEB.x.b=mirvar(0);dEB.y.a=mirvar(0);dEB.y.b=mirvar(0);
   dEB.z.a=mirvar(0);dEB.z.b=mirvar(0);dEB.marker=MR_EPOINT_INFINITY;

   Z=(char *)malloc(sizeof(char)*(Zlen+1));
   memcpy(Z,ID,IDlen);
   memcpy(Z+IDlen,hid,1);

   buf=SM9_H1(Z,Zlen,N,h1);
   if(buf!=0)     return buf;
   add(h1,ke,t1);//t1=H1(IDA||hid,N)+ks
   xgcd(t1,N,t1,t1,t1);//t1=t1(-1)
   multiply(ke,t1,t2); divide(t2,N,rem);//t2=ks*t1(-1)

   //Ppub=[ke]P2
   ecurve_mult(ke,P1,Ppub);

   //deB=[t2]P2
   ecn2_copy(&P2,&dEB);
   ecn2_mul(t2,&dEB);

   epoint_get(Ppub,xPpub,yPpub);
   big_to_bytes(BNLEN,xPpub,Ppubs,1);
   big_to_bytes(BNLEN,yPpub,Ppubs+BNLEN,1);

   redc(dEB.x.b,tmp);big_to_bytes(BNLEN,tmp,deB,1);
   redc(dEB.x.a,tmp);big_to_bytes(BNLEN,tmp,deB+BNLEN,1);
   redc(dEB.y.b,tmp);big_to_bytes(BNLEN,tmp,deB+BNLEN*2,1);
   redc(dEB.y.a,tmp);big_to_bytes(BNLEN,tmp,deB+BNLEN*3,1);

   free(Z);
   return 0;
}
```

```
/****************************************************************
  Function:       SM9_KeyEx_InitA_I
  Description:    calculate RA (Step A1-A4)
  Calls:          MIRACL functions,SM9_H1
  Called By:      SM9_SelfCheck()
  Input:
                  hid:0x02
                  IDB          //identification of userB
                  randA        //a random number K lies in [1,N-1]
                  Ppubs        //encryption public key
                  deA          //decryption private key of user A
  Output:
                  RA           //RA=[rA]QB
  Return:
                  0: success
                  1: asking for memory error
  Others:
****************************************************************/
int SM9_KeyEx_InitA_I(unsigned char hid[],unsigned char *IDB,unsigned char randA[],
                      unsigned char Ppub[],unsigned char deA[],epoint *RA)
{
     big h,x,y,rA;
     epoint *Ppube,*QB;
     unsigned char *Z=NULL;
     int Zlen,buf;

     //initiate
     h=mirvar(0);rA=mirvar(0);x=mirvar(0);y=mirvar(0);
     QB=epoint_init();Ppube=epoint_init();

     bytes_to_big(BNLEN,Ppub,x);
     bytes_to_big(BNLEN,Ppub+BNLEN,y);
     epoint_set(x,y,0,Ppube);

     //----------A1:calculate QB=[H1(IDB||hid,N)]P1+Ppube----------
     Zlen=strlen(IDB)+1;
     Z=(char *)malloc(sizeof(char)*(Zlen+1));
     if(Z==NULL) return SM9_ASK_MEMORY_ERR;
     memcpy(Z,IDB,strlen(IDB));
     memcpy(Z+strlen(IDB),hid,1);
     buf=SM9_H1(Z,Zlen,N,h);
     if(buf) return buf;
     ecurve_mult(h,P1,QB);
     ecurve_add(Ppube,QB);
```

```
    printf("\n******************QB:=[H1(IDB||hid,N)]P1+Ppube****************\n");
    epoint_get(QB,x,y);
    cotnum(x,stdout);cotnum(y,stdout);

    //--------------- Step A2:randnom -------------------
    bytes_to_big(BNLEN,randA,rA);
    printf("\n*********************randnum rA:****************************\n");
    cotnum(rA,stdout);

    //----------------Step A3:RA=[r]QB
    ecurve_mult(rA,QB,RA);

    free(Z);
    return 0;
}

/****************************************************************
  Function:       SM9_KeyEx_ReB_I
  Description:    calculate RB ,a hash Value SB and a shared key SKB(Step B1-A7)
  Calls:          MIRACL functions,SM9_H1,Test_Point,ecap(),member(),
                  zzn12_pow,zzn12_ElementPrint(),SM9_KeyEx_Hash
  Called By:      SM9_SelfCheck()
  Input:
                  hid:0x02
                  IDA,IDB      //identification of userA and B
                  randB        //a random number K lies in [1,N-1]
                  Ppub         //encryption public key
                  deB          //decryption private key of user B
                  RA           //temporary value received from initiator A
  Output:
                  RB           //RB=[rB]QA
                  SB           //(option) calculates a hash value SB that initiator A might
verifies
                  g1,g2,g3     //R-ate pairings used to calculate S2 in function
SM9_KeyEx_ReB_II
  Return:
                  0: success
                  1: asking for memory error
                  2: element is out of order q
                  3: R-ate calculation error
                  4: RA is not valid
  Others:
****************************************************************/
```

```
int SM9_KeyEx_ReB_I(unsigned char hid[],unsigned char *IDA,unsigned char *IDB,unsigned char
randB[],unsigned char Ppub[],
                    unsigned char deB[],epoint *RA,epoint *RB,unsigned char SB[],zzn12
*g1,zzn12 *g2,zzn12 *g3)
{
     big h,x,y,rB;
     epoint *Ppube,*QA;
     unsigned char *Z=NULL,hashid[]={0x82};
     unsigned char SKB[16];
     ecn2 dEB;
     int Zlen,buf,i;

     //initiate
     h=mirvar(0);rB=mirvar(0);x=mirvar(0);y=mirvar(0);
     QA=epoint_init();Ppube=epoint_init();
     dEB.x.a=mirvar(0); dEB.x.b=mirvar(0);dEB.y.a=mirvar(0);dEB.y.b=mirvar(0);
     dEB.z.a=mirvar(0); dEB.z.b=mirvar(0);dEB.marker=MR_EPOINT_INFINITY;

     bytes_to_big(BNLEN,Ppub,x);bytes_to_big(BNLEN,Ppub+BNLEN,y);
     bytes128_to_ecn2(deB,&dEB);
     epoint_set(x,y,0,Ppube);

     //----------B1:calculate QA=[H1(IDA||hid,N)]P1+Ppube----------
     Zlen=strlen(IDA)+1;
     Z=(char *)malloc(sizeof(char)*(Zlen+1));
     if(Z==NULL) return SM9_ASK_MEMORY_ERR;
     memcpy(Z,IDA,strlen(IDA));
     memcpy(Z+strlen(IDA),hid,1);

     buf=SM9_H1(Z,Zlen,N,h);
     if(buf) return buf;
     ecurve_mult(h,P1,QA);
     ecurve_add(Ppube,QA);
     printf("\n********************QA:=[H1(IDA||hid,N)]P1+Ppube*****************\n");
     epoint_get(QA,x,y);
     cotnum(x,stdout);cotnum(y,stdout);

     //--------------- Step B2:randnom -------------------
     bytes_to_big(BNLEN,randB,rB);
     printf("\n************************randnum rB:*********************************\n");
     cotnum(rB,stdout);

     //----------------Step B3:RB=[rB]QA------------------
     ecurve_mult(rB,QA,RB);
```

```
    printf("\n************************:RB=[rB]QA*****************************\n");
    epoint_get(RB,x,y);
    cotnum(x,stdout);cotnum(y,stdout);

    //test if RA is on G1
    if(Test_Point(RA)) return SM9_NOT_VALID_G1;

    //----------------Step B4:g1=e(deB,RA),g2=(e(P2,Ppube))^rB,g3=g1^rB
     if(!ecap(dEB, RA, para_t, X, g1)) return SM9_MY_ECAP_12A_ERR;
    if(!ecap(P2, Ppube, para_t, X, g2)) return SM9_MY_ECAP_12A_ERR;
    //test if a ZZn12 element is of order q
    if((!member(*g1, para_t, X))||(!member(*g2, para_t, X))) return SM9_MEMBER_ERR;

    *g2=zzn12_pow(*g2,rB);
    *g3=zzn12_pow(*g1,rB);

    printf("\n**********************g1=e(RA,deB):***************************\n");
    zzn12_ElementPrint(*g1);
    printf("\n******************g2=(e(P2,Ppub3))^rB:************************\n");
    zzn12_ElementPrint(*g2);
    printf("\n**********************g3=g1^rB:******************************\n");
    zzn12_ElementPrint(*g3);

    //---------------- B5:SKB=KDF(IDA||IDB||RA||RB||g1||g2||g3,klen)----------
    buf=SM9_KeyEx_KDF(IDA,IDB,RA,RB,*g1,*g2,*g3,16,SKB);
    if(buf) return buf;
    printf("\n***********SKB=KDF(IDA||IDB||RA||RB||g1||g2||g3,klen):***********\n");
    for(i=0;i<16;i++) printf("%02x",SKB[i]);

    //----------------
B6(optional):SB=Hash(0x82||g1||Hash(g2||g3||IDA||IDB||RA||RB))----------
    buf=SM9_KeyEx_Hash(hashid,IDA,IDB,RA,RB,*g1,*g2,*g3,SB);
    if(buf) return buf;
    printf("\n********SB=Hash(0x82||g1||Hash(g2||g3||IDA||IDB||RA||RB))********\n");
    for(i=0;i<SM3_len/8;i++) printf("%02x",SB[i]);

    free(Z);
    return 0;
}

/***************************************************************
  Function:       SM9_KeyEx_InitA_II
  Description:    initiator A calculate the secret key SKA and a hash
                   SA which responder B might verifies(Step A5-A7)
  Calls:          MIRACL functions,SM9_H1,Test_Point,ecap(),member(),zzn12_init
                  zzn12_pow,zzn12_ElementPrint(),SM9_KeyEx_KDF,SM9_KeyEx_Hash
  Called By:      SM9_SelfCheck()
  Input:
                  IDA,IDB      //identification of userA and B
                  randA        //a random number K lies in [1,N-1]
                  Ppub         //encryption public key
                  deA          //decryption private key of initiator A
                  RA,RB        //temporary value received from initiator A and responder B
                  SB           //a hash value SB calculated by responder B,verified in this
function
  Output:
                  SA:          //(option) calculates a hash value SA that responder B might
verifies
  Return:
                  0: success
                  1: asking for memory error
                  2: element is out of order q
                  3: R-ate calculation error
                  4: RA is not valid
                  9: key exchange failed,form B to A,S1!=SB
  Others:
****************************************************************/
int SM9_KeyEx_InitA_II(unsigned char *IDA,unsigned char *IDB,unsigned char randA[],unsigned char
Ppub[],
                       unsigned char deA[],epoint *RA,epoint *RB,unsigned char SB[],unsigned
char SA[])
{
     big h,x,y,rA;
     epoint *Ppube;
     unsigned char hashid[]={0x82};
     unsigned char S1[SM3_len/8],SKA[16];
     zzn12 g1,g2,g3;
     ecn2 dEA;
     int buf,i;

     //initiate
     h=mirvar(0);rA=mirvar(0);x=mirvar(0);y=mirvar(0);
     Ppube=epoint_init();
     dEA.x.a=mirvar(0); dEA.x.b=mirvar(0);dEA.y.a=mirvar(0);dEA.y.b=mirvar(0);
     dEA.z.a=mirvar(0); dEA.z.b=mirvar(0);dEA.marker=MR_EPOINT_INFINITY;
     zzn12_init(&g1);zzn12_init(&g2);zzn12_init(&g3);

     bytes_to_big(BNLEN,Ppub,x);bytes_to_big(BNLEN,Ppub+BNLEN,y);
     bytes_to_big(BNLEN,randA,rA);
     bytes128_to_ecn2(deA,&dEA);
     epoint_set(x,y,0,Ppube);

     //test if RB is on G1
     if(Test_Point(RB)) return SM9_NOT_VALID_G1;

     //----------------Step A5:g1=(e(P2,Ppube))^rA,g2=e(deA,RB),g3=g2^rA---------
      if(!ecap(P2, Ppube, para_t, X, &g1)) return SM9_MY_ECAP_12A_ERR;
     if(!ecap(dEA,RB, para_t, X, &g2)) return SM9_MY_ECAP_12A_ERR;
     //test if a ZZn12 element is of order q
     if((!member(g1, para_t, X))||(!member(g2, para_t, X))) return SM9_MEMBER_ERR;

     g1=zzn12_pow(g1,rA);
     g3=zzn12_pow(g2,rA);
     printf("\n************************g1=e(Ppub,P2):******************************\n");
     zzn12_ElementPrint(g1);
     printf("\n********************g2=(e(RB,deA))^rB:***************************\n");
     zzn12_ElementPrint(g2);
     printf("\n************************g3=g2^rB:**********************************\n");
     zzn12_ElementPrint(g3);

     //------------------ A6:S1=Hash(0x82||g1||Hash(g2||g3||IDA||IDB||RA||RB))----------
     buf=SM9_KeyEx_Hash(hashid,IDA,IDB,RA,RB,g1,g2,g3,S1);
     if(buf) return buf;
     printf("\n*********S1=Hash(0x82||g1||Hash(g2||g3||IDA||IDB||RA||RB))********\n");
     for(i=0;i<SM3_len/8;i++) printf("%02x",S1[i]);

     if(memcmp(S1,SB,SM3_len/8)) return SM9_ERR_CMP_S1SB;

     //---------- A7: SKA=KDF(IDA||IDB||RA||RB||g1||g2||g3,klen)----------
     buf=SM9_KeyEx_KDF(IDA,IDB,RA,RB,g1,g2,g3,16,SKA);
     if(buf) return buf;
     printf("\n************SKA=KDF(IDA||IDB||RA||RB||g1||g2||g3,klen)************\n");
     for(i=0;i<16;i++) printf("%02x",SKA[i]);

     //---------  A8(optional):SA=Hash(0x83||g1||Hash(g2||g3||IDA||IDB||RA||RB))----------
     hashid[0]=(unsigned char)0x83;
     buf=SM9_KeyEx_Hash(hashid,IDA,IDB,RA,RB,g1,g2,g3,SA);
     if(buf) return buf;
     printf("\n*********SA=Hash(0x83||g1||Hash(g2||g3||IDA||IDB||RA||RB))********\n");
     for(i=0;i<SM3_len/8;i++) printf("%02x",SA[i]);

     return 0;
```

```
}


/**************************************************************
  Function:       SM9_KeyEx_ReB_II
  Description:    Step B10 (optional) verifies the hash value SA received from initiator A
  Calls:          SM9_KeyEx_Hash
  Called By:      SM9_SelfCheck()
  Input:
                  IDA,IDB      //identification of userA and B
                  g1,g2,g3     //R-ate pairings geted from function
SM9_KeyEx_ReB_I,g1=e(RA,deB)g2=(e(P2,Ppub3))^rBg3=g1^rB
                  RA,RB        //temporary value received from initiator A and responder B
                  SA           //a hash value SA calculated by initiator A,verified in this
function
  Output:
                  NULL
  Return:
                  0: success
                  1: asking for memory error
                  A: key exchange failed,form A to B,S2!=SA
  Others:
**************************************************************/
int SM9_KeyEx_ReB_II(unsigned char *IDA,unsigned char *IDB,zzn12 g1,zzn12 g2,zzn12 g3,epoint
*RA,epoint *RB,unsigned char SA[])
{
    unsigned char hashid[]={0x83};
    unsigned char S2[SM3_len/8];
    int buf,i;

    //----------------
B8(optional):S2=Hash(0x83||g1||Hash(g2||g3||IDA||IDB||RA||RB))----------
    buf=SM9_KeyEx_Hash(hashid,IDA,IDB,RA,RB,g1,g2,g3,S2);
    if(buf) return buf;
    printf("\n***************
S2=Hash(0x83||g1||Hash(g2||g3||IDA||IDB||RA||RB))****************\n");
    for(i=0;i<SM3_len/8;i++) printf("%02x",S2[i]);

    if(memcmp(S2,SA,SM3_len/8)) return SM9_ERR_CMP_S2SA;
    return 0;
}

/**************************************************************
  Function:       SM9_SelfCheck
  Description:    SM9 self check
  Calls:          MIRACL functions,SM9_Init(),SM9_GenerateEncryptKey(),SM9_KeyEx_InitA_I,
                  SM9_KeyEx_InitA_II,SM9_KeyEx_ReB_I,SM9_KeyEx_ReB_II
  Called By:
  Input:
  Output:
  Return:         0: self-check success
                  1: asking for memory error
                  2: element is out of order q
                  3: R-ate calculation error
                  4: test if C1 is on G1
                  5: base point P1 error
                  6: base point P2 error
                  7: Encryption public key generated error
                  8: Encryption private key generated error
                  9: key exchange failed,form B to A,S1!=SB
                  A: key exchange failed,form A to B,S2!=SA
                  B: RA generated error
                  C: RB generated error
                  D: SA generated error
                  E: SB generated error
  Others:
*****************************************************************/
int SM9_SelfCheck()
{
    //the master private key
    unsigned char KE[32] =
{0x00,0x02,0xE6,0x5B,0x07,0x62,0xD0,0x42,0xF5,0x1F,0x0D,0x23,0x54,0x2B,0x13,0xED,
0x8C,0xFA,0x2E,0x9A,0x0E,0x72,0x06,0x36,0x1E,0x01,0x3A,0x28,0x39,0x05,0xE3,0x1F};

    unsigned char
randA[32]={0x00,0x00,0x58,0x79,0xDD,0x1D,0x51,0xE1,0x75,0x94,0x6F,0x23,0xB1,0xB4,0x1E,0x93,
 0xBA,0x31,0xC5,0x84,0xAE,0x59,0xA4,0x26,0xEC,0x10,0x46,0xA4,0xD0,0x3B,0x06,0xC8};
    unsigned char
randB[32]={0x00,0x01,0x8B,0x98,0xC4,0x4B,0xEF,0x9F,0x85,0x37,0xFB,0x7D,0x07,0x1B,0x2C,0x92,
 0x8B,0x3B,0xC6,0x5B,0xD3,0xD6,0x9E,0x1E,0xEE,0x21,0x35,0x64,0x90,0x56,0x34,0xFE};
    //standard datas
    unsigned char std_Ppub[64]=
{0x91,0x74,0x54,0x26,0x68,0xE8,0xF1,0x4A,0xB2,0x73,0xC0,0x94,0x5C,0x36,0x90,0xC6,
     0x6E,0x5D,0xD0,0x96,0x78,0xB8,0x6F,0x73,0x4C,0x43,0x50,0x56,0x7E,0xD0,0x62,0x83,
     0x54,0xE5,0x98,0xC6,0xBF,0x74,0x9A,0x3D,0xAC,0xC9,0xFF,0xFE,0xDD,0x9D,0xB6,0x86,
     0x6C,0x50,0x45,0x7C,0xFC,0x7A,0xA2,0xA4,0xAD,0x65,0xC3,0x16,0x8F,0xF7,0x42,0x10};
    unsigned char std_deA[128]=
{0x0F,0xE8,0xEA,0xB3,0x95,0x19,0x9B,0x56,0xBF,0x1D,0x75,0xBD,0x2C,0xD6,0x10,0xB6,
      0x42,0x4F,0x08,0xD1,0x09,0x29,0x22,0xC5,0x88,0x2B,0x52,0xDC,0xD6,0xCA,0x83,0x2A,
      0x7D,0xA5,0x7B,0xC5,0x02,0x41,0xF9,0xE5,0xBF,0xDD,0xC0,0x75,0xDD,0x9D,0x32,0xC7,
      0x77,0x71,0x00,0xD7,0x36,0x91,0x6C,0xFC,0x16,0x5D,0x8D,0x36,0xE0,0x63,0x4C,0xD7,
      0x83,0xA4,0x57,0xDA,0xF5,0x2C,0xAD,0x46,0x4C,0x90,0x3B,0x26,0x06,0x2C,0xAF,0x93,
      0x7B,0xB4,0x0E,0x37,0xDA,0xDE,0xD9,0xED,0xA4,0x01,0x05,0x0E,0x49,0xC8,0xAD,0x0C,
      0x69,0x70,0x87,0x6B,0x9A,0xAD,0x1B,0x7A,0x50,0xBB,0x48,0x63,0xA1,0x1E,0x57,0x4A,
      0xF1,0xFE,0x3C,0x59,0x75,0x16,0x1D,0x73,0xDE,0x4C,0x3A,0xF6,0x21,0xFB,0x1E,0xFB};
    unsigned char std_deB[128]=
{0x74,0xCC,0xC3,0xAC,0x9C,0x38,0x3C,0x60,0xAF,0x08,0x39,0x72,0xB9,0x6D,0x05,0xC7,
      0x5F,0x12,0xC8,0x90,0x7D,0x12,0x8A,0x17,0xAD,0xAF,0xBA,0xB8,0xC5,0xA4,0xAC,0xF7,
      0x01,0x09,0x2F,0xF4,0xDE,0x89,0x36,0x26,0x70,0xC2,0x17,0x11,0xB6,0xDB,0xE5,0x2D,
      0xCD,0x5F,0x8E,0x40,0xC6,0x65,0x4B,0x3D,0xEC,0xE5,0x73,0xC2,0xAB,0x3D,0x29,0xB2,
      0x44,0xB0,0x29,0x4A,0xA0,0x42,0x90,0xE1,0x52,0x4F,0xF3,0xE3,0xDA,0x8C,0xFD,0x43,
      0x2B,0xB6,0x4D,0xE3,0xA8,0x04,0x0B,0x5B,0x88,0xD1,0xB5,0xFC,0x86,0xA4,0xEB,0xC1,
      0x8C,0xFC,0x48,0xFB,0x4F,0xF3,0x7F,0x1E,0x27,0x72,0x74,0x64,0xF3,0xC3,0x4E,0x21,
      0x53,0x86,0x1A,0xD0,0x8E,0x97,0x2D,0x16,0x25,0xFC,0x1A,0x7B,0xD1,0x8D,0x55,0x39};
    unsigned char std_RA[64] =
{0x7C,0xBA,0x5B,0x19,0x06,0x9E,0xE6,0x6A,0xA7,0x9D,0x49,0x04,0x13,0xD1,0x18,0x46,
      0xB9,0xBA,0x76,0xDD,0x22,0x56,0x7F,0x80,0x9C,0xF2,0x3B,0x6D,0x96,0x4B,0xB2,0x65,
      0xA9,0x76,0x0C,0x99,0xCB,0x6F,0x70,0x63,0x43,0xFE,0xD0,0x56,0x37,0x08,0x58,0x64,
      0x95,0x8D,0x6C,0x90,0x90,0x2A,0xBA,0x7D,0x40,0x5F,0xBE,0xDF,0x7B,0x78,0x15,0x99};
    unsigned char std_RB[64] =
{0x86,0x1E,0x91,0x48,0x5F,0xB7,0x62,0x3D,0x27,0x94,0xF4,0x95,0x03,0x1A,0x35,0x59,
      0x8B,0x49,0x3B,0xD4,0x5B,0xE3,0x78,0x13,0xAB,0xC7,0x10,0xFC,0xC1,0xF3,0x44,0x82,
      0x32,0xD9,0x06,0xA4,0x69,0xEB,0xC1,0x21,0x6A,0x80,0x2A,0x70,0x52,0xD5,0x61,0x7C,
      0xD4,0x30,0xFB,0x56,0xFB,0xA7,0x29,0xD4,0x1D,0x9B,0xD6,0x68,0xE9,0xEB,0x96,0x00};
    unsigned char std_SA[32] =
{0x19,0x5D,0x1B,0x72,0x56,0xBA,0x7E,0x0E,0x67,0xC7,0x12,0x02,0xA2,0x5F,0x8C,0x94,
      0xFF,0x82,0x41,0x70,0x2C,0x2F,0x55,0xD6,0x13,0xAE,0x1C,0x6B,0x98,0x21,0x51,0x72};
    unsigned char std_SB[32] =
{0x3B,0xB4,0xBC,0xEE,0x81,0x39,0xC9,0x60,0xB4,0xD6,0x56,0x6D,0xB1,0xE0,0xD5,0xF0,
      0xB2,0x76,0x76,0x80,0xE5,0xE1,0xBF,0x93,0x41,0x03,0xE6,0xC6,0x6E,0x40,0xFF,0xEE};

    unsigned char hid[]={0x02},*IDA="Alice",*IDB="Bob";
    unsigned char Ppub[64],deA[128],deB[128];
    unsigned char xy[64],SA[SM3_len/8],SB[SM3_len/8];
    epoint *RA,*RB;
    big ke,x,y;
    zzn12 g1,g2,g3;
    int tmp,i;

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

```
x=mirvar(0);y=mirvar(0);ke=mirvar(0);
bytes_to_big(32,KE,ke);
RA=epoint_init();RB=epoint_init();
zzn12_init(&g1);zzn12_init(&g2);zzn12_init(&g3);

tmp=SM9_Init();
if(tmp!=0) return tmp;

printf("\n***********************  SM9 key Generation     ***************************\n");
tmp=SM9_GenerateEncryptKey(hid,IDA,strlen(IDA),ke,Ppub,deA);
if(tmp!=0)  return tmp;
tmp=SM9_GenerateEncryptKey(hid,IDB,strlen(IDB),ke,Ppub,deB);
if(tmp!=0)  return tmp;
if(memcmp(Ppub,std_Ppub,64)!=0)
    return SM9_GEPUB_ERR;
if(memcmp(deA,std_deA,128)!=0)
    return SM9_GEPRI_ERR;
if(memcmp(deB,std_deB,128)!=0)
    return SM9_GEPRI_ERR;

printf("\n**********************PublicKey Ppubs=[ke]P1: *************************\n");
for(i=0;i<64;i++) printf("%02x",Ppub[i]);
printf("\n**************The private key deA = (xdeA, ydeA): *********************\n");
for(i=0;i<128;i++) printf("%02x",deA[i]);
printf("\n**************The private key deB = (xdeB, ydeB): *********************\n");
for(i=0;i<128;i++) printf("%02x",deB[i]);


printf("\n////////////////////  SM9 Key exchange A1-A4://////////////////////////\n");
tmp=SM9_KeyEx_InitA_I(hid,IDB,randA,Ppub,deA,RA);
if(tmp!=0) return tmp;
printf("\n ////////////////////////////:RA=[r]QB  //////////////////////////////\n");
epoint_get(RA,x,y);
cotnum(x,stdout);cotnum(y,stdout);
big_to_bytes(BNLEN,x,xy,1);big_to_bytes(BNLEN,y,xy+BNLEN,1);
if(memcmp(xy,std_RA,BNLEN*2)!=0)
     return SM9_ERR_RA;


printf("\n//////////////////////// SM9 Key exchange B1-B7:///////////////////////\n");
tmp=SM9_KeyEx_ReB_I(hid,IDA,IDB,randB, Ppub, deB,RA,RB,SB,&g1,&g2,&g3);
if(tmp!=0) return tmp;
epoint_get(RB,x,y);
big_to_bytes(BNLEN,x,xy,1);big_to_bytes(BNLEN,y,xy+BNLEN,1);
```

```
    if(memcmp(xy,std_RB,BNLEN*2)!=0)
        return SM9_ERR_RB;
    if(memcmp(SB,std_SB,SM3_len/8)!=0)
        return SM9_ERR_SB;

    printf("\n////////////////////////  SM9 Key exchange A5-A8:///////////////////////\n");
    tmp=SM9_KeyEx_InitA_II(IDA, IDB, randA, Ppub, deA,RA,RB, SB, SA);
    if(tmp!=0) return tmp;
    if(memcmp(SA,std_SA,SM3_len/8)!=0)
        return SM9_ERR_SA;

    printf("\n////////////////////////  SM9 Key exchange B8:///////////////////////\n");
    tmp=SM9_KeyEx_ReB_II(IDA, IDB,g1,g2,g3,RA,RB,SA);
    if(tmp!=0) return tmp;

    return 0;
}
```