```
///*********************************************************************
//  File name:    SM9_Key_encap.c
//  Version:      SM9_Key_encap_V1.0
//  Date:         Jan 11,2017
//  Description:  implementation of SM9 Key encapsulation mechanism
//                all operations based on BN curve line function
//  Function List:
//        1.bytes128_to_ecn2        //convert 128 bytes 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_H1                  //function H1 in SM9 standard 5.4.2.2
//        6.SM9_Init                //initiate SM9 curve
//        7.SM9_GenerateEncryptKey  //generate encrypted private and public key
//        8.SM9_Key_Encap           //Key encapsulation
//        9.SM9_Key_Decap           //Key decapsulation
//        10.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_encap.h"
#include "kdf.h"


/*************************************************************
  Function:       bytes128_to_ecn2
  Description:    convert 128 bytes into ecn2
  Calls:          MIRACL functions
  Called By:      SM9_Init,SM9_Key_decap
  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 elements of struct zzn12
  Calls:          MIRACL functions
  Called By:      SM9_Key_encap,SM9_Key_decap
  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_Key_encap,SM9_Key_decap
  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_Key_decap
  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_H1
  Description:    function H1 in SM9 standard 5.4.2.2
  Calls:          MIRACL functions,SM3_KDF
```

```
  Called By:      SM9_GenerateEncryptKey,SM9_Key_encap
  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_Key_encap
  Description:    Key encapsulation
  Calls:          MIRACL functions,zzn12_init,ecap,member,zzn12_pow,SM9_H1,
                  SM3_KDF,LinkCharZzn12,zzn12_ElementPrint,
  Called By:      SM9_SelfCheck()
  Input:
                  hid:0x03
                  IDB         //identification of userB
                  rand        //a random number K lies in [1,N-1]
                  Ppubs       //encryption public key
  Output:
                  C           //cipher
                  K           //Key
  Return:
                  0: success
                  1: asking for memory error
                  2: a zzn12 element is of order
                  3: R-ate pairing generated error
                  9: K equals 0
  Others:
*************************************************************/
```

```
int SM9_Key_encap(unsigned char hid[],unsigned char *IDB,unsigned char rand[],
                    unsigned char Ppub[],unsigned char C[],unsigned char K[],int Klen)
{
    big h,x,y,r;
    epoint *Ppube,*QB,*Cipher;
    unsigned char *Z=NULL;
    int Zlen,buf,i,num=0;
    zzn12 g,w;

    //initiate
    h=mirvar(0);r=mirvar(0);x=mirvar(0);y=mirvar(0);
    QB=epoint_init();Ppube=epoint_init();Cipher=epoint_init();
    zzn12_init(&g);zzn12_init(&w);

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

    //----------Step1: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);
    free(Z);
    if(buf) return buf;
    printf("\n********************** H1(IDB||hid,N) **********************\n");
    cotnum(h,stdout);

    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);

    //-------------------- Step2:randnom -------------------
    bytes_to_big(BNLEN,rand,r);
    printf("\n**********************randnum r: ****************************\n");
    cotnum(r,stdout);

    //---------------Step3:C=[r]QB-----------------------
    ecurve_mult(r,QB,Cipher);
    epoint_get(Cipher,x,y);
```

```
    printf("\n********************** C=[r]QB: ****************************\n");
    cotnum(x,stdout);cotnum(y,stdout);
    big_to_bytes(BNLEN,x,C,1);big_to_bytes(BNLEN,y,C+BNLEN,1);

    //----------------Step4:g=e(Ppube,P2)-----------------------
    if(!ecap(P2, Ppube, para_t, X, &g)) return SM9_MY_ECAP_12A_ERR;
    //test if a ZZn12 element is of order q
    if(!member(g, para_t, X)) return SM9_MEMBER_ERR;

    printf("\n**********************g=e(Ppube,P2):****************************\n");
    zzn12_ElementPrint(g);

    //----------------Step5:w=g^r-------------------------
    w=zzn12_pow(g,r);
    printf("\n************************ w=g^r:*********************************\n");
    zzn12_ElementPrint(w);

    //----------------Step6:K=KDF(C||w||IDB,klen)-------------------------
    Zlen=strlen(IDB)+BNLEN*14;
    Z=(char *)malloc(sizeof(char)*(Zlen+1));
    if(Z==NULL) return SM9_ASK_MEMORY_ERR;
    LinkCharZzn12(C,BNLEN*2, w,Z,BNLEN*14);
    memcpy(Z+BNLEN*14,IDB,strlen(IDB));

    SM3_KDF(Z,Zlen,Klen,K);
    free(Z);
    //----------------test if K equals 0-------------------------
    printf("\n******************* K=KDF(C||w||IDB,klen):**********************\n");
    for(i=0;i<Klen;i++)
    {
        if(K[i]==0) num+=1;
        printf("%02x",K[i]);
    }
    if(num==Klen) return SM9_ERR_K1_ZERO;

    return 0;
}

/*****************************************************************
  Function:       SM9_Key_decap
  Description:    Key decapsulation
  Calls:          MIRACL functions,zzn12_init,ecap,member,Test_Point,
                  zzn12_ElementPrint,SM3_KDF,bytes128_to_ecn2,LinkCharZzn12
  Called By:      SM9_SelfCheck()
```

```
  Input:
                hid:0x03
                IDB         //identification of userB
                rand        //a random number K lies in [1,N-1]
                Ppubs       //encryption public key
  Output:
                C           //cipher
                K           //Key
  Return:
                0: success
                1: asking for memory error
                2: a zzn12 element is of order
                3: R-ate pairing generated error
                4: C is not valid element of G1
                9: K equals 0
  Others:
*************************************************************/
int SM9_Key_decap(unsigned char *IDB,unsigned char deB[],unsigned char C[],int Klen,unsigned char K[])
{
    big h,x,y;
    epoint *Cipher;
    unsigned char *Z=NULL;
    int Zlen,i,num=0;
    zzn12 w;
    ecn2 dEB;

    //initiate
    h=mirvar(0);x=mirvar(0);y=mirvar(0);
    Cipher=epoint_init();
    zzn12_init(&w);
    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,C,x);
    bytes_to_big(BNLEN,C+BNLEN,y);
    epoint_set(x,y,0,Cipher);
    bytes128_to_ecn2(deB,&dEB);

    //----------Step1:test if C is on G1-----------------
    if(Test_Point(Cipher)) return SM9_NOT_VALID_G1;

    //----------Step2:calculate w=e(C,deB)-----------------
    if(!ecap(dEB, Cipher, para_t, X, &w)) return SM9_MY_ECAP_12A_ERR;
```

```
    //test if a ZZn12 element is of order q
    if(!member(w, para_t, X)) return SM9_MEMBER_ERR;

    printf("\n**********************w=e(C,deB):**************************\n");
    zzn12_ElementPrint(w);

    //----------Step3:K=KDF(C||w'||IDB,klen)-----------------------
    Zlen=strlen(IDB)+BNLEN*14;
    Z=(char *)malloc(sizeof(char)*(Zlen+1));
    if(Z==NULL) return SM9_ASK_MEMORY_ERR;
    LinkCharZzn12(C,BNLEN*2, w,Z,BNLEN*14);
    memcpy(Z+BNLEN*14,IDB,strlen(IDB));
    SM3_KDF(Z,Zlen,Klen,K);

    //----------------test if K equals 0-----------------------
    printf("\n******************* K=KDF(C||w||IDB,klen):**********************\n");
    for(i=0;i<Klen;i++)
    {
        if(K[i]==0) num+=1;
        printf("%02x",K[i]);
    }
    if(num==Klen) return SM9_ERR_K1_ZERO;

    free(Z);
    return 0;
}

/*************************************************************
  Function:       SM9_SelfCheck
  Description:    SM9 self check
  Calls:          MIRACL functions,SM9_Init(),SM9_GenerateEncryptKey(),SM9_Key_encap,
                  SM9_Key_decap
  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 C 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: K equals 0
                A: cipher error in key encapsulation
                B: key to be encapsulated
                C: key generated by decapsulation
  Others:
*************************************************************/
int SM9_SelfCheck()
{
    //the master private key
    unsigned char KE[32] =
{0x00,0x01,0xED,0xEE,0x37,0x78,0xF4,0x41,0xF8,0xDE,0xA3,0xD9,0xFA,0x0A,0xCC,0x4E,
0x07,0xEE,0x36,0xC9,0x3F,0x9A,0x08,0x61,0x8A,0xF4,0xAD,0x85,0xCE,0xDE,0x1C,0x22};

    unsigned char
rand[32]={0x00,0x00,0x74,0x01,0x5F,0x84,0x89,0xC0,0x1E,0xF4,0x27,0x04,0x56,0xF9,0xE6,0x47,
 0x5B,0xFB,0x60,0x2B,0xDE,0x7F,0x33,0xFD,0x48,0x2A,0xB4,0xE3,0x68,0x4A,0x67,0x22};
    //standard datas
    unsigned char std_Ppub[64]=
{0x78,0x7E,0xD7,0xB8,0xA5,0x1F,0x3A,0xB8,0x4E,0x0A,0x66,0x00,0x3F,0x32,0xDA,0x5C,
      0x72,0x0B,0x17,0xEC,0xA7,0x13,0x7D,0x39,0xAB,0xC6,0x6E,0x3C,0x80,0xA8,0x92,0xFF,
      0x76,0x9D,0xE6,0x17,0x91,0xE5,0xAD,0xC4,0xB9,0xFF,0x85,0xA3,0x13,0x54,0x90,0x0B,
      0x20,0x28,0x71,0x27,0x9A,0x8C,0x49,0xDC,0x3F,0x22,0x0F,0x64,0x4C,0x57,0xA7,0xB1};
    unsigned char std_deB[128]=
{0x94,0x73,0x6A,0xCD,0x2C,0x8C,0x87,0x96,0xCC,0x47,0x85,0xE9,0x38,0x30,0x1A,0x13,
      0x9A,0x05,0x9D,0x35,0x37,0xB6,0x41,0x41,0x40,0xB2,0xD3,0x1E,0xEC,0xF4,0x16,0x83,
      0x11,0x5B,0xAE,0x85,0xF5,0xD8,0xBC,0x6C,0x3D,0xBD,0x9E,0x53,0x42,0x97,0x9A,0xCC,
      0xCF,0x3C,0x2F,0x4F,0x28,0x42,0x0B,0x1C,0xB4,0xF8,0xC0,0xB5,0x9A,0x19,0xB1,0x58,
      0x7A,0xA5,0xE4,0x75,0x70,0xDA,0x76,0x00,0xCD,0x76,0x0A,0x0C,0xF7,0xBE,0xAF,0x71,
      0xC4,0x47,0xF3,0x84,0x47,0x53,0xFE,0x74,0xFA,0x7B,0xA9,0x2C,0xA7,0xD3,0xB5,0x5F,
      0x27,0x53,0x8A,0x62,0xE7,0xF7,0xBF,0xB5,0x1D,0xCE,0x08,0x70,0x47,0x96,0xD9,0x4C,
      0x9D,0x56,0x73,0x4F,0x11,0x9E,0xA4,0x47,0x32,0xB5,0x0E,0x31,0xCD,0xEB,0x75,0xC1};
    unsigned char std_K[64] =
{0x4F,0xF5,0xCF,0x86,0xD2,0xAD,0x40,0xC8,0xF4,0xBA,0xC9,0x8D,0x76,0xAB,0xDB,0xDE,
    0x0C,0x0E,0x2F,0x0A,0x82,0x9D,0x3F,0x91,0x1E,0xF5,0xB2,0xBC,0xE0,0x69,0x54,0x80};
    unsigned char std_C[64] =
{0x1E,0xDE,0xE2,0xC3,0xF4,0x65,0x91,0x44,0x91,0xDE,0x44,0xCE,0xFB,0x2C,0xB4,0x34,
    0xAB,0x02,0xC3,0x08,0xD9,0xDC,0x5E,0x20,0x67,0xB4,0xFE,0xD5,0xAA,0xAC,0x8A,0x0F,
    0x1C,0x9B,0x4C,0x43,0x5E,0xCA,0x35,0xAB,0x83,0xBB,0x73,0x41,0x74,0xC0,0xF7,0x8F,
    0xDE,0x81,0xA5,0x33,0x74,0xAF,0xF3,0xB3,0x60,0x2B,0xBC,0x5E,0x37,0xBE,0x9A,0x4C};

    unsigned char hid[]={0x03},*IDB="Bob";
    unsigned char Ppub[64],deB[128],C[64],K[32],K_decap[32];
    big ke;
    int tmp,i;
```

```
int Klen=32;

mip=mirsys(1000, 16);
mip->IOBASE=16;
ke=mirvar(0);
bytes_to_big(32,KE,ke);

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

printf("\n**********************  SM9 key Generation    **************************\n");
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(deB,std_deB,128)!=0)
    return SM9_GEPRI_ERR;

printf("\n**********************PublicKey Ppubs=[ke]P1: ************************\n");
for(i=0;i<64;i++)
{if(i==32) printf("\n");
  printf("%02x",Ppub[i]);}
printf("\n**************The private key deB = (xdeB, ydeB): ********************\n");
for(i=0;i<128;i++)
{   if(i==64) printf("\n");
    printf("%02x",deB[i]);}

printf("\n//////////////////SM9 Key encapsulation mechanism//////////////////////\n");
tmp=SM9_Key_encap( hid,IDB,rand, Ppub, C, K,Klen);
if(tmp!=0) return tmp;

if(memcmp(C,std_C,64)!=0)
     return SM9_ERR_Encap_C;
if(memcmp(K,std_K,Klen)!=0)
     return SM9_ERR_Encap_K;

printf("\n//////////////////SM9 Key decapsulation mechanism//////////////////////\n");
tmp=SM9_Key_decap(IDB, deB, C,Klen,K_decap);
if(tmp!=0) return tmp;

if(memcmp(K_decap,std_K,32)!=0)
     return SM9_ERR_Decap_K;

return 0;
```

```
}
```