```
///**********************************************************************
//  File name:    SM9_enc_dec.c
//  Version:      SM9_enc_dec_V1.0
//  Date:         Dec 29,2016
//  Description:  implementation of SM9 encryption algorithm and decryption algorithm
//                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.ecn2_Bytes128_Print     //print 128 bytes of ecn2
//        4.LinkCharZzn12           //link two different types(unsigned char and zzn12)to
one(unsigned char)
//        5.Test_Point              //test if the given point is on SM9 curve
//        6.SM4_Block_Encrypt       //encrypt the message with padding,according to PKS#5
//        7.SM4_Block_Decrypt       //decrypt the cipher with padding,according to PKS#5
//        8.SM9_H1                  //function H1 in SM9 standard 5.4.2.2
//        9.SM9_Enc_MAC             //MAC in SM9 standard 5.4.5
//        10.SM9_Init               //initiate SM9 curve
//        11.SM9_GenerateEncryptKey //generate encrypted private and public key
//        12.SM9_Encrypt            //SM9 encryption algorithm
//        13.SM9_Decrypt            //SM9 decryption algorithm
//        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_enc_dec.h"
#include "kdf.h"
#include "SM4.h"


/*************************************************************
  Function:       bytes128_to_ecn2
  Description:    convert 128 bytes into ecn2
  Calls:          MIRACL functions
  Called By:      SM9_Init,SM9_Decrypt
  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_Encrypt,SM9_Decrypt
  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:       ecn2_Bytes128_Print
  Description:    print 128 bytes of ecn2
  Calls:          MIRACL functions
  Called By:      SM9_Encrypt,SM9_Decrypt
  Input:          ecn2 x
  Output:         NULL
  Return:         NULL
  Others:
*****************************************************************/
void ecn2_Bytes128_Print(ecn2 x)
{
    big tmp;
    tmp=mirvar(0);

    redc(x.x.b,tmp);cotnum(tmp,stdout);
    redc(x.x.a,tmp);cotnum(tmp,stdout);
    redc(x.y.b,tmp);cotnum(tmp,stdout);
    redc(x.y.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_Encrypt,SM9_Decrypt
  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:
  Called By:      SM9_Decrypt
  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:       SM4_Block_Encrypt
  Description:    encrypt the message with padding,according to PKS#5
  Calls:          SM4_Encrypt
  Called By:      SM9_Encrypt
  Input:
                  key:the key of SM4
                  message:data to be encrypted
                  mlen:   the length of message
  Output:
                  cipher: ciphertext
                  cipher_len:the length of ciphertext
  Return:         NULL
  Others:
*****************************************************************/
void SM4_Block_Encrypt(unsigned char key[],unsigned char * message,int mlen,unsigned char
*cipher,int * cipher_len)
{
    unsigned char mess[16];
    int i,rem=mlen%16;

    for(i=0;i<mlen/16;i++)
        SM4_Encrypt(key,&message[i*16],&cipher[i*16]);
```

```
    //encrypt the last block
    memset(mess,16-rem,16);
    if(rem)
        memcpy(mess,&message[i*16],rem);
    SM4_Encrypt(key,mess,&cipher[i*16]);
}
/*************************************************************
  Function:       SM4_Block_Decrypt
  Description:    decrypt the cipher with padding,according to PKS#5
  Calls:          SM4_Decrypt
  Called By:      SM9_Decrypt
  Input:
                  key:the key of SM4
                  cipher: ciphertext
                  mlen:   the length of ciphertext
  Output:
                  plain: plaintext
                  plain_len:the length of plaintext
  Return:         NULL
  Others:
*************************************************************/
void SM4_Block_Decrypt(unsigned char key[],unsigned char *cipher,int len,unsigned char
*plain,int *plain_len)
{
    int i;
    for(i=0;i<len/16;i++)
       SM4_Decrypt(key,cipher+i*16,plain+i*16);
    *plain_len=len-plain[len-1];
}


/*************************************************************
  Function:       SM9_H1
  Description:    function H1 in SM9 standard 5.4.2.2
  Calls:          MIRACL functions,SM3_KDF
  Called By:      SM9_Encrypt
  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_Enc_MAC
  Description:    MAC in SM9 standard 5.4.5
  Calls:          SM3_256
  Called By:      SM9_Encrypt,SM9_Decrypt
  Input:
                  K:key
                  Klen:the length of K
                  M:message
                  Mlen:the length of message
```

```
  Output:         C=MAC(K,Z)
  Return:         0: success;
                  1: asking for memory error
  Others:
*****************************************************************/
int SM9_Enc_MAC(unsigned char *K,int Klen,unsigned char *M,int Mlen,unsigned char C[])
{
    unsigned char *Z=NULL;
    int len=Klen+Mlen;
    Z=(char *)malloc(sizeof(char)*(len+1));
    if(Z==NULL) return SM9_ASK_MEMORY_ERR;
    memcpy(Z,M,Mlen);
    memcpy(Z+Mlen,K,Klen);
    SM3_256(Z,len,C);

    free(Z);
    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;

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

    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,ecn2_Bytes128_Print
  Called By:      SM9_SelfCheck
  Input:          hid:0x03
                  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);

printf("\n**************The private key deB = (xdeB, ydeB)：**********\n");
ecn2_Bytes128_Print(dEB);
printf("\n**********PublicKey Ppubs=[ke]P1：**********\n");
epoint_get(Ppub,xPpub,yPpub);
cotnum(xPpub,stdout);cotnum(yPpub,stdout);

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_Encrypt
  Description:    SM9 encryption algorithm
  Calls:          MIRACL functions,zzn12_init(),ecap(),member(),zzn12_ElementPrint(),
                  zzn12_pow(),LinkCharZzn12(),SM3_KDF(),SM9_Enc_MAC(),SM4_Block_Encrypt()
  Called By:      SM9_SelfCheck()
  Input:
                  hid:0x03
                  IDB          //identification of userB
                  message      //the message to be encrypted
                  len          //the length of message
                  rand         //a random number K lies in [1,N-1]
                  EncID        //encryption identification,0:stream cipher  1:block cipher
                  k1_len       //the byte length of K1 in block cipher algorithm
                  k2_len       //the byte length of K2 in MAC algorithm
                  Ppubs        //encrtption public key

  Output:         C            //cipher C1||C3||C2
                  Clen         //the byte length of C
  Return:
                  0: success
                  1: asking for memory error
                  2: element is out of order q
                  3: R-ate calculation error
                  A: K1 equals 0
  Others:
*************************************************************/
int SM9_Encrypt(unsigned char hid[],unsigned char *IDB,unsigned char *message,int mlen,unsigned
char rand[],
                int EncID,int k1_len,int k2_len,unsigned char Ppub[],unsigned char C[],int
*C_len)
{
     big h,x,y,r;
     zzn12 g,w;
     epoint *Ppube,*QB,*C1;
     unsigned char *Z=NULL,*K=NULL,*C2=NULL,C3[SM3_len/8];
     int i=0,j=0,Zlen,buf,klen,C2_len;
```

```
//initiate
h=mirvar(0);r=mirvar(0);x=mirvar(0);y=mirvar(0);
QB=epoint_init();Ppube=epoint_init();C1=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);
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);

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

//Step3:C1=[r]QB
ecurve_mult(r,QB,C1);
printf("\n*************************:C1=[r]QB*******************************\n");
epoint_get(C1,x,y);
cotnum(x,stdout);cotnum(y,stdout);
big_to_bytes(BNLEN,x,C,1);big_to_bytes(BNLEN,y,C+BNLEN,1);

//Step4:g = e(P2, Ppub-e)
 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(P2,Ppube):***************************\n");
zzn12_ElementPrint(g);

//Step5:calculate w=g^r
```

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

free(Z);
//Step6:calculate C2
if(EncID==0)
{
    C2_len=mlen;
    *C_len=BNLEN*2+SM3_len/8+C2_len;

    //Step:6-1: calculate K=KDF(C1||w||IDB,klen)
    klen=mlen+k2_len;
    Zlen=strlen(IDB)+BNLEN*14;
    Z=(char *)malloc(sizeof(char)*(Zlen+1));
    K=(char *)malloc(sizeof(char)*(klen+1));
    C2=(char *)malloc(sizeof(char)*(mlen+1));
    if(Z==NULL|| K==NULL|| C2==NULL) return SM9_ASK_MEMORY_ERR;

    LinkCharZzn12( C,BNLEN*2,w,Z,(Zlen-strlen(IDB)) );
    memcpy(Z+BNLEN*14,IDB,strlen(IDB));
    SM3_KDF(Z,Zlen,klen,K);
    printf("\n*****************K=KDF(C1||w||IDB,klen):**********************\n");
    for(i=0;i<klen;i++) printf("%02x",K[i]);

    //Step:6-2: calculate C2=M^K1,and test if K1==0?
    for(i=0;i<mlen;i++)
    {
        if(K[i]==0) j=j+1;
        C2[i]=message[i]^K[i];
    }
    if(j==mlen) return SM9_ERR_K1_ZERO;
    printf("\n************************ C2=M^K1 :***************************\n");
    for(i=0;i<C2_len;i++) printf("%02x",C2[i]);

    //Step7:calculate C3=MAC(K2,C2)
    SM9_Enc_MAC(K+mlen,k2_len,C2,mlen,C3);
    printf("\n********************** C3=MAC(K2,C2):************************\n");
    for(i=0;i<32;i++) printf("%02x",C3[i]);

    memcpy(C+BNLEN*2,C3,SM3_len/8);
    memcpy(C+BNLEN*2+SM3_len/8,C2,C2_len);
    free(Z);free(K);free(C2);
}
```

```
    else
    {
        C2_len=(mlen/16+1)*16;
        *C_len=BNLEN*2+SM3_len/8+C2_len;

        //Step:6-1: calculate K=KDF(C1||w||IDB,klen)
        klen=k1_len+k2_len;
        Zlen=strlen(IDB)+BNLEN*14;
        Z=(char *)malloc(sizeof(char)*(Zlen+1));
        K=(char *)malloc(sizeof(char)*(klen+1));
        C2=(char *)malloc(sizeof(char)*(C2_len+1));
        if(Z==NULL|| K==NULL|| C2==NULL) return SM9_ASK_MEMORY_ERR;

        LinkCharZzn12(C,BNLEN*2,w,Z,Zlen-strlen(IDB));
        memcpy(Z+BNLEN*14,IDB,strlen(IDB));
        SM3_KDF(Z,Zlen,klen,K);
        printf("\n*****************K=KDF(C1||w||IDB,klen):**********************\n");
        for(i=0;i<klen;i++) printf("%02x",K[i]);

        //Step:6-2: calculate C2=Enc(K1,M),and also test if K1==0?
        for(i=0;i<k1_len;i++)
        {
            if(K[i]==0) j=j+1;
        }
        if(j==k1_len) return SM9_ERR_K1_ZERO;

        SM4_Block_Encrypt(K,message,mlen,C2,&C2_len);
        printf("\n********************** C2=Enc(K1,M) :************************\n");
        for(i=0;i<C2_len;i++) printf("%02x",C2[i]);

        //Step7:calculate C3=MAC(K2,C2)
        SM9_Enc_MAC(K+k1_len,k2_len,C2,C2_len,C3);
        printf("\n********************** C3=MAC(K2,C2):************************\n");
        for(i=0;i<32;i++) printf("%02x",C3[i]);

        memcpy(C+BNLEN*2,C3,SM3_len/8);
        memcpy(C+BNLEN*2+SM3_len/8,C2,C2_len);
        free(Z);free(K);free(C2);
    }
    return 0;
}
/*************************************************************
  Function:       SM9_Decrypt
  Description:    SM9 Decryption algorithm
```

```
  Calls:          MIRACL functions,zzn12_init(),Test_Point(), ecap(),
                  member(),zzn12_ElementPrint(),LinkCharZzn12(),SM3_KDF(),
                  SM9_Enc_MAC(),SM4_Block_Decrypt(),bytes128_to_ecn2()

  Called By:      SM9_SelfCheck()
  Input:
                  C           //cipher C1||C3||C2
                  C_len       //the byte length of C
                  deB         //private key of user B
                  IDB         //identification of userB
                  EncID       //encryption identification,0:stream cipher  1:block cipher
                  k1_len      //the byte length of K1 in block cipher algorithm
                  k2_len      //the byte length of K2 in MAC algorithm
  Output:
                  M           //message
                  Mlen:       //the length of message
  Return:
                  0: 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
                  A: K1 equals 0
                  B: compare error of C3
  Others:
*****************************************************************/
int SM9_Decrypt (unsigned char C[],int C_len,unsigned char deB[],unsigned char *IDB,int EncID,
                 int k1_len,int k2_len,unsigned char M[],int * Mlen)
{
    big x,y;
    epoint *C1;
    zzn12 w;
    ecn2 dEB;
    int mlen,klen,Zlen,i,number=0;
    unsigned char *Z=NULL,*K=NULL,*K1=NULL,u[SM3_len/8];

    x=mirvar(0);y=mirvar(0);
    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;
    C1=epoint_init();zzn12_init(&w);

    bytes_to_big(BNLEN,C,x);bytes_to_big(BNLEN,C+BNLEN,y);
    bytes128_to_ecn2(deB,&dEB);
```

```
//Step1:get C1,and test if C1 is on G1
epoint_set(x,y,1,C1);
if(Test_Point(C1)) return SM9_C1_NOT_VALID_G1;

//Step2:w = e(C1, deB)
 if(!ecap(dEB, C1, 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(C1, deB):**************************\n");
zzn12_ElementPrint(w);

//Step3:Calculate plaintext
mlen=C_len-BNLEN*2-SM3_len/8;
if(EncID==0)
{
    //Step3-1:calculate K=KDF(C1||w||IDB,klen)
    klen=mlen+k2_len;
    Zlen=strlen(IDB)+BNLEN*14;
    Z=(char *)malloc(sizeof(char)*(Zlen+1));
    K=(char *)malloc(sizeof(char)*(klen+1));
    if(Z==NULL || K==NULL)  return SM9_ASK_MEMORY_ERR;

    LinkCharZzn12(C,BNLEN*2,w,Z,Zlen-strlen(IDB));
    memcpy(Z+BNLEN*14,IDB,strlen(IDB));
    SM3_KDF(Z,Zlen,klen,K);
    printf("\n******************K=KDF(C1||w||IDB,klen):**********************\n");
    for(i=0;i<klen;i++) printf("%02x",K[i]);

    //Step:3-2: calculate M=C2^K1,and test if K1==0?
    for(i=0;i<mlen;i++)
     {
         if(K[i]==0) number+=1;
         M[i]=C[i+C_len-mlen]^K[i];
     }
    if(number==mlen) return SM9_ERR_K1_ZERO;
    *Mlen=mlen;

    //Step4:calculate u=MAC(K2,C2)
    SM9_Enc_MAC(K+mlen,k2_len,&C[C_len-mlen],mlen,u);
    if(memcmp(u,&C[BNLEN*2],SM3_len/8)) return SM9_C3_MEMCMP_ERR;

    printf("\n****************************** M:******************************\n");
    for(i=0;i<mlen;i++) printf("%02x",M[i]);
    free(Z);free(K);
```

```
    }
    else
    {
        //Step:3-1: calculate K=KDF(C1||w||IDB,klen)
        klen=k1_len+k2_len;
        Zlen=strlen(IDB)+BNLEN*14;
        Z=(char *)malloc(sizeof(char)*(Zlen+1));
        K=(char *)malloc(sizeof(char)*(klen+1));
        K1=(char *)malloc(sizeof(char)*(k1_len+1));
        if(Z==NULL|| K==NULL|| K1==NULL) return SM9_ASK_MEMORY_ERR;

        LinkCharZzn12(C,BNLEN*2,w,Z,Zlen-strlen(IDB));
        memcpy(Z+BNLEN*14,IDB,strlen(IDB));
        SM3_KDF(Z,Zlen,klen,K);
        printf("\n*****************K=KDF(C1||w||IDB,klen):**********************\n");
        for(i=0;i<klen;i++) printf("%02x",K[i]);

        //Step:3-2: calculate M=dec(K1,C2),and test if K1==0?
        for(i=0;i<k1_len;i++)
        {
            if(K[i]==0) number+=1;
            K1[i]=K[i];
        }
        if(number==k1_len) return SM9_ERR_K1_ZERO;
        SM4_Block_Decrypt(K1,&C[C_len-mlen],mlen,M,Mlen);

        //Step4:calculate u=MAC(K2,C2)
        SM9_Enc_MAC(K+k1_len,k2_len,&C[C_len-mlen],mlen,u);
        if(memcmp(u,&C[BNLEN*2],SM3_len/8)) return SM9_C3_MEMCMP_ERR;
        free(Z);free(K);free(K1);
    }
    return 0;
}

/*************************************************************
  Function:       SM9_SelfCheck
  Description:    SM9 self check
  Calls:          MIRACL functions,SM9_Init(),SM9_GenerateEncryptKey(),
                  SM9_Encrypt,SM9_Decrypt
  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: encryption error
                    A: K1 equals 0
                    B: compare error of C3
                    C: decryption error
  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]={0x00,0x00,0xAA,0xC0,0x54,0x17,0x79,0xC8,0xFC,0x45,0xE3,0xE2,0xCB,0x25,0xC1,0x2B,
0x5D,0x25,0x76,0xB2,0x12,0x9A,0xE8,0xBB,0x5E,0xE2,0xCB,0xE5,0xEC,0x9E,0x78,0x5C};
    //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_C_stream[116]=
{0x24,0x45,0x47,0x11,0x64,0x49,0x06,0x18,0xE1,0xEE,0x20,0x52,0x8F,0xF1,0xD5,0x45,
   0xB0,0xF1,0x4C,0x8B,0xCA,0xA4,0x45,0x44,0xF0,0x3D,0xAB,0x5D,0xAC,0x07,0xD8,0xFF,
   0x42,0xFF,0xCA,0x97,0xD5,0x7C,0xDD,0xC0,0x5E,0xA4,0x05,0xF2,0xE5,0x86,0xFE,0xB3,
   0xA6,0x93,0x07,0x15,0x53,0x2B,0x80,0x00,0x75,0x9F,0x13,0x05,0x9E,0xD5,0x9A,0xC0,
```

```
    0xBA,0x67,0x23,0x87,0xBC,0xD6,0xDE,0x50,0x16,0xA1,0x58,0xA5,0x2B,0xB2,0xE7,0xFC,
    0x42,0x91,0x97,0xBC,0xAB,0x70,0xB2,0x5A,0xFE,0xE3,0x7A,0x2B,0x9D,0xB9,0xF3,0x67,
    0x1B,0x5F,0x5B,0x0E,0x95,0x14,0x89,0x68,0x2F,0x3E,0x64,0xE1,0x37,0x8C,0xDD,0x5D,
    0xA9,0x51,0x3B,0x1C};
     unsigned char std_C_cipher[128]=
{0x24,0x45,0x47,0x11,0x64,0x49,0x06,0x18,0xE1,0xEE,0x20,0x52,0x8F,0xF1,0xD5,0x45,
    0xB0,0xF1,0x4C,0x8B,0xCA,0xA4,0x45,0x44,0xF0,0x3D,0xAB,0x5D,0xAC,0x07,0xD8,0xFF,
    0x42,0xFF,0xCA,0x97,0xD5,0x7C,0xDD,0xC0,0x5E,0xA4,0x05,0xF2,0xE5,0x86,0xFE,0xB3,
    0xA6,0x93,0x07,0x15,0x53,0x2B,0x80,0x00,0x75,0x9F,0x13,0x05,0x9E,0xD5,0x9A,0xC0,
    0xFD,0x3C,0x98,0xDD,0x92,0xC4,0x4C,0x68,0x33,0x26,0x75,0xA3,0x70,0xCC,0xEE,0xDE,
    0x31,0xE0,0xC5,0xCD,0x20,0x9C,0x25,0x76,0x01,0x14,0x9D,0x12,0xB3,0x94,0xA2,0xBE,
    0xE0,0x5B,0x6F,0xAC,0x6F,0x11,0xB9,0x65,0x26,0x8C,0x99,0x4F,0x00,0xDB,0xA7,0xA8,
    0xBB,0x00,0xFD,0x60,0x58,0x35,0x46,0xCB,0xDF,0x46,0x49,0x25,0x08,0x63,0xF1,0x0A};
     unsigned char *std_message="Chinese IBE standard";
     unsigned char hid[]={0x03};
     unsigned char *IDB="Bob";

     unsigned char Ppub[64],deB[128];
     unsigned char message[1000],C[1000];
     int M_len,C_len;//M_len the length of message //C_len the length of C
     int k1_len=16,k2_len=32;
     int EncID=0;//0,stream  //1 block
     int tmp,i;
     big ke;

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

     ke=mirvar(0);
     bytes_to_big(32,KE,ke);

     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**********  SM9 encrypt algorithm **********\n");
     tmp= SM9_Encrypt(hid,IDB,std_message,strlen(std_message),rand,
EncID,k1_len,k2_len,Ppub,C,&C_len);
     if(tmp!=0) return tmp;
     printf("\n**********
```

```
Cipher:**********************************\n");
          for(i=0;i<C_len;i++) printf("%02x",C[i]);
    if(EncID==0) tmp=memcmp(C,std_C_stream,C_len);else tmp=memcmp(C,std_C_cipher,C_len);
    if(tmp) return SM9_ENCRYPT_ERR;

    printf("\n**********************  SM9 Decrypt algorithm ************************\n");
    tmp=SM9_Decrypt (std_C_cipher,128,deB,IDB,2, k1_len, k2_len, message,&M_len);
    printf("\n***************************** Message:**********************************\n");
          for(i=0;i<M_len;i++) printf("%02x",message[i]);
    if(tmp!=0) return tmp;
    if(memcmp(message,std_message,M_len)!=0)
        return SM9_DECRYPT_ERR;

    return 0;
}
```