```
///*********************************************************************
//  File name:    SM9_sv.c
//  Version:      SM9_sv_V1.0
//  Date:         Dec 15,2016
//  Description:  implementation of SM9 signature algorithm and verification 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.Test_Range           //test if the big x belong to the range[1,N-1]
//        7.SM9_Init             //initiate SM9 curve
//        8.SM9_H1               //function H1 in SM9 standard 5.4.2.2
//        9.SM9_H2               //function H2 in SM9 standard 5.4.2.3
//        10.SM9_GenerateSignKey //generate signed private and public key
//        11.SM9_Sign            //SM9 signature algorithm
//        12.SM9_Verify          //SM9 verification
//        13.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_sv.h"
#include "kdf.h"


/*************************************************************
  Function:       bytes128_to_ecn2
  Description:    convert 128 bytes into ecn2
  Calls:          MIRACL functions
  Called By:      SM9_Init
  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_Sign,SM9_Verify
  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_Sign,SM9_Verify
  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_Sign,SM9_Verify
  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_Verify
  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:       Test_Range
  Description:    test if the big x belong to the range[1,n-1]
  Calls:
  Called By:      SM9_Verify
  Input:          big x    ///a miracl data type
  Output:         null
  Return:         0: success
                  1: x==n,fail
  Others:
*****************************************************************/
int Test_Range(big x)
{
    big one,decr_n;

    one=mirvar(0);
    decr_n=mirvar(0);

    convert(1,one);
    decr(N,1,decr_n);

    if( (mr_compare(x,one) < 0)| (mr_compare(x,decr_n)>0) )
        return 1;
```

```
    return 0;
}

/*****************************************************************
  Function:       SM9_Init
  Description:    Initiate SM9 curve
  Calls:          MIRACL functions
  Called By:      SM9_SelfCheck
  Input:          null
  Output:         null
  Return:         0: success;
                  7: base point P1 error
                  8: 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_H1
  Description:    function H1 in SM9 standard 5.4.2.2
  Calls:          MIRACL functions,SM3_KDF
  Called By:      SM9_Verify
  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_H2
  Description:    function H2 in SM9 standard 5.4.2.3
  Calls:          MIRACL functions,SM3_KDF
  Called By:      SM9_Sign,SM9_Verify
  Input:          Z:
                  Zlen:the length of Z
                  n:Frobniues constant X
  Output:         h2=H2(Z,Zlen)
  Return:         0: success;
                  1: asking for memory error
  Others:
*****************************************************************/
int SM9_H2(unsigned char Z[],int Zlen,big n,big h2)
{
    int hlen,ZHlen,i;
    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]=0x02;
    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,h2);
   free(ZH);free(ha);
   return 0;
}


/*************************************************************
  Function:       SM9_GenerateSignKey
  Description:    Generate Signed key
  Calls:          MIRACL functions,SM9_H1,xgcd,ecn2_Bytes128_Print
  Called By:      SM9_SelfCheck
  Input:          hid:0x01
                  ID:identification
                  IDlen:the length of ID
                  ks:master private key used to generate signature public key and private key
  Output:         Ppub:signature public key
                  dSA: signature private key
  Return:         0: success;
                  1: asking for memory error
  Others:
*************************************************************/
int SM9_GenerateSignKey(unsigned char hid[],unsigned char *ID,int IDlen,big ks,unsigned char
Ppubs[],unsigned char dsa[])
```

```
{
    big h1,t1,t2,rem,xdSA,ydSA,tmp;
    unsigned char *Z=NULL;
    int Zlen=IDlen+1,buf;
    ecn2 Ppub;
    epoint *dSA;

    h1=mirvar(0);t1=mirvar(0);
    t2=mirvar(0);rem=mirvar(0);tmp=mirvar(0);
    xdSA=mirvar(0);ydSA=mirvar(0);
    dSA=epoint_init();
    Ppub.x.a=mirvar(0);Ppub.x.b=mirvar(0);Ppub.y.a=mirvar(0);Ppub.y.b=mirvar(0);
    Ppub.z.a=mirvar(0);Ppub.z.b=mirvar(0);Ppub.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,ks,t1);//t1=H1(IDA||hid,N)+ks
    xgcd(t1,N,t1,t1,t1);//t1=t1(-1)
    multiply(ks,t1,t2); divide(t2,N,rem);//t2=ks*t1(-1)

    //dSA=[t2]P1
    ecurve_mult(t2,P1,dSA);

    //Ppub=[ks]P2
    ecn2_copy(&P2,&Ppub);
    ecn2_mul(ks,&Ppub);

    printf("\n**********The signed key = (xdA, ydA)：**********\n");
    epoint_get(dSA,xdSA,ydSA);
    cotnum(xdSA,stdout);cotnum(ydSA,stdout);
    printf("\n**********PublicKey Ppubs=[ks]P2：**********\n");
    ecn2_Bytes128_Print(Ppub);

    epoint_get(dSA,xdSA,ydSA);
    big_to_bytes(BNLEN,xdSA,dsa,1);
    big_to_bytes(BNLEN,ydSA,dsa+BNLEN,1);

    redc(Ppub.x.b,tmp);big_to_bytes(BNLEN,tmp,Ppubs,1);
    redc(Ppub.x.a,tmp);big_to_bytes(BNLEN,tmp,Ppubs+BNLEN,1);
    redc(Ppub.y.b,tmp);big_to_bytes(BNLEN,tmp,Ppubs+BNLEN*2,1);
```

```
    redc(Ppub.y.a,tmp);big_to_bytes(BNLEN,tmp,Ppubs+BNLEN*3,1);

    free(Z);
    return 0;
}


/*************************************************************
  Function:       SM9_Sign
  Description:    SM9 signature algorithm
  Calls:          MIRACL functions,zzn12_init(),ecap(),member(),zzn12_ElementPrint(),
                  zzn12_pow(),LinkCharZzn12(),SM9_H2()
  Called By:      SM9_SelfCheck()
  Input:
                  hid:0x01
                  IDA          //identification of userA
                  message      //the message to be signed
                  len          //the length of message
                  rand         //a random number K lies in [1,N-1]
                  dSA          //signature private key
                  Ppubs        //signature public key

  Output:         H,S        //signature result
  Return:         0: success
                  1: asking for memory error
                  4: element is out of order q
                  5: R-ate calculation error
                  9: parameter L error
  Others:
*************************************************************/
int SM9_Sign (unsigned char hid[],unsigned char *IDA,unsigned char *message,int len,unsigned char rand[],
              unsigned char dsa[],unsigned char Ppub[],unsigned char H[],unsigned char S[])
{
     big h1,r,h,l,xdSA,ydSA;
     big xS,yS,tmp,zero;
     zzn12 g,w;
     epoint *s,*dSA;
     ecn2 Ppubs;
     int Zlen,buf;
     unsigned char *Z=NULL;

     //initiate
     h1=mirvar(0);r=mirvar(0);h=mirvar(0);l=mirvar(0);
```

```
tmp=mirvar(0);zero=mirvar(0);
xS=mirvar(0);yS=mirvar(0);
xdSA=mirvar(0);ydSA=mirvar(0);
s=epoint_init();dSA=epoint_init();
Ppubs.x.a=mirvar(0);
 Ppubs.x.b=mirvar(0);
 Ppubs.y.a=mirvar(0);
 Ppubs.y.b=mirvar(0);
Ppubs.z.a=mirvar(0);
 Ppubs.z.b=mirvar(0);
 Ppubs.marker=MR_EPOINT_INFINITY;
zzn12_init(&g);zzn12_init(&w);

bytes_to_big(BNLEN,rand,r);
bytes_to_big(BNLEN,dsa,xdSA);
bytes_to_big(BNLEN,dsa+BNLEN,ydSA);
epoint_set(xdSA,ydSA,0,dSA);
bytes128_to_ecn2(Ppub,&Ppubs);

//Step1:g = e(P1, Ppub-s)
 if(!ecap(Ppubs, P1, 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(P1,Ppubs):**********\n");
zzn12_ElementPrint(g);

//Step2:calculate w=g(r)
printf("\n**********randnum r:**********\n");
cotnum(r,stdout);
w=zzn12_pow(g,r);
printf("\n**********w=gr:**********\n");
zzn12_ElementPrint(w);

//Step3:calculate h=H2(M||w,N)
Zlen=len+32*12;
Z=(char *)malloc(sizeof(char)*(Zlen+1));
if(Z==NULL)
      return SM9_ASK_MEMORY_ERR;

LinkCharZzn12(message,len,w,Z,Zlen);
buf=SM9_H2(Z,Zlen,N,h);
```

```
    if(buf!=0)
        return buf;
    printf("\n****************************h:************************************\n");
    cotnum(h,stdout);

    //Step4:l=(r-h)mod N
    subtract(r,h,l);
     divide(l,N,tmp);
    while(mr_compare(l,zero)<0)
        add(l,N,l);
    if(mr_compare(l,zero)==0)
        return SM9_L_error;
    printf("\n**************************l=(r-h)mod N:*****************************\n");
    cotnum(l,stdout);

    //Step5:S=[l]dSA=(xS,yS)
    ecurve_mult(l,dSA,s);
    epoint_get(s,xS,yS);
    printf("\n**************************S=[l]dSA=(xS,yS):************************\n");
    cotnum(xS,stdout);cotnum(yS,stdout);

    big_to_bytes(32,h,H,1);
    big_to_bytes(32,xS,S,1);
    big_to_bytes(32,yS,S+32,1);

    free(Z);
    return 0;
}
/*****************************************************************
  Function:       SM9_Verify
  Description:    SM9 signature verification algorithm
  Calls:          MIRACL functions,zzn12_init(),Test_Range(),Test_Point(),
                  ecap(),member(),zzn12_ElementPrint(),SM9_H1(),SM9_H2()
  Called By:      SM9_SelfCheck()
  Input:
                  H,S          //signature result used to be verified
                  hid          //identification
                  IDA          //identification of userA
                  message      //the message to be signed
                  len          //the length of message
                  Ppubs        //signature public key

  Output:         NULL
  Return:         0: success
```

```
                    1: asking for memory error
                    2: H is not in the range[1,N-1]
                    6: S is not on the SM9 curve
                    4: element is out of order q
                    5: R-ate calculation error
                    3: h2!=h,comparison error
  Others:
*****************************************************************/
int SM9_Verify (unsigned char H[],unsigned char S[],unsigned char hid[],unsigned char
*IDA,unsigned char *message,int len,
                 unsigned char Ppub[])
{
    big h,xS,yS,h1,h2;
    epoint *S1;
    zzn12 g,t,u,w;
    ecn2 P,Ppubs;
    int Zlen1,Zlen2,buf;
    unsigned char * Z1=NULL,*Z2=NULL;

    h=mirvar(0);
     h1=mirvar(0);
     h2=mirvar(0);
    xS=mirvar(0);
     yS=mirvar(0);
    P.x.a=mirvar(0);
     P.x.b=mirvar(0);
     P.y.a=mirvar(0);
     P.y.b=mirvar(0);
    P.z.a=mirvar(0);
     P.z.b=mirvar(0);
     P.marker=MR_EPOINT_INFINITY;
    Ppubs.x.a=mirvar(0);
     Ppubs.x.b=mirvar(0);
     Ppubs.y.a=mirvar(0);
     Ppubs.y.b=mirvar(0);
    Ppubs.z.a=mirvar(0);
     Ppubs.z.b=mirvar(0);
     Ppubs.marker=MR_EPOINT_INFINITY;
    S1=epoint_init();
    zzn12_init(&g),zzn12_init(&t);
    zzn12_init(&u);zzn12_init(&w);

    bytes_to_big(BNLEN,H,h);
    bytes_to_big(BNLEN,S,xS);
```

```
bytes_to_big(BNLEN,S+BNLEN,yS);
bytes128_to_ecn2(Ppub,&Ppubs);

//Step 1:test if h in the rangge [1,N-1]
if(Test_Range(h))
     return SM9_H_OUTRANGE;

//Step 2:test if S is on G1
epoint_set(xS,yS,0,S1);
if(Test_Point(S1))
     return SM9_S_NOT_VALID_G1;

//Step3:g = e(P1, Ppub-s)
 if(!ecap(Ppubs, P1, 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(P1,Ppubs):**********\n");
zzn12_ElementPrint(g);

//Step4:calculate t=g(h)
t=zzn12_pow(g,h);
printf("\n**********w=gh:**********\n");
zzn12_ElementPrint(t);

 //Step5:calculate h1=H1(IDA||hid,N)
 Zlen1=strlen(IDA)+1;
 Z1=(char *)malloc(sizeof(char)*(Zlen1+1));
 if(Z1==NULL) return SM9_ASK_MEMORY_ERR;

 memcpy(Z1,IDA,strlen(IDA));
 memcpy(Z1+strlen(IDA),hid,1);
 buf=SM9_H1(Z1,Zlen1,N,h1);
 if(buf!=0) return buf;
 printf("\n**********h1:**********\n");
 cotnum(h1,stdout);

 //Step6:P=[h1]P2+Ppubs
 ecn2_copy(&P2,&P);
 ecn2_mul(h1,&P);
 ecn2_add(&Ppubs,&P);
```

```
    //Step7:u=e(S1,P)
    if(!ecap(P, S1, para_t, X, &u)) return SM9_MY_ECAP_12A_ERR;
    //test if a ZZn12 element is of order q
    if(!member(u, para_t, X)) return SM9_MEMBER_ERR;
    printf("\n************************* u=e(S1,P):*****************************\n");
    zzn12_ElementPrint(u);

    //Step8:w=u*t
    zzn12_mul(u,t,&w);
    printf("\n*************************  w=u*t: ********************************\n");
    zzn12_ElementPrint(w);

    //Step9:h2=H2(M||w,N)
    Zlen2=len+32*12;
    Z2=(char *)malloc(sizeof(char)*(Zlen2+1));
    if(Z2==NULL)
        return SM9_ASK_MEMORY_ERR;

    LinkCharZzn12(message,len,w,Z2,Zlen2);
    buf=SM9_H2(Z2,Zlen2,N,h2);
    if(buf!=0) return buf;
    printf("\n*************************** h2:*********************************\n");
    cotnum(h2,stdout);

    free(Z1);
     free(Z2);
    if(mr_compare(h2,h)!=0)
        return SM9_DATA_MEMCMP_ERR;

    return 0;
}

/*************************************************************
  Function:       SM9_SelfCheck
  Description:    SM9 self check
  Calls:          MIRACL functions,SM9_Init(),SM9_GenerateSignKey(),
                  SM9_Sign,SM9_Verify
  Called By:
  Input:
  Output:
  Return:         0: self-check success
                  1: asking for memory error
                  2: H is not in the range[1,N-1]
                  3: h2!=h,comparison error
```

```
                    4: element is out of order q
                    5: R-ate calculation error
                    6: S is not on the SM9 curve
                    7: base point P1 error
                    8: base point P2 error
                    9: parameter L error
                    A: public key generated error
                    B: private key generated error
                    C: signature result error
  Others:
************************************************************/
int SM9_SelfCheck()
{
    //the master private key
    unsigned char dA[32] =
{0x00,0x01,0x30,0xE7,0x84,0x59,0xD7,0x85,0x45,0xCB,0x54,0xC5,0x87,0xE0,0x2C,0xF4,
0x80,0xCE,0x0B,0x66,0x34,0x0F,0x31,0x9F,0x34,0x8A,0x1D,0x5B,0x1F,0x2D,0xC5,0xF4};

    unsigned char
rand[32]={0x00,0x03,0x3C,0x86,0x16,0xB0,0x67,0x04,0x81,0x32,0x03,0xDF,0xD0,0x09,0x65,0x02,
0x2E,0xD1,0x59,0x75,0xC6,0x62,0x33,0x7A,0xED,0x64,0x88,0x35,0xDC,0x4B,0x1C,0xBE};

    unsigned char h[32],S[64];// Signature
    unsigned char Ppub[128],dSA[64];

    unsigned char std_h[32]=
{0x82,0x3C,0x4B,0x21,0xE4,0xBD,0x2D,0xFE,0x1E,0xD9,0x2C,0x60,0x66,0x53,0xE9,0x96,
  0x66,0x85,0x63,0x15,0x2F,0xC3,0x3F,0x55,0xD7,0xBF,0xBB,0x9B,0xD9,0x70,0x5A,0xDB};
    unsigned char std_S[64]=
{0x73,0xBF,0x96,0x92,0x3C,0xE5,0x8B,0x6A,0xD0,0xE1,0x3E,0x96,0x43,0xA4,0x06,0xD8,
  0xEB,0x98,0x41,0x7C,0x50,0xEF,0x1B,0x29,0xCE,0xF9,0xAD,0xB4,0x8B,0x6D,0x59,0x8C,
  0x85,0x67,0x12,0xF1,0xC2,0xE0,0x96,0x8A,0xB7,0x76,0x9F,0x42,0xA9,0x95,0x86,0xAE,
  0xD1,0x39,0xD5,0xB8,0xB3,0xE1,0x58,0x91,0x82,0x7C,0xC2,0xAC,0xED,0x9B,0xAA,0x05};
    unsigned char
std_Ppub[128]={0x9F,0x64,0x08,0x0B,0x30,0x84,0xF7,0x33,0xE4,0x8A,0xFF,0x4B,0x41,0xB5,0x65,0x
01,
     0x1C,0xE0,0x71,0x1C,0x5E,0x39,0x2C,0xFB,0x0A,0xB1,0xB6,0x79,0x1B,0x94,0xC4,0x08,
     0x29,0xDB,0xA1,0x16,0x15,0x2D,0x1F,0x78,0x6C,0xE8,0x43,0xED,0x24,0xA3,0xB5,0x73,
     0x41,0x4D,0x21,0x77,0x38,0x6A,0x92,0xDD,0x8F,0x14,0xD6,0x56,0x96,0xEA,0x5E,0x32,
     0x69,0x85,0x09,0x38,0xAB,0xEA,0x01,0x12,0xB5,0x73,0x29,0xF4,0x47,0xE3,0xA0,0xCB,
     0xAD,0x3E,0x2F,0xDB,0x1A,0x77,0xF3,0x35,0xE8,0x9E,0x14,0x08,0xD0,0xEF,0x1C,0x25,
     0x41,0xE0,0x0A,0x53,0xDD,0xA5,0x32,0xDA,0x1A,0x7C,0xE0,0x27,0xB7,0xA4,0x6F,0x74,
     0x10,0x06,0xE8,0x5F,0x5C,0xDF,0xF0,0x73,0x0E,0x75,0xC0,0x5F,0xB4,0xE3,0x21,0x6D};
    unsigned char
```

```
std_dSA[64]={0xA5,0x70,0x2F,0x05,0xCF,0x13,0x15,0x30,0x5E,0x2D,0x6E,0xB6,0x4B,0x0D,0xEB,0x92
,
   0x3D,0xB1,0xA0,0xBC,0xF0,0xCA,0xFF,0x90,0x52,0x3A,0xC8,0x75,0x4A,0xA6,0x98,0x20,
   0x78,0x55,0x9A,0x84,0x44,0x11,0xF9,0x82,0x5C,0x10,0x9F,0x5E,0xE3,0xF5,0x2D,0x72,
   0x0D,0xD0,0x17,0x85,0x39,0x2A,0x72,0x7B,0xB1,0x55,0x69,0x52,0xB2,0xB0,0x13,0xD3};

    unsigned char hid[]={0x01};
    unsigned char *IDA="Alice";
    unsigned char *message="Chinese IBS standard";//the message to be signed
    int mlen=strlen(message),tmp;//the length of message
    big ks;

    tmp=SM9_Init();

    if(tmp!=0) return tmp;
    ks=mirvar(0);

    bytes_to_big(32,dA,ks);

    printf("\n**********************  SM9 key Generation    **************************\n");
    tmp=SM9_GenerateSignKey(hid,IDA,strlen(IDA),ks,Ppub,dSA);
    if(tmp!=0)
        return tmp;
    if(memcmp(Ppub,std_Ppub,128)!=0)
        return SM9_GEPUB_ERR;
    if(memcmp(dSA,std_dSA,64)!=0)
        return SM9_GEPRI_ERR;

    printf("\n**********************  SM9 signature algorithm**************************\n");
    tmp= SM9_Sign (hid,IDA,message, mlen,rand, dSA, Ppub,h,S);
    if(tmp!=0) return tmp;
    if(memcmp(h,std_h,32)!=0)
        return SM9_SIGN_ERR;
    if(memcmp(S,std_S,64)!=0)
        return SM9_SIGN_ERR;
    printf("\n********************  SM9 verification algorithm ************************\n");
    tmp= SM9_Verify ( h, S, hid,IDA,message, mlen, Ppub);
    if(tmp!=0) return tmp;

    return 0;
}
```