```
/*************************************************************************
  File name:    SM2_sv.c
  Version:      SM2_sv_V1.0
  Date:         Sep 27,2016
  Description:  implementation of SM2 signature algorithm and verification algorithm
  Function List:
        1.SM2_Init                      //initiate SM2 curve
        2.Test_Point                    //test if the given point is on SM2 curve
        3.Test_PubKey                   //test if the given public key is valid
        4.Test_Zero                     //test if the big x equals zero
        5.Test_n                        //test if the big x equals n
        6.Test_Range                    //test if the big x belong to the range[1,n-1]
        7.SM2_KeyGeneration             //generate public key
        8.SM2_Sign                      //SM2 signature algorithm
        9.SM2_Verify                    //SM2 verification
        10.SM2_SelfCheck()              //SM2 slef-check
        11.SM3_256()                    //this function can be found in SM3.c and SM3.h

 Notes:
    This SM2 implementation source code can be used for academic, non-profit making or
non-commercial use only.
    This SM2 implementation is created on MIRACL. SM2 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 "SM2_sv.h"
#include "KDF.h"


/*************************************************************
  Function:       SM2_Init
  Description:    Initiate SM2 curve
  Calls:          MIRACL functions
  Called By:      SM2_KeyGeneration,SM2_Sign,SM2_Verify,SM2_SelfCheck
  Input:          null
  Output:         null
  Return:         0: sucess;
                  1: parameter initialization error;
                  4: the given point G is not a point of order n
  Others:
*************************************************************/
int SM2_Init()
```

```
{
    Gx=mirvar(0);
    Gy=mirvar(0);
    p=mirvar(0);
    a=mirvar(0);
    b=mirvar(0);
    n=mirvar(0);

    bytes_to_big(SM2_NUMWORD,SM2_Gx,Gx);
    bytes_to_big(SM2_NUMWORD,SM2_Gy,Gy);
    bytes_to_big(SM2_NUMWORD,SM2_p,p);
    bytes_to_big(SM2_NUMWORD,SM2_a,a);
    bytes_to_big(SM2_NUMWORD,SM2_b,b);
    bytes_to_big(SM2_NUMWORD,SM2_n,n);

    ecurve_init(a,b,p,MR_PROJECTIVE);
    G=epoint_init();
    nG=epoint_init();

    if (!epoint_set(Gx,Gy,0,G))//initialise point G
    {
        return ERR_ECURVE_INIT;
    }
    ecurve_mult(n,G,nG);
    if (!point_at_infinity(nG))    //test if the order of the point is n
    {
        return ERR_ORDER;
    }

    return 0;

}


/*************************************************************
  Function:       Test_Point
  Description:    test if the given point is on SM2 curve
  Calls:
  Called By:      SM2_KeyGeneration
  Input:          point
  Output:         null
  Return:         0: sucess
                  3: not a valid point on curve
```

```
  Others:
*************************************************************/
int Test_Point(epoint* point)
{
    big x,y,x_3,tmp;
    x=mirvar(0);
    y=mirvar(0);
    x_3=mirvar(0);
    tmp=mirvar(0);


    //test if y^2=x^3+ax+b
    epoint_get(point,x,y);
    power (x, 3, p, x_3);        //x_3=x^3 mod p
    multiply (x, a,x);           //x=a*x
    divide (x, p, tmp);          //x=a*x mod p  , tmp=a*x/p
    add(x_3,x,x);                     //x=x^3+ax
    add(x,b,x);                  //x=x^3+ax+b
    divide(x,p,tmp);             //x=x^3+ax+b mod p
    power (y, 2, p, y);          //y=y^2 mod p
    if(compare(x,y)!=0)
        return ERR_NOT_VALID_POINT;
    else
        return 0;
}


/*************************************************************
  Function:       Test_PubKey
  Description:    test if the given public key is valid
  Calls:
  Called By:      SM2_KeyGeneration
  Input:          pubKey    //a point
  Output:         null
  Return:         0: sucess
                  2: a point at infinity
                  5: X or Y coordinate is beyond Fq
                  3: not a valid point on curve
                  4: not a point of order n
  Others:
*************************************************************/
int Test_PubKey(epoint *pubKey)
{
    big x,y,x_3,tmp;
```

```
    epoint *nP;
    x=mirvar(0);
    y=mirvar(0);
    x_3=mirvar(0);
    tmp=mirvar(0);

    nP=epoint_init();

    //test if the pubKey is the point at infinity
    if (point_at_infinity(pubKey))// if pubKey is point at infinity, return error;
        return ERR_INFINITY_POINT;

    //test if x<p   and  y<p  both hold
    epoint_get(pubKey,x,y);
    if((compare(x,p)!=-1)  || (compare(y,p)!=-1))
        return ERR_NOT_VALID_ELEMENT;

    if(Test_Point(pubKey)!=0)
        return ERR_NOT_VALID_POINT;

    //test if the order of pubKey is equal to n
    ecurve_mult(n,pubKey,nP);        // nP=[n]P
    if (!point_at_infinity(nP))           // if np is point NOT at infinity, return error;
        return ERR_ORDER;
    return 0;
}


/*************************************************************
  Function:       Test_Zero
  Description:    test if the big x is zero
  Calls:
  Called By:      SM2_Sign
  Input:          pubKey    //a point
  Output:         null
  Return:         0: x!=0
                  1: x==0
  Others:
*************************************************************/
int Test_Zero(big x)
{
    big zero;
    zero=mirvar(0);
    if(compare(x,zero)==0)
```

```
        return 1;
    else return 0;

}


/*************************************************************
  Function:       Test_n
  Description:    test if the big x is order n
  Calls:
  Called By:      SM2_Sign
  Input:          big x    //a miracl data type
  Output:         null
  Return:         0: sucess
                  1: x==n,fail
  Others:
*************************************************************/
int Test_n(big x)
{
    //   bytes_to_big(32,SM2_n,n);
    if(compare(x,n)==0)
        return 1;
    else return 0;
}


/*************************************************************
  Function:       Test_Range
  Description:    test if the big x belong to the range[1,n-1]
  Calls:
  Called By:      SM2_Verify
  Input:          big x    ///a miracl data type
  Output:         null
  Return:         0: sucess
                  1: 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( (compare(x,one) < 0)| (compare(x,decr_n)>0) )
        return 1;
    return 0;
}


/*************************************************************
  Function:       SM2_KeyGeneration
  Description:    calculate a pubKey out of a given priKey
  Calls:          SM2_SelfCheck()
  Called By:      SM2_Init()
  Input:          priKey       // a big number lies in[1,n-2]
  Output:         pubKey       // pubKey=[priKey]G
  Return:         0: sucess
                  2: a point at infinity
                  5: X or Y coordinate is beyond Fq
                  3: not a valid point on curve
                  4: not a point of order n
  Others:
*************************************************************/
int SM2_KeyGeneration(unsigned char PriKey[],unsigned char Px[],unsigned char Py[])
{
    int i=0;
    big d,PAx,PAy;
    epoint *PA;

    SM2_Init();
    PA=epoint_init();

    d=mirvar(0);
    PAx=mirvar(0);
    PAy=mirvar(0);

    bytes_to_big(SM2_NUMWORD,PriKey,d);

    ecurve_mult(d,G,PA);
    epoint_get(PA,PAx,PAy);


    big_to_bytes(SM2_NUMWORD,PAx,Px,TRUE);
    big_to_bytes(SM2_NUMWORD,PAy,Py,TRUE);
```

```
    i=Test_PubKey(PA);
    if(i)
        return i;
    else
        return 0;

}


/*************************************************************
  Function:       SM2_Sign
  Description:    SM2 signature algorithm
  Calls:          SM2_Init(),Test_Zero(),Test_n(), SM3_256()
  Called By:      SM2_SelfCheck()
  Input:          message    //the message to be signed
                  len        //the length of message
                  ZA         // ZA=Hash(ENTLA|| IDA|| a|| b|| Gx || Gy || xA|| yA)
                  rand       //a random number K lies in [1,n-1]
                  d          //the private key

  Output:         R,S        //signature result
  Return:         0: sucess
                  1: parameter initialization error;
                  4: the given point G is not a point of order n
                  6: the signed r equals 0 or r+rand equals n
                  7  the signed s equals 0
  Others:
*************************************************************/
int SM2_Sign(unsigned char *message,int len,unsigned char ZA[],unsigned char rand[],unsigned
char d[],unsigned char R[],unsigned char S[])
{
    unsigned char hash[SM3_len/8];
    int M_len=len+SM3_len/8;
    unsigned char *M=NULL;
    int i;

    big dA,r,s,e,k,KGx,KGy;
    big rem,rk,z1,z2;
    epoint *KG;

    i=SM2_Init();
    if(i)  return i;
```

```
//initiate
dA=mirvar(0);
e=mirvar(0);
k=mirvar(0);
KGx=mirvar(0);
KGy=mirvar(0);
r=mirvar(0);
s=mirvar(0);
rem=mirvar(0);
rk=mirvar(0);
z1=mirvar(0);
z2=mirvar(0);

bytes_to_big(SM2_NUMWORD,d,dA);//cinstr(dA,d);

KG=epoint_init();

//step1,set M=ZA||M
M=(char *)malloc(sizeof(char)*(M_len+1));
memcpy(M,ZA,SM3_len/8);
memcpy(M+SM3_len/8,message,len);

//step2,generate e=H(M)
SM3_256(M, M_len, hash);
bytes_to_big(SM3_len/8,hash,e);

//step3:generate k
bytes_to_big(SM3_len/8,rand,k);

//step4:calculate kG
ecurve_mult(k,G,KG);

//step5:calculate r
epoint_get(KG,KGx,KGy);
add(e,KGx,r);
divide(r,n,rem);

//judge r=0 or n+k=n?
add(r,k,rk);
if( Test_Zero(r) | Test_n(rk))
    return ERR_GENERATE_R;

//step6:generate s
incr(dA,1,z1);
```

```
    xgcd(z1,n,z1,z1,z1);
    multiply(r,dA,z2);
    divide(z2,n,rem);
    subtract(k,z2,z2);
    add(z2,n,z2);
    multiply(z1,z2,s);
    divide(s,n,rem);

    //judge s=0?
    if(Test_Zero(s))
        return ERR_GENERATE_S ;

    big_to_bytes(SM2_NUMWORD,r,R,TRUE);
    big_to_bytes(SM2_NUMWORD,s,S,TRUE);

    free(M);
    return 0;
}


/*************************************************************
  Function:       SM2_Verify
  Description:    SM2 verification algorithm
  Calls:          SM2_Init(),Test_Range(), Test_Zero(),SM3_256()
  Called By:      SM2_SelfCheck()
  Input:          message    //the message to be signed
                  len        //the length of message
                  ZA         //ZA=Hash(ENTLA|| IDA|| a|| b|| Gx || Gy || xA|| yA)
                  Px,Py      //the public key
                  R,S        //signature result

  Output:
  Return:         0: sucess
                  1: parameter initialization error;
                  4: the given point G is not a point of order n
                  B: public key error
                  8: the signed R out of range [1,n-1]
                  9: the signed S out of range [1,n-1]
                  A: the intermediate data t equals 0
                  C: verification fail
  Others:
*************************************************************/
int SM2_Verify(unsigned char *message,int len,unsigned char ZA[],unsigned char Px[],unsigned
char Py[],unsigned char R[],unsigned char S[])
```

```
{
    unsigned char hash[SM3_len/8];
    int M_len=len+SM3_len/8;
    unsigned char *M=NULL;
    int i;

    big PAx,PAy,r,s,e,t,rem,x1,y1;
    big RR;
    epoint *PA,*sG,*tPA;

    i=SM2_Init();
    if(i) return i;

    PAx=mirvar(0);
    PAy=mirvar(0);
    r=mirvar(0);
    s=mirvar(0);
    e=mirvar(0);
    t=mirvar(0);
    x1=mirvar(0);
    y1=mirvar(0);
    rem=mirvar(0);
    RR=mirvar(0);

    PA=epoint_init();
    sG=epoint_init();
    tPA=epoint_init();

    bytes_to_big(SM2_NUMWORD,Px,PAx);
    bytes_to_big(SM2_NUMWORD,Py,PAy);

    bytes_to_big(SM2_NUMWORD,R,r);
    bytes_to_big(SM2_NUMWORD,S,s);

    if (!epoint_set(PAx,PAy,0,PA))//initialise public key
    {
        return ERR_PUBKEY_INIT;
    }

     //step1: test if r belong to [1,n-1]
     if (Test_Range(r))
         return ERR_OUTRANGE_R;

     //step2: test if s belong to [1,n-1]
```

```
    if (Test_Range(s))
        return ERR_OUTRANGE_S;

    //step3,generate M
    M=(char *)malloc(sizeof(char)*(M_len+1));
    memcpy(M,ZA,SM3_len/8);
    memcpy(M+SM3_len/8,message,len);

    //step4,generate e=H(M)
    SM3_256(M, M_len, hash);
    bytes_to_big(SM3_len/8,hash,e);

    //step5:generate t
    add(r,s,t);
    divide(t,n,rem);

    if( Test_Zero(t))
        return ERR_GENERATE_T;

    //step 6: generate(x1,y1)
    ecurve_mult(s,G,sG);
    ecurve_mult(t,PA,tPA);
    ecurve_add(sG,tPA);
    epoint_get(tPA,x1,y1);

    //step7:generate RR
    add(e,x1,RR);
    divide(RR,n,rem);

    free(M);
    if(compare(RR,r)==0)
        return 0;
    else
        return ERR_DATA_MEMCMP;
}


/*****************************************************************
  Function:       SM2_SelfCheck
  Description:    SM2 self check
  Calls:          SM2_Init(), SM2_KeyGeneration,SM2_Sign, SM2_Verify,SM3_256()
  Called By:
  Input:
  Output:
```

```
  Return:         0: sucess
                  1: paremeter initialization error
                  2: a point at infinity
                  5: X or Y coordinate is beyond Fq
                  3: not a valid point on curve
                  4: not a point of order n
                  B: public key error
                  8: the signed R out of range [1,n-1]
                  9: the signed S out of range [1,n-1]
                  A: the intermediate data t equals 0
                  C: verification fail
  Others:
*************************************************************/
int SM2_SelfCheck()
{
    //the private key
    unsigned char
dA[32]={0x39,0x45,0x20,0x8f,0x7b,0x21,0x44,0xb1,0x3f,0x36,0xe3,0x8a,0xc6,0xd3,0x9f,
0x95,0x88,0x93,0x93,0x69,0x28,0x60,0xb5,0x1a,0x42,0xfb,0x81,0xef,0x4d,0xf7,0xc5,0xb8};

    unsigned char
rand[32]={0x59,0x27,0x6E,0x27,0xD5,0x06,0x86,0x1A,0x16,0x68,0x0F,0x3A,0xD9,0xC0,0x2D,
0xCC,0xEF,0x3C,0xC1,0xFA,0x3C,0xDB,0xE4,0xCE,0x6D,0x54,0xB8,0x0D,0xEA,0xC1,0xBC,0x21};

    //the public key
   /* unsigned char
xA[32]={0x09,0xf9,0xdf,0x31,0x1e,0x54,0x21,0xa1,0x50,0xdd,0x7d,0x16,0x1e,0x4b,0xc5,
0xc6,0x72,0x17,0x9f,0xad,0x18,0x33,0xfc,0x07,0x6b,0xb0,0x8f,0xf3,0x56,0xf3,0x50,0x20};
    unsigned char
yA[32]={0xcc,0xea,0x49,0x0c,0xe2,0x67,0x75,0xa5,0x2d,0xc6,0xea,0x71,0x8c,0xc1,0xaa,
0x60,0x0a,0xed,0x05,0xfb,0xf3,0x5e,0x08,0x4a,0x66,0x32,0xf6,0x07,0x2d,0xa9,0xad,0x13};*/

    unsigned char xA[32],yA[32];
    unsigned char r[32],s[32];// Signature

    unsigned char IDA[16]={0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x31,0x32,0x33,
        0x34,0x35,0x36,0x37,0x38};//ASCII code of userA's identification
    int IDA_len=16;
    unsigned char ENTLA[2]={0x00,0x80};//the length of userA's identification,presentation in
ASCII code

    unsigned char *message="message digest";//the message to be signed
    int len=strlen(message);//the length of message
```

```
    unsigned char ZA[SM3_len/8];//ZA=Hash(ENTLA|| IDA|| a|| b|| Gx || Gy || xA|| yA)
    unsigned char Msg[210]; //210=IDA_len+2+SM2_NUMWORD*6

    int temp;

    miracl *mip=mirsys(10000,16);
    mip->IOBASE=16;

    temp=SM2_KeyGeneration(dA,xA,yA);
    if(temp)
        return temp;

    // ENTLA|| IDA|| a|| b|| Gx || Gy || xA|| yA
    memcpy(Msg,ENTLA,2);
    memcpy(Msg+2,IDA,IDA_len);
    memcpy(Msg+2+IDA_len,SM2_a,SM2_NUMWORD);
    memcpy(Msg+2+IDA_len+SM2_NUMWORD,SM2_b,SM2_NUMWORD);
    memcpy(Msg+2+IDA_len+SM2_NUMWORD*2,SM2_Gx,SM2_NUMWORD);
    memcpy(Msg+2+IDA_len+SM2_NUMWORD*3,SM2_Gy,SM2_NUMWORD);
    memcpy(Msg+2+IDA_len+SM2_NUMWORD*4,xA,SM2_NUMWORD);
    memcpy(Msg+2+IDA_len+SM2_NUMWORD*5,yA,SM2_NUMWORD);
    SM3_256(Msg,210,ZA);

    temp=SM2_Sign(message,len,ZA,rand,dA,r,s);
    if(temp)
        return temp;

    temp=SM2_Verify(message,len,ZA,xA,yA,r,s);
    if(temp)
        return temp;

    return 0;
}
```