```
/*************************************************************************
  File name:    zzn12_operation.h
  Version:
  Date:         Dec 15,2016
  Description:  this code is achieved according to zzn12a.h and zzn12a.cpp in MIRCAL C++ source
file writen by M. Scott.
                so,see zzn12a.h and zzn12a.cpp for details.
                this code define one struct zzn12,and based on it give many fuctions.
  Function List:
        1.zzn12_init           //Initiate struct zzn12
        2.zzn12_copy           //copy one zzn12 to another
        3.zzn12_mul            //z=x*y,achieve multiplication with two zzn12
        4.zzn12_conj           //achieve conjugate complex
        5.zzn12_inverse        //element inversion
        6.zzn12_powq           //
        7.zzn12_div            //division operation
        8.zzn12_pow            //regular zzn12 powering

 Notes:
*************************************************************************/
miracl* mip;
zzn2 X; //Frobniues constant

typedef struct
{
    zzn4 a,b,c;
    BOOL unitary;// "unitary property means that fast squaring can be used, and inversions are
just conjugates
    BOOL miller; // "miller" property means that arithmetic on this instance can ignore
multiplications
                 // or divisions by constants - as instance will eventually be raised to (p-1).
} zzn12;


/*************************************************************
  Function:       zzn12_init
  Description:    Initiate struct zzn12
  Calls:          MIRACL functions
  Called By:
  Input:          zzn12 *x
  Output:         null
  Return:         null
  Others:
*************************************************************/
```

```
void zzn12_init(zzn12 *x)
{
    x->a.a.a=mirvar(0);x->a.a.b=mirvar(0);
    x->a.b.a=mirvar(0);x->a.b.b=mirvar(0);x->a.unitary=FALSE;
    x->b.a.a=mirvar(0);x->b.a.b=mirvar(0);
    x->b.b.a=mirvar(0);x->b.b.b=mirvar(0);x->b.unitary=FALSE;
    x->c.a.a=mirvar(0);x->c.a.b=mirvar(0);
    x->c.b.a=mirvar(0);x->c.b.b=mirvar(0);x->c.unitary=FALSE;
    x->miller=FALSE;x->unitary=FALSE;
}

/*****************************************************************
  Function:       zzn12_copy
  Description:    copy y=x
  Calls:          MIRACL functions
  Called By:
  Input:          zzn12 *x
  Output:         zzn12 *y
  Return:         null
  Others:
*****************************************************************/
void zzn12_copy(zzn12 *x,zzn12 *y)
{
    zzn4_copy(&x->a,&y->a);zzn4_copy(&x->b,&y->b);zzn4_copy(&x->c,&y->c);
    y->miller=x->miller;y->unitary=x->unitary;
}

/*****************************************************************
  Function:       zzn12_mul
  Description:    z=x*y,see zzn12a.h and zzn12a.cpp for details in MIRACL c++ source file
  Calls:          MIRACL functions
  Called By:
  Input:          zzn12 x,y
  Output:         zzn12 *z
  Return:         null
  Others:
*****************************************************************/
zzn12_mul(zzn12 x,zzn12 y,zzn12 *z)
{
   // Karatsuba
    zzn4 Z0,Z1,Z2,Z3,T0,T1;
     BOOL zero_c,zero_b;

    Z0.a.a=mirvar(0);Z0.a.b=mirvar(0);Z0.b.a=mirvar(0);Z0.b.b=mirvar(0);Z0.unitary=FALSE;
```

```
    Z1.a.a=mirvar(0);Z1.a.b=mirvar(0);Z1.b.a=mirvar(0);Z1.b.b=mirvar(0);Z1.unitary=FALSE;
    Z2.a.a=mirvar(0);Z2.a.b=mirvar(0);Z2.b.a=mirvar(0);Z2.b.b=mirvar(0);Z2.unitary=FALSE;
    Z3.a.a=mirvar(0);Z3.a.b=mirvar(0);Z3.b.a=mirvar(0);Z3.b.b=mirvar(0);Z3.unitary=FALSE;
    T0.a.a=mirvar(0);T0.a.b=mirvar(0);T0.b.a=mirvar(0);T0.b.b=mirvar(0);T0.unitary=FALSE;
    T1.a.a=mirvar(0);T1.a.b=mirvar(0);T1.b.a=mirvar(0);T1.b.b=mirvar(0);T1.unitary=FALSE;

    zzn12_copy(&x,z);
    if( zzn4_compare(&x.a,&y.a) && zzn4_compare(&x.a,&y.a)&&zzn4_compare(&x.a,&y.a))
    {
        if (x.unitary==TRUE)
        {

zzn4_copy(&x.a,&Z0);zzn4_mul(&x.a,&x.a,&z->a);zzn4_copy(&z->a,&Z3);zzn4_add(&z->a,&z->a,&z->
a);
          zzn4_add(&z->a,&Z3,&z->a);zzn4_conj(&Z0,&Z0);
zzn4_add(&Z0,&Z0,&Z0);zzn4_sub(&z->a,&Z0,&z->a);
          zzn4_copy(&x.c,&Z1);zzn4_mul(&Z1,&Z1,&Z1);zzn4_tx(&Z1);
          zzn4_copy(&Z1,&Z3);zzn4_add(&Z1,&Z1,&Z1);zzn4_add(&Z1,&Z3,&Z1);
          zzn4_copy(&x.b,&Z2);zzn4_mul(&Z2,&Z2,&Z2);
          zzn4_copy(&Z2,&Z3);zzn4_add(&Z2,&Z2,&Z2);zzn4_add(&Z2,&Z3,&Z2);

          zzn4_conj(&x.b,&z->b);zzn4_add(&z->b,&z->b,&z->b);
          zzn4_conj(&x.c,&z->c);zzn4_add(&z->c,&z->c,&z->c);zzn4_negate(&z->c,&z->c);
          zzn4_add(&z->b,&Z1,&z->b);zzn4_add(&z->c,&Z2,&z->c);
        }
        else
        {
            if(!x.miller)
            {// Chung-Hasan SQR2
              zzn4_copy(&x.a,&Z0);zzn4_mul(&Z0,&Z0,&Z0);
              zzn4_mul(&x.b,&x.c,&Z1);zzn4_add(&Z1,&Z1,&Z1);
              zzn4_copy(&x.c,&Z2);zzn4_mul(&Z2,&Z2,&Z2);
              zzn4_mul(&x.a,&x.b,&Z3);zzn4_add(&Z3,&Z3,&Z3);

zzn4_add(&x.a,&x.b,&z->c);zzn4_add(&z->c,&x.c,&z->c);zzn4_mul(&z->c,&z->c,&z->c);

              zzn4_tx(&Z1);zzn4_add(&Z0,&Z1,&z->a);
              zzn4_tx(&Z2);zzn4_add(&Z3,&Z2,&z->b);
              zzn4_add(&Z0,&Z1,&T0);zzn4_add(&T0,&Z2,&T0);
              zzn4_add(&T0,&Z3,&T0);zzn4_sub(&z->c,&T0,&z->c);
            }
            else
            {// Chung-Hasan SQR3 - actually calculate 2x^2 !
             // Slightly dangerous - but works as will be raised to p^{k/2}-1
```

```
            // which wipes out the 2.
            zzn4_copy(&x.a,&Z0);zzn4_mul(&Z0,&Z0,&Z0);// a0^2     = S0
            zzn4_copy(&x.c,&Z2);zzn4_mul(&Z2,&x.b,&Z2);zzn4_add(&Z2,&Z2,&Z2); // 2a1.a2  =
S3
              zzn4_copy(&x.c,&Z3);zzn4_mul(&Z3,&Z3,&Z3);;        // a2^2     = S4
            zzn4_add(&x.c,&x.a,&z->c);             // a0+a2

            zzn4_copy(&x.b,&Z1);zzn4_add(&Z1,&z->c,&Z1);zzn4_mul(&Z1,&Z1,&Z1);//
(a0+a1+a2)^2  =S1
            zzn4_sub(&z->c,&x.b,&z->c);zzn4_mul(&z->c,&z->c,&z->c);// (a0-a1+a2)^2  =S2
            zzn4_add(&Z2,&Z2,&Z2);zzn4_add(&Z0,&Z0,&Z0);zzn4_add(&Z3,&Z3,&Z3);

            zzn4_sub(&Z1,&z->c,&T0);zzn4_sub(&T0,&Z2,&T0);
            zzn4_sub(&Z1,&Z0,&T1);zzn4_sub(&T1,&Z3,&T1);zzn4_add(&z->c,&T1,&z->c);
            zzn4_tx(&Z3);zzn4_add(&T0,&Z3,&z->b);
            zzn4_tx(&Z2);zzn4_add(&Z0,&Z2,&z->a);
           }
        }
    }
    else
    {
    // Karatsuba
    zero_b=zzn4_iszero(&y.b);
    zero_c=zzn4_iszero(&y.c);

    zzn4_mul(&x.a,&y.a,&Z0);  //9
    if (!zero_b)  zzn4_mul(&x.b,&y.b,&Z2);  //+6

    zzn4_add(&x.a,&x.b,&T0);
    zzn4_add(&y.a,&y.b,&T1);
    zzn4_mul(&T0,&T1,&Z1); //+9
    zzn4_sub(&Z1,&Z0,&Z1);
    if (!zero_b) zzn4_sub(&Z1,&Z2,&Z1);

    zzn4_add(&x.b,&x.c,&T0);
    zzn4_add(&y.b,&y.c,&T1);
    zzn4_mul(&T0,&T1,&Z3);//+6
    if (!zero_b) zzn4_sub(&Z3,&Z2,&Z3);

    zzn4_add(&x.a,&x.c,&T0);
    zzn4_add(&y.a,&y.c,&T1);
    zzn4_mul(&T0,&T1,&T0);//+9=39 for "special case"
    if (!zero_b) zzn4_add(&Z2,&T0,&Z2);
     else        zzn4_copy(&T0,&Z2);
```

```
    zzn4_sub(&Z2,&Z0,&Z2);

    zzn4_copy(&Z1,&z->b);
     if (!zero_c)
     { // exploit special form of BN curve line function
        zzn4_mul(&x.c,&y.c,&T0);
        zzn4_sub(&Z2,&T0,&Z2);
        zzn4_sub(&Z3,&T0,&Z3);zzn4_tx(&T0);
        zzn4_add(&z->b,&T0,&z->b);
     }

    zzn4_tx(&Z3);
    zzn4_add(&Z0,&Z3,&z->a);
    zzn4_copy(&Z2,&z->c);
    if (!y.unitary) z->unitary=FALSE;
    }
}

/*************************************************************
  Function:       zzn12_conj
  Description:    achieve conjugate complex
                  see zzn12a.h and zzn1212.cpp for details in MIRACL c++ source file
  Calls:          MIRACL functions
  Called By:
  Input:          zzn12 x,y
  Output:         zzn12 *z
  Return:         null
  Others:
*************************************************************/
void zzn12_conj(zzn12 *x,zzn12 *y)
{
    zzn4_conj(&x->a,&y->a);
    zzn4_conj(&x->b,&y->b);
    zzn4_negate(&y->b,&y->b);
    zzn4_conj(&x->c,&y->c);
    y->miller=x->miller;y->unitary=x->unitary;
}
/*************************************************************
  Function:       zzn12_inverse
  Description:    element inversion,
                  see zzn12a.h and zzn1212.cpp for details in MIRACL c++ source file
  Calls:          MIRACL functions,zzn12_init,zzn12_conj
  Called By:
  Input:          zzn12 w
```

```
  Output:
  Return:          zzn12
  Others:
*************************************************************/
zzn12 zzn12_inverse(zzn12 w)
{
    zzn4 tmp1,tmp2;
    zzn12 res;

    tmp1.a.a=mirvar(0);tmp1.a.b=mirvar(0);
    tmp1.b.a=mirvar(0);tmp1.b.b=mirvar(0);tmp1.unitary=FALSE;
    tmp2.a.a=mirvar(0);tmp2.a.b=mirvar(0);
    tmp2.b.a=mirvar(0);tmp2.b.b=mirvar(0);tmp2.unitary=FALSE;
    zzn12_init(&res);

    if(w.unitary)
    {
        zzn12_conj(&w,&res);
        return res;
    }
    //res.a=w.a*w.a-tx(w.b*w.c);
    zzn4_mul(&w.a,&w.a,&res.a);
    zzn4_mul(&w.b,&w.c,&res.b);zzn4_tx(&res.b);
    zzn4_sub(&res.a,&res.b,&res.a);

    //res.b=tx(w.c*w.c)-w.a*w.b;
    zzn4_mul(&w.c,&w.c,&res.c);zzn4_tx(&res.c);
    zzn4_mul(&w.a,&w.b,&res.b);zzn4_sub(&res.c,&res.b,&res.b);

    //res.c=w.b*w.b-w.a*w.c;
    zzn4_mul(&w.b,&w.b,&res.c);zzn4_mul(&w.a,&w.c,&tmp1);zzn4_sub(&res.c,&tmp1,&res.c);

    //tmp1=tx(w.b*res.c)+w.a*res.a+tx(w.c*res.b);
    zzn4_mul(&w.b,&res.c,&tmp1);zzn4_tx(&tmp1);
    zzn4_mul(&w.a,&res.a,&tmp2);zzn4_add(&tmp1,&tmp2,&tmp1);
    zzn4_mul(&w.c,&res.b,&tmp2);zzn4_tx(&tmp2);zzn4_add(&tmp1,&tmp2,&tmp1);

    zzn4_inv(&tmp1);
    zzn4_mul(&res.a,&tmp1,&res.a);
    zzn4_mul(&res.b,&tmp1,&res.b);
    zzn4_mul(&res.c,&tmp1,&res.c);

    return res;
}
```

```
/*************************************************************
  Function:       zzn12_powq
  Description:    Frobenius F=x^p. Assumes p=1 mod 6
                  see zzn12a.h and zzn1212.cpp for details in MIRACL c++ source file
  Calls:          MIRACL functions
  Called By:
  Input:          zzn2 F
  Output:         zzn12 *y
  Return:         NULL
  Others:
*************************************************************/
void zzn12_powq(zzn2 F,zzn12 *y)
{
    zzn2 X2,X3;
    X2.a=mirvar(0);X2.b=mirvar(0);
    X3.a=mirvar(0);X3.b=mirvar(0);
    zzn2_mul(&F,&F,&X2);
    zzn2_mul(&X2,&F,&X3);

    zzn4_powq(&X3,&y->a);zzn4_powq(&X3,&y->b);zzn4_powq(&X3,&y->c);
    zzn4_smul(&y->b,&X,&y->b);
    zzn4_smul(&y->c,&X2,&y->c);
}
/*************************************************************
  Function:       zzn12_div
  Description:    z=x/y
                  see zzn12a.h and zzn1212.cpp for details in MIRACL c++ source file
  Calls:          MIRACL functions,zzn12_inverse,zzn12_mul
  Called By:
  Input:          zzn12 x,y
  Output:         zzn12 *z
  Return:         NULL
  Others:
*************************************************************/
void zzn12_div(zzn12 x,zzn12 y,zzn12 *z)
{
    y=zzn12_inverse(y);
    zzn12_mul(x,y,z);
}

/*************************************************************
  Function:       zzn12_pow
  Description:    regular zzn12 powering,If k is low Hamming weight this will be just as good.
                  see zzn12a.h and zzn1212.cpp for details in MIRACL c++ source file
```

```
  Calls:          MIRACL functions,zzn12_inverse,zzn12_mul,zzn12_copy,zzn12_init
  Called By:
  Input:          zzn12 x,big k
  Output:
  Return:         zzn12
  Others:
*************************************************************/
zzn12 zzn12_pow(zzn12 x,big k)
{
    big zero,tmp,tmp1;
    int nb,i;
    BOOL invert_it;
    zzn12 res;

    zero=mirvar(0);tmp=mirvar(0);tmp1=mirvar(0);
    zzn12_init(&res);
    copy(k,tmp1);
    invert_it=FALSE;

    if(mr_compare(tmp1,zero)==0)
    {
        tmp=get_mip()->one;
        zzn4_from_big(tmp,&res.a);
        return res;
    }
    if(mr_compare(tmp1,zero)<0)
    {
        negify(tmp1,tmp1);invert_it=TRUE;
    }
    nb=logb2(k);
    zzn12_copy(&x,&res);
    if (nb>1) for (i=nb-2;i>=0;i--)
    {
        zzn12_mul(res,res,&res);
        if (mr_testbit(k,i)) zzn12_mul(res,x,&res);
    }
    if (invert_it) res=zzn12_inverse(res);

    return res;
}
```