

```

/*****
File name:    R-ate.h
Version:
Date:        Dec 15, 2016
Description:  this code is achieved according to akel2bnx.cpp in MIRCAL C++ source file.
              so, see akel2bnx.cpp for details.
              this code gives calculation of R-ate pairing
Function List:
    1. zzn2_pow          //regular zzn2 powering
    2. set_frobenius_constant //calculate frobenius_constant X
    3. q_power_frobenius
    4. line
    5. g
    6. fast_pairing
    7. ecap

```

Notes:

```

*****/

```

```

#include "zzn12_operation.h"

```

```

/*****
Function:      zzn2_pow
Description:   regular zzn2 powering
              see zzn2.cpp for details in MIRACL c++ source file
Calls:        MIRACL functions
Called By:     set_frobenius_constant
Input:        zzn2 x, big k
Output:       null
Return:       zzn2
Others:

```

```

*****/

```

```

zzn2 zzn2_pow(zzn2 x, big k)
{
    int i, j, nb, n, nbw, nzs;
    big zero;
    zzn2 res, u2, t[16];

    zero=mirvar(0);
    res.a=mirvar(0);res.b=mirvar(0);
    u2.a=mirvar(0);u2.b=mirvar(0);

    if (zzn2_iszero(&x))
        {zzn2_zero(&res); return res;}

```

```

    if ( size(k)==0 )
    { zzn2_from_int(1,&res);return res;}
    if (size(k)==1) return x;

// Prepare table for windowing
    zzn2_mul (&x,&x,&u2);
    t[0].a=mirvar(0);
    t[0].b=mirvar(0);
    zzn2_copy (&x,&t[0]);
    for(i=1;i<16;i++)
    {   t[i].a=mirvar(0);
        t[i].b=mirvar(0);
        zzn2_mul (&t[i-1],&u2,&t[i]);
    }

// Left to right method - with windows
    zzn2_copy (&x,&res);
    nb=logb2(k);
    if (nb>1) for (i=nb-2;i>=0;)
    {
        //Note new parameter of window_size=5. Default to 5, but reduce to 4 (or even 3) to save
RAM
        n=mr_window(k, i, &nbw, &nzs, 5);
        for(j=0;j<nbw;j++) zzn2_mul (&res,&res,&res);
        if (n>0) zzn2_mul (&res,&t[n/2],&res);
        i-=nbw;
        if (nzs)
        {
            for (j=0;j<nzs;j++) zzn2_mul (&res,&res,&res);
            i-=nzs;
        }
    }
    return res;
}

```

/\*\*\*\*\*

Function:	set_frobenius_constant
Description:	calculate frobenius_constant X see akel2bnx.cpp for details in MIRACL c++ source file
Calls:	MIRACL functions, zzn2_pow
Called By:	SM9_init
Input:	NULL
Output:	zzn2 *X
Return:	NULL
Others:	

\*\*\*\*\*/

```
void set_frobenius_constant(zzn2 *X)
{
    big p, zero, one, two;
    p=mirvar(0);
    zero=mirvar(0);
    one=mirvar(0);
    two=mirvar(0);

    convert(0, zero);
    convert(1, one);
    convert(2, two);

    mip=get_mip();
    copy(mip->modulus, p);

    switch (get_mip()->pmod8)
    {
    case 5:
        zzn2_from_bigs(zero, one, X); // = (sqrt(-2))^(p-1)/2
        break;
    case 3:
        // = (1+sqrt(-1))^(p-1)/2
        zzn2_from_bigs(one, one, X);
        break;
    case 7:
        zzn2_from_bigs(two, one, X); // = (2+sqrt(-1))^(p-1)/2
    default: break;
    }

    decr(p, 1, p);
    subdiv(p, 6, p);

    *X=zzn2_pow(*X, p);
}
```

/\*\*\*\*\*/

Function: q\_power\_frobenius  
 Description: F is frobenius\_constant X  
 see akel2bnx.cpp for details in MIRACL c++ source file  
 Calls: MIRACL functions  
 Called By: fast\_pairing  
 Input: ecn2 A, zzn2 F  
 Output: zzn2 A  
 Return: NULL  
 Others:

```

*****/
void q_power_frobenius(ecn2 A, zzn2 F)
{
    // Fast multiplication of A by q (for Trace-Zero group members only)
    zzn2 x, y, z, w, r;
    x.a=mirvar(0);x.b=mirvar(0);
    y.a=mirvar(0);y.b=mirvar(0);
    z.a=mirvar(0);z.b=mirvar(0);
    w.a=mirvar(0);w.b=mirvar(0);
    r.a=mirvar(0);r.b=mirvar(0);

    ecn2_get(&A, &x, &y, &z);
    zzn2_copy(&F, &r); //r=F
    if (get_mip()->TWIST==MR_SEXTIC_M) zzn2_inv(&r); // could be precalculated
    zzn2_mul(&r, &r, &w); //w=r*r
    zzn2_conj(&x, &x); zzn2_mul(&w, &x, &x);
    zzn2_conj(&y, &y); zzn2_mul(&w, &r, &w); zzn2_mul(&w, &y, &y);
    zzn2_conj(&z, &z);
    ecn2_setxyz(&x, &y, &z, &A);
}

```

\*\*\*\*\*/

Function: line

Description: Line from A to destination C. Let A=(x, y)

Line  $Y - \text{slope} \cdot X - c = 0$ , through A, so intercept  $c = y - \text{slope} \cdot x$

Line  $Y - \text{slope} \cdot X - y + \text{slope} \cdot x = (Y - y) - \text{slope} \cdot (X - x) = 0$

Now evaluate at Q -> return  $(Q_y - y) - \text{slope} \cdot (Q_x - x)$

see akel2bnx.cpp for details in MIRACL c++ source file

Calls: MIRACL functions, zzn12\_init

Called By: g

Input: ecn2 A, ecn2 \*C, ecn2 \*B, zzn2 slope, zzn2 extra, BOOL Doubling, big Qx, big Qy

Output:

Return: zzn12

Others:

\*\*\*\*\*/

```

zzn12 line(ecn2 A, ecn2 *C, ecn2 *B, zzn2 slope, zzn2 extra, BOOL Doubling, big Qx, big Qy)
{
    zzn12 res;
    zzn2 X, Y, Z, Z2, U, QY, CZ;
    big QX;

    QX=mirvar(0);
    X.a=mirvar(0);X.b=mirvar(0);
    Y.a=mirvar(0);Y.b=mirvar(0);
    Z.a=mirvar(0);Z.b=mirvar(0);

```

```

Z2.a=mirvar(0);Z2.b=mirvar(0);
U.a=mirvar(0);U.b=mirvar(0);
QY.a=mirvar(0);QY.b=mirvar(0);
CZ.a=mirvar(0);CZ.b=mirvar(0);
zzn12_init(&res);

ecn2_getz(C,&CZ);
// Thanks to A. Menezes for pointing out this optimization...
if (Doubling)
{
    ecn2_get(&A,&X,&Y,&Z);
    zzn2_mul(&Z,&Z,&Z2); //Z2=Z*Z

    //X=slope*X-extra
    zzn2_mul(&slope,&X,&X);
    zzn2_sub(&X,&extra,&X);

    zzn2_mul(&CZ,&Z2,&U);

    //(-(Z*Z*slope)*Qx);
    nres(Qx,QX);
    zzn2_mul(&Z2,&slope,&Y);
    zzn2_smul(&Y,QX,&Y);
    zzn2_negate(&Y,&Y);

    if (get_mip()->TWIST==MR_SEXTIC_M)
    { // "multiplied across" by i to simplify
        zzn2_from_big(Qy,&QY);
        zzn2_txx(&QY);
        zzn2_mul(&U,&QY,&QY);
        zzn4_from_zzn2s(&QY,&X,&res.a);
        zzn2_copy(&Y,&(res.c.b));
    }
    if (get_mip()->TWIST==MR_SEXTIC_D)
    {
        zzn2_smul(&U,Qy,&QY);
        zzn4_from_zzn2s(&QY,&X,&res.a);
        zzn2_copy(&Y,&(res.b.b));
    }
}
else
{ //slope*X-Y*Z
    ecn2_getxy(B,&X,&Y);
    zzn2_mul(&slope,&X,&X);

```

```

    zzn2_mul (&Y, &CZ, &Y);
    zzn2_sub (&X, &Y, &X);

    //(-slope*Qx)
    nres (Qx, QX);
    zzn2_smul (&slope, QX, &Z);
    zzn2_negate (&Z, &Z);

    if (get_mip()->TWIST==MR_SEXTIC_M)
    {
        zzn2_from_big (Qy, &QY);
        zzn2_txx (&QY);
        zzn2_mul (&CZ, &QY, &QY);

        zzn4_from_zzn2s (&QY, &X, &res. a);
        zzn2_copy (&Z, &(res. c. b));
    }
    if (get_mip()->TWIST==MR_SEXTIC_D)
    {
        zzn2_smul (&CZ, Qy, &QY);
        zzn4_from_zzn2s (&QY, &X, &res. a);
        zzn2_copy (&Z, &(res. b. b));
    }
}

return res;
}

/*****
Function:      g
Description:    Add A=A+B (or A=A+A), Return line function value
                see ake12bnx.cpp for details in MIRACL c++ source file
Calls:         MIRACL functions, zzn12_init, line
Called By:
Input:         ecn2 *A, ecn2 *B, big Qx, big Qy
Output:
Return:        zzn12
Others:
*****/
zzn12 g(ecn2 *A, ecn2 *B, big Qx, big Qy)
{
    zzn2 lam, extra;
    BOOL Doubling;
    ecn2 P;
    zzn12 res;

```

```

    lam.a=mirvar(0);lam.b=mirvar(0);
    extra.a=mirvar(0);extra.b=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;

    zzn12_init(&res);
    ecn2_copy(A,&P);
    Doubling=ecn2_add2(B,A,&lam,&extra);
    if(A->marker==MR_EPOINT_INFINITY)
    {
        zzn4_from_int(1,&res.a);
        res.miller=FALSE;
        res.unitary=TRUE;
    }
    else res=line(P,A,B,lam,extra,Doubling,Qx,Qy);
    return res;
}

/*****
Function:      fast_pairing
Description:    R-ate Pairing  $G_2 \times G_1 \rightarrow GT$ 
                P is a point of order q in  $G_1$ .  $Q(x,y)$  is a point of order q in  $G_2$ .
                Note that P is a point on the sextic twist of the curve over  $F_{p^2}$ ,
                 $Q(x,y)$  is a point on the curve over the base field  $F_p$ 
                see akel2bnx.cpp for details in MIRACL c++ source file
Calls:         MIRACL functions, zzn12_init, g, q_power_frobenius
                zzn12_copy, zzn12_conj, zzn12_div, zzn12_powq, zzn12_inverse
Called By:     ecap
Input:         ecn2 P, big Qx, big Qy, big x, zzn2 X
Output:        zzn12 *r
Return:        FALSE: r=0
                TRUE: correct calculation

Others:
*****/
BOOL fast_pairing(ecn2 P, big Qx, big Qy, big x, zzn2 X, zzn12 *r)
{
    int i,nb;
    big n,zero,negify_x;
    ecn2 A,KA;
    zzn12 t0,x0,x1,x2,x3,x4,x5,res;

    zero=mirvar(0);n=mirvar(0);negify_x=mirvar(0);
    A.x.a=mirvar(0); A.x.b=mirvar(0); A.y.a=mirvar(0);A.y.b=mirvar(0);
    A.z.a=mirvar(0); A.z.b=mirvar(0); A.marker=MR_EPOINT_INFINITY;
    KA.x.a=mirvar(0); KA.x.b=mirvar(0); KA.y.a=mirvar(0);KA.y.b=mirvar(0);

```

```

KA.z.a=mirvar(0); KA.z.b=mirvar(0); KA.marker=MR_EPOINT_INFINITY;
zzn12_init(&t0);zzn12_init(&x0);zzn12_init(&x1);zzn12_init(&x2);
zzn12_init(&x3);zzn12_init(&x4);zzn12_init(&x5);zzn12_init(&res);

premult(x, 6, n);incr(n, 2, n);//n=(6*x+2);
if (mr_compare(x, zero)<0) //x<0
    negify(n, n); //n=-(6*x+2);

ecn2_copy(&P, &A);
nb=logb2(n);
zzn4_from_int(1, &res.a);res.unitary=TRUE; //res=1
// Short Miller loop
res.miller=TRUE;

for (i=nb-2;i>=0;i--)
{
    zzn12_mul(res, res, &res);
    zzn12_mul(res, g(&A, &A, Qx, Qy), &res);
    if(mr_testbit(n, i))
        zzn12_mul(res, g(&A, &P, Qx, Qy), &res);
}
// Combining ideas due to Longa, Aranha et al. and Naehrig
ecn2_copy(&P, &KA);
q_power_frobenius(KA, X);
if(mr_compare(x, zero)<0)
{
    ecn2_negate(&A, &A);
    zzn12_conj(&res, &res);
}
zzn12_mul(res, g(&A, &KA, Qx, Qy), &res);
q_power_frobenius(KA, X);
ecn2_negate(&KA, &KA);
zzn12_mul(res, g(&A, &KA, Qx, Qy), &res);

if(zzn4_iszero(&res.a)&&zzn4_iszero(&res.b)&&zzn4_iszero(&res.c)) return FALSE;

// The final exponentiation
zzn12_copy(&res, &t0);//t0=r;
zzn12_conj(&res, &res);
zzn12_div(res, t0, &res);

res.miller=FALSE;res.unitary=FALSE;

zzn12_copy(&res, &t0);//t0=r;

```



```

    zzn12_powq(X,&res);
    zzn12_powq(X,&res);
    zzn12_mul(res,t0,&res);// r^[(p^6-1)*(p^2+1)]
    res.miller=FALSE;res.unitary=TRUE;

    // Newer new idea...
    // See "On the final exponentiation for calculating pairings on ordinary elliptic curves"
    // Michael Scott and Naomi Benger and Manuel Charlemagne and Luis J. Dominguez Perez and Ezekiel
J. Kachisa
    zzn12_copy(&res,&t0);zzn12_powq(X,&t0);
    zzn12_copy(&t0,&x0); zzn12_powq(X,&x0);    //x0=t0

    zzn12_mul(res,t0,&x1);zzn12_mul(x0,x1,&x0);// x0*=(res*t0);
    zzn12_powq(X,&x0);

    x1=zzn12_inverse(res);// just a conjugation!
    negify(x,negify_x); x4=zzn12_pow(res,negify_x);//negify_x=-x    x is sparse.
    zzn12_copy(&x4,&x3);zzn12_powq(X,&x3);

    x2=zzn12_pow(x4,negify_x);
    x5=zzn12_inverse(x2);
    t0=zzn12_pow(x2,negify_x);

    zzn12_powq(X,&x2);
    zzn12_div(x4,x2,&x4);

    zzn12_powq(X,&x2);
    zzn12_copy(&t0,&res);// res=t0
    zzn12_powq(X,&res);
    zzn12_mul(t0,res,&t0);

    zzn12_mul(t0,t0,&t0);zzn12_mul(t0,x4,&t0);zzn12_mul(t0,x5,&t0);//t0*=t0;t0*=x4;t0*=x5;
    zzn12_mul(x3,x5,&res);zzn12_mul(res,t0,&res);//res=x3*x5;res*=t0;
    zzn12_mul(t0,x2,&t0);//t0*=x2;
    zzn12_mul(res,res,&res);zzn12_mul(res,t0,&res);zzn12_mul(res,res,&res);//res*=res;
res*=t0;res*=res;
    zzn12_mul(res,x1,&t0);// t0=res*x1;
    zzn12_mul(res,x0,&res);//res*=x0;
    zzn12_mul(t0,t0,&t0);zzn12_mul(t0,res,&t0);//t0*=t0;t0*=res;

    zzn12_copy(&t0,r);//r= t0;

    return TRUE;
}

```

```

/*****
Function:      ecap
Description:   caculate Rate pairing
               see ake12bnx.cpp for details in MIRACL c++ source file
Calls:        MIRACL functions, fast_pairing
Called By:
Input:        ecn2 P, epoint *Q, big x, zzn2 X
Output:       zzn12 *r
Return:       FALSE: calculation error
               TRUE: correct calculation

Others:

```

\*\*\*\*\*/

```

BOOL ecap(ecn2 P, epoint *Q, big x, zzn2 X, zzn12 *r)

```

```

{
    BOOL Ok;
    big Qx, Qy;
    Qx=mirvar(0); Qy=mirvar(0);

    ecn2_norm(&P);
    epoint_get(Q, Qx, Qy);

    Ok=fast_pairing(P, Qx, Qy, x, X, r);

    if (Ok) return TRUE;

    return FALSE;
}

```

\*\*\*\*\*/

```

Function:      member
Description:   ctest if a zzn12 element is of order q
               test  $r^q = r^{(p+1-t)} = 1$ , so test  $r^p = r^{(t-1)}$ 
               see ake12bnx.cpp for details in MIRACL c++ source file
Calls:        MIRACL functions, zzn12_init, zzn12_copy, zzn12_powq
Called By:
Input:        zzn12 r, big x, zzn2 F
Output:       NULL
Return:       FALSE: zzn12 element is not of order q
               TRUE: zzn12 element is of order q

Others:

```

\*\*\*\*\*/

```

BOOL member(zzn12 r, big x, zzn2 F)

```

```

{
    zzn12 w;
    big six;

```

```

six=mirvar(0);
zzn12_init(&w);

convert(6,six);
zzn12_copy(&r,&w); //w=r
zzn12_powq(F,&w);
r=zzn12_pow(r,x);r=zzn12_pow(r,x);r=zzn12_pow(r,six); // t-1=6x^2
if(zzn4_compare(&w.a,&r.a)&&zzn4_compare(&w.a,&r.a)&&zzn4_compare(&w.a,&r.a)) return
TRUE;
return FALSE;
}

```