Mathematisches Institut
der Universität München
Pascal Reisert

Wintersemester 2015/2016
Übungsblatt 12
27. Januar 2016

Übungen zur Algorithmischen Zahlentheorie

Aufgabe 46

Wir erkennen sofort, dass $(0,0) \in E$ liegt. Weiter hat $(0,0)$ die Ordnung 2 und diese muss die Gruppenordnung teilen. Man beachte, dass eine nicht-singuläre[1] Kurve $a_3x^3+a_2x^2+a_1x-y^3=0$ in $x_0=y_0=0$ die Tangentialgleichung

$$(3a_3x_0^2+2a_2x_0+a_1)(x-x_0)-2y_0(y-y_0)=a_1x=0,$$

und damit eine Tangente senkrecht zur x-Achse, hat.

Aufgabe 47

Sei $p=3 \bmod 4$.
Für $A=0$ und $B=b$ wird die erste Form (i) dargestellt. Sei $\tilde{a}_0 \in \{1,a_0\}$. Durch umformen der Gleichung erhalten wir den Bedingungen $\beta^2=\alpha^3$ und $\alpha A=\tilde{a}_0\beta^2=\alpha^3 a_0 \Rightarrow (\pm\alpha)^2=\tilde{a}_0/A$. Ein solches α existiert, wenn $\left(\frac{a_0}{p}\right)=-1=\left(\frac{A}{p}\right)$ (Fall (iii)) oder $\left(\frac{1}{p}\right)=1=\left(\frac{A}{p}\right)$ (Fall (ii)).
Man beachte weiter

$$\left(\frac{\alpha^3}{p}\right)\left(\frac{-\alpha^3}{p}\right)=\left(\frac{\alpha}{p}\right)\left(\frac{-\alpha}{p}\right)=\left(\frac{-1}{p}\right)=-1,$$

d.h. wir können α als Quadrat $\alpha=q^2$ wählen. Entsprechend finden wir ein geeignetes $\beta=q^3$.

Aufgabe 48

Wir ändern die Berechnung des diskreten Logarithmus aus Aufgabe 44 auf elliptische Kurven. Dazu benötigen wir die folgenden `aribas`-Funktionen:

```
const
    Origin = (0,-1);
end;

function ecp_add(p,a: integer; P,Q: array[2]) : array[2];
external
    Origin: const;
var
    m,m1,den,x,x1,x2,y,y1,y2: integer;
begin
    if Q = Origin then
        return P;
    elsif P = Origin then
        return Q;
    elsif P = Q then
```

[1] $4a_3^2a_1^3-a_1^2a_2^2a_3 \neq 0$, also insbesondere $a_1 \neq 0$.

```
        return ecp_dup(p,a,P);
    end;
    (x1,y1) := P; (x2,y2) := Q;
    den := (x1 - x2) mod p;
    if den = 0 then return Origin; end;
    m1 := mod_inverse(den,p);
    m := (y1 - y2)*m1 mod p;
    x := (m*m - x1 - x2) mod p;
    y := (m * (x1 - x) - y1) mod p;
    return (x,y);
end;

(*-------------------------------------------------------------*)
(*
** Verdopplung des Punktes P auf der elliptischen Kurve
** mit affiner Gleichung  Y**2 = X**3 + a*X + b
*)
function ecp_dup(p,a: integer; P: array[2]) : array[2];
external
    Origin: const;
var
    m,m1,den,x,x1,y,y1: integer;
begin
    if P = Origin then
        return P;
    end;
    (x1,y1) := P;
    m := (3 * x1*x1 + a) mod p;
    den := 2*y1 mod p;
    if den = 0 then return Origin; end;
    m1 := mod_inverse(den,p);
    m := m*m1 mod p;
    x := (m*m - 2*x1) mod p;
    y := (m * (x1 - x) - y1) mod p;
    return (x,y);
end;

(*-------------------------------------------------------------*)
(*
** Multiplikation eines Punktes P mit der ganzen Zahl s >= 0
*)
function ecp_mult(p,a:integer; P:array[2]; s:integer): array[2];
external
    Origin: const;
var
    i: integer;
    Q: array[2];
begin
    if (s = 0) or (P = Origin) then
```

```
        return Origin;
    end;
    Q := P;
    for i := bit_length(s)-2 to 0 by -1 do
        Q := ecp_dup(p,a,Q);
        if bit_test(s,i) then
            Q := ecp_add(p,a,Q,P);
        end;
    end;
    return Q;
end;

function lrho_stepec(g,x,y:array[2]; p,q:integer; var nu,mu:integer): array[2];
var
s: integer;
begin
s := y[0] mod 3;
if s = 0 then
y := ecp_add(p,1,y,g)
inc(mu);
elsif s = 1 then y := ecp_add(p,1,y,y);
nu := 2*nu mod q;
mu := 2*mu mod q;
else
y := ecp_add(p,1,y,x);
inc(nu);
end;
return y;
end;

function dlog_rhoec(g,x:array[2]; p,q: integer): integer;
var
nu, mu, nu1, mu1, nu2, mu2, i: integer;
y,z:array[2];
found: boolean;
begin
nu1 := nu2 := 1;
mu1 := mu2 := 1 + random(q-1);
y := z := ecp_add(p,1,ecp_mult(p,1,g,mu1),x);
found := false;
for i := i to 10*isqrt(q) do
y := lrho_stepec(g,x,y,p,q,nu1,mu1);
z := lrho_stepec(g,x,z,p,q,nu2,mu2);
z := lrho_stepec(g,x,z,p,q,nu2,mu2);
if i mod 1024 = 1 then write('.'); end;
if z = y then writeln(" (",i," steps)");
found := true; break;
end;
end;
```

```
if found then
nu := nu2 - nu1; mu := mu1 - mu2;
if gcd(nu,q) = 1 then
return (mod_inverse(nu,q)*mu) mod q;
end;
end;
return -1;
end.
```

Wir wenden diese Funktion auf unsere Zahlen an und erhalten

```
P:= (68316_95389, 2867859999);
P0:= (1000000001, 48575_61959);
q0:= 85469_44457;
p:= 85470_62921;

omega:=dlog_rhoec(P0,P,p,q0).
```

Nun lässt sich für

```
(xi01,eta021,C1):= (17143_95687, 0, 78155_21784);
(xi02,eta022,C2):=(715_77867, 1, 84588_52664);
(xi03,eta023,C3):=(84439_74843, 0, 60487_94212);
(xi04,eta024,C4):=(43573_74450, 1, 82421_77945);
(xi05,eta025,C5):=(7576_68770, 0, 56744_00855);
```

der Geheimtext wie folgt berechnen:

```
function order(y:array[2]; p:integer):integer;
var
m:array[2];
begin
m:=ecp_point(p,1,60,y);
if m[1] mod 2 = y[1] then return m;
else return (m[0],(-m[1]) mod p);
end;
-1;
end.

(xi01,eta01):=order(xi01,p);
(xi02,eta02):=order(xi02,p);
(xi03,eta03):=order(xi03,p);
(xi04,eta04):=order(xi04,p);
(xi05,eta05):=order(xi05,p);

(xi1,eta1):=ecp_mult(p,1,(xi01,eta01),omega);
(xi2,eta2):=ecp_mult(p,1,(xi02,eta02),omega);
(xi3,eta3):=ecp_mult(p,1,(xi03,eta03),omega);
(xi4,eta4):=ecp_mult(p,1,(xi04,eta04),omega);
(xi5,eta5):=ecp_mult(p,1,(xi05,eta05),omega);
```

```
K1:=mod_inverse(xi1,p)*C1 mod p;
K2:=mod_inverse(xi2,p)*C2 mod p;
K3:=mod_inverse(xi3,p)*C3 mod p;
K4:=mod_inverse(xi4,p)*C4 mod p;
K5:=mod_inverse(xi5,p)*C5 mod p;

bb1:=string(byte_string(K1));
bb2:=string(byte_string(K2));
bb3:=string(byte_string(K3));
bb4:=string(byte_string(K4));
bb5:=string(byte_string(K5));
concat(bb1,bb2,bb3,bb4,bb5).

"M_74207281 is prime "
```

Unter

`http://www.mersenne.org/primes/?press=M74207281`

findet man Informationen zu der zurzeit größten bekannten Mersenne-Primzahl $M_{74207281}$.