The University of Sydney
Math3925 Public Key Cryptography

Semester 2 **Exercises and Solutions for Week 12** 2004

The Menezes, Okamoto, and Vanstone (MOV) algorithm is one of the few known subexponential algorithms for tackling the discrete logarithm on an elliptic curve $E/\mathbb{F}_q$. It applies when the full n-torsion subgroup $E[n] \subset E(\overline{\mathbb{F}}_q)$ is defined over a small extension field $\mathbb{F}_{q^r}$. The primary application of this method is to *supersingular elliptic curves.*

The MOV algorithm makes use of the *Weil pairing* to map an elliptic curve discrete logarithm problem into a finite field discrete logarithm problem. In this exercise we use `Magma` to investigate the properties of the Weil pairing and its application to discrete logarithms.

1. Let $e_n(R, S)$ be the Weil pairing of points R and S. In `Magma` this is constructed as `WeilPairing(R,S,n)`. For points R and S in the subgroup $\langle P, Q\rangle$ verify the properties

a. $e_n(R, R) = 1$;

b. $e_n(R, S) = e_n(S, R)^{-1}$; and

c. $e_n(xR, yS) = e_n(R, S)^{xy}$ for all $x, y \in \mathbb{Z}$;

Solution We demonstrate that the image of the Weil pairing is an n-torsion element:

```
> WeilPairing(P,Q,500041)^500041;
1
```

and the alternating property of the Weil pairing:

```
> WeilPairing(P,Q,500041) * WeilPairing(Q,P,500041);
1
```

2. The MOV reduction algorithm makes use of property (3) to reduce a discrete logarithm problem $\log_P(xP)$ on an elliptic curve to the discrete logarithm problem $\log_\alpha(\beta)$ where $\alpha = e_n(P, Q)$ and $\beta = e_n(xP, Q)$. Using your points P and Q, verify the equivalence of these two discrete logarithms for several values of x.

Solution The equality of the discrete logarithm in $E(\mathbb{F}_p)$ and its image in $\mathbb{F}_{p^2}^*$ can be verified:

```
> Log(WeilPairing(P,Q,500041),WeilPairing(7*P,Q,500041));
7
> Log(WeilPairing(P,Q,500041),WeilPairing(17*P,Q,500041));
17
> Log(WeilPairing(P,Q,500041),WeilPairing(2000*P,Q,500041));
2000
```

3. Use the constructor `SupersingularEllipticCurve` to create larger examples and compare the performance of the elliptic curve and finite field discrete logarithms.

Solution The command `SupersingularEllipticCurve` constructs an elliptic curve on which a similar MOV reduction of the discrete logarithm can be carried out.

4. Let $E/\mathbb{F}_p$, where $p = 1000081$ be the supersingular elliptic curve

$$y^2 = x^3 + 394763x + 255869,$$

and let $P = (416961 : 144117 : 1)$. Show that P has prime order 500041, and find a point $Q \in E(\mathbb{F}_{p^2})$ such that $E[500041] = \langle P, Q\rangle$.

Solution We first present some `Magma` code which explores the Weil pairings.

Note that $p + 1 = 2n$ so the curve is supersingular. A random point over the quadratic extension, mapped by $[2]$ into the n-torsion subgroup, should be independent of P.

```
> p := 1000081;
> F := FiniteField(p);
> E := EllipticCurve([F|394763,255869]);
> P := E![416961,144117];
> Order(P);
500041
> K<a> := FiniteField(p^2);
> // Note that E is supersingular, so E(K) = E[p+1],
> // where p+1 = 2*500041.  Thus the duplicate of any
> // random point will be 500041-torision.
> Q := 2*Random(E(K));
> Order(Q);
500041
```

We verify this with the Weil pairing, which should return 1 if and only if P and Q are linearly dependent (i.e. lie in the same cyclic subgroup of order n):

```
> // First map P from E(F) to the set E(K).
> P := E(K)
> WeilPairing(P,Q,500041);
247478*a + 211942
```

5. The multiplication-by-n maps $[n]$ on an elliptic curve $E : y^2 = x^3 + ax + b$ induces a well-defined rational map on the x-coordinates. In order to allow for roots of the denominator polynomial, we express E in projective coordinates and write

$$E : Y^2Z = X^3 + aXZ^2 + bZ^3.$$

Then we express the maps $[n]$ on the XZ-projective line:

$$[n](X : Z) = \begin{cases} \big(\Phi_n(X,Z) : \Psi_n(X,Z)^2\, Z\big) & n \text{ odd} \\ \big(\Phi_n(X,Z) : \Psi_n(X,Z)^2\, F_2(X,Z)\, Z\big) & n \text{ even} \end{cases}$$

with initializations

$$\begin{array}{l}\Psi_0 = 0 \quad \Psi_1 = 1 \quad \Psi_2 = 1 \\ \Psi_3 = 3X^4 + 6aX^2Z^2 + 12bXZ^3 - a^2Z^4 \\ \Psi_4 = 2X^6 + 10aX^4Z^2 + 40bX^3Z^3 - 10a^2X^2Z^4 - 8abXZ^5 - (2a^3 - 16b^2)Z^6\end{array}$$

$F_2 = 4X^3 + 4aXZ^2 + 4bZ^3$ and subsequent recursions

$$\begin{array}{l}\Psi_{4m+1} = F_2^2\Psi_{2m+2}\Psi_{2m}^3 - \Psi_{2m-1}\Psi_{2m+1}^3 \\ \Psi_{4m+3} = \Psi_{2m+3}\Psi_{2m+1}^3 - F_2^2\Psi_{2m}\Psi_{2m+2}^3 \\ \Psi_{2m} = \Psi_m\big(\Psi_{m+2}\Psi_{m-1}^2 - \Psi_{m-2}\Psi_{m+1}^2\big).\end{array}$$

The recursions for $\Phi_n(X, Z)$ are given by $\Phi_0 = 1$ and

$$\Phi_n = \begin{cases} XF_2\Psi_n^2 - \Psi_{n+1}\Psi_{n-1} & n \text{ even} \\ X\Psi_n^2 - F_2\Psi_{n+1}\Psi_{n-1} & n \text{ even}\end{cases}$$

for all $n \geq 1$.

Note that `Magma` does not handle elliptic curves over rings such as $\mathbb{Z}/N\mathbb{Z}$ which are not fields, but using the above formulas you can determine the application of exponention in the group law on the x-coordinates of points over general rings. In the following exercise, let E be the elliptic curve $y^2 = x^3 - x + 1$ with point $P = (1, 1)$.

a. Compute $[11]P$ over $\mathbb{Q}$, over $\mathbb{F}_{101}$ and over $\mathbb{F}_{103}$. Use these results to find $[11]P$ in $E(\mathbb{Z}/N\mathbb{Z})$ where $N = 10403 = 101 \cdot 103$.

b. Use the recursions above to verify the value of the x-coordinates of $[n]P$ in the group $E(\mathbb{Q})$ of points over $\mathbb{Q}$. You may use the function:

```
function EllipticExponential(n,a,b,X,Z)
    if n mod 2 eq 1 then
        return [ Phi(n,a,b,X,Z), Psi(n,a,b,X,Z)^2 * Z ];
    else
        F2 := 4*(X^3 + a*X*Z^2 + b*Z^3);
        return [ Phi(n,a,b,X,Z), Psi(n,a,b,X,Z)^2 * F2 * Z ];
    end if;
end function;
```

together with the `Magma` functions `Psi` and `Phi` below.

c. Compute the x-coordinates of $[n]P$ in $E(\mathbb{Z}/N\mathbb{Z})$ for n a product of high powers of small primes. At what point can you identify the factorization of N?

Solution

a. In order to compute $[11]P$, one can create the point P as a Magma elliptic curve. The reduction of the coordinates over $\mathbb{Q}$ modulo 101 and modulo 103 will agree with $[11]P$ generated over these fields. Similarly, the point over $\mathbb{Z}/N\mathbb{Z}$ will agree with the reconstruction of the points over $\mathbb{F}_{101}$ and $\mathbb{F}_{103}$ by the CRT or by reduction of the point over $\mathbb{Q}$ modulo N.

b. The x-coordinates of $[n]P$ are determined by the recursions above, whenever the denominator is a unit.

c. Using a naive implementation of elliptic curve exponentiation followed by a Pollard ρ algorithm, one can determine the factorization of N. See the code which follows for a sketch of this approach.

```
N := 101*103;
R := ResidueClassRing(N);
a := Random(R);
E := [ a, 1]; // y^2 = x^3 + ax + 1
P := [ 0, 1 ]; // the point (x:*:z) = (0:*:1) on E
B := Floor(Log(N)^2);
print "Powering by all primes up to", B;
P, M := EllipticPrimePowering(E,P,B);
t := 2;
Q := EllipticExponential(E,P,t);
print "Now run:
> EllipticPollardRho(E,P,Q,t,1000);
to search for cycles in the sequence P, [t]P, [t^2]P, ...";

function EllipticPrimePowering(E,P,B);
    p := 1;
    N := Modulus(Universe(E));
    while p lt B do
        p := NextPrime(p);
        e := Ceiling(Log(p,N));
        print "Exponentiating by p =", p;
        for j in [1..e] do
            P := EllipticExponential(E,P,p);
            M := Integers()!GCD(N,P[2]);
            if M eq 1 then
                P := [ P[1]*P[2]^-1, 1 ];
            else
                K := N div M;
                X := Integers()!P[1]; Z := Integers()!P[2];
                print "Found divisor:", M;
                printf "P = %o = %o mod %o\n", P, [ X mod M, Z mod M ], M;
                print "w/ complement:", K;
                printf "P = %o = %o mod %o\n", P, [ X mod K, Z mod K ], K;
                break;
            end if;
        end for;
        print "P =", P;
    end while;
    return P, M;
end function;
```

```
function EllipticPollardRho(E,P,Q,a,k)
    N := Modulus(Universe(E));
    for i in [1..k] do
        P := EllipticExponential(E,P,2);
        Q := EllipticExponential(E,Q,2);
        Q := EllipticExponential(E,Q,2);
        M := Integers()!GCD(N,P[2]);
        if M eq 1 then
            P := [ P[1]*P[2]^-1, 1 ];
        else
            print "Found divisor:", M;
            break i;
        end if;
        M := Integers()!GCD(N,Q[2]);
        if M eq 1 then
            Q := [ Q[1]*Q[2]^-1, 1 ];
        else
            print "Found divisor:", M;
            break i;
        end if;
        xP := Integers()!P[1];
        xQ := Integers()!Q[1];
        M := Integers()!GCD([xP-xQ,N]);
        if M ne 1 then
            print "Found match:";
            printf "%6o: xP = %o xQ = %o\n", i, xP, xQ;
            break i;
        end if;
    end for;
    return M, P, Q;
end function;
```

`Magma` code for the functions $\Psi_n(X,Y)$ and $\Phi_n(X,Y)$, given any a and b in a ring R are given below, first for Ψ_n:

```
function Psi(n,a,b,X,Z)
    if n eq 0 then return 0; end if;
    if n le 2 then return 1; end if;
    if n eq 3 then
        return 3*X^4+(6*X*(a*X+2*b*Z)-(a*Z)^2)*Z^2;
    elif n eq 4 then
        return 2*X^6 + 10*a*X^4*Z^2 + 40*b*X^3*Z^3
            - 10*a^2*X^2*Z^4 - 8*a*b*X*Z^5 - (2*a^3+16*b^2)*Z^6;
    end if;
    m := n div 2;
    if n mod 2 eq 0 then
        return Psi(m,a,b,X,Z) * (
            Psi(m+2,a,b,X,Z) * Psi(m-1,a,b,X,Z)^2
            - Psi(m-2,a,b,X,Z) * Psi(m+1,a,b,X,Z)^2);
    else
        F2 := 4*(X^3+(a*X+b*Z)*Z^2);
        if m mod 2 eq 0 then
            return F2^2 * Psi(m+2,a,b,X,Z) * Psi(m,a,b,X,Z)^3
                - Psi(m-1,a,b,X,Z) * Psi(m+1,a,b,X,Z)^3;
        else
            return Psi(m+2,a,b,X,Z) * Psi(m,a,b,X,Z)^3
                - F2^2 * Psi(m-1,a,b,X,Z) * Psi(m+1,a,b,X,Z)^3;
        end if;
    end if;
end function;
```

and subsequently for Φ_n:

```
function Phi(n,a,b,X,Z)
    if n eq 0 then return 0; end if;
    if n eq 1 then return X; end if;
    F2 := 4*(X^3 + (a*X + b*Z)*Z^2);
    if n mod 2 eq 0 then
        return X * Psi(n,a,b,X,Z)^2 * F2
            - Psi(n+1,a,b,X,Z) * Psi(n-1,a,b,X,Z);
    else
        return X * Psi(n,a,b,X,Z)^2
            - Psi(n+1,a,b,X,Z) * Psi(n-1,a,b,X,Z) * F2;
    end if;
end function;
```