CS 259C/Math 250: Elliptic Curves in Cryptography
Homework 3 Solutions

1. The basic idea of the new signature scheme is that \((e, s)\) can be computed from \((R, s)\) and vice versa, given \(M\). Given \(M\) and \(\sigma = (R, s)\), we can compute \(\sigma' = (e, s)\) where

\[ e = H(M||R) \]

Given \(M\) and \(\sigma' = (e, s)\), we can compute \(\sigma = (R, s)\) where

\[ R = [s]P - [e]Q = [s - ae]P = [k]P = R \]

Hence, any party can convert a signature from \(\text{Sign}\) to one from \(\text{Sign}'\) and vice versa.

(a) According to the note above, \(\text{Verify}'(pk, M, \sigma') = \text{Verify}(pk, M, \sigma)\) where \(\sigma = ([s]P - [e]Q, s)\). Specifically, \(\text{Verify}'(pk, M, \sigma)\) works as follows:

1. Compute \(R = [s]P - [e]Q\)
2. Compute \(e' = H(M||R)\)
3. Accept if \(R + [e']Q = [s]P\). Note that this condition is equivalent to \(e = e'\)

(b) Suppose we have an adversary \(A\) for \(\text{Sign}'\) with advantage \(\epsilon\). We construct \(B\), and adversary for \(\text{Sign}\), as follows:

1. On input \(pk\), simulate \(A\) on \(pk\).
2. When \(A\) asks for a signature on \(M\), \(B\) asks its challenger for a signature on \(M\). When the challenger responds with \(\sigma = (R, s)\), compute \(e = H(M||R)\) and send \(\sigma' = (e, s)\) to \(A\).
3. When \(A\) returns a forgery candidate \((M, \sigma')\) where \(\sigma' = (e, s)\), \(B\) returns \((M, \sigma)\) where \(\sigma = (R, s)\) and \(R = [s]P - [e]Q\).

To show that the signatures seen by \(A\) are from the same distribution as signatures from \(\text{Verify}'\), note that the signature queries are answered as follows:

1. \(B\)'s challenger chooses a random \(k\) and computes \(R = [k]P\)
2. \(B\)'s challenger computes \(e = H(M||R)\)
3. \(B\)'s challenger sets \(s = k + ae\), and sends \((R, s)\) to \(B\).
4. \(B\) computes \(e' = H(M||R) = e\)
5. \(B\) returns \(\sigma' = (e', s) = (e, s)\) to \(A\).
Hence, this computation is equivalent to the computation of $\sigma'$ from $\text{Verify}'$. This means that the view of $A$ as a subroutine of $B$ is identical to that as an adversary for the modified scheme. Thus, $A$ outputs a valid forgery as a subroutine of $B$ with probability $\epsilon$.

If $A$ outputs a valid forgery $(M, \sigma' = (e, s))$, it means that $A$ never asked for a signature on $M$ and that $\text{Verify}(pk, M, \sigma')$ accepts. But this means that $B$ also never asked for a signature on $M$ and that $\text{Ver}(pk, M, \sigma = ([s]P - [e]Q, s)) = \text{Ver}(pk, M, \sigma' = (e, s))$ accepts. Hence, $(M, \sigma)$ is a valid forgery for $\text{Sign}$. Thus, $B$ outputs a valid forgery with probability $\epsilon$, so its advantage is $\epsilon$.

(c) This scheme has the advantage that signatures are two integers mod $r$ (which takes $2 \log r$ bits to represent) as opposed to a point on the curve and a integer mod $r$ (which takes $2 \log q + \log r$ bits if we naively encode the point using its $x$ and $y$ coordinates) in the original scheme. Even if we compress the representation of the point to $2 \log q + 1$ bits, the modified scheme will still have shorter signatures when $r$ is smaller than $q$.

2. (a) If an adversary could compute a $k$ such that $R = [k]G$ with $x$ coordinate 0, then a valid signature on a document $m$ would be

$$(R, k^{-1}(m + ax) \mod r) = (R, k^{-1}m \mod r)$$

This is easily computable for any $m$.

(b) If $s = 0$, then $k^{-1}(m + ax) \mod r = 0$. This means $a = m/x \mod r$.

<table>
<thead>
<tr>
<th>Time to solve DLP</th>
<th>Size of $p$ for $E(\mathbb{F}_p)$</th>
<th>Size of $p$ for $\mathbb{F}_p^\times$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{38}$</td>
<td>$2^{112}$</td>
<td>$2^{383}$</td>
</tr>
<tr>
<td>$2^{80}$</td>
<td>$2^{160}$</td>
<td>$2^{853}$</td>
</tr>
<tr>
<td>$2^{112}$</td>
<td>$2^{224}$</td>
<td>$2^{1859}$</td>
</tr>
<tr>
<td>$2^{128}$</td>
<td>$2^{256}$</td>
<td>$2^{2547}$</td>
</tr>
<tr>
<td>$2^{192}$</td>
<td>$2^{384}$</td>
<td>$2^{6732}$</td>
</tr>
<tr>
<td>$2^{256}$</td>
<td>$2^{512}$</td>
<td>$2^{13599}$</td>
</tr>
</tbody>
</table>

3. (a) We know from the previous homework that if $E(\mathbb{F}_p)$ is supersingular for a prime $p$, then either $E(\mathbb{F}_p)$ is cyclic or isomorphic to

$$\mathbb{Z}_2 \times \mathbb{Z}_{p+1}/2$$

In the latter case, the entire 2-torsion is contained in $E(\mathbb{F}_p)$. Recall from homework 1 that elements of order 2 are points with $y$ coordinate 0. The $x$ coordinates are thus solutions to

$$x^3 + x = 0$$

One solution is $x = 0$, and the other two are solutions to $x^2 + 1 = 0$. But $-1$ is not a square mod $p$ (since $p$ is 3 mod 4). Therefore, the only point of order 2 in $E(\mathbb{F}_p)$ is $(0, 0)$, meaning the 2-torsion is not contained in $E(\mathbb{F}_p)$. Thus $E(\mathbb{F}_p)$ is cyclic.
(b)-(f) I wrote a routine that solves the discrete log problem mod a given integer, assuming that integer divides the order of $P$.

```python
def DLSolve(P, Q, n):
    # 'Solve for a mod n where Q=aP, assuming n divides the order of P'''
    k=P.order()
    r = floor(k/n)
    pp=r*P
    QQ=r*Q
    return pp.discrete_log(QQ)

b = DLSolve(P, Q, 2); print(b)
c = DLSolve(P, Q, 4); print(c)
d = DLSolve(P, Q, 3); print(d)
e = DLSolve(P, Q, 41); print(e)
a = CRT([c,d,e,s],[4,3,41,t]); print(a)
Q == a*P

1
1
2
39
77717311163486632508230870388156148925713969641
True
```

For part (c) we could also have solved for $a$ mod 4 using part (b). Part (b) tells us that $a$ is odd, so we can write $a = 2a' + 1$. Then defining $Q' = Q - P$, we have $Q' = a'(2P)$. We can then solve for $a'$ mod 2, and find that it is equal to 0. This tells us that $a = 1$ mod 4.

5. (a) Let $m$ be some integer such that $m^2 \geq w + 1$. Compute and save $[b + i]P$ for all $i \in [0, m - 1]$. Now, for each $j \in [0, m - 1]$, compute $Q - [mj]P$, and check if it matches one of the stored values. If we have a match, we have

$$[b + i]P = Q - [mj]P$$

Therefore $[b + mj + i]P = Q$, so $a = b + mj + i$.

This scheme uses $\log b$ group operations to compute $[b]P$, and then $m \approx \sqrt{w}$ group operations to compute $[b + i]P$ for each $i$. Then we need to compute $[mj]P$ using $\log m$ group operations, and we compute $Q - [mj]P$ for each $j$ using another $m \approx \sqrt{w}$ operations. $\log m \approx \frac{1}{2} \log w$ which is much smaller than $\sqrt{w}$. Also, $\log b$ is most likely much smaller than $\sqrt{w}$. Therefore, the total number of group operations is about $2\sqrt{w}$.

This scheme works because we can write $a = b + a_0 + ma_1$ for some $0 \leq a_0, a_1 < m$. When $i = a_0$ and $j = a_1$,

$$Q - [mj]P = [a]P - [ma_1]P = [a_0]P = [i]P$$

So we have found a match.

Alternatively, we can just compute $Q' = Q - [b]P$, and solve $Q' = [a']P$ with $a'$ in the interval $[0, w]$ using the standard baby step-giant step algorithm with an upper bound $w$.

(b) First, compute $Q' = Q - [t]P$, which equals $[a-t]P$. Letting $\bar{a} = a - t$, we are now solving the problem of computing $\bar{a}$ where $Q' = [\bar{a}]P$ and $\bar{a} \equiv 0 \mod m$. That is, $\bar{a} = mt$ for some $t$
Next, compute $P' = [m]P$. We now have $Q' = [a']P'$ where $a' = \frac{\tilde{a}}{m} = \ell$. We know that
\[
a' = \frac{\tilde{a}}{m} = \frac{a - t}{m} < \frac{r}{m}
\]
Therefore, we have reduced this problem to finding the discrete log on an interval of length approximately $r/m$.

6. (a)

```python
def floyd_rho(P, Q):
    """Compute discrete log using Floyd cycle finding."""

    # Initialize as above.
    n = P.order()
    walk = walk_setup(P, Q) # set up the walk function
    u0 = randint(1, P.order())
    xi = (u0*P, u0, 0)
    X2i = Xi
    nWalkCalls = 0

    # Repeat until P_{i+1} = P_{2i}
    while True:
        # Compute P_{i+1} and P_{2i}
        Xi = walk(Xi)
        X2i = walk(walk(X2i))
        nWalkCalls += 3

        (Pi, ui, vi) = Xi
        (P2i, u2i, v2i) = X2i

        if Pi == P2i:
            if (v2i - vi) % n != 0:
                a = ((ui - u2i) / (v2i - vi)) % n
                return a, nWalkCalls
            else:
                u0 = randint(1, P.order())
                Xi = (u0*P, u0, 0)
                X2i = Xi

(b)-(c)
```python
def distpt_rho(P,Q,d):
    """Compute discrete log of q to the base P using distinguished points.
    1/d = probability of hitting a distinguished point"""

    # Initialize as above.
    n = P.order()
    walk = walk_setup(P,Q) # set up the walk function
    u0 = randint(1, P.order())
    Xi = (u0*P, u0, 0)
    nWalkCalls = 0
    iterations = 0
    D = {}

    while(True):
        # Iterate the random walk
        Xi = walk(Xi)
        nWalkCalls += 1
        iterations += 1
        (Pi, ui, vi) = Xi
        x = Pi[0]

        # Test for distinguished points
        if ZZ(x)/d == 0:  # here x is the x-coordinate of P
            if Pi in D:
                (u,v) = D[Pi]
                if ((v-vi) % n) != 0:
                    a = (ui-u)/(v-vi) % n
                    return a,len(D),nWalkCalls
            else:
                D[Pi] = (ui,vi)
                u0 = randint(1, P.order())
                Xi = (u0*P, u0, 0)
                iterations = 0
            elif iterations > 100 * d:
                u0 = randint(1, P.order())
                Xi = (u0*P, u0, 0)
                iterations = 0

print(mean([collision_search(P,Q)[1].N() for i in range(10000)])/sqrt(P.order())).N())
print(mean((ficyd_rho(P,Q)[1].N() for i in range(10000))/sqrt(P.order())).N())
print(mean((distpt_rho(P,Q,32)[2].N() for i in range(10000))/sqrt(P.order())).N())

1.33860182429175
3.23329579472458
1.64856398334897
```
Notice that this plot seems roughly linear in $d$.

7. (a)

$$\hat{f}(-R) = \hat{f}(x, -y) = \begin{cases} 
    f(x, -y) & \text{if } -y < y \pmod{p} \\
    f(x, y) & \text{if } -y > y \pmod{p} 
\end{cases}$$

$$= \begin{cases} 
    f(x, y) & \text{if } y > -y \pmod{p} \\
    f(x, -y) & \text{if } y < -y \pmod{p} 
\end{cases} = \hat{f}(x, y)$$

(b) If $\hat{P}_i$ and $\hat{P}_j$ have the same $x$-coordinate, then $\hat{P}_i = \pm \hat{P}_j$, and we can tell weather it is a plus or minus (by comparing $y$-coordinates). Thus, we have

$$u_iP + v_iQ = \pm (u_jP + v_jQ)$$

This can be rearranged as (assuming $v_i \neq \pm v_j$):

$$Q = -\frac{u_i \mp u_j}{v_i \mp v_j}P$$

Hence, we have computed the discrete log.
(c) Before, we were in a space of $N$ objects, the points on the elliptic curve. Now we are in a space of about $N' = N/2$ objects, the $x$-coordinates of those points. Thus, the average number of iterations is about $c\sqrt{N'} = c\sqrt{N}/2$.

(d) Recall that $f(P) = P + M_x \mod s$. Further, $y_{i+1} > -y_{i+1}$, so $\hat{f}(P_{i+1}) = f(-P_{i+1})$. Thus

$$\hat{P}_{i+2} = \hat{f}(\hat{P}_{i+1}) = f(-\hat{P}_{i+1}) = -\hat{P}_{i+1} + M_{x_{i+1}} \mod s$$
$$= -\hat{f}(\hat{P}_i) + M_{x_{i+1}} \mod s = -f(\pm \hat{P}_i) + M_{x_{i+1}} \mod s$$
$$= \mp \hat{P}_i - M_{x_i} \mod s + M_{x_{i+1}} \mod s = \mp \hat{P}_i$$

Therefore, $\hat{P}_i$ and $\hat{P}_{i+2}$ have the same $x$-coordinate.

(e) Basically, we need to show that the only way to get a cycle of size two is to satisfy the conditions $y_{i+1} > -y_{i+1}$ and $x_i \mod s = x_{i+1} \mod s$. It is not hard to see that the probability that $x_i \mod s = x_{i+1} \mod s$ is $1/s$ if the $x$-coordinates are random. Combined with the assumption that $y_{i+1} > -y_{i+1}$, the probability of meeting these conditions is $1/2s$.

8. (a) Since $\phi$ has degree $q$, according to Washington 3.15, the determinant of $\phi$ as an endomorphism on $E[n]$ is $q \mod n$. This determinant is the product of two eigenvalues $\alpha$ and $\beta$. Since $E(F_q)$ has a point $P$ of order $n$, $E[n]$ contains a point on $E(F_q)$, which is fixed by $\phi$. This means that $P$ is an eigenvector of $\phi$ with eigenvalue 1. Thus, the other eigenvalue is $q$.

(b) $$\hat{e}(P, P) = \hat{e}(\phi P, \phi P) = \hat{e}(P, P)^{\deg \phi} = \hat{e}(P, P)^q$$

Thus $\hat{e}(P, P) \in F_q$. Since $n$ is prime, if $\hat{e}(P, P)$ is not 1, it is a primitive $n$th root of 1, and thus all the $n$th roots of 1 are in $F_q$, a contradiction. Therefore $\hat{e}(P, P) = 1$.

Similarly,

$$\hat{e}(Q, Q)^q = \hat{e}(\phi Q, \phi Q) = \hat{e}(qQ, qQ) = \hat{e}(Q, Q)^{q^2}$$

Thus, the order of $\hat{e}(Q, Q)^q$ divides $q - 1$. By the same argument as above, this means $\hat{e}(Q, Q)^q = 1$. However, the order of $\hat{e}(Q, Q)$ is either $n$ or 1, and $n$ does not divide $q$, so $\hat{e}(Q, Q) = 1$.

(c) $$\hat{e}((1 + \alpha)P, (1 + \alpha)P) = \hat{e}(P, P)\hat{e}(\alpha P, \alpha P)\hat{e}(P, \alpha P)\hat{e}(\alpha P, P)$$
$$= \hat{e}(P, P)\hat{e}(P, P)^{\deg \alpha} \hat{e}(P, \alpha P)\hat{e}(\alpha P, P)$$
$$= \hat{e}(P, \alpha P)\hat{e}(\alpha P, P)$$

But

$$\hat{e}((1 + \alpha)P, (1 + \alpha)P) = \hat{e}(P, P)^{\deg (1 + \alpha)} = 1$$

Thus $\hat{e}(P, \alpha P)) = \hat{e}(\alpha P, P)^{-1}$
(d) Since $n$ is prime and $P$ has order $n$, $\alpha(P)$ must have order $n$ or order 1. However, since $\alpha(P) \notin \langle P \rangle$, $\alpha(P)$ must have order $n$. Thus, $P$ and $\alpha(P)$ must span $E[n]$. Therefore, we can write $T = aP + b\alpha(P)$. 

$$\hat{e}(T, T) = \hat{e}(aP + b\alpha(P), aP + b\alpha(P)) = \hat{e}(P, P)^{\deg(a + b\alpha)} = 1$$

(e) 

$$1 = \hat{e}(S + T, S + T) = \hat{e}(S, S)\hat{e}(T, T)\hat{e}(S, T)\hat{e}(T, S) = \hat{e}(S, T)\hat{e}(T, S)$$

9. In lecture, we saw that if $E[r] \not\subseteq E(\mathbb{F}_q)$, then $r$ divides $q^k - 1$ if and only if $E[r] \subset E(\mathbb{F}_{q^k})$.

(a) 

$$q^3 - 1 = (q - 1)(q^2 + q + 1) = (q - 1)(q + \sqrt{q} + 1)(q - \sqrt{q} + 1)$$

By assumption, $r$ divides $\#E(\mathbb{F}_q) = q + 1 \pm \sqrt{q}$, so $r$ divides $q^3 - 1$

(b) 

$$q^4 - 1 = (q^2 - 1)(q^2 + 1) = (q + 1)(q - 1)(q + \sqrt{2q} + 1)(q - \sqrt{2q} + 1)$$

By assumption, $r$ divides $\#E(\mathbb{F}_q) = q + 1 \pm \sqrt{2q}$, so $r$ divides $q^4 - 1$

(c) 

$$q^6 - 1 = (q^3 - 1)(q^3 + 1) = (q^3 - 1)(q + 1)(q + \sqrt{3q} + 1)(q - \sqrt{3q} + 1)$$

By assumption, $r$ divides $\#E(\mathbb{F}_q) = q + 1 \pm \sqrt{3q}$, so $r$ divides $q^6 - 1$