Applied Cryptography: Assignment 3

Group number 16

Elwin Tamminga s1013846

Lucas van der Laan s1047485

1

(a) We know because of Euler's totient function that the order of a multiplicative prime group is $p-1$, thus g^{p-2} is the last element of the group from g.

This happens, because both x and k are a randomly generated number in $\mathbb{Z}_{p-1}$, thus $y \longleftarrow g^x \pmod p$ with a proper randomness of x will be uniformly distributed over $\mathbb{Z}_p^*$. C_2 is then $C_2 \longleftarrow y^k M \pmod p$, which is the same as $C_2 \longleftarrow (g^x)^k M \pmod p$ which will then be again uniformly distributed over $\mathbb{Z}_p^*$ for any $M < p$.

(b) It is said to be one-way (OW-CPA) if the probability that a polynomial time attacker $\mathcal{A}$ can invert a randomly generated ciphertext $C_2 \longleftarrow y^k M \pmod p$ is negligible. If the computational Diffie-Hellman (CDH) problem is hard, then given the group g, $y \longleftarrow g^x$ and $C_1 \longleftarrow g^k$, it is infeasible to determine g^{xk}. The attacker $\mathcal{A}$ can only inverse C_2 by finding g^{xk} because $y^k = (g^x)^k = g^{xk}$. Thus the cryptosystem is OW-CPA secure if the computational Diffie-Hellman (CDH) problem is hard.

(c) We know that:

- $y = g^x \pmod p$
- $C_2 = y^k M \pmod p$

We now raise C_2 to the order (q) of the p that was used for g, which creates the formula ${C_2}^q = (y^{kq}) \cdot M^q$. If $M^q = 1$, then that means M is in the subgroup generated by g and the attack fails. If $M^q \neq 1$, then we can reduce y^{kq} to $y^{kq} = g^{xkq} = (g^q)^{xk} = 1^{xk} = 1$, this is possible because $g^q = g^0 = 1$ as the subgroup generated by g is cyclic. This then simplifies ${C_2}^q$ to ${C_2}^q = M^q \pmod p$. We can do a Meet-in-the-middle attack by doing the following steps:

- Let $l = log_2(p-1)$
- With non-negligible probability $M = M_1 \cdot M_2$ with $M_1, M_2 < 2^{l/2}$
- The cryptosystem is multiplicative: ${C_2}^q = {M_1}^q \cdot {M_2}^q \pmod p$
- The attacker builds a sorted database $\{1^q, 2^q, \ldots, (2^{l/2})^q\} \pmod p$
- And searches for $c = {C_2}^q / i^q, i \in \{1, 2, \ldots, 2^{l/2}\}$ in the database
- c will show up in at most $2^{l/2}$ and is of shape j^q

- We can now recover the message $M = i \cdot j$

(d) With a randomly chosen k' and M' we can do:

- $C = (C_1, C_2) = (g^k, y^k \cdot M)$
- $C' = (C_1 \cdot g^{k'}, C_2 \cdot y^{k'} \cdot M')$
- We now send C' to Alice
- Alice sends back $M'' \longleftarrow (C_2 \cdot y^{k'} \cdot M') \cdot (C_1 \cdot g^{k'})^{-x}$
- We can now calculate:

$$\begin{aligned} M &= M'' \cdot (M')^{-1} \\ &= \frac{C_2 \cdot y^{k'} \cdot M'}{(C_1 \cdot g^{k'})^x \cdot M'} \\ &= \frac{C_2 \cdot y^{k'} \cdot M'}{({C_1}^x \cdot g^{k'x}) \cdot M'} \\ &= \frac{C_2 \cdot y^{k'}}{{C_1}^x \cdot g^{k'x}} \\ &= \frac{C_2 \cdot g^{xk'}}{{C_1}^x \cdot g^{k'x}} \\ &= C_2 \cdot {C_1}^{-x} \end{aligned}$$

2

(a)

- Generate a $k \xleftarrow{\$} \mathbb{Z}_{p-1}$
- $r \leftarrow g^k \bmod p$
- $s \leftarrow (M - x \cdot r) \cdot k^{-1} \bmod (p-1)$
- If $s \geq r$, try again with another k

This works, because:

$$\begin{aligned} g^M &= y^r \cdot r^s \pmod p \\ &= g^{xg^k} \cdot g^{ks} \pmod p \end{aligned}$$

The following calculations are in $\pmod{p-1}$:

$$\begin{aligned} M &= xg^k + ks \\ &= xg^k + k \cdot ((M - xg^k) \cdot k^{-1}) \\ &= xg^k + (M - xg^k) \\ &= M + xg^k - xg^k \\ M &= M \end{aligned}$$

(b)

- Generate a $k \xleftarrow{\$} \mathbb{Z}_{p-1}$
- $r \leftarrow g^k \cdot y \bmod p$
- $s \leftarrow -r \bmod (p-1)$
- If $s \geq r$, try again with another k

- $M = k \cdot s \pmod{p-1}$

This tuple (r, s) is a valid message for M, because:

$$\begin{aligned} g^M &= y^r \cdot r^s \pmod{p} \\ &= g^{xr} \cdot g^{ks} \cdot y^s \pmod{p} \\ &= g^{xr} \cdot g^{ks} \cdot g^{xs} \pmod{p} \end{aligned}$$

The following calculations are in $\pmod{p-1}$:

$$\begin{aligned} M &= xr + ks + xs \\ &= xr + ks + -rx \\ &= xr - rx + ks \\ &= ks \\ M &= M \end{aligned}$$

3

Binding

Perfect binding: It is not perfectly binding because there exists different B, R values for the same commitment, because the adversary is unbounded and thus it can calculate $\log_g(h) = (R_0 - R_1)/((1 - B) + R_1) - (B + R_0)) = (R_0 - R_1)/(1 - 2B + R_1 + R_0)$.
Statistical binding: It is not statistically binding, because the probability that there is more than one B value for the same commitment is not negligible for an unbounded adversary.
Computational binding: Just like the Pedersen Commitment Scheme, this new scheme is also computationally binding, as it still relies on the same group and this group is computationally hard in the discrete log problem (it is not possible to compute $\log_g(h)$).

Hiding

Perfect hiding: It is perfect hiding, because $g^R \cdot h^R$ and $g^R \cdot h^{1+R}$ are perfectly indistinguishable.
Statistical hiding: Because it is perfect hiding, it is also statistical hiding.
Computational hiding: Because it is perfect hiding, it is also computationally hiding.

4

To prove that this a Σ-protocol, we will prove that the protocol satisfies the following properties:

- **Completeness**

$$\begin{aligned} ah^{ch} &= g_1^{z_1} g_2^{z_2} \\ g_1^{r_1} g_2^{r_2} (g_1^{w_1} g_2^{w_2})^{ch} &= g_1^{r_1 + ch \cdot w_1} g_2^{r_2 + ch \cdot w_2} \\ g_1^{r_1} g_2^{r_2} g_1^{w_1 \cdot ch} g_2^{w_2 \cdot ch} &= g_1^{r_1 + ch \cdot w_1} g_2^{r_2 + ch \cdot w_2} \\ g_1^{r_1 + ch \cdot w_1} g_2^{r_2 + ch \cdot w_2} &= g_1^{r_1 + ch \cdot w_1} g_2^{r_2 + ch \cdot w_2} \end{aligned}$$

- **Special soundness**
  Given two accepting transcripts for the same commitment:

trans = (com, ch, resp) = $(a, ch, (z_1, z_2))$
trans′ = (com, ch′, resp′) = $(a, ch', ({z_1}', {z_2}'))$
We have

$$\begin{aligned} w_1 &= \frac{z_1 - {z_1}'}{ch - ch'} \\ &= \frac{r_1 + ch \cdot w_1 - (r_1 + ch' \cdot w_1)}{ch - ch'} \\ &= \frac{ch \cdot w_1 - ch' \cdot w_1}{ch - ch'} \\ &= \frac{(ch - ch') \cdot w_1}{ch - ch'} \\ w_1 &= w_1 \end{aligned}$$

$$\begin{aligned} w_2 &= \frac{z_2 - {z_2}'}{ch - ch'} \\ &= \frac{r_2 + ch \cdot w_2 - (r_2 + ch' \cdot w_2)}{ch - ch'} \\ &= \frac{ch \cdot w_2 - ch' \cdot w_2}{ch - ch'} \\ &= \frac{(ch - ch') \cdot w_2}{ch - ch'} \\ w_2 &= w_2 \end{aligned}$$

So the witness w_1, w_2 can be extracted with probability 1.

- **Special Honest Verifier ZK**
  For given $ch \in \mathbb{Z}_2^t, 2^t < q$
  Choose $z_1 \xleftarrow{\$} \mathbb{Z}_q, z_2 \xleftarrow{\$} \mathbb{Z}_q$ and calculate

$$a = g_1^{z_1} g_2^{z_2} \cdot h^{-ch}$$

  The distributions of the real transcripts and the simulated transcripts are the same. A given valid transcript occurs with probability $1/2^t$.

5

(a)

$$\begin{aligned} R &= [a]Q \\ &= [a][b]G \\ &= [b][a]G \\ &= [a]P \\ R &= R' \end{aligned}$$

(b)
```
sage: a = 2; b = 2; p = 17
sage: F = FiniteField(p); E = EllipticCurve(F, [a,b])
```

(i)
```
sage: E.cardinality()
19
```

So the cardinality of the group is $\#E(\mathbb{F}_p) = 19$.

(ii)
```
sage: G=E(5,1); G.order()
19
```

So the order of the generator is 19. This is a good generator because the order is the same as the cardinality of the elliptic curve, meaning that the subgroup generated by the generator contains all possible points on the curve. However, 19 is a prime number, so every generator of this curve will have an order of 19 (Lagrange theorem). Thus any point on the curve is a good generator.

(iii)
```
sage: a = 6; P = a*G; P
(16 : 13 : 1)
```

So Alice's public key is $P = (16, 13)$

(iv)
```
sage: Q = E(7,6); R = a*Q; R
(10 : 11 : 1)
```

So the shared secret is $R = (10, 11)$

(v)
```
sage: b = G.discrete_log(Q); b
9
```

So Bob's secret key is: $b = 9$, because $9[G] = Q$.

(c)
```
sage: p = 2^255-19
sage: F = FiniteField(p)
sage: E = EllipticCurve(F, [0,486662, 0, 1, 0])
sage: Gx = 9
sage: Gy = 43114425171068552920764898935933967039370386...
...19820380673076391016620097858254 8
sage: G = E(Gx, Gy)
```

(i)
```
sage: n = E.cardinality(); n
57896044618658097711785492504343953926856...
...93087503926084801560750628363400791 2
sage: q = G.order(); q
72370055773322622139731865630429942408571 1...
...63593799076060019509382854542509 89
```

$n/q = 8$, this means there are 8 times fewer elements in the subgroup of the generator G than the elliptic curve group $\mathbb{F}_p$.

(ii)
```
sage: a = 28111612080006492741872676474874404260...
...74380751304135169920166107709508893727
sage: P = a*G; P
(37048414743519733025193263783831109212639304451...
...10080410409497750778032577183 7 :
79483483033736335310743405124108401743...
...32123202786516445812187645996223942543 : 1)
```

So our public key is: $P =$
$(37048414743519733025193263783831109212639304451100804104094977507780325771837,$
$79483483033736335310743405124108401743321232027865164458121876459962239425 43)$

(iii)
```
sage: Q=E(43943787516356481247689314833207090678477 94...
...39258408729485474542528244416377 27,
21356658827884665961183205449166535999349292496264 27...
...34910547395955462178303)
sage: R = a * Q; R
(12694057863029409910518855127562537703457904794426889564008604777220378...
...465981 : 13554456243083127232280450417120767460444645552951962238781744328587077080733 : 1)
```

So the shared secret is: $R =$
$(12694057863029409910518855127562537703457904794426889564008604777220378465981,$
$13554456243083127232280450417120767460444645552951962238781744328587077080733)$

6

(a) Starting from below:

$$\begin{aligned}
r &= x_1 \\
(x_1, y_1) &= X \\
X &= u_1 G + u_2 Q \\
&= hs^{-1}G + rs^{-1}Q \\
&= \frac{hG + rQ}{s} \\
&= \frac{hG + rQ}{(h + dr)k^{-1}} \\
&= \frac{k(hG + rQ)}{h + dr} \\
&= \frac{k(hG + rdG)}{h + dr} \\
&= \frac{khG + krdG}{h + dr} \\
&= \frac{h + dr(kG)}{h + dr} \\
&= kG = (x_1, y_1) \\
x_1 &= r
\end{aligned}$$

(b)
```
sage: p = 2^255-19
sage: F = FiniteField(p)
sage: E = EllipticCurve(F, [0,486662, 0, 1, 0])

sage: a = 28111612080006492741872676474874404260...
...74380751304135169920166107709508893727
sage: P = E(37048414743519733025193263783831109212639304451...
...10080410409497750778032577 1837,
79483483033736335310743405124108401743...
...32123202786516445812187645996223942543)
```

```
sage: Gx = 9
sage: Gy = 43114425171068552920764898935933967039370386...
...19820380673076391016620097858254 8
sage: G = E(Gx, Gy)

sage: q = G.order(); q
72370055773322622139731865630429942408 5...
...71163593799076060019509382854542509 89

sage: Fq = FiniteField(q)
sage: h = 5
sage: k = int(Fq.random_element()) // to remove the finite field element type
sage: kG = k*G
sage: r = mod(kG[0], q)
sage: s = mod((h + (a * r)) * inverse_mod(k, q), q)
```

$r = 166261667554047798984903964725172678233790993596323376804436638009941774232 6$
$s = 715002568060201476845438503175732479071647765826743518493384013475080943889 2$
$\sigma = (r, s)$

(c)
```
sage: r = mod(193703920910737566124082452472 03...
...59737526758893291527738179502027465721858973, q)
sage: s = mod(237871765932909663400314091483268182...
...57040827989550721523870291836347260061 41, q)
sage: Qb_x = 532327828701224171975911730973085 6...
...87978404961071535861994274005179240746097 69
sage: Qb_y = 37816893436578182161316212686238608330 6...
...98167484762815358233244960714516821501
sage: Qb = E(Qb_x, Qb_y)
sage: h = mod(659207561052421550123056874400183606 6...
...03413890117084054606099726355180250829 7, q)

sage: 0 < int(r), int(s) < q
(True, True)
sage: u1 = h*s^-1; u2 = r * s^-1
sage: X = int(u1) * G + int(u2) * Qb
sage: X!=E(0)
True
sage: mod(X[0], q) == r
True
```

This means that the pair (r, s) is indeed a valid signature for h for the public key Q_b.

(d) $r_1 = r_2$ as k determines r. The value s depends on (h, d, r, k). d is always the same and with the static k, r and k will also always be the same. So s only changes when h is different.

(e) Yes, an adversary can recover both the private key d if k is re-used. So an adversary can sign any message M using two different valid signature pairs.

The retrieval of d is shown below:

$$h_1 = H(m_1)$$
$$h_2 = H(m_2)$$

For the first hash we have (r, s_1) and for the second hash (r, s_2).
If you solve d given s_1 and s_2 in WolframAlpha, you get the first equation.

$$\begin{aligned} d &= \frac{s_2 \cdot h_1 - s_1 \cdot h_2}{r(s_1 - s_2)} \\ &= \frac{h_1 \cdot h_2 + r \cdot h_1 \cdot d - h_1 \cdot h_2 - r \cdot h_2 \cdot d}{r \cdot h_1 \cdot r \cdot d - r \cdot h_2 - r \cdot d} \\ &= \frac{r \cdot h_1 \cdot d - r \cdot h_2 \cdot d}{r \cdot h_1 - r \cdot h_2} \\ d &= d \end{aligned}$$

(f)
```
sage: T_x = 32560625091655743179598362635611063129400 8...
...1157278488055600233871679272335 04
sage: T_y = 32026303591712962751241307547882981359278 2...
...1271791023669770631650376492250 0307
sage: T = E(T_x, T_y)
sage: Qohno = Qb + T


sage: r = mod(193703920910737566124082452472 03...
...597375267588932915277381795020274657218589 73, q)
sage: s = mod(237871765932909663400314091483268 182...
...570408279895507215238702918363472600614 1, q)

sage: 0 < int(r), int(s) < q
(True, True)
sage: u1 = h*s^-1; u2 = r * s^-1
sage: X = int(u1) * G + int(u2) * Qohno
sage: X!=E(0)
True
sage: mod(X[0], q) == r
True
```

(g) T is one of the special points with a very low order $<= 8$ that can be used as public key. All of these points, if you multiply them by u_2, you get point at infinity.

```
sage: T.order()
8
sage: int(u2)*T
(0 : 1 : 0)
```

Other points can be found using (from the SageMath documentation):

```
sage: P = E(0)
sage: P.division_points(8)
```

This means that the point $Q_{\text{oh no}}$ also works as a public key, because $u_2 \cdot Q_{\text{oh no}} = u_2 \cdot Q_b + u_2 \cdot T = u_2 \cdot Q_b + \mathcal{O} = u_2 \cdot Q_b$.

(h) Using the private key of Q_b, an attacker can sign a transaction and send it together with Q_b. Then, the attacker can spend his coins again by sending the same signed transaction together with $Q_{\text{oh no}}$.

(i) By checking if both the signature and public key has been used.

(j) As much as you can forge cryptos by breaking/misusing bad crypto and sell it unnoticeably.