Homework 5

Task 1 - Group Generators (8 points)

(a)

Since $a \in \mathbb{Z}_m^*$, by the inverse property of the multiplicative groups, we know $\exists\ a^{-1}$ such that $a^{-1} \cdot a = 1$.
Given the equation $ax \equiv b \pmod{m}$, we can multiply both sides by a^{-1} to get:

$$a^{-1} \cdot a \cdot x \equiv a^{-1} \cdot b \pmod{m}$$

which simplifies to:

$$x \equiv a^{-1} \cdot b \pmod{m}.$$

This shows that for any $b \in \mathbb{Z}_m$, the value $x = a^{-1} \cdot b \pmod{m}$ is a solution to the modular equation $ax \equiv b \pmod{m}$.
Thus, we have shown that there always exists a solution to the modular equation $ax \equiv b \pmod{m}$ for any $b \in \mathbb{Z}_m$, given that $a \in \mathbb{Z}_m^*$ and $m \geq 2$.

(b)

Since $a \in \mathbb{Z}_m \setminus \mathbb{Z}_m^*$, we know that a is not coprime to m (i.e. $\gcd(a, m) > 1$).
Let $d = \gcd(a, m)$. We choose a $b \in \mathbb{Z}_m$ that is not divisible by d.
Our goal is to show that for this b, the equation $ax \equiv b \pmod{m}$ has no solution in terms of x.

Assume for contradiction that there exists an $x \in \mathbb{Z}_m$ such that $ax \equiv b \pmod{m}$, which means that m divides $ax - b$.
Since d divides a and m, it must also divide ax. However, b is chosen such that it is not divisible by d, therefore $ax - b$ cannot be divisible by m, since m is divisible by d.
This is a contradicts to our assumption that such an x exists.
Thus, for any $b \in \mathbb{Z}_m$ that is not divisible by d, the equation $ax \equiv b \pmod{m}$ has no solution.
Therefore, there exists a $b \in \mathbb{Z}_m$ such that the modular equation $ax \equiv b \pmod{m}$ has no solution when $a \in \mathbb{Z}_m \setminus \mathbb{Z}_m^*$.

(c)

Assume h is a generator of $\mathbb{G}$ where $h = g^a$, and $a \in \mathbb{Z}_m$.
By the definition of a generator, we know that the set $\{h^x : x \in \{0, 1, \ldots, |\mathbb{G}| - 1\}\} = \mathbb{G}$.
Since g is a generator of $\mathbb{G}$, for some $b \in \mathbb{Z}_m$ and $x \in \{0, 1, \ldots, |G| - 1\}$, we have

$$h^x \equiv g^{ax} \equiv g^b \pmod{m}.$$

For h to be a generator, the equation $ax \equiv b \pmod{m}$ must have a solution for every $b \in \mathbb{Z}_m$.
From part (a) and part (b), we know that the equation $ax \equiv b \pmod{m}$ has a solution for all $b \in \mathbb{Z}_m$ iff $a \in \mathbb{Z}_m^*$ (since a and m must be coprime).
Thus, $h = g^a$ is a generator of $\mathbb{G}$ iff a is an element of $\mathbb{Z}_m^*$.
Since the number of elements in $\mathbb{Z}_m^*$ is $\phi(m)$ (by Euler's totient function), we know that there are exactly $\phi(m)$ such a's in $\mathbb{Z}_m$ for which $h = g^a$ is a generator.
Therefore, there are exactly $\phi(m)$ generators in $\mathbb{G}$.

(d)

When m is prime, every element of $\mathbb{G}$ (other than the identity 1) is a generator of the group.
Since m is prime, $\phi(m) = m - 1$.
In a cyclic group $\mathbb{G}$, every element can be represented as g^a for some integer a where $0 \leq a < m$, and g^0 is the identity element.
For every a in $1 \leq a < m$, a is coprime with m since m is prime.
From part (c), we know that an element $h = g^a$ is a generator of $\mathbb{G}$ if and only if a is coprime with m.
Since every a in the range $1 \leq a < m$ satisfies this condition, every such $h = g^a$ is a generator.
Therefore, all $m - 1$ elements of G other than the identity element are generators.

Task 2 - Group Order Factorization & Discrete Logarithms (15 points)

(a)

Since the elements in $\langle g^\alpha \rangle$ are $\{g^\alpha, g^{2\alpha}, g^{3\alpha}, \ldots\}$, to know the order of $\langle g^\alpha \rangle$, we need to find the smallest positive integer i such that $(g^\alpha)^i = 1$.
Since g is a generator of $\mathbb{G}$, we have $g^m = 1$, where 1 is the identity element in $\mathbb{G}$.
Given $m = \alpha \cdot n$, we know that $g^{\alpha n} = g^m = 1$.
Since for any $i' < n$, $\alpha \cdot i' < \alpha \cdot n = m$, we know that $g^{i'\alpha} \neq 1$.
Since $g^{\alpha n} = 1$ and $g^{i'\alpha} \neq 1$ for all $i' < n$, the smallest i for which $(g^\alpha)^i = 1$ is n.
Therefore, the order of $\langle g^\alpha \rangle$ is n.

(b)

Since $X = g^x$, we have $X^\alpha = (g^x)^\alpha = g^{x\alpha}$.
Since the discrete logarithm of X to the base g^α is the smallest non-negative integer i such that $(g^\alpha)^i = X$, our goal is to find i such that $(g^\alpha)^i = X = g^{x\alpha}$.
Since $(g^\alpha)^i = g^{\alpha i} = g^{x\alpha}$ and $|\mathbb{G}| = m$, we know that $\alpha i \equiv x\alpha \pmod{m}$.
From part (a) we know that the group $\langle g^\alpha \rangle$ has order n, and since $m = \alpha \cdot n$, we can simplify $\alpha i \equiv x\alpha \pmod{m}$ to $i \equiv x \pmod{n}$
Since i is a discrete logarithm, it has to be between 0 and $n - 1$.
Therefore, the discrete logarithm of X^α to the base g^α is the smallest non-negative integer i such that $i \equiv x \pmod{n}$, where $0 \leq i \leq n - 1$ (i.e. i is the remainder of x divided by n).

(c)

Let $\mathbb{G}$ be the group $\mathbb{Z}_p^*$, and g be a generator of $\mathbb{G}$.
Since p is prime, we know that $\mathbb{Z}_p^*$ has order $p-1$.
We first need to construct a function that takes $X = g^x$ and compute the discrete logarithm x in group $\mathbb{G}$ using Baby-Step Giant-Step:

function BSGS(X):
$m \leftarrow |\mathbb{G}|$
$k \leftarrow \lceil \sqrt{m} \rceil$
For $q = 0$ **to** $k-1$ **do**
    $G[q] \leftarrow g^{q \cdot k}$
For $r = 0$ **to** $k-1$ **do**
    If $\exists q : G[q] = X \cdot g^{-r}$ **then return** $x = q \cdot k + r$

We can then construct a distinguisher that calls the function BSGS():

distinguisher $D^{\mathsf{DDH}_{b[\mathbb{G},g]}}$:
$(X, Y, Z) \leftarrow \mathsf{DHH}_{b[\mathbb{G},g]}$
$x \leftarrow \mathsf{BSGS}(X)$
$y \leftarrow \mathsf{BSGS}(Y)$
If $Z \neq g^{xy}$ **then return** 1
else return 0

Note that $\Pr[D^{\mathsf{DDH}_{0[\mathbb{G},g]}} \Rightarrow 1] = 0$, because the BSGS function, in this case, will return x and y such that $Z = g^{xy}$.
Also, $\Pr[D^{\mathsf{DDH}_{1[\mathbb{G},g]}} \Rightarrow 1] = 1 - \frac{1}{|\mathbb{G}|} = 1 - \frac{1}{p-1}$, because in this case, $Z = g^{xy}$ only when the discrete log z of $Z = g^z$ collides with the product of x and y which has probability $\frac{1}{|\mathbb{G}|} = \frac{1}{p-1}$.
Therefore, we have

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g} = |\Pr[D^{\mathsf{DDH}_{0[\mathbb{G},g]}} \Rightarrow 1] - \Pr[D^{\mathsf{DDH}_{1[\mathbb{G},g]}} \Rightarrow 1]| = |0 - (1 - \frac{1}{p})| = 1 - \frac{1}{p}$$

which is non-negligible.
Since the Baby-Step-Giant step function runs in time polynomial in $\log(p)$, the distinguisher D also runs in time polynomial in $\log(p)$.
Therefore, the DDH assumption cannot be true in $\mathbb{Z}_p^*$ for a prime p.

(d)

- For each prime p_i, calculate the discrete logarithm x_i of $X \equiv g^{x_i} \pmod{p_i}$ using Baby-Step Giant-Step similar to part(c) (note $i = 1, ..., 44$):

function BSGS(X):
$x \leftarrow$ empty array
For i $= 1$ to 44 **do**
    $m \leftarrow |\mathbb{Z}_{p_i}^*|$
    $k \leftarrow \lceil \sqrt{m} \rceil$
    For $q = 0$ **to** $k - 1$ **do**
        $G[q] \leftarrow g^{q \cdot k}$
    For $r = 0$ **to** $k - 1$ **do**
        If $\exists q : G[q] = X \cdot g^{-r}$ **then** $x[i] = q \cdot k + r$
return x

Since the complexity of each discrete logarithm calculation is $O(\sqrt{p_i}\log(p_i))$, we need roughly $44 \times \sqrt{193} \times \log(193) \approx 44 \times 31.8 \approx 1408$ group exponentiations to calculate all 44 discrete logarithms.

- Since the Chinese Remainder Theorem states that if we have a set of congruences $x \equiv x_i \pmod{p_i}$ for $i = 1, \ldots, 44$, then there is a unique $x \pmod{m}$ that satisfies all these congruences.
Once we have x_i for each p_i, we can use CTR to reconstruct $x \pmod{m}$, and the reconstruction of $x \pmod{m}$ from $x_1 \pmod{p_1}, \ldots, x_{44} \pmod{p_{44}}$ should be efficient.

- Once we finished reconstructing $x \pmod{m}$, we can return x as the discrete logarithm x of $X = g^x$ in the group $\mathbb{G}$.
Since total number of group exponentiations is mostly discrete logarithm calculations and we are only performing these calculations for 44 different primes, the total number of operations, as shown above, should be in the order of a few thousand.

Task 3 - A Liitle Number Theory (5 points)

(a)

By the definition of the Euler's totient function, since P and Q are prime and $N = PQ$, we know that

$$\phi(N) = \phi(PQ) = (P-1)(Q-1)$$

We can simply it further:

$$\begin{aligned}
\phi(N) &= (P-1)(Q-1) \\
\phi(N) &= PQ - P - Q + 1 \\
\phi(N) &= N - P - \frac{N}{P} + 1 && \text{Since } N = PQ,\ Q = \tfrac{N}{P} \\
\phi(N) \cdot P &= N \cdot P - P^2 - N + P && \text{Multiply both sides by } P \\
0 &= -P^2 + N \cdot P - \phi(N) \cdot P + P - N && \text{Move all terms to the left side} \\
0 &= -P^2 + (N - \phi(N) + 1) \cdot P - N && \text{Factor out } P
\end{aligned}$$

We can then use the quadratic formula to solve for P:

$$P = \frac{-(N - \phi(N) + 1) \pm \sqrt{(N - \phi(N) + 1)^2 - 4 \cdot (-1) \cdot (-N)}}{2 \cdot (-1)}$$

Once we have P, we can find Q by plugging P into $Q = \frac{N}{P}$. Since all the calculation described above including the square root operation can be implemented in polynomial time of the bitlength of N ($\log N$), we know the algorithm can efficiently find P and Q given N and $\phi(N)$ (from the black box algorithm) in time polynomial in $\log N$.
Therefore, we have shown we can efficiently factor N given N and the black box algorithm that computes $\phi(N)$.

Task 4 - Collision-Resistance from Discrete Logarithms (8 points)

(a)

By the definition of the hash function, we know that $\mathsf{H}(S,(x_1,x_2)) = g^{x_1}S^{x_2}$, and $\mathsf{H}(S,(x_1',x_2')) = g^{x_1'}S^{x_2'}$.
Given a collision $H(S,(x_1,x_2)) = H(S,(x_1',x_2'))$, we know that $g^{x_1}S^{x_2} = g^{x_1'}S^{x_2'}$.
Since $S = g^x$ for some unknown x, we can rewrite the equation in terms of x:

$$\begin{aligned} g^{x_1}S^{x_2} &= g^{x_1'}S^{x_2'} \\ g^{x_1}(g^x)^{x_2} &= g^{x_1'}(g^x)^{x_2'} && \text{Since } S = g^x \\ g^{x_1+x\cdot x_2} &= g^{x_1'+x\cdot x_2'} \end{aligned}$$

Now, we can use the fact that $\mathbb{G}$ is of prime order p. In a group of prime order p with generator g, we know that $g^p = 1$ (the identity element). Also, since the group is cyclic, we can reduce the exponents by modulo p without altering the group element it represents. Therefore, we can reduce the equation to:

$$\begin{aligned} x_1 + x\cdot x_2 &\equiv x_1' + x\cdot x_2' \pmod p \\ x_1 - x_1' &\equiv x\cdot(x_2' - x_2) \pmod p \\ x &\equiv (x_1 - x_1')(x_2' - x_2)^{-1} \pmod p && \text{Since } x_2' - x_2 \text{ is coprime with } p \end{aligned}$$

Therefore, we have shown that given a collision $H(S,(x_1,x_2)) = H(S,(x_1',x_2'))$, we can efficiently compute x such that $S = g^x$ simply by calculating $(x_1 - x_1')(x_2' - x_2)^{-1} \pmod p$.

(b)

To find a collision for the hash function H for a chosen $S = g^x$, our goal is to show that it's possible to find two different inputs $(x_1, x_2) \neq (x_1', x_2')$ such that $H(S, (x_1, x_2)) = H(S, (x_1', x_2'))$. (i.e. show the reverse of part (a) is true.)

Since the hash function $H : G \times \mathbb{Z}_p^2 \to G$ is defined as $H(S, (x_1, x_2)) = g^{x_1} S^{x_2}$. If the malicious entity sets $S = g^x$, the hash function becomes $H(g^x, (x_1, x_2)) = g^{x_1}(g^x)^{x_2} = g^{x_1 + x x_2}$.
The entity knows x and can manipulate x_1 and x_2 to create a collision. For example, they can choose two different pairs (x_1, x_2) and (x_1', x_2') such that:

$$g^{x_1 + x x_2} = g^{x_1' + x x_2'}$$

Using the result from the part(a), we know that if a collision exists:

$$x_1 + x x_2 \equiv x_1' + x x_2' \pmod{p}$$

Since the adversary knows x, they can easily find pairs (x_1, x_2) and (x_1', x_2') that satisfy this equation. One option could be to choose x_2 and x_2' arbitrarily and then compute x_1 and x_1':

$$x_1 = x_1' + x x_2' - x x_2 \pmod{p}$$

Since the malicious entity controls x and can freely choose x_2 and x_2', they can manipulate the equation to find a collision.
Therefore, we have shown that if the seed S is chosen maliciously as g^x for a known x, the hash function H effectively has a backdoor to create collisions.

Task 5 - Security of Key-Agreement Protocols (9 points)

(a)

From the key-generation protocol, we know the following:

$$s = k \oplus r$$
$$u = s \oplus t$$
$$w = u \oplus r$$

To show that Alice and Bob output the same key, we need to show that $k = w \oplus t$.
We can start by substituting w with $u \oplus r$:

$$\begin{aligned} w \oplus t &= u \oplus r \oplus t && \text{Substitute } w \text{ with } u \oplus r \\ &= s \oplus t \oplus r \oplus t && \text{Substitute } u \text{ with } s \oplus t \\ &= k \oplus r \oplus t \oplus r \oplus t && \text{Substitute } s \text{ with } k \oplus r \\ &= k \oplus r \oplus r \oplus t \oplus t && \text{Rearrange terms} \\ &= k && \text{Since } r \oplus r = 0 \text{ and } t \oplus t = 0 \end{aligned}$$

Therefore, we have shown that $k = w \oplus t$, and Alice and Bob output the same key.

(b)

Assume we have the following oracles defined on the key-agreement protocol described in the question:

oracle LR_0[KA]:

private **procedure** Init():
$K, r, t \xleftarrow{\$} \{0,1\}^n$
$S \leftarrow K \oplus r$
$U \leftarrow S \oplus t$
$W \leftarrow U \oplus r$
return (S, U, W, K)

oracle LR_1[KA]:

private **procedure** Init():
$K, r, t \xleftarrow{\$} \{0,1\}^n$
$S \leftarrow K \oplus r$
$U \leftarrow S \oplus t$
$W \leftarrow U \oplus r$
$K' \xleftarrow{\$} \{0,1\}^n$
return (S, U, W, K')

With the above oracle we can construct the following distinguisher:

distinguisher $D^{\mathsf{LR}_{b[\mathsf{KA}]}}$:
$(S, U, W, K) \leftarrow \mathsf{LR}_{b[\mathsf{KA}]}$
If $S \oplus U \oplus W \oplus K = 0$ **then return** 1
else return 0

Note that $\Pr[D^{\mathsf{LR}_{0[\mathsf{KA}]}} \Rightarrow 1] = 1$, because by the definition, we know the following:

$$\begin{aligned} S \oplus U \oplus W \oplus K &= (K \oplus r) \oplus (K \oplus r \oplus t) \oplus (K \oplus r \oplus t \oplus r) \oplus K \\ &= K \oplus r \oplus K \oplus r \oplus t \oplus K \oplus r \oplus t \oplus r \oplus K \\ &= K \oplus K \oplus K \oplus K \oplus r \oplus r \oplus r \oplus r \oplus t \oplus t \\ &= 0 \end{aligned}$$

So the distinguisher will always return 1.
However, $\Pr[D^{\mathsf{LR}_{1[\mathsf{KA}]}} \Rightarrow 1] = \frac{1}{2^n}$, because we know the following:

$$\begin{aligned} S \oplus U \oplus W \oplus K' &= (K \oplus r) \oplus (K \oplus r \oplus t) \oplus (K \oplus r \oplus t \oplus r) \oplus K' \\ &= K \oplus K \oplus K \oplus K' \oplus r \oplus r \oplus r \oplus r \oplus t \oplus t \\ &= K \oplus K' \end{aligned}$$

Since K and K' are chosen uniformly at random, the probability that $K \oplus K' = 0$ is $\frac{1}{2^n}$.
Therefore we have

$$\mathsf{Adv}^{\mathsf{LR}}_{\mathsf{KA}} = |\Pr[D^{\mathsf{LR}_{0[\mathsf{KA}]}} \Rightarrow 1] - \Pr[D^{\mathsf{LR}_{1[\mathsf{KA}]}} \Rightarrow 1]| = |1 - \frac{1}{2^n}| = 1 - \frac{1}{2^n}$$

which is not negligible.
Therefore, we have shown that the key-agreement protocol is not secure because (S, U, W, K) and (S, U, W, K') are easily distinguishable.