CS698G
Assignment 2

Md Arsal Asif

Indian Institute of Technology, Kanpur

Question 1

Question (a)

Alice wants to send a message M to both Bob and Carlo, so she calculates $C_B = M^{e_B} \mod n$ and $C_C = M^{e_C} \mod n$. Explain how an adversary can recover the message M without knowing private keys d_B and d_C.

Solution:
Given: e_B and e_C are co-prime. This implies that, by extended Euclidean GCD, we can write

$$x \cdot e_B + y \cdot e_c \equiv 1 \mod n$$

$$\Rightarrow M^{xe_B} \cdot M^{ye_C} \equiv M \mod n$$

$$\Rightarrow \boxed{C_B^x \cdot C_C^y \equiv M \mod n}$$

x and y can be calculated using public keys e_B and e_C. M can be computed using x, y, C_B and C_C.

Question (b)

Carlo is interested in messages sent to Alice. Outline a strategy that may help Carlo recover an alternative private key d'_A that can perform the same function as d_A to decrypt messages sent to Alice.

Solution:

$$e_C d_C \equiv 1 \pmod{\phi(n)},$$

Carlo can define a K such that

$$K = e_C d_C - 1$$

which satisfies $K = m\phi(n)$ for some integer m. We can also compute the inverse of e_A in field of K and call it d'_A. Thus we can write the expression of Extended Euclidean gcd here as,

$$e_A d'_A \equiv 1 \pmod{K}$$

Assumption: K and d_A are coprime

$$e_A d'_A = jK + 1 = (jm)\phi(n) + 1$$

so

$$\boxed{e_A d'_A \equiv 1 \pmod{\phi(n)}}$$

Question 2

Question (a)

Does this hash function satisfy each of the following requirements?

(i)

Yes, the output size is fixed. The output of the function is modulo n, and since n is fixed, the hash output will be an integer in the range $[0, n-1]$. Therefore, the output size is determined by the bit-length of n, which is constant.

(ii)

Yes, the function is efficient. The XOR operation is linear in the number of blocks and can be computed very quickly. The RSA exponentiation $x^e \mod n$ can also be computed efficiently using modular exponentiation techniques (e.g., square-and-multiply), assuming e is small.

(iii)

No, it is not preimage resistant. To find a message M such that $H(M) = h$, an attacker only needs to compute:

$$X = h^d \mod n$$

where d is the private RSA key. If the attacker has access to the private key (or if the RSA problem is not hard), the preimage is easily computed. Moreover, since XOR is linear and reversible, it's easy to find $M_1, M_2, \ldots, M_m$ such that their XOR equals X.
Requirement **not** satisfied.

(iv)

No, the function is not second preimage resistant. Given a message M and its hash value h, one can compute the XOR of its blocks:

$$X = M_1 \oplus M_2 \oplus \ldots \oplus M_m$$

Now, to find another message $M' \neq M$ that gives the same hash, construct another set of blocks with the same XOR value X. Since many different combinations of inputs can yield the same XOR, multiple second preimages exist.
Requirement **not** satisfied.

(v)

No, the function is not collision resistant. There are many different inputs that can yield the same XOR value. Since the final hash depends solely on the XOR of all blocks, different message sets with the same XOR will produce the same hash after exponentiation. For example, $M = (1, 2, 3)$ and $M' = (0, 1, 0, 4)$ might have the same XOR.
Requirement **not** satisfied.

Question (b)

Explain how to efficiently find collisions in the following hash functions

We want to find a collision: two distinct pairs (x_1, y_1) and (x_2, y_2) such that:

$$H_a(x_1, y_1) = H_a(x_2, y_2)$$

(i)

$$H_a(x, y) = F(y, x \oplus y) \oplus y$$

Observation: The function uses y as the key and $x \oplus y$ as the plaintext to the block cipher F. Then the result is XORed with y again.
Let us fix $y_1 = y_2 = y$, and find $x_1 \neq x_2$ such that:

$$F(y, x_1 \oplus y) \oplus y = F(y, x_2 \oplus y) \oplus y \Rightarrow F(y, x_1 \oplus y) = F(y, x_2 \oplus y)$$

This reduces to finding a **collision in the block cipher** with fixed key y — i.e., find two different plaintexts $p_1 \neq p_2$ such that $F(y, p_1) = F(y, p_2)$. This is feasible using a birthday attack in time approximately 2^{128} due to the 256-bit output size.

(ii)

$$H_b(x, y) = F(y \oplus x, x)$$

Fix $x_1 = x_2 = x$ and find $y_1 \neq y_2$ such that:

$$F(y_1 \oplus x, x) = F(y_2 \oplus x, x) \Rightarrow F(k_1, x) = F(k_2, x)$$

So again, this is a collision in F with fixed plaintext x and differing keys $k_1 \neq k_2$. This can be attacked similarly by finding two keys $k_1 \neq k_2$ such that:

$$F(k_1, x) = F(k_2, x)$$

(iii) $H_c(x, b) = H(x)$ if $b = 0$, and $H(H(x))$ if $b = 1$

Here, H is a standard collision-resistant hash function mapping arbitrary-length inputs to 256-bit values. We want to find $(x_1, b_1) \neq (x_2, b_2)$ such that:

$$H_c(x_1, b_1) = H_c(x_2, b_2)$$

Observation: Since b only controls whether H is applied once or twice, try:

$$H(x) = H(H(x'))$$

Efficient Strategy: Choose a value x, compute $H(x)$, and then search for x' such that:

$$H(H(x')) = H(x) \Rightarrow H(x) = H(H(x'))$$

This is equivalent to finding a **preimage of H under itself**, i.e., solving:

$$x = H(x') \Rightarrow x' = H^{-1}(x)$$

Assuming H is preimage-resistant, this is hard.
However, if we can find a fixed point x such that:

$$H(x) = x \Rightarrow H_c(x, 0) = H_c(x, 1)$$

Then, $(x, 0)$ and $(x, 1)$ will collide..

Question 3

Given: q is a large prime, and a be a primitive element in the finite field GF(q). Alice's public key is $y_A = a^{x_A} \mod q$, where x_A is her private key such that $1 < x_A < q - 1$. The hash function H is assumed to be publicly known.

Question (a(i))

Describe the Decryption Function D(C1,C2) that Alice can use to recover the message

Solution:

$$\boxed{\text{K} = (a^k)^{x_A} \mod q}$$

$$\boxed{\text{M} = \text{C2} \oplus K}$$

x_A is known to Alice.

Question (a(ii))

Show how the security of the encryption function is based on Computational Diffie-Hellman (CDH) problem

Solution: The message is encrypted using the shared key a^{xy}, which is derived from the combination of the sender's ephemeral secret y and the recipient's long-term private key x. To recover the original message M, an attacker would need to isolate and remove the factor a^{xy} from the ciphertext. However, doing so requires knowledge of the private exponent x (or y), since a^{xy} is not directly accessible from the public values a^x and a^y. Extracting x from a^x, or y from a^y, involves solving the discrete logarithm problem, which is widely regarded as computationally infeasible in appropriately chosen groups.

Question (b(i))

What are the signing and verification equations?

Solution:
The signature for a message M uses the hash value $m = H(M)$. Alice chooses a random k (such that $\gcd(k, q-1) = 1$), and computes:

$$S_1 = a^k \mod q$$

Then, she computes:

$$mS_2 + x_A S_1 \equiv k \pmod{q-1}$$

The signature is the pair (S_1, S_2) attached to message M.
Signing Equations:

$$\boxed{\text{S}_1 = a^k \mod q}$$

$$\boxed{\text{S}_2 \equiv m^{-1}(k - x_A S_1) \mod (q-1)}$$

Verification Equation:
Given message M, signature (S_1, S_2), and public key y_A, the verifier checks:

$$a^k \equiv a^{mS_2 + x_A S_1} \pmod{q}$$

$$\Rightarrow \boxed{\text{S}_1 \equiv a^{mS_2 + x_A S_1} \pmod{q}}$$

Or equivalently, since $S_1 = a^k$ and $y_A = a^{x_A}$:

$$\boxed{\text{S}_1 \overset{?}{\equiv} a^{mS_2} \cdot y_A^{S_1} \mod q}$$

Question (b(ii))

Is this scheme secure? Justify your answer with reasons.

The security of this ElGamal variant relies on the difficulty of the **discrete logarithm problem (DLP)** in the multiplicative group of GF(q), and the cryptographic properties of the hash function H.
Justification:

- If k is chosen randomly and kept secret, then recovering x_A from the signature remains as hard as solving DLP.
- The use of a hash function H ensures that signature forgery on raw messages is difficult (assuming H is collision-resistant and pre image resistant).

Question 4

Question (a)

How do A and B know that the key is freshly generated?

Solution: To ensure freshness, the protocol must use a **nonce** - a random number used only once. In the Needham–Schroeder protocol, this is typically achieved by including a nonce generated by A (denoted N_A) in the initial request and then receiving it back encrypted with the newly generated session key.
This confirms to A that the message was constructed *after* the nonce was generated and thus that the session key K_{AB} is fresh. Similarly, B receives the key within a message that contains the nonce N_B ensuring it was generated recently. Note that KDC is trusted is a huge point.

Question (b)

How could A and B know that the key is not available to other users in the system?

A and B can be confident that the session key K_{AB} is not available to anyone else because:

- The key is generated by the **trusted KDC**, which is assumed to maintain secrecy and not reveal session keys to unauthorized parties.
- The session key K_{AB} is transmitted to A and B in **separate encrypted messages**:

  $$E(K_B, [K_S||ID_a||N_B]||E(K_A, [K_S||ID_b||N_b]$$

  These are encrypted under the long-term keys K_A and K_B, which are known only to A, B, and the KDC. Unless the long-term keys are compromised, the communication is safe.

Question (c)

At this stage, A and B cannot authenticate with each other. Explain why and extend the scheme with a few steps so that A and B can authenticate with each other

At this point, A and B share the session key K_{AB}, but:

- B cannot be sure the message came from A - it might have been replayed by an attacker.
- A cannot confirm B actually received the key or is who they claim to be.

Solution:

1. **A** sends a nonce N_1 to **B**, encrypted under K_{AB}:

$$A \to B : E(K_{AB}, N_A)$$

2. **B** responds with the same nonce $N_A$along with a new nonce N_B:

$$B \to A : E(K_{AB}, N_A || N_B)$$

3. **A** responds with the new nonce to complete mutual authentication:

$$A \to B : E(K_A B, N_B)$$

Outcome: Both A and B are assured that they are communicating with the intended party who possesses the shared session key K_{AB}.