Assignment 02

231120

Ques1 :1

Key Assumption: We assume that $\gcd(c, n) = 1$, because otherwise $\gcd(c, n)$ would be either p or q, which would break RSA itself.

Since we also know that $\gcd(e_c, e_b) = 1$, by the Extended Euclidean Algorithm, there exist integers x and y such that:

$$e_b x + e_c y = 1$$

Now, given the ciphertexts:

$$C_B = M^{e_b} \bmod n \quad \text{and} \quad C_C = M^{e_c} \bmod n,$$

we can compute:

$$C_B^x \cdot C_C^y \equiv M^{e_b x} \cdot M^{e_c y} = M^{e_b x + e_c y} \equiv M \pmod{n}.$$

Thus, we recover the original message M. If either x or y is negative, we compute the modular inverse of the corresponding ciphertext (e.g., $C_B^{-1} \bmod n$) to evaluate the expression correctly.

Ques1 :2

We are given that $\gcd(e_a, e_c) = 1$ and $\gcd(e_c, d_c) = 1$. This implies:

$$e_c \cdot d_c \equiv 1 \pmod{\phi(n)}$$

Let us write:

$$e_c d_c - 1 = k \cdot \phi(n)$$

Now, define a modified totient:

$$\phi' = e_c d_c - 1$$

We compute:

$$d'_a = e_a^{-1} \bmod \phi(n)'$$

Assumption involved is $gcd(e_a, \phi(n)') == 1$**;**

Verification of Decryption Key

We now verify that d'_a behaves as a valid private exponent for e_a. We compute:

$$e_a d'_a = 1 + r \cdot \phi(n)' = 1 + r \cdot k \cdot \phi(n) = 1 + \alpha \cdot \phi(n)$$

Thus:

$$e_a d'_a \equiv 1 \pmod{\phi(n)}$$

Therefore, d'_a can be used as a valid decryption exponent for e_a.

Ques2 :A 1

If We are assume that we are counting the number of leading zero bits in the output then the output of $H(m)$ will be of fixed size, which corresponds to the total number of bits in n.

Since $H(m) < n$, the maximum number of bits required to represent $H(m)$ is:

$$\lfloor \log_2 n \rfloor + 1$$

Therefore, the output length of $H(m)$ is at most $\lfloor \log_2 n \rfloor + 1$ bits.

Ques2 :A 2

A series of XOR operations can be performed in very little time, typically in linear time with respect to the input size. In contrast, RSA exponentiation is computationally expensive, especially when large keys are involved.

Therefore, the overall efficiency of the system is difficult to evaluate precisely, as it is dominated by the cost of RSA exponentiation rather than the lightweight XOR operations.

Ques2 :A 3

As long as RSA remains unbroken, recovering the original message m from its hash $H(m)$ is computationally infeasible due to the hardness of the RSA problem.

Therefore, given only $H(m)$, it is not possible to efficiently compute m. This implies that the hash function H satisfies the property of **preimage resistance**.

Ques3 :A 4

Consider the case where a string of zeros, denoted by 0^k, is appended to the message M, where k is equal to the block size. Since the XOR operation is used in the hash construction, we have:

$$H(M \parallel 0^k) = H(M)$$

because

$$M_i \oplus 0^k = M_i.$$

This means that two different inputs, m and $m \parallel 0^k$, produce the same hash output:

$$H(M) = H(M \parallel 0^k).$$

Thus, the function fails to satisfy **second preimage resistance**, as it is possible to find a distinct input with the same hash.

Ques2 :A 5

Since the hash function is not second preimage resistant, it also cannot be collision resistant. This is because **collision resistance** implies **second preimage resistance**.Therefore, the above hash function **is not collision resistant**.

Ques2 :B 1

Let $x, y \in \{0,1\}^{256}$. Here is an approach that can be used to efficiently find a collision in the hash function H_a. Suppose we have an input pair (x_1, y_1) and we know the corresponding output $H_a(x_1, y_1)$. Now, take any random $y_2 \neq y_1$, and our goal is to find x_2 such that:

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

We start from the definition:

$$H_a(x_1, y_1) = F(y_1, x_1 \oplus y_1) \oplus y_1$$

$$H_a(x_2, y_2) = F(y_2, x_2 \oplus y_2) \oplus y_2$$

Set these two outputs equal:

$$F(y_1, x_1 \oplus y_1) \oplus y_1 = F(y_2, x_2 \oplus y_2) \oplus y_2$$

Let

$$t = F(y_1, x_1 \oplus y_1) \oplus y_1$$

Then the equation becomes:

$$t \oplus y_2 = F(y_2, x_2 \oplus y_2)$$

Now invert the function F:

$$F^{-1}(y_2, t \oplus y_2) = x_2 \oplus y_2$$

$$F^{-1}(y_2, t \oplus y_2) \oplus y_2 = x_2$$

Hence, we have found distinct pairs $(x_1, y_1) \neq (x_2, y_2)$ such that:

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

which demonstrates a collision, violating the collision-resistance property of the hash function.

Ques2 :B 2

First, compute

$$H_b(x_1, y_1) = F(y_1 \oplus x_1, x_1) = t.$$

Now choose any y_2, and claim

$$t = F(y_2, x_2)$$

Find the corresponding x_2:

$$x_2 = F^{-1}(y_2, t)$$

Then we have:

$$H_b(x_2, x_2 \oplus y_2) = F(y_2 \oplus x_2 \oplus x_2, x_2) = F(y_2, x_2) = t$$

Thus, we have found two distinct inputs (x_1, y_1) and $(x_2 \oplus y_2, y_2)$ such that:

$$H_b(x_1, y_1) = H_b(x_2, x_2 \oplus y_2)$$

This results in a collision, violating the collision resistance property of the hash function H_b.

Ques2 :B 3

Now consider the following:

Let us take an arbitrary input $x \in \{0,1\}^{256}$. Then:

$$H_c(x, 1) = H(H(x))$$

Also consider the input $(H(x), 0)$. Then:

$$H_c(H(x), 0) = H(H(x))$$

Hence, we observe that:

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

But note that the inputs $(x, 1)$ and $(H(x), 0)$ are different (since $x \neq H(x)$ in general, assuming H is not the identity function).

Therefore, we have found a collision in H_c:

$$(x, 1) \neq (H(x), 0) \quad \text{but} \quad H_c(x, 1) = H_c(H(x), 0)$$

This contradicts the collision resistance of H_c. Thus, H_c **is not collision-resistant**, even though H is.

Ques3 :A 1

Given the ciphertext:

$$C_1 = a^k \mod q, \quad C_2 = K \oplus M$$

where

$$K = y_A^k = (a^{x_A})^k = a^{kx_A} \mod q$$

Since Alice knows her private key x_A, she can compute:

$$K = C_1^{x_A} \mod q = (a^k)^{x_A} \mod q = a^{kx_A} \mod q$$

Now, the original message M is recovered as:

$$M = C_2 \oplus K$$

Thus, the decryption function is:

$$D(C_1, C_2) = C_2 \oplus (C_1^{x_A} \mod q)$$

Ques3 :A 2

In the ElGamal variant encryption:

$$y_A = a^{x_A} \mod q, \quad C_1 = a^k \mod q, \quad K = y_A^k = a^{kx_A} \mod q$$

An adversary observing y_A and C_1 would need to compute:

$$K = a^{kx_A} \mod q$$

to break the encryption. However, this is exactly the CDH problem with inputs a^k and a^{x_A}, aiming to compute a^{kx_A}.

Conclusion: The security of this encryption scheme relies on the hardness of the CDH problem. If CDH is hard, then recovering the message without the private key is infeasible, ensuring the encryption's security.

Ques3 :B 1

Alice signs a message M using the equation:

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

Signing:

$$S_1 = a^k \mod q$$
$$S_2 \equiv (k - x_A S_1) \cdot m^{-1} \mod (q-1)$$

Verification: Given the signature (M, S_1, S_2), the verifier computes $m = H(M)$ and checks whether:

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

Ques3 :B 2

NO, Provided scheme is not secure. Surely, we cannot find k due to discrete logarithm problem but since we know

$$(k - x_A S_1) \equiv m S_2 \pmod{q-1}$$

we can use it to find a valid S_2' for any other message say m' whose gcd with $q-1$ is 1 as given below :

$$S_2' \equiv (k - x_A S_1) \cdot (m')^{-1} \mod (q-1) \Rightarrow m' S_2' + x_A S_1 \equiv k \pmod{q-1}$$

Thus, signature forgery is possible, proving the scheme is insecure.

Ques4: 1

A and B can be confident that the session keys are freshly generated due to the inclusion of the nonces N_a and N_b in the protocol messages. Since only A and B share their respective long-term keys k_a and k_b with the Key Distribution Center (KDC), an attacker without knowledge of these keys cannot forge valid messages.

If an attacker attempts a replay attack by resending an old message containing a previously used session key, the nonces N_a and N_b included in the new session messages will differ from those in the replayed message. The recipients A and B will notice this mismatch, indicating that the session keys are not freshly generated.

Thus, the freshness of the session keys is guaranteed by the presence and verification of the unique nonces, and without access to the long-term keys k_a and k_b, an attacker cannot replace or replay old keys as valid new session keys.

Ques4: 2

Until and unless A and B share their session keys or master keys with any other party, it is guaranteed that only they know the session keys. This is because A and B each share their respective long-term keys k_a and k_b exclusively with the Key Distribution Center (KDC).

When the KDC generates a session key, it encrypts this key separately with k_a and k_b and sends these encrypted session keys in two different messages to A and B respectively. Since only A knows k_a and only B knows k_b, no other party can decrypt these messages.

Thus, the confidentiality of the session keys is ensured, and only A and B can obtain and use the session keys.

Ques4: 3

Surely, A and B cannot authenticate each other in the current protocol because of replay and message forgery attacks. For example, A sends a packet:

$$\text{IDA} \parallel E(k_a, N_a)$$

to B, where $E(k_a, N_a)$ denotes the encryption of nonce N_a using key k_a.

An adversary can send B a previously captured packet:

$$\text{IDA} \parallel E(k_a, N_{\text{previous}})$$

which is a replay attack blocking IDA $\parallel$ $E(k_a, N_a)$ and b will not be able to find whether it is from A or someone else because he does not know the private key of A.

Moreover, the adversary can forge a message to A by concatenating a previous message from B to any other user, say:

$$\text{IDB} \parallel E(k_b, N_{\text{previous}})$$

with the intercepted packet(IDA $\parallel$ $E(k_a, N_a)$) from A. Then the adversary can send this concatenated message to the KDC and KDC will not be able to differentiate whether it is came from B or not. After that adversary will send the message (IDA $\parallel$ $E(k_a, [Ks||ID_b||N_a])$) received from KDC containing a valid nonce N_a to A without involving B in the full communication.

To mitigate this and ensure authentication, since only A and B can decrypt the session key, the following steps can be added:

1. A sends to B the encrypted nonce with the session key:

   $$E(k_s, N_c)$$

   where k_s is the session key and N_c is a challenge nonce.

2. B decrypts the message to recover N_c, applies a known function $f(\cdot)$ and sends back:

   $$E(k_s, f(N_c))$$

3. A decrypts the response and verifies whether $f(N_c)$ matches the expected value.

If the verification succeeds, A and B can be sure of each other's authenticity.

This protocol ensures mutual authentication by leveraging the secrecy of the session key and challenge-response mechanism.