ChatGPT - CSE

Cryptography Jan2025.pdf
PDF

see these pdf. and record it.

I have recorded the contents of the PDF "Cryptography Jan2025.pdf" for future reference. Let me know if you need any specific analysis, summary, or extraction of information from it.

Uploaded image
provide answer for this

The recommended approach is:

(b) Add error correction to the plaintext and then encrypt the modified plaintext.

Explanation:

1. **Encryption Scrambles Data**: If error correction is added after encryption, the ciphertext is usually random-looking data. Applying error correction at this stage would be ineffective because even small errors in transmission could completely alter the decrypted message.
2. **Error Correction Works Better on Structured Data**: Error correction techniques (such as Hamming codes or Reed-Solomon codes) rely on known structures in the data. If applied to plaintext before encryption, they can help recover the original message even if some bits are altered in transmission.
3. **Decryption Should Be Error-Free**: If errors occur in the ciphertext, decryption might fail completely, producing unreadable data. However, if error correction is added before encryption, the receiver can first decrypt the message and then use error correction to restore any corrupted data.

Conclusion:

Approach (b) ensures that error correction mechanisms can effectively restore the original message after decryption, making the communication more robust and reliable.

Uploaded image

This modification is a **bad idea** and does not improve OTP encryption. Here's why:

1. OTP Security Relies on Perfect Randomness

- The **One-Time Pad (OTP)** encryption scheme is perfectly secure because the key is chosen uniformly at random from all possible n-bit values.
- By excluding one possible key (the all-zero key), the distribution of keys is no longer uniform. Instead, it introduces a slight bias, making it possible for an attacker to exploit this weakness.

2. Information Leakage

- In the original OTP, each ciphertext is equally likely for any plaintext, making it impossible for an attacker to infer anything about the plaintext.
- By removing the zero key, the probability distribution of ciphertexts changes slightly, meaning some ciphertexts are less likely than others.
- This **small statistical deviation** can be used by an adversary to **infer information about the plaintext** over many encryptions.

3. Negligible Practical Gain

- The concern that using the zero key results in ciphertexts identical to plaintexts is unnecessary. This case happens with a probability of only $1/2^n$, which is negligible for large n.
- In real cryptographic applications, such small probabilities do not justify altering the fundamental randomness of OTP.

Conclusion:

- This modification **weakens** the perfect security of OTP rather than improving it.
- **OTP should use a truly uniform random key from all 2^n possibilities, including the all-zero key, to maintain its perfect secrecy.**

give a mathematical proof

To prove that modifying OTP by removing the zero key $k = 0^n$ weakens its security, let's analyze how this change affects the probability distribution of ciphertexts.

---

1. One-Time Pad (OTP) Security

In the standard OTP encryption scheme:

- The key k is chosen **uniformly** at random from the set $\{0, 1\}^n$ (all 2^n possible n-bit values).
- The encryption function is:

$$c = m \oplus k$$

- Given a ciphertext c, the probability of any plaintext m is **equal** for all possible plaintexts, i.e.,

$$P(C = c \mid M = m) = \frac{1}{2^n}$$

  for all c, ensuring perfect secrecy.

This means that no attacker can gain any information about m from c because every plaintext is equally likely for any given ciphertext.

---

2. Modified OTP and Its Impact

In the modified scheme:

- The key k is chosen uniformly at random, **excluding** the all-zero key $k = 0^n$. This means we now select from only $2^n - 1$ possible keys instead of 2^n.
- Encryption is still:

$$c = m \oplus k$$

Now, let's compute the probability of a given ciphertext given a plaintext:

$$P(C = c \mid M = m) = P(K = c \oplus m)$$

Since the key is now chosen from a **biased** set of size $2^n - 1$, the probability of choosing any specific key is:

$$P(K = k) = \frac{1}{2^n - 1}$$

for each $k \neq 0^n$. Therefore, the probability distribution of ciphertexts is now:

$$P(C = c \mid M = m) = \frac{1}{2^n - 1}$$

for each c, which is **not equal** to $\frac{1}{2^n}$ as in the original OTP.

---

3. Information Leakage

To see how this modification leaks information, consider an attacker who intercepts a ciphertext c. They now compute:

$$P(C = c)$$

Summing over all possible plaintexts:

$$P(C = c) = \sum_m P(C = c \mid M = m) P(M = m)$$

In the original OTP, this sum was constant for all c, ensuring perfect secrecy. But in the modified scheme, since each $P(C = c \mid M = m)$ is slightly larger due to the exclusion of $k = 0^n$, some ciphertexts become slightly more probable than others.

This deviation from uniformity introduces a small **bias** in the ciphertext distribution, which an attacker could exploit over many encryptions.

---

4. Conclusion: OTP's Perfect Secrecy is Broken

The Shannon definition of perfect secrecy states:

$$P(M = m \mid C = c) = P(M = m)$$

for all m, c. That is, learning c should not change the probability of any plaintext. However, in the modified scheme:

$$P(C = c) \neq \frac{1}{2^n}$$

so $P(M = m \mid C = c)$ **is no longer uniform**, violating perfect secrecy.

Thus, by excluding $k = 0^n$, the encryption introduces **a slight statistical dependence** between ciphertexts and plaintexts, making it possible to infer some information about the plaintext over many encryptions.

This proves mathematically that **removing the zero key weakens OTP security** and breaks its perfect secrecy.

Uploaded image

(a) How can you decrypt the ciphertext?

The encryption algorithm is defined as:

1. **Encryption Process:**
   - A random initialization value r_0 is chosen.
   - Each block m_i is encrypted using a function $E_k(m_i)$, producing an intermediate value r_i.
   - The ciphertext block c_i is computed as:

     $$c_i = r_i \oplus r_{i-1}$$

   - The full ciphertext consists of:

     $$c_0||c_1|| \dots ||c_t$$

     where $c_0 = r_0$.
2. **Decryption Process:** To decrypt, we recover m_i using:
   - Extract r_0 from c_0.
   - Compute each r_i iteratively:

     $$r_i = c_i \oplus r_{i-1}$$

   - Recover each plaintext block:

     $$m_i = D_k(r_i)$$

   - Continue this for all blocks to obtain the full plaintext.

(b) Show that the above mode is not CPA secure.

Chosen Plaintext Attack (CPA) Definition:
An encryption scheme is **CPA secure** if an attacker, even when allowed to encrypt arbitrary messages of their choice, **cannot gain any useful information** about the secret key or any future encrypted messages.

Attack on CPA Security

We show that an attacker can distinguish encryptions of different messages, violating CPA security.

1. **Observation:** The encryption scheme uses a deterministic function $E_k(m_i)$ to generate r_i, meaning that for the same plaintext message m_i, the value $r_i = E_k(m_i)$ is always the same.
2. **CPA Attack Strategy:**
    - The attacker can query the encryption oracle with two chosen messages m and m'.
    - Since r_i is **deterministic** (i.e., $r_i = E_k(m_i)$), it means the relation:

      $$c_i = r_i \oplus r_{i-1}$$

      will reveal **patterns** between ciphertexts for repeated plaintexts.
    - If the attacker sends the same plaintext multiple times, they can check whether the ciphertexts follow the same structure.
3. **Breaking CPA Security:**
    - Given two plaintext messages (m_1, m_2) such that $m_1 \neq m_2$, the attacker can compute:

      $$c_1 \oplus c_1' = (r_1 \oplus r_0) \oplus (r_1' \oplus r_0')$$

    - Since r_1 and r_1' are deterministic (because $r_1 = E_k(m_1)$), they **always** produce the same output for the same input.
    - This means an attacker can **detect when the same message is encrypted twice**, allowing them to distinguish between different messages.

Thus, the mode **does not achieve CPA security** because an attacker can detect patterns in ciphertexts, violating the indistinguishability requirement of CPA security.

Conclusion:

- **Decryption** is straightforward by reconstructing r_i and applying D_k.
- **CPA security is broken** because an attacker can detect repeated encryptions of the same message, allowing them to distinguish between messages.

Thus, this encryption mode should **not** be used for secure encryption under CPA.

Uploaded image

Determining Whether Each Construction is a PRG

A **Pseudorandom Generator (PRG)** is a deterministic function $G : \{0,1\}^n \to \{0,1\}^{3n}$ that takes a short random seed and expands it into a longer pseudorandom string. A construction is a secure PRG if its output is computationally indistinguishable from a truly random string of the same length.

Let's analyze each construction.

(a) $G_1(s) : G(s) = x \parallel y \parallel z$, Return $G(x) \parallel G(y)$.

- The given PRG G expands s into three parts x, y, z.
- Then, instead of using z, the function calls G again on x and y, and returns $G(x) \parallel G(y)$.

Analysis:

- If G is a secure PRG, then its output should be indistinguishable from uniform randomness.
- Since $G(x)$ and $G(y)$ are individually pseudorandom, their concatenation should still appear pseudorandom.
- However, G_1 **loses entropy**: The original PRG expands s to $3n$ bits, but then only uses $2n$ bits in the final output.
- This **weakens** the randomness of the output and makes it potentially distinguishable from a true $3n$-bit random string.

Conclusion: Not a PRG

Since it **does not produce a full-length pseudorandom output**, an attacker could distinguish it from true randomness.

(b) $G_2(s) : G(s) = x \parallel y \parallel z$, Return $x \parallel y$.

- The output consists only of x and y, discarding z.

Analysis:

- The function G originally expands s into $3n$ bits, but G_2 only outputs $2n$ bits.
- This means that the function is **shrinking entropy** instead of expanding it, which contradicts the purpose of a PRG.
- A secure PRG must expand randomness; here, we are throwing away z, which could contain valuable entropy.

Conclusion: Not a PRG

It fails to **expand** randomness and thus does not meet the definition of a secure PRG.

(c) $G_3(s) : G(s) = x$, $G(0^n) = y$, Return $x \oplus y$.

- The function computes $G(s) = x$ and then computes $G(0^n) = y$, outputting $x \oplus y$.

Analysis:

- The value $G(0^n)$ is a fixed output since it is computed independently of s.

- If an adversary knows or can predict $G(0^n)$, then they can **partially determine the output** of $G_3(s)$.
- A secure PRG must ensure that its outputs are indistinguishable from truly random strings. Here, since $G(0^n)$ is fixed for all evaluations, the output $x \oplus y$ can be distinguished from a purely random string by an adversary.

Conclusion: Not a PRG

It **introduces correlation** between different outputs, making it distinguishable from a truly random source.

Final Answer:

- $G_1(s)$ **is not a PRG** (loses entropy by dropping z).
- $G_2(s)$ **is not a PRG** (only outputs $2n$ bits instead of expanding randomness).
- $G_3(s)$ **is not a PRG** (fixed $G(0^n)$ allows an attacker to correlate outputs).

Thus, **none of these constructions are secure PRGs**.

Uploaded image

Proof: Feistel Structure is Invertible with Reversed Round Keys

A **Feistel network** consists of an even number of rounds where a function F and a key schedule K_i are used to transform an input into an output. The crucial property of the Feistel structure is that it is **invertible**, meaning decryption can be performed using the same structure but applying the round keys in reverse order.

Step 1: Feistel Encryption Process

The Feistel network operates on a message split into two halves (L_0, R_0), and applies the following iterative transformation for n rounds:

For each round i (where $1 \leq i \leq n$):

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

After n rounds, the final ciphertext is (L_n, R_n).

Step 2: Feistel Decryption Process

To decrypt, we apply the same Feistel structure but in **reverse order**, using the round keys in reverse order:

For each round i (where $n \geq i \geq 1$):

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus F(L_i, K_i)$$

After applying n reverse rounds, we recover the original plaintext (L_0, R_0).

Step 3: Verifying Correctness

Let's substitute back to check if decryption correctly recovers the original plaintext.

From encryption:

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

In decryption, using $L_i = R_{i-1}$:

$$L_{i-1} = R_i \oplus F(L_i, K_i)$$

Substituting $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$:

$$L_{i-1} = (L_{i-1} \oplus F(R_{i-1}, K_i)) \oplus F(R_{i-1}, K_i)$$

Since $x \oplus x = 0$ for any x, this simplifies to:

$$L_{i-1} = L_{i-1}$$

which is correct.

Similarly, using $R_{i-1} = L_i$, we see that reversing the process step-by-step recovers the original plaintext.

Conclusion

Since each step in decryption correctly inverts the encryption step, we conclude that:

- **A Feistel network with n rounds can be inverted using the same structure but applying the round keys in reverse order $(K_n, K_{n-1}, \ldots, K_1)$.**
- This property ensures **Feistel ciphers like DES (Data Encryption Standard) are invertible**, making them a fundamental structure in cryptographic design.

Thus, **the Feistel structure is inherently reversible** with reversed round keys. ✅

Uploaded image

(a) Computing i using $\mathcal{A}$ without Directly Querying x

We are given:

- A cyclic group $G = \langle g \rangle$.

- An element $x = g^i$ in G.
- An algorithm $\mathcal{A}$ that, given g^j, returns j, but we cannot query $\mathcal{A}$ with x.

Solution Approach:

1. Since G is cyclic, every element is of the form g^j for some integer j.
2. We want to find i but cannot call $\mathcal{A}$ with x directly.
3. Choose a random integer r, compute g^r, and query $\mathcal{A}$ with $g^r x$ instead of x:

$$\mathcal{A}(g^r x) = \mathcal{A}(g^r g^i) = \mathcal{A}(g^{r+i}) = r + i$$

4. Since r is known, we can compute i as:

$$i = \mathcal{A}(g^r x) - r$$

Thus, we successfully compute i without querying x directly.

(b) Defining CDH and DDH, and Showing CDH can Solve DDH

Computational Diffie-Hellman (CDH) Problem

The **CDH problem** in a cyclic group $G = \langle g \rangle$ of order p is:

- Given g^a and g^b, compute g^{ab}.
- A **CDH solver** is an algorithm that, given (g^a, g^b), outputs g^{ab}.

Decisional Diffie-Hellman (DDH) Problem

The **DDH problem** is to distinguish between the following two distributions:

1. **Real DDH tuple**: (g^a, g^b, g^{ab}), where g^a and g^b are random elements in G.
2. **Random tuple**: (g^a, g^b, g^c), where g^c is a random element in G.

A **DDH solver** is an algorithm that, given (g^a, g^b, g^z), determines whether $z = ab$ (real) or random.

Using CDH to Solve DDH

1. Given a DDH instance (g^a, g^b, g^z), a CDH solver can compute g^{ab}.
2. We compare g^{ab} with g^z:
   - If $g^{ab} = g^z$, then the tuple is from the DDH distribution.
   - Otherwise, g^z is random.
3. Thus, if we have a **CDH solver**, we can always verify whether g^z matches g^{ab}, which solves DDH.

Conclusion:
A solver for **CDH** can be used to **solve DDH**, implying that CDH is at least as hard as DDH. This shows that breaking CDH **implies breaking DDH**, but not necessarily vice versa.

Minor 2_2024 crypto.pdf
PDF

provide answer for all these question. Provide the mathematical proof were necessary

Solutions to the Questions in "Minor 2_2024 Crypto.pdf"

---

1. Birthday Paradox and Its Implication on Cryptographic Hash Functions

(a) Explanation of the Birthday Paradox

The **birthday paradox** is a probability result that states that in a group of **23** people, there is a **50% probability** that at least two people share the same birthday. This result is counterintuitive because, despite there being 365 possible birthdays, the number of comparisons grows quadratically as the number of people increases.

Formally, for a hash function with an output size of n bits, the probability of finding a **collision** (i.e., two different inputs hashing to the same value) becomes significant after approximately $2^{n/2}$ attempts. This is much smaller than 2^n, which would be required in a brute-force search.

(b) Implication on Hash Functions

- The birthday paradox **reduces the security** of a hash function from 2^n **to** $2^{n/2}$ against **collision attacks**.
- For a 128-bit hash function, an adversary needs only about 2^{64} attempts to find a collision, rather than 2^{128}.
- Cryptographic hash functions like **SHA-256** and **SHA-3** are designed to resist such attacks by using sufficiently large hash output sizes.

---

2. How Two Untrusting Parties Can Toss a Fair Coin Using Public Messages

Solution: The Coin-Flipping Protocol

Two parties, **Alice** and **Bob**, want to fairly decide a random bit (heads or tails) over a public channel without trusting each other.

1. **Step 1**: Alice chooses a random bit $b_A \in \{0, 1\}$ and commits to it using a cryptographic hash:

   $$H_A = H(b_A || r_A)$$

   where r_A is a random nonce to prevent guessing.
2. **Step 2**: Bob chooses a random bit b_B and sends it to Alice.
3. **Step 3**: Alice reveals b_A and r_A. Bob verifies that:

$$H(b_A||r_A) = H_A$$

4. **Step 4**: The fair coin result is computed as:

$$\text{Coin Flip Result} = b_A \oplus b_B$$

If both parties are honest, this produces an unbiased result.

Security Features

- Alice cannot change b_A after seeing b_B since she is committed with a hash.
- Bob has no influence over b_A, ensuring fairness.

---

3. Elliptic Curve ElGamal Cryptosystem

(a) Key Setup

1. Choose an **elliptic curve** E over a finite field $\mathbb{F}_p$.
2. Select a **generator point** G of a large prime order q.
3. Alice selects a **private key** $x_A \in [1, q-1]$ and computes her **public key**:

$$P_A = x_A G$$

4. The public key is (E, G, P_A) and the private key is x_A.

(b) Encryption

To encrypt a message M represented as a point on the elliptic curve:

1. Bob selects a **random integer** $k \in [1, q-1]$.
2. Compute **ciphertext pair**:

$$C_1 = kG$$

$$C_2 = M + kP_A$$

3. Bob sends (C_1, C_2) to Alice.

(c) Decryption

Alice decrypts the message by computing:

$$M = C_2 - x_A C_1$$

since:

$$C_2 - x_A C_1 = M + kP_A - x_A kG = M$$

---

4. Multi-Prime RSA

(a) Finding the Decryption Exponent for $n = 120, e = 3$

1. Factorize $n = 120$:

$$n = 2^3 \times 3 \times 5$$

2. Compute **Euler's totient function**:

$$\phi(n) = (2-1)(2^2)(3-1)(5-1) = 1 \times 4 \times 2 \times 4 = 32$$

3. Find **decryption exponent** d:

$$d = e^{-1} \mod \phi(n)$$

   Solving $3d \equiv 1 \mod 32$, we get:

$$d = 11$$

(b) Justification for Multi-Prime RSA

- **Faster decryption**: Multi-prime RSA allows **faster modular exponentiation** using **Chinese Remainder Theorem (CRT)**.
- **More efficient key generation**: Reduces the bit length of prime factors while maintaining security.

---

5. El-Gamal Encryption Over $\mathbb{Z}_{29}$

(a) Alice's Public Key

Given:

- $g = 7$
- Private key $x = 9$

Compute:

$$h = g^x \mod 29 = 7^9 \mod 29$$

Using fast exponentiation:

$$7^9 \equiv 9 \mod 29$$

Thus, Alice's public key is **(29, 7, 9)**.

(b) Decrypting the Ciphertext (3,2)

1. Compute **shared secret**:

$$s = 3^9 \mod 29$$

   Using fast exponentiation:

$$s = 3^9 \equiv 27 \mod 29$$

2. Compute **inverse of** s modulo 29:

$$s^{-1} \equiv 27^{-1} \equiv 16 \mod 29$$

3. Recover the message:

$$m = (2 \times 16) \mod 29 = 32 \mod 29 = 3$$

Message sent: 3

---

6. RSA-Based Encryption Scheme

(a) Decryption Process

Given ciphertext:

$$(c_1, c_2) = (r^e \mod n, H(r) \oplus m)$$

1. Compute r:

$$r = c_1^d \mod n$$

2. Compute m:

$$m = c_2 \oplus H(r)$$

(b) Is It CCA Secure?

No, it is NOT CCA secure because:

- Given (c_1, c_2), an attacker can query the decryption oracle for a **modified ciphertext** (c_1', c_2).
- Since decryption **only depends on** r, the attacker can recover $H(r)$ and use it to decrypt **other ciphertexts**.

Thus, the scheme is vulnerable to chosen-ciphertext attacks (CCA).

Uploaded image
Provide the answers for this major exam questions

Solutions to Major Exam Questions

---

1. True or False Statements on Pseudorandom Generators (PRGs)

(a) "There exists a pseudorandom generator G_n where for every n, $G_n : \{0,1\}^n \to \{0,1\}^{2n}$ such that for every $x \in \{0,1\}^n$, the first $n/3$ bits of $G_n(x)$ are zero."

Answer: False

Justification:

- A PRG is defined as a function that takes a short random seed of length n and expands it into a longer string that appears random.
- If the first $n/3$ bits of $G_n(x)$ are always zero, then an adversary can trivially distinguish $G_n(x)$ from a truly random string, breaking the pseudorandom property.
- In a truly random string of length $2n$, the first $n/3$ bits would be **random**, not deterministically zero.
- This contradicts the indistinguishability property of PRGs.

(b) "There exists a pseudorandom generator G_n where for every n, $G_n : \{0,1\}^n \to \{0,1\}^{2n}$ such that for every $x \in \{0,1\}^n$, there exist $n/3$ bits of $G_n(x)$ that are zero."

Answer: True

Justification:

- Unlike (a), this statement does not specify which $n/3$ bits are zero.
- If the location of the zero bits is random across different executions of $G_n(x)$, then an adversary **cannot reliably distinguish** the output from a truly random string.
- This allows G_n to still be pseudorandom as long as the remaining bits are **sufficiently unpredictable**.

---

2. PRP Construction from PRF

We are given a **pseudorandom function (PRF)** $f_K(x)$ and we construct a **pseudorandom permutation (PRP)** $g_K(x)$ as:

- Input: $x = (x_1||x_2||x_3)$, where x_1, x_2, x_3 are n-bit strings.
- Outputs:
  - $y_2 = f_K(x_1) \oplus x_2 \oplus f_K(x_3)$
  - $y_1 = x_1 \oplus f_K(y_2)$
  - $y_3 = x_3 \oplus f_K(y_2)$

(a) Prove that $g_K(x)$ is a permutation.

To prove that $g_K(x)$ is a permutation, we must show that the mapping is **bijective** (i.e., invertible).

- Given y_1, y_2, y_3, we solve for x_1, x_2, x_3:
  - $x_3 = y_3 \oplus f_K(y_2)$
  - $x_1 = y_1 \oplus f_K(y_2)$
  - $x_2 = y_2 \oplus f_K(x_1) \oplus f_K(x_3)$

Since the mapping can be reversed uniquely, $g_K(x)$ is a **permutation**.

(b) Show that $g_K(x)$ is not a PRP.

To be a **secure PRP**, $g_K(x)$ must be computationally indistinguishable from a truly random permutation.

- We are allowed to query $g_K(x)$ without knowing how $f_K(x)$ is constructed.

- However, since the function involves direct applications of $f_K(x)$ in a predictable structure, an adversary could analyze the output for patterns.
- Specifically, if an attacker queries the same x_1, they may notice that y_2 **does not behave truly randomly**.
- This makes $g_K(x)$ distinguishable from a random permutation.

Thus, $g_K(x)$ **is not pseudorandom**, meaning it fails to be a **secure PRP**.

---

3. SHA256 Collision Probability Analysis

The given function:

$$h(\text{user}) = \text{SHA256}(\text{user}@iitj.ac.in)$$

where:

- x is your email ID.
- y is another email ID.
- We need to find y such that the **first 100 bits** of $h(x)$ and $h(y)$ match.

How Many SHA256 Calls Are Needed?

- The **birthday paradox** tells us that the number of hash evaluations required to find a partial collision on **100 bits** is approximately:

$$2^{100/2} = 2^{50}$$

- This is because for a hash function producing n bits, the expected number of random trials to find a collision on k bits is $2^{k/2}$.

Justification

- A truly random SHA-256 hash behaves like a random function.
- We need to find two different email IDs such that their hash values match in the first **100 bits**.
- The **birthday bound** tells us that, with high probability, a match occurs after approximately 2^{50} queries.

Thus, we need around 2^{50} **SHA-256 calls** to find such a username y.

Uploaded image

Solutions to the Given Problems

---

4. CPA Security of Modified Encryption Scheme Π^*

Given that $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ is a **CPA-secure encryption scheme**, we analyze whether the modified scheme Π^* retains CPA security.

Case (a):

$$\text{Enc}^*_K(M) = \text{Enc}_K(M \oplus 1^n)$$

Analysis:

- Since Π is CPA-secure, the encryption function should produce indistinguishable ciphertexts for different messages.
- The transformation $M \to M \oplus 1^n$ flips all bits of M.
- However, **applying a deterministic transformation** before encryption does not leak information because an adversary still cannot distinguish between encrypted messages.
- Since $M \oplus 1^n$ remains **unpredictable to the adversary**, **CPA security is preserved**.

Conclusion:
✅ **CPA-secure**.

Case (b):

$$\text{Enc}^*_K(M) = \text{Enc}_K(M) || \text{LSB}(M)$$

Analysis:

- Here, the encryption function outputs the encryption of M, but also **explicitly leaks** the least significant bit (LSB) of M.
- Since the adversary receives extra information about M (specifically its last bit), the encryption scheme no longer satisfies **indistinguishability** under a chosen plaintext attack.
- Given two messages M_0 and M_1 that differ in the last bit, an adversary can **distinguish** which message was encrypted by simply checking the leaked LSB.

Conclusion:
❌ **Not CPA-secure**.

---

5. Secure Message Transmission from Alice to Bob

Alice wants to send Bob a message m containing:

- Her **name** N
- Her **computer's IP address** IP
- A **request** R **for Bob**
  while ensuring:
- **Message integrity** (the message has not been altered).
- **Authentication** (Bob can verify that Alice sent the message).

(a) Using Symmetric Key Cryptography

Assume Alice and Bob share a **symmetric key** K.

Solution:

$$m = (N, IP, R)||\text{MAC}_K(N, IP, R)$$

Where:

- MAC_K is a **Message Authentication Code (MAC)** computed using the secret key K.
- Bob verifies the integrity and authenticity by **recomputing the MAC** and checking if it matches.

✅ **Ensures:**
✔ **Integrity**: Any modification in the message would result in an invalid MAC.
✔ **Authentication**: Only Alice (who knows K) can generate a valid MAC.

(b) Using Public Key Cryptography

Assume Alice and Bob have public-private key pairs **(K_A^{pub}, K_A^{priv})** and **(K_B^{pub}, K_B^{priv})**.

Solution:

$$m = (N, IP, R)||\text{Sign}_{K_A^{priv}}(N, IP, R)$$

Where:

- $\text{Sign}_{K_A^{priv}}$ is a **digital signature** computed using Alice's private key.
- Bob verifies the signature using **Alice's public key** K_A^{pub}.

✅ **Ensures:**
✔ **Integrity**: Any modification in the message will cause the signature verification to fail.
✔ **Authentication**: Only Alice (who owns K_A^{priv}) can produce a valid signature.

Final Answers

1. **(a) CPA-secure, (b) Not CPA-secure.**
2. **(a) Use MAC with a symmetric key, (b) Use digital signature with a public key.**

6. Your friend uses $n = 51$ in the RSA encryption scheme. Compute a valid key pair (e, d) for this parameter. How many values of e are possible for this n ? (2+2 = 4 marks)

7. Diffie-Hellman Key Exchange allows two parties to share a secret key in one step of communication by these parties. Explain how can you share a secret key among 3 parties in a single step. Formally state the assumption which ensures that the scheme you presented is secure. (2+2 = 4 marks)

8. Formally define a Commitment scheme. What are the security properties of the same? (2+2 = 4 marks)

9. Explain what is meant by completeness and soundness with respect to a Zero-knowledge proof (ZKP). Show a ZKP for two large graphs being isomorphic. (2+2 = 4 marks)

10. In Shamir's Secret Sharing scheme, a secret has been shared among n parties using a polynomial of degree k. Party i has an input (x_i, y_i) with them. Is is observed that there are two users i and j with inputs (x_i, y_i) and (x_j, y_j) such that $x_i = x_j$ but $y_i \neq y_j$. It is clear that one of them is trying to cheat the system (by sharing an incorrect secret). How can you know who is the cheater? What restriction do you need on n in terms of k? (4 Marks)

*****

Solutions to the Given Questions

6. RSA Key Pair Computation for $n = 51$

We are given $n = 51$ and need to compute a valid RSA key pair (e, d).

Step 1: Factorize n

$$n = 51 = 3 \times 17$$

Step 2: Compute Euler's Totient Function

$$\phi(n) = (3 - 1)(17 - 1) = 2 \times 16 = 32$$

Step 3: Choose Public Key e

- e must satisfy $\gcd(e, \phi(n)) = 1$.
- Possible choices: Any number **coprime to 32**.
- Valid values: $e \in \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31\}$ (i.e., numbers < 32 that are **odd and not multiples of 2**).

Step 4: Compute Private Key d

- d is the modular inverse of e modulo $\phi(n)$, i.e.,

$$d \equiv e^{-1} \mod 32$$

- Example: If $e = 3$, then solving $3d \equiv 1 \mod 32$ gives $d = 11$ (since $3 \times 11 = 33 \equiv 1 \mod 32$).

Final Answer

- **Total possible values for e: 16**
- **Example RSA Key Pair**: $(e, d) = (3, 11)$.

---

7. Diffie-Hellman Key Exchange for 3 Parties

Standard 2-Party Diffie-Hellman (DH)

For two parties:

1. Alice and Bob agree on a **public base** g and a **prime** p.
2. Alice selects a private key a and computes $A = g^a \mod p$.
3. Bob selects a private key b and computes $B = g^b \mod p$.
4. They exchange A and B, and both compute the shared secret:

$$S = B^a \mod p = A^b \mod p = g^{ab} \mod p$$

Extending to 3 Parties

For three parties (Alice, Bob, Charlie):

1. Each selects private keys a, b, c.
2. They compute and exchange:

$$A = g^a \mod p, \quad B = g^b \mod p, \quad C = g^c \mod p$$

3. Each party combines:

$$X = (B^a C^a) \mod p, \quad Y = (A^b C^b) \mod p, \quad Z = (A^c B^c) \mod p$$

4. Final shared secret:

$$S = g^{abc} \mod p$$

Assumption for Security: **Computational Diffie-Hellman (CDH) Assumption**

- Given g^a, g^b, and g^c, an adversary **cannot compute** g^{abc}.

---

8. Definition and Security of Commitment Scheme

Definition

A **commitment scheme** allows a sender to commit to a value while keeping it hidden, with the ability to reveal it later.

- **Commit Phase:**
  Sender chooses a value m and a random nonce r, then computes:

$$C = \text{Commit}(m, r)$$

  and sends C to the receiver.

- **Reveal Phase:**
  Sender discloses (m, r) and the receiver verifies:

$$C = \text{Commit}(m, r)$$

Security Properties

1. **Hiding:** Ensures that C reveals no information about m.
2. **Binding:** Prevents sender from changing m after commitment.

---

9. Completeness and Soundness in Zero-Knowledge Proofs (ZKP)

Definition of Completeness

- If the prover is honest and follows the protocol, the verifier **always** accepts.

Definition of Soundness

- A cheating prover **cannot convince** the verifier of a false statement, except with **negligible probability**.

ZKP for Graph Isomorphism

- Given two graphs G_1 and G_2, a prover wants to convince a verifier they are **isomorphic** without revealing the mapping.

Protocol:

1. Prover picks a random permutation π of G_1 and computes $H = \pi(G_1)$.
2. Prover sends H to the verifier.
3. Verifier randomly asks for:
   - The mapping between G_1 and H **or**
   - The mapping between G_2 and H.
4. Prover reveals the requested mapping.
5. The verifier checks that the revealed mapping is valid.

✅ **Completeness:** An honest prover always passes.
✅ **Soundness:** A cheater must guess correctly, succeeding with probability $1/2$ **per round**. Repeating reduces this probability exponentially.

---

10. Detecting a Cheater in Shamir's Secret Sharing

Problem Setup

- A secret is split among n parties using a polynomial $P(x)$ of degree k.
- Each party receives a share (x_i, y_i).
- Two users i and j have $x_i = x_j$ but $y_i \neq y_j$, meaning one of them is lying.

Solution: Lagrange Interpolation

- The **correct shares must lie on the polynomial** $P(x)$.
- We can reconstruct $P(x)$ using **any** $k + 1$ **correct shares**.
- Compute **two polynomials**:
  - $P_i(x)$ using (x_i, y_i) and other k shares.
  - $P_j(x)$ using (x_j, y_j) and other k shares.
- If $P_i(x) \neq P_j(x)$, then one must be **incorrect**.

Restriction on n and k

- Need **at least** $k + 1$ **honest shares** to reconstruct the correct polynomial.
- Ensure $n > 2k$ to guarantee enough honest shares to detect cheaters.

✅ **Final Answer:** Check which polynomial is inconsistent and reject the incorrect party. $n > 2k$ ensures correctness.