CSCI971/CSCI471 Modern Cryptography
Assignment 1

Name: Karan Goel
Student Number: 7836685

September 1, 2024

1 Task A: Monoalphabetic Substitution Cipher

The Monoalphabetic Substitution Cipher is a classical encryption technique where each letter in the plaintext is replaced by a different letter from a fixed, shuffled alphabet. This establishes a one-to-one correspondence between the letters of the plaintext and the ciphertext, enabling both encryption and decryption.

How It Works

1. **Substitution Table:** A substitution table is first created, mapping each letter of the alphabet to a different letter. For example, with the substitution alphabet `BQVXTFZDEUSGYMWLJOCPHIANKR`, the mappings would be:
   - A $\rightarrow$ B
   - B $\rightarrow$ Q
   - C $\rightarrow$ V
   - *and so on.*
2. **Encryption:** Each letter in the plaintext is replaced by its corresponding letter in the substitution table. For instance, using the table above, the plaintext `KARAN` would be encrypted as `SBOBM`.
3. **Decryption:** To decrypt, the inverse of the substitution table is used, mapping each letter in the ciphertext back to the original letter in the plaintext.

Breaking the Monoalphabetic Substitution Cipher

Monoalphabetic substitution ciphers are relatively easy to break since the ciphertext still resembles the structure of the original language. The frequency of letters in the ciphertext will often mirror those in the plaintext, making it easier to identify.

In English, for example, letters like E, T, A, O, and N appear most frequently. If the ciphertext shows a similar distribution, it likely uses a monoalphabetic substitution cipher.

To break a monoalphabetic substitution cipher, several methods can be employed:

- **Statistical Analysis:** By analyzing letter frequencies and patterns, one can infer the substitution alphabet. Starting with a random guess and refining it based on statistical patterns often leads to deciphering the plaintext.

  Example 1.1. *For instance, given the ciphertext* `MK MWXXW!`*, an attacker might guess that the most frequent letters correspond to common letters like E, T, A, or O in English. By adjusting the substitution accordingly, the plaintext could be revealed as* `HI HELLO!`*.*

- **Known Plaintext Attack:** If parts of the plaintext are known or guessed, this information can help deduce the substitution pattern. By identifying common words or phrases in the ciphertext, the attacker can reverse-engineer the substitution.

  Example 1.2. *Suppose the ciphertext* `MWXXW!` *corresponds to the known plaintext* `HELLO!`*. The attacker could deduce:*

  - $M \to H$
  - $W \to E$
  - $X \to L$
  - $W \to O$

  Applying these mappings to the ciphertext would reveal the plaintext as `HELLO!`*.*

2 Task B: Modifying the Digital Signature Algorithm

In digital signatures, the main objective is to ensure that only Bob, who has the secret key sk_B, can verify a signature created by Alice.

To enhance the security of the digital signature algorithm, the following modifications are proposed:

Signing Algorithm

- Alice uses her secret key sk_A to generate a signature S for the message m. After that, the signature is encrypted using Bob's public key pk_B to ensure that only Bob can decrypt and verify it.

$$S = \text{sign}(m, sk_A)$$

$$\text{encrypted\_signature} = \text{Encrypt}(S, pk_B)$$

Verification Algorithm

- Bob uses his private key sk_B to decrypt the encrypted signature. Once decrypted, he uses Alice's public key pk_A to verify that the signature matches the message m.

$$\text{decrypted\_signature} = \text{Decrypt}(\text{encrypted\_signature}, sk_B)$$

$$\text{verify}(\text{decrypted\_signature}, m, pk_A)$$

Why These Changes Are Necessary

1. **Ensuring Only Bob Can Verify the Signature:**

   - Encrypting the signature with Bob's public key pk_B ensures that only Bob, who has the corresponding private key sk_B, can decrypt and verify it. This modification makes sure that, even though Alice uses her secret key sk_A to sign the message, only Bob's key pair can verify the signature. Without this step, anyone with Alice's public key pk_A could verify the signature.

2. **Enhancing Security and Integrity:**

   - Adding encryption with Bob's public key provides an extra layer of security. It ties the signature not only to Alice (via sk_A) but also to Bob's verification process, maintaining the integrity and authenticity of the signature. This way, the signature can only be verified by Bob, ensuring its exclusivity.

3 Task C: Output of a 16-Round Feistel Network

In a Feistel network, the input data is divided into two halves, and a series of rounds are applied using a specific round function. In this task, we'll examine the output of a 16-round Feistel network where the round function is simply the identity function, meaning $F(R) = R$.

Overview of the Feistel Network

A Feistel network operates as follows:

1. **Initial Split:** The input data is split into two equal halves: L_0 (left half) and R_0 (right half).

2. **Round Function Application:** In each round i, the round function F is applied to R_i along with a round sub-key K_i. The result is XORed with L_i to create the new left half, while the right half remains unchanged.

3. **Swap:** After each round, the left and right halves are swapped, so the output (L_{i+1}, R_{i+1}) from one round becomes the input for the next. This swapping occurs in every round except the final one.

Detailed Steps

Given:

- The round function F is defined as $F(R_i, K_i) = R_i$, where K_i is the round sub-key, though it isn't used here.
- The network consists of 16 rounds.

Round 1:

$$\begin{aligned}\text{Apply the round function:} \quad F(R_0, K_0) &= R_0 \\ L_1 &= R_0 \\ R_1 &= L_0 \oplus F(R_0, K_0) = L_0 \oplus R_0\end{aligned}$$

Round 2:

$$\begin{aligned}\text{Apply the round function:} \quad F(R_1, K_1) &= R_1 = L_0 \oplus R_0 \\ L_2 &= R_1 = L_0 \oplus R_0 \\ R_2 &= L_1 \oplus F(R_1, K_1) = R_0 \oplus (L_0 \oplus R_0) = L_0\end{aligned}$$

Round 3:

$$\begin{aligned}\text{Apply the round function:} \quad F(R_2, K_2) &= R_2 = L_0 \\ L_3 &= R_2 = L_0 \\ R_3 &= L_2 \oplus F(R_2, K_2) = (L_0 \oplus R_0) \oplus L_0 = R_0\end{aligned}$$

Round 4:

$$\begin{aligned}\text{Apply the round function:} \quad F(R_3, K_3) &= R_3 = R_0 \\ L_4 &= R_3 = R_0 \\ R_4 &= L_3 \oplus F(R_3, K_3) = L_0 \oplus R_0\end{aligned}$$

Round 5:

$$\begin{aligned}\text{Apply the round function:} \quad F(R_4, K_4) &= R_4 = L_0 \oplus R_0 \\ L_5 &= R_4 = L_0 \oplus R_0 \\ R_5 &= L_4 \oplus F(R_4, K_4) = R_0 \oplus (L_0 \oplus R_0) = L_0\end{aligned}$$

Round 6:

$$\begin{aligned}\text{Apply the round function:} \quad F(R_5, K_5) &= R_5 = L_0 \\ L_6 &= R_5 = L_0 \\ R_6 &= L_5 \oplus F(R_5, K_5) = (L_0 \oplus R_0) \oplus L_0 = R_0\end{aligned}$$

Observing the Pattern

We observe that after every three rounds, the pattern repeats:

- After 1 round: $(L_1, R_1) = (R_0, L_0 \oplus R_0)$
- After 2 rounds: $(L_2, R_2) = (L_0 \oplus R_0, L_0)$
- After 3 rounds: $(L_3, R_3) = (L_0, R_0)$

Final Output

After 16 rounds, the Feistel network will output:

$$L_{16} = L_0 \oplus R_0$$
$$R_{16} = R_0$$

Thus, the output of the 16-round Feistel network, with the identity round function $F(R_i, K_i) = R_i$, will be $(L_0 \oplus R_0, R_0)$. This occurs because the final round does not include a swap.

4 Task D: Security of the MAC Scheme

Consider a Message Authentication Code (MAC) generation algorithm based on a block cipher F with block length n. The MAC generation algorithm operates as follows:

1. **Input:** A secret key k for the block cipher F and a message $M \in \{0,1\}^{nl}$.
2. **Parsing:** The message M is divided into l blocks $m_1, m_2, \ldots, m_l$.
3. **Tag Computation:** For each block m_i, compute $t_i = F_k(m_i)$.
4. **Output:** The MAC tag $T = (t_1, t_2, \ldots, t_l)$.

Attacks on the MAC Scheme

The MAC scheme described is vulnerable to several attacks due to the block-wise independence of tag computation. Below are three such attacks:

Simple Substitution Attack

1. The adversary queries the MAC for a message $M = (m_1, m_2, \ldots, m_l)$ to obtain the tag $T = (t_1, t_2, \ldots, t_l)$.
2. The adversary replaces a block m_1 with a new block m_1' for which the tag $t_1' = F_k(m_1')$ is known.
3. The adversary forms a modified message $M' = (m_1', m_2, \ldots, m_l)$ and its corresponding valid tag $T' = (t_1', t_2, \ldots, t_l)$.

Birthday Attack

1. The adversary generates a large set of messages $M_1, M_2, \ldots, M_N$.
2. The adversary queries the MAC for these messages to obtain their tags.
3. The adversary looks for two messages M_i and M_j that produce the same tag for at least one block, exploiting the birthday paradox.
4. The adversary combines blocks from M_i and M_j to create a new message with a valid MAC.

Block Swapping Attack

1. The adversary selects two messages $M = (m_1, m_2)$ and $M' = (m_3, m_4)$.
2. The adversary queries the MAC for these messages to obtain their tags $T = (t_1, t_2)$ and $T' = (t_3, t_4)$.
3. The adversary swaps blocks between the two messages to form a new message $M'' = (m_1, m_4)$.
4. The adversary constructs the new tag $T'' = (t_1, t_4)$, which is valid for the new message M''.

Summary

These attacks demonstrate that the block-wise independent processing in this MAC scheme allows an attacker to forge valid message/tag pairs, compromising the scheme's security.

5 Task E: Forging an RSA Signature

In this task, we explore how to forge the RSA signature for the message 45, given the signatures for the messages 3 and 5, using the properties of modular arithmetic in the RSA signature scheme.

1. Overview of the RSA Signature Scheme

The RSA signature process works as follows:

- **Signing**: To sign a message m, you compute the signature as signature $= m^d \mod n$, where d is the private key and n is the RSA modulus.
- **Verification**: To verify a signature s, check that $s^e \mod n = m$, where e is the public key.

2. What We Have

We are given the signatures for the messages 3 and 5:

- $s_3 = 3^d \mod n$
- $s_5 = 5^d \mod n$

Here, d is the private key.

3. Our Goal

We want to forge the signature for the message 45. Specifically, we need to find s_{45} such that:

$$s_{45} = 45^d \mod n$$

4. Breaking Down 45

Let's express 45 using its prime factors:

$$45 = 3^2 \cdot 5$$

So, when we raise 45 to the power d, we have:

$$45^d = (3^2 \cdot 5)^d = 3^{2d} \cdot 5^d$$

5. Using the Given Signatures

We already know:

- $s_3 = 3^d \mod n$
- $s_5 = 5^d \mod n$

Thus, we can express 3^{2d} and 5^d as:

$$3^{2d} = (3^d)^2 = s_3^2$$

$$5^d = s_5$$

So, we can combine them to find:

$$45^d = 3^{2d} \cdot 5^d \equiv (s_3^2 \cdot s_5) \mod n$$

6. Forging the Signature

We can now compute the forged signature s_{45} as:

$$s_{45} = (s_3^2 \cdot s_5) \mod n$$

7. Verifying the Forged Signature

To verify that s_{45} is indeed the correct signature for 45, check:

$$s_{45}^{e} \mod n$$

This should equal 45, where e is the public exponent.

Result

- First, obtain the signatures for 3 and 5, denoted as s_3 and s_5.
- Then, compute the forged signature:

$$s_{45} = (s_3^2 \cdot s_5) \mod n$$

- This s_{45} is the valid RSA signature for the message 45.

This approach works because the RSA signature scheme's reliance on modular arithmetic allows us to construct signatures for composite messages from the signatures of their factors.