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 letter from a fixed alphabet that has been shuffled or reordered. It establishes a one-to-one correspondence between the letters of the plaintext and ciphertext, facilitating both encryption and decryption.

How It Works

1. **Substitution Table:** A substitution table is created, mapping each letter of the alphabet to a different letter. For instance, with the substitution alphabet `BQVXTFZDEUSGYMWLJOCPHIANKR`, the mappings are:
   - A $\rightarrow$ B
   - B $\rightarrow$ Q
   - C $\rightarrow$ V
   - *and so on.*
2. **Encryption:** Each letter in the plaintext is substituted with its corresponding letter from the substitution table. For example, the plaintext `KARAN` encrypted using the table becomes `SBOBM`.
3. **Decryption:** To reverse the process, the inverse of the substitution table is used. Each letter in the ciphertext is mapped back to its original letter in the plaintext.

Breaking the Monoalphabetic Substitution Cipher

Ciphered text will exhibit a similar index of coincidence to the plaintext language. For example, English text encrypted with a monoalphabetic substitution cipher retains a letter frequency distribution akin to normal English text.

For instance, in English, the letters E, T, A, O, and N are the most frequent. If a ciphertext has a similar distribution of letters, it might be a monoalphabetic substitution cipher.

- **Statistical Analysis:** This method involves using statistical techniques to infer the substitution alphabet. A common approach is to start with a random substitution, then calculate the likelihood of the resulting plaintext based on letter frequency and patterns.

  Example 1.1. *Suppose the ciphertext is* `XLMW MWXLIW!`*. An attacker might use statistical analysis to guess that the most frequent letters in the ciphertext correspond to the most frequent letters in English, such as E, T, A, etc. By iterating and adjusting the substitution based on letter frequency and common English words, the attacker might deduce that:*

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

  Eventually, the plaintext might be revealed as `THIS HELLO!`*.*

- **Known Plaintext Attack:** If parts of the plaintext are known or can be guessed, they can be used to deduce the substitution pattern. This approach often involves identifying common words or phrases in the ciphertext.

  Example 1.2. *Suppose the ciphertext is* `KXK LYF!` *and the attacker knows that the plaintext includes the phrase* `HELLO!`*. By comparing this known plaintext with the ciphertext, the attacker might deduce:*

  - $K \rightarrow H$
  - $X \rightarrow E$
  - $L \rightarrow L$
  - $Y \rightarrow O$

  Applying this mapping to the rest of the ciphertext, the plaintext could be revealed as `HELLO!`*.*

2 Task B: Digital Signature Algorithm Modification

In digital signatures, the goal is to ensure that a signature generated by Alice can only be verified by Bob, who knows the secret key sk_B.

To ensure the security of the digital signature algorithm, the following modifications are needed:

Signing Algorithm:

- The signing algorithm uses Alice's secret key sk_A to produce a signature S on the message m. The signature is then encrypted using Bob's public key pk_B to ensure that only Bob can verify it.

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

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

Verification Algorithm:

- The verification algorithm uses Bob's private key sk_B to decrypt the encrypted signature. It then uses Alice's public key pk_A to verify the decrypted signature against the message m.

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

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

2. Explanation of the Necessity for Changes

1. **Ensuring Bob's Exclusive Verification Capability:**
   - By encrypting the signature with Bob's public key pk_B, the signature is made such that only Bob, who possesses the corresponding private key sk_B, can decrypt and verify it. This ensures that, even though Alice uses her own secret key sk_A to sign the message, the signature is verifiable only by Bob's key pair. Without this adjustment, anyone who has Alice's public key pk_A could verify the signature, which would not meet the requirement.
2. **Security and Integrity of the Signature:**
   - Encrypting the signature with Bob's public key adds an extra layer of security. It ensures that the signature, while proving that the message came from Alice (via sk_A), is also tied to Bob's verification process. This change maintains the integrity and authenticity of the signature and ensures it can only be verified by Bob.
3. **Preventing Unauthorized Verification:**
   - If the signature does not involve Bob's public key, other parties who possess Alice's public key might be able to verify the signature. By embedding Bob's public key in the signature creation and verification process, the system restricts verification to Bob alone, thus preventing unauthorized parties from verifying the signature. This ensures that Bob is the sole verifier of Alice's signatures.

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

In a Feistel network, the input is split into two halves, and a series of rounds are performed using a round function. For this task, we will analyze the output of a 16-round Feistel network where the round function is the identity function, meaning $F(R) = R$.

Feistel Network Overview

A Feistel network consists of the following steps:

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

2. **Round Function Application:** In each round i, the round function F is applied to R_i using the round sub-key K_i. The output of this function is then XORed with L_i to produce the new left half, and the right half remains unchanged.

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

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 is not used in this case).
- The network has 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:
It becomes evident that after every three rounds this 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 produce:

$$\begin{aligned}L_{15} &= R_0 \\ R_{15} &= L_0\end{aligned}$$

Thus, after 16 rounds, the output of the Feistel network with the round function defined as $F(R_i, K_i) = R_i$ will be the same as the initial input but swapped (R_0, L_0). This is because the Feistel network does not perform the final swap at the end of the last round, which would otherwise restore the input order.

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 $T = (t_1, t_2, \ldots, t_l)$.

To justify this scheme is not secure we can see what kind of attack an adversory can perform.

5 Attacks on the MAC Scheme

The MAC scheme described is vulnerable to several chosen-message attacks where an attacker can exploit the block-wise independence of the tag computation to forge valid message/tag pairs. Below, we describe three such attacks: the Simple Substitution Attack, the Birthday Attack, and the Block Swapping Attack.

Simple Substitution Attack

In the Simple Substitution Attack, the attacker first chooses a message $M = (m_1, m_2, \ldots, m_l)$ and queries the MAC scheme to receive the tag $T = (t_1, t_2, \ldots, t_l)$. To generate a new message/tag pair, the attacker creates a modified message $M' = (m'_1, m_2, \ldots, m_l)$ by replacing one block of the original message with a different block m'_1, for which they already have a tag $t'_1 = F_k(m'_1)$ from a previous query. The new valid tag is constructed as $T' = (t'_1, t_2, \ldots, t_l)$.

Birthday Attack

In the Birthday Attack, the attacker leverages the birthday paradox, which suggests that in a sufficiently large set of data, the probability of a collision increases significantly. Here's how this can be applied to the MAC scheme:

- The attacker generates a large number of different messages $M_1, M_2, \ldots, M_N$, where each message is structured such that each block is independently chosen (e.g., $M_i = (m_{i1}, m_{i2}, \ldots, m_{il})$).
- The attacker queries the MAC scheme with each of these messages to receive the corresponding tags $T_1, T_2, \ldots, T_N$, where each tag $T_i = (t_{i1}, t_{i2}, \ldots, t_{il})$ is composed of the block-wise MACs.
- Due to the birthday paradox, when the number of messages N is sufficiently large (approximately $2^{n/2}$, where n is the block size in bits), there is a high probability that two different messages M_i and M_j will produce the same tag for at least one block, i.e., $t_{ik} = t_{jk}$ for some k.
- Once the attacker finds such a collision, they can create a new valid message M' by combining blocks from M_i and M_j, and the corresponding tag will also be valid.

Block Swapping Attack

In the Block Swapping Attack, the attacker selects two messages, $M = (m_1, m_2)$ and $M' = (m_3, m_4)$, and queries the MAC scheme to receive the tags $T = (t_1, t_2)$ and $T' = (t_3, t_4)$. The attacker then generates a new message $M'' = (m_1, m_4)$ by swapping blocks between the two original messages. The new tag $T'' = (t_1, t_4)$ is formed by combining the corresponding tags from each original message.

Summary

Each of these attacks demonstrates how the lack of interdependence between blocks in the MAC scheme allows an attacker to manipulate known message/tag pairs to forge new valid pairs, ultimately compromising the security of the MAC scheme. By carefully selecting and manipulating blocks from known messages, the attacker can generate new messages with valid MACs.

6 Task E: Forging RSA Signature

Consider the textbook RSA signature scheme. Given the signatures for messages 3 and 5, we want to forge the signature for message 45. Here is the step-by-step process:

1. RSA Signature Scheme Overview

In the RSA signature scheme:

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

2. Given Information

Suppose the signatures for messages 3 and 5 are:

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

where d is the private key.

3. Goal

Forge the signature for message 45. We need to find s_{45} such that:

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

4. Use Modular Arithmetic Properties

Decompose 45 into its prime factors:

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

Thus:

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

5. Relate to Given Signatures

We know:

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

Thus:

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

$$5^d = s_5$$

Combining these:

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

6. Compute the Forged Signature

The forged signature s_{45} can be computed as:

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

7. Verification

To verify the forged signature, check:

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

should equal 45, where e is the public exponent.

Summary

- Obtain the signatures for 3 and 5: s_3 and s_5.
- Compute:

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

- The result s_{45} is the forged signature for message 45.

This method works because the RSA signature scheme's property of exponentiation and multiplication allows the creation of signatures for products of known messages, leveraging modular arithmetic.