Team:

Ahmed Mahmoud Hassan - 2205155

Ali Mohamed Oqab - 2205077

Rewan Ahmed Elwardany - 2205218

Background Study

What is a MAC and its purpose in data integrity/authentication?

A **Message Authentication Code (MAC)** is a cryptographic technique used to:

- Verify the **integrity** of a message
- Confirm the **authenticity** of the sender

The MAC is created using a **secret key** and the message content. It produces a fixed-size output called the MAC value. The recipient, who also knows the key, recalculates the MAC to verify the message.

Purpose:

- **Integrity** – Detects message modifications
- **Authentication** – Confirms sender identity
- **Non-repudiation** – Prevents sender denial (in some cases)

Common MAC types:

- **HMAC** (Hash-based MAC): Combines hash functions like SHA-256 with a secret key
- **CBC-MAC** (Cipher Block Chaining MAC): Uses block ciphers

---

How does a length extension attack work in hash functions like MD5/SHA1?

A **length extension attack** targets hash functions like MD5 and SHA-1 that use the Merkle-Damgård structure.

These hash functions process input in blocks. The internal state after processing one block is used for the next.

Steps:

1. Attacker knows:
   `H = hash(secret || message)`
   and the length of **`secret || message`**

2. Attacker appends new data:
   `message || padding || new_data`
   The padding aligns the total length to the block size.

3. Attacker uses **`H`** as the internal state and calculates:
   `H' = hash(H || padding || new_data)`

4. **`H'`** is valid for:
   `secret || message || padding || new_data`
   without knowing the secret.

Vulnerable functions:

- MD5
- SHA-1
- SHA-256

Safe against this: HMAC and other special constructions

Why is MAC = hash(secret || message) insecure?

Using **`MAC = hash(secret || message)`** is vulnerable to length extension attacks.

Why:

The hash's internal state after processing **`secret || message`** can be reused. Attackers can create valid MACs for longer messages without knowing the secret.

Example:

If
`MAC = SHA1(secret || message)`

An attacker can forge a valid MAC for:
`secret || message || padding || new_data`

Fixes:

- **HMAC** – Uses nested hashing to break the chain
  `HMAC = hash(outer_key || hash(inner_key || message))`

- **Truncation** – Shortens the MAC to hide internal states

- **Alternative MACs** – Use SHA-3-based KMAC or CBC-MAC