Background Study

Introduction

Message Authentication Codes (MACs) play a crucial role in ensuring the integrity and authenticity of digital communications. In this report, we explore a cryptographic vulnerability that arises when MACs are implemented insecurely using naive hash constructions. Specifically, we demonstrate a length extension attack against a MAC defined as:

```
MAC = hash(secret || message)
```

A. Message Authentication Code (MAC) and Its Purpose in Data Integrity and Authentication

A **Message Authentication Code (MAC)** is a fundamental cryptographic tool designed to verify both the **integrity** and **authenticity** of a message transmitted between parties sharing a secret key. It is a fixed-size tag or checksum generated from the message content combined with a secret key, and it provides a way to ensure that the message has not been altered and that it comes from a legitimate sender.

How MAC Works

When a sender wants to transmit a message securely, they use a MAC algorithm — typically based on cryptographic hash functions or block ciphers — along with a secret key shared with the receiver. The MAC is computed as follows:

```
MAC=MAC_Function(secret_key,message)
```

This MAC value is then appended to the original message and sent over the communication channel. Upon receiving the message and its MAC, the receiver uses the same secret key and MAC function to recompute the MAC from the received message. If the recomputed MAC matches the received MAC, the receiver can trust that:

- The message was not modified in transit (data integrity).
- The message was generated by someone possessing the shared secret key (authentication).

If the MACs do not match, this indicates potential tampering or an illegitimate sender.

Purpose of MAC

1. **Data Integrity:**
   The MAC ensures that the message content remains unchanged during transmission. Any modification, whether accidental or malicious, will cause the recomputed MAC at the receiver's side to differ from the MAC sent by the sender, thus signaling data corruption or tampering.
2. **Authentication:**
   Since only parties that possess the secret key can generate a valid MAC, it serves as proof that the message comes from a trusted source. This prevents unauthorized entities from injecting fake messages into the communication.

B. How Does a Length Extension Attack Work in Hash Functions Like MD5/SHA-1?

A **Length Extension Attack** exploits a vulnerability in certain hash functions like **MD5** and **SHA-1**, which are based on the **Merkle–Damgård construction**. In such hash designs, the internal state of the hash after processing some data can be used to continue hashing more data — without knowing the original input.

Attack Scenario:

Suppose an attacker knows:

- The hash of a message `H = hash(secret || message)`
- The length of the original message (or can guess it)

Then, without knowing the secret, the attacker can:

1. Append additional data `X` to the message.
2. Recalculate a valid hash of `secret || message || padding || X` using the internal state of the original hash.
3. Generate a new, valid MAC tag for this forged extended message.

This results in a valid-looking MAC, even though the attacker doesn't know the secret — defeating the integrity protection.

Implication:

Hash functions like MD5 and SHA-1 are **vulnerable** to length extension attacks when used improperly as MACs, making them unsafe for constructs like `MAC = hash(secret || message)`.

---

c. Why is MAC = hash(secret || message) Insecure?

Using `MAC = hash(secret || message)` as a Message Authentication Code is insecure, especially when the underlying hash function is vulnerable to length extension attacks, such as MD5 or SHA-1.

Reasons for insecurity:

1. **Predictable Structure:** The attacker can guess the format and structure of the input since the secret is simply prepended to the message.
2. **Key Exposure:** The secret key is not protected adequately because it is placed before the message in a predictable way, allowing manipulation of the input by attackers.
3. **Length Extension Vulnerability:** Hash functions like MD5 and SHA-1 follow the Merkle–Damgård construction, which allows an attacker to take the hash of an unknown secret concatenated with a message, then extend the message with additional data and compute a valid hash without knowing the secret key.

Secure Alternative:

To avoid this vulnerability, the **HMAC** construction is used:

$$\text{HMAC}(K,M)=H((K' \oplus \text{opad}) || H((K' \oplus \text{ipad}) || M))$$

where K′ is a block-sized key. HMAC protects against length extension attacks by hashing the key and message in two separate passes with additional padding and key mixing.