Background Study: MAC and Length Extension Attacks

1 What is a MAC and Its Purpose in Data Integrity/Authentication?

A Message Authentication Code (MAC) is a cryptographic checksum that ensures both *data integrity* and *authentication*. It is computed by applying a cryptographic function to a message combined with a secret key. The MAC is sent alongside the message, allowing the recipient to verify its integrity and authenticity by recomputing the MAC with the shared key. If the recomputed MAC matches the received MAC, it confirms:

- **Data Integrity**: The message has not been altered during transit. Any tampering with the message or MAC results in a mismatch.
- **Authentication**: The message originates from a party possessing the secret key.

1.1 How MACs Work

The process involves two steps:

- **Generation**: Compute $\text{MAC} = F(\text{secret\_key}, \text{message})$, where F is a cryptographic function (e.g., hash function or block cipher).
- **Verification**: The recipient computes $\text{MAC}' = F(\text{secret\_key}, \text{message})$ and checks if $\text{MAC}' = \text{MAC}$.

1.2 Applications

MACs are widely used in:

- **Network Security**: Protocols like SSL/TLS, SSH, and IPsec.
- **Data Storage**: Protecting files and database records.
- **APIs and Webhooks**: Verifying trusted request sources.

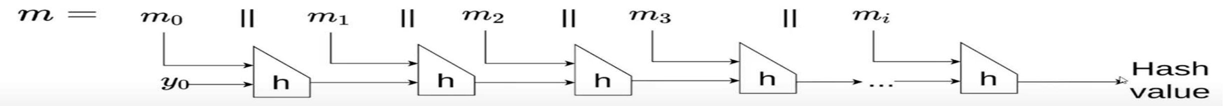
1.3 Types of MACs

- **HMAC (Hash-based MAC)**: Uses hash functions (e.g., SHA-256) in a secure construction.
- **CMAC (Cipher-based MAC)**: Employs block ciphers like AES.

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

A length extension attack exploits the Merkle–Damgård construction used by hash functions like MD5, SHA-1, and SHA-256. These functions process input in fixed-size blocks, maintaining an internal state.

In a Merkle-Damgard system, the hash value is the last chaining value:



2.1 How the Attack Works

When computing hash(secret || message), the hash function processes the secret and message, producing a hash. An attacker who knows the hash and the secret's length (but not the secret) can:

1. Use the known hash as the starting internal state.
2. Append additional data (e.g., || extra), apply padding, and continue hashing.
3. Compute a valid hash for hash(secret || message || padding || extra).

2.2 Real-World Impact

This allows attackers to forge valid MACs for extended messages. For example, if a banking API uses MAC = MD5(secret || ”transfer $100”), an attacker could compute a valid MAC for MD5(secret || ”transfer $100” || ”&& transfer $1000 to attacker”), compromising integrity and authenticity.

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

Using MAC = hash(secret || message) is insecure due to vulnerabilities in hash function design, particularly for MD5 and SHA-1.

3.1 Main Reasons for Insecurity

- **Length Extension Vulnerability**: Attackers can forge MACs for extended messages, as described above.
- **Key Placement**: The secret is only prefixed (or suffixed), allowing exploitation of the hash's iterative nature.
- **Collision Attacks**: Non-collision-resistant hash functions allow attackers to find different messages with the same MAC.

3.2 Secure Alternatives

HMAC (Hash-based Message Authentication Code) addresses these issues:

- Mixes the key with the message using two pads (inner and outer).
- Proven secure against length extension attacks if the hash function is secure.

HMAC is defined as:

$$\text{HMAC}(\text{key}, \text{message}) = \text{hash}((\text{key} \oplus \text{opad}) \parallel \text{hash}((\text{key} \oplus \text{ipad}) \parallel \text{message}))$$

4 Conclusion

The construction MAC = hash(secret || message) is insecure for hash functions like MD5 and SHA-1 due to length extension attacks. Secure systems should use HMAC or other robust MAC constructions to ensure data integrity and authentication.