

# MAC Forgery and Mitigation Report

---

## Team members:

Shadwa ahmed :2205026

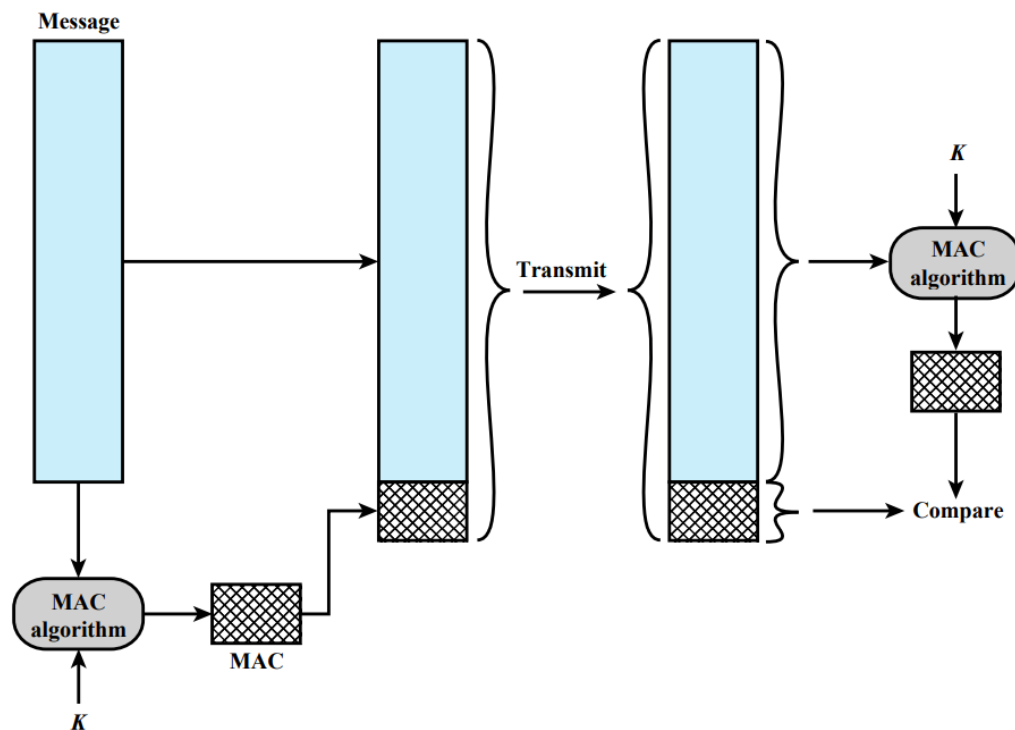
Sara ahmed:2205094

Yehia tarek:2205062

## 1. Background

### 1.1 What is a MAC?

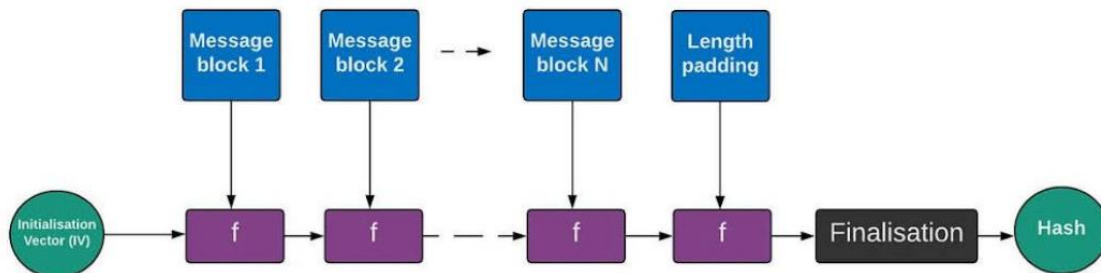
A Message Authentication Code (MAC) is a cryptographic checksum used to ensure both the integrity and authenticity of a message. MACs are generated using a secret key and the message content. Only parties who share the secret key can generate or verify the MAC. If the MAC is valid, the recipient can be confident the message has not been altered.



**Figure:** Message Authentication Code (MAC) generation and verification process. The sender uses a shared secret key  $K$  to generate the MAC. The receiver uses the same key to validate the message integrity.

## 1.2 How does a Length Extension Attack work?

Length Extension Attacks exploit the internal structure of certain hash functions such as MD5 and SHA1. When a MAC is implemented as  $\text{hash}(\text{secret} || \text{message})$ , an attacker who knows the hash output for some message can append extra data to the message and compute a valid MAC for the new message — all without knowing the secret key.



**Figure:** Hash functions like MD5 and SHA1 process input in a block-wise manner using a compression function. Length extension attacks exploit this structure by resuming the hashing process from an intermediate state using additional data and padding.

## 1.3 Why is $\text{MAC} = \text{hash}(\text{secret} || \text{message})$ insecure?

This construction is insecure because it allows attackers to use length extension techniques. Hash functions process data in blocks, and intermediate hash states can be extended to forge valid MACs. Since the attacker knows  $\text{hash}(\text{secret} || \text{message})$  and the original message, they can compute  $\text{hash}(\text{secret} || \text{message} || \text{padding} || \text{extra\_data})$ .

## 2. Attack Demonstration

In our demonstration, we use a vulnerable MAC implementation that computes the MAC as  $\text{SHA256}(\text{secret} || \text{message})$ . We intercept a valid message and its MAC, then use the 'hashpumpy' tool to forge a new MAC after appending additional data.

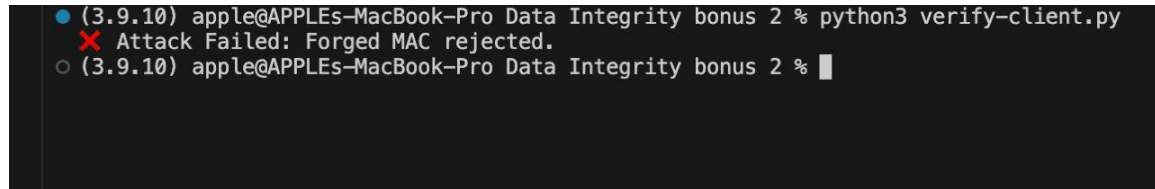
The attack succeeds when the forged message and MAC are accepted by the server, proving the vulnerability of the construction.



### 3. Mitigated Secure Implementation

To mitigate the length extension attack, we replace the insecure MAC with a proper HMAC construction. HMAC uses a nested hashing approach:  $H((K \oplus \text{opad}) \parallel H((K \oplus \text{ipad}) \parallel \text{message}))$ . This prevents attackers from accessing internal hash states.

We implemented the secure version in 'server-secure.py' using the HMAC module. Attempts to replay the same attack against this version failed, demonstrating the effectiveness of the mitigation.

A terminal window with a dark background. The prompt is '(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 %'. The user has entered 'python3 verify-client.py'. The output is 'Attack Failed: Forged MAC rejected.' followed by a new line and the prompt again, where a cursor is visible.

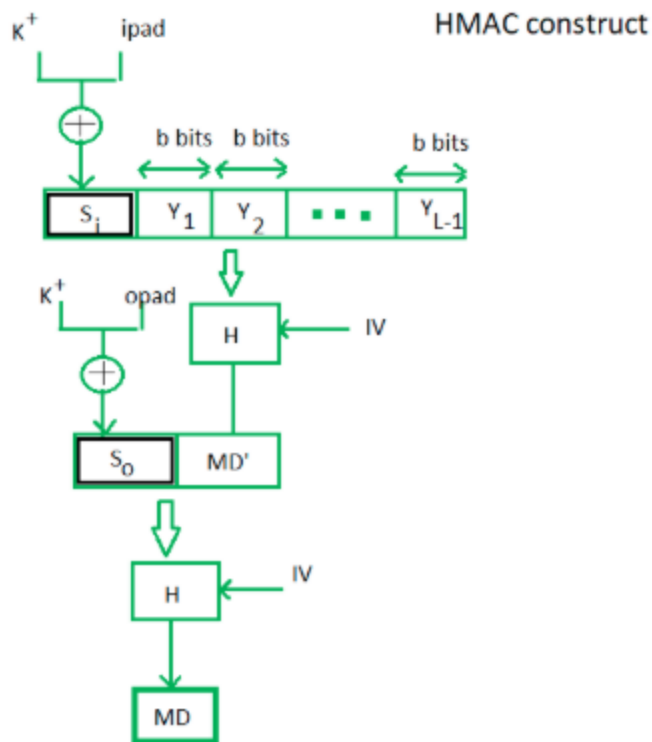
```
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 % python3 verify-client.py
✗ Attack Failed: Forged MAC rejected.
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 %
```

Figure: The secure HMAC-based server correctly verifies the original message. This confirms that the system works as expected when not under attack.

## 4. Mitigation Write-up

### 4.1 Why HMAC is Secure

HMAC includes the secret key in both the inner and outer hashes, making it immune to length extension attacks. The use of opad and ipad constants prevents attackers from controlling the internal state of the hash function.



**Figure:** HMAC construction uses nested hashing with a secret key, an inner padded hash, and an outer padded hash. This structure prevents length extension attacks and ensures message authenticity.

## 4.2 Demonstration Results

Our results show that the insecure server accepts forged MACs generated via hashpumpy. In contrast, the secure HMAC implementation rejects all forged attempts, as expected.

[illegible]

Figure: Attack attempt using a forged MAC fails against the HMAC-secured server. The message is rejected, confirming that the system is no longer vulnerable.

```
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 % python3 server-secure.py
=== Secure Server Simulation ===
Original message: amount=100&to=alice
HMAC: 616843154afc11960423deb0795b1e68

--- Verifying message ---
✅ Message is authentic and verified.
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 %
```

Figure: Full demonstration sequence showing success of attack on the insecure server and failure of the same attack on the secure HMAC-based server. This validates the effectiveness of HMAC as a mitigation.