Data Integrity

BOUNS ASSIGNMENT

Supervised by :

Dr.Maged Abdelaty

Mahmoud Zaghloula 2205055
Abdelrahman Hisham 2205032
Yehia Ahmed Tawfik 2205126

Background Study on MAC and Length Extension Attacks

1.1 Message Authentication Code (MAC) and Its Purpose

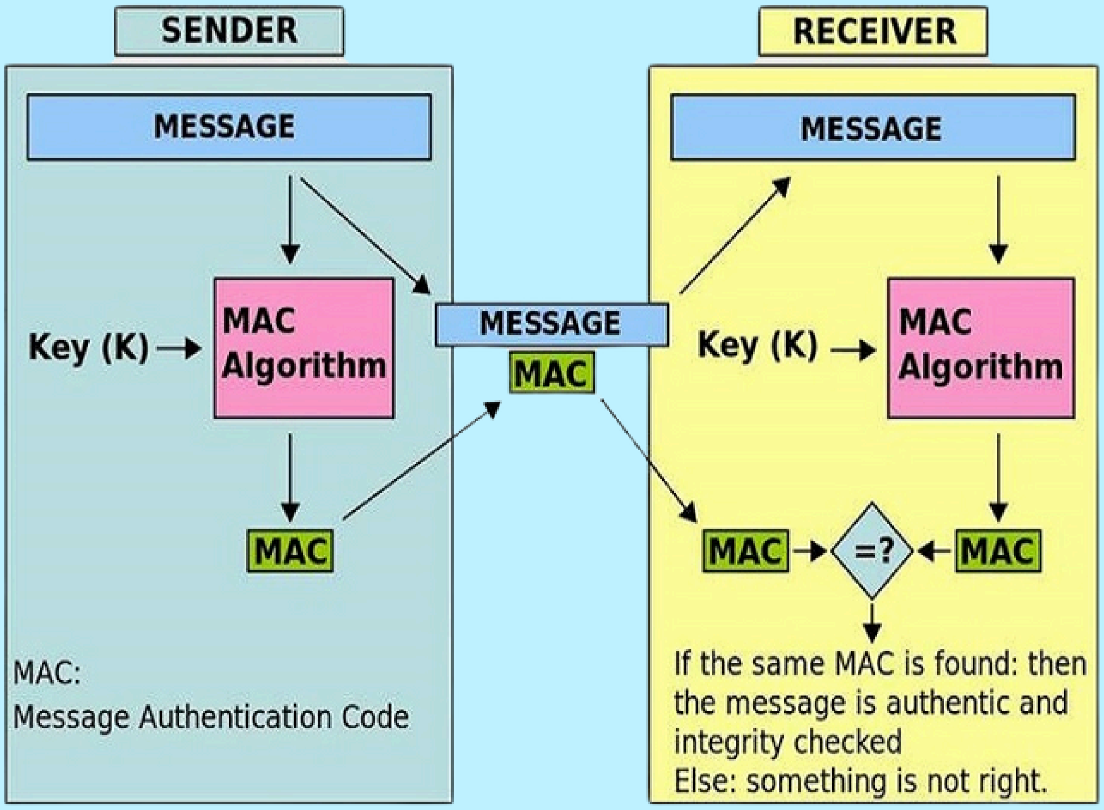
A Message Authentication Code (MAC) is a cryptographic mechanism that ensures the integrity and authenticity of a message. By combining a secret key with the message using a cryptographic hash function (e.g., SHA-256) or symmetric cipher (e.g., AES-CMAC), a MAC generates a fixed-length tag transmitted alongside the message. The recipient, sharing the same secret key, recomputes the MAC to verify two key properties: data integrity, confirming the message has not been altered during transmission, and authentication, ensuring it originates from a trusted source. MACs are essential in secure communication protocols like TLS, IPsec, and API authentication, protecting critical operations such as transaction requests (e.g., \`amount=100&to=alice\`) from tampering or forgery. Unlike asymmetric digital signatures, MACs use symmetric cryptography, offering computational efficiency for high-performance applications. However, their security relies on the strength of the underlying cryptographic primitive and proper implementation, as flaws in the construction can compromise their effectiveness. The secret key must be securely shared between sender and receiver to maintain these guarantees, making MACs a cornerstone of secure systems where unauthorized modifications or impersonation could lead to significant breaches.

1.2 Length Extension Attacks on Hash Functions

Length extension attacks exploit a structural vulnerability in hash functions like MD5 and SHA-1, which rely on the Merkle-Damgård construction. This design processes input messages in fixed-size blocks (e.g., 512 bits for SHA-1), updating an internal state with each block and finalizing it to produce the hash output. The vulnerability stems from the hash output directly exposing the final internal state. If an attacker knows the hash H(secret || message) and the length of the secret, they can append new data (e.g., &admin=true) and compute a valid hash for the extended message H(secret || message || padding || append) without needing the secret key. The attacker uses the known hash as the initial state, adds the new data with appropriate padding to align with block boundaries, and continues the hashing process. Padding is critical, as Merkle-Damgård functions append a length field that the attacker must account for. Tools like hashpumpy automate this by calculating the correct padding and new hash. For example, given a message "amount=100&to=alice" and its hash H(secret || message), an attacker can append "&admin=true" and generate a valid hash for the extended message. This vulnerability is particularly dangerous when hash functions are used as MACs, enabling attackers to forge messages that bypass integrity and authentication checks.

1.3 Insecurity of MAC = hash(secret || message)

Constructing a MAC as MAC = hash(secret || message) is insecure due to its vulnerability to length extension attacks. Hash functions like MD5 and SHA-1, often used in this approach, were not designed for MAC security and expose their internal state in the hash output. This allows an attacker who intercepts a valid (message, MAC) pair to initialize the hash function with the MAC, append new data, and compute a valid MAC for the extended message without knowing the secret key. For example, consider a web server generating a MAC for the transaction request "amount=100&to=alice" using hashlib.md5(secret + message).hexdigest(). An attacker, knowing the secret's length (e.g., 14 bytes), can perform a length extension attack to append "&admin=true". By accounting for MD5's Merkle-Damgård padding, the attacker produces a valid MAC for the tampered message, which the server accepts. This compromises the system's integrity and authentication, enabling unauthorized actions like granting administrative privileges or altering transactions. Secure alternatives, such as HMAC, mitigate this vulnerability with a robust keying mechanism.


SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.