Background Report: Message Authentication Codes and Length Extension Attacks

1. Message Authentication Codes (MACs) and Their Purpose

Message Authentication Codes (MACs) are cryptographic constructs designed to ensure both the **integrity** and **authenticity** of data. When a sender transmits a message to a receiver, a MAC provides verification that:

1. The message has not been altered during transmission (integrity)
2. The message originated from the claimed sender (authenticity)

MACs accomplish this by generating a fixed-size tag derived from both the message and a secret key shared between the sender and receiver. The basic process works as follows:

1. The sender computes a MAC tag using the secret key and the message
2. The sender transmits both the message and its MAC tag
3. The receiver recomputes the MAC using the same key and received message
4. If the computed MAC matches the received MAC, the message is verified as authentic

Unlike digital signatures, MACs rely on symmetric cryptography where both parties share the same secret key. This makes MACs computationally efficient but requires secure key distribution between communicating parties.

2. Length Extension Attacks in Hash Functions

Length extension attacks exploit a fundamental vulnerability in Merkle–Damgård-based hash functions such as MD5, SHA-1, and SHA-2 (excluding SHA-3). These hash functions process data in fixed-size blocks and maintain an internal state that is updated with each block. After processing all blocks, this internal state becomes the final hash output.

The attack works as follows:

1. An attacker observes a hash value H = hash(M) for some unknown message M
2. Based on the hash function's structure and knowing the length of M, the attacker can compute hash(M || padding || M') for any additional data M' without knowing the original M
3. This works because the hash value H represents the internal state of the hash function after processing M
4. The attacker can continue hashing from this state, effectively "extending" the original message

This vulnerability becomes particularly dangerous when applied to authentication schemes.

3. Insecurity of MAC = hash(secret || message)

The naive MAC construction `MAC = hash(secret || message)` is vulnerable to length extension attacks precisely because of how Merkle–Damgård hash functions operate.

Consider this scenario:

1. A server computes `MAC = hash(secret || message)` and shares the message and MAC publicly
2. An attacker who doesn't know the secret key obtains this MAC value
3. The MAC value represents the internal state of the hash function after processing `secret || message`
4. Using length extension, the attacker can compute `hash(secret || message || padding || extension)` without knowing the secret
5. The attacker can now forge a valid MAC for the extended message `message || padding || extension`

This attack succeeds because:

1. The hash function processes data sequentially in blocks
2. The hash output reveals the internal state after processing the original data
3. The attacker can resume hashing from this state to extend the message
4. The server's verification will compute the same extended hash, resulting in a successful forgery

This vulnerability undermines the fundamental security properties of the MAC, allowing an attacker to create valid authentication tags for messages that were never processed by the legitimate sender.

Recommended Secure Alternatives

To mitigate length extension attacks, several secure MAC constructions have been developed:

1. **HMAC (Hash-based Message Authentication Code)**:

   ```
   HMAC(K, m) = hash((K' ⊕ opad) || hash((K' ⊕ ipad) || m))
   ```

   The nested hash structure prevents length extension by isolating the hash state from the attacker.
2. **KMAC (Keccak Message Authentication Code)**: Using SHA-3 or its variants, which are not vulnerable to length extension attacks due to their sponge construction.
3. **Encrypt-then-MAC**: Using a secure encryption algorithm followed by a MAC on the ciphertext.

4. **Alternative MAC constructions**:
   - `MAC = hash(secret || message || secret)`
   - `MAC = hash(secret || hash(secret || message))`

Of these alternatives, HMAC has emerged as the standard, offering provable security properties and wide implementation across cryptographic libraries.

References

1. Krawczyk, H., Bellare, M., & Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. RFC 2104. https://tools.ietf.org/html/rfc2104
2. Rivest, R. L. (1992). The MD5 Message-Digest Algorithm. RFC 1321. https://tools.ietf.org/html/rfc1321
3. NIST. (2015). Secure Hash Standard (SHS). FIPS PUB 180-4. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
4. Boneh, D., & Shoup, V. (2020). A Graduate Course in Applied Cryptography. Retrieved from https://crypto.stanford.edu/~dabo/cryptobook/
5. SkullSecurity. (2012). Everything you need to know about hash length extension attacks. https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks
6. Wang, X., Yin, Y. L., & Yu, H. (2005). Finding collisions in the full SHA-1. In Advances in Cryptology – CRYPTO 2005. Springer.