Assignment #2 Report

1. Andreandhiki Riyanta Putra (23/517511/PA/22191)
2. Andrian Danar Perdana (23/513040/PA/21917)
3. Daffa Indra Wibowo (23/518514/PA/22253)
4. Muhammad Argya Vityasy (23/522547/PA/22475)

How HMAC differs from plain hashing:

1. **Use of a Secret Key:**

   Plain Hashing: A standard cryptographic hash function (like SHA-256) takes a single input (the message) and produces a fixed-size digest (the hash). The formula is essentially Hash = H(message). Anyone who has access to the message can compute this hash.

   HMAC (Hash-based Message Authentication Code): HMAC incorporates a secret cryptographic key into the hashing process. The general HMAC construction is HMAC(K, m) = H((K_0 XOR opad) || H((K_0 XOR ipad) || m)), where K is the secret key, m is the message, H is the chosen hash function, and ipad and opad are specific padding constants. K_0 is the key K processed to fit the block size of the hash function (hashed if too long, padded if too short). Crucially, only parties possessing the secret key K can generate or correctly verify the HMAC.

2. **Purpose and Guarantees:**

   Plain Hashing: Primarily provides **data integrity**. It can detect if the data has been accidentally or maliciously altered after the hash was computed. However, it does *not* provide **authentication** on its own because anyone can compute the hash of any message. If an attacker changes a message, they can simply recompute the plain hash for the modified message.

   HMAC: Provides both **data integrity** (like a plain hash) AND **message authentication**.

   a. **Integrity:** If the message is changed, the HMAC will change.
   b. **Authentication:** Because the HMAC calculation depends on the shared secret key, a valid HMAC tag verifies that the message originated from a party that knows this secret key. It confirms the *source* of the message, not just its state.

3. **Security Against Specific Attacks:**

   Plain hashes, if used naively for authentication (e.g., H(secret || message) or H(message || secret)), can be vulnerable to certain attacks like "length

extension attacks" (for H(secret || message)) or offline hashing of the secret if the hash output is known.

The HMAC construction (with its specific nested hashing and use of ipad and opad) was formally proven to be secure (a PRF - Pseudorandom Function) as long as the underlying hash function H meets certain cryptographic properties (e.g., collision resistance). This design specifically thwarts length extension attacks and other common pitfalls of ad-hoc keyed hashing.

Why HMAC is resistant to forgery (include an example where verification fails due to tampering):

HMAC's resistance to forgery is fundamentally tied to the secrecy of the shared key. An attacker who does not know the key cannot produce a valid HMAC tag for a message of their choosing (or a modified message).

1. **Inability to Compute a Valid MAC without the Key:**

   If an attacker intercepts a message M and its legitimate HMAC tag T = HMAC(K, M), and they modify M to M', they cannot compute the correct new tag T' = HMAC(K, M') because they lack K.

   If they simply send (M', T) (tampered message, original tag), the receiver will compute HMAC(K, M') using their copy of K. This computed tag will not match the received tag T, and the verification will fail. This is demonstrated in **Scenario 1** in the code above:

   Legitimate Sender (Alice):

   - Message (M): "Transfer $100 to Bob."
   - Secret Key (K): shared_secret_key
   - HMAC (Tag_Alice): generate_hmac_sha256_custom(K, M)
   - Alice sends (M, Tag_Alice) to the Bank.

   Attacker (Eve):

   - Intercepts (M, Tag_Alice).
   - Changes M to M' ("Transfer $1000 to Eve.").
   - Eve does *not* know K. She cannot compute Tag_Eve_Correct = generate_hmac_sha256_custom(K, M').
   - Eve sends (M', Tag_Alice) to the Bank (i.e., tampered message with the original, now incorrect, MAC).

   Receiver (Bank):

   - Receives (M', Tag_Alice).
   - The Bank knows K.

- Bank computes Tag_Bank_Computed = generate_hmac_sha256_custom(K, M').
- Bank compares Tag_Bank_Computed with the received Tag_Alice. Since M' is different from M, Tag_Bank_Computed will (with overwhelming probability) be different from Tag_Alice.
- Verification fails. The bank detects tampering and rejects the fraudulent transaction. The output in Scenario 1 shows: "HMAC VERIFICATION FAILED (Custom)! Message may have been tampered with or key is incorrect."

2. **Difficulty of Guessing/Predicting HMAC Output:**

   Cryptographic hash functions (like SHA-256) used within HMAC are designed to be one-way (hard to reverse) and to exhibit a strong "avalanche effect" (a small change in input drastically changes the output).

   This makes it computationally infeasible for an attacker to guess a valid HMAC tag for a given message without knowing the key, or to find a different message that produces the same HMAC tag (a collision related to the key).

3. **Protection Against Replay (Requires Additional Mechanisms):**

   It's important to note that HMAC itself authenticates the origin and integrity of a *single* message. It does not inherently protect against "replay attacks," where an attacker resends a valid, previously captured message and its HMAC.

   To prevent replay attacks, systems using HMAC often incorporate additional mechanisms like sequence numbers, timestamps, or nonces within the message content *before* the HMAC is computed. These make each message unique, so a replayed message would be detected.

Why using a shared secret key is crucial:

The shared secret key is the absolute linchpin of HMAC's security. Its cruciality can be highlighted as follows:

1. **Establishes the Basis of Authentication:** The key is the shared secret that forms the trust relationship between the communicating parties. Only entities possessing this key can generate a valid HMAC. If there's no secret, or if the key is public knowledge, the HMAC offers no more authentication than a plain hash – anyone could generate it.
2. **Enables Source Verification:** A valid HMAC tag assures the receiver that the message came from someone who knows the key. This is the "authentication" part of "Message Authentication Code." Without the key, this assurance vanishes.

3. **Provides the "MAC" in HMAC:** The "MA" in HMAC stands for Message Authentication. This authentication is entirely predicated on the key being secret and shared only among authorized parties.
4. **Prevents Forgery:** As detailed above, the inability of an attacker (who doesn't know the key) to compute a valid HMAC for an arbitrary or modified message is the primary defense against forgery. The secrecy of the key is what makes this defense effective.
5. **Integrity Becomes Authenticated Integrity:** While a plain hash provides integrity, HMAC provides *authenticated integrity*. This means the receiver not only knows the message hasn't changed since the HMAC was computed but also trusts that it was computed by a legitimate party (one knowing the key).

Consequences of a Compromised Shared Secret Key: If the shared secret key is compromised (i.e., an unauthorized party learns it):

- **Complete Loss of Authentication:** The attacker can now generate valid HMACs for any fraudulent messages they create. These messages will appear legitimate to the receiver.
- **Undetectable Tampering:** The attacker can intercept legitimate messages, modify them, and then re-calculate and attach a new, valid HMAC using the compromised key. This tampering would be undetectable by the HMAC verification process alone.
- **Impersonation:** The attacker can impersonate any of the legitimate parties who share that key.
- The security provided by HMAC for that specific key is entirely nullified. All past and future messages secured with that compromised key are suspect.

Therefore, the security of the HMAC system is critically dependent on the security of the shared secret key. Robust key management practices are essential:

- **Secure Generation:** Keys should be generated using a cryptographically secure random number generator.
- **Secure Distribution:** Keys must be shared between parties over a secure channel.
- **Secure Storage:** Keys must be protected from unauthorized access, both at rest and in memory.
- **Limited Scope/Lifetime:** Keys should ideally be used for a limited purpose or time, and rotated periodically.