Name: Shrey Pendurkar
Class: D15C
Batch: C
Roll No: 64

CNS - Experiment 7

Aim: Write a Program to implement Cryptographic Hash Functions and Applications (HMAC): To Understand the need, design and applications of collision resistant hash functions.

Theory:

HMAC (Hash-based Message Authentication Code) is a type of a message authentication code (MAC) that is acquired by executing a cryptographic hash function on the data (that is) to be authenticated and a secret shared key. Like any of the MAC, it is used for both data integrity and authentication. Checking data integrity is necessary for the parties involved in communication. HTTPS, SFTP, FTPS, and other transfer protocols use HMAC. The cryptographic hash function may be MD-5, SHA-1, or SHA-256. Digital signatures are nearly similar to HMACs i.e they both employ a hash function and a shared key. The difference lies in the keys i.e HMACs use symmetric key(same copy) while Signatures use asymmetric (two different keys).

Applications of HMAC:-

Verification of e-mail address during activation or creation of an account.

Authentication of form data that is sent to the client browser and then submitted back.

HMACs can be used for Internet of things (IoT) due to less cost.

Whenever there is a need to reset the password, a link that can be used once is sent without adding a server state.

It can take a message of any length and convert it into a fixed-length message digest. That is even if you got a long message, the message digest will be small and thus permits maximizing bandwidth.

HMACs provide clients and servers with a shared private key that is known only to them. The client makes a unique hash (HMAC) for every request. When the client requests the server, it hashes the requested data with a private key and sends it as a part of the request. Both the message and key are hashed in separate steps making it secure. When the server receives the request, it makes its own HMAC. Both the HMACS are compared and if both are equal, the client is considered legitimate.

Review Questions

1. **Why is HMAC considered more secure than just using a hash function for message integrity?**

   HMAC (Hash-based Message Authentication Code) is considered more secure than directly using a hash function because a plain hash function only ensures data integrity but not authenticity. If only hashing is used, an attacker who knows the data can recompute the hash easily. In contrast, HMAC combines a cryptographic hash function with a secret key, which means only parties who know the secret key can generate or verify the MAC. This prevents attackers from forging valid authentication codes even if they know the hash algorithm and the message. Hence, HMAC provides both integrity and authenticity.

2. **Explain the general structure of an HMAC. How is it calculated?**

   The general structure of HMAC involves two stages of hashing with the secret key combined with the message using specific padding. It is calculated as:

   $$HMAC(K, M) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, M))$$

   Where:
   - K = secret key
   - M = message
   - H = cryptographic hash function (e.g., SHA-256)
   - ipad = inner padding (0x36 repeated block size times)
   - opad = outer padding (0x5C repeated block size times)
   - || = concatenation

   So first, the key is XORed with the ipad and concatenated with the message, then hashed. The result is concatenated with the key XORed with opad, and hashed again. This two-layer hashing prevents attacks like extension attacks.

3. **What role does the secret key play in the HMAC process?**

   The secret key is the critical element that provides authentication in HMAC. It ensures that only entities who possess the key can compute a valid HMAC for a message. Without the key, even if the attacker knows the hash function and the message, they

cannot generate the correct HMAC. Thus, the secret key prevents forgery and provides proof that the message came from an authenticated source.

4. **Describe the process of key padding in HMAC. Why is it necessary?**

   The secret key must match the block size of the hash function. The key padding process ensures this:
   - If the key is longer than the block size, it is first hashed to reduce its length.
   - If the key is shorter than the block size, it is padded with zeros to make it exactly the block size.

   This step is necessary because the XOR operation with ipad and opad requires the key to be the same length as the hash function's block size. Without padding, the calculation would be inconsistent and insecure. Padding guarantees a uniform input size and proper mixing of the key with the inner and outer padding.

Code:

```
import hashlib
import hmac
import os

def generate_hmac(key, message):
    """Generates HMAC for a given message using the provided key."""
    return hmac.new(key.encode(), message.encode(), hashlib.sha256).hexdigest()

def main():
    # Generate a random key (for demonstration; in practice, use a secure key management strategy)
    key = os.urandom(16).hex()  # 16 bytes key for HMAC
    print(f"Generated Key: {key}")

    # Example message
    message = "This is a secret message."

    # Generate HMAC
    hmac_value = generate_hmac(key, message)
    print(f"Message: {message}")
    print(f"HMAC: {hmac_value}")

    # Simulate verifying the HMAC
    message_to_verify = "This is a secret message."
    received_hmac = hmac_value  # Assuming this was received with the message

    # Generate HMAC for the received message
```

```
    generated_hmac = generate_hmac(key, message_to_verify)

    if hmac.compare_digest(received_hmac, generated_hmac):
        print("HMAC verification successful: The message is authentic.")
    else:
        print("HMAC verification failed: The message has been altered.")

if __name__ == "__main__":
    main()
```

Output:

```
Generated Key: d7a4e27768d92f44f8606c0856752883
Message: This is a secret message.
HMAC: 3c6f73a5eaa9e023798c7981658e74ac9796991e31376e1ad872424666f01bb8
HMAC verification successful: The message is authentic.
```