Task 1: Report

Student Information

Frederic Hamelink - 20204030
Anass Hamzaoui - 20210294

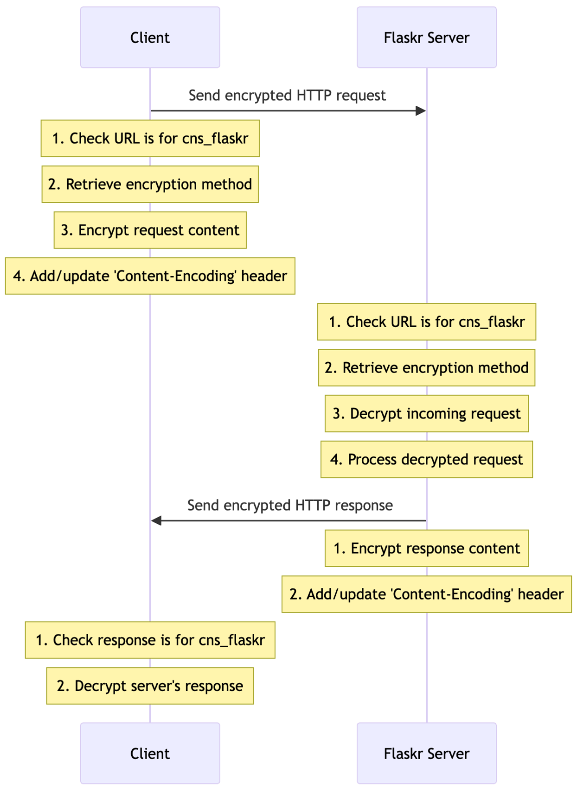
Task 1.1

Implementation

AES: Used an existing implementation from the Hashlib library.
Salsa20: Using specifications from slides given in class and pseudo-code found on Wikipedia.

How we ensure confidentiality

1. The content is encrypted using Salsa20 or AES.
2. Use of a preshared key ensures only parties with this key can decrypt the content.
3. We only process traffic for http://cns_flaskr/
4. We update (or insert) Content Encoding Header, so the receiving party can decrypt the content.
5. The receiving party is the Flaskr server, which is equipped to decrypt the content following the instructions above.
6. And vice versa (other direction)


Client
Flaskr Server
Send encrypted HTTP request
1. Check URL is for cns_flaskr
2. Retrieve encryption method
3. Encrypt request content
4. Add/update 'Content-Encoding' header
1. Check URL is for cns_flaskr
2. Retrieve encryption method
3. Decrypt incoming request
4. Process decrypted request
Send encrypted HTTP response
1. Encrypt response content
2. Add/update 'Content-Encoding' header
1. Check response is for cns_flaskr
2. Decrypt server's response
Client
Flaskr Server

MitM Attack Overview

An attacker intercepts communication between two parties, relaying and potentially altering the conversation.

1. Interception: Intercept the communication between the client and the Flaskr server without detection. This can be done using the cns_student proxy.
2. Key Reuse Attack:
    a. Observation: Note if the client sends multiple messages using the same encryption key and nonce.
    b. Exploit: Use XOR on two ciphertexts encrypted with the same key to reveal the XOR of their plaintexts.
3. Dictionary Attack:
    a. Observation: Identify if the client uses a simple encryption key.
    b. Dictionary Preparation: Prepare a dictionary of commonly used words.
    c. Brute-force: Try decrypting the message using each word in the dictionary as a potential key. Since the content is in key=value format, look for deciphered content that matches this pattern.
    d. Exploit: Once the right key is identified, modify or re-encrypt the client's future messages.
4. Traffic Manipulation: Using the identified vulnerabilities, manipulate the data packets or alter the content of messages.
5. Re-encryption and Forwarding: Encrypting the altered message and sending it to the intended recipient.
6. Defense: Always use unique encryption keys, pick strong keys, and monitor traffic for anomalies to combat MiTM threats.

Task 1.2

Implementation

MAC:

H(K+N+C) where H is the SHA1 hashing algorithm, K is a key and N is a random nonce. We used the SHA1 given in class. We added padding according to the specs given in the slides. It first converts the message to its binary representation. A '1' bit is appended, followed by enough '0' bits to ensure the length is 448 mod 512. Finally, a 64-bit binary representation of the original message length is appended to the end.

HMAC:

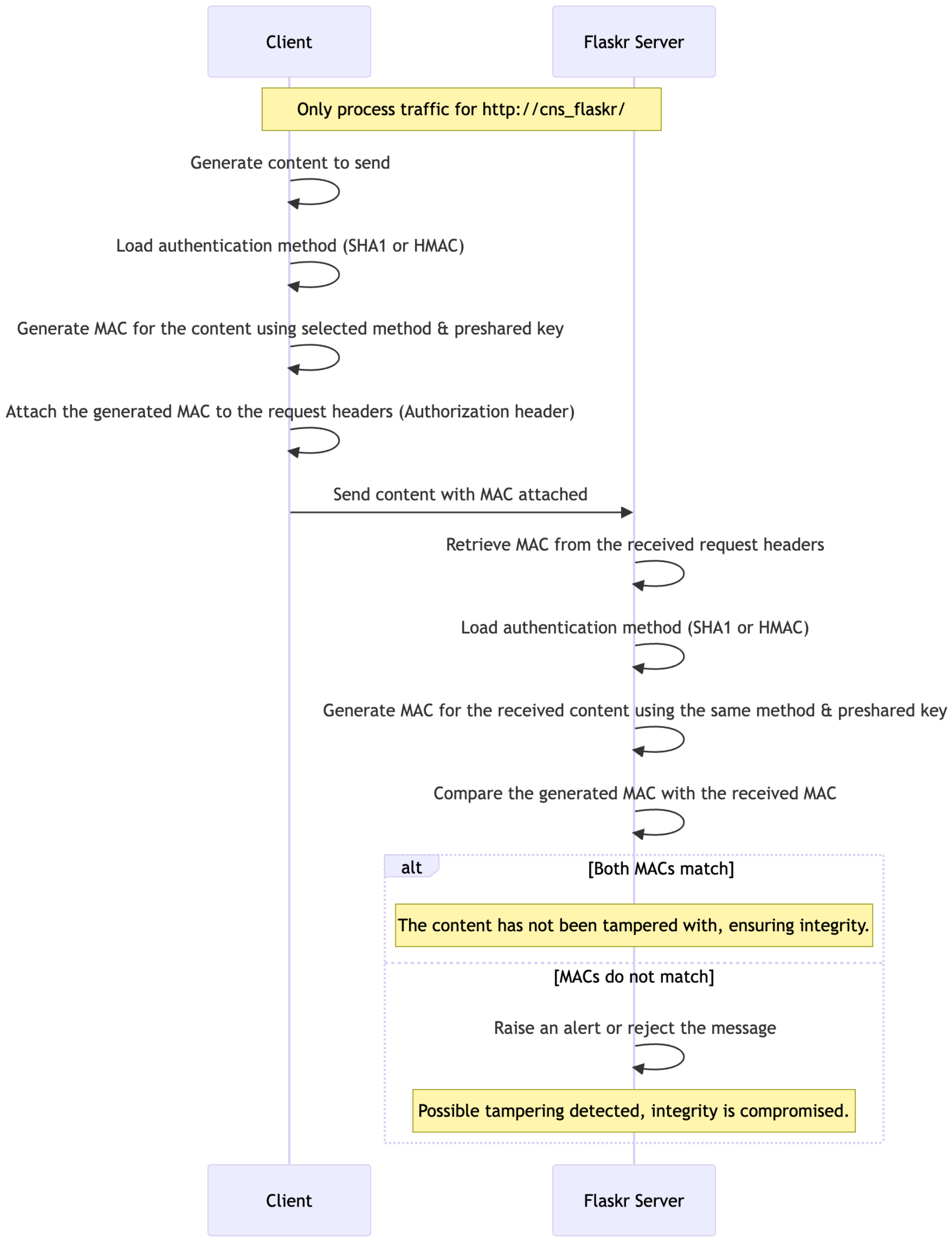
This HMAC implementation uses the SHA-512 hashing algorithm. It creates an HMAC by first XORing the key with inner and outer padding values, then hashing the concatenated results of the inner-padded key, nonce, and content, and finally hashing the concatenated results of the outer-padded key and the previous hash. If the key is longer than the block size, it's hashed; if shorter, it's padded. Specs were given in class. Both the key and nonce are checked on length and if necessary either shortened or lengthened so no problems occur when the hashing is done.

HTTP-Traffic:

Authorization needs to be added to the existing HTTP traffic to make use of the implemented MAC and HMAC as specified in the config files. in both the client and flaskr request and response we will authenticate after decrypting/encrypting. The authentication is pretty simple, first the necessary request headers are made as specified in the given mac_spec.md. Then the content we need to send is created and we pass it along to the MAC/HMAC generation to create our mac. This same mac will be checked in the response authorization to see if the content has been tampered with. There are extra checks done besides the mac like checking the timestamps of both the request and response authentication to see if it has been too long in between them.

Integrity of Traffic

1. The content is authenticated using MAC with either SHA1 or HMAC.
2. Use of a preshared key ensures that only parties with this key can validate the content's integrity.
3. We only process traffic for http://cns_flaskr/.
4. The computed MAC is attached to the request headers (Authorization header) to validate the data's integrity.
5. Timestamps ("X-Authorization-Timestamp") are included in the headers to prevent replay attacks and ensure that the message is recent.
6. The Flaskr server verifies the received MAC against its own computed MAC to ensure the content hasn't been tampered with.
7. And vice versa (other direction)


Client
Flaskr Server
Only process traffic for http://cns_flaskr/
Generate content to send
Load authentication method (SHA1 or HMAC)
Generate MAC for the content using selected method & preshared key
Attach the generated MAC to the request headers (Authorization header)
Send content with MAC attached
Retrieve MAC from the received request headers
Load authentication method (SHA1 or HMAC)
Generate MAC for the received content using the same method & preshared key
Compare the generated MAC with the received MAC
alt
[Both MACs match]
The content has not been tampered with, ensuring integrity.
[MACs do not match]
Raise an alert or reject the message
Possible tampering detected, integrity is compromised.
Client
Flaskr Server