Secure File Transfer Application

Application Design

Author: Shinsaku Segawa

Protocol Version: 1.0

A.Overview

This application consists of a client program and a server program. The server and client communicate to send, receive and manage files securely.
File content, file name and password must be encrypted so that the attacker cannot gain any information about them.

The application implements the following operations.

- MKD - creating a folder on the server
- RMD - removing a folder from the server
- GWD - presenting the name of the current folder on the server
- CWD - changing the current folder folder on the server
- LST - listing the content of a folder on the server
- UPL - uploading a file to the server (maximum file size to be determined later)
- DNL - downloading a file from the server (maximum file size to be determined later)
- RMF - removing a file from a folder on the server
- END - end the current session

This application is NOT capable of the following tasks

- The server cannot make connections with multiple users simultaneously. The previous connection is destroyed when a new connection is established with another client.
- Once a client sends a request message to the server, the client cannot send a new request message for a period of time (20sec) or until the client receives the corresponding response message. Thus, a client cannot download two files simultaneously.

B.Attacker Models

Assumptions on the attacker

1. The attacker is able to intercept all the messages sent by the client and the server.
2. The attacker is able to send any chosen message to the client and the server.
3. The attacker is able to replay old messages to the client and the server.
4. The attacker is able to intercept and modify any message before the message is received by the server or client.

C.Assumptions on Participants

Assumptions on the server

1. The server has a valid private-public key pair (long term)
2. The server securely stores the login password of each account ID (shared out-of-bound)
3. The server responds only when the MAC value is valid. Thus, the server does not tell whether or not the received request had a correct format. For example, the server does not respond whether the padding was correct or not.

Assumptions on the client and user

1. The client has a valid private-public key pair
2. The client has the public key of the server (long term, shared out-of-bound)
3. The user knows his/her account ID and login password (shared out-of-bound)
4. The user stores his/her login password securely

D.Security Requirements

The following security requirements must be satisfied.

1. The server must be authenticated when a client wants to establish a connection.
    - To prevent establishing a connection with an attacker.
2. The server has to send a response to every request message as long as it's valid.
    - To confirm that a request message reached the server.
3. A fresh session key must be generated and shared when a new session is established. Messages encrypted with an old session key should not be accepted by the server and the client.
    - To prevent replay attacks.
4. The client and server keep track of the sequence number asynchronously. Only the messages with a sequence number greater than the last accepted message should be accepted by the server and the client.
    - To prevent replay attacks.
5. The client cannot send a new request message while waiting for a response from the server. A request times out after 20 seconds.
6. Primitives for generating a nonce and an initialization vector must not be predictable by an attacker.

E.Protocol Specifications

E-1.General approach

Two protocols are used in this application: **Session Establishment Protocol (SEP)** and **File Management Protocol (FMP)**. SEP establishes a connection between a server and a client. A symmetric session key is shared through SEP. All the file management requests and responses are sent as FMP messages. Only the client sends requests to the server and only the server sends responses. FMP messages use the symmetric key shared through SEP to encrypt the contents. A MAC value is included in every message of FMP to confirm the integrity and the origin.

E-2.Protocol Details

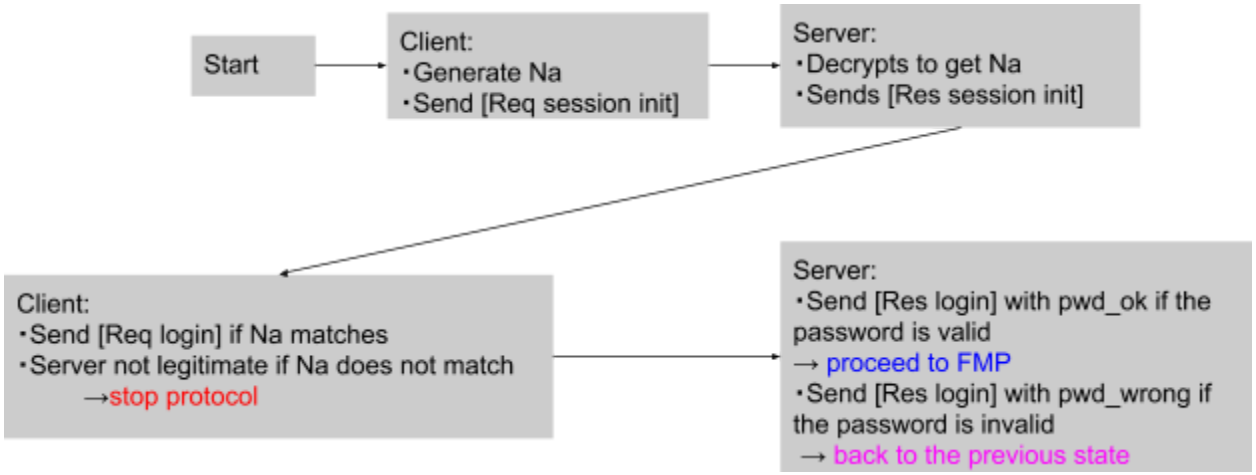
E-2a.Session Establishment Protocol (SEP)

- **Protocol Overview**

Message contents are encrypted by RSA with a 2048-bit public key. OAEP is used for padding. Then the ciphertext is decrypted with a 256 bit long private key by the recipient. Each message is signed with PKCS#1 v1.5 by the sender using the same private key used for RSA decryption. The goal of the first phase is to authenticate the server by sending a 256-bit nonce encrypted with the server's public key. The nonce is generated by the client A using Python's secret module. The second phase aims to login to the account by sending an account ID and the password.

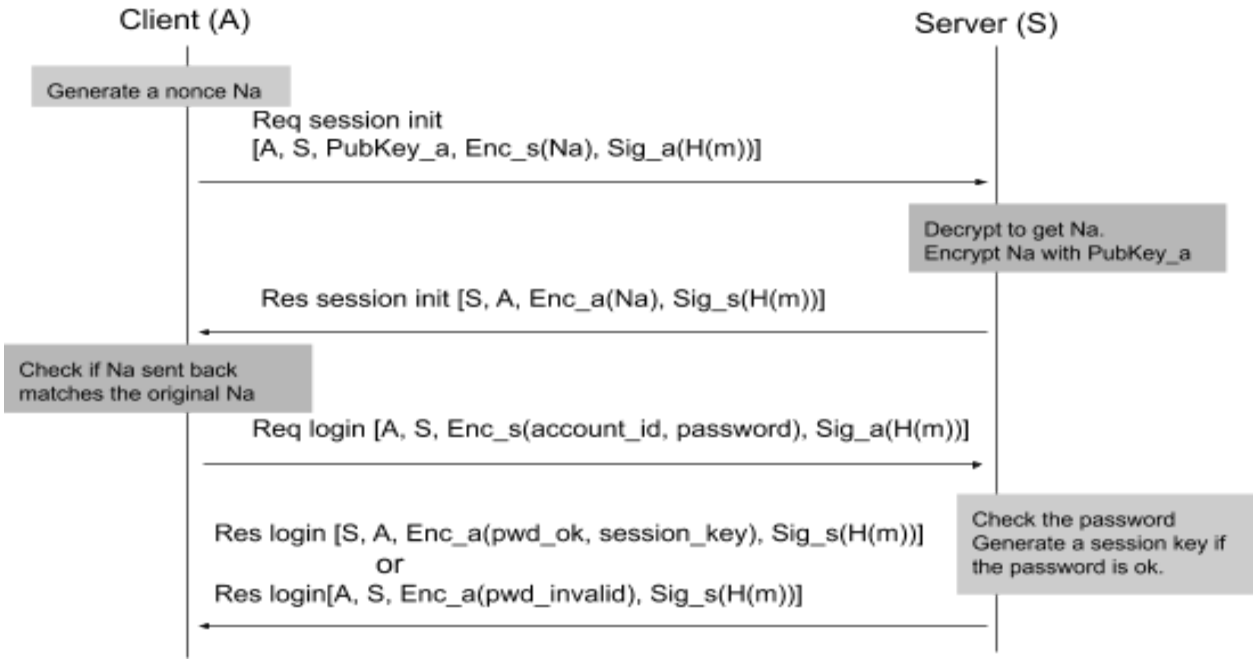
SEP ends when the client successfully logs in. If the login fails, the client may send a login request again without repeating the session init request.

- **Protocol State diagram**



***message rejected if signature is invalid**
***20 sec timeout at each state →back to start.**

- **Message Sequence Diagram**
  - A/S: client/server
  - Enc_a/Enc_s: RSA encryption with the 2048-bit public key of A/S
  - PubKey_a: 2048-bit public key of A
  - Na: 256-bit nonce generated by the client A using Python's secret module
  - Sig_a/s: PKCS#1 v1.5 signature using the 256-bit private key of A/S
  - H: SHA-256 hash function
  - m: the entire message excluding the signature



- **Assumptions**
  - Client has the long-term public key of the server.
  - The public key of the server is shared out-of-bound whenever it gets updated

▪ **Protocol Format**

Ver	Type (SEP)	Type (message)	Len	Sender ID	Recipient ID
Encrypted Content					
…….					
Signature					

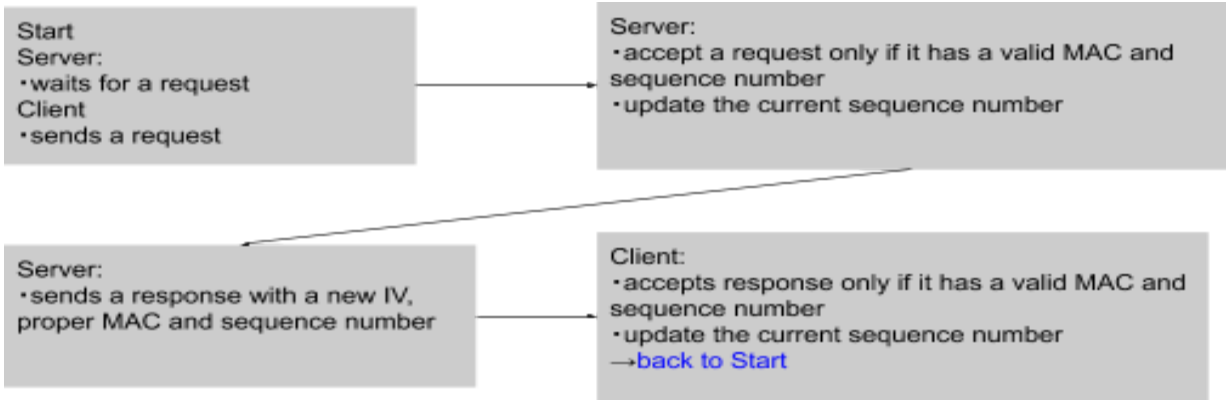
- Ver: protocol version (8 bits)
- Type(SEP): 0 which represents SEP (8 bit)
- Type(message): specifies Req/Res and Session init/Login (8 bits)
- Len: length of the encrypted content of the message (40 bits)
- Sender id: identifier of the sender (32 bits)
- Recipient id: identifier of the recipient (32 bits)
- Encrypted Content: encrypted content of the message (variable length)
- Signature: signature of the sender (256 bits)

E-2b. File Management Protocol (FMP)

▪ **Protocol Overview**

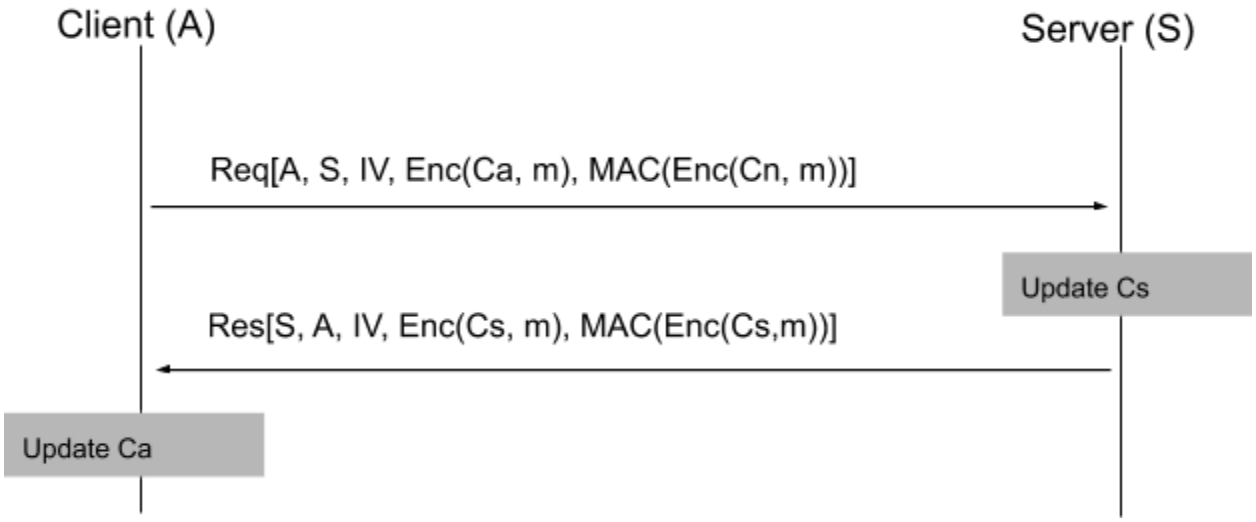
FMP uses AES-256 with CBC mode for encryption and decryption. The key is the 256-bit symmetric key shared through SEP. PKCS#7 is used for padding. HMAC with SHA256 hash function is used to generate a 256-bit MAC value. HMAC uses the symmetric 256-bit key shared through SEP. The initialization vector(IV) must be updated every time a new message is sent. IV is a 256-bit random number Python's secret module. Both the client and the server keeps track of message sequence numbers asynchronously. Messages with a sequence number older or equal to the last accepted message's sequence number must be rejected.

▪ **Protocol State Diagram**



***20 sec timeout at each state → back to start**

- **Message Sequence Diagram**
  - A/S: client/server
  - IV: 256-bit initialization vector for AES-256 CBC
  - Enc AES-256 CBC mode encryption with the symmetric key
  - MAC: HMAC with SHA256 using the symmetric key
  - Ca/Cs current message sequence number
  - m content of the message



- **Assumptions**
  - A symmetric key has been shared between the client and serve through SEP
  - IV is updated every time a new message is generated and sent along with the encrypted message.
  - The next IV is unpredictable by the attacker.

- **Protocol Format**

Ver	Type (FMP)	Type (command)	Len	Sender ID	Recipient ID
Encrypted Content					

…….

MAC

  - Ver: protocol version (8 bits)

- Type(SEP): 1 which represents FMP (8 bit)
- Type(command): specifies Res/Req and the type of operation (8 bits)
  Ex. Req-MKD, Res-UPL
- Len: length of the encrypted content of the message (40 bits)
- Sender id: identifier of the sender (32 bits)
- Recipient id: identifier of the recipient (32 bits)
- Encrypted Content: encrypted content of the message (variable length)
- MAC: HMAC value (256 bits)

F.Security benefits of this protocol

F1. SEP's security benefits

The primary goal of SEP is to authenticate the server. In order to do so, a nonce is encrypted with the public key of the server and sent. Showing that the server has the corresponding private key is sufficient to prove the authenticity, since only the legitimate server has the private key. The secondary goal is to tell the server the login password. This task is trivial after the server is proven to be legitimate. The client can send the password by encrypting it with the public key of the server.

Signature is attached to every message of SEP in order to prevent modification of messages by an attacker. If there were no integrity check, the attacker might be able to modify the public key of A in the first message so that the server will encrypt the nonce with the attacker's public key.

F2. FMP's security benefits

FMP uses a symmetric session key provided by the server through SEP. The content is encrypted by the AES-CBC cipher. In order to make sure that the content was not modified by the attacker, every message has MAC value appended to the message. The input to the MAC value is the encrypted content. Thus, the recipient can verify the integrity before decryption. This reduces the risk of padding oracle attack. Since only the server and the client have the symmetric key, having a valid MAC value confirms that the message is sent from a valid client and is not modified by an attacker.

Session key is included in the message content in order to prevent replay attacks. Any message with a sequence number older or equal to the current sequence number is rejected, so that an attacker won't be able to eavesdrop and replay a message to the server. The input to the MAC value has to include the sequence number so that the attacker won't be able to modify it.