INF-2310
Assignment2, Cryptographic file sharing

Magnus Dahl-Hansen

March 2023

Introduction:

The task for this assignment was to transmit an encrypted file from a server to a client. This had to be solved by using both symmetric and asymmetric encryption techniques to be able to (among other things) get access and agree upon a “session key”, encrypt a file and decrypt a file.

The purpose of this assignment was to gain an insight into how encryption and decryption works and how files are transmitted over a network to prevent three types of attacks: *man-in-the-middle attacks, eavesdropping* and *replay attacks*

Background theory

Before reading this report the reader should be familiar with: symmetric and asymmetric encryption, how safe file transferring takes place over a network, man-in-the-midde attacks, eavesdropping and replay attacks.

Symmetric encryption is a method where a file can be encrypted and decrypted with a single key. The biggest flaw of this method is how to securely share the single key between two parties without letting any unwanted users to get access to the key.

Asymmetric encryption is a method where two communicating parties each has a *private key* and a *public key* where the public key is exchanged between both parties. A message can be encrypted with a users public key but can only be decrypted by the users private key. This way a server can encrypt a message with the clients public key and send the encrypted file which can only be decrypted by the clients private key.

Safe file transferring over a network is done by first establishing a connection between a server and a client. When the connection has been established both sides exchange their public keys so that they can use asymmetric encryption to send and agree upon a private *session key*. When the *session key* has been established a file can be encrypted, sent and decrypted using this *session key*.

Eavesdropping attack occurs when a hacker takes advantage of an insecure network connection and places himself in between a server and a client to eavesdrop and get access to the data being transmitted between the two devices.

Man-in-the-middle attacks is similar to *eavesdropping attacks*. The attacker places himself in the middle of a connection between two parties so that he can listen and modify data being transmitted.

Replay attacks occurs when the attacker intercept an ongoing communication and delays or re-sends data to misdirect the receiver into doing what the hacker wants. An example is to provide the receiver with a fake website where he enters private authorization information without being aware of it.

Design & implementation

This program consists of two sides: a server and a client. The server starts by creating a socket and listens for incoming connection requests by using the python *socket* library. The client side also creates a socket which it uses to send a connection request to the server socket. Once the server receives this request it responds with an acknowledgment message and establishes a continuous connection with the client. To include both asymmetric and symmetric encryption techniques both sides will use asymmetric encryption to agree upon a session key which can then incorporate symmetric encryption to send the file.

When the connection is established the client will have used the *"Pycryptodome"* library to produced a public key that is sent to the server. The server then uses this public key to encrypt a session key with the Rivest-Shamir-Adleman (RSA) algorithm. The server proceeds to encrypt a local file using the session key and the *Advanced Encryption Standard* (AES) algorithm which is available from the *Pycryptodome* library. The AES algorithm converts the file into ciphertext and produces a *nonce* and a *tag* variable. The *nonce* is an arbitrary number used to fight against replay attacks, and the *tag* is used to authenticate the file to check if it has been tampered with. After the file has been encrypted it is sent along with the *nonce* and *tag* variable to the client.

The client receives the encrypted file, *nonce* and *tag* variable from the server and uses these to decrypt the file using the *Pycryptodome* library functions. When the file is decrypted and authenticated a new file is opened, the decrypted information is then written to it and the connection between the server and the client is closed.

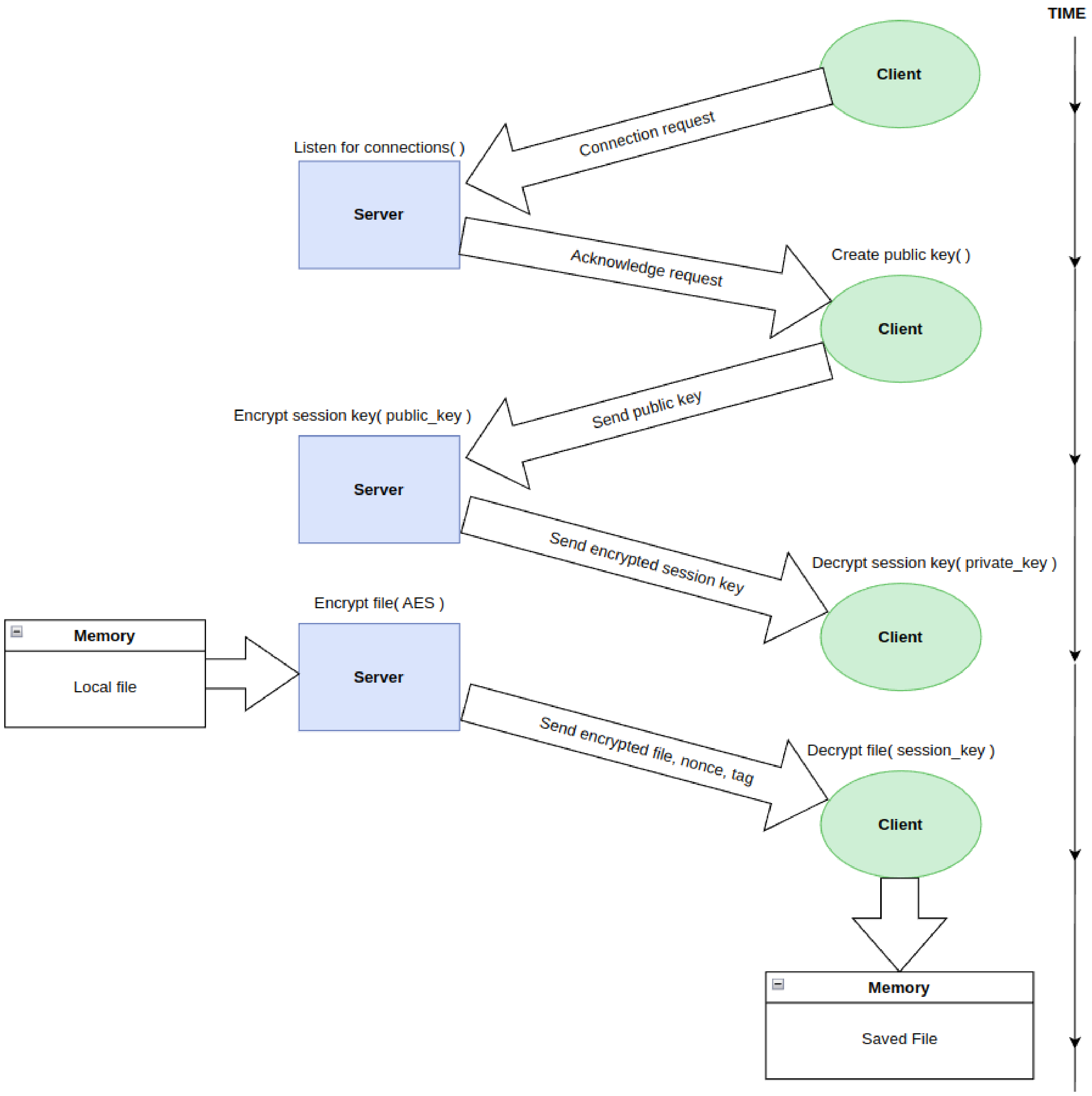
TIME
Client
Connection request
Listen for connections( )
Server
Acknowledge request
Create public key( )
Client
Send public key
Encrypt session key(public_key)
Server
Send encrypted session key
Decrypt session key(private_key)
Client
Encrypt file(AES)
Memory
Local file
Server
Send encrypted file, nonce, tag
Decrypt file(session_key)
Client
Memory
Saved File

Figure 1. ***Program organization***

Evaluation/discussion

The program have been implemented using both symmetric and asymmetric encryption techniques and is able to encrypt, send and decrypt session keys and files. Proof that the program works and that encryption is being used can be seen by inspecting the code and the printed cipher text in the terminal during run time. There will also be created a new file in the *src* directory with the decrypted content after running the program, this file will by default be called “*clientFile.txt*”.

The usage of the asymmetric RSA and symmetric EAS algorithms from the Pycryptodome library can provide proof that the implemented program can protect against *Replay attack*, *Eavesdropping* and *Man-in-the-middle attacks*. The most common ways to protect against these kind of attack is by using encrypted messages and authentication, which this implementation heavily relies upon. All messages sent between the server and the client are encrypted and the authentication is equivalent to the server and the client exchanging public keys and agreeing upon a session key. The authentication part could of course be made more realistic by implementing a login function, but I deemed this unnecessary for this assignment. Other ways of preventing such attacks, which has not been implemented, are to use a secure connection such as HTTPS, using Virtual Private Networking (VPN) and adding a timestamp to each message so that an eventual attacker has less time to modify and/or re-send messages.

Conclusion

The implemented code is able to establish a stable connection between a server and a client, exchange public keys, establish a session key using asymmetric encryption and encrypt, send and decrypt a file using symmetric encryption. The usage of these functionalities allows the server and the client to exchange files in a secure way and solve this assignment by preventing from three types of attacks: *Eavesdropping*, *Man-in-the-middle attacks* and *Replay attacks.*

References

1. Tutorial for using pycryptodome and encryption/decryption with AES: https://nitratine.net/blog/post/python-encryption-and-decryption-with-pycryptodome/

2. Tutorial for using pycryptodome and encryption/decryption with RSA: https://pycryptodome.readthedocs.io/en/latest/src/examples.html