Lab3 (Report) – Problem-2

VATSAL AGRAWAL

Master Of Technology
COMPUTER SCIENCE and ENGINEERING
2nd Sem

Entry Number - 2021MCS2157

NETWORK & SYSTEM SECURITY

SIL 765

Introduction

My programme is working correctly. The screenshot of working is below. I have read a python doc about OpenSSL and cryptography modules to help solve the program. I have performed all tasks, and it's all part. I have used some info same from my previous submitted assignment of this course.

I have implemented the TLS protocol from scratch using the cryptography module only. I have used the os module for creating random numbers and the sys library for taking CLI input. I have also provided analysis in the end and for each part.

Time for running the whole program is negligible. I have implemented protocol negotiation, exchange of keys master, and session both in an authenticated way. Also, we have authenticated the sender and generated the ca, server, and client certificate, which is used for authentication. I have done handshakes and also used record protocol to send a message. We have done permutation on all given cipher suites as much as possible. A detailed explanation is given below. All communication is done through the socket. And code is not hardcoded.

INDEX

Problem 2

Trusted Third Party

The main tasks of TTP in our program are to provide a ca certificate to client and server and accept Certificate sign request certificate (CSR certificate) from client and server and provide them their respective certificate, which can be used by peers for authentication.

We have given TTP port as 1036 and hostname as localhost (127.0.0.1) as default. If we pass one argument, then it should be the port number of TTP, and if two arguments are passed, then it should be the port number followed by hostname. If 0 arguments passed, then the default will run.

We have generated an asymmetric key using the ECDSA SECP256R1 curve (256-bit key) and fixed the common name as “Agrawal.” Our certificate contains the only common name in issuer name and subject name. Serial number, validity range, subject public key, and DNS name in extension as localhost are also included. We have used SHA 256 for hash.

We keep on constantly listening to the port number of our TTP and do not die. We first receive a CSR certificate from a peer, then we load it and give a validity range of 20 days to certificate and create a serial number randomly. Then we sign it using the private key of ECDSA algo and hash SHA256. After this, we first send the peer certificate and wait for ACK from the sender; then, we send the CA certificate, which can be used to verify the certificate.

As the certificate is encoded using the private key, it can only be seen and decoded by anyone but cannot be generated by anyone else. Also, they contain signs which can be used to find their integrity. Hostname check helps in detecting impersonation. Also, any message encoded by a public key cannot be decoded by anyone else due to a lack of a private key. So encrypting the certificate is not a good idea, and thus we have not encrypted our certificate.

In the end, we print “All certificates” sent to indicate sending of all certificates for a CSR. If we do not get ACK from a peer, we can close our connection for that CSR, and the peer has to resend CSR.

Server

The main tasks of the Server in our program are to generate RSA keys and Certificate sign request certificate (CSR certificate) and send CSR to TTP and get a respective certificate and CA certificate. Followed by listening to the client and performing a TLS handshake with protocol negotiation, authenticated key exchange, and transcript confirmation; finally, it should exchange messages through record protocol.

We have given the Server port as 1069 and hostname as localhost (127.0.0.1) by default. If we pass one argument, then it should be the port number of TTP, and if two arguments are passed, then it should be the port number of TTP followed by the port number of the Server. For three arguments, the third argument should be hostname now. If 0 arguments passed, then the default will run.

We have generated an asymmetric key using RSA (2048 bit key) and fixed the common name as "server." We have used SHA 384 for hash.

We have generated CSR and sent it to TTP, which includes the common name as server name and DNS name as localhost. Then we first receive the server certificate and send ACK and then receive CA certificate and print "certificate exchange is done."

We have defined acceptable cipher suite as TLS1.3 – RSA –Chacha 256—AES 128 GCM --Sha 256 and TLS1.3 – RSA –Chacha 256—AES 128 OCB3 --Sha 256.
Here entry is separated by – and first entry corresponds to TLS version supported, the second entry is asymmetric key algo used, 3rd entry is Authenticated Master key generation algo, 4th entry is authenticated session key and message encryption algo, 5th entry is Hash algorithm supported.

The client sends a hello packet with any one of the cipher suites, and the server checks the cipher suite in its database and sends its certificate if the cipher suite is accepted; otherwise, it sends a termination request if no cipher suite is accepted by the server. Then we receive the client certificate and send ACK if verified. The server then waits for the encrypted master key (by server public key) and decrypts it using its private key to get the master key. We then used this master key to encrypt a session key. We have used CHACHA20Poly1305 for master key generation and authentication and

encryption of session keys. Session key generated through AES GCM. Then we send ACK and wait for nonce followed by the session key. Then we verify and decrypt the session key and send ACK. Finally, in the last step of handshake, the client sends the cipher suite we agreed upon along with the session key encrypted by the session key, and we accept and send ACK or send terminate requests. We have used AES GCM for encryption and Authentication with SHA 256. We then print “Handshake established and Start message exchange.”

In the record protocol for each and every message sent, we encrypt and sign it using AES GCM. Then we print the send message and print “Closing of the socket.” Then we start new listening after sending all messages. Our server does not die

Client

The main tasks of the Client in our program are to generate RSA keys and Certificate sign request certificate (CSR certificate) and send CSR to TTP and get a respective certificate and CA certificate. Followed by connecting to the server and performing a TLS handshake with protocol negotiation, authenticated key exchange, and transcript confirmation; finally, it should exchange messages through record protocol.

We have given Client port as 1056 and hostname as localhost (127.0.0.1) as default. If we pass two arguments, then it should be the port number of TTP followed by the port number of the Server. For the three arguments, the third argument should be the port of the client. For four arguments, the fourth argument should be hostname now. If 0 arguments passed, then the default will run.

We have generated an asymmetric key using RSA (2048 bit key) and fixed the common name as “Vatsal.” We have used SHA 256 for hash.

We have generated CSR and sent it to TTP, which includes the common name as client name and DNS name as localhost. Then we first receive the server certificate and send ACK and then receive the CA certificate and print “certificate exchange is done.”

We have defined an acceptable cipher suite as TLS1.3 – RSA –Chacha 256—AES 128 GCM --Sha 256 only. Here entry is separated by – and first entry corresponds to TLS version supported, the second entry is asymmetric key

algo used, 3rd entry is Authenticated Master key generation algo, 4th entry is authenticated session key and message encryption algo, 5th entry is Hash algorithm supported.

The client sends a hello packet with the above cipher suite and receives a server certificate if the cipher suite is accepted; if the termination request is received, then close connection. Then we send the client certificate and receive ACK if verified. The client then sends an encrypted master key (by server public key) which is generated with the help of CHACHA20Poly1305 algo. After receiving the ACK, We then used this master key to encrypt and sign the session key. We have used AES GCM for the generation of the session key. Then we send this session key and nonce. After getting ACK, Finally, in the last step of handshake, the client sends cipher suite we agreed upon along with session key encrypted by session key, and we accept and send ACK or send terminate requests. We have used AES GCM for encryption and Authentication with SHA 256. We then print "Handshake established and Start message exchange."

In the record protocol for each and every message received, we decrypt and verify it using AES GCM. Then we print the received message and print the Closing of the socket.

Analysis

1. We have to first set up a TTP server because all peers will be taking certificates from it. Then we set up the Server because the client needs to connect to a running server
2. The client has used SHA 256 and RSA2048. The server has used SHA 384 and RSA2048, and TTP has used RSA256 and ECDSA. Session Key generated through AES GCM and Master Key through CHACHA20Poly1305. We have used CHACHA 256 for the master key because it provided more security and was used less, thus eliminating vulnerabilities of stream cipher. As the session key is used constantly for sending the message, we have used AES GCM for it.
3. We have done both sides of peer authentication because OTP is sent from server to client. Also, every message is authenticated and signed.

4. Certificates are signed but not encrypted because they contain only a public key which is to be public and duplicate certificates cannot be generated without the private key.
5. We have not incorporated proper TCP protocol; thus, the timeout can result in loss of connection, and the connection needs to be established again.
6. We have tried to minimize the number of packet flow as happens in the real world. The ACK packet can be increased, which can give more durability and fault tolerance to our code. But this will increase time also.
7. AES GCM uses CTR mode for its encryption and uses Galois field multiplication polynomial MAC. The same nonce leads to security vulnerabilities. So we have to send a new nonce for each message sent. It does both encryption and authentication in parallel, and thus it is fast. It is also vulnerable to cycling attacks and cache timing attacks.
8. CHACHA20Poly1305 is a stream cipher with poly 1305 for MAC. It can be implemented by software easily without any hardware and used in mobile, etc. The same nonce leads to vulnerabilities.
9. If we have hardware for AES, our setup of session key using AES GCM is better; if we do not have the hardware, we can swap Master and session key algo because chacha is faster in software.
10. The key generated by peers can be upgraded to ECDSA instead of using RSA, which is of large size and takes time for encryption and decryption of the master key.
11. We have not incorporated the message to be sent in CLI arguments because it is given in the problem statement what needs to be sent.
12. We have printed Completion Message after every step as well as message in both client and server.
13. We can see that TTP run two times, indicating it sends the certificate to 2peers client and server
14. Also, we see that client and server perform all steps of TLS
15. Also, the send and receive the message are also the same
16. The total real-time is 0.324s with CPU time only 0.229s, as seen below

Screenshot of Result

Fig1. Execution Of TTP (left-up), Client (right), Server (left-bottom)

Pre-Requisite

How To Make Executable and Run Programme

1) **First Run my_TTP** (If we pass one argument, then it should be the port number of TTP, and if two arguments are passed, then it should be a port number followed by hostname. If 0 arguments passed, then the default will run.)
2) **Then run my_server** (If we pass one argument, then it should be port number of TTP, and if two arguments are passed, then it should be port number of TTP followed by port number of Server. For three arguments, the third argument should be hostname now. If 0 arguments passed, then the default will run.)
3) **Then run my_client** (If we pass two arguments, then it should be port number of TTP followed by port number of Server. For the three arguments, the third argument should be the port of the client. For four arguments, the fourth argument should be hostname now. If 0 arguments passed, then the default will run.)

Structure Of Programme

Program is made up of directory structure which includes

1) Problem-1 – code for problem1
2) Problem 2 – code for problem2
3) Makefile – Makefile command
4) Problem-1/my_ttp - TTP code
5) Problem-1/my_client – client code
6) Problem-1/my_server – server code
7) Env.txt – Containing Environment info on which program was tested.

Future Scope

Better Authenticated encryption algo can be used.
The use of ACK can be reduced.
Duplex communication can be made.
Better curves can be used instead of RSA
Certificate sharing and content can be improved.

References

I have completed my assignment by learning from the following sources

1. https://soatok.blog/2020/07/12/comparison-of-symmetric-encryption-methods/#aes-gcm-vs-chacha20poly1305
2. https://cryptography.io/en/latest/
3. https://www.openssl.org/
4. https://docs.python.org/3/library/ssl.html