

**Project Title:**

Secure File Transfer using AES- 128 Encryption and Diffie-Hellman Key Exchange

**Name:** Jinto Jose

**Date:** 04/24/2017

**Abstract**

The Objective of the project is as follows:

1. Establish SSL Socket Connection between Server and Client using SSL certificates.
2. Establish Shared Key between Client and Server using Diffie-Hellman Key Exchange Protocol
3. Encrypt the File to be sent from the Server with AES-128 Encryption using the Shared key established with the Diffie-Hellman Key Exchange.
4. The Encrypted file is sent from the server to Client across the SSL connection
5. Once the Encrypted File received at the Client side, the Client decrypts the encrypted file using AES-128 Decryption using the Shared key established with the Diffie-Hellman Key Exchange.

**1. Introduction:**

The report is divided into the following parts:

- Network Security Building Blocks used in the project.
- Working of the Project.
- Implementation with Code Snippets.
- Achieved Goals
- Individual Contribution.
- Screen Shots

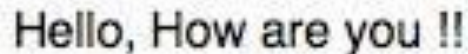
The project is aimed at secure file transfer. The projects uses the following Network Security Building Blocks to achieve that.

- SSL Socket based connection using SSL Client/Server certificates
- AES-128 Encryption and Decryption.
- Diffie-Hellman Key Exchange.

## 2. Network Security Building Blocks:

### 2.1. Input Plain Text

The Plain-Text can be any text of any Format or it can be any file containing any Letter Format or Numbers. It has to be a file which the program can pick up and transfer it to the receiver.

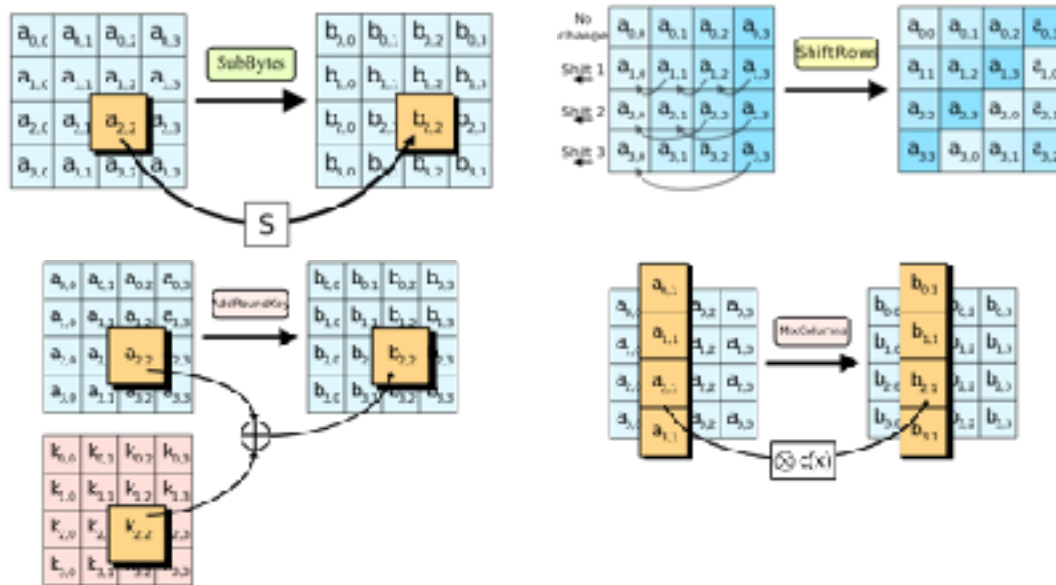


Hello, How are you !!

### 2.2. AES - 128 Encryption

The AES - 128 Algorithm can be explained in a High Level as Follows:

- KeyExpansions—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.
- AddRoundKey—each byte of the state is combined with a block of the round key using bitwise xor.
- SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
- ShiftRows—a transposition step where the last three
- MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
- AddRoundKey
- The Final Round will have the step of SubBytes , ShiftRows , AddRoundKey excluding Mixing Columns.



## 2.3. SSL File Transfer

- SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral. The Connection is established after the SSL Handshake between the Client and the Server.

## 2.4. Diffie - Hellmann Key Exchange

- Diffie-Hellman is a way of generating a shared secret between two people in such a way that the secret can't be seen by observing the communication. That's an important distinction: You're not sharing information during the key exchange, you're creating a key together.
- This is particularly useful because you can use this technique to create an encryption key with someone, and then start encrypting your traffic with that key. And even if the traffic is recorded and later analyzed, there's absolutely no way to figure out what the key was, even though the exchanges that created it may have been visible.
- This is where perfect forward secrecy comes from. Nobody analyzing the traffic at a later date can break in because the key was never saved, never transmitted, and never made visible anywhere.

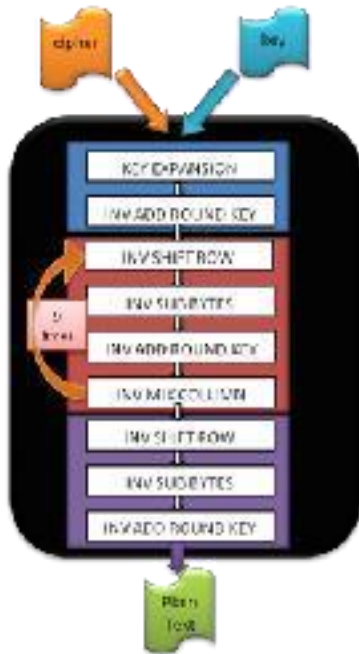
$$\begin{aligned} (g^a \bmod p)^b \bmod p &= g^{ab} \bmod p \\ (g^b \bmod p)^a \bmod p &= g^{ba} \bmod p \end{aligned}$$

## 2.5. AES - 128 Decryption

The AES decryption basically traverses the encryption algorithm in the opposite direction. The basic modules constituting AES Decryption are explained in detail below:

a) Key Expansion - The algorithm for generating the 10 rounds of the round key is as follows: The 4th column of the  $i-1$  key is rotated such that each element is moved up one row.

It then puts this result through a forwards Sub Box algorithm which replaces each 8 bits of the matrix with a corresponding 8-bit value from S-Box. To generate the first column of the  $i^{\text{th}}$  key, this result is XOR-ed with the first column of the  $i-1^{\text{th}}$  key as well as a constant (Row constant) which is dependent on  $i$ .



The second column is generated by XOR-ing the 1st column of the  $i^{\text{th}}$  key with the second column of the  $i-1^{\text{th}}$  key.

This continues iteratively for the other two columns in order to generate the entire  $i^{\text{th}}$  key.

Additionally this entire process continues iteratively for generating all 10 keys. As a final note, all of these keys are stored statically once they have been computed initially as the  $i^{\text{th}}$  key generated is required for the  $(10-i)^{\text{th}}$  round of decryption.

b) Inverse Add Round Key – Performs XOR operation between the cipher text and intermediate expanded key corresponding to that particular iteration. E.g., if the diagrams on the left represent the cipher and the key

values, the final value after it has generated by this step is shown on the right.

c) Inverse Shift Row – This step rotates each  $i^{\text{th}}$  row by  $i$  elements right wise, as shown in the figure.

d) Inverse Sub Bytes – This step replaces each entry in the matrix from the corresponding entry in the inverse S-Box[2] as shown in figure.

e) Inverse Mix Column - The Inverse MixColumns operation performed by the Rijndael cipher, along with the shift-rows step, is the primary source of all the 10 rounds of diffusion in Rijndael.

To sum it all up, the AES decryption initially performs key-expansion on the 128-bit key block. Then the round key signals the start of the actual decryption process once the data process is ready. It starts by executing an inverse add round key between cipher text with the key (generated in the last iteration of the encryption process) from key expansion. After this step, the AES decryption repeats the inverse shift row, inverse sub, inverse add round key, and inverse mix column steps nine times. At the last iteration, it does an inverse shift row, inverse sub bytes and inverse add round key to generate the original data.

### **3. Working of the Project.**

#### **SSL Connection:**

- SSL Certificates are generated and stored in Keystore files at both Server and Client side. These certificates are used in the SSL Handshake to establish the connection between the Client and Server.

#### **Diffie-Hellman Key Exchange:**

- Diffie-Hellman Key Exchange is used to establish shared key between client and server. The Diffie Helman parameter are generated at the Server side and send to the client.

#### **AES-128 Encryption:**

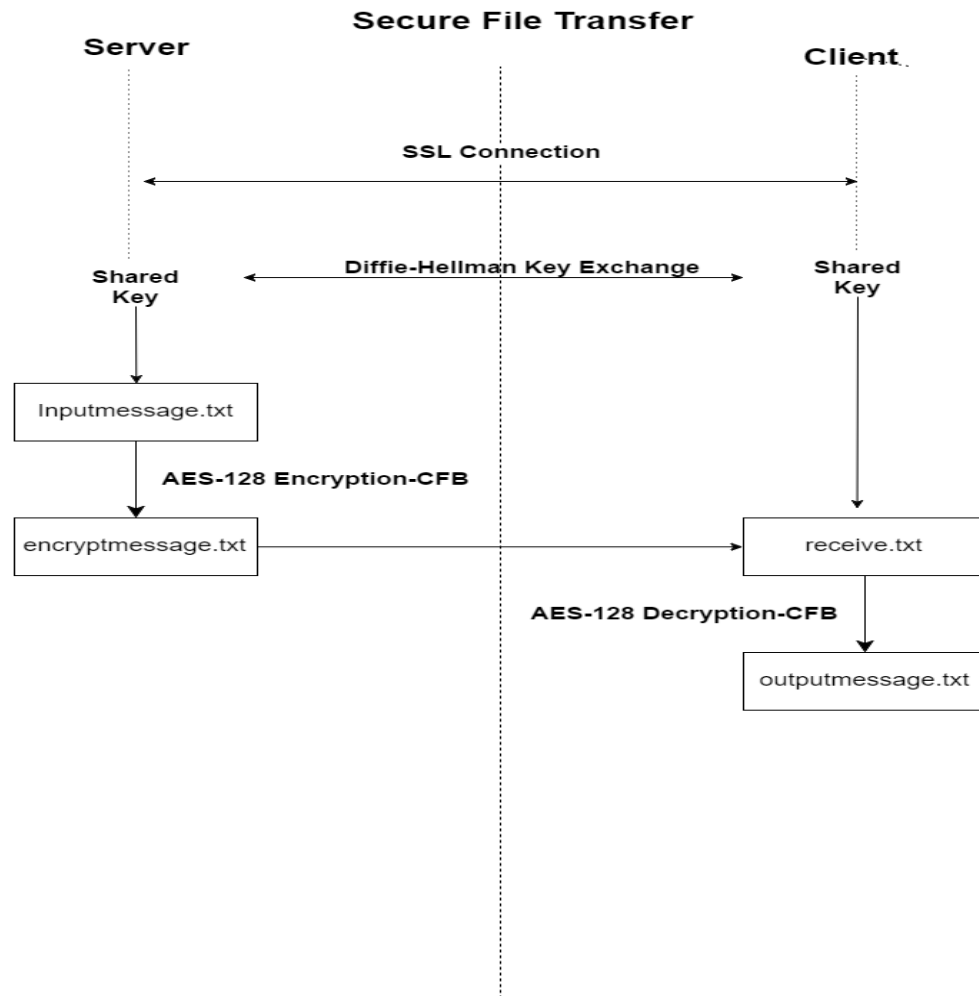
- AES-128 Encryption is used to encrypt the inputMessage.txt to encrypt.txt at the server side using the Shared key got from the Diffie-Hellman Key Exchange. The encrypted file is sent to the Client.

#### **Secure File Transfer:**

- The encrypted file encrypt.txt is sent from server to the client using secure SSL Socket connection present between the client and the server. The Client receives the encrypted content and stores it in receive.txt.

#### **AES-128 Decryption:**

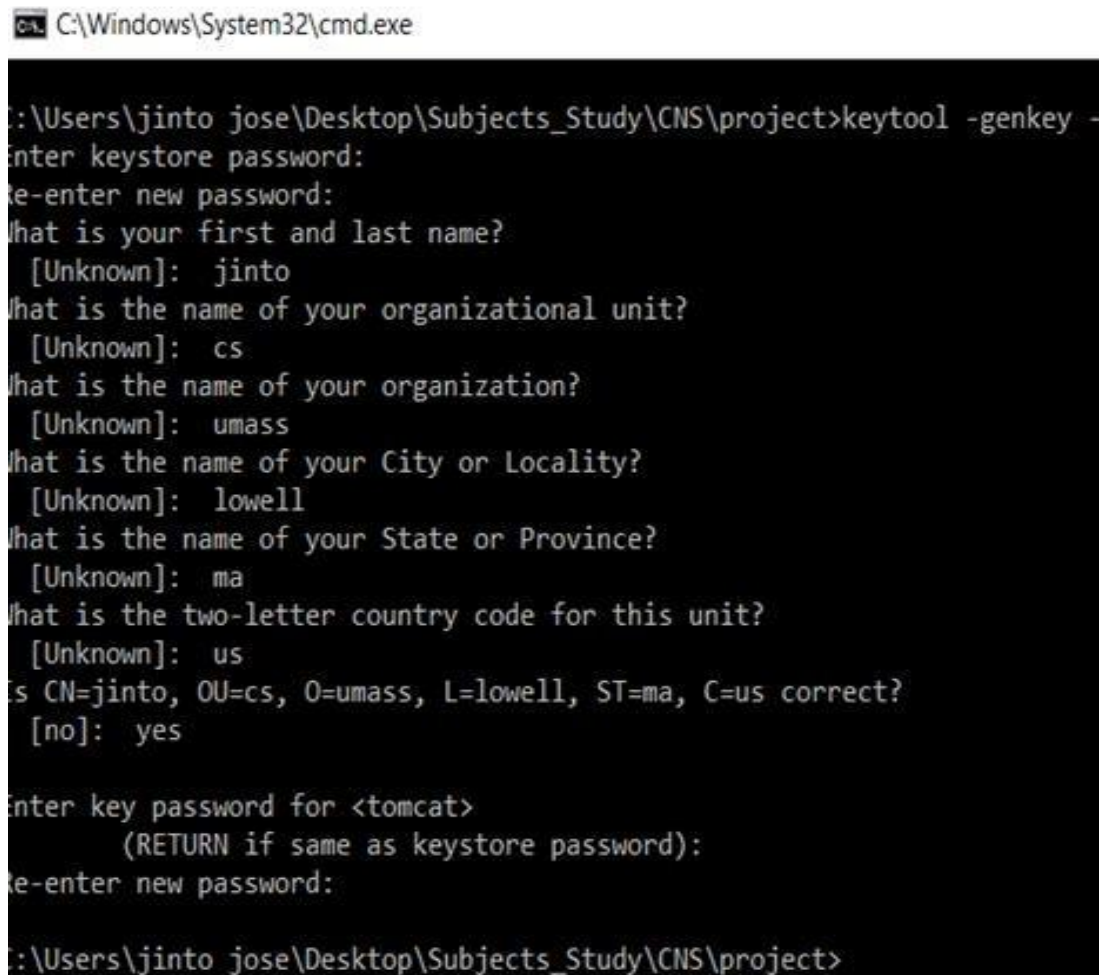
- AES-128 Decryption is used to decrypt the encrypted file receive.txt received at the client to the original file. After Decryption outputmessage.txt is created with contains the original content sent by the server.

**Overview of Server-Client Communication:**

## 4. Implementation:

### Step 1. Secure Socket Layer Connection:

1. Generating a self-signed SSL certificate using the Java keytool command
2. Command:
3. "keytool -genkey -keyalg RSA -alias test -keystore keystoreFile.jks -validity 365 -keysize 2048"
4. Both Client and Server require the self-signed SSL certificate to establish the SSL connection.



```
C:\Windows\System32\cmd.exe

C:\Users\jinto\jose\Desktop\Subjects_Study\CNS\project>keytool -genkey -
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: jinto
What is the name of your organizational unit?
[Unknown]: cs
What is the name of your organization?
[Unknown]: umass
What is the name of your City or Locality?
[Unknown]: lowell
What is the name of your State or Province?
[Unknown]: ma
What is the two-letter country code for this unit?
[Unknown]: us
Is CN=jinto, OU=cs, O=umass, L=lowell, ST=ma, C=us correct?
[no]: yes

Enter key password for <tomcat>
(RETURN if same as keystore password):
Re-enter new password:

C:\Users\jinto\jose\Desktop\Subjects_Study\CNS\project>
```

## SSL Code :

```
try {
    int serverPort = 7777; //server is listening for connection request on port via TCP.
    //ServerSocket s = new ServerSocket(serverPort);
    System.setProperty("javax.net.ssl.keyStore", "keystoreFile.jks");
    System.setProperty("javax.net.ssl.keyStorePassword", "password");
    KeyStore ks = KeyStore.getInstance("JKS");
    ks.load(new FileInputStream("keystoreFile.jks"), "password".toCharArray());

    KeyManagerFactory kmf = KeyManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
    kmf.init(ks, "password".toCharArray());

    TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
    tmf.init(ks);

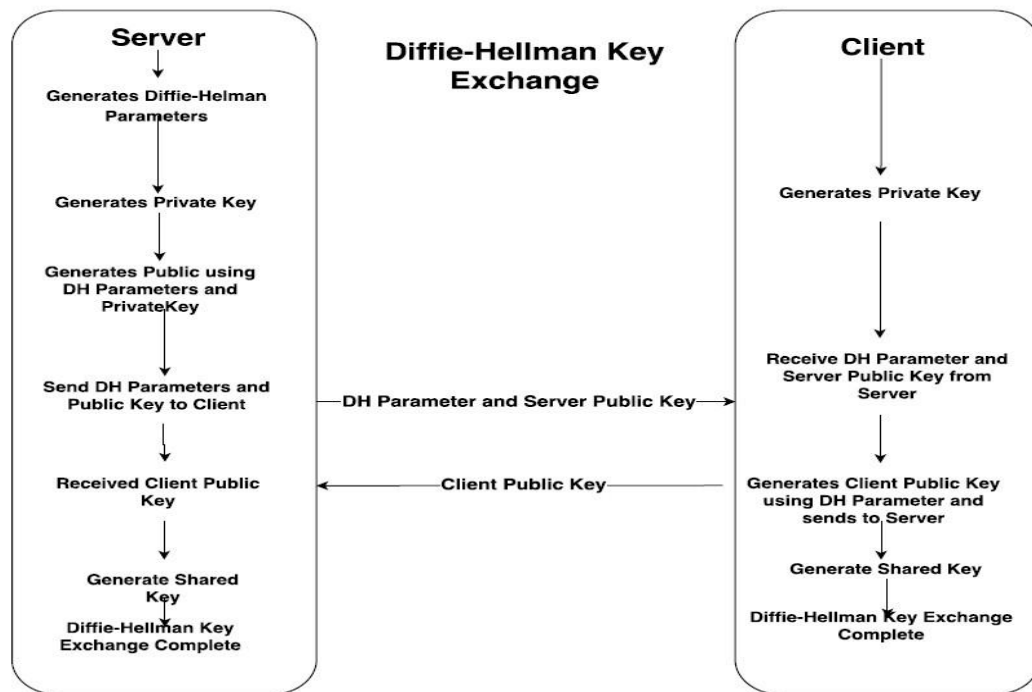
    SSLContext sc = SSLContext.getInstance("TLS");
    TrustManager[] trustManagers = tmf.getTrustManagers();
    sc.init(kmf.getKeyManagers(), trustManagers, null);

    SSLServerSocketFactory ssf = sc.getServerSocketFactory();
    SSLServerSocket s = (SSLServerSocket) ssf.createServerSocket(serverPort);
    s.setSoTimeout(50000); //set Time out
    System.out.println("Server : Server Started on Port Number 7777\n");
    System.out.println("Server : Waiting for client\n");
    System.out.println("-----Waiting for client-----");
    server = (SSLSocket) s.accept();
    //key exchange
    System.out.println("Server : Connection Recieved from Client\n");
}
```

## Step 2. Diffie-Hellman Key Exchange:

1. Server generates Diffie-Hellman Parameters
2. Server generates its Private key.
3. Server using DH Parameters and its Private Key generates its Public Key.
4. Server sends its Public key and DH parameter to the Client.
5. Client generates its Private Key.
6. Client using its Private Key and DH Parameters generated its Public Key and sends to the Server.
7. Client using the DH Parameters and Server Public Key creates the Shared Key.
8. Server using the Client Public Key and DH Parameters generates the Shared Key. Hence completing the Diffie-Hellman Key Exchange.





## Diffie-Hellman Key Exchange Code:

```
DHParameterSpec dhSkipParamSpec;
```

```
if (mode.equals("GENERATE_DH_PARAMS")) {
    // Some central authority creates new DH parameters
    System.out.println
        ("Server : Creating Diffie-Hellman parameters...\n");
    AlgorithmParameterGenerator paramGen
        = AlgorithmParameterGenerator.getInstance("DH");
    paramGen.init(512);
    AlgorithmParameters params = paramGen.generateParameters();
    dhSkipParamSpec = params.getParameterSpec
        (DHParameterSpec.class);
} else {
```

```
    // use some pre-generated, default DH parameters
    System.out.println("Using SKIP Diffie-Hellman parameters");
    dhSkipParamSpec = new DHParameterSpec(skip1024Modulus,
        skip1024Base);
}
```

```
System.out.println("Server : Generate DH keypair ...\n");
KeyPairGenerator aliceKpairGen = KeyPairGenerator.getInstance("DH");
aliceKpairGen.initialize(dhSkipParamSpec);
KeyPair aliceKpair = aliceKpairGen.generateKeyPair();
```

```
// Alice creates and initializes her DH KeyAgreement object
System.out.println("Server: Initialization of DH KeyAgreement object...\n");
KeyAgreement aliceKeyAgree = KeyAgreement.getInstance("DH");
aliceKeyAgree.init(aliceKpair.getPrivate());
```

```
// Alice encodes her public key, and sends it over to Bob.
byte[] alicePubKeyEnc = aliceKpair.getPublic().getEncoded();
```

### **Step 3. AES Encryption Using 128 bit Key**

1. The file to be send by the server to client: "inputmessage.txt"
2. The file is encrypted using AES Encryption using 128 bit key.
3. The 128-bit shared key is generated by the Diffie-Hellman Key Exchange.
4. Cipher Feedback Mode –CFB is used in the AES Encryption.
5. The encrypted file is send to the Client through the SSL connection.
6. After receiving the encrypted file, the Client decrypts the encrypted file sent by the Server using the shared key and gets the Original File

## **5.Achieved Goals**

1. Successfully established SSL Socket connection between Client and Server using SSL Certificates.
2. Successfully performed Diffie-Hellman Key Exchange between Client and Server to establish Shared key.
3. Successfully performed AES-128-bit encryption of the file to be sent from the Server.
4. Successfully sent the encrypted file using SSL connection from Server to Client.
5. Successfully performed AES-128-bit decryption of the file sent by the server at client side to retrieve the original file. Hence securely transferring the file from Server to the client.

## **6.Future Goals**

1. Improve the program to securely send files types other text(.txt). Example:PDF,ZIP,RAR
2. Improve the program to send files outside the LAN Network.

## **7.System Framework and Programming Platforms**

Java Security Libraries and Frameworks used:

- java.net.
- javax.crypto.BadPaddingException
- javax.crypto.IllegalBlockSizeException
- javax.crypto.NoSuchPaddingException
- javax.net.ssl
- java.security
- java.security.cert.CertificateException

### Programming Platforms used:

- Eclipse Java EE IDE for Web Developers.

## 8. Project Effort and Tasks Completed:

- Worked on establishing SSL Socket Connection between Server and Client using SSL certificates in the KeyStore File.
- Worked on implementing the Diffie-Hellman Parameter exchange and Public Key Exchange between the Client/Server through the SSL connection
- Troubleshooting problems in SSL connection, transferring of encrypted files from server to client and Diffie-Hellman Key exchange.
- Integration of the AES Encryption/Decryption with Diffie-Hellman Key Exchange and worked on the compatibility of all individual components.
- Worked on creating the framework that held together the Various Security Blocks.
- Worked on AES-Decryption of the file received at the client
- Worked on converting 'receive.txt' to 'decrypt.txt' using AES
- Troubleshooting problems related to decryption and problems relating to correctness of the encrypted file received at the client side.
- Helped in creating the executable Jar files needed for running the Client/server programs
- Worked on AES-128 Encryption of the file to be sent.
- Worked on converting 'inputMessage.txt' to 'encrypt.txt' using AES
- Worked on sending the Initialization Vector(IV) from the Server to Client which is used in AES-128 Decryption at the client side
- Worked on Server Certificate Generation and Client Certificate Generation.
- Worked on creating the KeyStore File and Key Pair for Diffie-Hellman.
- Worked on Troubleshooting SSL Handshakes Errors and Authentication errors.
- Worked on Creating Shared Key at the client/server side using the DH Parameters and Public Keys exchanged between the client/server.
- Troubleshooting problems related to the mismatches of Shared Keys at the server/client.
- Fixing problems with the key size compatibility issues of Shared Key with AES-128

## 9.SERVER SCREENSHOT

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\jinto jose\Desktop\Subjects_Study\CNS\projectnew\server>java -jar SSLServer.jar
Server : Server Started on Port Number 7777

Server : Waiting for client
-----Waiting for client-----
Server : Connection Recieved from Client
Client Message : Client : Awaiting to Receive from Server: Server Public Key and DH Parameters
-----Diffie-Hellman KeyExchange-----
Server : Initiating Diffie-Hellman KeyExchange:
Server : Creating Diffie-Hellman parameters...
Server : Generate DH keypair ...
Server: Initialization of DH KeyAgreement object..
Server: Sending Server Public Key along with the DH parameters
Server: Recieved Client Public Key
Server : Server Creates Shared key using Server Private Key and Client Public key
Server : Shared Key Generated by Server : F029D242CD669C3E815E8230193AA615
Client Message : Shared Key is generated ! Client Ready for file Tranfer
Server : Diffie-Hellman KeyExchange is Complete.
-----Encryption-----
Server : Starting Encryption of File to send using AES-128
Server : Reading from Input File 'inputMessage.txt'
-----File to be sent-----
File : inputMessage.txt
-----inputMessage.txt-----
Hello and welcome, jaorizabal. I slightly corrected your question. Maybe you can install a spell
with your mouse in the terminal, and then paste it into the edit field of your browser (and elsew
his is much faster than taking a screen shot, and uploading it.
```

Continued:

C:\Windows\System32\cmd.exe

```
-----Encryption-----
Server : Starting Encryption of File to send using AES-128
Server : Reading from Input File 'inputMessage.txt'

-----File to be sent-----
File : inputMessage.txt
-----inputMessage.txt-----
Hello and welcome, jaorizabal. I slightly corrected your question. Maybe you can inst
with your mouse in the terminal, and then paste it into the edit field of your brows
his is much faster than taking a screen shot, and uploading it.

-----
Server : Encryption Completed.'encrypt.txt' is generated
-----Encrypted File-----
File : encrypt.txt
-----encrypt.txt-----
99lr2l2pFOEOpRRrcNws/T3eTK5NmyZrwJC1sRCOE9Ekeys6yDTGBT/jRa5zBPvJSvU0WJ2f0Z6EwyJiq1iC
0GGmcPn1uiUKQtN50wKyQ7s4kj0TXUtwSLSxYiB0nDDI1RAvHD4/NfSPZ/5AKMw03JMis1om+20cACVQH6C
g0FNy+aABdz2aJSGmpmBFOE3JbAimeAXcfNwCLAfqZVSHzILRpwpvKd1PprbKx7BSW/WiidhJcV6soPaAUus
a+i/Re5e0Cd+SpM

-----
-----Sending encrypted file-----
Server : Sending encrypted file to the Client

Done! Secure file Transfer is Complete

Used:
-----Used-----
1.Connection between Server and Client using SSL
2.File Encryption/Decryption using AES-128
2.File Encryption/Decryption using Cipher Feedback Mode (CFB)
3.128 bit Key Exchange using Diffie-Hellman KeyExchange

C:\Users\jinto jose\Desktop\Subjects_Study\CNS\projectnew\server>
```

## 10.CLIENT SCREENSHOT

C:\Windows\System32\cmd.exe

Microsoft Windows [Version 10.0.14393]

(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\jinto jose\Desktop\Subjects\_Study\CNS\projectnew\client>java -jar SSLClient.jar

Client : Connecting to server on port 7777

Client : Connected to localhost/127.0.0.1:7777

-----Diffie-Hellman KeyExchange-----

Client : Initiating Diffie-Hellman KeyExchange:

Client : Recieved 'Server Public Key' and 'DH Parameters'

Client : Generate DH keypair ...

Client : Initialization ...

Client : Sending Client Public Key to Server

Client : Client creates Shared key using Client Private Key and Server Public key

Client : Shared Key Generated by Client : 'F029D242CD669C3E815E8230193AA615'

Client : Diffie-Hellman KeyExchange is Complete.

-----Receiving File from Server-----

Client : File received from Server saved in 'receive.txt'

-----Encrypted File Received from Server-----

File : receive.txt

-----receive.txt-----

991r2l2pF0E0pRRrcNws/T3eTK5NmyZrwJC1sRC0E9Ekeys6yDTGBT/jRa5zBPvJSvU0WJ2f0Z6EwyJiq1iCdZ7xkksKG  
0GGmcPn1uiUKQtN50wKyQ7s4kj0TXUtwSLSxYiB0nDDI1RAvHD4/NfSPZ/5AKMw03JMis1om+20cACVQH6Cf2s3jPB1F  
g0FNy+aABdz2aJSGmpmBFOE3JbAimeAXcfNwCLafqZVSHzILRpwpvKdIPrbKx7BSW/WiidhJcV6soPaAUusifJykefBg  
a+i/Re5e0Cd+SpM

-----Decryption-----

Client : File decryption is complete and saved in 'outputMessage.txt'

-----File received from Server-----

File : outputMessage.txt

**Continued:**

C:\Windows\System32\cmd.exe

```
Client : Client creates Shared key using Client Private Key and Server Public key

Client : Shared Key Generated by Client : 'F029D242CD669C3E815E8230193AA615'

Client : Diffie-Hellman KeyExchange is Complete.

-----Receiving File from Server-----

Client : File received from Server saved in 'receive.txt'

-----Encrypted File Received from Server-----
File : receive.txt

-----receive.txt-----
991r212pFOE0pRRrcNws/T3eTK5NmyZrwJC1sRCOE9Ekeys6yDTGBT/jRa5zBPvJSvU0WJ2f0Z6EwyJiq1iCdz
0GGmcPn1uiUKQtN50wKyQ7s4kj0TXUtwSLSxYiB0nDDI1RAvHD4/NfSPZ/5AKMw03JMis1om+20cACVQH6Cf2
g0FNy+aABdz2aJSGmpmBFOE3JbAimeAXcfNwCLAfqZVSHzILRpwpvKdlPprbKx7BSW/WiidhJcV6soPaAUusif
a+i/Re5e0Cd+SpM

-----Decryption-----

Client : File decryption is complete and saved in 'outputMessage.txt'

-----File received from Server-----
File : outputMessage.txt

-----outputMessage.txt-----
Hello and welcome, jaorizabal. I slightly corrected your question. Maybe you can insta
with your mouse in the terminal, and then paste it into the edit field of your browser
his is much faster than taking a screen shot, and uploading it.

-----Closing Connection with Server-----

C:\Users\jinto jose\Desktop\Subjects_Study\CNS\projectnew\client>
```

**Code References Used:**

1. AES-128 : [http://wiki.seconddlife.com/wiki/AES\\_Java\\_Implementation](http://wiki.seconddlife.com/wiki/AES_Java_Implementation)
2. Diffie-Hellman Key Exchange:  
<http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html#DH2>  
[Ex](#)