VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SECURE CHAT WITH IMPLEMENTATION OF AES WITH DIFFIE HELLMAN KEY EXCHANGE

A PROJECT REPORT

for

INFORMATION SECURITY MANAGEMENT(CSE3502)

in

B.Tech – CSE

by

Tushar Takshak -18BCE0203
Shikha Sah - 18BCE2458
Shadab Alam - 18BCE2491

Under the Guidance of

Prof. Rajarajan G.

May,2021

ABSTRACT

People need to communicate over insecure media in a secure manner, for this reason cryptography is an important tool to achieve this goal. In the past cryptography is mostly used for military applications to keep important information from enemies. Nowadays it is also used in other applications than military, and due to the fast technological progress, we need more and more sophisticated methods. The system proposed in the project uses Diffie-Hellman Key Exchange along with AES for encryption/decryption which provides an elegant method for data security.

INTRODUCTION

The motive of the project is to make a secure data transmission application with the help of Diffie-Hellman key exchange protocol and AES and study its applications especially for the local server key exchange and encryption.
Create a platform for an electronic business. Encipher sensitive data AES such as customer credit number, SSN etc. before transporting them from client to server.
The idea is simple, implementation of a secure instant message system between a client and server program. the client can access the server by typing java Client localhost 8080 it will get client connected to the already started server with same port number if the security criteria are matching on both client and server side. The client will communicate with server by Diffie-Hellman key- exchange protocol to generate the common shared key which is used in the later communication. After key exchange, both server and client will use the shared key to encipher and decipher the sensitive data such as customer credit number, SSN etc.

PROJECT DOMAIN

NETWORK SECURITY
INFORMATION SECURITY
LOCAL SERVER SECURE CHAT APPLICATION (Using java server programming)

PROBLEM STATEMENT

- In the current system, the key generation is very slow.
- Slow signing and decryption, which are slightly tricky to implement securely. The key is vulnerable to various attacks if poorly implemented.

PROPOSED SYSTEM ARCHITECTURE

The algorithm generates a public key and a private key for the client (say Alice). Alice then encodes her public key and sends it to Bob, who then retrieves the Diffie-Hellman parameters associated with Alice's public key and uses it to generate his own key pair.
Bob initializes his own Diffie-Hellman key pair.Bob encodes his public key and sends it over to Alice.At this stage, both Bob and Alice have created their public and private keys, and have exchanged their public keys. They will now generate a shared key.Bob and Alice write down their shared keys into a file, which are then read by the server and client servlets respectively.

a) Server Side

- Navigate to: <root-to-directory>/Secure-Chat-Client
- Compile the program: javac *.java
- Start the server: java Server 8080
- You will be prompted to enable/disable the three security properties.
- Confidentiality
- Integrity
- Authentication
- If authentication is enabled, you will be prompted for a password. Use: serverPass

b) Client Side

- Start the client: java Client localhost 8080
- You will be prompted to enable/disable the three security properties above.
- If authentication is enabled, you will be prompted for a password. Use: clientPass
- If the security settings for the server/client do not match connection will fail to establish.

LIST OF MODULES USED

Java security library (Key Generator for Diffie-Hellman)
Java crypto library (AES Encryption/ Decryption)

ALGORITHMS USED
DEFFIE HELLMAN KEY EXCHANGE ALGORITHM
ADVANCED ENCRYTION STANDARD (AES) ALGORITHM

TOOLS REQUIRED

IDE: Java integrated development environment software like NETBEANS or ECLIPSE.
JAVA DEVELOPMENT KIT
JAVA SERVER PROGRAMMING
Text editor like NOTEPAD++ or NOTEPAD
Commodity hardware shall be enough to get output

FINALISED SYSTEM ARCHITECTURE

Purpose
Implementation of a secure instant message system between a client and server program.

Security Requirements

- Confidentiality
- Integrity
- Authentication

The following requirements can be enabled/disabled in instructions 3 and 6 (see section 3).

Confidentiality
Confidentiality is ensuring that information is secure during transfer.

Key Establishment
If the server and client have both selected integrity, confidentiality or both, the client and server begin establishing a symmetric key pair amongst themselves to be used for AES encryption/decryption as well as the generation of message authentication codes.
The key establishment is done via the Diffie Hellman key establishment protocol, implemented using the Java Crypto library. This protocol uses

properties of modular arithmetic to establish a symmetric key between two parties without broadcasting the key over the network.
This is done using public values p and g, secret values a and b for each of the parties respectively and the property ((ga mod p)b mod p) = ((gb mod p)a mod p).
This means both parties can compute the key, without sending a or b across. First each party computes (gmySecret mod p) and sends it to the other. Now each party has (gothersSecret mod p) and can perform ((gothersSecret mod p)mySecret mod p) resulting in the same key for both parties without 'mySecret' or 'othersSecret' being relayed. The technical sequence of these events can be found in Section 5.

Encryption and Decryption

The confidentiality property is achieved by encrypting messages on the sending side and decrypting on the receiving side between the client and server communication using AES CBC mode. This requirement is implemented using the Java.crypto and Java.security libraries along with the symmetric session key (see section 2.2.1).

Integrity

Integrity is ensuring data has not been tampered with during transfer.

Message Authentication Codes

The integrity property is achieved by sending a message authentication code (MAC) with the message. A MAC is a one way hash of a message and is used in a similar way as a virtual checksum. The sender sends a MAC along with the original message. On the receiving end, the original message is hashed using the same algorithm and compared to the received MAC to ensure that data has not been tampered with. This requirement is implemented using the Java.crypto and Java.security libraries, using SHA-256.

Authentication

Authentication is ensuring the person sending data is who they say they are.

Password Authentication

If the user selects the authentication security requirement, they will be prompted to enter their password in order to prove that they own the private key they intend to sign their messages with. This is done by storing the hash of the respective client/server password in the pass file contained in their private directories. Once the user enters their password, the value will be hashed using SHA-256 and compared with the hash contained inside their private directory.
In the real world this password would be used to authenticate that the user attempting to access the private key used in step 2 is who they say they are. This

could be achieved by having a password protected directory/file which contains a private key. For the sake of this assignment, please assume that directories containing private hashes and keys are properly access controlled.

Signing and Verification

Authentication is also enforced by sending a digital signature that is verified on the receiving end. A digital signature is made by creating a one way hash of a message and encrypting it with the sender's private key. It is verified on the receiving end by decrypting the signature with the sender’s public key, creating the hash of the original message, and comparing the two outcomes. This protocol ensures that the message could have only been sent by the owner of the public key used to decrypt the signature. This requirement is implemented using the Java.crypto and Java.security libraries.

ALGORITHMS USED

ADVANCED ENCRYPTION STANDARD(AES)

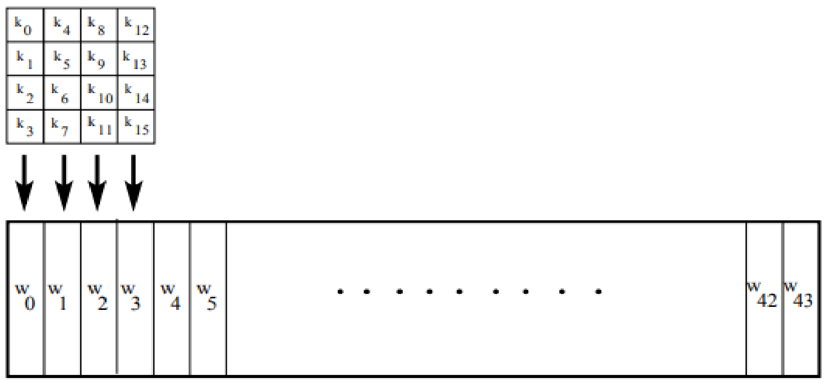
- AES is a block cipher with a block length of 128 bits.
- AES allows for three different key lengths: 128, 192, or 256 bits. Most of our discussion will assume that the key length is 128 bits.
- Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.
- Except for the last round in each case, all other rounds are identical.
- Each round of processing includes one single-byte based substitution step, a row-wise permutation step, a column-wise mixing and the addition of the round key. The order in which these four steps are executed is different for encryption and decryption.
- To appreciate the processing steps used in a single round, it is best to think of a 128-bit block as consisting of a 4 × 4 array of bytes,
- AES also has the notion of a word. A word consists of four bytes, that is 32 bits. Therefore, each column of the state array is a word, as is each row.
- Each round of processing works on the input state array and produces an output state array.
- The output state array produced by the last round is rearranged into a 128-bit output block.
- Unlike DES, the decryption algorithm differs substantially from the encryption algorithm. Although, overall, very similar steps are used in encryption and decryption, their implementations are not identical and the order in which the steps are invoked is different, as mentioned previously.
- Like DES, AES is an iterated block cipher in which plaintext is subject to

multiple rounds of processing, with each round applying the same overall transformation function to the incoming block. [When we say that each round applies the same transformation function to the incoming block, that similarity is at the functional level. However, the implementation of the transformation function in each round involves a key that is specific to that round — this key is known as the round key. Round keys are derived from the user-supplied encryption key.]

- Unlike DES, AES is an example of key-alternating block ciphers. In such ciphers, each round first applies a diffusion-achieving transformation operation — which may be a combination of linear and nonlinear steps — to the entire incoming block, which is then followed by the application of the round key to the entire block.
- For another point of contrast between DES and AES, whereas DES is a bit-oriented cipher, AES is a byte-oriented cipher. The substitution step in DES requires bit-level access to the block coming into a round. On the other hand, all operations in AES are purely byte-level, which makes for convenient and fast software implementation of AES.
- About the security of AES, considering how many years have passed since the cipher was introduced in 2001, all of the threats against the cipher remain theoretical — meaning that their time complexity is way beyond what any computer system will be able to handle for a long time to come.

ENCRYTPION KEY AND ITS EXPANSION

- Assuming a 128-bit key, the key is also arranged in the form of an array of 4 × 4 bytes. As with the input block, the first word from the key fills the first column of the array, and so on.
- The four column words of the key array are expanded into a schedule of 44 words. (As to how exactly this is done, we will explain that later in Section 8.8.) Each round consumes four words from the key schedule.
- Figure 1 below depicts the arrangement of the encryption key in the form of 4-byte words and the expansion of the key into a key schedule consisting of 44 4-byte words. Of these, the first four words are used for adding to the input state array before any round-based processing can begin, and the remaining 40 words used for the ten rounds of processing that are required for the case a 128-bit encryption key.

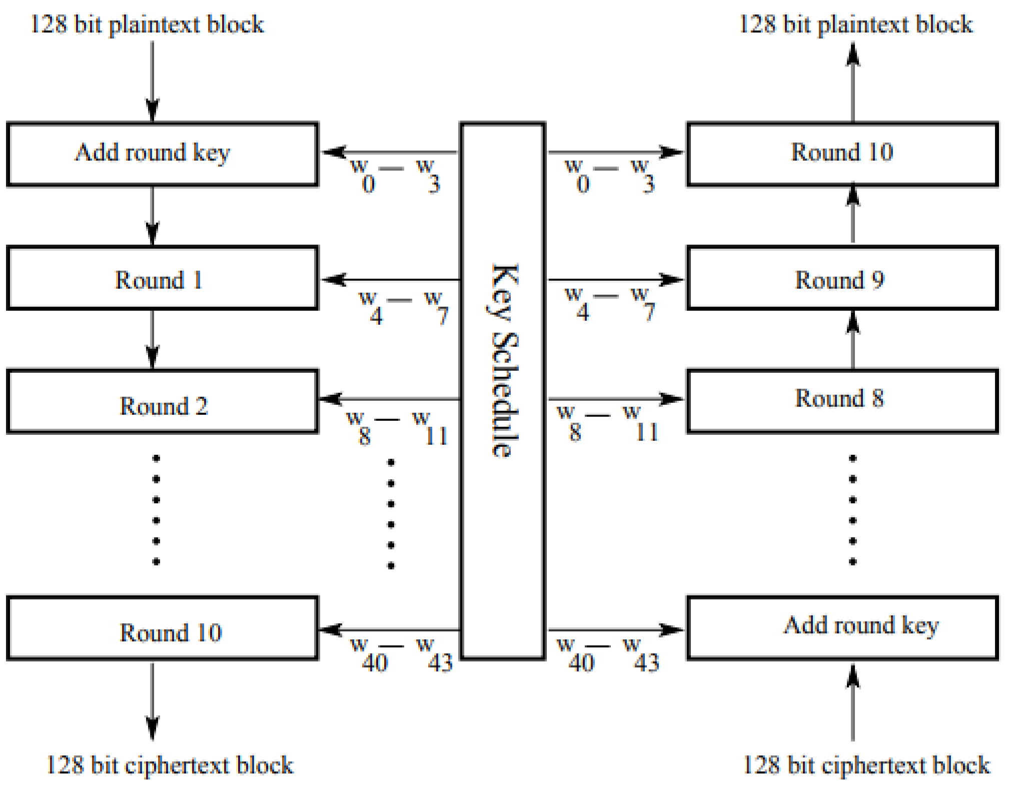


OVERALL STRUCTURE

1. The overall structure of AES encryption/decryption is shown in Figure 2.
2. The number of rounds shown in Figure 2, 10, is for the case when the encryption key is 128 bit long. (As mentioned earlier, the number of rounds is 12 when the key is 192 bits, and 14 when the key is 256)
3. Before any round-based processing for encryption can begin, the input state array is XORed with the first four words of the key schedule. The same thing happens during decryption — except that now we, XOR the ciphertext state array with the last four words of the key schedule.
4. For encryption, each round consists of the following four steps:
   - Substitute bytes,
   - Shift rows,
   - Mix columns, and
   - Add round key. T
   - The last step consists of XORing the output of the previous three steps with four words from the key schedule
5. For decryption, each round consists of the following four steps:
   - Inverse shift rows,
   - Inverse substitute bytes,
   - Add round key, and
   - Inverse mix columns.
6. The third step consists of XORing the output of the previous two steps with four words from the key schedule. Note the differences between the order in which substitution and shifting operations are carried out in a decryption round vis-a-vis the order in which similar operations are carried out in an encryption round.
7. The last round for encryption does not involve the “Mix columns” step. The last round for decryption does not involve the “Inverse mix columns” step.

THE FOUR STEPS IN EACH ROUND OF PROCESSING

1. Called SubBytes for byte-by-byte substitution during the forward process. The corresponding substitution step used during decryption is called InvSubBytes. This step consists of using a 16 × 16 lookup table to find a replacement byte for a given byte in the input state array. The entries in the lookup table are created by using the notions of multiplicative inverses in GF (28) and bit scrambling to destroy the bit-level correlations inside each byte
2. Called ShiftRows for shifting the rows of the state array during the forward process.The goal of this transformation is to scramble the byte order inside each 128-bit block.
3. Called MixColumns for mixing up of the bytes in each column separately during the forward process. The corresponding transformation during decryption is denoted InvMixColumns and stands for inverse mix column transformation. The goal is here is to further scramble up the 128-bit input block. The shift-rows step along with the mix-column step causes each bit of the ciphertext to depend on every bit of the plaintext after 10 rounds of processing.
4. Called AddRoundKey for adding the round key to the output of the previous step during the forward process. The corresponding step during decryption is denoted InvAddRoundKey for inverse add round key transformation.



AES Encryption AES Decryption

DIFFIE HELLMAN KEY EXCHANGE ALGORITHM

The question of key exchange was one of the first problems addressed by a cryptographic protocol. This was prior to the invention of public key cryptography. The Diffie-Hellman key agreement protocol (1976) was the first practical method for establishing a shared secret over an unsecured

communication channel. The point is to agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key.

STEPS IN THE ALGORITHM

- Alice and Bob agree on a prime number p and a base g.
- Alice chooses a secret number a, and sends Bob (g a mod p).
- Bob chooses a secret number b, and sends Alice (g b mod p).
- Alice computes ((g b mod p) a mod p).
- Bob computes ((g a mod p) b mod p). Both Alice and Bob can use this number as their key. Notice that p and g need not be protected.

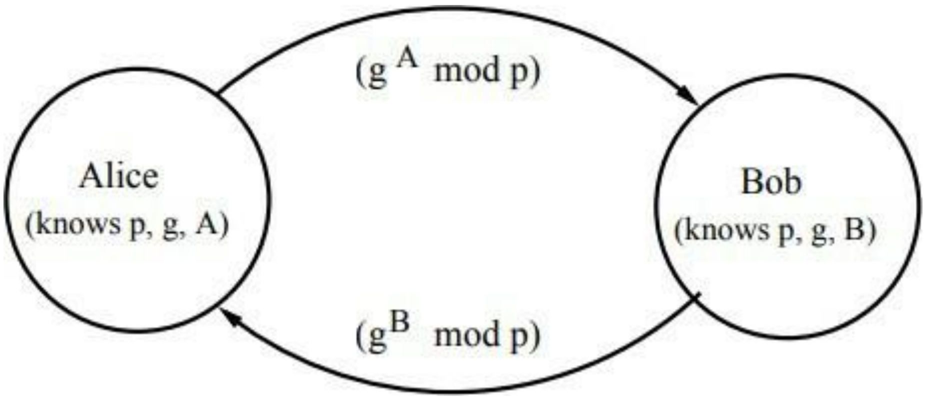
EXAMPLE

- Alice and Bob agree on p = 23 and g = 5.

- Alice chooses a = 6 and sends 5 6 mod 23 = 8.
- Bob chooses b = 15 and sends 515 mod 23 = 19.
- Alice computes 19 6 mod 23 = 2.
- Bob computes 815 mod 23 = 2. Then 2 is the shared secret. Clearly, much larger values of a, b, and p are required. An eavesdropper cannot discover this value even if she knows p and g and can obtain each of the messages.

DEFFIE HELLMAN KEY EXCHANGE

Suppose ‘p’ is a prime of around 300 digits, and ‘a’ and ‘b’ at least 100 digits each. Discovering the shared secret given g, p, g a mod p and g b mod p would take longer than the lifetime of the universe, using the best known algorithm. This is called the discrete logarithm problem.



Additional details related to the client/server interaction

Connection Establishment
Once the server has received an open session message, it verifies that the client has the same security protocol as the one chosen when the server was first started. This is done by having the client send a string containing "CIAN" each of which corresponds to confidentiality, integrity, authentication and none respectively. For example, if the server receives the string "CI" It checks if its own security requirements have confidentiality and integrity but not authentication. If the client's security requirements match the server's security requirements, their connection is successfully established.

Communication
The IM application implements chat using Java's socket programming libraries. The server binds a socket to the specified port and then listens for connections on that socket. Once the client connects to the socket, the server creates a new client handler object and initializes it. The client handler object allows for the

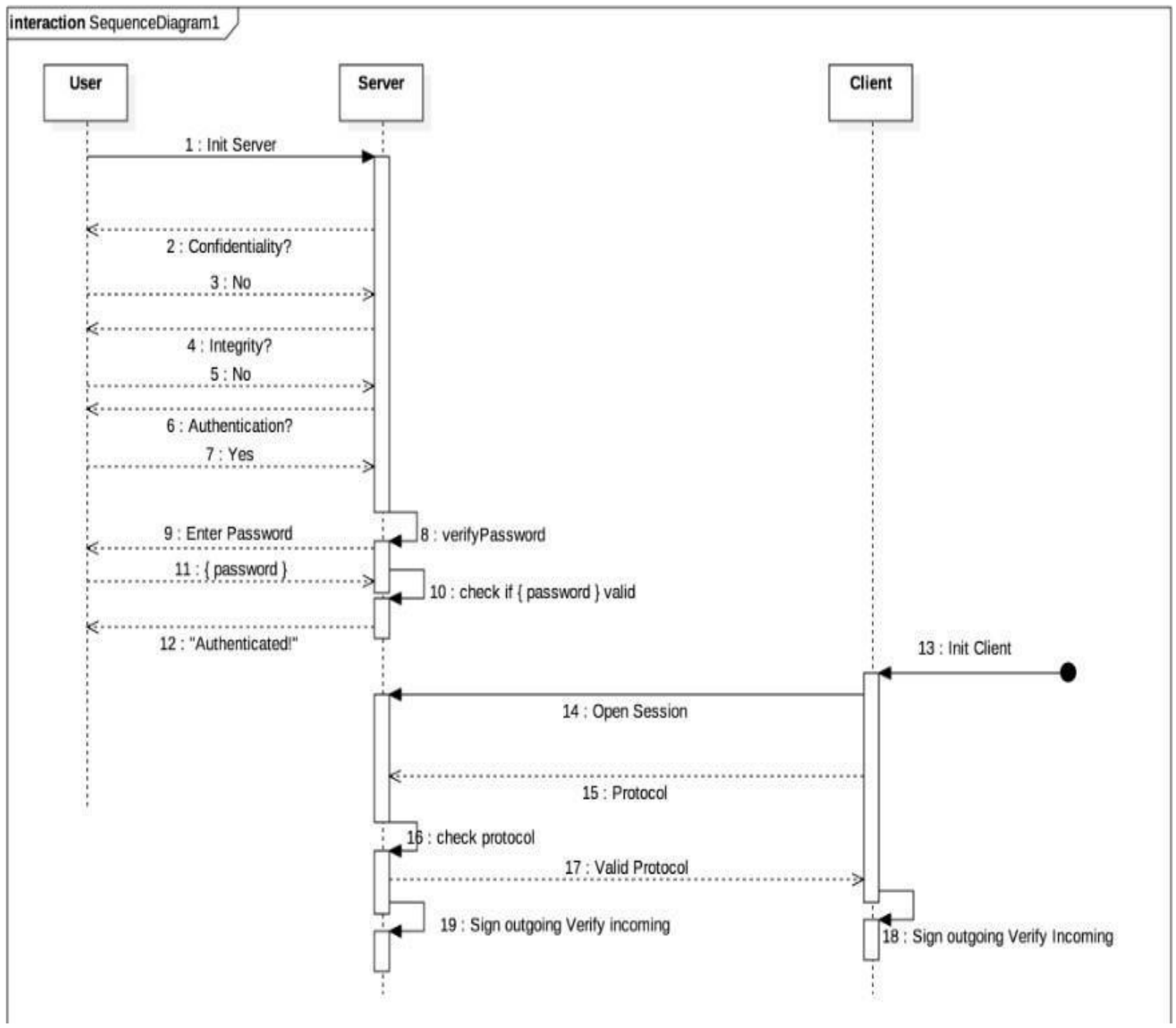
server to handle clients connecting and disconnecting without interrupting the server. Both the client and the server use two threads to communicate with each other. One thread is for reading from standard input and writing to the output stream of the socket, the other thread is for reading from the socket and printing to the console. The client and server are then able to send and receive messages simultaneously. Once the client is finished they can simple send the bye message and both the client and server will clean up properly.

Message Formatting and Parsing

Because the client and server need to send not only the intended message, but also a signature as well as a message authentication code, there needs to be some sort of formatting and parsing mechanism. Once the sender has successfully gathered the necessary information(message,signature,mac) they will format the message by separating each piece of information with ;;;. Three semicolons were chosen because it is unlikely that a user will ever enter 3 semicolons in their normal message. Once the message is received, the value is parsed back into the Communications object which allows for easy access of the the authentication, integrity and confidentiality fields.

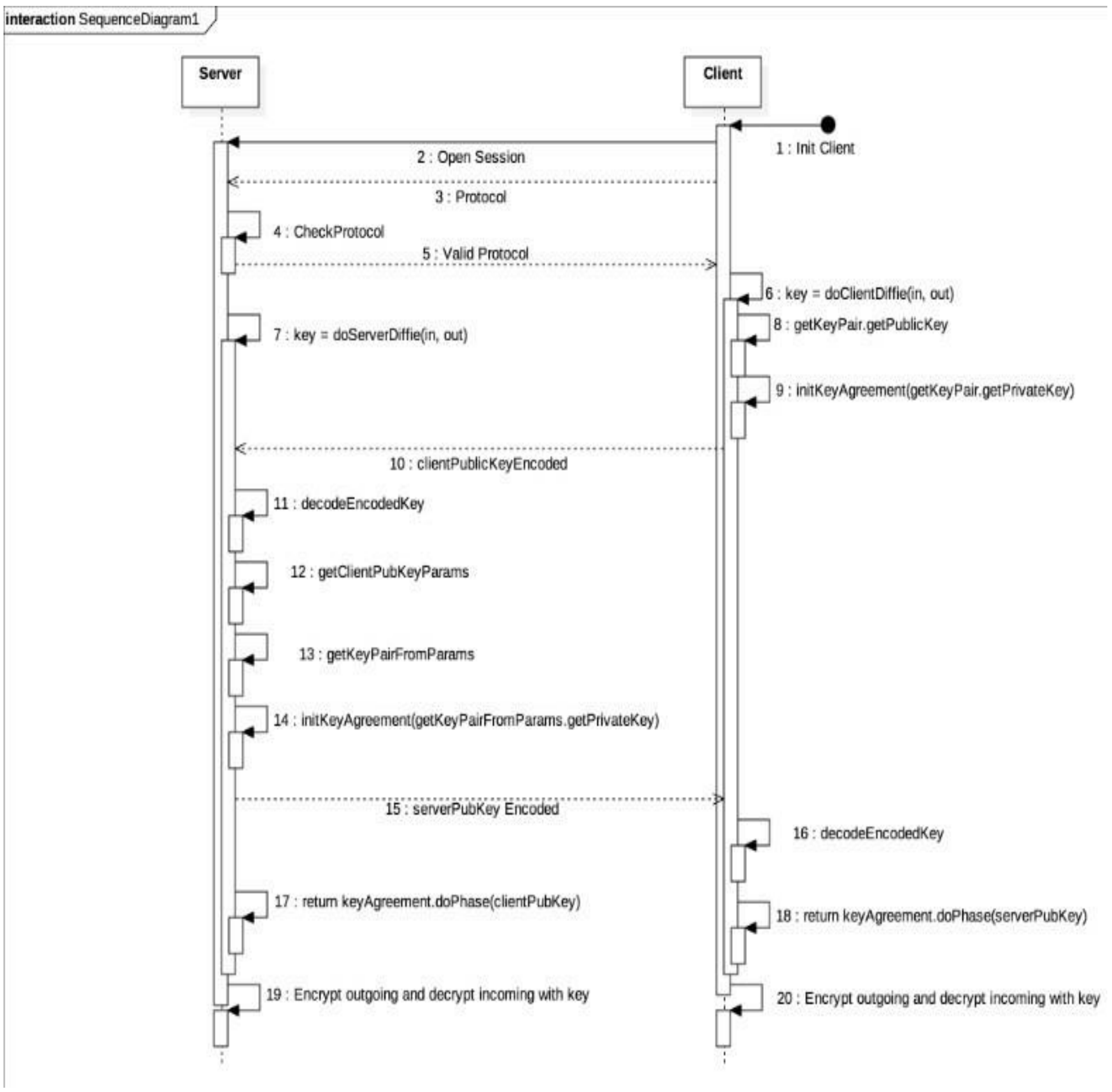
FLOW DIAGRAM

Authentication Feature Interaction:



The Authentication Feature Interaction diagram above displays the sequence of events, as they would occur, should the authentication security feature be required. The above diagram is a ‘happy case’ and as such, assumes the password entered is correct and that the protocols on both client and server match.

Diffie Hellman Key Establishment Interaction:


interaction SequenceDiagram1
Server
Client
1 : Init Client
2 : Open Session
3 : Protocol
4 : CheckProtocol
5 : Valid Protocol
6 : key = doClientDiffie(in, out)
7 : key = doServerDiffie(in, out)
8 : getKeyPair.getPublicKey
9 : initKeyAgreement(getKeyPair.getPrivateKey)
10 : clientPublicKeyEncoded
11 : decodeEncodedKey
12 : getClientPubKeyParams
13 : getKeyPairFromParams
14 : initKeyAgreement(getKeyPairFromParams.getPrivateKey)
15 : serverPubKey Encoded
16 : decodeEncodedKey
17 : return keyAgreement.doPhase(clientPubKey)
18 : return keyAgreement.doPhase(serverPubKey)
19 : Encrypt outgoing and decrypt incoming with key
20 : Encrypt outgoing and decrypt incoming with key

The Diffie Hellman Key Establishment Interaction diagram, shown on the previous page, displays the sequence of interactions between the client and server as they establish an asymmetric key for further communication. Steps 1-5 are covered in the previous diagram as well, but to summarize, these steps establish a connection between the server and client and ensure that the security protocols match on both. Steps 6 and 7 represent both the client and server calling their diffieHellman functions respectively. In step 8, the client generates a KeyPair (private and public key) using an instance of ‘DH’ or DiffieHellman, with default parameters and then in 9 a KeyAgreement object, which is initialized using the private key from 8. This KeyAgreement object provides the functionality of a key agreement protocol using the specified algorithm (in our case, diffie hellman). In step 10, the client public key encoded is sent to the server. In steps 11-14 the server extracts the DH parameters stored in the client public key encoded, and generates its own KeyPair and KeyAgreement object (NOTE: the server KeyAgreement object is initialized using the server's private key). In step 15 the server then sends its own public key encoded to the client. After step 16, in which the client decodes the server's key, both the client and server are in a state where they have their own KeyPairs, generated using the same DH parameters as each other, as well as the public key of the other party. Step 17 and 18 represent the ‘next phase’ of this key agreement. The doPhase() is done on both server and client KeyAgreement objects, but it is done with the other’s public key. (i.e server does doPhase(clientPubKey)). The doPhase of both server and client, if everything is done correctly, should return the same shared secret, which can then be used as the asymmetric key for future communication. The public key sent across by the client in step 10 represents the public values p and g, where each of the parties private keys act as their respective secret a and b’s.

CODING IMPLEMENTATION:

Github Link https://github.com/AlamShadab/Secure-Chat-Application

Client.java

```
import java.net.*;
import java.io.*;
importjava.util.*;
importjava.nio.file.*;
importjavax.crypto.Mac;

//Class that handles client side communication
public class Client {

//Define needed objects/variables
static byte[] key = null;
static Security security;
static Communication communication;
static Cryptography crypto;
static final String closed = "Closed connection with server.";

//Starts the client and connects to the specific server:port
public static void startClient(String serverName, intserverPort) throws
UnknownHostException, IOException {
System.out.println("Trying to connect to host: " + serverName + ": " + serverPort);
Socket socket = new Socket(serverName, serverPort);
System.out.println("Sending open session message.");
```

```
//Make the socket and get the I/O streams.
BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
OutputStreamclientStream = socket.getOutputStream();
InputStreamserverStream = socket.getInputStream();
crypto = new Cryptography();
communication = new Communication();

//Send the protocol(CIA) that the client is using, disconnect if not matching.
if (!verifyProtocol(serverStream, clientStream)) {
disconnect(input, serverStream, clientStream, socket);
return;
}

//If we want encryption or checksum we need to establish a symetric key pair using
diffiehelmen
if (security.confidentiality || security.integrity) {
key = ClientDiffie.doDiffie(serverStream, clientStream);
}

//Spins up a thread for reading from input and sending formatted messages to the server
Thread sendMessage = new Thread(new Runnable() {
@Override
public void run() {
String send;
try {
while (true) {
//These functions will only return something useful if we have the respective setting turned
on.
send = input.readLine();
//Format the message
byte[] signature = crypto.sign(send.getBytes(),
Paths.get("client_private", "private.der"), security.authentication);
byte[] mac = crypto.generateMAC(send.getBytes(), key, security.integrity);
byte[] message = crypto.encrypt(send.getBytes(), key, security.confidentiality);
byte[] formattedMessage = communication.format(message, signature, mac);
clientStream.write(formattedMessage);

//The client can close the connection by writing bye to the server
if (send.toString().equals("bye")) {
disconnect(input, serverStream, clientStream, socket);
break;
}
}
} catch (IOExceptionioe) {
System.out.println(closed);
}
}
});

//Spins up a thread for reading from the input socket stream and printing to console.
```

```
Thread readMessage = new Thread(new Runnable() {
@Override
public void run() {
try {
while (true) {
byte[] msg = new byte[16 * 1024];
int count = serverStream.read(msg);
msg = Arrays.copyOf(msg, count);
String message = communication.handleMessage(msg,
Paths.get("client_private", "publicServer.der"), crypto, key, security);
System.out.println("server: " + message);
}
} catch (IOExceptionioe) {
System.out.println(closed);
}
}
});

sendMessage.start();
readMessage.start();
}

//Clean up the connection with the server.
public static void disconnect(BufferedReader input, InputStreamserverStream,
OutputStreamclientStream,
Socket socket) {
try {
input.close();
serverStream.close();
clientStream.close();
socket.close();
} catch (IOExceptionioe) {
System.out.println(closed);
}
}

//Sends the protocol chosen to the server, acts depending on its reply.
public static booleanverifyProtocol(InputStreamserverStream, OutputStreamclientStream) {
String protocol = "N";
if (security.authentication) {
protocol += "a";
}
if (security.integrity) {
protocol += "i";
}
if (security.confidentiality) {
protocol += "c";
}
String s = "";
try {
```

```
clientStream.write(protocol.getBytes());
byte[] msg = new byte[16 * 1024];
int count = serverStream.read(msg);
s = new String(msg, 0, count, "US-ASCII");
} catch (IOExceptionioe) {
System.out.println(closed);
}
if (s.contains("Valid protocol")) {
System.out.println(s);
return true;
} else {
System.out.println(s);
return false;
}
}

//Reads from the command line and starts the client.
public static void main(String args[]) {
if (args.length != 2) {
System.out.println("Usage: java Client <host><port>");
return;
}
security = new Security();

//If they want authentication, verify their password.
if (security.authentication) {
PasswordTools.verifyPassword(Paths.get("client_private", "pass"));
}

String hostName = args[0];
intportNumber = Integer.parseInt(args[1]);
try {
startClient(hostName, portNumber);
} catch (UnknownHostExceptionuhe) {
System.out.println("Host unknown: " + uhe.getMessage());
} catch (IOExceptionioe) {
System.out.println("Unexpected exception: " + ioe.getMessage());
}

}
}
```

Server.java

```
import java.net.*;
import java.io.*;
importjava.util.*;
importjava.nio.file.*;
importjavax.crypto.Mac;

//Server side of the chat
```

```
public class Server {
//reads users input and listens on the specific port.
public static void main(String[] args) throws Exception {
intportNumber = 8080;
Security security = null;
if (args.length != 1) {
System.out.println("Usage: java server <port>");
return;
}
security = new Security();

//If they want authentication, verify their password.
if (security.authentication) {
PasswordTools.verifyPassword(Paths.get("server_private", "pass"));
}
portNumber = Integer.parseInt(args[0]);

//Setup server
booleanisOver = false;
BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Binding to port " +portNumber);
ServerSocket server = new ServerSocket(portNumber);
System.out.println("Server started: " + server);

//Loops and creates new ClientHandler objects, allows for server to persist even when client
closes connection.
while (!isOver) {
Socket socket = server.accept();
System.err.println("Open session message recieved, comparing protocol.");
ClientHandler handler = new ClientHandler(socket, input, security);
handler.handleClient();
}
server.close();
input.close();
}
}

//Class used to handle new clients accessing the socket
classClientHandler {
//Needed variables
staticBufferedReader input;
staticOutputStreamserverStream;
staticInputStreamclientStream;
static Socket socket;
staticboolean connected;
static Security security;
static byte[] key = null;
static Cryptography crypto;
static Communication communication;
```

```
//Constructor for established socket and I/O
publicClientHandler(Socket socket, BufferedReader input, Security security) {
this.socket = socket;
this.input = input;
this.security = security;
this.connected = true;
}

//Actually read and write to the client
public static void handleClient() throws IOException {
serverStream = socket.getOutputStream();
clientStream = socket.getInputStream();
crypto = new Cryptography();
communication = new Communication();

try {
if (invalidProtocol(clientStream, serverStream)) {
return;
}
} catch (IOExceptionioe) {
System.out.println("Client closed connection.");
}
if (security.confidentiality || security.integrity) {
key = ServerDiffie.doDiffie(clientStream, serverStream);
}

//Spins up a thread for reading from input and sending to outputstream
Thread sendMessage = new Thread(new Runnable() {
@Override
public void run() {
String send;
byte[] encrypted;
try {
while (connected) {
send = input.readLine();
if (!socket.isClosed()) {
//Format the message.
byte[] signature = crypto.sign(send.getBytes(),
Paths.get("server_private", "private.der"),
security.authentication);
byte[] mac = crypto.generateMAC(send.getBytes(), key, security.integrity);
byte[] message = crypto.encrypt(send.getBytes(), key,
security.confidentiality);
byte[] finalMessage = communication.format(message, signature, mac);
serverStream.write(finalMessage);
}
}
} catch (IOExceptionioe) {
System.out.println("Client closed connection.");
}
}
});

//Spin up a thread to read from inputstream and write to command line.
Thread readMessage = new Thread(new Runnable() {
@Override
public void run() {
try {
while (true) {
byte[] msg = new byte[16 * 1024];
int count = clientStream.read(msg);
msg = Arrays.copyOf(msg, count);
String message = communication.handleMessage(msg,
Paths.get("server_private", "publicClient.der"), crypto, key, security);
System.out.println("client: " + message);
if (message.equals("bye")) {
System.out.println("Client closed connection.");
disconnect();
connected = false;
break;
}
}
} catch (IOExceptionioe) {
System.out.println("Client closed connection.");
}
}
});

sendMessage.start();
readMessage.start();
}

//Clean up streams.
public static void disconnect() {
try {
serverStream.close();
clientStream.close();
socket.close();
} catch (IOException e) {
e.printStackTrace();
}
}

//Recieve protocol from client and validate that it is the same as servers.
public static booleaninvalidProtocol(InputStreamclientStream, OutputStreamserverStream)
throws IOException {
byte[] msg = new byte[16 * 1024];
int count = clientStream.read(msg);
String s = new String(msg, 0, count, "US-ASCII");
String errorLog = "invalid security protocol, dropping connection.";
```

```
if ((s.contains("a") && !security.authentication) || (!s.contains("a")
&&security.authentication)) {
serverStream.write("invalid security protocol, authentication not matching. re-establish
connection."
.getBytes());
System.out.println(errorLog);
return true;
}
if ((s.contains("i") && !security.integrity) || (!s.contains("i") &&security.integrity)) {
serverStream.write(
"invalid security protocol, integrity not matching. re-establish connection.".getBytes());
System.out.println(errorLog);
return true;
}
if ((s.contains("c") && !security.confidentiality) || (!s.contains("c")
&&security.confidentiality)) {
serverStream.write("invalid security protocol, confidentiality not matching. re-establish
connection."
.getBytes());
System.out.println(errorLog);
return true;
}
String reply = "Valid protocol.";
if (security.confidentiality || security.integrity) {
reply += "Beginning DH.";
}
System.out.println(reply);
serverStream.write(reply.getBytes());
return false;
}
}
```

DHkeyAgreement.java

```
import java.io.*;
importjava.util.*;
importjava.math.BigInteger;
importjava.security.*;
importjava.security.spec.*;
importjava.security.interfaces.*;
importjavax.crypto.*;
importjavax.crypto.spec.*;
importjavax.crypto.interfaces.*;
importcom.sun.crypto.provider.SunJCE;

//Class used to establish symetric keys between client and server.
classDiffieTools {

//Define needed variables
public byte[] pubKeyEncoded;
public final PublicKeypubKey;
public final KeyPairkPair;
public final byte[] pubKeyEnc;
public final intkeySize = 2048;

//Server Constructor
publicDiffieTools(byte[] pubKeyEncoded) {
this.pubKeyEncoded = pubKeyEncoded;
this.kPair = genKeyPair();
this.pubKey = getPubKey(this.kPair);
this.pubKeyEnc = getPubKeyEncFromDHPublicKey(this.pubKey);
}

//Client Constructor
publicDiffieTools() {
this(null);
}

//Takes encoded public key and returns a PublicKey object
publicPublicKeygetDHPublicKeyFromEncoded(byte[] pubKeyEncoded)
throwsNoSuchAlgorithmException, InvalidKeyException {
KeyFactorykf = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(pubKeyEncoded);
try {
PublicKeydhPubKey = kf.generatePublic(x509KeySpec);
returndhPubKey;
} catch (Exception e) {
System.out.println("The algorithm specified in the encoded public key does not match any in
record");
return null;
}
}

//Takes PublicKey object and returns encoded version.
public byte[] getPubKeyEncFromDHPublicKey(PublicKeydhPublicKey) {
returndhPublicKey.getEncoded();
}

//Given keypair returns the public key.
publicPublicKeygetPubKey(KeyPairkPair) {
returnkPair.getPublic();
}

//Initializes a diffiehelmen key agreement instance with private key.
publicKeyAgreementgetInitializedKeyAgreement(PrivateKeyprivKey) {
KeyAgreementkAgree;
try {
kAgree = KeyAgreement.getInstance("DH");
} catch (Exception e) {
System.out.println("The algorithm specified in the encoded public key does not match any in
record");
```

```
return null;
}

try {
kAgree.init(privKey);
} catch (Exception e) {
System.out.println("Invalid Key");
return null;
}
returnkAgree;
}

//Generates a key pair for diffiehelmen.
publicKeyPairgenKeyPair() {
DHParameterSpecdhParamFromSomeonesPubKey = null;
KeyPairGeneratormyKpairGen;
try {
myKpairGen = KeyPairGenerator.getInstance("DH");
} catch (NoSuchAlgorithmException e) {
System.out.println("The algorithm specified does not match any on record");
return null;
}

//Creates a Keypair using someones encoded pubKeys DH Params
if (this.pubKeyEncoded != null) {
byte[] myPubKeyEncoded;
KeyFactorymyKeyFac;
try {
myKeyFac = KeyFactory.getInstance("DH");
} catch (Exception e) {
System.out.println(e);
return null;
}
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(pubKeyEncoded);
PublicKeysomeonesPubKey;

try {
//Get PublicKey Object from Encoded
someonesPubKey = myKeyFac.generatePublic(x509KeySpec);
//Extract the DH Params from the PublicKey
dhParamFromSomeonesPubKey = ((DHPublicKey) someonesPubKey).getParams();
} catch (Exception e) {
System.out.println("The key used by the other person was invalid");
return null;
}

try {
//Initialize the generator with the same DH parameters as someone.
myKpairGen.initialize(dhParamFromSomeonesPubKey);
} catch (Exception e) {
```

```
System.out.println("The algorithm specified does not match any on record");
return null;
}

} else {
//initialize the generator with a 2048 keysize.
myKpairGen.initialize(keySize);
}

//Generate the KeyPair from the initialized generator.
KeyPairmyKpair;
try {
myKpair = myKpairGen.generateKeyPair();
returnmyKpair;
} catch (Exception e) {
System.out.println("Some error occured in keypair generation");
return null;

}
}
}

//Generates a symetric key pair for the server
classServerDiffie {
public static byte[] doDiffie(InputStream in, OutputStream out) {
byte[] clientPubKeyEnc = new byte[16 * 1024];
DiffieToolsserverTools;
PublicKeyserverPubKey;
PublicKeyclientPubKey;
KeyPairserverKpair;
byte[] serverPubKeyEnc;
intnumRead;
KeyAgreementserverAgree;
byte[] serverSharedSecret = new byte[256];

try {
//Get PubKeyEncoded
numRead = in.read(clientPubKeyEnc);
serverTools = new DiffieTools(clientPubKeyEnc);
serverPubKeyEnc = serverTools.pubKeyEnc;
serverKpair = serverTools.kPair;
serverAgree = serverTools.getInitializedKeyAgreement(serverKpair.getPrivate());

out.write(serverPubKeyEnc);

clientPubKey = serverTools.getDHPublicKeyFromEncoded(clientPubKeyEnc);
serverAgree.doPhase(clientPubKey, true);
numRead = serverAgree.generateSecret(serverSharedSecret, 0);
} catch (Exception ioe) {
System.out.println(ioe);
```

```
}
returnserverSharedSecret;
}
}

//Generates a symetric key pair for the client.
classClientDiffie {
public static byte[] doDiffie(InputStream in, OutputStream out) {
byte[] serverPubKeyEnc = new byte[16 * 1024];
DiffieToolsclientTools;
PublicKeyclientPubKey;
PublicKeyserverPubKey;
KeyPairclientKpair;
byte[] clientPubKeyEnc;
intnumRead;
KeyAgreementclientAgree;
byte[] clientSharedSecret = new byte[256];

try {
clientTools = new DiffieTools();
clientPubKey = clientTools.pubKey;
clientPubKeyEnc = clientTools.pubKeyEnc;
clientKpair = clientTools.kPair;
clientAgree = clientTools.getInitializedKeyAgreement(clientKpair.getPrivate());

//Send PubKeyEncoded
out.write(clientPubKeyEnc);
numRead = in.read(serverPubKeyEnc);

serverPubKey = clientTools.getDHPublicKeyFromEncoded(serverPubKeyEnc);
clientAgree.doPhase(serverPubKey, true);
numRead = clientAgree.generateSecret(clientSharedSecret, 0);
} catch (Exception ioe) {
System.out.println(ioe);
}
returnclientSharedSecret;
}
}
```

Password.java

```
importjava.io.File;
importjava.io.FileReader;
importjava.io.IOException;
importjava.util.*;
importjava.nio.file.*;
importjava.security.MessageDigest;

//A helper file for comparing password entered with hashed password stored in secure file.
classPasswordTools {
```

```
//Hash the password and compare it to the hash stored in the file.
public static booleanvalidPassword(String password, Path path) {
bytebyteData[] = null;
bytetoCompare[] = null;
try {
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(password.getBytes());
byteData = md.digest();
toCompare = Files.readAllBytes(path);
} catch (Exception e) {
System.out.println("Failed to read file");
}
if (Arrays.equals(toCompare, byteData)) {
return true;
}
return false;
}

//Prompts the user to enter their password and then verifies it.
static void verifyPassword(Path path) {
Scanner reader = new Scanner(System.in);
System.out.println("What is your password?");
while (true) {
if (validPassword(reader.nextLine(), path)) {
System.out.println("Authenticated!");
break;
} else {
System.out.println("Incorrect password, try again.");
}
}
}
}
```

Cryptography.java

```
importjavax.crypto.Cipher;
importjavax.crypto.Mac;
importjavax.crypto.spec.IvParameterSpec;
importjavax.crypto.spec.SecretKeySpec;
importjava.security.spec.*;
importjava.security.*;
import java.io.*;
importjava.util.*;
importjava.nio.file.*;

//Class that handles all of the cryptographic functions used by the client and server
public class Cryptography {

//Define the needed variables
static private intivSize = 16;
static private intkeySize = 16;
static private PrivateKeyprivateKey;
static private PublicKeypublicKey;

//Reads public key into memory, returns it if already read once.
private static PublicKeyreadPublicKey(Path path)
throwsIOException, NoSuchAlgorithmException, InvalidKeySpecException {
if (publicKey == null) {
X509EncodedKeySpec publicSpec = new X509EncodedKeySpec(Files.readAllBytes(path));
KeyFactorykeyFactory = KeyFactory.getInstance("RSA");
publicKey = keyFactory.generatePublic(publicSpec);
}
returnpublicKey;
}

//Reads private key into memory, returns the key if already read once.
private static PrivateKeyreadPrivateKey(Path path)
throwsIOException, NoSuchAlgorithmException, InvalidKeySpecException {
if (privateKey == null) {
PKCS8EncodedKeySpec keySpec = new
PKCS8EncodedKeySpec(Files.readAllBytes(path));
KeyFactorykeyFactory = KeyFactory.getInstance("RSA");
privateKey = keyFactory.generatePrivate(keySpec);
}
returnprivateKey;
}

//Compares 2 message authentication codes if integrity was chosen.
public static Boolean compareMAC(byte[] m1, byte[] m2, boolean integrity) {
if (!integrity) {
return true;
}
try {
if (!Arrays.equals(m1, m2))
return false;
} catch (Exception e) {
System.out.println("Arrays.equal failure");
}
return true;
}

//Generates a message authentication code if integrity was chosen.
public static byte[] generateMAC(byte[] message, byte[] key, boolean integrity) {
try {
if (!integrity) {
return "".getBytes();
}
//Cast key to a byte array and generate a SecretKeySpec needed for mac.init
SecretKeySpeckeySpec = new SecretKeySpec(key, "AES");
// Generate MAC
Mac mac = Mac.getInstance("HmacSHA256");
mac.init(keySpec);
byte[] result = mac.doFinal(message);
return result;
} catch (Exception e) {
System.out.println("Error in AES.generateMAC: " + e);
return null;
}
}

//Helper function for encrypt/decrpt
public static byte[] digestMessage(byte[] message) {
try {
MessageDigest digest = MessageDigest.getInstance("SHA-1");
digest.update(message);
byte[] messageDigest = digest.digest();
returnmessageDigest;
} catch (Exception e) {
System.out.println("Error in AES.digestMessage: " + e);
return null;
}
}

//Compares two digests
public static Boolean compareDigests(byte[] d1, byte[] d2) {
if (!Arrays.equals(d1, d2))
return false;
return true;
}

//Combines plaintext with senders private key to create a signature which can be verified
using the senders public key.
public static byte[] sign(byte[] plainText, Path path, boolean authenticity) {
byte[] sign = null;
try {
if (!authenticity) {
return "".getBytes();
}
PrivateKeyprivKey = readPrivateKey(path);
Signature signer = Signature.getInstance("SHA256withRSA");
signer.initSign(privKey);
signer.update(plainText);

sign = signer.sign();
} catch (Exception e) {
System.out.println("Error in signing");
}
return sign;
}
```

```
//Verifies the recieved signature by using the senders public key.
public static Boolean verify(byte[] plainText, byte[] signature, Path path, boolean
authenticity) {
byte[] signatureBytes = null;
Signature verifier = null;
boolean verified = false;
try {
if (!authenticity) {
return true;
}
PublicKeypubKey = readPublicKey(path);
verifier = Signature.getInstance("SHA256withRSA");
verifier.initVerify(pubKey);
verifier.update(plainText);

signatureBytes = signature;
verified = verifier.verify(signatureBytes);
} catch (Exception e) {
System.out.println("Error in verify");
}
return verified;
}

//Encrypts a message using AES symetric keys established during diffiehelmen.
public static byte[] encrypt(byte[] message, byte[] key, boolean encryption) {
try {
if (!encryption) {
return message;
}
// Generating IV Spec
byte[] iv = new byte[ivSize];
SecureRandom random = new SecureRandom();
random.nextBytes(iv);
IvParameterSpecivSpec = new IvParameterSpec(iv);

// GeneratekeySpec
key = Arrays.copyOf(digestMessage(key), 16);
SecretKeySpeckeySpec = new SecretKeySpec(key, "AES");

// Create and initialize the cipher for encryption
Cipher aesCipher;
aesCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
aesCipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);

// Encrypt the cleartext
byte[] cleartext = message;
byte[] ciphertext = aesCipher.doFinal(cleartext);

//return IV + ciphertext
byte[] encryptedMessage = new byte[ivSize + ciphertext.length];
System.arraycopy(iv, 0, encryptedMessage, 0, ivSize);
System.arraycopy(ciphertext, 0, encryptedMessage, ivSize, ciphertext.length);
returnencryptedMessage;
} catch (Exception e) {
System.out.println("Error in AES.encrypt: " + e);
return null;
}
}

//Decrypts the message using symetric AES keys.
public static byte[] decrypt(byte[] message, byte[] key, boolean decryption) {
try {
if (!decryption) {
return message;
}
// Split the encrypted message into IV and Ciphertext
byte[] iv = new byte[ivSize];

byte[] ciphertext = new byte[message.length - ivSize];

System.arraycopy(message, 0, iv, 0, ivSize);
System.arraycopy(message, ivSize, ciphertext, 0, message.length - ivSize);

// CreateivSpec and keySpec
IvParameterSpecivSpec = new IvParameterSpec(iv);
key = Arrays.copyOf(digestMessage(key), 16);
SecretKeySpeckeySpec = new SecretKeySpec(key, "AES");

// Create and initialize the cipher for encryption
Cipher aesCipher;
aesCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
aesCipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);

// Encrypt the cleartext
byte[] cleartext = aesCipher.doFinal(ciphertext);
returncleartext;
} catch (Exception e) {
System.out.println("Error in AES.decrypt: " + e);
return null;
}
}
}
```

Communication.java

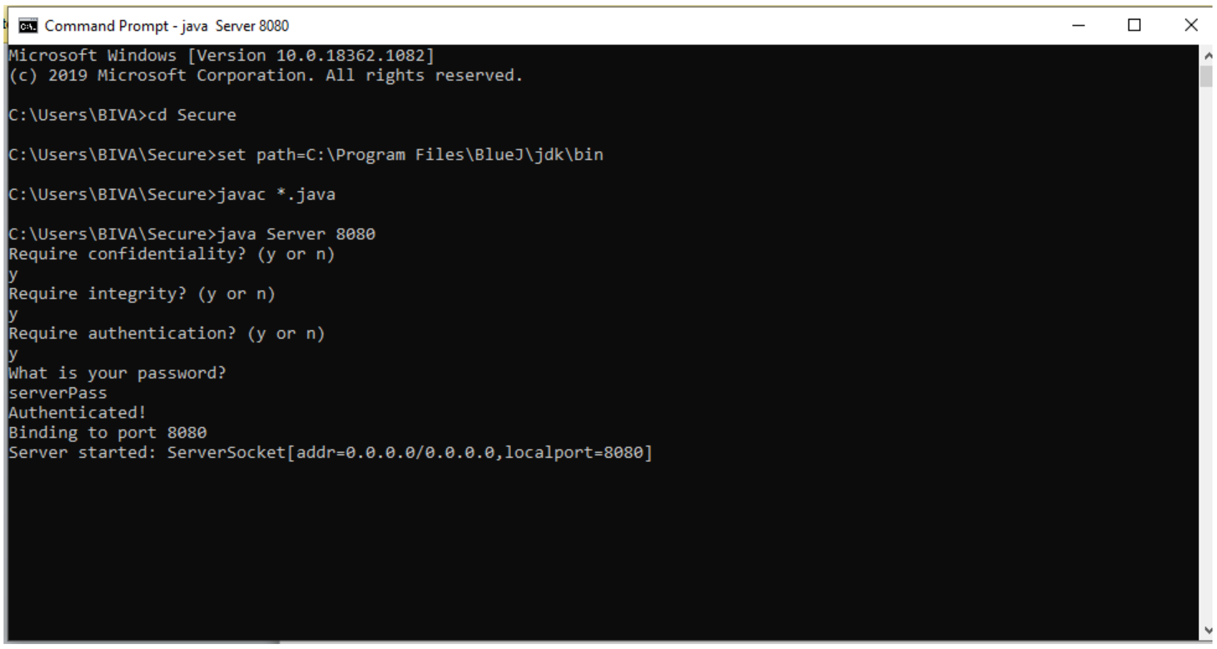
```
importjava.util.*;
importjava.math.BigInteger;
importjava.nio.file.*;
importjava.lang.*;
```

```
//Class that handles the functions needed for the client and server to communicate with
eachother.
class Communication {

//The parsed field values
public static byte[] message;
public static byte[] signature;
public static byte[] mac;

//Converts a message into the respective message, signature, mac(checksum) fields.
public static void parse(byte[] com) {
try {
String str = new String(com);
String[] portions = str.split(";;;");

if (portions.length> 3) {
System.out.println("The Communication object contains incorrect coding");
}

if (portions[0].equals("")) {
System.out.println("The Communication object does not contain the message");
} else {
message = Base64.getDecoder().decode(portions[0].getBytes());
if (portions.length> 1) {
signature = Base64.getDecoder().decode(portions[1].getBytes());
}
if (portions.length> 2) {
mac = Base64.getDecoder().decode(portions[2].getBytes());
}
}
} catch (Exception e) {
System.out.println("Parse error");
}
}

//Formats the message into a string using ;;; as a delimiter between message, signature and
mac.
public byte[] format(byte[] message, byte[] signature, byte[] mac) {
String delimeter = ";;;";

byte[] encodedMessage = Base64.getEncoder().encode(message);
byte[] encodedSignature = Base64.getEncoder().encode(signature);
byte[] encodedMac = Base64.getEncoder().encode(mac);

String messageString = new String(encodedMessage);
String sigString = new String(encodedSignature);
String macString = new String(encodedMac);
String communication = messageString + delimeter + sigString + delimeter + macString;
returncommunication.getBytes();
}
//1. Decrypts the message if confidentiality was chosen.
//2. Verifys the signature sent with the message if authentication was chosen.
//3. Compares the computed checksum with the recieved checksum if integrity was chosen.
public static String handleMessage(byte[] information, Path path, Cryptography crypto,
byte[] key,
Security security) {
parse(information);
byte[] msg = crypto.decrypt(message, key, security.confidentiality);
if (!crypto.verify(msg, signature, path, security.authentication)) {
System.out.println("Authentication failed, signature from message does not match");
}
byte[] macActual = crypto.generateMAC(msg, key, Security.integrity);
byte[] macExpected = mac;
if (!crypto.compareMAC(macActual, macExpected, security.integrity)) {
System.out.println("Integrity failed, checksum of message does not match expected!");
}
String s = "";
try {
s = new String(msg);
} catch (Exception e) {
System.out.println("byte to string conversion error");
}
return s;
}
}
```

Security.java

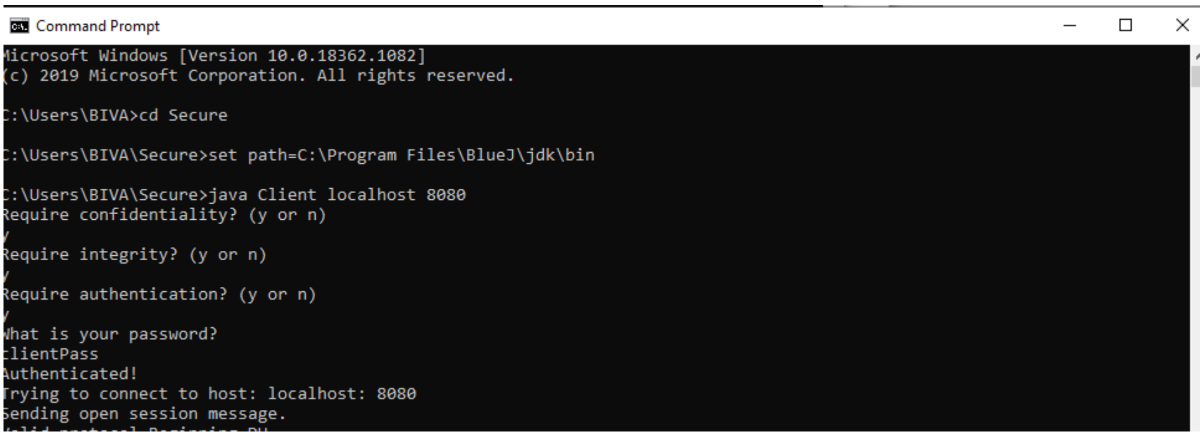
```
importjava.util.*;

//Checks the security options that the user wants
public class Security {
//Fields used for the security object
public static Boolean confidentiality;
public static Boolean integrity;
public static Boolean authentication;

public Security() {
checkUserInput();
}

//Deals with users inputs and sets the respective field values
public static void checkUserInput() {
// Prompt user to enable/disable security properties
Scanner reader = new Scanner(System.in);
String c, i, a;

System.out.println("Require confidentiality? (y or n)");
while (true) {
c = reader.nextLine();
if (c.equals("y")) {
confidentiality = true;
break;
} else if (c.equals("n")) {
confidentiality = false;
break;
}
System.out.println("Invalid input, try again.");
}

System.out.println("Require integrity? (y or n)");
while (true) {
i = reader.nextLine();
if (i.equals("y")) {
integrity = true;
break;
} else if (i.equals("n")) {
integrity = false;
break;
}
System.out.println("Invalid input, try again.");
}

System.out.println("Require authentication? (y or n)");
while (true) {
a = reader.nextLine();
if (a.equals("y")) {
authentication = true;
break;
} else if (a.equals("n")) {
authentication = false;
break;
}
System.out.println("Invalid input, try again.");
}
}
}
```

OUTPUT SCREENSHOTS:



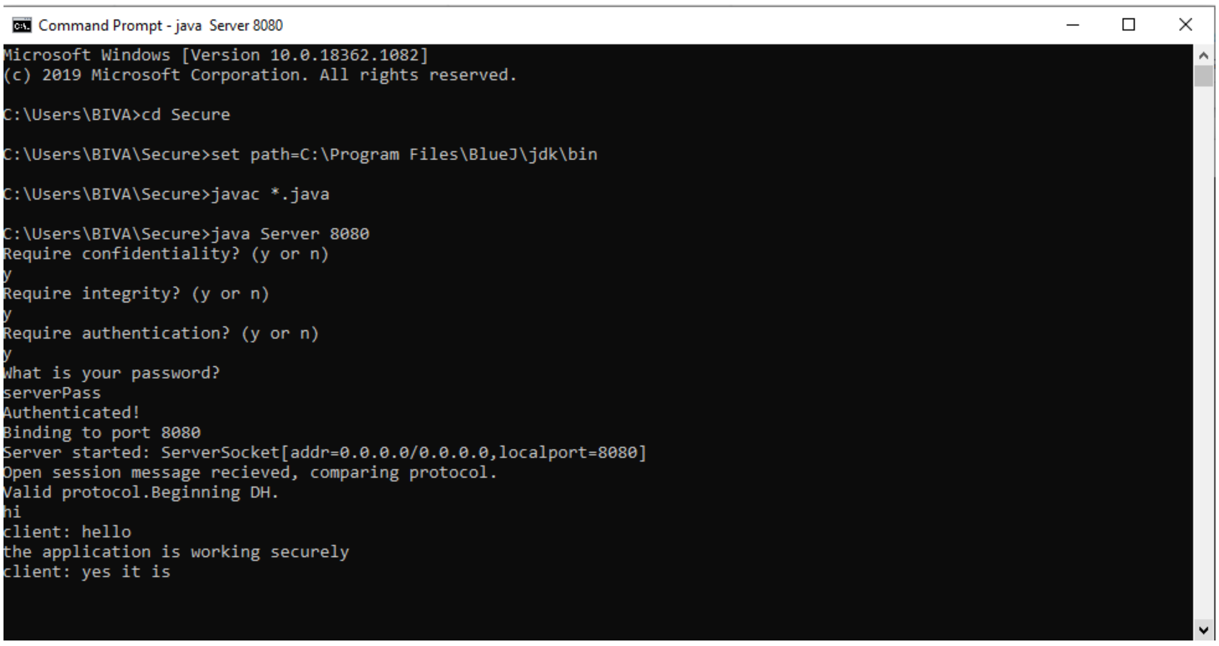
```
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\BIVA>cd Secure

C:\Users\BIVA\Secure>set path=C:\Program Files\BlueJ\jdk\bin

C:\Users\BIVA\Secure>javac *.java

C:\Users\BIVA\Secure>java Server 8080
Require confidentiality? (y or n)
y
Require integrity? (y or n)
y
Require authentication? (y or n)
y
What is your password?
serverPass
Authenticated!
Binding to port 8080
Server started: ServerSocket[addr=0.0.0.0/0.0.0.0,localport=8080]
```

This is the server side where connection is established if the security protocols are present and matched with the client side.



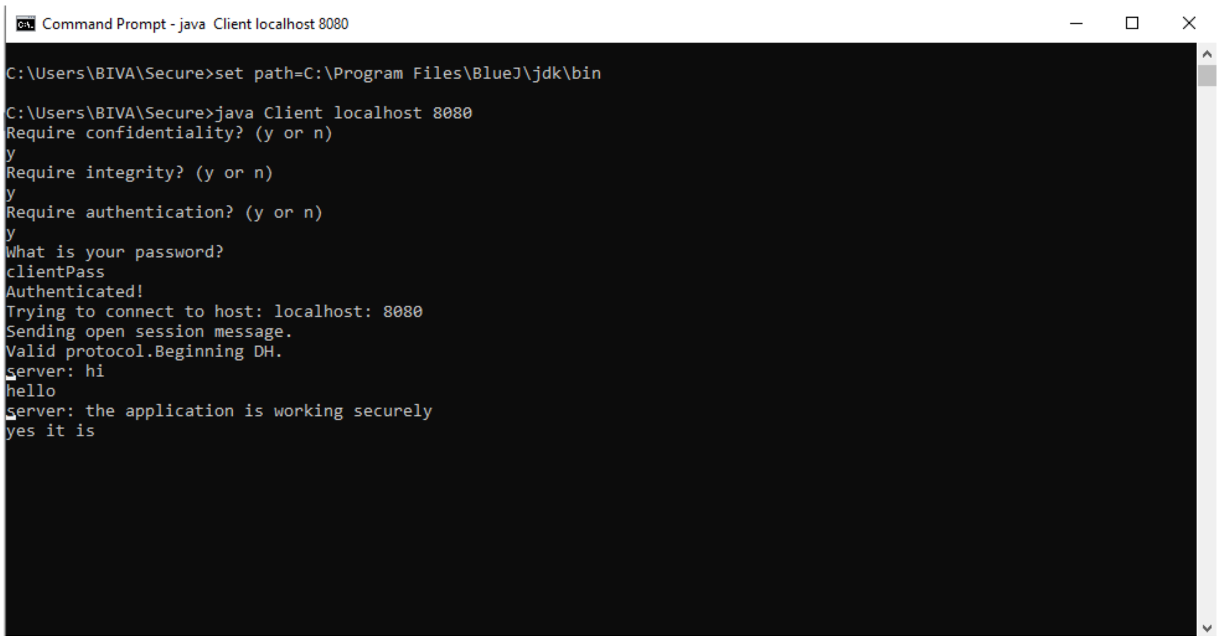
```
icrosoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

:\Users\BIVA>cd Secure

:\Users\BIVA\Secure>set path=C:\Program Files\BlueJ\jdk\bin

:\Users\BIVA\Secure>java Client localhost 8080
Require confidentiality? (y or n)

Require integrity? (y or n)

Require authentication? (y or n)

What is your password?
lientPass
uthenticated!
rying to connect to host: localhost: 8080
ending open session message.
```

This is the client side where connection is established due to the matching protocols with server side.



Command Prompt - java Server 8080

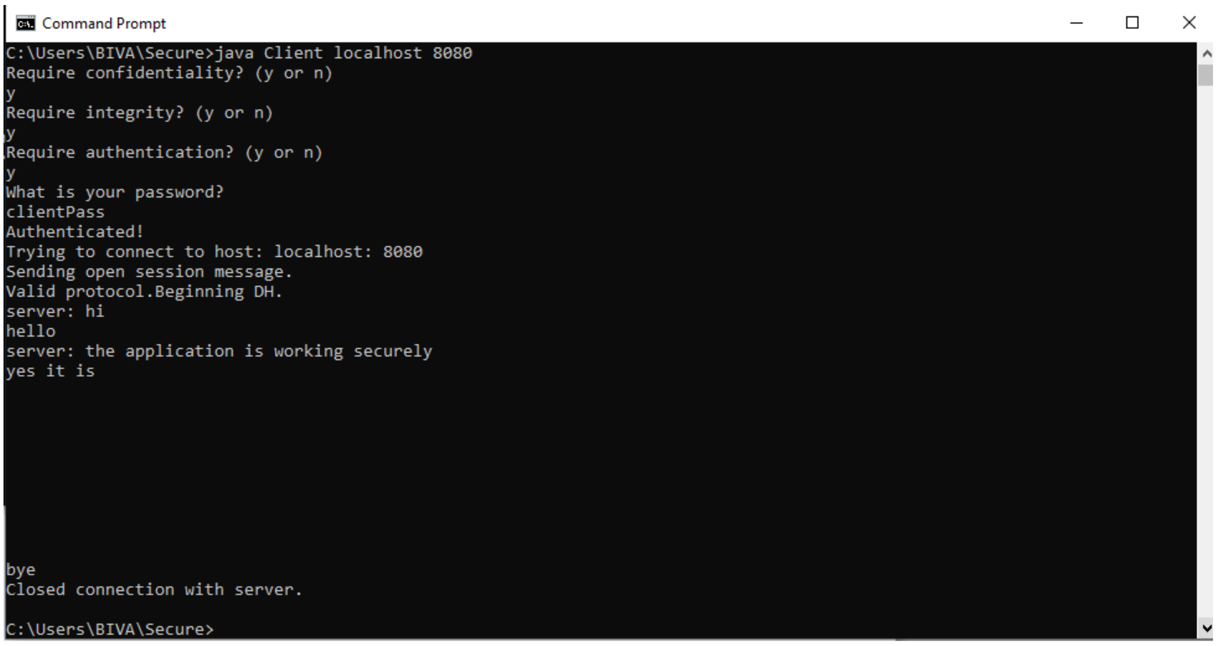
```
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\BIVA>cd Secure

C:\Users\BIVA\Secure>set path=C:\Program Files\BlueJ\jdk\bin

C:\Users\BIVA\Secure>javac *.java

C:\Users\BIVA\Secure>java Server 8080
Require confidentiality? (y or n)
y
Require integrity? (y or n)
y
Require authentication? (y or n)
y
What is your password?
serverPass
Authenticated!
Binding to port 8080
Server started: ServerSocket[addr=0.0.0.0/0.0.0.0,localport=8080]
Open session message recieved, comparing protocol.
Valid protocol.Beginning DH.
hi
client: hello
the application is working securely
client: yes it is
```



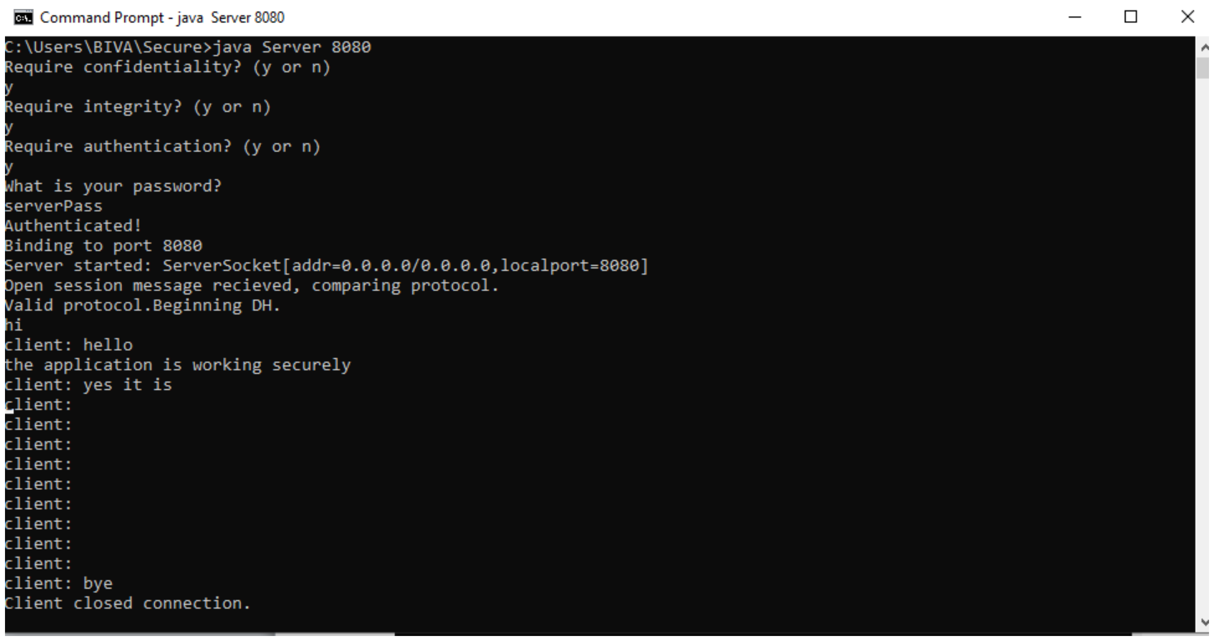
Command Prompt - java Client localhost 8080

```
C:\Users\BIVA\Secure>set path=C:\Program Files\BlueJ\jdk\bin

C:\Users\BIVA\Secure>java Client localhost 8080
Require confidentiality? (y or n)
y
Require integrity? (y or n)
y
Require authentication? (y or n)
y
What is your password?
clientPass
Authenticated!
Trying to connect to host: localhost: 8080
Sending open session message.
Valid protocol.Beginning DH.
server: hi
hello
server: the application is working securely
yes it is
```

Server and Client are securely chatting after the connection is established.



```
C:\Users\BIVA\Secure>java Client localhost 8080
Require confidentiality? (y or n)
y
Require integrity? (y or n)
y
Require authentication? (y or n)
y
What is your password?
clientPass
Authenticated!
Trying to connect to host: localhost: 8080
Sending open session message.
Valid protocol.Beginning DH.
server: hi
hello
server: the application is working securely
yes it is









bye
Closed connection with server.

C:\Users\BIVA\Secure>
```



```
C:\Users\BIVA\Secure>java Server 8080
Require confidentiality? (y or n)
y
Require integrity? (y or n)
y
Require authentication? (y or n)
y
What is your password?
serverPass
Authenticated!
Binding to port 8080
Server started: ServerSocket[addr=0.0.0.0/0.0.0.0,localport=8080]
Open session message recieved, comparing protocol.
Valid protocol.Beginning DH.
hi
client: hello
the application is working securely
client: yes it is
client:
client:
client:
client:
client:
client:
client:
client:
client:
client: bye
Client closed connection.
```

The connection is closed when the client says bye.

APPLICATIONS

SSL: The Diffie-Hellman protocol has been applied to many security protocols including the Security Sockets Layer (SSL), secure shell (SSH), and IP Sec. The SSL is the standard security technology developed by Netscape in 1994 to establish an encrypted link between a web server and a browser. This link ensures privacy and integrity of all data passed between the web server and browsers. SSL is used by millions of websites in the protection of their online transactions with their customers. SSL is all about encryption. SSL uses certificates, private/public key exchange pairs and Diffie-Hellman key agreements to provide privacy (key exchange), authentication and integrity with Message Authentication Code (MAC). This information is known as a cipher suite and exists within a Public Key Infrastructure (PKI). SSL is useful for business/financial traffic, e.g. credit card transactions. SSL ensures confidentiality (it prevents eavesdropping), authenticity (the sender is really who he says he is), and integrity (the message has not been changed en route). It is possible that a user might not know SSL is used in the course of communication but they are likely to notice some blockages.

SSH: SSH is a network security protocol very common for secure remote login on the Internet. The secure shell has come to replace the unsecured Telnet on the network and FTP on the system, mostly because both Telnet and FTP do not encrypt data, and instead send them in plaintext. SSH, on the other hand, can automatically encrypt, authenticate and compress transmitted data. Diffie-Hellman algorithm is used for this purpose.

IPSec (IP Security): IPSec (IP Security) is an extension of the Internet Protocol (IP)—it is a suite of protocols introduced by the Internet Engineering Task Force (IETF) to aid in configuring a communications channel between multiple machines. Operating at the IP layer of the seven-layer model, it does its job by authenticating and encrypting IP packets. Like the previous protocols, IPSec uses D-H and asymmetric cryptography to establish identities, preferred algorithms, and a shared secret. Before IPSec can begin encrypting the data stream, some preliminary information exchange is necessary. This is accomplished with the Internet Key Exchange (IKE) protocol.

CONCLUSION

The growing need for adding security around chat applications has become very important. With a large number of people using mobile chat applications, it becomes very important that chat applications are developed using useful security practices for better resistance to vulnerability. Otherwise, there is high risk of data getting stolen by hackers. In this project, a successful secure chat communication between a client and a server is developed using Diffie Hellman key exchange method and Advanced Encryption Standard (AES) for encrypting/decrypting. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. Advanced

Encryption Standard (AES) algorithm is one on the most common and widely symmetric block cipher algorithm used in worldwide.
Thus, the two algorithms together provide a good base for developing a secure system. We also discussed some of the applications of the proposed system.

REFERENCES

[1] Hamdani, H., Ismanto, H., Munir, A. Q., Rahmani, B., Syafrianto, A., Suprihanto, D., &Septiarini, A. (2018). The Proposed Development of Prototype with Secret Messages Model in Whatsapp Chat. International Journal of Electrical & Computer Engineering (2088- 8708)

[2] Wardhono, W. S., Priandani, N. D., Ananta, M. T., Brata, K. C., & Tolle, H. (2018). End-toend privacy protection for facebook mobile chat based on aes with multi-layered md5. International Journal of Interactive Mobile Technologies (iJIM), 12(1)

[3] Bhattacharjya, A., Zhong, X., & Wang, J. (2018). An end-to-end user two-way authenticated double encrypted messaging scheme based on hybrid RSA for the future internet architectures. International Journal of Information and Computer Security, 10(1).

[4] Nyakomitta, P. S., Ogara, S., &Abeka, S. O. (2017). Secure end point data security using java application programming interface.

[5] Ali, A. H., &Sagheer, A. M. (2017). Design of an Android Application for Secure Chatting. International Journal of Computer Network & Information Security, 9(2).

[6] Sabah, N., Kadhim, J. M., & Dhannoon, B. N. (2017). Developing an End-to-End Secure Chat Application. IJCSNS, 17(11)

[7] Wadkar, P., & Honale, S. SECURITY USER DATA IN LOCAL CONNECTIVITY USING MULTICAST KEY AGREEMENT.

[8] Samiksha Sharma and Vinay Chopra(2017), Data Encryption using Advanced Encryption Standard with Key Generation by Elliptic Curve Diffie-Hellman.International Journal of Security and its Applications Vol11,No. 3

[9] Saifurrab, C., & Mirza, S. (2016). AES algorithm using advance key implementation in MATLAB. International Research Journal of Engineering

and Technology, 3(09).

[10] S. Grace Sophia and S. Prabakeran (2016). Efficient and Secure Data Sharing Using AES and Diffie Hellman Key Exchange Algorithm in cloud. Middle-East Journal of Scientific Research 24 (Special Issue on Innovations in Information, Embedded and Communication Systems):