Project 1:

Introduction to Cryptography Application

By:

Rakshit Shetty

#50133314

Overview:

The goal of this project is to create a simple server client login system and make it more secure by implementing cryptographic algorithms available and get a better understanding of these cryptographic algorithm. In this day of age when most of your communication is done electronically and most of our data is stored on the internet there is a need for good cryptographic algorithms. One such instance where we need such algorithms is in a login system. There are many types of attacks that an intruder that wants to know your personal details may use, some of which are:

- Brute force attack
- Dictionary attack
- Meet in the middle attack
- Eavesdropping

What we are trying to do is make our login system immune against most of these cryptographic attacks. The project itself is divided into three phases

- Target 0: Here we are just creating a simple server client login system with no cryptographic algorithms what so ever.

- Target 1: We make use of SHA-1 for password storage in the server along with AES to encrypt the communication between the server and the client.

- Target 2: Key agreement algorithm Diffie-Hellman is implemented

Target 0:

A remote login system between a server and client was designed using java programming. Sockets where used to connect the server and client. This forms the base of the entire project.

The server creates a socket at port 9999 and the client creates a socket connecting to the server socket.

Server: *echoServer = **new** ServerSocket(9999);*

Client : *smtpSocket = **new** Socket("localhost", 9999);*

The server creates another socket to communicate with client.

Server: *clientSocket = echoServer.accept();*

Both server and client use object input stream and object output stream for communication.

Server: *is = **new** ObjectInputStream(clientSocket.getInputStream());*

*os = **new** ObjectOutputStream(clientSocket.getOutputStream());*

Client: *os = **new** ObjectOutputStream(smtpSocket.getOutputStream());*

*is = **new** ObjectInputStream(smtpSocket.getInputStream());*

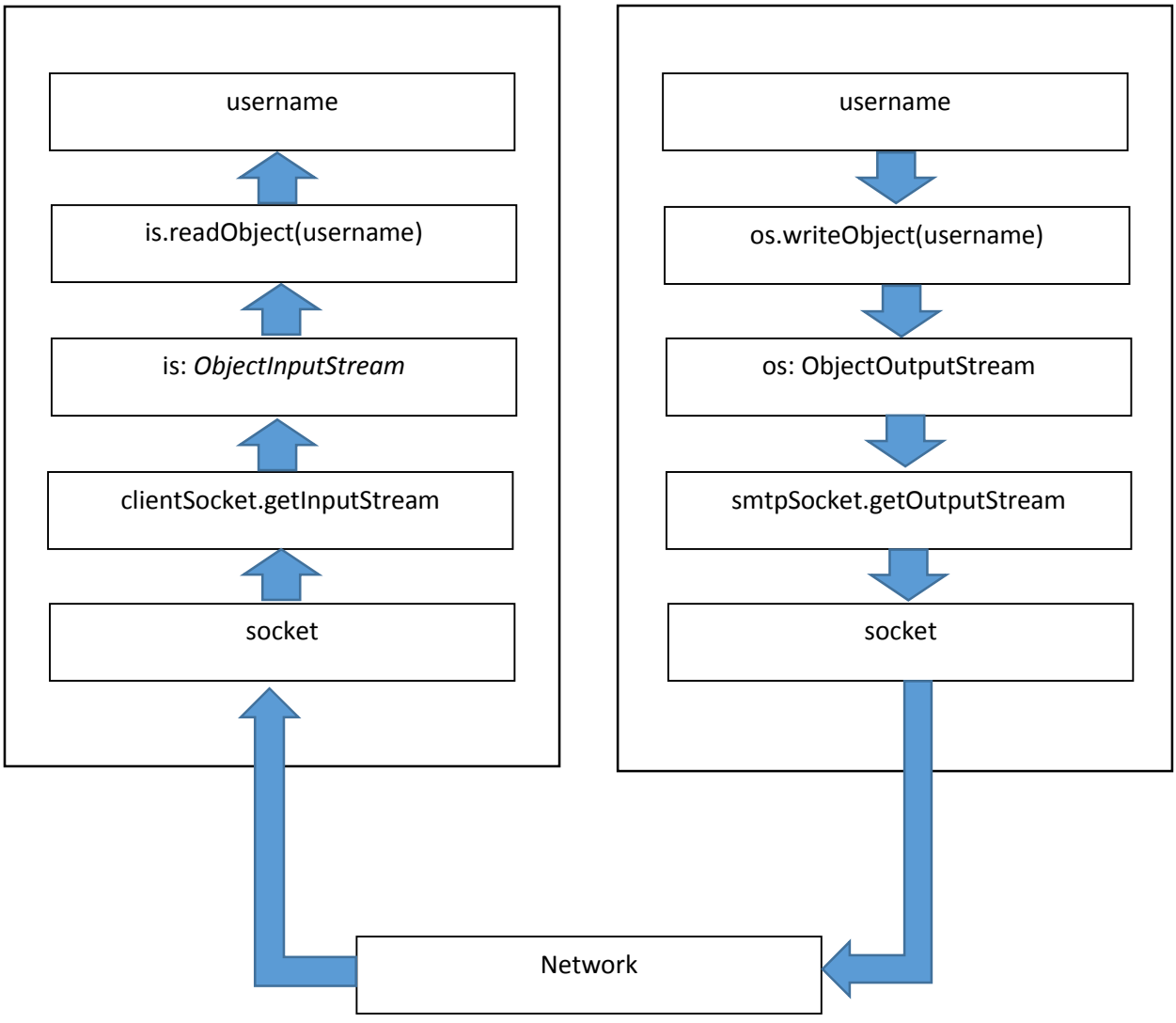
The user (on the client side) is asked whether he wants to sign up or login by the server. If the user selects signup he is asked to enter username and password. This username and password is stored in the server using a hashmap where the username is the key and the password is the value. During signup if the user enters a username that already exists the server prompts him to enter a different username.

If the user selects login, he is asked to enter his username and password which is cross checked by the server using its hashmap and if that combination of username and password does exist the user can log in. If the user types the wrong username and password the server prompts the user to retype the username and password. The user is given four chances to type the correct username and password after which the server goes into a 3 minute lockdown.

Sending username from client to server:

Server:

Client:



Target 1:

Target 0 shows us a very naïve server client remote login system with no concept of security what so ever. There were several flaws in the program implemented in target 0 to name a few:

- Plain text storage of the password in the server. Anyone can get access to the server and view all the passwords.
- The communication itself between the server and client is not very secure. Eavesdroppers can easily find out the username and the password.

In target 1 we try to make the login system more secure by tackling these flaws. First to avoid plain text storage of the passwords we need to use an appropriate cryptographic hash function. We make use of the very popularly used Secure Hash Algorithm (SHA-1). SHA-1 produces a 160-bit hash value and is typically rendered as a hexadecimal number.

```
public class SimpleSHA1
{
    private static String convertToHex(byte[] data) {
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < data.length; i++) {
            int halfbyte = (data[i] >>> 4) & 0x0F;
            int two_halfs = 0;
            do {
                if ((0 <= halfbyte) && (halfbyte <= 9))
                    buf.append((char) ('0' + halfbyte));
                else
                    buf.append((char) ('a' + (halfbyte - 10)));
                halfbyte = data[i] & 0x0F;
            } while(two_halfs++ < 1);
        }
        return buf.toString();
    }

    public static String SHA1(String text)
    throws NoSuchAlgorithmException, UnsupportedEncodingException  {
    MessageDigest md;
    md = MessageDigest.getInstance("SHA-1");
    byte[] sha1hash = new byte[40];
    md.update(text.getBytes("iso-8859-1"), 0, text.length());
    sha1hash = md.digest();
    return convertToHex(sha1hash);
    }
}
```

Now we try and solve the second problem with target 0 and that was insecure means of communication. Basically we need to encrypt the messages being exchanged between the server and client. For which we have several cryptographic tools in our arsenal. We are going with a symmetric key algorithm in target 1 which uses the same key for encryption and decryption and this requires the key to be shared between the server and the client before the communication takes place. DES was considered first but due to its short key size and its cipher text space being of the same size as the clear text space it was dropped. We are going with a symmetric key algorithm that is used worldwide which is AES. AES can be used with 128,192 or 256 bit keys. For this project we are using AES with a 128 bit key. AES uses a combination of substitution and permutation which makes it extremely secure.

```
public class AES
{
	static String IV = "AAAAAAAAAAAAAAAA";

	public static String CipherToString(byte[] cipher)
	{
		StringBuilder sb = new StringBuilder();
		for (int i = 0; i < cipher.length; i++)
			sb.append(cipher[i] + " ");
		return sb.toString();
	}

	public static byte[] encrypt(String plainText, String encryptionKey)
			throws Exception
	{
		Cipher = Cipher.getInstance("AES/CBC/PKCS5Padding", "SunJCE");
		SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF-8"),
				"AES");
		cipher.init(Cipher.ENCRYPT_MODE, key,
				new IvParameterSpec(IV.getBytes("UTF-8")));
		return cipher.doFinal(plainText.getBytes("UTF-8"));
	}

	public static String decrypt(byte[] cipherText, String encryptionKey)
			throws Exception
	{
		Cipher = Cipher.getInstance("AES/CBC/PKCS5Padding", "SunJCE");
		SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF-8"),
				"AES");
		cipher.init(Cipher.DECRYPT_MODE, key,
				new IvParameterSpec(IV.getBytes("UTF-8")));
		return new String(cipher.doFinal(cipherText), "UTF-8");
	}
}
```

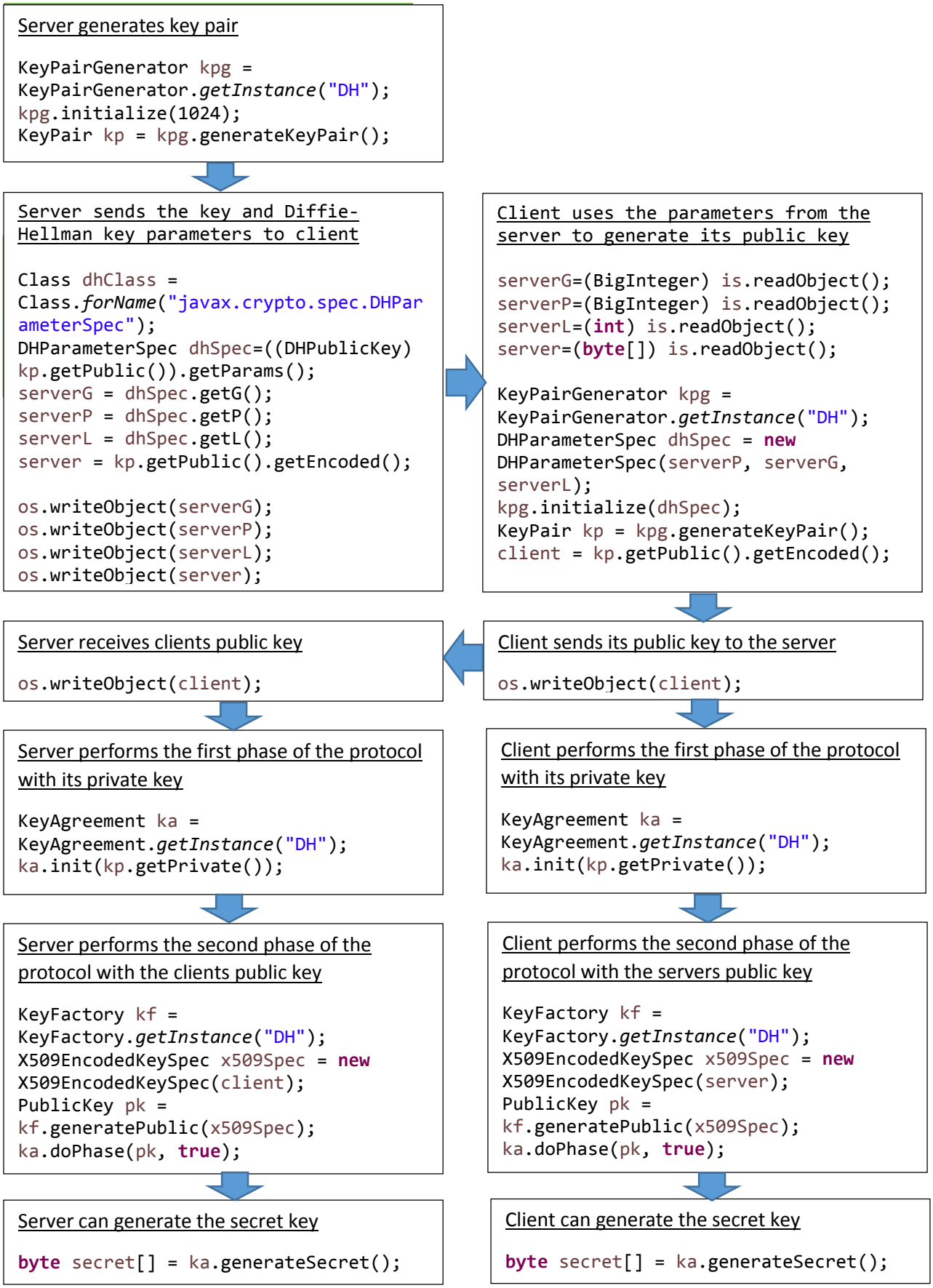
Target 2:

After completion of target 1 we end up with a pretty secure login system. But it still has few drawbacks like the key needs to be shared beforehand between the client and server. This can be done by meeting each other which sometimes just is not possible and involves a lot of work. The other way is deciding on a key over the network which is not safe at all and the key can easily fall in the wrong hands. So we need to device a secure way in which the server and client can decide on a key. There are several key agreement algorithms out there like RSA, ELGammal, Diffie-Hellman etc. For this project we will be using Diffie-Hellman. Diffie-Hellman might be less versatile than ELGammal but it's sufficiently powerful for most protocols and has better performance than RSA.

Comparison on RSA, ELGammal, Diffie-Hellman:

RSA uses discrete logarithms modulo a big prime while ELGammal and Diffie-Hellman rely on hardness of discrete logarithm on elliptic curves. They use smaller fields, so while the mathematics are a bit more complex, performance is better. RSA encryption is faster than the other two whereas decryption is slower.

Diffie-Hellman:



Vulnerabilities:

Although our server client login system has become pretty secure compared to our login system in target 0, few vulnerabilities do still exist like:

- The password entered by the user on the client system is not masked. Someone standing behind the user could read the password being typed.
- SHA-1 is used to encrypt the passwords and store it in the system. SHA-1 though secure is not secure enough, the user passwords are crack able. Something with sequential iteration count not that small like bcrypt and scrypt would make better choices.

References:

- www.wikipedia.org
- http://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html?null
- http://www.anyexample.com/programming/java/java_simple_class_to_compute_sha_1_hash.xml
- http://karanbalkar.com/2014/02/tutorial-76-implement-aes-256-encryptiondecryption-using-java/
- http://www.cs.ait.ac.th/~on/O/oreilly/java-ent/security/ch13_07.htm

The screenshots of the system are given in the following pages demonstrating various conditions like login error, sign up error, server lockdown etc. The screenshots are followed by the server and client code.

Code :(server side)

```
package experimentdh;

//www.wikipedia.org
//http://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html?null
//http://www.anyexample.com/programming/java/java_simple_class_to_compute_sha_1_hash.xml
//http://karanbalkar.com/2014/02/tutorial-76-implement-aes-256-encryptiondecryption-using-java/
//http://www.cs.ait.ac.th/~on/O/oreilly/java-ent/security/ch13_07.htm

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.math.BigInteger;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.util.HashMap;
import java.util.concurrent.TimeUnit;

import javax.crypto.KeyAgreement;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.interfaces.DHPublicKey;
import javax.crypto.spec.DHParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class New_Server
{
	public static void main(String args[]) throws Exception
	{

		byte server[],client[];
		BigInteger serverP, serverG;
		int serverL;

		//final String encryptionKey = "letstrythisonet1";

		String encoded=SimpleSHA1.SHA1("123");

		System.out.println("server running\n");
		int count =0;
		int set=0;

		HashMap<String,String> storage=new HashMap<String,String>();
		storage.put("rakshith", encoded);
		storage.put("naveen", encoded);
		storage.put("sid", encoded);
```

```
storage.put("bhargav", encoded);
storage.put("shwetha", encoded);

ServerSocket echoServer = null;
String user_name = null,pwd = null,choice;
ObjectInputStream is;
ObjectOutputStream os;
Socket clientSocket = null;

try
{
	echoServer = new ServerSocket(9999);
}
catch (IOException e)
{
	System.out.println(e);
}

try
{
	clientSocket = echoServer.accept();
	is = new ObjectInputStream(clientSocket.getInputStream());
	os = new ObjectOutputStream(clientSocket.getOutputStream());

	KeyPairGenerator kpg = KeyPairGenerator.getInstance("DH");
	kpg.initialize(1024);
	KeyPair kp = kpg.generateKeyPair();

	Class dhClass=Class.forName("javax.crypto.spec.DHParameterSpec");
	DHParameterSpec dhSpec=((DHPublicKey)kp.getPublic()).getParams();
	serverG = dhSpec.getG();
	serverP = dhSpec.getP();
	serverL = dhSpec.getL();
	server = kp.getPublic().getEncoded();

	os.writeObject(serverG);
	os.writeObject(serverP);
	os.writeObject(serverL);
	os.writeObject(server);

	client=(byte[]) is.readObject();

	KeyAgreement ka = KeyAgreement.getInstance("DH");
	ka.init(kp.getPrivate());

	KeyFactory kf = KeyFactory.getInstance("DH");
	X509EncodedKeySpec x509Spec = new X509EncodedKeySpec(client);
	PublicKey pk = kf.generatePublic(x509Spec);
	ka.doPhase(pk, true);

	byte secret[] = ka.generateSecret();
	String temp = AES.CipherToString(secret);
	temp = temp.substring(0, 16);

	final String encryptionKey =temp;
```

```
                    while(set==0)
                    {
                           os.writeObject("\n \t Enter 1 for signup\n \t Enter 2 for
login\n");
                           choice=(String) is.readObject();
                           System.out.println("choice :"+choice+"\n");

                           if(choice.equals("1"))
                           {
                                  System.out.println("entered signup\n");
                                  while(true)
                                  {
                                         os.writeObject("enter username");
                                         byte[] cipher = (byte[])is.readObject();
                                         System.out.println("server side cipher
username:"+AES.CipherToString(cipher)+"\n");
                                         user_name=AES.decrypt(cipher, encryptionKey);
                                         System.out.println("client username
:"+user_name);

                                         os.writeObject("enter password");
                                         cipher = (byte[])is.readObject();
                                         System.out.println("server side cipher
password:"+AES.CipherToString(cipher)+"\n");
                                         pwd=AES.decrypt(cipher, encryptionKey);
                                         System.out.println("client password :"+pwd);

                                         if(storage.containsKey(user_name))
                                         {
                                                os.writeObject("username already exists
please enter again\n");
                                         }
                                         else
                                         {
                                                String pwd1=SimpleSHA1.SHA1(pwd);
                                                storage.put(user_name, pwd1);
                                                System.out.println("password stored
as:"+pwd1);
                                                os.writeObject("sign up successful");
                                                break;
                                         }
                                  }
                           }
                           else if(choice.equals("2"))
                           {
                                  System.out.println("entered login");
                                  while(true)
                                  {
                                         os.writeObject("enter username");
                                         byte[] cipher =(byte[]) is.readObject();
                                         System.out.println("server side cipher
username:"+AES.CipherToString(cipher)+"\n");
                                         user_name=AES.decrypt(cipher, encryptionKey);
```

```
                                        System.out.println("client username
:"+user_name);

                                        os.writeObject("enter password");
                                        cipher = (byte[])is.readObject();
                                        System.out.println("server side cipher
password:"+AES.CipherToString(cipher)+"\n");
                                        pwd=AES.decrypt(cipher, encryptionKey);
                                        System.out.println("client password :"+pwd);

                                        String pwd1=SimpleSHA1.SHA1(pwd);

                                        if(storage.containsKey(user_name))
                                        {
                                               if(storage.get(user_name).equals(pwd1))
                                               {
                                                      os.writeObject("login
successful");

                                                      set=1;
                                                      break;
                                               }
                                               else
                                               {
                                                      count++;
                                                      if(count<4)
                                                      {
                                                             os.writeObject("password
incorrect please try again");

                                                      }
                                                      else
                                                      {

      os.writeObject("consecutive logins failed 3 minute lockdown initialised");
                                                             TimeUnit.MINUTES.sleep(3);
                                                      }
                                               }
                                        }
                                        else
                                        {
                                               count++;
                                               if(count<4)
                                               {
                                                      os.writeObject("username
incorrect please try again");

                                               }
                                               else
                                               {
                                                      os.writeObject("consecutive
logins failed 3 min lockdown initialised");

                                                      TimeUnit.MINUTES.sleep(3);
                                               }
                                        }
                                 }
                          }
                          else
```

```
                        {
                                System.out.println("wrong choice");
                        }
                }
        }
        catch (IOException e)
        {
                System.out.println(e);
        }
    }
}
```

Code:(Client side)

```
package experimentdh;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.math.BigInteger;
import java.net.Socket;
import java.net.UnknownHostException;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.util.Scanner;

import javax.crypto.KeyAgreement;
import javax.crypto.spec.DHParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class New_Client
{
    public static void main(String[] args)
    {
        byte server[],client[];
        BigInteger serverP, serverG;
        int serverL;

        //final String encryptionKey = "letstrythisonet1";

        int set=0;
        String username = null;
        String password = null;
        String choice=null;
        System.out.println("client running\n");

        Scanner input = new Scanner(System.in);
        Socket smtpSocket = null;
```

```
		ObjectOutputStream os = null;
		ObjectInputStream is = null;

		try
		{
			smtpSocket = new Socket("localhost", 9999);
			os = new ObjectOutputStream(smtpSocket.getOutputStream());
			is = new ObjectInputStream(smtpSocket.getInputStream());
		}
		catch (UnknownHostException e)
		{
			System.err.println("Don't know about host: hostname");
		}
		catch (IOException e)
		{
			System.err.println("Couldn't get I/O for the connection to:
hostname");
		}


		if (smtpSocket != null && os != null && is != null)
		{
			try {

				String responseLine2;

				serverG=(BigInteger) is.readObject();
				serverP=(BigInteger) is.readObject();
				serverL=(int) is.readObject();
				server=(byte[]) is.readObject();

				KeyPairGenerator kpg = KeyPairGenerator.getInstance("DH");
				DHParameterSpec dhSpec = new DHParameterSpec(
											serverP, serverG,
serverL);
				kpg.initialize(dhSpec);
				KeyPair kp = kpg.generateKeyPair();
				client = kp.getPublic().getEncoded();

				os.writeObject(client);

				KeyAgreement ka = KeyAgreement.getInstance("DH");
				ka.init(kp.getPrivate());

				KeyFactory kf = KeyFactory.getInstance("DH");
				X509EncodedKeySpec x509Spec =
									new X509EncodedKeySpec(server);
				PublicKey pk = kf.generatePublic(x509Spec);
				ka.doPhase(pk, true);

				byte secret[] = ka.generateSecret();

				String temp = AES.CipherToString(secret);
				temp = temp.substring(0, 16);
```

```
                        final String encryptionKey =temp;

                        while(set==0)
                        {
                               responseLine2 = (String) is.readObject();
                               System.out.println("Server: " + responseLine2);
                               choice = input.nextLine();
                               os.writeObject(choice);
                               os.flush();

                               while(true)
                               {
                                      responseLine2 = (String) is.readObject();
                                      System.out.println("Server: " +
responseLine2);
                                      username = input.nextLine();
                                      byte[] cipher=AES.encrypt(username,
encryptionKey);

                                      System.out.println("client side cipher
username:"+ AES.CipherToString(cipher)+"\n");
                                      os.writeObject(cipher);
                                      os.flush();

                                      responseLine2 = (String) is.readObject();
                                      System.out.println("Server: " +
responseLine2);
                                      username=input.nextLine();
                                      cipher=AES.encrypt(username, encryptionKey);
                                      System.out.println("client side cipher
password:"+AES.CipherToString(cipher)+"\n");
                                      os.writeObject(cipher);
                                      os.flush();

                                      responseLine2 = (String) is.readObject();
                                      System.out.println("Server: " +
responseLine2);
                                      System.out.println("");

                                      if(responseLine2.equals("sign up successful")
)
                                      {
                                             break;
                                      }
                                       if(responseLine2.equals("login successful")
)
                                      {
                                             set=1;
                                             break;
                                      }
                               }
                        }
                        os.close();
                        is.close();
                        smtpSocket.close();
```

```
                        }
                        catch (UnknownHostException e)
                        {
                                System.err.println("Trying to connect to unknown host: " +
e);
                        }
                        catch (IOException e)
                        {
                                System.err.println("IOException:  " + e);
                        }
                        catch (Exception e)
                        {
                                System.err.println("Exception:  " + e);
                        }
                }
        }
```