Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU33032-202122 Advanced Computer Networks Assignment #2: Secure-Social-App

Saumya Bahuguna, 21344349

April 17, 2022

Contents

1 Introduction

The problem description for this assignment required us to create a Social-Secure-App, where only the members of those groups will be able to decrypt those posts(or messages in my case) and all the other non members would just see a set of random string, we were advised to use cryptography(network and security) algorithms and inbuilt packages to achieve the final goal. This report will be my approach towards the assignments of how I accomplished the given tasks, I will be starting with the theoretical explanation of algorithms used by me and then will proceed with programmatic approach finally ending the report with a summary.

2 Overall Design

In the First two Sections I will define my Understanding of **Symmetric Key Cryptography,Advanced Encryption Standards(AES) & Diffie-Hellman Key Exchange**

2.1 Symmetric Key Cryptography

This is a type of cryptography based on two ends of a network using a sharing a shared key.The problem is to transmit the key between the computers(Users/Devices) on the two different ends to agree or share a key which they will use for encryption and decryption, without a third person being able to access or alter those transmissions, especially if there is no cryptosystem.

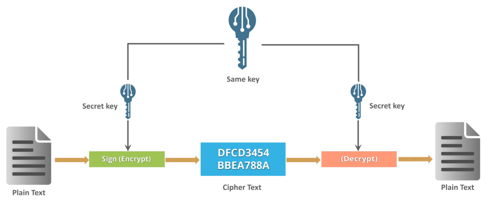

Figure 1: The figure above demonstrates a general idea of how a symmetric Key cryptography works

2.2 AES

The Advanced Encryption standard is a encryption algorithm is a symmetrical block cipher algorithm with a block length of 128 bits, it converts these blocks using keys of size 129,912 and 256 bits to encrypt them and once finished it joins them together to form cipher text. It has a substitution permutation network it consists of 1- to 14 rounds depending upon the key and block size . it takes a single key up during the the first stage and then multiplies into multiple keys used in records some properties of AES;

- it is very safe without any backdoor as in without the key encryption and decryption cannot be perform on the cyphertext.

- the large sizes of keys and blocks makes it close to impossible to break by brute force in an general idea it would take $3 \times 10^{-51} to exhaust a 256 bit key space.$

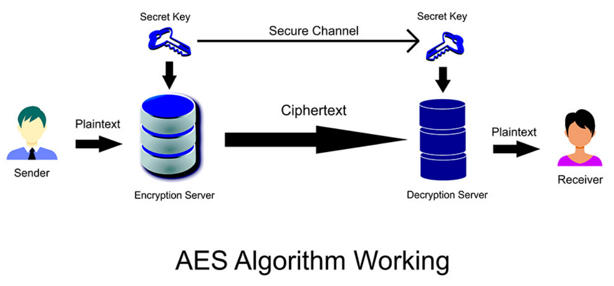

Figure 2: The figure shows a AES Encryption algorithm working

2.3 Diffie-Hellman Key Exchange

As the major problem with an Symmetric Key Algorithm is to share the key between devices and to prevent the key getting leaked at the same time. This algorithm provides a practical solution to the key distribution problem i.e enables two parties to derive a common secret key by communicating over an insecure channel, this algorithm is based ion a Elliptical curve's understanding I wont be getting into these details but rather explain the steps of how the keys are shared;

- there are two keys available or agreed upon by the two devices one.
- then there are two random private keys generated on respective devices
- using the formula x=G$^{a} mod^{p} Thegeneratedkeysusingaboveformulaaresharedbetweentwodevices$
- using the received keys and the formulae above secret symmetric keys are generates to encrypt and decrypt the messages.

As seen above using these steps above we can solve the problems of the AES key transfer problem.

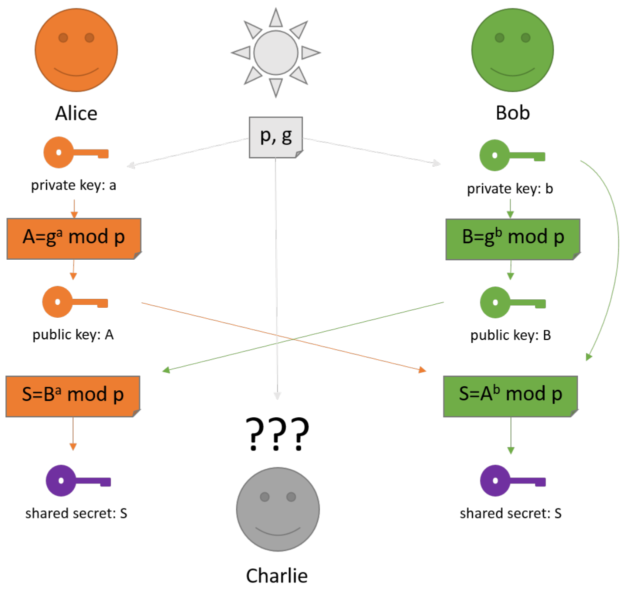

Figure 3: The figure shows a descriptive working of Diffie-Hellman Key Exchange

3 Implementation

This section describes on the implementation details of the project I have used three classes **Client-Handler, Client, Server**. The class Server as the name suggests starts the server on a server socket of the sockets package to listen constantly on a local host and a port. The client class as the name suggest behaves as a client (members of the group) . The client handler class is called by the server to handle all the client requests and messages form the server side.

- 0: the P and G values form the Diffie-Hellman Key Exchange are set by default on both server and client class
- step 1: Starts the client by setting up a username for the client
- step 2: the username received by the server is stored and an exchange of generated keys happens between the client and server.
- The generated public shared keys are now processed again to create secret symmetric keys and are stored in Hash map along with each clients username;
- whenever a message is received by the server it decrypts the code received from one client using the keys stored in the Hash Map and then encrypted again to transfer to all respective group members as per their generated secret symmetric keys.

3.1 Client Handler

As described above this class handles all the sockets sending messages to the server, as soon as a message is received the client handler performs the DHKE algorithm and generates a secret key using the code shown in the 1.

```
     broadcastMessage("SERVER: " +
      clientUsername + " has entered the
      chat!");
      B = ((Math.pow(g, privKey)) % pub);
            bstring=Double.toString(B);
         this.bufferedWriter.write("/"+bstring);
            this.bufferedWriter.newLine();
         this.bufferedWriter.flush();
             String input=bufferedReader.readLine();
             System.out.println(input);
             Astring=input.split("/")[1];
             A=Double.parseDouble(Astring);
             BDash = ((Math.pow(A, privKey)) % pub);
             System.out.println("Symmetric Key:  "+BDash);
             clientHandlers.put(this,BDash);
         } catch (IOException e) {
             // Close everything more gracefully.
             closeEverything(socket, bufferedReader, bufferedWriter);
         }
    }
```

Listing 1: The class handles all the requests to the server and genrates a secret key and stores it

The client class performs the same operation with the server to generate the private key.

3.2 Encryption

This function encryptes the message send to the clients and by the clients using the AES Algorithm and the key generated by the above algorithm
the ??lso shows teh set key function which sets the key using the Secret key package of the java library form the string value generted.

```
public static String encrypt(final String strToEncrypt, final String secret) {
        try {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return Base64.getEncoder()
                    .encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
        } catch (Exception e) {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

     public static void setKey(final String myKey) {
        MessageDigest s = null;
        try {
            key = myKey.getBytes("UTF-8");
            s = MessageDigest.getInstance("SHA-1");
            key = s.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        } catch (Exception e) {
            ((Throwable) e).printStackTrace();
        }
    }
```

Listing 2: The snippet above shows how the the class handles if the content is an Image

3.3 Decryption

The code shown in ?? decryptes the code using the secret key stored as the keys are same for both Server, client therefore the decryptiona nd encryption of files happens accurately.

```
public static String decrypt(final String strToDecrypt, final String secret) {
        try {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");

                        cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal(Base64.getDecoder()
                    .decode(strToDecrypt)));
        } catch (Exception e) {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }
```

Listing 3: The snippet shows the code for decryption

4 Reflection

To look back to reflect what benefits this assignment had, I could have most certainly improved in this project by having a multi group page which was not too difficult to achieve but because of time constraints I couldn't. all I would have needed is a admin class to manage all the groups. other than that I really enjoyed working on the project. and learnt a lot. This project gave me so much information about cryptography and while performing different algorithms and choosing algorithms to use gave me insights on how a secure connection can be achieved.

5 references

Understaning Crytography

6 Appendix

```
import java.io.*;
import java.net.Socket;
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.MessageDigest;
import java.security.PrivateKey;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.HashMap;
import java.util.Random;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.spec.GCMParameterSpec;
/**

public class ClientHandler implements Runnable {


    public static HashMap<ClientHandler,Double>
    clientHandlers = new HashMap<>();
    // Id that will increment with each new client.
    int pub = 23;
    int g =   9          ;
    and sending data respectively.
    private Socket socket;
    private BufferedReader bufferedReader;
    private BufferedWriter bufferedWriter;
    private static byte[] key;
    private final int KEY_SIZE = 128;
    private final int DATA_LENGTH = 128;
    private Cipher encryptionCipher;
    private String clientUsername;
    static Key secretKey;
    double B,A,BDash;
    String bstring ,Astring;
    Random rn = new Random();

    public ClientHandler(Socket socket) {
        try {
            this.socket = socket;
            int privKey=0;
                        try {
                                privKey = rn.nextInt(12 - 1 + 1) + 4;

                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
   System.out.println(privKey);

      this.bufferedReader = new BufferedReader(new
     InputStreamReader(socket.getInputStream()));
   this.bufferedWriter= new BufferedWriter(new
   OutputStreamWriter(socket.getOutputStream()));

   this.clientUsername = bufferedReader.readLine();

            broadcastMessage("SERVER: " +
            clientUsername + " has entered the chat!");
            B = ((Math.pow(g, privKey)) % pub);
            bstring=Double.toString(B);
            this.bufferedWriter.write("/"+bstring);
            this.bufferedWriter.newLine();
            this.bufferedWriter.flush();
            String input=bufferedReader.readLine();
            System.out.println(input);
            Astring=input.split("/")[1];
            A=Double.parseDouble(Astring);
            BDash = ((Math.pow(A, privKey)) % pub);
            System.out.println("Symmetric Key:  "+BDash);
            clientHandlers.put(this,BDash);
        } catch (IOException e) {
            // Close everything more gracefully.
            closeEverything(socket,
            bufferedReader, bufferedWriter);
        }
    }

    ClientHandler check(String user) {
        for (ClientHandler clientHandler :
        clientHandlers.keySet()) {

                System.out.println("name:
                "+clientHandler.clientUsername);
                if (clientHandler.clientUsername.
                equals(clientUsername)) {
                    System.out.println("name:
            "+clientHandler.clientUsername);
```

```
                        return(clientHandler);


        }
        return (null);    }
    private byte[] decode(String data) {
        return Base64.getDecoder().decode(data);
    }
    public static String encrypt(final String
    strToEncrypt, final
    String secret) {
        try {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return Base64.getEncoder()
    .encodeToString(cipher.doFinal
    (strToEncrypt.getBytes("UTF-8")));
        } catch (Exception e) {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }
    private String encode(byte[] data) {
        return Base64.getEncoder().encodeToString(data);
    }
    public static String
    decrypt(final String strToDecrypt,
    final String secret) {
        try {
            setKey(secret);
            Cipher cipher =
            Cipher.getInstance("AES/ECB/PKCS5PADDING");

                        cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal
            (Base64.getDecoder()
            .decode(strToDecrypt)));
        } catch (Exception e) {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }
    public static void setKey(final String myKey) {
        MessageDigest s = null;
        try {
            key = myKey.getBytes("UTF-8");
            s = MessageDigest.getInstance("SHA-1");
            key = s.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        } catch (Exception e) {
            ((Throwable) e).printStackTrace();
        }
    }
    @Override
    public void run() {
        String messageFromClient;
        String Decrypted;
       client is still established.
        while (socket.isConnected()) {

                try {

                messageFromClient = bufferedReader.readLine();
     System.out.println("before decryption from client :
    "+messageFromClient);
                Decrypted=decrypt(messageFromClient,
                Double.toString(clientHandlers.get
                (check(clientUsername))));

                System.out.println("After Decrption "
                + Decrypted);

                broadcastMessage(Decrypted);
                }
                catch (Exception e) {


closeEverything(socket, bufferedReader,
                bufferedWriter);
                break;
            }
        }
    }



    public void broadcastMessage
    (String messageToSend) {
        String encrypted="";
      System.out.println("before encryption:"+
      messageToSend);
    for (ClientHandler clientHandler :
    clientHandlers.keySet()) {

                try {
     System.out.println(clientHandler.
     clientUsername);

if(!clientHandler.clientUsername.equals
 (clientUsername)
    {
        try {
        encrypted=encrypt(messageToSend,Double.toStrin
    g(clientHandlers.get(clientHandler)));
                    }
                  catch(Exception e)
                  {
                  e.printStackTrace();
```

```
                    }
    System.out.println("sending a message:
    "+encrypted);


     clientHandler.bufferedWriter.write(encrypted);
clientHandler.bufferedWriter.newLine();
clientHandler.bufferedWriter.flush();
                    }
          } catch (IOException e) {
                    // Gracefully close everything.
                    closeEverything(socket, bufferedReader, bufferedWriter);
               }
          }
}


public void removeClientHandler() {
       clientHandlers.remove(this);
broadcastMessage("SERVER: " + clientUsername
  + " has left the chat!");
     }


     public void closeEverything(Socket socket,
     BufferedReader bufferedReader,
     BufferedWriter bufferedWriter) {

          removeClientHandler();
          try {
               if (bufferedReader != null) {
                    bufferedReader.close();
               }
               if (bufferedWriter != null) {
                    bufferedWriter.close();
               }
               if (socket != null) {
                    socket.close();
               }
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
}
```

Listing 4: ClientHandler.java

```
//package main;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    private final ServerSocket serverSocket;

    public Server(ServerSocket serverSocket) {
        this.serverSocket = serverSocket;
    }

    public void startServer() {
        try {

            while (!serverSocket.isClosed()) {
 Socket socket = serverSocket.accept();
  System.out.println("A␣new␣client␣has␣connected!");
ClientHandler clientHandler = new
  ClientHandler(socket);
  Thread thread = new Thread(clientHandler);

                thread.start();
            }
        } catch (IOException e) {
            closeServerSocket();
        }
    }

    // Close the server socket gracefully.
    public void closeServerSocket() {
        try {
            if (serverSocket != null) {
                serverSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Run the program.
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(1234);
        Server server = new Server(serverSocket);
        server.startServer();
    }

}
```

Listing 5: Server.java

```
//package main;
import java.io.*;
import java.net.Socket;
import java.security.Key;
import java.security.MessageDigest;
import java.util.Arrays;
import java.util.Base64;
import java.util.Random;
import java.util.Scanner;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;



public class Client {

    private static  Key secretKey = null;

    private Socket socket;

    private BufferedReader bufferedReader;
    private BufferedWriter bufferedWriter;
    private static byte[] key;
    private final int KEY_SIZE = 128;
    private final int DATA_LENGTH = 128;
    private Cipher encryptionCipher;
    private String username;
int pub = 23;
int g = 9 ;
Double A,B,Adash;
String Astring ,BString;
int privKey=0;
boolean check=false;
Random rn= new Random();
public static String encrypt(final String strToEncrypt, final String secret) {
    try {
        setKey(secret);
        Cipher cipher = Cipher.getInstance
        ("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        return Base64.getEncoder()
   .encodeToString(cipher.doFinal
   (strToEncrypt.getBytes("U
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣TF-8")));
    } catch (Exception e) {
        System.out.println("Error␣while␣encrypting:␣" + e.toString());
    }
    return null;
}
private String encode(byte[] data) {
    return Base64.getEncoder().encodeToString(data);
}
public static String decrypt(final String
strToDecrypt, final String
secret) {
    try {
        setKey(secret);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");

                cipher.init(Cipher.DECRYPT_MODE, secretKey);
        return new String(cipher.doFinal(Base64.getDecoder()
                .decode(strToDecrypt)));
    } catch (Exception e) {
        System.out.println("Error while decrypting: " + e.toString());
    }
    return null;
}
public static void setKey(final String myKey) {
    MessageDigest s = null;
    try {
        key = myKey.getBytes("UTF-8");
        s = MessageDigest.getInstance("SHA-1");
        key = s.digest(key);
        key = Arrays.copyOf(key, 16);
        secretKey = new SecretKeySpec(key, "AES");
    } catch (Exception e) {
        ((Throwable) e).printStackTrace();
    }
}
    public Client(Socket socket, String username) {
        try {
            this.socket = socket;
            this.username = username;

            try {
        privKey = rn.nextInt(8 - 4 + 1) + 4;
                        } catch (Exception e) {

                e.printStackTrace();
                        }
          A = ((Math.pow(g, privKey)) % pub);
           Astring=Double.toString(A);

        this.bufferedReader = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));

            this.bufferedWriter= new BufferedWriter(new
            OutputStreamWriter(socket.getOutputStream()));

        } catch (IOException e) {
            // Gracefully close everything.
            closeEverything(socket, bufferedReader,
            bufferedWriter);
        }
    }
public void createkey() {
        try {
        BString=bufferedReader.readLine().split("/")[1];
          bufferedWriter.write("/"+Astring);
      bufferedWriter.newLine();
      bufferedWriter.flush();

      B=Double.parseDouble(BString);
      Adash = ((Math.pow(B, privKey)) % pub);
      System.out.println("Symmetric␣Key:␣␣"+Adash);
}catch(IOException e) {e.printStackTrace();}
}

    public void sendMessage() {
        try {
            // Initially send the username of the client.
            bufferedWriter.write(username);
            bufferedWriter.newLine();
            bufferedWriter.flush();

            // Create a scanner for user input.
            @SuppressWarnings("resource")
                        Scanner scanner = new Scanner(System.in);
            while (socket.isConnected()) {
                String messageToSend = scanner.nextLine();

                try{messageToSend=encrypt(username+":
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"+messageToSend,Double.toString(Adash));}
                catch(Exceptio
                n e) {e.printStackTrace();}
                System.out.println("sent␣text:" +messageToSend);
                bufferedWriter.write(messageToSend);
                bufferedWriter.newLine();
                bufferedWriter.flush();
            }
        } catch (IOException e) {
            // Gracefully close everything.
            closeEverything(socket, bufferedReader,
            bufferedWriter);
        }
    }


    public void listenForMessage() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                String msgFromGroupChat;
      while (socket.isConnected()) {
                   try {
                         if(check==false) {
                             createkey();
                             check=true;
                         }
                         else {
        msgFromGroupChat = bufferedReader.readLine();
              System.out.println("encrypted:
␣␣␣␣␣␣␣␣␣␣␣␣"+msgFromGroupChat);
                       try {
```

```
                    msgFromGroupChat=decrypt(msgFromGroupChat,
                    Double.to
                 String(Adash));
                          }
                          catch(Exception e) {
                                  e.printStackTrace();
                          }
System.out.println("decrypted␣:"+msgFromGroupChat);
                            }} catch (IOException e) {
                      closeEverything(socket,
                       bufferedReader,
                       bufferedWriter);
                  }
              }
          }
      }).start();
  }


  public void closeEverything(Socket socket, BufferedReader
   bufferedReader, BufferedWriter bufferedWriter) {

      try {
          if (bufferedReader != null) {
              bufferedReader.close();
          }
          if (bufferedWriter != null) {
              bufferedWriter.close();
          }
          if (socket != null) {
              socket.close();
          }
      } catch (IOException e) {
          e.printStackTrace();
      }
  }


  public static void main(String[] args) throws IOException {

      Scanner scanner = new Scanner(System.in);
 System.out.print("Enter␣your␣username␣for␣the␣group␣chat:␣");
 String username = scanner.nextLine();

  Socket socket = new Socket("localhost", 1234);


 Client client = new Client(socket, username);

client.listenForMessage();
 client.sendMessage();
  }
}
```

Listing 6: Client.java