Lab 1: SKIP Server

March 2nd, 2023

William Pearce

DFSC 3316

MWF

“A” Option

**Skip.java**

This file was borrowed from the notes and is used in both client side and server side code.

import java.math.BigInteger;

import javax.crypto.spec.\*;

public class Skip {

// http://skip.incog.com/spec/numbers.html

// Simple Key Management for Internet Protocols – SKIP.

// Using DH (Diffie-Hellman standard). 1024 DH parameter defined by SKIP. First

// 79 bytes of ASCII

// representation of a quote by Gandhi. "Whatever you do is insignificant, but

// it is very important that

// you do it." 512, 1024, and 2048 bit modulus parameters are supported. The

// resulting keys are

// the length of the modulus, i.e., 512, 1024, or 2048 bits.

private static final String skip1024String =

"F488FD584E49DBCD" + "20B49DE49107366B" + "336C380D451D0F7C" + "88B31C7C5B2D8EF6" +

"F3C923C043F0A55B" + "188D8EBB558CB85D" + "38D334FD7C175743" + "A31D186CDE33212C" +

"B52AFF3CE1B12940" + "18118D7C84A70A72" + "D686C40319C80729" + "7ACA950CD9969FAB" +

"D00A509B0246D308" + "3D66A45D419F9C7C" + "BD894B221926BAAB" + "A25EC355E92F78C7";

// Create modulus from string => “p”

private static final BigInteger skip1024Modulus = new BigInteger(skip1024String, 16);

// Base => “g”

private static final BigInteger skip1024Base = BigInteger.valueOf(2);

// DH parameter specification

public static final DHParameterSpec sDHParameterSpec = new DHParameterSpec(skip1024Modulus, skip1024Base);

}

**Server-side:**

The server side is designed to run multi-threaded to serve multiple clients simultaneously. The main thread listens for new connections and executes a new thread for each new connection established. Shown below is the User object used to store and authenticate users.

**User.java**

import java.security.\*;

import java.util.Arrays;

public class User {

private String username;

private byte[] passwordHash;

public User() {

username = null;

passwordHash = null;

}

public void setName(String username) {

this.username = username;

}

public String getName() {

return username;

}

public void setPassword(String password) {

try {

MessageDigest md = MessageDigest.getInstance("SHA-256");

byte[] hash = md.digest(password.getBytes());

passwordHash = hash;

} catch(NoSuchAlgorithmException e) {

e.printStackTrace();

}

}

public Boolean validLogin(String name, String password) {

MessageDigest md;

try {

md = MessageDigest.getInstance("SHA-256");

byte[] hash = md.digest(password.getBytes());

return (Arrays.equals(hash, passwordHash) &&

username.equals(name));

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

return false;

}

public String toString() {

return "username: " + username + " - password: " + passwordHash;

}

}

Shown below is the main thread which listens for new client connections.

**Server.java**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

import java.util.ArrayList;

import java.security.\*;

class Server {

private static final int PORT = 7791;

private static KeyPair keyPair = null;

private static ArrayList<User> users = new ArrayList<>();

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

populateUsers();

System.out.println("Generating Diffie-Hellman key pair...");

KeyPairGenerator kpg = KeyPairGenerator.getInstance("DH");

kpg.initialize(Skip.sDHParameterSpec);

keyPair = kpg.genKeyPair();

try (ServerSocket server = new ServerSocket(PORT)) {

System.out.println("Server opened at " + InetAddress.getLocalHost());

System.out.println("Awaiting client connections...");

while(true) {

Socket client = server.accept();

System.out.println("Client connected: " + client.toString());

new ServerThread(client).start(); // send incoming client to new thread

}

}

}

public static KeyPair getKeyPair() {

// returns DH Key Pair

return keyPair;

}

public static void populateUsers() throws FileNotFoundException {

// read user data from file and load into memory

File file = new File("users.db");

Scanner input = new Scanner(file, "utf-8");

for(int i = 0; i < 10; i++) {

users.add(new User());

users.get(i).setName(input.nextLine());

users.get(i).setPassword(input.nextLine());

}

input.close();

}

public static ArrayList<User> getUsers() {

return users;

}

public static User getUser(String name, String password) {

// select user from table

for(int i = 0; i < users.size(); i++) {

if(users.get(i).validLogin(name, password))

return users.get(i);

}

return null;

}

}

Server thread opened upon new client connections. In this code is the Diffie-Hellman exchange, and based on client input, the encryption algorithm is selected and a key generated. Based on client input, the thread will then either send or receive a file with the client using the encryption algorithm.

**ServerThread.java**

// Server thread class

// Clients are accepted by the main thread and a new thread is

// opened on the server to handle each client

import java.util.Base64;

import java.net.\*;

import java.io.\*;

import java.security.\*;

import java.security.spec.\*;

import javax.crypto.\*;

import javax.crypto.spec.\*;

public class ServerThread extends Thread {

private Socket client;

private User user;

private DataInputStream dataIn;

private DataOutputStream dataOut;

private byte[] secret;

private SecretKey sessionKey;

private String algorithm;

ServerThread(Socket client) throws IOException {

this.client = client;

dataIn = new DataInputStream(client.getInputStream());

dataOut = new DataOutputStream(client.getOutputStream());

}

public void run() {

try {

// init io

BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));

PrintWriter out = new PrintWriter(client.getOutputStream());

// get algorithm choice from client

algorithm = in.readLine();

// login

String userIn = in.readLine();

String passIn = in.readLine();

user = Server.getUser(userIn, passIn);

if(user == null) {

out.println("Login Failed");

client.close();

}

out.println("Login Successful");

// init keys

try {

// Diffie-Hellman

System.out.println("Receiving public key...");

byte[] keyBytes = new byte[dataIn.readInt()];

dataIn.readFully(keyBytes);

KeyFactory kf = KeyFactory.getInstance("DH");

X509EncodedKeySpec x509Spec = new X509EncodedKeySpec(keyBytes);

PublicKey clientPub = kf.generatePublic(x509Spec);

System.out.println("Sending public key...");

keyBytes = Server.getKeyPair().getPublic().getEncoded();

dataOut.writeInt(keyBytes.length);

dataOut.write(keyBytes);

System.out.println("Generating session key...");

KeyAgreement ka = KeyAgreement.getInstance("DH");

ka.init(Server.getKeyPair().getPrivate());

ka.doPhase(clientPub, true);

secret = ka.generateSecret();

System.out.println("Session secret generated.");

System.out.println(Base64.getEncoder().encodeToString(secret));

// convert key for selected encryption algorithm

KeySpec keyspec = null;

SecretKeyFactory keyfactory = null;

switch(algorithm) {

case "AES":

sessionKey = new SecretKeySpec(secret, 0, 32, "AES");

break;

case "DES":

keyspec = new SecretKeySpec(secret, "DES");

keyfactory = SecretKeyFactory.getInstance("DES");

sessionKey = keyfactory.generateSecret(keyspec);

break;

case "DESede":

keyspec = new SecretKeySpec(secret, "DESede");

keyfactory = SecretKeyFactory.getInstance("DESede");

sessionKey = keyfactory.generateSecret(keyspec);

break;

}

System.out.println("Session Key generated.");

} catch(Exception e) {

System.out.println(e);

}

// receive operation choice from client

String option = in.readLine();

switch(option) {

case "send":

get();

break;

case "get":

send();

break;

default:

System.out.println("error");

break;

}

} catch(Exception e) {

e.printStackTrace();

}

}

public void send() {

try {

// preset server file to send

FileInputStream fileIn = new FileInputStream("serverfile.flag");

// init cipher

Cipher cipher = Cipher.getInstance(algorithm + "/CBC/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, sessionKey);

byte[] iv = cipher.getIV();

dataOut.writeInt(iv.length);

dataOut.write(iv);

byte[] input = new byte[64];

while (true) {

int bytesRead = fileIn.read(input);

if (bytesRead == -1)

break;

byte[] output = cipher.update(input, 0, bytesRead);

if (output != null)

dataOut.write(output);

}

byte[] output = cipher.doFinal();

if (output != null)

dataOut.write(output);

fileIn.close();

dataOut.close();

dataIn.close();

} catch(Exception e) {

e.printStackTrace();

}

}

public void get() {

try {

FileOutputStream fileOut = new FileOutputStream("output" + user.getName() + ".txt");

int ivSize = dataIn.readInt();

byte[] iv = new byte[ivSize];

dataIn.readFully(iv);

IvParameterSpec ivps = new IvParameterSpec(iv);

// init cipher

Cipher cipher = Cipher.getInstance(algorithm + "/CBC/PKCS5Padding");

cipher.init(Cipher.DECRYPT\_MODE, sessionKey, ivps);

byte[] input = new byte[64];

while (true) {

int bytesRead = dataIn.read(input);

if (bytesRead == -1)

break;

byte[] output = cipher.update(input, 0, bytesRead);

if (output != null) {

fileOut.write(output);

System.out.print(new String(output));

}

}

byte[] output = cipher.doFinal();

if (output != null) {

fileOut.write(output);

System.out.print(new String(output));

}

fileOut.close();

} catch(Exception e) {

e.printStackTrace();

}

}

}

Below is the file read by the server to populate the user table with usernames and password hashes.

**users.db**

alice1

password1

bob2

password2

chuck3

password3

dennis4

password4

user5

password5

user6

password6

user7

password7

user8

password8

user9

password9

userx

password

Below is a test file to be sent by the server upon request from the client.

**serverfile.flag**

SERVER File

you issued a get request and you got this server file.

this file has nothing cool in it.

**Client-side:**

The client side takes command line arguments and makes requests to the server based on selections from the user.

**Client.java**

import java.net.\*;

import java.io.\*;

import java.security.\*;

import java.security.spec.\*;

import javax.crypto.\*;

import javax.crypto.spec.\*;

import java.util.Base64;

import java.util.Scanner;

public class Client {

private static final int PORT = 7791;

private static Socket server;

private static SecretKey sessionKey;

private static String algorithm;

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

String option = args[1];

algorithm = args[3];

// initialize connections and data streams

server = new Socket(InetAddress.getByName(args[0]), PORT);

PrintWriter out = new PrintWriter(server.getOutputStream(), true);

Scanner userIn = new Scanner(System.in);

out.println(algorithm); // send choice of algorithm to server

// login

System.out.println("Please Login");

System.out.println("Username: ");

out.println(userIn.nextLine());

System.out.println("Password: ");

out.println(userIn.nextLine());

userIn.close();

// Diffie-Hellman exchange

System.out.println("Generating Diffie-Hellman key pair...");

KeyPairGenerator kpg = KeyPairGenerator.getInstance("DH");

kpg.initialize(Skip.sDHParameterSpec);

KeyPair keyPair = kpg.genKeyPair();

System.out.println("Sending public key...");

byte[] keyBytes = keyPair.getPublic().getEncoded();

DataOutputStream dataOut = new DataOutputStream(server.getOutputStream());

dataOut.writeInt(keyBytes.length);

dataOut.write(keyBytes);

System.out.println("Receiving public key...");

DataInputStream dataIn = new DataInputStream(server.getInputStream());

keyBytes = new byte[dataIn.readInt()];

dataIn.readFully(keyBytes);

KeyFactory kf = KeyFactory.getInstance("DH");

X509EncodedKeySpec x509Spec = new X509EncodedKeySpec(keyBytes);

PublicKey serverPub = kf.generatePublic(x509Spec);

System.out.println("Generating session key...");

KeyAgreement ka = KeyAgreement.getInstance("DH");

ka.init(keyPair.getPrivate());

ka.doPhase(serverPub, true);

byte[] secret = ka.generateSecret();

System.out.println("Session key generated.");

System.out.println(Base64.getEncoder().encodeToString(secret));

// convert key based on user choice of algorithm

KeySpec keyspec = null;

SecretKeyFactory keyfactory = null;

switch(algorithm) {

case "AES":

sessionKey = new SecretKeySpec(secret, 0, 32, "AES");

break;

case "DES":

keyspec = new SecretKeySpec(secret, "DES");

keyfactory = SecretKeyFactory.getInstance("DES");

sessionKey = keyfactory.generateSecret(keyspec);

break;

case "DESede":

keyspec = new SecretKeySpec(secret, "DESede");

keyfactory = SecretKeyFactory.getInstance("DESede");

sessionKey = keyfactory.generateSecret(keyspec);

break;

}

System.out.println("Session Key generated.");

// send or receive file based on user choice

switch(option) {

case "send":

out.println("send"); // communicate choice to server

send(args[2]);

break;

case "get":

out.println("get");

get(args[2]);

break;

default:

System.out.println("Invalid Command: type 'send' or 'get'");

}

}

public static void send(String filename) throws Exception {

FileInputStream fileIn = new FileInputStream(filename);

DataOutputStream dataOut = new DataOutputStream(server.getOutputStream());

// initialize cipher

Cipher cipher = Cipher.getInstance(algorithm + "/CBC/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, sessionKey);

byte[] iv = cipher.getIV();

dataOut.writeInt(iv.length);

dataOut.write(iv);

byte[] input = new byte[64]; // Encrypt 64 byte blocks

while (true) {

int bytesRead = fileIn.read(input);

if (bytesRead == -1)

break; // Check EOF.

byte[] output = cipher.update(input, 0, bytesRead);

if (output != null)

dataOut.write(output); // Write encrypted info to server.

}

byte[] output = cipher.doFinal();

if (output != null)

dataOut.write(output); // Write remaining to client.

dataOut.close();

fileIn.close();

}

public static void get(String filename) throws Exception {

FileOutputStream fileOut = new FileOutputStream(filename);

DataInputStream dataIn = new DataInputStream(server.getInputStream());

// Read the initialization vector.

int ivSize = dataIn.readInt();

byte[] iv = new byte[ivSize];

dataIn.readFully(iv);

IvParameterSpec ivps = new IvParameterSpec(iv);

// initialize cipher

Cipher des = Cipher.getInstance(algorithm + "/CBC/PKCS5Padding");

des.init(Cipher.DECRYPT\_MODE, sessionKey, ivps);

byte[] input = new byte[64];

while (true) {

int bytesRead = dataIn.read(input);

if (bytesRead == -1)

break;

byte[] output = des.update(input, 0, bytesRead);

if (output != null) {

fileOut.write(output);

System.out.print(new String(output));

}

}

byte[] output = des.doFinal();

if (output != null) {

fileOut.write(output);

System.out.print(new String(output));

}

fileOut.close();

dataIn.close();

}

}

Below is a test file to be sent from the client to the server.

**send.test**

test file for client to send

if this file appears serverside, the transfer was successful

**Server Terminal:**

PS C:\Users\billp\Documents\GitHub\Crypto-Lab1\server> java Server

Generating Diffie-Hellman key pair...

Server opened at DESKTOP-1E3JPTH/10.14.0.8

Awaiting client connections...

Client connected: Socket[addr=/127.0.0.1,port=59220,localport=7791]

Receiving public key...

Sending public key...

Generating session key...

Session secret generated.

ar2HIMrBPY8lHBegUmrWNbYW1WPLZpHetFyBqw5nPI/FOW5ITpmMcbFMktoUasU7G7c5sgSThSx+gA5mEAJetexvIL6K3ETKv9PcnGoWpSrvtU3NQRe/pwjKOgJsS/Ag1tPGRCOEgyOlKa50wijc1lKojBbAxuSX1YMrWMB/0SY=

Session Key generated.

test file for client to send

if this file appears serverside, the transfer was successful

**Client Terminal:**

PS C:\Users\billp\Documents\GitHub\Crypto-Lab1\client> java Client localhost send send.test AES

Please Login

Username:

alice1

Password:

password1

Generating Diffie-Hellman key pair...

Sending public key...

Receiving public key...

Generating session key...

Session key generated.

ar2HIMrBPY8lHBegUmrWNbYW1WPLZpHetFyBqw5nPI/FOW5ITpmMcbFMktoUasU7G7c5sgSThSx+gA5mEAJetexvIL6K3ETKv9PcnGoWpSrvtU3NQRe/pwjKOgJsS/Ag1tPGRCOEgyOlKa50wijc1lKojBbAxuSX1YMrWMB/0SY=

Session Key generated.

**Multi-threaded functionality may be demonstrated in person.**