Ryerson University

Department of Electrical, Computer & Biomedical Engineering
Faculty of Engineering & Architectural Science

Department of Electrical, Computer, and Biomedical Engineering

Course Number	**COE817**
Course Title	**Network Security**
Semester/Year	**Winter 2021**
Instructor	**Bahauddin Kazi**

Final Project

Project Title	**Digital Certified Mail**
Group Number	**07**

Submission Date	April 11th,2021
Due Date	April 11th,2021

Name	Student ID	Signature*
Duanwei Zhang (Section 03)	***24903***	**D.Z**
Haoran Zhou (Section 02)	***02305***	**H.Z**
Lai Shan Nixon Mark (Section 03)	**44772**	**N.M**
Yu Zhang (Section 01)	**41786**	**Y.Z**

(Note: remove the first 4 digits from your student ID)

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:*
www.ryerson.ca/senate/current/pol60.pdf.

Introduction

This project discusses the implementation of the digitally certified mail scheme. The goal of the project is to design and implement a full digital certified mail scheme where it has to fulfill two main objectives, it has to require that a recipient sign for a message prior to receiving it to acknowledge both sender and receiver, and also it has to stop the sender forging a receipt. The project solution includes several exchanges of keys to make sure that when the sender sends a valid receipt, the recipient will also receive a decrypted version of the message. In addition, Both sides need to generate random DES keys for encryption and encrypt receipts with one bogus message that is to be sent by the Sender. Oblivious transfer protocol will be used on both sides for transferring the keys to each other. The project was built using Java through Netbean 8.2 for its implementation.

Oblivious Transfer Protocol(OT) is a mechanism for a receiver that wants to get a specific text or message from a sender without letting the sender know its selection, and the sender will send all encrypted messages based on the protocol oblivious to the receiver and it gets as it wants. The reason behind it is the key security characteristic comes from the fact that the receiver and sender will not have to specify selection and keys, but they can still exchange what they want to transfer. As the result, oblivious transfer is symmetric, it can construct efficient $OT\ {}^{n}_{1}$ and $OT\ {}^{n}_{k}$ from $OT\ {}^{2}_{1}$. In this project, $OT\ {}^{2}_{1}$ will be the right approach for setting up the foundation of transferring messages. Further discussion will be discussed in the design and implementation parts.

Partial Secrets Exchange Subprotocol(PSE) will also be used for the transaction protocols for preventing the sender from forge a receipt. There will be two parties to the subprotocol and it will be called A and B. The proposal is that A holds 2n secrets that are all recognizable by B when B also holds 2n secrets that are all recognizable by A. The secrets protocol are assumed to be binary strings of length l. The secrets of both sides are subdivided into pairs. A [B] will constructively know one of B's [A's] pairs if there exists an i, $1 \leq i \leq n$, so that A [B] can constructively compute both b_i and b_{n+i} and vice versa B[A] (a_i and a_{n+i}). Exchanging effective knowledge of any one pair of secrets is the goal of this specific subprotocol. This subprotocol will be referred to as the Partial Secret Exchange (PSE) subprotocol in the following report.

Design and Implementation

The project was implemented in three steps, first, 1 out of 2 Oblivious Transfer implementation, second, the partial secrets exchange subprotocol (PSE) implementation, and the last step is to combine and use the first two-step with RSA and DES algorithm to implement the Certified Mail Protocol.

Step 1: OT's Implementation

A 1 out of 2 Oblivious Transfer is a protocol by which sender S transfers ignorantly to a receiver R one message out of two recognizable secret messages. More precisely, an OT_1^2 (S, R, M1, M2) is a protocol that satisfies the following three axioms:

1) If S executes OT_1^2 (S, R, M1, M2) properly, then R can read exactly one message: either M1 or M2: the probability of each to be read is one-half. Furthermore, in case R does not read Mi. he gains (by the execution of OT_1^2) no "helpful partial information" about Mi, where i belong {1,2}.

2) For S, the a- posterior probability that R got M1 remains one-half, provided S and R have executed OT_1^2 properly.

3) If S tries to increase S's a-posteriori probability of guessing which message was read by R, then R can detect this attempt with a probability of at least one half.

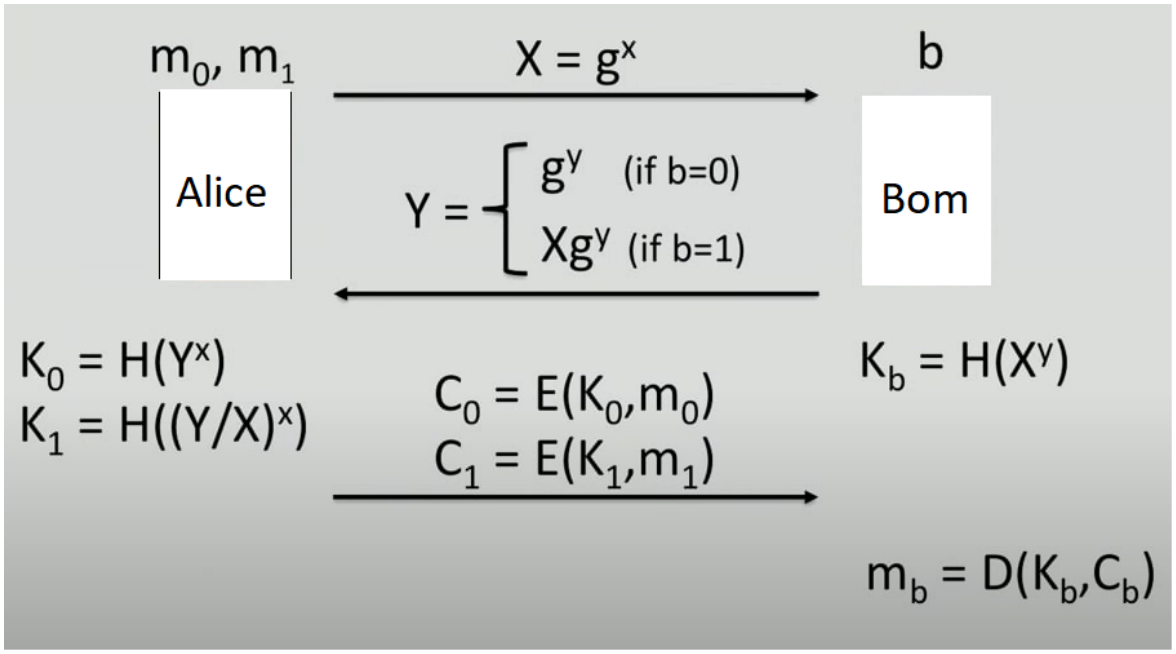

Figure 1-Flow of Oblivious Transfer

Step 2: The partial secrets exchange subprotocol (PSE)

PSE is a subprotocol that is mainly used for key transactions. The process is divided into two steps, in the first step, a for loop was used that runs 1 to n, n is the number of key pairs. During the loop, A sends a_i and a_{n+i} to B through the Oblivious Transfer protocol, and B sends b_i and b_{n+i} to A through the oblivious transfer protocol as well. In this step, both Alice and Bom have exactly one element of each pair of this counterparts' secrets; while his counterparts are ignorant of which elements x knows. In the second step, a loop was used again that runs 1 to L, where L is the length of each of the secrets. During this loop, A and B transmit the jth bit of each a_i and b_i to the other side. This completes the function of key exchange.

In order to achieve the goal mention at the beginning, cheat needs to be avoided, and in order to accomplish this, each party X take the following precautions:

1) During the first step, while playing the role of the receiver in OT_1^2. {The existence of this "mechanism" is guaranteed by axiom (3) of OT_1^2.

2) While executing the second step, X checks whether the bits revealed (to him) during the alternating substeps match the bits of the secrets which have been disclosed (to him) in the first step.

A party should stop further execution of the protocol as soon as he detects an attempt to cheat. This is sufficient to protect oneself against cheating.

Step 3: The Certified Mail Protocol

The step of the Certified Mail protocol was implemented inside Alice.java and Bom.java. The process can be divided into two steps. In step 1, the user denotes the signature of D to the message M. A randomly chooses a key K to F, then A transmits $F_K(M)$ and $F_K(S)$ to B. This means B has the encryption of the mail as well as an S puzzle, the solution of which is the key used for encrypting the mail. On the other hand, B transmits S_B ('from A', $F_K(M)$, $F_K(S)$) to A to acknowledge having received the message and keys being sent from A. In step 2, A transmits Ki, the ith bit of K to B, and B transmits S_B ($F_K(M)$, I, Ki) to A just to acknowledge each bit of the key he gets to A. The actual implementation procedures are organized and listed down below.

Step 1A:

- A randomly generates n+1 keys to F
- A computes $a_{n+i} = a_o$ xor a_i, for $1 \leq i \leq n$
- A computes $C = Fa_o(M)$ and $C_i^A = F_{a_i}(S)$, for $1 \leq i \leq 2n$
- A transmits C and $C_1^A, C_2^A, \ldots, C_{2n}^A$ to B

Step 1B:

- B randomly generates 2n keys to F
- Symbols K1 to K2n denote the solutions of the corresponding S puzzles C1 to C2n
- Make a acknowledge statement for the declaration
- B signs this declaration and transmits it to A

Step 2:

PSE(A, B, $|(a_i, a_{n+i})$ where i = 1 to n, $|(b_i, b_{n+i})$ where i = 1 to n)

The following figures show how each step is related, combined and used as a group to achieve the overall project. As it can be seen that OT was used in PSE to avoid cheating, which further protects the message transmission.

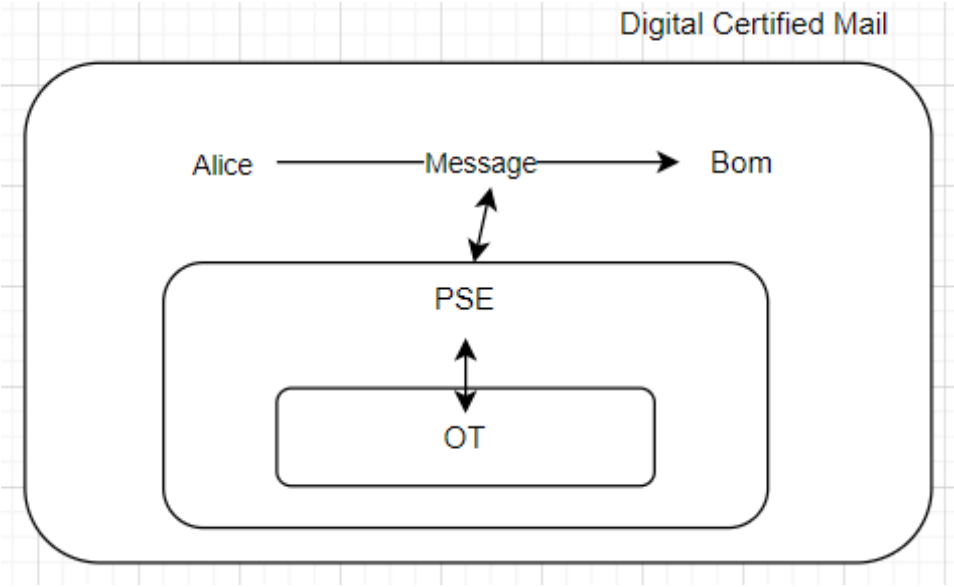

Figure 2 - Digital Certified Mail Flow

The implementation of the whole digital certified mail scheme is constructed using Java class library, consisting of four important classes, namely Alice.java, Bom.java, ObliviousTransfer.java and PSESubprotocol.java. To start with, Alice.java and Bom.java build up the socket connection between those parties. Most of the communication is through socket messaging in plain text or encrypted ciphers, the communications are transferred by the oblivious protocol, and happens between Alice and Bom. Between these communications, the Java security package's API was used to generate the receipt, generate and store RSA or DES keys within the PSESubprotocol.

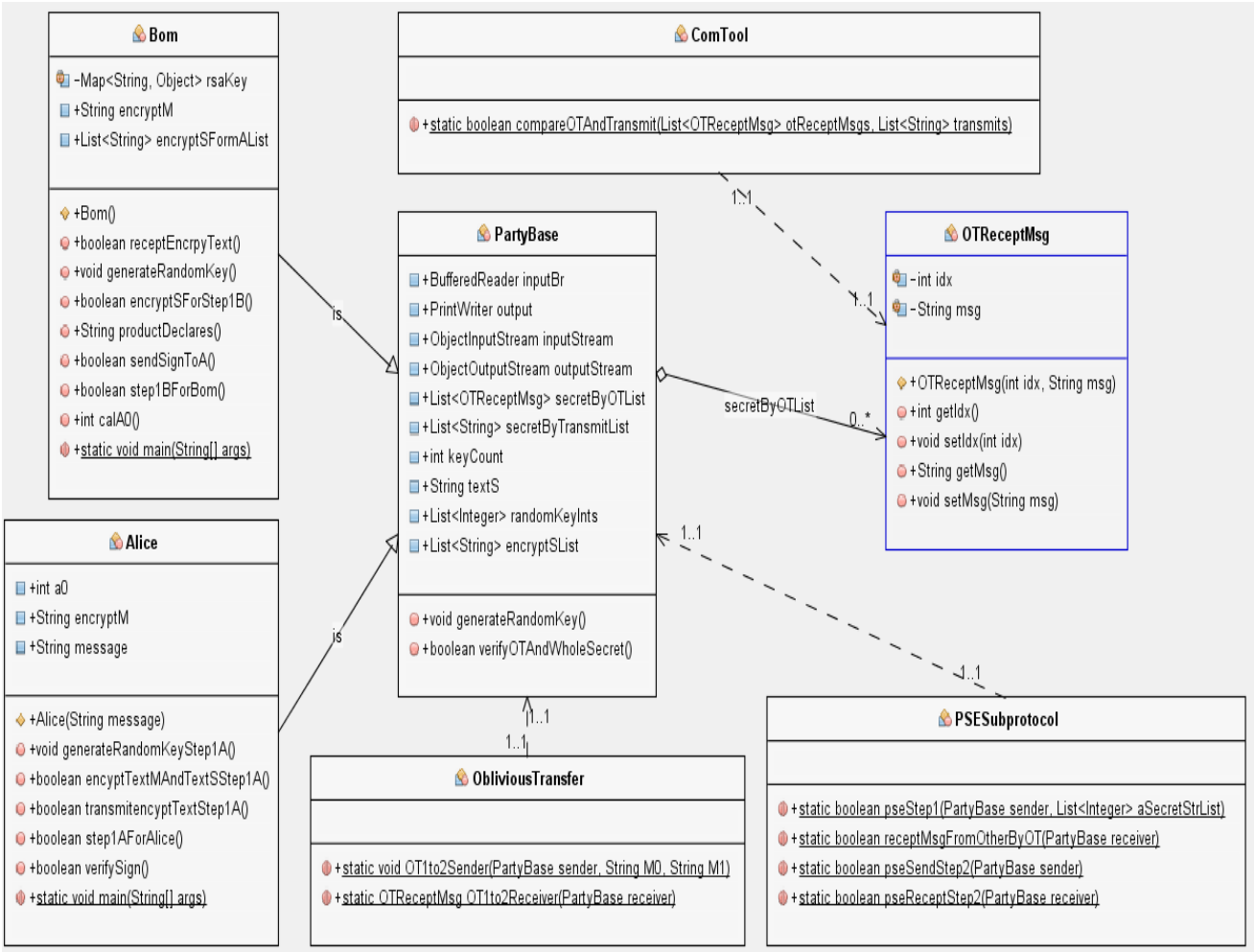

Figure 3 - Class diagram of Digital Certified Mail implementation

Figure 4 - Class diagram of Encrypt Key(RSA)

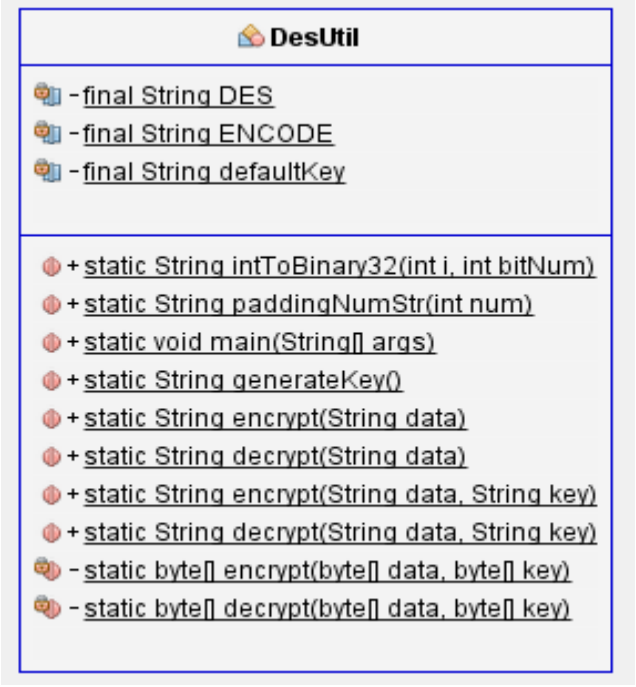

Figure 5 - Class diagram of Encrypt Key(DES)

(A) Alice.java and (B) Bom.java are based on the result from Lab 2 for setting up the socket connection, it provided a brief idea for setting up a client-server connection.

```
Server Application
Waiting client
ID recieved: INITIATOR A

Encrypting : RYERSON |INITIATOR A|RESPONDER B
Sending Encrypted code in byte [B@5f150435

Recieved cipher code in byte: [B@1c53fd30

Decrypted in string : RESPONDER B
```

Figure 6 - Setting up socket connection(Server)

```
Client Appication
Connected to localhost/127.0.0.1:60000

Sending ID: INITIATOR A

Recieved cipher code in byte:[B@769c9116
Decrypted code in byte[B@1c53fd30

Decrypted in string: RYERSON |INITIATOR A|RESPONDER B

Extracted Session key: RYERSON
Extracted host ID: RESPONDER B
Sending Last ID
```

Figure 7 - Setting up socket connection(Client)

Results

_____Figure 8 shows one pair of RSA keys in a key hashmap, the key pair was assigned to a data and the data was verified. The print statement of "true" means the verification was valid.

```
run:
Public key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrNE6qWPkxaSGtXcPVbxeMEiHBJl27ufKxjGsa
OnEr0sldkIQclhI4SiWK5bk8TmWYHZ2NKu6TaZ05ELQQ3ZPpOlbZcv6RNcRwbgEMwVTbQK0ETBpy
HYGQyRMTnzeFPleUuBw4Q5Xfl74CkVnNT2bKT5dQd/uNWNQq7yHW0BTGawIDAQAB

Private key: MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAKs0TqpY+TFpIaldw9VvF4wSIcEm
Xbu58rGMaxo6cSvSyV2QhByWEjhKJYrluTxOZZgdnY0q7pNpnTkQtBDdk+k6Vtly/pElxHBuAQzB
VNtArQRMGnIdgZDJExOfN4U/V5S4HDhDld+XvgKRWc1PZspPllB3+4lYlCrvIdbQFMZrAgMBAAEC
gYEApxxdlMVYJ7S43KJjnxfa4ixoqkIqyCzPCiNH/8f/44jSaPPBIK4ThsJJNvTZBFZ5sCtoN/0p
9tXmU2dMfBHsZuDaJQzsifIamU6dQpEVp6SxdJHQF40n0ofQnvJxybhmpk4W+aMClpUJ948564nM
FqV6QsKfNjCSwIvxEcncIhECQQD+Fdj0P5mVsicnUJRiWPtoTppuuZ8UezPXQ0Y2iithizIO7P+t
qZw6vz9hdqj8aYGgMvk/CIWvh8p2laNfrwDNAkEArH6TP2PRLDT6EvuazUI5djjvVpenEM6YHBlU
bYxqOa4Shjwl79wzhG96FYJNKx/LWR2qqHUs+cWlkCJl/rqEFwJBAKK2eP5JcGBl+y7pIm0s0KtG
kwlPX7ltAJDc6yzoJMnNqxvll/JtiLfyW3Nd2u/x3OorWoRsP7hd/GFHhN00pA0CQE1IAEycOfDm
Y0vx8e8YRopTA9fl5eG/zaJ5LrxN9qGU/bOqawzFyUcCct4EHPK/MenhQr5qkRXuoGPNJYk0o00C
QQDkhfM97Jy5MBKQde6PPj3N3VbXiYl+2XaT+6hiMEfjEHtl0BvfukyLUJs5b7rpDze0ivW2QANj
XVnekMLfatO2

true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 8 - RSA Test Result

Figure 9 shows a test of the encryption and decryption of the data using the DES algorithm, as it can be seen from the text being printed, the data was encrypted and decrypted successfully.

```
9emdLh4Hmbj+uVm31GQvyw==
I get it
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 9 - DES Test Result

The following figure 10 and figure 11 shows the resulting output between A and B, as it can be seen that it first reads the user input as a message, then the message was transmitted from A to B followed by step 1 that is described in step 3 of the design and implementation part. B prints “Step 1A: bom success” and “Step1B: bom success” to acknowledge the first step of certified mail protocol was executed successfully. After that, A verifies the receipt send from Bom, and prints “receipt from Bom is valid, B had received the message” to make acknowledgement. This completes all the identity verification and safety issues.

From Alice (A):

```
please input you want send message:
my name is haha
------receipt from Bom is valid, B had recepted the message-------
------alice send finish,the receiver is no problem------
BUILD SUCCESSFUL (total time: 14 seconds)
```

Figure 10 - Output display of Alice.java

From Bom (B):

```
Step1A: bom success
Step1B: bom success
---bom had decrypy message, message: my name is haha----
BUILD SUCCESSFUL (total time: 13 seconds)
```

Figure 11 - Output display of Alice.java

Conclusion

To conclude, this project was created to implement digital certified mail within the java programming language. The important parts of this program consist of three different phases: implementation of Oblivious transfer protocol, Partial Secrets Exchange Subprotocol and certified mail protocol. Those parts were the core of the whole project and took us a long time to understand the theory and concepts behind it.

The first phase allowed-sender and receiver to exchange cipher suites and details on protocol when the receiver could get an expected message and the sender was not able to know the receiver's decision which imposed the level of network security and prevented attacks such as backward replay. Next, with the help of the Partial Secrets Exchange Subprotocol, it would ensure that the message details were not changed. During the transfer of the protocol, RSA and DES keys were generated and ensured the encryption/decryption matches with the message, not letting any possibility of forging a receipt. Last but not least, phase 3 combined the two protocols and provided the whole application flow which builds up the entire digital certified mail scheme.

In this project, we believed the prior project objectives were accomplished as we have successfully designed a digital certified mail system. Our implementation has met the initial requirements stated in the outline of the project. Through this project, we were able to implement the certified mail protocol using experiences from previous laboratory works, for example, encryption and decryption of RSA and DES and materials apart from coursework. Also, it was a great experience to learn how to implement multiple cipher and protocol types by using java applications.

The project was completed in an engineering way that fell with the waterfall method process with every member contributing and showed leadership skills during this project. There are 6 stages in the waterfall process which are: system engineering, analysis, design, code, testing and maintenance. Yu took the responsibility for analysis and explained what we have to do and the requirements listed in the project. Haoran and Nixon have worked on the design of the application structure. Everyone has been assigned to finish a java class for implementation. Duanwei was the one who gathered the code and testing. Then after some expected result has shown, we enhance the whole coding together. Apart from that, in terms of group management, Duanwei started up the discussion of researching and providing information for choosing the topics. Also, Yu has created a chat room for us to communicate and express our ideas throughout the whole project process in the designed timetable for meetings. Nixon also took the responsibility for keeping track of the work and making sure everyone updates its process. Moreover, Haoran helped members to overcome its difficulties. All in all, every member has been participating in work and we worked together to complete this project actively.

Reference

https://courses.ryerson.ca/d2l/le/content/445575/viewContent/3445296/View

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.7448&rep=rep1&type=pdf

Appendix

Alice.java

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

import encrypt.DES;
import encrypt.RSA;



public class Alice extends PartyBase{
	public int a0;
	public String encryptM;
	public String message;

	public Alice(String message) {
		this.message = message;
	}

	public void generateRandomKeyStep1A(){
		a0 = Integer.parseInt(DES.generateKey());
		this.generateRandomKey();
		for (int i = 0; i < this.keyCount; i++) {
			this.randomKeyInts.add(a0 ^ this.randomKeyInts.get(i));
		}
	}

	public boolean encyptTextMAndTextSStep1A(){
		try {
			String key0 = DES.paddingNumStr(a0);
			encryptM = DES.encrypt(message,key0);
			for (int i = 0; i < 2*keyCount; i++) {
				String key = DES.paddingNumStr(this.randomKeyInts.get(i));
				this.encryptSList.add(DES.encrypt(textS,key));
```

```
                }
        } catch (Exception e) {
                e.printStackTrace();
                this.encryptSList.clear();
                return false;
        }
        return true;
}

public boolean transmitencyptTextStep1A(){

        this.output.println("Step1A");
        try {
                String line = this.inputBr.readLine();
                if(!line.equals("#OK#")){
                        return false;
                }

                this.outputStream.writeObject(encryptM);

                for (int i = 0; i < 2*keyCount; i++) {
                        this.outputStream.writeObject(this.encryptSList.get(i));
                }
                this.outputStream.flush();
        } catch (IOException e) {
                e.printStackTrace();
                return false;
        }
        return true;
}

public boolean step1AForAlice(){

        this.generateRandomKeyStep1A();

        if(!this.encyptTextMAndTextSStep1A()){
                return false;
        }

        return this.transmitencyptTextStep1A();
```

```
	}

	public boolean verifySign(){

		this.output.println("#OK#");
		boolean verifyFlag;
		try {

			String sign = (String) this.inputStream.readObject();

			String originText = (String) this.inputStream.readObject();

			String pk = (String) this.inputStream.readObject();

			verifyFlag = RSA.verify(originText,pk,sign);
		} catch (Exception e) {
			e.printStackTrace();
			return false;
		}
		return verifyFlag;
	}
	public static void main(String[] args) throws IOException {
		ServerSocket server = new ServerSocket(8888);
		Socket client = server.accept();
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		System.out.println("please input you want send message:");
		String mailMessage = br.readLine();
		Alice alice = new Alice(mailMessage);

		alice.inputBr = new BufferedReader(new
InputStreamReader(client.getInputStream()));
		alice.output = new PrintWriter(client.getOutputStream(),true);
		alice.inputStream = new ObjectInputStream(client.getInputStream());
		alice.outputStream = new ObjectOutputStream(client.getOutputStream());

		alice.step1AForAlice();

		if(!alice.inputBr.readLine().equals("Step1B")){
			System.err.println("protocol step is wrong");
			return;
```

```
                }

                if(alice.verifySign()){

                        System.out.println("------receipt from Bom is valid, B had recepted the
message-------");
                }else{
                        System.err.println("receipt from Bom is error");
                }

                PSESubprotocol.pseStep1(alice,alice.randomKeyInts);

                PSESubprotocol.receptMsgFromOtherByOT(alice);

                PSESubprotocol.pseSendStep2(alice);

                PSESubprotocol.pseReceptStep2(alice);

                if(alice.verifyOTAndWholeSecret()){
                        System.out.println("------alice send finish,the receiver is no
problem------");
                }else{
                        System.out.println("alice send finish,the receiver is not correct");
                }

                alice.inputBr.close();
                alice.outputStream.close();
                alice.outputStream.close();
                alice.inputStream.close();
                server.close();

        }
}
```

Bom.java

```
import java.io.*;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

import encrypt.DES;
import encrypt.RSA;
import entity.OTReceptMsg;

public class Bom extends PartyBase{
	private Map<String,Object> rsaKey;
	public Bom() {

		try {
			rsaKey = RSA.initKey();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	public String encryptM;

	public List<String> encryptSFormAList = new ArrayList<>();

	public boolean receptEncrpyText(){

		this.output.println("#OK#");

		try {

			encryptM = (String) this.inputStream.readObject();

			for (int i = 0; i <2*keyCount ; i++) {
				encryptSFormAList.add((String) this.inputStream.readObject());
			}
		} catch (IOException | ClassNotFoundException e) {
```

```
                e.printStackTrace();
                return false;
            }
        return true;
    }


    @Override
    public void generateRandomKey() {
        super.generateRandomKey();
        for (int i = 0; i < keyCount; i++) {
            this.randomKeyInts.add(Integer.parseInt(DES.generateKey()));
        }
    }

    public boolean encryptSForStep1B(){

        this.generateRandomKey();

        for (int i = 0; i < 2*keyCount; i++) {
            try {
                String key = DES.paddingNumStr(this.randomKeyInts.get(i));
                this.encryptSList.add(DES.encrypt(textS,key));
            } catch (Exception e) {
                e.printStackTrace();
                return false;
            }
        }
        return true;
    }

    public String productDeclares(){
        String beginDeclare = "[Beginning of B's Declaration]\n";
        String denotation = "[Denotation:]\n";
        StringBuilder content = new StringBuilder();
        String statement = "[Statement:] I acknowledge having received the mail, which
result from decrypting C by F\n";
        String endDeclare = "[End of B's Declaration]\n";
        for (int i = 0; i < this.encryptSFormAList.size()-1; i++) {
            content.append(encryptSFormAList.get(i)).append(',');
```

```
            }

content.append(encryptSFormAList.get(this.encryptSFormAList.size()-1)).append(';');
            for (int i = 0; i < this.encryptSList.size()-1; i++) {
                content.append(encryptSList.get(i)).append(',');
            }
            content.append(encryptSList.get(this.encryptSList.size()-1)).append(';');
            content.append('\n');

            return beginDeclare + denotation + content.toString() +  statement + endDeclare;
        }

        public boolean sendSignToA(){

            String declaration = this.productDeclares();
            try {

                String sign = RSA.sign(declaration, RSA.getPrivateKey(rsaKey));

                this.output.println("Step1B");
                if(!this.inputBr.readLine().equals("#OK#")){
                    System.err.println("A can not receiver step1B command");
                    return false;
                }

                this.outputStream.writeObject(sign);

                this.outputStream.writeObject(declaration);

                this.outputStream.writeObject(RSA.getPublicKey(rsaKey));
                this.outputStream.flush();
            } catch (Exception e) {
                e.printStackTrace();
                return false;
            }
            return true;
        }

        public boolean step1BForBom(){
```

```
                if(!encryptSForStep1B()){
                        return false;
                }
                return sendSignToA();
        }

        public int calA0(){
                int a0 = Integer.parseInt(secretByTransmitList.get(keyCount)) ^
Integer.parseInt(secretByTransmitList.get(0));

                return a0;
        }
        public static void main(String[] args) throws IOException {
                Socket client = new Socket("127.0.0.1", 8888);
                Bom bom = new Bom();
                bom.output = new PrintWriter(client.getOutputStream(), true);
                bom.inputBr = new BufferedReader(new
InputStreamReader(client.getInputStream()));
                bom.outputStream = new ObjectOutputStream(client.getOutputStream());
                bom.inputStream = new ObjectInputStream(client.getInputStream());

                String line=bom.inputBr.readLine();

                if(line.equals("Step1A")){
                        if(bom.receptEncrpyText()){
                                System.out.println("Step1A: bom success");
                        }else{
                                System.err.println("Step1A: bom error");
                        }
                }else{
                        return;
                }

                if(bom.step1BForBom()){
                        System.out.println("Step1B: bom success");
                }else{
                        System.err.println("Step1B: bom error");
                }

                PSESubprotocol.receptMsgFromOtherByOT(bom);
```

```
                PSESubprotocol.pseStep1(bom,bom.randomKeyInts);

                PSESubprotocol.pseReceptStep2(bom);

                PSESubprotocol.pseSendStep2(bom);

                if(bom.verifyOTAndWholeSecret()){

                        int a0 = bom.calA0();
                        try {
                                String realMessage =
DES.decrypt(bom.encryptM,DES.paddingNumStr(a0));
                                System.out.println("---bom had decrypy message, message:
"+realMessage+"----");
                        } catch (Exception e) {
                                e.printStackTrace();
                                System.err.println("bom decrypt message fail");
                                return;
                        }
                }else{
                        System.err.println("there is cheat, bom can not know message");
                }

                bom.inputBr.close();
                bom.output.close();
                bom.inputStream.close();
                bom.outputStream.close();
                client.close();

        }


}
```

ObliviousTransfer.java

```
import entity.OTReceptMsg;
import encrypt.DES;
import encrypt.RSA;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Map;
import java.util.Random;

import sun.security.krb5.internal.crypto.Des;

public class ObliviousTransfer {

	public static void OT1to2Sender(PartyBase sender,String M0, String M1){

		try {

			Map<String, Object> keyMap0 = RSA.initKey();

			Map<String, Object> keyMap1 = RSA.initKey();

			String r0 = RSA.getPublicKey(keyMap0);
			String r1 =RSA.getPublicKey(keyMap1);
			sender.outputStream.writeObject(r0);
			sender.outputStream.writeObject(r1);
			sender.outputStream.flush();

			String desKey = (String) sender.inputStream.readObject();

			String decryptDesKey0;
			try {
				decryptDesKey0 = RSA.decryptByPrivateKey(desKey,
RSA.getPrivateKey(keyMap0));
			}catch(Exception e){
//				System.err.println("no select 0");
				decryptDesKey0 = "00000000";
			}
			String decryptDesKey1;
			try {
```

```
                        decryptDesKey1 = RSA.decryptByPrivateKey(desKey,
RSA.getPrivateKey(keyMap1));
                    }catch (Exception e){
//                      System.err.println("no select 1");
                        decryptDesKey1 = "00000000";
                    }

                    String desEncrypt0 = DES.encrypt(M0, decryptDesKey0);
                    String desEncrypt1 = DES.encrypt(M1,decryptDesKey1);

                    sender.outputStream.writeObject(desEncrypt0);
                    sender.outputStream.writeObject(desEncrypt1);
                } catch (Exception e) {
                    e.printStackTrace();
                    return;
                }
        }

        public static OTReceptMsg OT1to2Receiver(PartyBase receiver){

                try {

                    String rsaPk0 = (String) receiver.inputStream.readObject();
                    String rsaPk1 = (String) receiver.inputStream.readObject();

                    String desKey = DES.generateKey();

                    Random random = new Random();
                    int bitIdx = random.nextInt(2);
                    String realRSAPK = bitIdx == 0?rsaPk0:rsaPk1;


                    String encryptDesKey = RSA.encryptByPublicKey(desKey, realRSAPK);


receiver.outputStream.writeObject(encryptDesKey);receiver.outputStream.flush();

                    String recMsg0 = (String) receiver.inputStream.readObject();
                    String recMsg1 = (String) receiver.inputStream.readObject();
                    String realRecMsg = bitIdx == 0?recMsg0:recMsg1;
```

```
                String decryptMsg = DES.decrypt(realRecMsg, desKey);
                return new OTReceptMsg(bitIdx, decryptMsg);
            } catch (IOException e) {
                e.printStackTrace();
            } catch (Exception e) {
                e.printStackTrace();
            }
            //7gd3BJxbDTU=  33552969
            return null;
        }
}
```

PSESubprotocol.java

```
import entity.OTReceptMsg;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class PSESubprotocol {

	public static boolean pseStep1(PartyBase sender,List<Integer> aSecretStrList){
		sender.output.println("#PSE1.0#");
		try {
			String line = sender.inputBr.readLine();
			if(!line.equals("#OK#")){
				return false;
			}
		} catch (IOException e) {
			e.printStackTrace();
			return false;
		}
		int n = aSecretStrList.size()/2;
		for (int i = 0; i < n; i++) {
			String ai= String.valueOf(aSecretStrList.get(i));String aiN=
String.valueOf(aSecretStrList.get(i+n));
			ObliviousTransfer.OT1to2Sender(sender,ai,aiN);
		}
		return true;
	}

	public static boolean receptMsgFromOtherByOT(PartyBase receiver){
		try {
			if(!receiver.inputBr.readLine().equals("#PSE1.0#")){
				return false;
			}
			receiver.output.println("#OK#");
		} catch (IOException e) {
			e.printStackTrace();
			return false;
		}
		for (int i = 0; i < receiver.keyCount; i++) {
```

```
                OTReceptMsg msg = ObliviousTransfer.OT1to2Receiver(receiver);
                receiver.secretByOTList.add(msg);
        }
        return true;
}

public static boolean pseSendStep2(PartyBase sender){
        sender.output.println("#PSE2.0#");
        List<Integer> aiList = sender.randomKeyInts;
        try {
                String line = sender.inputBr.readLine();
                if(!line.equals("#OK#")){
                        return false;
                }
        } catch (IOException e) {
                e.printStackTrace();
                return false;
        }

        sender.output.println(aiList.size());

        for (Integer ai:aiList){
                sender.output.println(ai);
        }
        return true;
}

public static boolean pseReceptStep2(PartyBase receiver){
        try {
                if(!receiver.inputBr.readLine().equals("#PSE2.0#")){
                        return false;
                }
        } catch (IOException e) {
                e.printStackTrace();
                return false;
        }
        receiver.output.println("#OK#");
        int n =0;
        try {
                 n = Integer.parseInt(receiver.inputBr.readLine());
```

```
            } catch (IOException e) {
                e.printStackTrace();
                return false;
            }
            for (int i = 0; i < n; i++) {
                try {
                    receiver.secretByTransmitList.add(receiver.inputBr.readLine());
                } catch (IOException e) {
                    e.printStackTrace();
                    receiver.secretByTransmitList.clear();
                    return false;
                }
            }
            return true;
        }

}
```

PartyBase.java

```
import entity.OTReceptMsg;
import encrypt.DES;
import java.io.BufferedReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;


public class  PartyBase {
	public BufferedReader inputBr=null;
	public  PrintWriter output=null;
	public ObjectInputStream inputStream =null;
	public ObjectOutputStream outputStream =null;
	public List<OTReceptMsg> secretByOTList = new ArrayList<>();
	public List<String> secretByTransmitList= new ArrayList<String>();

	public int keyCount = 5;

	public String textS = "I get it";

	public List<Integer> randomKeyInts = new ArrayList<>();

	public List<String> encryptSList = new ArrayList<>();

	public void generateRandomKey(){
		for (int i = 0; i < keyCount; i++) {
			randomKeyInts.add(Integer.parseInt(DES.generateKey()));
		}
	}

	public boolean verifyOTAndWholeSecret(){
		int idx =0;
		for (OTReceptMsg otReceptMsg: secretByOTList){

if(!otReceptMsg.getMsg().equals(secretByTransmitList.get(idx+otReceptMsg.getIdx()*keyCoun
t))){
```

```
                    return false;
                }
                idx ++;
            }
            return true;
        }
}
```

ComTool.java

```
package comutil;

import java.util.List;

import entity.OTReceptMsg;

public class ComTool {

    public static boolean compareOTAndTransmit(List<OTReceptMsg> otReceptMsgs,
List<String> transmits){
        int n = otReceptMsgs.size();
        for (int i = 0; i < n; i++) {
            OTReceptMsg otReceptMsg = otReceptMsgs.get(i);
            String targetStr = otReceptMsg.getIdx() ==
0?transmits.get(i):transmits.get(i+n);
            if(!targetStr.equals(otReceptMsg.getMsg())){
                return false;
            }
        }
        return true;
    }
}
```

DES.java

```
package encrypt;

import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
import java.io.IOException;
import java.security.SecureRandom;
import java.util.Random;

public class DES {
	private final static String DES = "DES";
	private final static String ENCODE = "GBK";
	private final static String defaultKey = "12345678";
	public static String intToBinary32(int i, int bitNum){
		String binaryStr = Integer.toBinaryString(i);
		while(binaryStr.length() < bitNum){
			binaryStr = "0"+binaryStr;
		}
		return binaryStr;
	}
	public static String paddingNumStr(int num){
		String numStr = String.valueOf(num);
		for (int i = numStr.length(); i < 8; i++) {
			numStr = "0" + numStr;
		}
		return numStr;
	}

	public static void main(String[] args) throws Exception {
		String data = "I get it";
//		/BHmI7JB0RQ8NCddMmHh3Q==
//		57737801
		String bs =intToBinary32(1,32);
		String secret = encrypt(data, "83676287");
		String secret1 = encrypt(data, "868064623");
```

```
        System.out.println(decrypt("/BHmI7JB0RQ8NCddMmHh3Q==","57737801"));
        System.out.println(encrypt(data));
        System.out.println(decrypt(encrypt(data)));

}

public static String generateKey(){
        Random random = new Random();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 8; i++) {
                sb.append(random.nextInt(10));
        }
        return sb.toString();
}

public static String encrypt(String data) throws Exception {
        byte[] bt = encrypt(data.getBytes(ENCODE), defaultKey.getBytes(ENCODE));
        String strs = new BASE64Encoder().encode(bt);
        return strs;
}


public static String decrypt(String data) throws IOException, Exception {
        if (data == null)
                return null;
        BASE64Decoder decoder = new BASE64Decoder();
        byte[] buf = decoder.decodeBuffer(data);
        byte[] bt = decrypt(buf, defaultKey.getBytes(ENCODE));
        return new String(bt, ENCODE);
}


public static String encrypt(String data, String key) throws Exception {
        byte[] bt = encrypt(data.getBytes(ENCODE), key.getBytes(ENCODE));
        String strs = new BASE64Encoder().encode(bt);
        return strs;
}


public static String decrypt(String data, String key) throws IOException,
```

```
            Exception {
        if (data == null)
            return null;
        BASE64Decoder decoder = new BASE64Decoder();
        byte[] buf = decoder.decodeBuffer(data);
        byte[] bt = decrypt(buf, key.getBytes(ENCODE));
        return new String(bt, ENCODE);
    }

    private static byte[] encrypt(byte[] data, byte[] key) throws Exception {

        SecureRandom sr = new SecureRandom();
        DESKeySpec dks = new DESKeySpec(key);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(DES);
        SecretKey securekey = keyFactory.generateSecret(dks);
        Cipher cipher = Cipher.getInstance(DES);
        cipher.init(Cipher.ENCRYPT_MODE, securekey, sr);
        return cipher.doFinal(data);
    }

    private static byte[] decrypt(byte[] data, byte[] key) throws Exception {

        SecureRandom sr = new SecureRandom();


        DESKeySpec dks = new DESKeySpec(key);


        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(DES);
        SecretKey securekey = keyFactory.generateSecret(dks);


        Cipher cipher = Cipher.getInstance(DES);


        cipher.init(Cipher.DECRYPT_MODE, securekey, sr);

        return cipher.doFinal(data);
    }
}
```

RSA.java

```
package encrypt;

import javax.crypto.Cipher;
import java.security.*;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.HashMap;
import java.util.Map;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;


public class RSA{
        public static final String KEY_ALGORITHM = "RSA";
        public static final String SIGNATURE_ALGORITHM = "MD5withRSA";

        private static final String PUBLIC_KEY = "RSAPublicKey";
        private static final String PRIVATE_KEY = "RSAPrivateKey";

      public static String encryptBASE64(byte[] key) throws Exception {
                return (new BASE64Encoder()).encodeBuffer(key);
        }

      public static byte[] decryptBASE64(String key) throws Exception {
                return (new BASE64Decoder()).decodeBuffer(key);
        }

        public static String sign(String data, String privateKey) throws Exception {
                return sign(data.getBytes(), privateKey);
        }


        public static String sign(byte[] data, String privateKey) throws Exception {
                byte[] keyBytes = decryptBASE64(privateKey);
```

```
            PKCS8EncodedKeySpec pkcs8KeySpec = new
PKCS8EncodedKeySpec(keyBytes);

            KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);

            PrivateKey priKey = keyFactory.generatePrivate(pkcs8KeySpec);

            Signature signature = Signature.getInstance(SIGNATURE_ALGORITHM);
            signature.initSign(priKey);
            signature.update(data);

            return encryptBASE64(signature.sign());
        }

        public static boolean verify(String data, String publicKey, String sign) throws Exception
{
            return verify(data.getBytes(), publicKey, sign);
        }

        public static boolean verify(byte[] data, String publicKey, String sign) throws Exception
{

            byte[] keyBytes = decryptBASE64(publicKey);

            X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);

            KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);

            PublicKey pubKey = keyFactory.generatePublic(keySpec);

            Signature signature = Signature.getInstance(SIGNATURE_ALGORITHM);
            signature.initVerify(pubKey);
            signature.update(data);

            return signature.verify(decryptBASE64(sign));
        }


        public static String decryptByPrivateKey(String data, String key) throws Exception {
```

```
        return new String(decryptByPrivateKey(decryptBASE64(data), key));
    }

    public static byte[] decryptByPrivateKey(byte[] data, String key) throws Exception {
        byte[] keyBytes = decryptBASE64(key);

        PKCS8EncodedKeySpec pkcs8KeySpec = new
PKCS8EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key privateKey = keyFactory.generatePrivate(pkcs8KeySpec);

        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.DECRYPT_MODE, privateKey);

        return cipher.doFinal(data);
    }

    public static String decryptByPublicKey(String data, String key) throws Exception {
        return new String(decryptByPublicKey(decryptBASE64(data), key));
    }

    public static byte[] decryptByPublicKey(byte[] data, String key) throws Exception {
        byte[] keyBytes = decryptBASE64(key);

        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key publicKey = keyFactory.generatePublic(x509KeySpec);

        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.DECRYPT_MODE, publicKey);

        return cipher.doFinal(data);
    }


    public static String encryptByPublicKey(String data, String key) throws Exception {
        return encryptBASE64(encryptByPublicKey(data.getBytes(), key));
    }

    public static byte[] encryptByPublicKey(byte[] data, String key) throws Exception {
```

```
        byte[] keyBytes = decryptBASE64(key);

        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key publicKey = keyFactory.generatePublic(x509KeySpec);

        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);

        return cipher.doFinal(data);
    }


    public static String encryptByPrivateKey(String data, String key) throws Exception {
        return encryptBASE64(encryptByPrivateKey(data.getBytes(), key));
    }

    public static byte[] encryptByPrivateKey(byte[] data, String key) throws Exception {
        byte[] keyBytes = decryptBASE64(key);

        PKCS8EncodedKeySpec pkcs8KeySpec = new
PKCS8EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key privateKey = keyFactory.generatePrivate(pkcs8KeySpec);

        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.ENCRYPT_MODE, privateKey);

        return cipher.doFinal(data);
    }

    public static String getPrivateKey(Map<String, Object> keyMap) throws Exception {
        Key key = (Key) keyMap.get(PRIVATE_KEY);

        return encryptBASE64(key.getEncoded());
    }

    public static String getPublicKey(Map<String, Object> keyMap) throws Exception {
        Key key = (Key) keyMap.get(PUBLIC_KEY);
```

```
        return encryptBASE64(key.getEncoded());
    }

    public static Map<String, Object> initKey() throws Exception {
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");

        keyPairGen.initialize(1024);

        KeyPair keyPair = keyPairGen.generateKeyPair();

        PublicKey publicKey = keyPair.getPublic();

        PrivateKey privateKey = keyPair.getPrivate();

        Map<String, Object> keyMap = new HashMap<String, Object>(2);

        keyMap.put(PUBLIC_KEY, publicKey);
        keyMap.put(PRIVATE_KEY, privateKey);
        return keyMap;
    }

    public static void main(String[] args) {
        try {
            Map<String, Object> map = RSA.initKey();
            String publicKey = RSA.getPublicKey(map);
            String privateKey = RSA.getPrivateKey(map);
            System.out.println("Public key:" + publicKey);
            System.out.println("Private key:" + privateKey);
            String data =
"sdfisfhdsfdohssdfsfdsfdsfafdsfsfafsfdsfdsfcxvxcvxvcxvvxv";
            String sign = sign(data, RSA.getPrivateKey(map));
            boolean verify = verify(data, RSA.getPublicKey(map), sign);
            System.out.println(verify);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

OTReceptMsg.java

```
package entity;

public class OTReceptMsg {

	private int idx;

	private String msg;

	public OTReceptMsg(int idx, String msg) {
		this.idx = idx;
		this.msg = msg;
	}

	public int getIdx() {
		return idx;
	}

	public void setIdx(int idx) {
		this.idx = idx;
	}

	public String getMsg() {
		return msg;
	}

	public void setMsg(String msg) {
		this.msg = msg;
	}
}
```