Computer Security : Projet 2

Pham Quang Vinh, Balon-Perin Alexandre & Rukundo Patrick

Groupe 15

Ecole Polytechnique de l'ULB
1er Master en Technologie de l'information et de la communication

Année Académique 2010-2011

Table of Contents

Introduction

Chiffrement asymétrique

Le chiffrement asymétrique repose sur le principe que la clé pour le chiffrement n'est pas la même que celle pour le déchiffrement. Dans cette paire de clés, l'une est publique, tout le monde y a accès, et l'autre est privée, connue d'une seule personne. Pour envoyer un message, l'expéditeur se servira de la clé publique du destinataire pour crypter le message et le destinataire utilisera sa clé privée afin de décrypter le message. Il est donc le seul à pouvoir décrypter le message.

Par contre, on inversera l'utilisation des clés afin de signer un message pour s'assurer de son authenticité. L'expéditeur signera de sa clé privée et le destinataire pourra vérifier l'authenticité du message grâce à la clé publique de l'expéditeur.

Le chiffrement RSA est une instance de la famille des chiffrements asymétriques.

Fonctions de hachage

Les fonctions de hachage sont utilisées pour s'assurer de l'intégrité des données. L'intégrité des données est l'assurance qu'elles n'aient pas été modifiées d'une manière non autorisée. On distingue les *manipulation detection codes* (MDC) qui vérifient l'intégrité des données et les *message authentification codes* (MAC) qui vérifient l'intégrité des données et réalisent également l'authentification de la source.

Signature

La signature digitale d'un message permet de vérifier l'authenticité de la source de ce message. En effet, le signataire signe le message de sa clé privée de signature et tout le monde peut alors vérifier l'authenticité des messages grâce à la clé publique du signataire. Les signatures utilisée lors de ce projet seront des signatures avec appendice utilisant les algorithmes de génération et vérification de signature RSA. Une signature avec appendice doit être accompagnée du message signé pour pouvoir être vérifiée. Le processus de signature utilise une fonction de hachage MDC.

Projet : application

Dans le cadre de ce projet, il nous est proposé d'utiliser les deux concepts mentionnés ci-dessus afin de réaliser un protocole de communication sécurisé entre deux personnes. Nous utiliserons l'algorithme RSA pour le chiffrement asymétrique et l'algorithme SHA-256 accompagné du RSA (SHA-256withRSA) pour la signature de messages.

Protocole de communication

Démarche

Afin de réaliser un protocole de communication sécurisé entre deux personnes nous suivrons une démarche qui permettra d'assurer une inviolabilité des données. A la fin du processus, les interlocuteurs communiqueront via des messages cryptés par l'algorithme 2-DES déjà développé lors du premier projet.
L'enjeu est donc que les utilisateurs échangent la paire de clés qu'ils utiliseront sans que celles-ci puissent être connues d'une tierce personne. Pour cela, il faudra au préalable créer un canal de communication crypté grâce au chiffrement asymétrique.

Les échanges entre utilisateur se faisant dans ce cas-ci par mail, la démarche à suivre pour établir ce canal de communication est la suivante :

1. ***L'utilisateur 1 choisit un nombre aléatoire r***
2. ***L'utilisateur 1 envoie à l'utilisateur 2 un message crypté. Il s'agit du chiffrement asymétrique (RSA) avec la clé publique de chiffrement de l'utilisateur 2 du message composé du nombre r, du numéro de l'utilisateur 1 (i) ainsi que la signature du groupe (r,i,j), j étant le numéro de l'utilisateur 2. La signature se fait grâce à la clé privée de signature de l'utilisateur 1. Le mail envoyé est muni du titre suivant : « Connection Request » et d'une pièce jointe « connection_request.txt » dans laquelle se trouve le résultat de la signature.***
3. ***L'utilisateur 2 déchiffre le message reçu grâce à sa clé privée de chiffrement. Il vérifie alors sa cohérence et la signature digitale de l'utilisateur 1 grâce à la clé publique de signature de l'utilisateur 1. Si tout va bien, il génère alors la paire de clés nécessaire à l'algorithme 2-DES.***
4. ***L'utilisateur 2 renvoie alors un message crypté avec la clé publique de l'utilisateur 1 comprenant la paire de clés générées ainsi que la signature correspondante, avec la clé privée de signature de l'utilisateur 2. Le mail envoyé est muni du titre suivant : « Key Proposal » et d'une pièce jointe « key_proposal.txt » dans laquelle se trouve le résultat de la signature.***
5. ***L'utilisateur 1 déchiffre alors le message reçu, vérifie également l'authentification du message. Si tout est en ordre, il récupère alors la paire de clés générées.***
6. ***L'utilisateur 1 signale cela en envoyant le message crypté grâce à 2-DES du nombre aléatoire r préalablement généré. . Le mail envoyé est muni du titre suivant : « Key Confirmation» et d'une pièce jointe « key_confirmation.txt » dans laquelle se trouve le résultat de la signature.***
7. ***L'utilisateur 2 déchiffre le message reçu au moyen de l'algorithme 2-DES pour vérifier que la clé a bien été envoyée.***
8. ***Les utilisateurs peuvent désormais communiquer en échangeant des messages chiffrés grâce à la paire de clés et l'algorithme 2-DES.***

Fonctionnement du programme

Le programme s'exécute en console et se déroule en plusieurs étapes.
Pour lancer une étape il suffit d'écrire le numéro de celle-ci.

- ***1: permet d'initialiser les clés (Privée pour l'utilisateur et Publique pour son correspondant). Il faudra cependant ajouter :***
  - ***Le nom du groupe avec qui on communique***
  - ***L'email du destinataire dans ce groupe***
- ***2: correspond aux étapes 1 et 2 du protocole de communication.***
- ***3: correspond aux étapes 3 et 4 du protocole.***
- ***4 : correspond aux étapes 5 et 6 du protocole.***
- ***5 : correspond à l'étape 7 du protocole.***
- ***6 : envoie un message par le canal sécurisé avec la clé de session***
- ***7 : reçoit et décrypte un message avec la clé de session***

Ajouts personnels

Mis à part les fonctionnalités requises pour le projet, nous avons décidé d'implémenter d'autres fonctionnalités pour la facilité d'utilisation du canal de communication.

Ces fonctionnalités se résument à un objet Mémoire et un outil de récupération et envoi automatiques de mails.

L'objet Memory permet de stocker certaines informations par sérialisation. Cela permet d'éviter de devoir recommencer tout le processus lors d'un arrêt imprévu ou volontaire du programme. On peut ainsi se permettre d'arrêter et reprendre le processus dans le cas où l'interlocuteur met plus de temps que prévu à répondre.

Pour la gestion des mails, on s'appuie sur la librairie JavaMail API. Grâce à celle-ci, on récupère aisément et stocke les pièces jointes reçues par mail. Il en est de même pour l'envoi. Tout ceci est facilité par l'établissement d'un standard de communication (objet des mails et noms des fichiers).

De plus, nous avons créé une classe outil pour tout ce qui est manipulation et traitement de bytes nécessaires dans le protocole.

Tout ce qui concerne les fonctionnalités de base requises se retrouve dans le package « crypto » qui sera développé plus longuement dans la section suivante.

Implémentation

Package crypto

Class Protocol

Cette classe reprend toutes les étapes concernant le protocole explicité plus tôt

Public void init(…)

On initialise et garde en objet Memory (voir ci-dessous) les numéros des utilisateurs concernés, leurs clés publiques et privées selon le cas et l'adresse e-mail du destinataire. Cette méthode doit être exécutée par les deux utilisateurs.

Public void initProtocol()

Ici, on va créer le premier message à envoyer, définir la signature correspondante et crypter le tout afin de l'envoyer au correspondant. Cette méthode est utilisée par le premier utilisateur.

On génère le nombre aléatoire r sous forme de 4 bytes, on crée également le numéro de l'utilisateur 1 et 2, on crée une signature correspondante et on concatène le tout dans un byteArray.

On garde r en mémoire pour la suite.

On découpe alors le byteArray en deux parties afin qu'elles puissent être chiffrées par un RSA (117 bytes maximum).

On crypte alors chacun des morceaux (private byte[] encrypt(…)), on concatène les résultats, on écrit le résultat dans un fichier .txt et on envoie le tout par mail, avec le sujet souhaité (package communication).

Cette méthode est utilisée par l'utilisateur 1.

Public boolean verifyProtocol(…)

On décrypte le fichier .txt reçu par mail pour retrouver le message (r,i,S(r,i,j)).

On manipule alors ce byteArray pour retrouver les différentes parties. Une fois que cela est fait, on peut vérifier l'authenticité du message (private byte[] verifySignature(…)).

On sauvegarde en mémoire le nombre aléatoire r reçu dans le message.

Une fois que cette vérification est fructueuse, on peut générer et envoyer la paire de clés DES.

Cette méthode est utilisée par l'utilisateur 2.

Public void sendSessionKey()

On génère deux clés DES grâce au KeyGenerator fourni par javax.crypto. On sauvegarde ces deux clés en mémoire.

Ensuite, on case ces clés dans un byteArray. On le découpe en deux parties pouvant être cryptées (117 bytes maximum). On crypte les deux parties, on les reconcatène, on les met dans un fichier texte, et on envoie le tout par mail.

Cette méthode est utilisée par l'utilisateur 2.

Public boolean receiveSessionKey(…)

On décrypte le contenu du fichier .txt reçu par mail. On découpe pour retrouver les différentes parties et on vérifie la signature du fichier. Si tout est bon, on récupère la paire de clés DES sous forme de bytes. On les stocke en mémoire pour la suite.

Cette méthode est utilisée par l'utilisateur 1.

Public void keyConfirmation()

On crypte le nombre r gardé en mémoire grâce à un 2-DES. On stocke dans un fichier .txt et on envoie par mail.

Cette méthode est utilisée par l'utilisateur 1.

Public void verifyResult(…)

On reçoit un message codé par 2-DES, on le décrypte, et si le résultat correspondant au nombre r stocké en mémoire, le canal de communication est établi et les deux utilisateurs peuvent communiquer en toute sécurité grâce à la paire de clés DES.

Cette méthode est utilisée par l'utilisateur 2.

Public void sendMessage()

Cette méthode crypte grâce à 2-DES et envoie des fichiers .txt une fois que le protocole est établi.

Public void readMessage()

Cette méthode décrypte grâce à 2-DES des fichiers .txt une fois que le protocole est établi.

Private byte[] encryptRSA (byte[] message, PublicKey otherEncryptionPublicKey)

Cette méthode permet de chiffrer un byteArray (117 bytes maximum) avec la clé publique de l'interlocuteur grâce à l'algorithme RSA et renvoie le résultat.

Private byte[] sign (byte[] message, PrivateKey mySignaturePrivateKey)

Cette méthode permet de signer un message grâce à l'algorithme RSAwith256. Elle utilise la clé privée de signature de l'expéditeur.

Private byte[] decryptRSATxtFile (PrivateKey myEncryptionSignatureKey, String address)

Cette méthode récupère le contenu du fichier texte, le découpe en deux byteArray de 128 bytes (taille d'output du chiffrement par RSA), déchiffre chacun de ces blocs grâce à la clé privée de signature de celui qui reçoit le message et les concatène pour obtenir le résultat avant encryption de l'autre côté.

Private boolean verifySignature (PublicKey otherSignaturePublicKey, byte[] toBeVerified, byte[] signature)

Cette méthode permet de vérifier si la signature correspond bien à ce qu'elle devrait être et permet donc de déterminer s'il y a eu des modifications inattendues. La vérification se fait grâce à la clé de signature publique de l'expéditeur du message.

Private byte[] createFirstBlock(…)

Cette méthode se charge de créer le premier bloc r,i,j sous forme de byteArray

Class KeyReader

Cette classe est un outil pour lire les fichiers .der contenant les clés publiques et privées de signature et de chiffrement.

Public PublicKey readPublicKey (String filename)

Public PrivateKey readPrivateKey (String filename)

Class DES2

Cette classe fournit un outil de chiffrement par l'algorithme 2-DES. Il s'agit de l'outil développé lors du premier projet.

Package main

Class Treatment

Cette classe gère tous les traitements de bytes, écriture et lecture des fichiers .txt.

Public static void toTxt (byte[] message, String address)

Cette méthode écrit un byteArray dans un fichier .txt.

Public static byte[] readTxtFile(…)

Cette méthode gère la lecture des fichiers .txt

Public static byte[] convertIntToByte(…)

Cette méthode se charge de transformer tout entier en byteArray.

Public static byte[] concatenate (byte[] message1, byte[] message2)

Cette méthode permet de concaténer deux byteArray

Public static byte[] cutByteArray (byte[] message, int startIndex, int endIndex)

Cette méthode prélève la partie située entre les deux index dans un byteArray.

Package communication

Class Mail

Public static void sendMail()

Cette méthode s'occupe d'envoyer les mails avec les pièces jointes nécessaires.

Public static void receiveMail()

Cette méthode s'occupe de récupérer les pièces jointes nécessaires et les stocke dans le dossier approprié.

Package memory

Class Memory implements Serializable

Les instances de cette classe servent à stocker les différentes données importantes au bon déroulement du protocole.

Cette classe est sérialisable, ce qui permet de stocker l'état de l'objet et les valeurs de ses variables. On peut donc les récupérer à n'importe quel instant et reprendre le déroulement du protocole quand on veut. Cela permet que chacun des utilisateurs puissent reprendre les étapes même s'ils ont interrompu le processus.

Public void save()

Cette méthode sauvegarde l'état de l'objet dans un fichier texte.

Public static void load()

Cette méthode permet de reprendre l'objet mémoire dans l'état dans lequel on l'a laissé avant d'interrompre le processus, si interruption il y a eu.

Conclusion

Résultats

Au terme de ce projet, nous avons réalisé un protocole de communication entre deux utilisateurs. Celui-ci nous permet d'échanger une paire de clés DES de façon sécurisée.

Afin de s'assurer que les deux utilisateurs soient les seuls à posséder cette paire de clés, il faut qu'ils communiquent dans un premier temps par des messages signés et ensuite cryptés par l'algorithme RSA. La signature et le chiffrement se font par chiffrement asymétrique à l'aide de paires de clés (privée et publique) de chiffrement et de signature.

Enseignements

Grâce à ce projet, nous avons appris à utiliser les algorithmes de chiffrement asymétrique et de signature.

Ce projet nous a également permis d'approcher les librairies externes de Java telles JavaMail API.

Annexes

Code entier

```
package main;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;

import crypto.Protocol;



public class Main {

	private int user1GroupNumber = 15;
	private int user2GroupNumber;

 private  String addressConnectionRequest =
 "received/connection_request.txt";
	private String addressKeyProposal =
"received/key_proposal.txt";
	private String addressKeyConfirmation =
"received/key_confirmation.txt";
	private String addressSecureCommunication =
"received/secure_communication.txt";

		private InputStreamReader isr;
	private BufferedReader br;
	private File file;

	public void init() {
		file = new File (addressConnectionRequest);
		file.delete();

		file = new File (addressKeyConfirmation);
		file.delete();

		file = new File (addressKeyProposal);
		file.delete();

		file = new File (addressSecureCommunication);
		file.delete();
```

```
            file = new File("received/clearTextFinal.txt");
            file.delete();



      }

      public void action(Protocol rsa){
            int etape = 0;

            while (etape<5){
                        isr = new InputStreamReader(System.in);
                        br = new BufferedReader(isr);
                        try {
                              System.out.print("Etape : ");
                              etape =
Integer.parseInt(br.readLine());
                        } catch (IOException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        }
                        if (etape==1){
                              isr = new
InputStreamReader(System.in);
                              br = new BufferedReader(isr);
                              int j = 0;
                              try {
                                    System.out.print("Numero de
l'autre Groupe : ");
                                    j =
Integer.parseInt(br.readLine());
                              } catch (IOException e) {
                                    // TODO Auto-generated catch
block
                                    e.printStackTrace();
                              }
                              this.user2GroupNumber = j;

                              this.init();
                              String s ="";
                              try {
                                    System.out.print("adresse mail
du contact de l'autre groupe : ");
                                    s = (br.readLine());
                              } catch (IOException e) {
                                    // TODO Auto-generated catch
```

```
block
                                    e.printStackTrace();
                              }
                              //on initialise le numero du groupe
                              rsa.init(user1GroupNumber,
user2GroupNumber, s);
                        }

                        else if (etape==2)
                              rsa.initProtocol();

                        else if (etape==3){
                              boolean resultVerification =
rsa.verifyProtocol( this.addressConnectionRequest);

                              if (resultVerification)

                                    rsa.sendSessionKey();

                        }

                        else if (etape==4){
                              //l utilisateur 1 verifie la
signature du message et recupere les deux clés envoyées par l
utlisateur 2
                              boolean resultVerification =
rsa.receiveSessionKey(addressKeyProposal);
                              if(resultVerification)
                                    rsa.keyConfirmation();
                        }

                        else if (etape==5)

      rsa.verifResult(addressKeyConfirmation);
                        else if (etape>5);
                     }
               while (etape < 8){
                 isr = new InputStreamReader(System.in);
                      br = new BufferedReader(isr);
                     try {
                          System.out.print("Etape : ");
                          etape = Integer.parseInt(br.readLine());
                     } catch (IOException e) {
                          // TODO Auto-generated catch block
                          e.printStackTrace();
                     }
                     if (etape==6){
```

```
                              String addressToSend =
"tmp/secure_communication.txt";// + address;
                              String addressTmp =
"toSend/secure_communication.txt";// + address;
                              rsa.sendMessage(addressTmp,
addressToSend);
                        }


                        else if (etape==7){
                              String address =
"received/secure_communication.txt";// + address;
                              rsa.readMessage(address);
                        }
            }
      }

      public static void main(String[] args) {
            // TODO Auto-generated method stub

            Protocol rsa = new Protocol();
            Main principal = new Main();

            principal.action(rsa);
            System.out.println("check");

      }

}
```

```java
package main;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;


public class Treatment {


	public static void toTxt(byte[] message, String
address0){
		//ecriture dans le fichier
		FileOutputStream fos;

		try {
			fos = new FileOutputStream(address0);
			fos.write(message);
			fos.close();
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
	public static byte[] convertIntToByte (int integer){

		byte [] result = new byte[4];
		result[0] = (byte) ((integer & 0xff000000) >>> 24);
		result[1] = (byte) ((integer & 0x00ff0000) >>> 16);
		result[2] = (byte) ((integer & 0x0000ff00) >>> 8);
		result[3] = (byte) ((integer & 0x000000ff));
//		System.out.print(result[0] + " " + result[1] + " "
+ result[2] + " " + result[3] + "\n");
		return result;

	}
	public static byte[] readTxtFile (String address){

		byte [] result = null ;
		FileInputStream fis;
		File file;
		try {
```

```
                file = new File(address);
                fis = new FileInputStream(file);

                int length = (int) file.length();
                result = new byte[length];

                int read = 0;
                int incr = 0;
                while ((read=fis.read()) != -1){
                      result[incr] = (byte)read;
                      incr++;
                }

          } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          }
          return result;
     }

     public static byte [] concatenate (byte[] byteArray1,
byte[] byteArray2){

          int length = byteArray1.length + byteArray2.length;
          byte[] result = new byte[length];

          for (int incr = 0 ; incr < length ; incr++){
                if (incr < byteArray1.length)
                      result[incr] = byteArray1[incr];
                else
                      result[incr] = byteArray2[incr-
byteArray1.length];
          }

          return result;
     }

     public static byte[] cutByteArray (byte[] byteArray,
int startIndex, int endIndex){
          byte[] result = new byte [endIndex-startIndex];
          for (int incr = startIndex ; incr < endIndex ;
incr++)
                result[incr-startIndex] = byteArray[incr];
          return result;
```

```
        }

    }
```

```java
package crypto;

import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
import java.util.Random;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

import communication.Mail;

import main.Treatment;
import memory.Memory;

public class Protocol {

	//declaration des variables
	private Signature signatureTool;
	private Cipher eCipher;
	private Cipher dCipher;

	private String signPrivateKeyFile;
	private String encryptPrivateKeyFile;
	private String otherEncryptPublicKeyFile;
	private String otherSignPublicKeyFile;


	private byte[] message;

	private byte[] rByte;
	private byte[] iByte;
	private byte[] jByte;

	private SecretKey sessionKey1;
	private SecretKey sessionKey2;

	private Memory memory;
	private PrivateKey myEncryptionPrivateKey;
```

```
	private PrivateKey mySignaturePrivateKey;
	private PublicKey otherEncryptionPublicKey;
	private PublicKey otherSignaturePublicKey;

	//initialise les informations pour lancer le protocole
	public void init(int myGroupNumber, int
otherGroupNumber, String addressContact) {
		//outil de lecture des fichiers .der
		KeyReader reader = new KeyReader();

		//determination chemins d acces des fichiers .der
en fct des numeros de groupe
		signPrivateKeyFile =
"keys/Groupe"+myGroupNumber+"_signature_private_key.der";
		encryptPrivateKeyFile =
"keys/Groupe"+myGroupNumber+"_encryption_private_key.der";
		otherEncryptPublicKeyFile =
"keys/Groupe"+otherGroupNumber+"_encryption_public_key.der";
		otherSignPublicKeyFile =
"keys/Groupe"+otherGroupNumber+"_signature_public_key.der";

		//recuperation des clés
		myEncryptionPrivateKey =
reader.readPrivateKey(encryptPrivateKeyFile);
		mySignaturePrivateKey =
reader.readPrivateKey(signPrivateKeyFile);
		otherEncryptionPublicKey =
reader.readPublicKey(otherEncryptPublicKeyFile);//
user2EncryptionKeyPair.getPublic();
		otherSignaturePublicKey =
reader.readPublicKey(otherSignPublicKeyFile);
//user2SignatureKeyPair.getPublic();

		//mise en memoire des données recuperees
		memory = new Memory();
		memory.setMyGroupNumber(myGroupNumber);
		memory.setOtherGroupNumber(otherGroupNumber);

	memory.setMyEncryptionPrivateKey(myEncryptionPrivateKey);

	memory.setMySignaturePrivateKey(mySignaturePrivateKey);

	memory.setOtherEncryptionPublicKey(otherEncryptionPublicK
ey);

	memory.setOtherSignaturePublicKey(otherSignaturePublicKey
);
```

```
            memory.setAddress(addressContact);
            memory.save();
      }

      //1ere etape du protocole  : connection request
      public void initProtocol() {
            //charge la memoire pour avoir acces aux donnees
sauvegardees
            memory = Memory.load();
            byte[] toBeSigned;
            byte[] signature;

            //creation du bloc r,i,j
            toBeSigned = createFirstBlock();

            //mise en memoire du r
            memory.setrByte(rByte);
            memory.save();


            //creation de la signature
            signature = sign(toBeSigned,
memory.getMySignaturePrivateKey());

            //concatenation r|i|signature
            message = Treatment.concatenate (rByte, iByte);
            message = Treatment.concatenate (message,
signature);

            //decoupage du message r|i|signature en blocs de
117 et 19 bytes ( = 136  = 8 (r,i) + 128 (signature) )
            byte[] message1 ;
            byte[] message2 ;
            message1 = Treatment.cutByteArray(message, 0, 117);
            message2 = Treatment.cutByteArray(message, 117,
136);

            //encryption
            byte[] encrypted1 = encryptRSA(message1,
memory.getOtherEncryptionPublicKey());
            byte[] encrypted2 = encryptRSA(message2,
memory.getOtherEncryptionPublicKey());

            //resultat de l encryption
            byte[] encrypted = Treatment.concatenate
(encrypted1, encrypted2);
```

```
            //envoi par mail du resultat
            Treatment.toTxt(encrypted,
"toSend/connection_request.txt");
            Mail.sendMail("toSend/connection_request.txt");
      }

      //verifie l'authenticite de la connection request
      public boolean verifyProtocol(String address){

            boolean resultVerification = false;

            memory = Memory.load();

            //dechiffrement du fichier connection_request.txt
            byte[] message =
decryptRSATxtFile(memory.getMyEncryptionPrivateKey(),
address);

            //retrouver les blocs r,i,j
            byte[] myGroupNumberByte =
Treatment.convertIntToByte(memory.getMyGroupNumber()); // j
            byte[] startOfMessage =
Treatment.cutByteArray(message, 0, 8); // r,i

            byte[] toBeVerified = Treatment.concatenate
(startOfMessage, myGroupNumberByte); // r,i,j

            //recuperation de la signature
            byte[] signature = new byte[128];

            for (int incr=0 ; incr < signature.length ; incr++)
                  signature[incr] = message[incr+8]; // les 8
premiers bytes est (r,i)

            //recuperer le r et mettre en memoire pour la suite
            this.rByte =
Treatment.cutByteArray(startOfMessage,0,4);
            memory.setrByte(rByte);
            memory.save();

            //verification de l'authenticite du message
            resultVerification =
verifySignature(memory.getOtherSignaturePublicKey(),
toBeVerified, signature);

            System.out.println("verification nombre, groupei,
groupej : " + resultVerification);
```

```
            return resultVerification;

      }

      //cree et envoie une paire de clés DES
      public void sendSessionKey() {
            KeyGenerator keygen;
            byte[] signature;
            try {
                  //generation de la paire de clés de session
pour 2-DES
                  keygen = KeyGenerator.getInstance("DES");
                  keygen.init(56);
                  SecretKey k1 = keygen.generateKey(); // 64
bytes
                SecretKey k2 = keygen.generateKey();

                this.sessionKey1 = k1;
                this.sessionKey2 = k2;

                //mise en memoire
                memory.setSessionKey1(sessionKey1);
                memory.setSessionKey2(sessionKey2);
                memory.save();

                //concatenation des deux clés pour avoir 16
bytes
                byte[] keys = Treatment.concatenate
(k1.getEncoded(), k2.getEncoded());

                //signature
                signature = sign(keys,
memory.getMySignaturePrivateKey());

                //concatenation du tout
                byte[] message;

                message = Treatment.concatenate (keys,
signature);

                //decoupage
                byte[] message1 = Treatment.cutByteArray
(message, 0, 117);
                byte[] message2 = Treatment.cutByteArray
(message, 117, 144);
```

```
                //encryption
                byte[] encrypted1 = encryptRSA(message1,
memory.getOtherEncryptionPublicKey());
                byte[] encrypted2 = encryptRSA(message2,
memory.getOtherEncryptionPublicKey());

                byte[] encrypted = Treatment.concatenate
(encrypted1, encrypted2);

                //envoi par mail
                Treatment.toTxt(encrypted,
"toSend/key_proposal.txt");
                Mail.sendMail("toSend/key_proposal.txt");

          } catch (NoSuchAlgorithmException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          }


     }

     //verifie l authenticite et recupere les clés de session
     public boolean receiveSessionKey(String address){

          memory = Memory.load();

          byte[] message =
decryptRSATxtFile(memory.getMyEncryptionPrivateKey(),
address);

          //retrouver les blocs
          byte[] toBeVerified =
Treatment.cutByteArray(message, 0, 16);
          byte[] signature = Treatment.cutByteArray(message,
16, 144);

          //verifie la signature
          boolean verify = verifySignature
(memory.getOtherSignaturePublicKey(), toBeVerified,
signature);
          System.out.println("verification authenticite des
cles : "  + verify);

          if (verify){
                System.out.println("cles de session acceptees
!");
```

```
                byte[] key1 = Treatment.cutByteArray(message,
0, 8);
                byte[] key2 = Treatment.cutByteArray(message,
8, 16);
                sessionKey1 = new SecretKeySpec(key1,"DES");
                sessionKey2 = new SecretKeySpec(key2,"DES");
                memory.setSessionKey1(sessionKey1);
                memory.setSessionKey2(sessionKey2);
                memory.save();

          }
          return verify;
     }

     //renvoie le r crypte par 2-DES pour confirmer la
reception des cles DES
     public void keyConfirmation() {
          memory = Memory.load();
          //definition des adresses de stockage des .txt
          String addressToSend =
"toSend/key_confirmation.txt";

          //outil 2-DES developpe au projet 1
          DES2 cryptoTool = new DES2();
          String IVstr ="00000000";
          cryptoTool.init(sessionKey1, sessionKey2, IVstr);

          //chiffrement par 2-DES
          byte[] encrypted =
cryptoTool.encrypt(memory.getrByte());

          Treatment.toTxt(encrypted, addressToSend);
          //envoi par mail
          Mail.sendMail(addressToSend);
     }

     //decrypte le r crypte par 2-DES si c est le resultat est
concluant c est la confirmation que l autre a bien recu la
paire de cles
     // on peut alors communiquer par 2-DES avec cette paire
de cles
     public void verifResult(String address){
          memory = Memory.load();

          //recepetion automatique du mail
          String title= "";
          if (address == "received/key_confirmation.txt"){
```

```
                title = "Key Confirmation";
            }
            Mail.receiveMail(title);

            //outil de chiffrement 2-DES
            String IVstr ="00000000";
            SecretKey k1 = memory.getSessionKey1();
            SecretKey k2 = memory.getSessionKey2();
            DES2 cryptoTool = new DES2();
            cryptoTool.init(k1, k2, IVstr);
            //dechiffrement du fichier recu
            byte[] read = Treatment.readTxtFile(address);
            byte [] decrypted = cryptoTool.decrypt(read);
            //comparaison avec le r garde en memoire
            boolean result = true;
            for (int incr = 0 ; incr < decrypted.length ; incr
++ )
                if(decrypted[incr] != memory.getrByte()[incr])
                    result = false;

            System.out.println(result);
      }

      //envoi des messages par 2-DES
      public void sendMessage(String addressTmp, String
addressToSend){
            memory = Memory.load();
            DES2 cryptoTool = new DES2();
            String IVstr = "00000000";
            cryptoTool.init(memory.getSessionKey1(),
memory.getSessionKey2(), IVstr);
            byte[] read = Treatment.readTxtFile(addressTmp);
            byte[] encrypted = cryptoTool.encrypt(read);
            Treatment.toTxt(encrypted, addressToSend);
            Mail.sendMail(addressToSend);
      }

      //reception de messages cryptes par 2-DES
      public void readMessage(String address){
            //recepetion automatique du mail
            String title= "";
            if (address ==
"received/secure_communication.txt"){
                title = "Secure Communication";
            }
            Mail.receiveMail(title);
```

```
            memory = Memory.load();
            DES2 cryptoTool = new DES2();
            String IVstr = "00000000";
            cryptoTool.init(memory.getSessionKey1(),
memory.getSessionKey2(), IVstr);
            byte[] read = Treatment.readTxtFile(address);
            byte[] decrypted = cryptoTool.decrypt(read);
            Treatment.toTxt(decrypted,
"received/decrypted_secure_communication.txt");
            System.out.println(new String(decrypted));
      }

      //chiffrement par RSA
      private byte[] encryptRSA(byte[] message, PublicKey
otherPublicKey){

            byte[] result = null;
            try {

                  //cryptage des deux blocs
                  eCipher =
Cipher.getInstance("RSA/ECB/PKCS1Padding");
                  eCipher.init(Cipher.ENCRYPT_MODE,
otherPublicKey );
                  result = eCipher.doFinal(message);


            } catch (NoSuchAlgorithmException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (InvalidKeyException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (BadPaddingException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
            return result;

      }
```

```
	//creation du bloc (r,i,j)
	private byte[] createFirstBlock () {

		memory = Memory.load();
		//creation du message
			//generation du nombre aleatoire
		Random randomGenerator = new Random();
		int r =  randomGenerator.nextInt((int)
java.lang.Math.pow(2,31));
		int signe = randomGenerator.nextInt(2);
		if (signe == 0)
			r = -1 - r;

		System.out.println("r : " + r);
		rByte = Treatment.convertIntToByte(r);
			//numero du groupe i
		int i = memory.getMyGroupNumber();
		iByte = Treatment.convertIntToByte(i);
			//numero du groupe j
		int j = memory.getOtherGroupNumber();
		jByte = Treatment.convertIntToByte(j);
			//concatenation
		byte[] result;
		result = Treatment.concatenate (rByte, iByte);
		result = Treatment.concatenate (result, jByte);

		return result;
	}

	//signe un message
	private byte[] sign(byte[] message, PrivateKey
myPrivateKey){
		byte[] result=null;
		try {
			signatureTool =
Signature.getInstance("SHA256withRSA");
			signatureTool.initSign(myPrivateKey);
				//signature du message cree
			signatureTool.update(message);
			result = signatureTool.sign();
		} catch (NoSuchAlgorithmException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InvalidKeyException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (SignatureException e) {
```

```
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            return result;
        }

        //recupere les mails et dechiffre les messages cryptes
RSA
        private byte[] decryptRSATxtFile (PrivateKey
myEncryptionPrivateKey, String address){

            byte result[] = null;
            String title = "";
            if(address == "received/connection_request.txt")
                title = "Connection Request";
            else if (address == "received/key_proposal.txt")
                title = "Key Proposal";

            Mail.receiveMail(title);
            //lecture du fichier
            byte[] inputTxt = Treatment.readTxtFile(address);

        //division en deux blocs de 128 bytes
        byte[] block1 = Treatment.cutByteArray(inputTxt, 0,
128);
        byte[] block2 = Treatment.cutByteArray(inputTxt, 128,
256);

        //decryption du message crypté
            try {
            dCipher =
Cipher.getInstance("RSA/ECB/PKCS1Padding");
                dCipher.init(Cipher.DECRYPT_MODE,
myEncryptionPrivateKey);
                byte[] decrypted1 = dCipher.doFinal(block1);
                byte[] decrypted2 = dCipher.doFinal(block2);

                //concatenation
                result = Treatment.concatenate (decrypted1,
decrypted2);

            } catch (InvalidKeyException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
                // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            } catch (BadPaddingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (NoSuchAlgorithmException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            return result;
      }

      //verifie une signature
      private boolean verifySignature (PublicKey
otherSignaturePublicKey, byte[] toBeVerified, byte[]
signature){
            //verification de la signature
            Signature verifyTool;
            boolean result = false;
            try {
                verifyTool =
Signature.getInstance("SHA256withRSA");

      verifyTool.initVerify(otherSignaturePublicKey);

                verifyTool.update(toBeVerified);
                result = verifyTool.verify(signature);
            } catch (NoSuchAlgorithmException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (InvalidKeyException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SignatureException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }


            return result;

      }

}
```

```
package crypto;

import java.security.InvalidAlgorithmParameterException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;

public class DES2 {

      Cipher ecipher1;
      Cipher ecipher2;
    Cipher dcipher1;
    Cipher dcipher2;
    CipherInputStream cis;

    public DES2() {

    }

      public void init(SecretKey k1, SecretKey k2, String
IVstr){
            try {

                  //Instance the ciphers
                  ecipher1 =
Cipher.getInstance("DES/CBC/PKCS5Padding");
               dcipher1 =
Cipher.getInstance("DES/CBC/PKCS5Padding");
               ecipher2 =
Cipher.getInstance("DES/CBC/NoPadding");
               dcipher2 =
Cipher.getInstance("DES/CBC/NoPadding");

               // Transform IV String --> byte
               byte[] ivBytes = IVstr.getBytes();
//             byte[] ivBytes = new byte[]{
//                     (byte)0x8E, 0x12, 0x39, (byte)0x9C,
//                     0x07, 0x72, 0x6F, 0x5A
//                 };
               IvParameterSpec IV = new
IvParameterSpec(ivBytes);

               // initiate the ciphers
```

```
                ecipher1.init(Cipher.ENCRYPT_MODE, k1,IV);
                dcipher1.init(Cipher.DECRYPT_MODE, k1,IV);
                ecipher2.init(Cipher.ENCRYPT_MODE, k2,IV);
                dcipher2.init(Cipher.DECRYPT_MODE, k2,IV);

            } catch (javax.crypto.NoSuchPaddingException e) {
                        e.printStackTrace();
            } catch (java.security.NoSuchAlgorithmException e)
{
                        e.printStackTrace();
            } catch (java.security.InvalidKeyException e) {
                  e.printStackTrace();
            } catch (InvalidAlgorithmParameterException e) {
                  e.printStackTrace();
            }
      }

      //crypte le message avec l'algorithme DES2
   public byte[] encrypt(byte[] message) {

            byte[] result=null;

            try {
                  result = ecipher1.doFinal(message);
                  result = ecipher2.doFinal(result);

            } catch (IllegalBlockSizeException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (BadPaddingException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }

            return result;
}

      //décrypte le message
      public byte[] decrypt(byte[] message) {

            byte[] result = null;
            try {
                  result = dcipher2.doFinal(message);
                  result = dcipher1.doFinal(result);

            } catch (IllegalBlockSizeException e) {
                  // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            } catch (BadPaddingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return result;
        }
    }
```

```
package crypto;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

public class KeyReader {

      public PrivateKey readPrivateKey(String filename){


            File f = new File(filename);
           FileInputStream fis;
            try {
                  fis = new FileInputStream(f);
                  DataInputStream dis = new
DataInputStream(fis);
                byte[] keyBytes = new byte[(int)f.length()];
                dis.readFully(keyBytes);
                dis.close();

                PKCS8EncodedKeySpec spec =
                  new PKCS8EncodedKeySpec(keyBytes);
                KeyFactory kf = KeyFactory.getInstance("RSA");
                PrivateKey privateKey =
kf.generatePrivate(spec);
                return privateKey;

            } catch (FileNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (NoSuchAlgorithmException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (InvalidKeySpecException e) {
```

```
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;

        }

        public PublicKey readPublicKey(String filename){
            File f = new File(filename);
           FileInputStream fis;
            try {

                fis = new FileInputStream(f);
                DataInputStream dis = new DataInputStream(fis);
                byte[] keyBytes = new byte[(int)f.length()];
                dis.readFully(keyBytes);
                dis.close();

                X509EncodedKeySpec spec =
                  new X509EncodedKeySpec(keyBytes);
                KeyFactory kf = KeyFactory.getInstance("RSA");
                PublicKey publicKey =  kf.generatePublic(spec);
                return publicKey;
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (NoSuchAlgorithmException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (InvalidKeySpecException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;
        }
}
```

```
package communication;

import java.io.File;
import java.io.IOException;
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.Part;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Store;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

import memory.Memory;

public class Mail {
	private static String user = "pandaziatic";
	private static String password = "samourai";
	private static String email ="pandaziatic@gmail.com";
	public static void sendMail(String address) {
		Memory memory = Memory.load();
		System.out.println(memory.getAddress());
		System.out.println("sending email: "+address);
		Properties props = new Properties();
	props.put("mail.smtp.host", "smtp.gmail.com");
	props.put("mail.smtp.socketFactory.port", "465");

	props.put("mail.smtp.socketFactory.class","javax.net.ssl.
SSLSocketFactory");
	props.put("mail.smtp.auth", "true");
	props.put("mail.smtp.port", "465");

		Session session = Session.getDefaultInstance(props,
		new javax.mail.Authenticator()
		{
			protected PasswordAuthentication
getPasswordAuthentication()
			{ return new
PasswordAuthentication(user,password);}
```

```
            });

      try {
            Message message = setMail(address, memory,
session);
                Transport.send(message);

           System.out.println("Done");

      } catch (MessagingException e) {
          throw new RuntimeException(e);
      } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
      }

      private static Message setMail(String address, Memory
memory,
                  Session session) throws MessagingException,
AddressException,
                  IOException {
            File file = new File(address);
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(email));
            message.setRecipients(Message.RecipientType.TO,

InternetAddress.parse(memory.getAddress()));
            if (address == "toSend/connection_request.txt")
                  message.setSubject("Connection Request");
            else if (address == "toSend/key_proposal.txt")
                  message.setSubject("Key Proposal");
            else if (address == "tmp/secure_communication.txt")
                  message.setSubject("Secure Communication");
            else if (address == "toSend/key_confirmation.txt")
                  message.setSubject("Key Confirmation");
            if (file != null) {
                  MimeBodyPart mbp1 = new MimeBodyPart();
                  mbp1.setText("groupe 15");
                  MimeBodyPart mbp2 = new MimeBodyPart();
                  mbp2.attachFile(file);
                  MimeMultipart mp = new MimeMultipart();
                  mp.addBodyPart(mbp1);
                  mp.addBodyPart(mbp2);
                  message.setContent(mp);
            }
            return message;
```

```
	}

	public static void receiveMail(String title){

		String host = "209.85.227.109";

		Properties properties = new Properties();
		properties.put("mail.imap.host", "imap.gmail.com");
	properties.put("mail.imap.socketFactory.port", "993");

	properties.put("mail.imap.socketFactory.class","javax.net
.ssl.SSLSocketFactory");
	properties.put("mail.imap.socketFactory.fallback",
"false");
	properties.put("mail.imap.auth", "true");
	properties.put("mail.imap.port", "993");
	properties.put("mail.imap.auth", "true");
	properties.put("mail.imap.user", email);
	properties.put("mail.imap.starttls.enable", "true");
	properties.put("mail.debug", "false");

		//rÈcupËre la session par dÈfaut et authentifie
l'utilisateur
		Session session =
Session.getInstance(properties,new javax.mail.Authenticator()
		{
			protected PasswordAuthentication
getPasswordAuthentication()
			{ return new
PasswordAuthentication(user,password);}
		});

		try {
			// permet d'implÈmenter le protocol imap
			Store store = session.getStore("imap");
			//connecte au host spÈcifiÈ (ici gmail) avec
le login (user,password)
			store.connect(host, user, password);

			Folder folder = getFolder(store);

			getAttachment(title, folder);
			folder.close(true);
			store.close();
		} catch (NoSuchProviderException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
```

```
            } catch (MessagingException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }

      private static Folder getFolder(Store store) throws
MessagingException {
            //CrÈe un dossier dans lequel on rÈcupËre les mails
qui existent dans le dossier "Crypto" sur gmail
            Folder folder = store.getFolder("Crypto");

            //si le nom de dossier n'existe pas on le spÈcifie
            if (folder == null) {
                  System.out.println("Invalid folder");
                  System.exit(1);
            }

            //ouvre le dossier en lecture seule
            folder.open(Folder.READ_ONLY);

            int totalMessages = folder.getMessageCount();
            //vÈrifie que le dossier n'est pas vide
            if (totalMessages == 0) {
                  System.out.println("Empty folder");
                  folder.close(false);
                  store.close();
                  System.exit(1);
            }
            return folder;
      }

      private static void getAttachment(String title, Folder
folder)
                  throws MessagingException, IOException {
            //rÈcupËre les messages du dossier "Crypto"
            Message[] message = folder.getMessages();

            //Inverse l'ordre des messages pour avoir les plus
rÈcents au-dessus
            message = reverse(message,folder);
```

```
            //parcours les mails
            for (int a = 0; a < message.length; a++) {

                  String subject = message[a].getSubject();
                  //si il y a un sujet au mail et que ce sujet
correspond au sujet recherchÈ on rÈcupËre la piËce jointe
                  if(subject != null){
                  if(subject.equals(title)){

                        MimeMultipart multipart =
(MimeMultipart) message[a].getContent();
                        String disposition =
message[a].getDisposition();
                        if (disposition == null ||
disposition.equalsIgnoreCase(Part.ATTACHMENT)) {
                              for(int j = 0;
j<multipart.getCount(); j++){
                                    MimeBodyPart bodyPart =
(MimeBodyPart) multipart.getBodyPart(j);
                                    String fileName =
bodyPart.getFileName();
                                    File file = new
File("received/"+fileName);
                                    bodyPart.saveFile(file);
                              }
                        }
                        break;
                  }
                  }
            }
      }


      private static Message[] reverse(Message[] message,
Folder folder){
            Message[] reverse = null;
            try {
                  reverse = folder.getMessages();
                  int counter = 0;

                  for(int i  = (message.length-1) ; i>=0;i--){
                        reverse[counter] = message[i];
                        counter++;
                  }
                  return reverse;
            } catch (MessagingException e) {
```

```
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;

        }
    }
```

```java
package memory;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.security.PrivateKey;
import java.security.PublicKey;

import javax.crypto.SecretKey;

import communication.Mail;

public class Memory implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	private byte[] rByte;
	private byte[] iByte;
	private byte[] jByte;
	private SecretKey sessionKey1;
	private SecretKey sessionKey2;
	private int myGroupNumber;
	private int otherGroupNumber;
	private boolean initializedMain;
	private PrivateKey myEncryptionPrivateKey;
	private PrivateKey mySignaturePrivateKey;
	private PublicKey otherEncryptionPublicKey;
	private PublicKey otherSignaturePublicKey;
	private String address;
	private Mail mail;

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	public boolean isInitializedMain() {
```

```
            return initializedMain;
      }

      public void setInitializedMain(boolean initializedMain)
{
            this.initializedMain = initializedMain;
      }

      public void save(){
            try {
                  ObjectOutputStream oos = new
ObjectOutputStream(new FileOutputStream("memory/memory.txt"));
                  oos.writeObject(this);
                  oos.close();
            } catch (FileNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }

      public static Memory load(){
            ObjectInputStream ois;
            Memory result = null;
            try {
                  ois = new ObjectInputStream(new
FileInputStream ("memory/memory.txt"));
                  result = (Memory) ois.readObject();

            } catch (FileNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (ClassNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
            return result;
      }
      public void setOtherGroupNumber(int otherGroupNumber) {
            this.otherGroupNumber = otherGroupNumber;
      }
      public int getOtherGroupNumber() {
```

```
            return otherGroupNumber;
      }
      public void setMyGroupNumber(int myGroupNumber) {
            this.myGroupNumber = myGroupNumber;
      }
      public int getMyGroupNumber() {
            return myGroupNumber;
      }
      public byte[] getrByte() {
            return rByte;
      }
      public void setrByte(byte[] rByte) {
            this.rByte = rByte;
      }
      public byte[] getiByte() {
            return iByte;
      }
      public void setiByte(byte[] iByte) {
            this.iByte = iByte;
      }
      public byte[] getjByte() {
            return jByte;
      }
      public void setjByte(byte[] jByte) {
            this.jByte = jByte;
      }
      public SecretKey getSessionKey1() {
            return sessionKey1;
      }
      public void setSessionKey1(SecretKey sessionKey1) {
            this.sessionKey1 = sessionKey1;
      }
      public SecretKey getSessionKey2() {
            return sessionKey2;
      }
      public void setSessionKey2(SecretKey sessionKey2) {
            this.sessionKey2 = sessionKey2;
      }

      public PrivateKey getMyEncryptionPrivateKey() {
            return myEncryptionPrivateKey;
      }

      public void setMyEncryptionPrivateKey(PrivateKey
myEncryptionPrivateKey) {
            this.myEncryptionPrivateKey =
myEncryptionPrivateKey;
```

```
	}

	public PrivateKey getMySignaturePrivateKey() {
		return mySignaturePrivateKey;
	}

	public void setMySignaturePrivateKey(PrivateKey
mySignaturePrivateKey) {
		this.mySignaturePrivateKey = mySignaturePrivateKey;
	}

	public PublicKey getOtherEncryptionPublicKey() {
		return otherEncryptionPublicKey;
	}

	public void setOtherEncryptionPublicKey(PublicKey
otherEncryptionPublicKey) {
		this.otherEncryptionPublicKey =
otherEncryptionPublicKey;
	}

	public PublicKey getOtherSignaturePublicKey() {
		return otherSignaturePublicKey;
	}

	public void setOtherSignaturePublicKey(PublicKey
otherSignaturePublicKey) {
		this.otherSignaturePublicKey =
otherSignaturePublicKey;
	}

	public void setMail(Mail mail) {
		this.mail = mail;
	}

	public Mail getMail() {
		return mail;
	}

}
```