Projet sécurité et cryptographie :

Implémenter l'algorithme **DES** complet et réaliser un programme **client-serveur** qui implémente un protocole d'échange sécurisé

Etablissement : Ecole d'ingénieurs Sup Galilée
Spécialité : Informatique
Année scolaire : Troisième année - équivalent M2
Lieu : Université Sorbonne Paris Nord - 93430 Villetaneuse

Réalisé par : Ludovik TEKAM

Année universitaire : 2019 - 2020

Table des matières

1 Présentation du projet **2**

2 Le Rendu **2**

3 Implémentation de l'algorithme DES complet **3**

4 Programme client-serveur qui implémente un protocole d'échange sécurisé **3**
4.1 Remarque à savoir 3
4.2 Fonctionnement 3
4.3 Comment exécuter ? 4

5 Problèmes rencontrés **4**

1 Présentation du projet

Le serveur écoute sur un port choisi (paramètre de lancement). Le client se connecte sur l'adresse ip et le port du serveur (paramètre de lancement). Le serveur a une paire de clés générés au préalable (avec le programme ou avec openssl). Le type de message échangés entre le client et le serveur est à votre discrétion. Par exemple vous pouvez implémenter un serveur de fichier simpliste qui réponds aux commandes suivantes ls,put,get

Fonctionnalités obligatoires :

- Les échanges doivent être sécurisés dans les deux sens
- Les messages échangés doivent être supérieurs à la taille du bloc
- Utiliser le mode ECB

Le client peut faire les actions suivantes :

- **Début de session**
  - Le client envoie un message de début de session (par exemple HELLO)
  - Le serveur à la réception répond avec sa clé publique
  - A la réception du la clé publique, le client génère une clé secrète DES (ou 3DES si vous avez fait le bonus) et la chiffre en RSA avec la clé publique du serveur
  - Le client envoie la clé secrète chiffré
  - Le serveur reçoit la clé secrète, la déchiffre et réponds au client avec un message d'acquittement
- **Fin de session**
  - Le client invalide la clé et informe le serveur qui à son tour invalide la clé
- **Les échanges eux-mêmes**
  - Les échanges peuvent se faire seulement si la session est valide

2 Le Rendu

- Server.java
- Client.java
- DES.java
- SDES.java
- dossier disk avec un dossier serveur et un dossier client (vide et qui servira d'espace de sauvegarde pour les clé généré)
- Le rapport

3 Implémentation de l'algorithme DES complet

Globalement, j'ai suivi l'algorithme. J'ai cependant rajouté quelques fonctions supplémentaires pour gérer les conversions hexadécimales - Binaire - utf8.

4 Programme client-serveur qui implémente un protocole d'échange sécurisé

Les échanges entre le client et le serveur se font de la manière suivante :

4.1 Remarque à savoir

L'échange se fait entre un client et un serveur. La nature des messages échangés est un **Mini tchat** entre le client et le serveur.

les variables importantes :

```
// --------------------------------//
String address_serveur = "127.0.0.1";
int port_con = 1254;

String msg_end = "bye";
String msg_init_session = "HELLO";
String messageFin = "vide";
int nbr_msg = 0;
//int key_SDES = 0;
String key_DES = "0000000000000000";
```

Sauvegarde des données, J'ai crée un dossier disk qui contient 2 dossiers :

- Le dossier serveur : qui simulera le disk serveur et contiendra notamment les différentes clés du serveur
- Le dossier client : qui simulera le disk client et contiendra notamment la copie de la clé publique du serveur (qui sera envoyée par le serveur au moment des différents échanges).

4.2 Fonctionnement

- **Début de session**
  - Le serveur est lancé, il génère ses clés (public et privé en utilisant RSA) puis il attend que le client se connecte
  - Le client se connecte, envoie un message de début de session **HELLO**
  - Le serveur à la réception répond avec sa clé publique
  - A la réception du la clé publique, le client génère une clé secrète DES (ici j'utilise une clé statique initialiser dans la variable $\mathbf{key}_DES)etlachiffreenRSAaveclaclpubliqueduserveurL$
  - Le serveur reçoit la clé secrète, la déchiffre et réponds au client avec un message d'acquittement
- **Fin de session**

— Le client invalide la clé et informe le serveur (en envoyant le message de fin de session **bye**) qui à son tour invalide la clé

— **Les échanges eux-mêmes**

— une fois que la clès DES est connue du client et du serveur, tous les message échages dans ce tchat seront chiffré

Je génère la clé publique et je la sauvegarde au format texte afin de faciliter l'envoi via la socket et je génère la clé privée au format binaire (mais j'aurais pu le faire au format texte)

4.3 Comment exécuter ?

— Exécuter et lancer le serveur

```
javac Serveur.java
java Serveur
```

— Exécuter et lancer client

```
javac Serveur.java
java Serveur
```

— Vous pouvez maintenant envoyer des messages sécuriser

5 Problèmes rencontrés

Envoie de données via les sockets : je me suis rendu compte qu'il fallait encoder certaines données avant de les envoyées, que se soit du côté serveur ou du côté client par exemple

— Pour l'envoi de la clé publique

— Envoie de la clé DES (qui est au préalable chiffré avec RSA)

```
messageToServer = Base64.getEncoder().encodeToString(secret);
```

— Difficulté à reconvertir les espaces : les espaces en fin et de début de blocs ne sont pas converti pas ma fonction.Ce qui fais en que "bonjour " -> "bonjour" sans l'espace de fin. Mais bonjour "bon jour" -> "bon jour" (conversion bien effectue)

— Envoie de la clé DES (qui est au préalable chiffré avec RSA)

pour résoudre ce problème, il faut simplement modifier ma fonction **hextoBin64Bit** de manière à avoir une fonction **hextoBin**

Remarque :J'ai copié certaines fonctions sur des sites tels que openclassroom, stackoverflow...

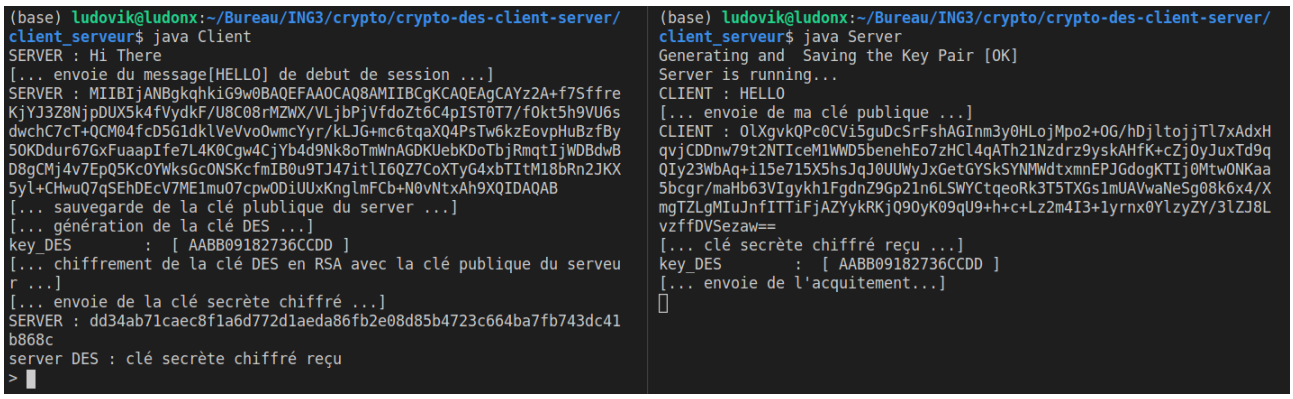

FIGURE 1 – Initialisation des échages

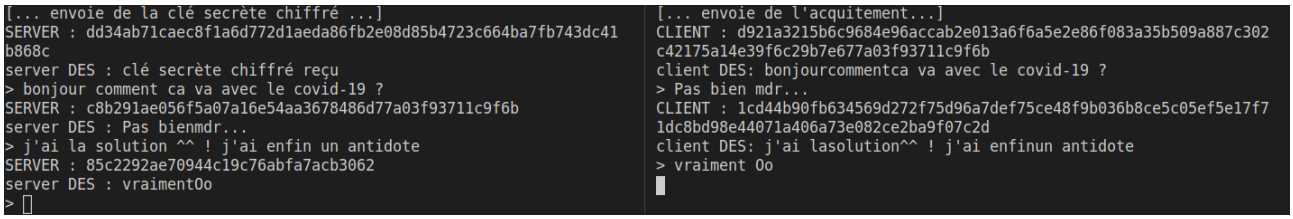

FIGURE 2 – Tchat entre le client et le serveur