# Rapport du projet : Chat sécurisé

Sophie Bousquet

05/11/2023

## 1 Introduction

Ce projet a pour objectif de concevoir et d'implémenter un chat sécurisé en utilisant le langage de programmation Python. Et en utilisant, le chiffrement RSA et le mode de chiffrement CBC (Cipher Block Chaining), sont utilisés pour renforcer la sécurité des échanges.

Pour lancer le projet il faut tout d'abord lancer `serveur.py` puis `client1` et ensuite `client2`. Lorsque nous les lancerons, une interface graphique propre à chacun se lancera.

Lorsque client 2 est lancé, cela enverra automatiquement un message à client 1 avec la clef publique de client2. Lorsque client1 reçoit le message, client1 envoie automatiquement la clef de session, chiffrée avec rsa, qui est nécessaire au chiffrement et déchiffrement des messages.

Le projet est constitué de 5 fichiers Python : `client1.py`, `client2.py`, `serveur.py`, `chiffrement.py` et `projet.py`.
Les fichier clients servent à envoyer et recevoir des messages au serveur afin de redirectionner les messages à l'autre client.

Le fichier serveur sert à faire la liaison entre client1 et client2 afin de transmettre les messages de l'un à l'autre.

Le fichier `chiffrement.py` contient les fonctions nécessaires au chiffrement CBC des messages.
Ensuite, le fichier projet, qui contient l'interface graphique que les clients vont utiliser, est constitué d'une image de fond ainsi que d'un bouton d'envoi, d'un bouton pour quitter et d'une zone de texte permettant d'écrire notre message.

## 2 Fonctionnement de `client1.py`

Au début du programme, on charge l'interface graphique afin de créer le fond en premier lieu.

Puis, on initialise HOST et PORT afin de faire une demande de connexion au serveur et on crée la socket du client.

On créer la variable clef de session qui est générée aléatoirement, et qui sera utilisée dans le chiffrement des messages. De même pour la variable `rsa_p` qui est pour l'instant une liste de 0 et 0 mais qui par la suite contiendra la clef publique du client2, `message_recu`, qui est égale à 0 pour l'instant et qui servira comme condition afin de récupérer la clef publique et d'envoyer la clef de session chiffrée avec rsa et v qui sera utilisée lors du chiffrement.

On définit une fonction rsa qui va permettre de chiffrer la clef de session avec la clef publique de client2.

Puis on va créer la fonction envoyer message qui prend comme paramètre un message et qui va en premier lieu vérifier si `message_recu` est égal à 1 afin d'envoyée le message, qui correspond à la clef de session chiffrer avec rsa, sans le montrer dans le terminal afin que l'envoi de la clef de session ne soit pas visible, puis passer `message_recu` à 2 afin de ne le faire qu'une seule fois. Sinon la fonction enverra le message chiffrer avec la fonction `enc_texte_cbc` de chiffrement.py. Ensuite on va faire l'initialisation de chatInterface afin de récupérer le message écrit dans la zone de texte de notre interface graphique.

Par la suite, nous allons définir la fonction recevoir message qui ne prend pas de paramètre. En premier lieu, la fonction va définir une boucle while qui est toujours vraie afin de vérifier continuellement s'il y a de nouveaux messages. Dans cette boucle while on a une condition qui vérifie si `message_recu` est à 1 afin de récupérer le message qui contient la clef publique de client2 et la stocker dans `rsa_p`, de chiffrer la clef de session avec la fonction rsa en utilisant `rsa_p` et de l'envoyer à client2. Sinon, on déchiffre le message avec `dec_texte_cbc` qui ensuite l'envoie à projet afin de l'afficher dans l'interface graphique. Par la suite on vérifie si le message correspond à quitter. Si c'est le cas on lance quitte chat du projet, on ferme la socket et on quitte le while.

A la fin du programme on crée les threads pour l'envoi et la réception de messages et on lance le thread et on le join.

## 3 Fonctionnement de `client2.py`

Le début du programme est très similaire à celui de client1, mais clef de session est une liste de deux 0 et à la place du message reçu, on a message envoyer, qui est initialisé à 0. Par la suite on crée la fonction `gen_rsa_keypair` qui va permettre de créer aléatoirement la clef publique et privée de client2.

Ensuite on a la même fonction rsa que client 1 qui va permettre de déchiffrer la clef de session que nous enverra client 1 par la suite.

La fonction envoyer message est exactement pareille que celle de client 1 sauf qu'au lieu de vérifier si `message_recu` est égal à 1, on vérifie si message envoyé est égal à 0. Ensuite on va faire l'initialisation de chatInterface afin de récupérer le message écrit dans la zone de texte de notre interface graphique.

La fonction `recevoir_message` est quasiment pareille que client1 sauf que l'on vérifie si `message_envoyer` est égal à 1 afin de mettre la clef de session que client1 nous a envoyé dans la variable `clef_session` qui nous servira à chiffrer et déchiffrer nos messages.

La fin du programme est pareille que client 1 sauf que l'on rajoute un if, après la création des thread, afin de vérifier si `envoyer_message` est égal à 0 afin d'envoyer la clef publique automatiquement au lancement du programme et d'ensuite passer `message_recu` à 1.

## 4 Fonctionnement de `serveur.py`

Au début du programme on définit la fonction communication qui va servir à envoyer le message reçu à l'autre client.

Dans la fonction on définit un while qui est toujours vrai, puis on crée un if qui vérifie si le message reçu correspond à "quitter" afin d'envoyer à l'autre client connecté que l'utilisateur s'est déconnecté, de le supprimer de la liste des clients connectés et de fermer la connexion du client. Sinon on va envoyer à l'autre client le message que l'on a reçu.

Puis on va créer la socket, lui associer l'adresse et le numéro de port et définir le nombre maximum de connexion possible à 2.

A la fin du programme on a créé un while qui sert à la connexion de nouveau client à la socket et quand il n'y a plus de client connecté la socket du serveur se ferme.

## 5 Fonctionnement de `projet.py`

Dans ce programme on définit une Class ChatInterface qui contient un callback qui permet de faire la liaison entre le client et l'interface graphique.

Dans cette Class on commence par créer la zone d'affichage ainsi que la zone de saisie qui nous permettra d'écrire nos messages, le bouton envoyer qui est relier à la fonction `envoie_message`, le bouton quitter qui est relier à la fonction `quitte_chat` et l'initialisation de la touche entrée qui permet aussi d'envoyer un message.

Puis on a plusieurs fonctions, tel qu'envoie message qui récupère le message saisi par l'utilisateur, qui le supprime de la zone de saisie et qui récupère l'heure et la date à laquelle le message est envoyé afin d'afficher le message sur l'interface graphique et qui envoie le message envoyer dans le fichier client grâce au callback.

Ensuite, on a la fonction `nouveau_message` qui permet d'afficher les messages reçus en récupérant la date et l'heure et affiche le message reçu dans l'interface graphique.

A la fin on a la fonction `quitte_chat` qui va désactiver la saisie de nouveaux messages ainsi que le bouton quitter et qui va envoyer le message quitter au client. Puis on va activer la zone d'affichage afin d'afficher le message " Le client a quitté la discussion. " et qui va désactiver la zone d'affichage.

## 6 Fonctionnement de `chiffrement.py`

Au début du programme on va initialiser sbox et nbox qui contiennent les boxs nécessaires au chiffrement et déchiffrement. Sbox sert au chiffrement et nbox au déchiffrement. Ensuite on a la fonction round qui va utiliser la valeur de k xor s dans la sbox et qui sera utilisé dans la fonction enc.

On définit la fonction enc qui prend une clef k et un message m. On va utiliser la fonction round sur la première partie de la clef ainsi que sur m. Ensuite on va refaire la même chose mais avec la deuxième partie de la clef et le résultat du premier round afin d'avoir fait 2 tours. Cette fonction permet de chiffrer une clef et un message.

Puis on a la fonction `back_round` qui va rechercher l'indice s dans la nbox et qui va retourner la valeur xor avec k qui est la clef.

Ensuite on définit la fonction dec qui est pareille que enc mais elle utilise `back_round` au lieu de round afin de déchiffrer le message.

Par la suite on va définir la fonction `enc_byte` qui permet de chiffrer un octet et qui a key et m comme paramètres. Tout d'abord on va diviser le message m en 2 nibble de 4 bits chacun, puis on va les chiffrer avec la fonction enc qui prend key et nibble1/nibble2 comme paramètre. On va ensuite reconcaténer les 2 nibbles ensemble et renvoyer le message chiffrer.

Ensuite, on va créer la fonction `dec_byte` qui permet de dechiffrer un octet et qui a key et m comme paramètres. La fonction `dec_byte` fonctionne pareil que `enc_byte` sauf qu'au lieu d'utiliser la fonction enc sur les nibbles on utilise la fonction dec.

Puis, on va créer la fonction `enc_texte_cbc` qui permet de chiffrer un message avec le chiffrement CBC. Elle prend 3 paramètres : k qui correspond à la clef, texte qui correspond au message que l'on va chiffrer, et vecteur qui va servir au chiffrement. On va initialiser la variable `texte_chiffre` qui va nous permettre de rajouter une à une les lettres que l'on va chiffrer afin d'obtenir le message complètement chiffré. On va aussi initialiser la variable `texte_chiff` égal à vecteur. Puis on va créer un for qui va parcourir chaque caractère de notre message et le transformer en bits, et qui va xor `texte_chiff` avec le caractère transformé en bits et qui sera stocké dans `texte_chiff`. Ensuite on va `enc_byte texte_chiff` avec la clef k et le stocker dans `texte_chiff`. Puis, la valeur de `texte_chiff` sera retransformée en caractères afin de l'ajouter à `texte_chiffre`. A la fin on renvoie le message chiffré.

A la fin du programme, on va créer la fonction `dec_texte_cbc` qui est quasiment pareille que `enc_texte_cbc` sauf que dans le for on utilise `dec_byte` sur k et le caractère transformé en bits que l'on stocke dans la variable `texte_dechiffre`, que l'on va xor avec `texte_avant` (qui remplace `texte_chiff` de `enc_texte_cbc`) et le re stocker dans `texte_dechiffre`. Puis on va transformer le résultat en caractères et l'ajouter dans `texte_dechiff` (qui correspond à `texte_chiff` dans `enc_texte_cbc`) et initialiser `texte_avant` au caractère de base transformé en bits. Puis, on renvoie le message déchiffré.