Documentation de Réponse au Projet

SAE 302 – Conception d’une architecture distribuée avec routage en oignon

Théo-Félix ADAM
BUT RT21 – DevCloud

Table des matières

1. ÉLÉMENTS

1.1. Éléments Implémentés

- Chiffrement RSA
- Enregistrement des clés RSA, chiffrée pour la clé privée
- Communications entre des clients sur des machines différentes, mais sur le même réseau
- Interface graphiques maître et client
- Le routage par des clients
- Convertisseur bases 64, avec 10 alphabets différents

1.2. Éléments Non Implémentés

- La mise en place de routeurs simples

2. STRUCTURE DU CODE, MODULES, PROTOCOLE, API

Le code comprend des descriptions des fonctions incorporées, de type Google Style Docstring.

J'ai utilisé les bibliothèques mariadb, socket, threading, sys et rotator, que j'ai codé.

Elle utilise le protocole TCP/IP pour la communication des messages.

L'API est réalisée avec Qt6.

3. DESCRIPTION DE L'ALGORITHME DE CHIFFRAGE

L'algorithme de chiffrage utilisé est celui mis en place par Rivest, Shamir et Adelman.

3.1. FORCES

3.1.1. Génération Des Nombres Premiers

Pour la génération de nombres premiers, l'algo utilise la classe **rds = random.SystemRandom()**, qui utilise os.urandom(). Ce choix a été fait, car la bibliothèque random utilise la Mersenne Twister pour générer des nombres pseudos-aléatoires. Or, la Mersenne Twister n'est pas sécurisé cryptographiquement, comme nous le fait remarquer la documentation : « Les générateurs pseudo-aléatoires de ce module ne doivent pas être utilisés à des fins de sécurité. »

Tandis que la classe rds est elle un générateur de nombres pseudos-aléatoires cryptographiquement sécurisé (CSPRNG). Pour tester la classe, j'ai utilisé un programme disponible sur GitHub : https://github.com/tna0y/Python-random-module-cracker, qui permet de prédire quelle prochaine valeur tombera.

serveur_v01.py | tests_random2.py

```
4  random.seed(time.time())
5
6  rc = RandCrack()
7
8  rds = random.SystemRandom()
9
10 """
11 for i in range(624):
12     rc.submit(rds.getrandbits(32))
13     # Could be filled with random.randint(0,4294967294) or random.randrange(0,4294967294)
14
15 print("Random System result: {}\nCracker result: {}"
16     .format(rds.randrange(0, 4294967295), rc.predict_randrange(0, 4294967295)))
17 """
18
19 for i in range(624):
```

Console

```
>>> %Run tests_random2.py
  Random result: 2100923845
  Cracker result: 2100923845
```

FIGURE 1: TEST GÉNÉRATEUR RANDOM

En utilisant le module random de base, le prédicteur arrive à trouver la valeur. Nous pouvons observer qu'il n'a juste besoin que de 624 nombres générés, pour prédire le suivant. Si j'avais utilisé cette classe, alors mes clés n'auraient pas été protégées.

En testant le module du système, le prédicteur n'arrive jamais à trouver la valeur qui sortira. J'ai réalisé plusieurs tests et voici ci-dessous l'un des tests.

serveur_v01.py × tests_random2.py ×

```python
import random, time
from randcrack import RandCrack

random.seed(time.time())

rc = RandCrack()

rds = random.SystemRandom()

for i in range(624):
    rc.submit(rds.getrandbits(32))
    # Could be filled with random.randint(0,4294967294) or random.randrange(0,4294967294)

print("Random System result: {}\nCracker result: {}"
    .format(rds.randrange(0, 4294967295), rc.predict_randrange(0, 4294967295)))

```

Console ×

```
>>> %Run tests_random2.py
  Random System result: 3722945007
  Cracker result: 2166408188
```

FIGURE 2: TEST GÉNÉRATEUR SYSTEM RANDOM

3.1.2. Enregistrement Des Clés

Les clés privées et publiques RSA sont enregistrées dans des fichiers avec l'extension PEM, pour Privacy Enhanced Mail, qui est l'extension généralisée des clés, certificats… Les clés publiques sont enregistrées en claire, tandis que les clés privées sont converties en utilisant la base 64 que j'ai conçu et un numéro d'alphabet choisi au hasard.

3.1.3. Attaque De Wiener

L'attaque de Wiener est possible, si $d < \frac{1}{3} n^{\frac{1}{4}}$. Or, j'ai une condition dans mon programme qui regénère les clés si cette condition est vraie.

3.1.4. Algo Rho De Pollard

Les clés sont générées de telle façon à ce que p et q soient très grands, tout comme delta tel que $\delta = |p - q|$.

3.1.5. Taille Des Messages Illimités

La taille des messages est illimitée. Normalement, la valeur du chiffré doit être inférieur au nombre modulo. Mais j'ai utilisé une fonction qui hache le message en plusieurs nombres plus petits, tous inférieurs à la valeur de n.

3.2. FAIBLESSES

3.2.1. Génération De Clés De 1024 Bits

Dans mon programme, il est possible de changer la taille des clés RSA. Néanmoins, pour mes tests et pour le programme tel qu'il est, les clés ont été plafonnées à 1024 bits.

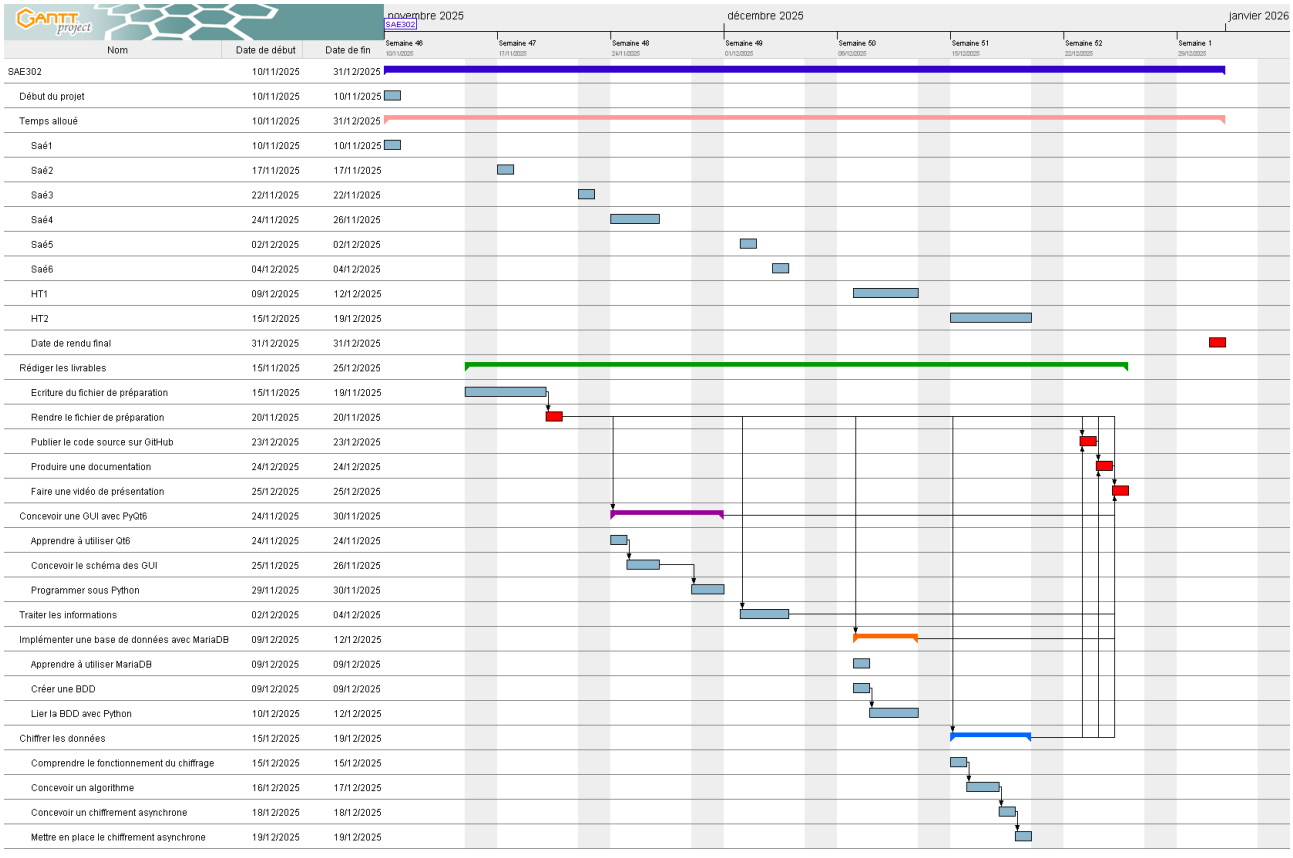
3.2.2. Attaque Temporelle

Possibilité d'utiliser l'attaque temporelle pour déterminer les clés RSA.

3.2.3. Enregistrement Des Clés

Les clés privées sont enregistrées avec un minimum de sécurité, mais ce n'est pas assez, car il n'y a que dix alphabets dans la fonction de conversion en base 64. Il est possible de brut force les chiffrés.

4. RAPPORT DE PROJET AVEC GESTION DU PROJET



Nom	Date de début	Date de fin
SAE302	10/11/2025	31/12/2025
Début du projet	10/11/2025	10/11/2025
Temps alloué	10/11/2025	31/12/2025
Saé1	10/11/2025	10/11/2025
Saé2	17/11/2025	17/11/2025
Saé3	22/11/2025	22/11/2025
Saé4	24/11/2025	26/11/2025
Saé5	02/12/2025	02/12/2025
Saé6	04/12/2025	04/12/2025
HT1	09/12/2025	12/12/2025
HT2	15/12/2025	19/12/2025
Date de rendu final	31/12/2025	31/12/2025
Rédiger les livrables	15/11/2025	25/12/2025
Ecriture du fichier de préparation	15/11/2025	19/11/2025
Rendre le fichier de préparation	20/11/2025	20/11/2025
Publier le code source sur GitHub	23/12/2025	23/12/2025
Produire une documentation	24/12/2025	24/12/2025
Faire une vidéo de présentation	25/12/2025	25/12/2025
Concevoir une GUI avec PyQt6	24/11/2025	30/11/2025
Apprendre à utiliser Qt6	24/11/2025	24/11/2025
Concevoir le schéma des GUI	25/11/2025	26/11/2025
Programmer sous Python	29/11/2025	30/11/2025
Traiter les informations	02/12/2025	04/12/2025
Implémenter une base de données avec MariaDB	09/12/2025	12/12/2025
Apprendre à utiliser MariaDB	09/12/2025	09/12/2025
Créer une BDD	09/12/2025	09/12/2025
Lier la BDD avec Python	10/12/2025	12/12/2025
Chiffrer les données	15/12/2025	19/12/2025
Comprendre le fonctionnement du chiffrage	15/12/2025	15/12/2025
Concevoir un algorithme	16/12/2025	17/12/2025
Concevoir un chiffrement asynchrone	18/12/2025	18/12/2025
Mettre en place le chiffrement asynchrone	19/12/2025	19/12/2025

Index des figures