# Ticpirsai

UE – Réseaux&Système
Enseignant – P-F Bonnefoi
Réalisation – Clément Hoffmann
Baptiste Beltzer

Réalisation d'un protocole de communication type "chat" basé TCP sécurisé par RSA.
Projet Git : https://github.com/Cirthy/TicpiRsai

## Réseau

Lors du démarrage du programme, une fois le chargement effectué, l'utilisateur·rice a la possibilité de choisir d'être client ou serveur. Le choix du mode client invite à entrer une adresse IP de connexion pour le port 8790 (imposé). Si la connexion est établie, les deux terminaux échangent leur clé publique. La fonction `def chat_run(socket)` engage alors l'échange des messages dans les deux sens.

Chaque message est découpé par blocs de 64 caractères et chiffré indépendamment des autres. Le dernier bloc envoyé contient uniquement le caractère `EXT (end of text)` afin de marquer la fin du message.

> Remarque :
> Nous sommes conscients que le non chainage du chiffrement peut présenter des failles de sécurité, cependant il nous semblait que ce n'était pas le sujet du projet. De plus, la probabilité que deux blocs de deux messages différents soit égaux est très faible (grâce à la grande taille des blocs).

La principale difficulté que nous avons rencontrée est de terminer proprement la transmission. Nous avons choisi un mot clé de demande d'arrêt de transmission. Un message contenant uniquement `quit` ou `exit` informe le receveur d'une demande d'arrêt, ce dernier envoie alors un message contenant uniquement `EOT (end of transmission)` et ferme sa connexion. Le receveur du `EOT` ferme a son tour sa connexion.

## RSA

Au démarrage, le programme va générer les clés nécessaires à l'utilisation du chiffrement RSA dans le fichier *config.py* :

- Les nombres premiers `p` et `q` ont respectivement 500 et entre 451 et 470 chiffres. On les choisit volontairement peu proches pour que notre `n = pq` ne devienne pas facilement factorisable (par exemple par l'algorithme de Fermat).
- Notre deuxième clé publique `e` vaut toujours 65537 (imposé), elle doit être première avec $\phi(n) = (p - 1)(q - 1)$ afin de pouvoir calculer son inverse modulaire.
- On finit par calculer la clé privée $d = e^{-1} \bmod \phi(n)$.

Pour le chiffrement par bloc, chaque chaine de 64 caractère est converti en chaîne d'octet via la méthode `.encode('UTF-8')`, puis en entier via la bibliothèque `binascii`.

La taille limitée de nos blocs nous garantit que ces entiers restent d'une taille inférieure à notre clé `n`, ce qui est nécessaire pour le chiffrement RSA. Ils sont ensuite chiffrés par la fonction `def encrypt(plain)`. Le chiffré est ensuite converti en chaîne d'octets pour être envoyé.

Les chaînes d'octets reçues sont reconverties en entiers puis déchiffrées par la fonction `def decrypt(cipher)`. Un buffer est alors rempli par les différents blocs reçus les uns après les autres jusqu'à la réception du `EXT`. Le message est finalement affiché à l'écran.

CDLT,
néanmoins bien à vous.