Projet H : Messagerie sécurisée de bout en bout par chiffrement symétrique AES 256 bits avec possibilité de transfert de fichiers sans limite de taille

Nous sommes parvenus à créer une messagerie instantanée en LAN (Local Area Network), sans serveur intermédiaire, parfaitement anonyme, chiffrée de bout en bout, utilisant l'un des protocoles actuels de chiffrement symétrique le plus rapide et le plus sûr : l'AES (ADVANCED ENCRYPTION STANDARD) 256 bits, certifié et très utilisé par la NSA.

Réalisation :
Notre projet fait appel à quatre principaux programmes présents dans les deux ordinateurs voulant communiquer (ce qui explique la présence de deux dossiers « pack 1 » et « pack 2 » contenant chacun ces quatre programmes) :

-code.py (pour chiffrer les fichiers à envoyer avant l'envoi sur l'autre ordinateur puis pour les déchiffrer après leur arrivée), composé de deux fonctions encrypt(nom_du_fichier, clé_de_chiffrement) et decrypt(nom_du_fichier, clé_de_déchiffrement). Dans le programme code.py nous utilisons le module Crypto afin de chiffrer nos données et nos fichiers à envoyer et le module os et sys pour lire nos fichiers en binaire puis écrire ces fichiers une fois déchiffrés.
-envoi.py pour envoyer les fichiers et des messages instantanés via le protocole TCP/IP. Il comprend notamment la fonction envoifichier(hôte,nom du fichier à envoyer). Cette fonction sera appelée par le client chat.py si l'utilisateur entre « envoyer fichier ».
Dans le programme envoi.py, le module socket sera utilisé pour envoyer les données chiffrées par le protocole TCP/IP et le module time pour obtenir la date et surtout l'heure des débuts de téléchargement.
Enfin nous utilisons également les deux programmes que nous exécuterons sur les deux machines pour obtenir une communication full-duplex (l'information peut être transportée simultanément dans les deux sens). La liaison full-duplex peut être comparée à une conversation téléphonique : les deux interlocuteurs peuvent parler en même temps… or sur un port le serveur et le client ne peuvent pas envoyer en même temps des informations, c'est pourquoi chaque machine ouvre un serveur et un client :

- serveur chat.py sert à ouvrir un serveur qui reçoit les informations que lui envoie le client chat.py du « pack 2 » de l'ordinateur distant.
-client chat.py sert à ouvrir un client qui envoie des messages ou des fichiers (si on tape « envoyer fichier ») que recevra le serveur chat.py du « pack 2 » de l'ordinateur distant.
Si ce réseau maillé est complexe à comprendre j'ai construit un schéma que j'ai mis à la fin du compte rendu. Il est très complet et montre toutes les interactions des différents programmes.

Python est un langage de haut niveau, ce qui fait qu'on a rarement besoin de taper directement dans les interfaces de couches basses comme les sockets : on utilise des abstractions comme urllib et consort.
Même quand on doit communiquer directement des paquets de données, on préférera utiliser des solutions comme ZeroMQ, bien plus fiables et faciles à mettre en œuvre.
Cela étant dit, pour la culture G, ça ne fait pas de mal de revenir aux bases des sockets en Python. De plus pour transmettre des données confidentielles il vaut mieux être certain de la fiabilité du matériel de communication c'est pourquoi nous avons choisi d'utiliser des sockets et non des libraires existantes à tout va… Dans notre projet on en utilise qu'une seule : Crypto et on peut difficilement s'en passer car le chiffrement par inversion matricielle est un peu trop ambitieux à notre niveau… ☺

La première difficulté de préparation de notre programme fut l'importation du module Crypto. Il n'était pas reconnu par python… Il nous a fallu remonter au dossier des librairies python et comprendre que python ne reconnaissait pas crypto car il était écrit avec un petit « c » au lieu d'un grand « C ». Erreur bête mais visiblement personne n'avait reporté le problème sur Internet ; c'est qu'après une petite heure d'acharnement que l'idée nous est venue.

```
Last login: Wed Dec 18 20:31:50 on ttys000
Louis-Alexandre@macbookpro-ac87a30960d0-1 ~ % pip3 install Crypto
Collecting Crypto
  Using cached https://files.pythonhosted.org/packages/fc/bb/0b812dc02e6357606228edfbf5808f5ca0a675a84273578c3a199e841cd8/crypto-1.4.1-py2.py3-none-any.whl
Collecting shellescape (from Crypto)
  Using cached https://files.pythonhosted.org/packages/51/b6/986c99a10040beaaefca1ad6c93bd7738cb8e4f52f6caed13d3ed1caa7e4/shellescape-3.4.1-py2.py3-none-any.whl
Collecting Naked (from Crypto)
  Using cached https://files.pythonhosted.org/packages/02/36/b8107b51adca73402ec1860d88f41d958e275e60eea6eeaa9c39ddb89a40/Naked-0.1.31-py2.py3-none-any.whl
Collecting requests (from Naked->Crypto)
  Using cached https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89d72b2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl
Collecting pyyaml (from Naked->Crypto)
  Downloading https://files.pythonhosted.org/packages/8d/c9/e5be955a117a1ac548cdd31e37e8fd7b02ce987f9655f5c7563c656d5dcb/PyYAML-5.2.tar.gz (265kB)
     |████████████████████████████████| 266kB 9.1MB/s
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 (from requests->Naked->Crypto)
  Downloading https://files.pythonhosted.org/packages/b4/40/a9837291310ee1ccc242ceb6ebfd9eb21539649f193a7c8c86ba15b98539/urllib3-1.25.7-py2.py3-none-any.whl (125kB)
     |████████████████████████████████| 133kB 17.5MB/s
Collecting idna<2.9,>=2.5 (from requests->Naked->Crypto)
  Using cached https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl
Collecting chardet<3.1.0,>=3.0.2 (from requests->Naked->Crypto)
  Using cached https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl
Collecting certifi>=2017.4.17 (from requests->Naked->Crypto)
  Downloading https://files.pythonhosted.org/packages/b9/63/df50cac98ea0d5b006c55a399c3bf1db9da7b5a24de7890bc9cfd5dd9e99/certifi-2019.11.28-py2.py3-none-any.whl (156kB)
     |████████████████████████████████| 163kB 20.4MB/s
Building wheels for collected packages: pyyaml
  Building wheel for pyyaml (setup.py) ... done
  Created wheel for pyyaml: filename=PyYAML-5.2-cp37-cp37m-macosx_10_15_x86_64.whl size=44215 sha256=d5ed4717730693470b6f32fe029015380fb34a3382571d9e993e8c39094748af
  Stored in directory: /Users/Louis-Alexandre/Library/Caches/pip/wheels/54/b7/c7/2ada654ee54483c9329871665aaf4a6056c3ce36f29cf66e67
Successfully built pyyaml
Installing collected packages: shellescape, urllib3, idna, chardet, certifi, requests, pyyaml, Naked, Crypto
Successfully installed Crypto-1.4.1 Naked-0.1.31 certifi-2019.11.28 chardet-3.0.4 idna-2.8 pyyaml-5.2 requests-2.22.0 shellescape-3.4.1 urllib3-1.25.7
Louis-Alexandre@macbookpro-ac87a30960d0-1 ~ %
```

Après avoir renommé notre librairie a été reconnue (pas d'erreur) :

```
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> import Crypto
>>>
```

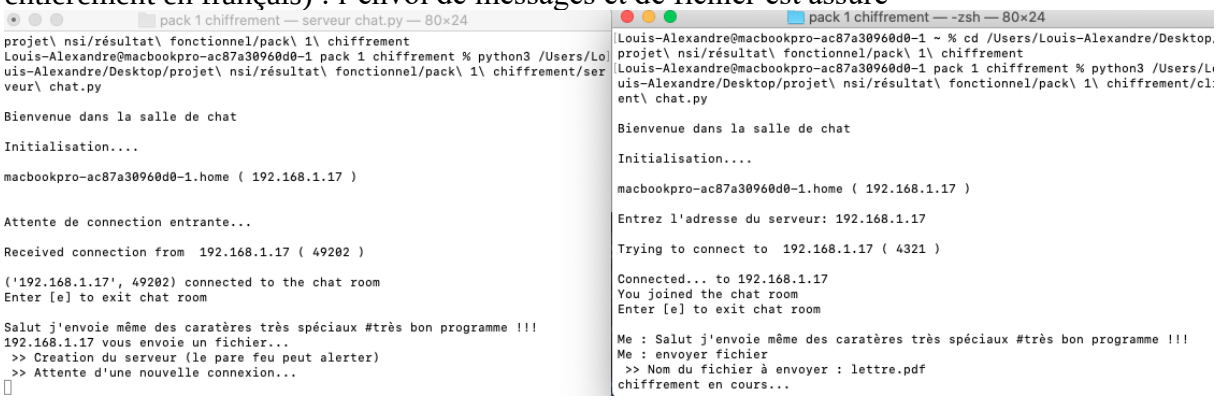
Une chose également très pénible était que nos programmes ne fermaient pas forcément les ports ouverts lorsqu'ils rencontraient des erreurs… Au redémarrage de nos programmes nous obtenions donc « socket.error: [Errno 98] Address already in use » car il fallait le temps que les port se ferment par l'absence d'utilisation. Environ 30 secondes entre chaque test, le try/except fut un excellant outil :

```
import socket
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try :
        socket.connect((host, port))
        truc qui provoque possiblement une erreur
        socket.close() # si toutefois il n'y avait pas d'erreur
except :
        socket.close()
```

Mis à part ces quelques erreurs qui nous prirent pas mal de temps nous nous en sommes toujours sorti avec l'aide d'internet et quelques dizaines de minutes de réflexion.

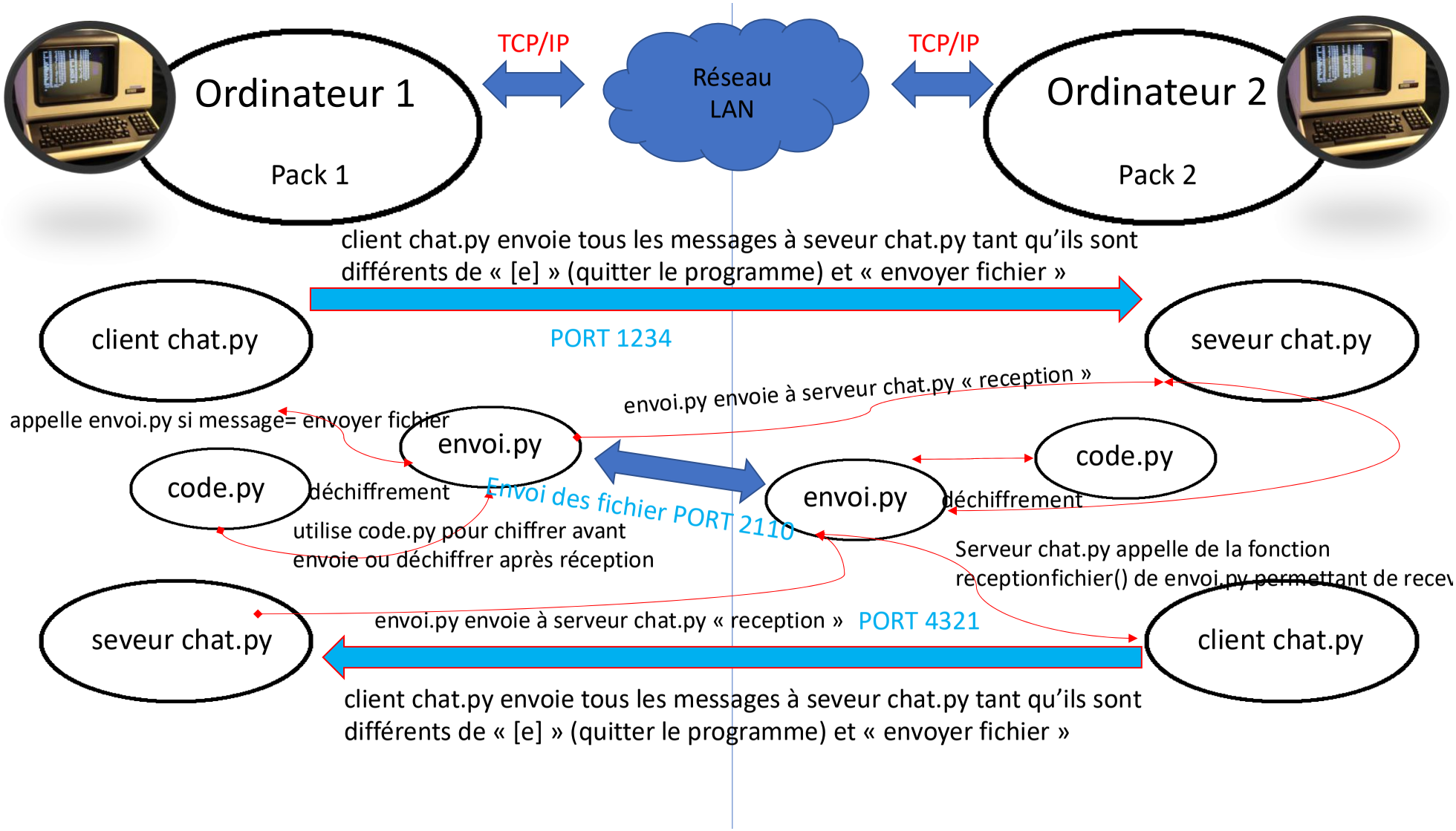
T

Voici un test sur un même ordinateur (la dernière version de nos programmes est normalement traduite entièrement en français) : l'envoi de messages et de fichier est assuré



```
pack 1 chiffrement — serveur chat.py — 80×24
projet\ nsi/résultat\ fonctionnel/pack\ 1\ chiffrement
Louis-Alexandre@macbookpro-ac87a30960d0-1 pack 1 chiffrement % python3 /Users/Lo
uis-Alexandre/Desktop/projet\ nsi/résultat\ fonctionnel/pack\ 1\ chiffrement/ser
veur\ chat.py

Bienvenue dans la salle de chat

Initialisation....

macbookpro-ac87a30960d0-1.home ( 192.168.1.17 )


Attente de connection entrante...

Received connection from  192.168.1.17 ( 49202 )

('192.168.1.17', 49202) connected to the chat room
Enter [e] to exit chat room

Salut j'envoie même des caratères très spéciaux #très bon programme !!!
192.168.1.17 vous envoie un fichier...
 >> Creation du serveur (le pare feu peut alerter)
 >> Attente d'une nouvelle connexion...
```

```
pack 1 chiffrement — -zsh — 80×24
Louis-Alexandre@macbookpro-ac87a30960d0-1 ~ % cd /Users/Louis-Alexandre/Desktop/
projet\ nsi/résultat\ fonctionnel/pack\ 1\ chiffrement
Louis-Alexandre@macbookpro-ac87a30960d0-1 pack 1 chiffrement % python3 /Users/Lo
uis-Alexandre/Desktop/projet\ nsi/résultat\ fonctionnel/pack\ 1\ chiffrement/cli
ent\ chat.py

Bienvenue dans la salle de chat

Initialisation....

macbookpro-ac87a30960d0-1.home ( 192.168.1.17 )

Entrez l'adresse du serveur: 192.168.1.17

Trying to connect to  192.168.1.17 ( 4321 )

Connected... to 192.168.1.17
You joined the chat room
Enter [e] to exit chat room

Me : Salut j'envoie même des caratères très spéciaux #très bon programme !!!
Me : envoyer fichier
 >> Nom du fichier à envoyer : lettre.pdf
chiffrement en cours...
```


Ordinateur 1
Pack 1
TCP/IP
Réseau
LAN
TCP/IP
Ordinateur 2
Pack 2
client chat.py envoie tous les messages à seveur chat.py tant qu'ils sont différents de « [e] » (quitter le programme) et « envoyer fichier »
client chat.py
PORT 1234
seveur chat.py
envoi.py envoie à serveur chat.py « reception »
appelle envoi.py si message= envoyer fichier
envoi.py
code.py
déchiffrement
Envoi des fichier PORT 2110
envoi.py
code.py
déchiffrement
utilise code.py pour chiffrer avant
envoie ou déchiffrer après réception
Serveur chat.py appelle de la fonction
receptionfichier() de envoi.py permettant de recev
seveur chat.py
envoi.py envoie à serveur chat.py « reception »
PORT 4321
client chat.py
client chat.py envoie tous les messages à seveur chat.py tant qu'ils sont différents de « [e] » (quitter le programme) et « envoyer fichier »