I. Sujet du projet

Le but du projet est de réaliser un simulateur de protocole HTTPS pour échanger des messages chiffrés.
Pour cela deux classes sont à disposition : une classe `Connexion` qui gère la partie physique de la connexion, et une classe `Interface` qui servira de base aux classes à réaliser.
L'idée est d'échanger des messages entre deux ordinateurs de la salle, un « client » et un « serveur ».
Les différentes étapes demandées suivent le schéma vu en cours rappelé figure 1.

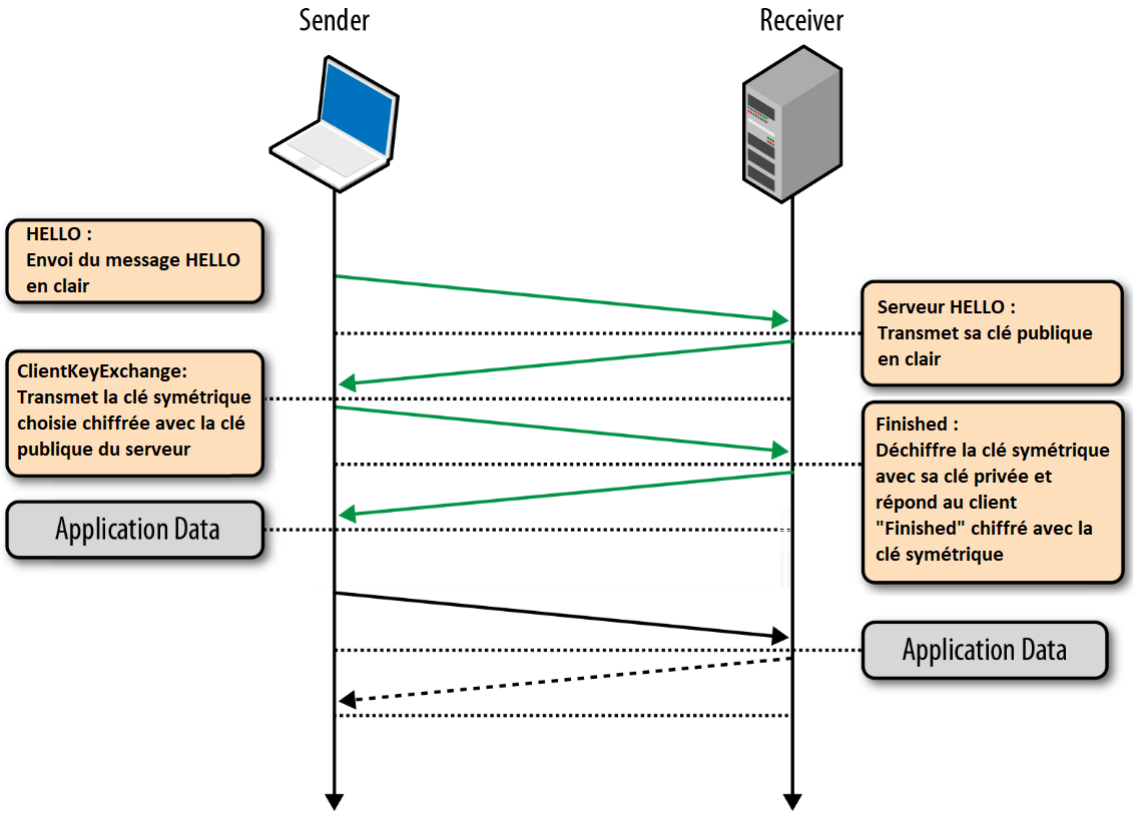

FIGURE 1 – Protocole d'échange de clé.

II. A réaliser

Par groupe de 2 il faudra :

InterfaceServeur.py Créer :

- un bouton « Serveur Hello » qui attend la réception du message `HELLO` depuis le client, puis envoie en clair sa clé publique ;
- un bouton « Finished » qui attend la réception de la clé symétrique, la déchiffre avec sa clé privée, la mémorise et envoie le message `Finished` chiffré avec cette clé.

InterfaceClient.py Créer :

- un bouton « Hello » qui envoie en clair le message `HELLO` ;
- un bouton « KeyExchange » qui
  - attend la clé publique du serveur ;
  - choisie une clé symétrique aléatoire de 7 caractères ;
  - la transmet en la chiffrant avec la clé publique du serveur ;
  - attend le message `Finished` du serveur chiffré avec cette clé.

Interface.py Compléter :

- la méthode `send` qui envoie le message saisi dans l'`Entry`, en le chiffrant avec la clé symétrique ;
- la méthode `receive` qui attend la réception d'un message, qui le déchiffre avec la clé symétrique, et que l'affiche dans le `Label`.

Il faudra bien sûr associer les méthodes correspondantes aux boutons ainsi créés. Il faudra aussi écrire les différentes fonctions de chiffrement et déchiffrement.

Remarques :

- Pour le bon fonctionnement, il faut dans l'ordre :
  1. configurer l'adresse IP du serveur dans le script `Connexion.py` sur les deux machines ;
  2. lancer le script `InterfaceServeur.py` sur la machine qui jouera le rôle de serveur ;
  3. lancer le script `InterfaceClient.py` sur la machine qui jouera le rôle de client.
- Les messages envoyés et reçus doivent être des chaînes d'octets (`bytes`).
  Pour obtenir une telle chaîne, on peut :
  - Ajouter un `b` avant les guillemets : `message = b"HELLO";`
  - Utiliser la méthode `encode` des chaînes de caractères (surtout utile s'il y a des accents) :
    `message = "noël à pâques".encode("utf8");`
  - Utiliser la méthode `to_bytes` des entiers. Si `n` est un entier : `message = n.to_bytes("big")`
    Le `big` correspond à un problème de boutisme [1].
  - Convertir directement une liste d'octets : `message = bytes([72, 69, 76, 76, 79])`

  Réciproquement, pour retrouver
  - une chaîne de caractères à partir d'une chaîne d'octets, on utilise la méthode décode :
    `chaine = message.decode("utf8");`
  - un entier à partir d'une chaîne d'octets, on utilise la méthode `from_bytes` :
    `n = int.from_bytes(message, "big");`
- L'échange de message pour fixer la clé publique se fera en utilisant le système **RSA**.
  On prendra comme clé publique du serveur le couple $n = 870567544966633301$ et $e = 42205$.
  La clé privée sera $d = 769020235650503533$.
  On rappelle les formules pour RSA, si m est le message en clair et c le message chiffré on a :
  Pour chiffrer : $c = m^e$ (modulo n)
  Pour déchiffrer : $m = c^d$ (modulo n)
- Pour le calcul de puissance on utilisera l'algorithme de puissance rapide vu en cours, en l'adaptant pour qu'il calcule la puissance modulo n.
- Pour le chiffrement avec la clé symétrique on utilisera le XOR. La clé symétrique fera 7 caractères.

III. Pour aller plus loin

En vous servant d'un serveur intermédiaire, vous pouvez simuler l'attaque de l'homme du milieu.

1. voir https://fr.wikipedia.org/wiki/Boutisme

IV. Annexe : scripts de départ

```python
# -*- coding: utf-8 -*-

import socket

IP_SERVEUR = '127.0.0.1'
PORT = 42000


class Connexion():
    def __init__(self, sens):
        """Si sens == "serveur" ouvre une connexion côté serveur,
        sinon ouvre une connexion côté client."""
        mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        if sens == "serveur":
            mySocket.bind((IP_SERVEUR, PORT))
            print(f"Serveur({IP_SERVEUR}:{PORT}) prêt. Attend un client...")
            mySocket.listen()
            self.connexion, adresse = mySocket.accept()
            print(f"Connecté avec le client({adresse}).")
        else:
            print(f"Connexion au serveur({IP_SERVEUR}:{PORT})...")
            mySocket.connect((IP_SERVEUR, PORT))
            print("Connexion établie avec le serveur")
            self.connexion = mySocket

    def send(self, message):
        """Envoie le message, et l'affiche."""
        print(f"Envoi de <{message}>.")
        self.connexion.send(message)

    def recv(self):
        """Reçoit un message d'au plus 1024 octets, l'affiche et le renvoie.
        Attention, la réception est bloquante : tant qu'il n'y a pas de message
        à recevoir, le programme attend."""
        message = self.connexion.recv(1024)
        print(f"Reçu <{message}>.")
        return message

    def close(self):
        """Ferme la connexion."""
        print("Fermeture de la connexion.")
        self.connexion.close()
```

Listing 1 – Connexion.py

```python
# -*- coding: utf-8 -*-

from Connexion import Connexion
import tkinter as tk


```

```python
class Interface(tk.Tk):
    """ interface  graphique pour envoyer/ recevoir / chiffrer / dechiffrer """
    def __init__(self, sens):
        tk.Tk.__init__(self)
        self.connexion = Connexion(sens)
        self.title(sens)
        self.entree = tk.StringVar()
        tk.Entry(self, textvariable=self.entree, width=30).pack()
        tk.Label(text="Message reçu : ").pack()
        self.message = tk.StringVar()
        tk.Label(textvariable=self.message).pack()
        self.sendButton = tk.Button(self, text="Send", command=self.send)
        self.sendButton.pack()
        self.recvButton = tk.Button(self, text="Receive", command=self.receive)
        self.recvButton.pack()

    def send(self):
        """Envoie  le  message  chiffr é. """
        message = self.entree.get().encode("utf8")
        # à compléter  pour  chiffrer  le  message
        self.connexion.send(message)

    def receive(self):
        """Réceptionne  un message depuis  le  serveur  et  l' affiche .
        Attention  :  bloque  tant  que  le  serveur  n'envoie  rien . """
        message = self.connexion.recv()
        # à compléter  pour dé chiffrer  le  message
        self.message.set(message.decode("utf8"))

    def destroy(self):
        self.connexion.close()
        tk.Tk.destroy(self)
```

Listing 2 – Interface.py

```python
# -*- coding:  utf-8 -*-

import tkinter as tk
from Interface import Interface


class InterfaceServeur(Interface):
    def __init__(self):
        Interface.__init__(self, "serveur")
        self.n = 870567544966633301
        self.e = 42205
        self.d = 769020235650503533
        # à compléter


It = InterfaceServeur()
It.mainloop()
```

Listing 3 – InterfaceServeur.py

```
# -*- coding: utf-8 -*-

import tkinter as tk
from Interface import Interface


class InterfaceClient(Interface):
    def __init__(self):
        Interface.__init__(self, "client")
        # à compléter


It = InterfaceClient()
It.mainloop()
```

Listing 4 – InterfaceClient.py