Rapport du Projet de Réseaux et Systèmes

A l'intention de Monsieur Bros Maxime

Membres du groupe : Dupont Rémi et Fauriat Mattieu

Promotion du Master Mathématiques CRYPTIS 2020-2021

1 – Présentation du sujet : Conception d'un protocole de communication basé TCP sécurisé par RSA

Le but ici est de simuler une communication basée TCP sécurisée par RSA entre un client et un serveur sur un réseau.

Pour réaliser ce projet nous avons d'abord commencé par recopier les fonctions mises à disposition dans le sujet, puis nous avons créer quelques fonctions utiles pour le chiffrage RSA, telle que la fonction « decoupage » (voir ci-dessous) qui comme son nom l'indique servira à découper le message en plusieurs morceaux de même longueur.

```
63  def decoupage(message, l):
64          #fonctions pour découper un message en morceaux de longueur l
65          #problème si len(message) n'est pas multiple de l
66          n = len(message)
67          if n%l !=0:
68                  message = message + (l - n%l)*"X" #padding avec des X
69                  n = n + l - n%l
70          Liste_message = []
71          q = n//l
72          for i in range(q):
73                  Liste_message.append(message[i*l : l+i*l])
74          return Liste_message
```

2 – Déroulement de notre travail

Nous avons commencé par RSA, en simulant un échange de nombres entiers entre Alice et Bob :

- Grâce à la méthode suggérée dans le sujet, Alice doit générer 2 nombres premiers p et q.
- Alice calcule $n = p * q$ ainsi que $\varphi(n) = (p-1)(q-1)$ où φ est l'indicatrice d'Euler.
- Alice calcule aussi $d = e^{-1}\ [\varphi(n)]$ qui sera sa clef privée.
- Alice envoie à Bob la clef publique (e, n).
- Bob choisi son message M et le chiffre, grâce à la clef que lui a envoyée Alice, avec l'opération suivante : $C = M^e$.
- Bob transmet le chiffré C à Alice.
- Alice déchiffre le message en effectuant le calcul suivant $C^d\ [n] = M$.

Jusqu'ici nous n'avons rencontré aucune difficulté dans la programmation car la plupart des fonctions à utiliser nous ont été données.

Ensuite, en nous aidant du cours nous avons essayé de coder un programme serveur qui représentera Bob et un programme client qui représentera Alice. Nous avons donc mis en

place un système de communication qui implique que le premier message envoyé par Alice ne contient que la clef n. Pour cela nous avons mis en place un compteur qui s'incrémente pour chaque message envoyé et donc si le compteur est à 0 c'est qu'il s'agit du premier message.

Voici un screenshot du programme serveur, ainsi que du programme client :

- voici Bob :

```
import socket, sys, os
from fonctions_projet import *

adresse_ip = 'localhost'
port = 8790
temps_attente = 15 #nombre de secondes d'attente de connexion
e = 65537


socquette = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP) # ouverture du socket
socquette.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
socquette.bind((adresse_ip, port))
socquette.listen(temps_attente) #attente de connexion

connexion, TSAP_depuis = socquette.accept()
cpt = 0
while True:
    if cpt == 0:
        print("Premiere connexion depuis : " , TSAP_depuis)
        cle=connexion.recv(1024)
        cle= cle.decode('utf-8')
        if cle !="":
            n = int(cle)
            print ("la clé est n = ", n, "\n")
            M =int(input("Saisissez votre message : \n"))
            C = powmod(M,e,n)
            C = str(C).encode('utf-8')
            connexion.sendall(bytes(C))
            cpt += 1
socquette.close()
```

- et voici Alice

```
import socket, sys, os
from fonctions_projet import *

adresse_serveur = socket.gethostbyname('localhost')
porc = 8790
socquette = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    socquette.connect((adresse_serveur, porc))
except Exception as e:
    print("On a un problème", e.args)
    sys.exit(1)


p = nombre_premier(3)
q = nombre_premier(4)
n= p * q
phi_n = (p-1) * (q-1)
e = 65537
d = modinv(e, phi_n)
cpt = 0



while True:
    if cpt == 0:
        message = str(n).encode('utf-8')
        socquette.sendall(bytes(message))
        chiffre = socquette.recv(1024)
        if not chiffre:
            break
        print(chiffre, 'Reçu')
        chiffre = int(chiffre.decode('utf-8'))
        print(chiffre)
        clair = powmod(chiffre, d, n)
        print(clair)
        cpt += 1
    #message = (input("Entrez un message à envoyer\n")).encode("utf-8")


socquette.close()
```

Dans l'état actuel, ces programmes ne peuvent que chiffrer qu'un échange d'entiers. Ils fonctionnent comme suit :

- on lance en premier le serveur, puis le client

- Alice crée immédiatement sa paire de clefs et envoie à Bob la clef publique n.
- Bob reçoit la clef, écrit le message qu'il veut envoyer (dans la cas présent un entier).
- Ensuite il chiffre son message avec la clef qu'il vient de recevoir.
- Il l'envoie à Alice
- Alice réceptionne le chiffré, et utilise sa clef privée pour obtenir le clair de base

Ensuite nous sommes passés à la manière de chiffrer des chaines de caractères. L'idée est la suivante : l'utilisateur (Bob) saisit au clavier une chaine de caractère. La chaîne de caractère est ensuite encodée en UTF-8. On peut ensuite convertir cette chaine de bytes en un entier avec la fonction $« entier = int.from\_bytes(bytes, byteorder = 'big') ».$ Ainsi nous pouvons manipuler l'entier pour effectuer les calculs nécessaires aux chiffrements RSA. Puis Bob reconverti l'entier trouvé en bytes grâce à la commande « $C = C.to\_bytes(C.bit\_length() //8 + 1, byteorder = 'big')$ » La chaîne de bytes ainsi chiffrée est envoyé à Alice.

Maintenant Alice reconverti la chaine de bytes en un entier, retrouve l'entier originelle puis reconvertit cet entier en une chaîne de bytes toujours avec les commandes précédentes. La chaine de bytes peut à présent décodée. Si tout s'est bien passée Alice retrouve le message que Bob lui a envoyé.

Voici une nouvelle fois les screenshots des programmes :

- Bob :

```python
#!/usr/bin/python3

import socket, sys, os
from fonctions_projet import *

adresse_ip = 'localhost'
port = 8790
temps_attente = 15 #nombre de secondes d'attente de connexion
e = 65537


socquette = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP) # ouverture du socket
socquette.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
socquette.bind((adresse_ip, port))
socquette.listen(temps_attente) #attente de connexion

connexion, TSAP_depuis = socquette.accept()
cpt = 0
while True:
    if cpt == 0:
        print("Premiere connexion depuis : " , TSAP_depuis)
        cle=connexion.recv(1024)
        cle= cle.decode('utf-8')
        if cle !="":
            n = int(cle)
            print ("la clé est n = ", n, "\n")
            M =input("Saisissez votre message : \n")
            M=M.encode('utf-8')
            C=int.from_bytes(M, byteorder='big')
            C = powmod(C,e,n)
            C=C.to_bytes(C.bit_length()//8 + 1, byteorder='big')
            #C = str(C).encode('utf-8')
            connexion.sendall(bytes(C))
            cpt += 1
socquette.close()
```

- Et Alice

```python
import socket, sys, os
from fonctions_projet import *

adresse_serveur = socket.gethostbyname('localhost')
porc = 8790
socquette = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    socquette.connect((adresse_serveur, porc))
except Exception as e:
    print("On a un problème", e.args)
    sys.exit(1)


p = nombre_premier(3)
q = nombre_premier(4)
n= p * q
phi_n = (p-1) * (q-1)
e = 65537
d = modinv(e, phi_n)
cpt = 0



while True:
    if cpt == 0:
        message = str(n).encode('utf-8')
        socquette.sendall(bytes(message))
        chiffre = socquette.recv(1024)
        if not chiffre:
            break
        print(chiffre, 'Reçu')
        m = int.from_bytes(chiffre,byteorder='big')
        print(m)
        clair = powmod(m, d, n)
        clair = clair.to_bytes(m.bit_length()//8+1, byteorder='big')
        clair=clair.decode()
        print(clair)
        cpt += 1
    #message = (input("Entrez un message à envoyer\n")).encode("utf-8")


socquette.close()
```

3 – Détails des fonctions

Fonction découpage

```python
def decoupage(message, l):
        #fonctions pour découper un message en morceaux de longueur l
        #problème si len(message) n'est pas multiple de l
        n = len(message)
        if n%l !=0:
                message = message + (l - n%l)*"X" #padding avec des X
                n = n + l - n%l
        Liste_message = []
        q = n//l
        for i in range(q):
                Liste_message.append(message[i*l : l+i*l])
        return Liste_message
```

Cette fonction prend en entrée une chaîne de caractères M et un entier l et ressort une liste qui contient la chaîne initiale découpée en morceaux de longueur l. De plus si la longueur de la chaîne n'est pas un multiple de l, la fonction complète la chaîne en faisant du padding avec des X de sorte que tous les bouts soient de même longueur.

Fonction chiffrage

```
76  def chiffrage(message,e,n):
77          M=message.encode('utf-8')
78          C=int.from_bytes(M, byteorder='big')
79          C = powmod(C,e,n)
80          C=C.to_bytes(C.bit_length()//8 + 1, byteorder='big')
81          return C
82
```

Cette fonction prend en argument une chaine de caractère M et renvoie une chaîne de bytes C chiffrée par RSA. Pour cela on encode M avec le format UTF-8, on transforme la chaîne en un entier, on chiffre l'entier avec la méthode RSA et enfin on reconvertit l'entier en une chaîne de bytes pour pouvoir la transmettre.

Fonction déchiffrage

```
83  def dechiffrage(chiffre,d,n):
84          m = int.from_bytes(chiffre,byteorder='big')
85          clair = powmod(m, d, n)
86          clair = clair.to_bytes(m.bit_length()//8+1, byteorder='big')
87          clair=clair.decode()
88          return clair
```

Cette fonction marche à l'inverse de la précédente. Elle prend en argument une chaîne de bytes, et renvoie une chaîne de caractères. Pour cela, elle transforme la chaîne de bytes en un entier, applique la fonction réciproque du RSA (c'est-à-dire l'inversion modulo n), transforme cet entier en une chaîne de bytes que l'on pourra ensuite décoder avec le format UTF-8.

Fonction nombre_premier

Cette fonction prend en argument un nombre entier et renvoie un nombre premier selon le protocole imposé dans le sujet. L'argument détermine la taille de la seed utilisée pour la génération.

```python
def nombre_premier(n):
        n_0=random.choice('1379')
        n_max=random.choice('123456789')

        n_i=n_0

        for i in range(1,n-1):
                n_i=n_i+random.choice('0123456789')

        n_i=n_i+n_max
        p=int(n_i)

        commande=subprocess.run("openssl prime %d 2> /dev/stdout"%p, shell=True, stdout=subprocess.PIPE)
        mon_exp_reguliere=re.compile(r"is not prime")

        resultat=mon_exp_reguliere.search(str(commande))

        while(resultat):
                n_i=n_i[1:]+random.choice('0123456789')+random.choice('1379')
                p=int(n_i)
                commande=subprocess.run("openssl prime %d 2> /dev/stdout"%p, shell=True, stdout=subprocess.PIPE)
                resultat=mon_exp_reguliere.search(str(commande))
        return p
```