Faculté des Sciences & Techniques — Université de Limoges

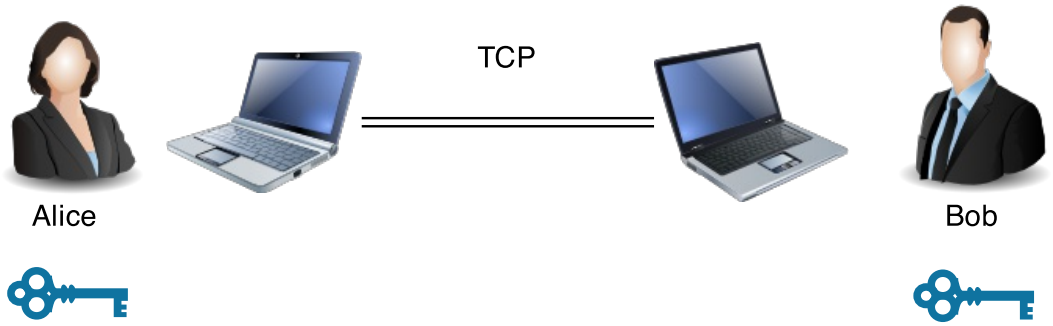
Master 1ère année

Réseaux & Système

Projet

Conception d'un protocole de communication basé TCP sécurisé par RSA

Présentation du protocole



Description du protocole :

- Alice et Bob vont exécuter chacun la version « Serveur » du programme :
  - le programme détermine les paramètres d'une biclé de chiffrement asymétrique basé sur RSA : les valeurs e, d et n (la taille de ces paramètres est exprimée en nombre de chiffre en base 10) ;
  - le programme se comporte comme un serveur TCP attendant sur le port n°8790 ;
- lors de la connexion du programme « client », l'échange est le suivant :

Alice		Bob
Détermine n_A, d_A		Détermine n_B, d_B
$e_A = 65537$		$e_B = 65537$
Client TCP		Serveur TCP
	Établie la connexion TCP →	
	Envoi n_A →	
		Chiffre$(m, e_A, n_A) = c$
	← Envoi c	
Déchiffre$(c, d_A, n_A) = m$		
Affiche m		

- les échanges entre Alice et Bob sont chiffrés respectivement avec la clé publique de Bob et d'Alice : chaque interlocuteur doit envoyer ses paramètres n_x à l'autre (le paramètre $e = 65537$ est connu).

Présentation de RSA

- ⋆ trouver deux nombres premiers p et q ;
- ⋆ calculer $n = pq$;
- ⋆ calculer $\phi(n) = (p-1)(q-1)$;
- ⋆ e est appelé « exposant de clé publique », il est choisi de manière standardisée $e = 65537$;
- ⋆ d est appelé « exposant de clé privée », il est choisi tel que $ed \equiv 1 \pmod{\phi(n)}$;

Le chiffrement d'une valeur m est obtenu par l'opération :

$$\text{chiffrement}(m) = m^e \pmod{n} = c$$

Le déchiffrement d'une valeur c est obtenu par l'opération :

$$\text{déchiffrement}(c) = c^d \pmod{n} = m$$

Pour envoyer un message m chiffré à un interlocuteur, il faut connaître e et son choix de n.

Génération de nombre premier de grande taille comportant un nombre choisi de chiffres

Cette génération va se dérouler en différentes étapes :

1. génération de manière aléatoire d'un nombre entier **candidat** de taille choisie (nombre de chiffres qui le composent) :

$n_{\max}$	$n_{\max-1}$	...	n_1	n_0

avec $0 < i <$ max, où max correspond aux nombre de chiffres choisi

Les chiffres n_i sont choisis de la manière suivante :

- $n_{\max} \in \{1,2,3,4,5,6,7,8,9\}$
- $n_i \in \{0,1,2,3,4,5,6,7,8,9\}$ avec $1 < i < \max - 1$
- $n_0 \in \{1,3,7,9\}$

On utilisera la fonction `random.choice('1379')` pour choisir de manière aléatoire un caractère parmi la liste fournie.

```
1 import random
2 random.seed()
3 caractere_aleatoire = random.choice('1379')
```

On considérera chaque chiffre comme un caractère, et on utilisera au besoin la fonction de conversion `int()` pour obtenir la valeur entière correspondante.

2. test de « primalité », c-à-d savoir si le nombre candidat est premier :
   - dans le cas où il **est premier**, on a réussi et on utilise le nombre trouvé ;
   - dans le cas où il **n'est pas premier**, il faudra essayer une **nouvelle valeur aléatoire** en réutilisant au maximum le tirage initial afin de limiter l'utilisation de la fonction de tirage aléatoire, consommatrice d'entropie :

$n_{\max}$	$n_{\max-1}$	...	n_1	n_0
		⇓		
$n_{\max-1}$	...	n_1	n_a	n_b

où n_a et n_b sont deux nouveaux chiffres choisis de manière aléatoire avec :

⋆ $n_a \in \{0,1,2,3,4,5,6,7,8,9\}$;
⋆ $n_b \in \{1,3,7,9\}$.

L'idée est de procéder à un décalage vers la gauche de la valeur précédente et de changer le « biais » introduit sur le dernier chiffre précédemment choisi.

Pour le test de « primalité » on utilisera la commande externe `openssl` :

```
pef@darkstar:~ openssl prime 13
D is prime
pef@darkstar:~ openssl prime 12
C is not prime
```

On notera que la commande retourne la valeur entrée en notation hexadécimale avec la mention `is prime` ou `is not prime`.

On utilisera la possibilité offerte par le module `subprocess` de lancer une commande externe et d'obtenir son résultat :

```
1 import subprocess
2 commande = "openssl prime "
3 n = 13
4 r  = subprocess.run(commande+str(n),shell=True,stdout=subprocess.PIPE) #
5 Récupère la sortie d'une commande
6 resultat_openssl = r.stdout
```

Il ne restera plus qu'à tester, à l'aide d'une expression régulière, la présence ou l'absence du texte `is prime` dans le résultat obtenu.

3. On recommencera à l'étape 2 jusqu'à obtenir notre nombre premier.

Génération des exposants et module RSA

a. Après avoir tiré au hasard la valeur de deux nombres premiers p et q, on pourra calculer :
   - $n = pq$,
   - $\phi(n) = (p-1)(q-1)$.

b. Connaissant $e = 65537$, il faut déterminer son « inverse modulaire », d, par rapport au module $\phi(n)$. On utilisera l'algorithme d'Euclide étendu dont une version Python est fournie ci-dessous :

```
def egcd(a, b):
        x,y, u,v = 0,1, 1,0
        while a != 0:
                q, r = b//a, b%a
                m, n = x-u*q, y-v*q
                b,a, x,y, u,v = a,r, u,v, m,n
        gcd = b
        return gcd, x, y

def modinv(a, m):
        gcd, x, y = egcd(a, m)
        if gcd != 1:
                return None
        return x % m
```

L'algorithme est tiré de la page :
`http://en.wikibooks.org/wiki/Algorithm_Implementation/Mathematics/Extended_Euclidean_algorithm`

Chiffrement et déchiffrement

a. Pour chiffrer une valeur m, comme par exemple le code ASCII d'un caractère, il faut calculer : $c = m^e \pmod n$.

L'opération d'exponentiation de Python, `**`, prend un temps excessivement long. En effet, cet opérateur d'exponentiation n'est pas adapté à l'arithmétique modulaire.
Il est nécessaire de programmer un opérateur d'exponentiation modulaire efficace, s'appuyant sur la méthode de l'« exponentiation indienne » ou « exponentation rapide » :

```
def lpowmod(x, y, n):
        """puissance modulaire: (x**y)%n avec x, y et n entiers"""
        result = 1
        while y>0:
                if y&1>0:
                        result = (result*x)%n
                y >>= 1
                x = (x*x)%n
        return result
```

Cette version est tirée de la page `http://python.jpvweb.com/mesrecettespython/doku.php?id=exponentiation`

b. pour déchiffrer, on calcule $m = c^d \pmod n$.

Protocole de communication

Vous devrez écrire deux programmes réseaux : un client et un serveur permettant d'établir une connexion TCP entre Alice et Bob, ce qui permettra d'établir la connexion suivant le choix de l'un ou l'autre :
- ▷ le client initie la connexion ;
- ▷ le serveur attend la connexion du client ;

Chaque programme, client ou serveur, réalise le travail suivant :
a. création de la biclé RSA, e, d et n ;
b. une fois la connexion TCP établie, le client et le serveur échange leur valeur n vu que e est connu ;
c. il peut échanger des données chiffrées avec son interlocuteur :
   - une séquence de caractères lue au clavier, puis chiffrée par e et n de son interlocuteur ;
   - une séquence de valeurs lue depuis la connexion TCP et déchiffrée par ses paramètres d et n propres.

   Les valeurs échangées seront organisées sous forme de lignes de texte envoyées et reçues au travers de la connexion TCP, suivant un format que vous établirez vous-même.

Travail demandé

Écrire les programmes Python réalisant le **protocole de communication sécurisé par RSA**.

Votre protocole doit être capable de prendre l'ensemble des symboles que l'utilisateur peut entrer en **codage UTF-8** : lettres accentuées, jeu de symboles non latin, smileys...

Proposez des améliorations :
- gérer un nombre de chiffres variable pour la génération de p et q ;
- chiffrer plus de caractères à la fois pour permettre de réduire la quantité de données échangées ;
- permettre un « chat » chiffré entre les deux interlocuteurs ;
- *etc.*

Remise du travail

La **date de remise** du projet est fixée au **dimanche 7 janvier à minuit**.

Le travail est à réaliser en **binôme**, voire en *monôme*.

Il devra être remis sous forme d'une archive à **bonnefoi+ressys@protonmail.com**.

- ▷ Vous mettrez les **programmes sources Python** du client et du serveur.
- ▷ Pour les commentaires sur votre programme, vous joindrez un **rapport au format PDF** à cette archive qui détaillera :
  - ◇ les choix d'implémentations ;
  - ◇ les fonctions ;
  - ◇ les améliorations introduites ;
  - ◇ les problèmes rencontrés ;
  - ◇ un jeu d'essai ;
  - ◇ *etc.*