Exercice 3 (8 points)

Cet exercice porte sur les protocoles de routage, la sécurisation des communications et la programmation orientée objet.

Bob, un particulier, souhaite communiquer de façon sécurisée avec un serveur web qui héberge un site dont le nom de domaine est *alice.fr*.

Dans tout l'exercice, pour plus de simplicité d'écriture, on appelle *Bob* le navigateur et *Alice* le serveur web.

Partie A

On donne ci-après le réseau entre Alice et Bob. Le coût d'une liaison est marqué à côté de chaque arête.

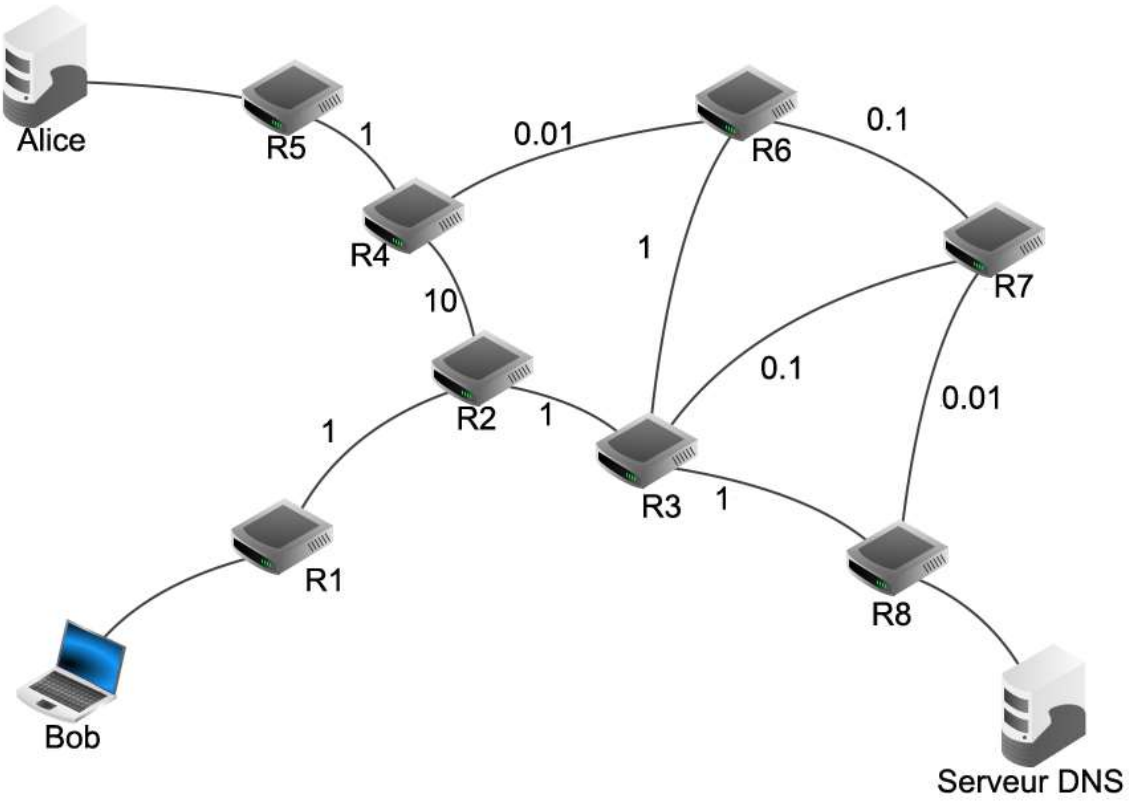


Figure 1. Schéma du réseau

La première fois que l'URL *http://alice.fr* est saisie dans la barre d'adresse de Bob, ce dernier envoie tout d'abord une première requête au serveur DNS afin d'obtenir l'adresse IP d'Alice avant de pouvoir envoyer une requête à Alice.

Le protocole RIP est un protocole de routage qui minimise le nombre de routeurs par lesquels les paquets transitent.

1. Donner, en utilisant le protocole RIP, le chemin que les paquets emprunteront afin qu'une requête de Bob parvienne au serveur DNS. On ne fera apparaître que les routeurs traversés.

Le protocole OSPF est un protocole de routage qui minimise le coût du transit des paquets.

2. On suppose que Bob a l'adresse IP d'Alice. Donner, en utilisant le protocole OSPF, le chemin que les paquets emprunteront afin qu'une requête de Bob parvienne à Alice. On ne fera apparaître que les routeurs traversés.

3. Donner le coût du chemin identifié dans la question 2.

4. Expliquer ce qui se passerait si le routeur R3 tombe en panne. On n'attend pas l'écriture des tables de routage.

5. Indiquer un protocole permettant à un navigateur et un serveur web de communiquer en chiffrant leurs données.

Bob et Alice souhaitent communiquer de façon sécurisée.

Bob génère une clé symétrique qu'il doit transmettre à Alice. Bob et Alice possèdent alors cette même clé symétrique.

6. Expliquer le rôle qu'aura cette clé symétrique dans leur communication.

Bob doit transmettre la clé symétrique qu'il a générée à Alice sans qu'un tiers puisse l'intercepter et l'utiliser pour les espionner.
Pour cela, Alice génère un couple de clés composé d'une clé publique et d'une clé privée.

7. Expliquer comment Alice et Bob vont utiliser cette clé publique et cette clé privée afin que Bob puisse transmettre de façon sécurisée sa clé symétrique à Alice. On ne parlera, dans cette question, ni de signature ni de certificat.

Lorsqu'Alice reçoit la clé symétrique chiffrée censée provenir de Bob, elle veut avoir la certitude que c'est bien Bob qui la lui a envoyée.

Pour cela, Bob utilise un système de signature digitale basé sur un couple (clé privée, clé publique) et dans lequel seule la clé privée permet de chiffrer et la clé publique permet de déchiffrer. On parle alors de signer plutôt que de chiffrer, et de signature plutôt que de chiffrement. Bob procède alors de la façon suivante :

- il envoie sa clé publique à Alice ;
- il crée un résumé, appelé condensé, des données de la clé symétrique qu'il veut transmettre à Alice. Il est alors possible de comparer le résumé et la clé pour voir s'ils correspondent bien l'un à l'autre ;
- il signe ce condensé en utilisant sa propre clé privée ;

- il transmet à Alice l'ensemble composé de la clé symétrique et de la signature, après l'avoir comme précédemment chiffré grâce à la clé publique transmise par Alice.

8. Expliquer pourquoi Alice est la seule à pouvoir accéder à la clé symétrique et peut avoir la certitude que c'est bien Bob qui la lui a envoyée.

Partie B

Dans cette partie, on se propose de coder, dans un module que l'on nommera `outils`, quelques fonctions qui serviront pour la suite de l'exercice.

9. Écrire le code d'une fonction `somme` qui prend en paramètre un tableau (type `list` en Python) d'entiers et renvoie la somme des éléments de ce tableau.

   Exemple :

```
>>> somme([1, 3, 5, 7, 9])
25
```

10. Écrire le code d'une fonction `permuter` qui :

    – prend en paramètres un tableau `tab` (type `list` en Python) et deux entiers `i` et `j` ;
    – modifie le tableau `tab` en permutant ses éléments d'indices `i` et `j`.

    Exemple :

```
>>> liste = [1, 3, 5, 7, 9]
>>> permuter(liste, 1, 3)
>>> liste
[1, 7, 5, 3, 9]
```

11. Recopier et compléter la ligne 3 du code ci-après de la fonction `inverser` qui modifie le tableau passé en paramètre en inversant l'ordre de ses éléments.

```
1 def inverser(tab):
2   for i in range(len(tab) // 2):
3     permuter(...)
```

On appelle `sac` un tableau (type `list` en Python) de 8 éléments entiers construit de la manière suivante.

- Le premier élément du `sac` est un entier choisi au hasard entre 1 et 5 tous deux inclus.
- Pour ajouter un nouvel entier au `sac` :
  – on calcule la somme `S` de tous les éléments du `sac` ;
  – on choisit au hasard un entier entre `S+1` et `2*S` tous deux inclus ;
  – on ajoute cet entier dans le `sac` à la suite des précédents.

- Une fois que le `sac` contient ses huit entiers, on inverse l'ordre de ses éléments afin que le premier élément du sac soit le plus grand.

On rappelle que la fonction `randint(a, b)` du module `random` renvoie un nombre entier choisi aléatoirement entre `a` et `b` tous deux inclus.

12. Recopier et compléter les lignes 7 à 13 du code ci-après de la fonction `generer_sac`.

```
1  from random import randint
2  def generer_sac():
3    '''
4    renvoie un sac (8 entiers)
5    : return (list) un sac
6    '''
7    sac = ...
8    for i in range(7):
9      s = ...
10     nouvel_entier = ...
11     sac.append(nouvel_entier)
12   ...
13   return sac
```

Partie C

Dans cette partie, on code une classe `Cle_symetrique` qui permet d'instancier une clé (`sac`) qui permet de chiffrer un octet en calculant un nombre entier et qui permet de déchiffrer un nombre entier en retrouvant l'octet d'origine.

Un octet est représenté par un tableau (type `list` en Python) dont les éléments sont les huit bits de l'octet.

Par exemple, le tableau `[1, 0, 1, 1, 1, 1, 0, 1]` représente l'octet `10111101`.

Cette classe est initialisée avec un unique attribut `sac` qui est un tableau renvoyé par la fonction `generer_sac` du module `outils`.

Afin de chiffrer un octet, on multiplie chaque bit d'indice `i` dans le tableau qui le représente par l'entier d'indice `i` du `sac` de l'instance de `Cle_symetrique` puis on renvoie la somme de tous les produits obtenus. Le chiffrement d'un octet est donc un entier.

Par exemple, si la clé est `[568, 205, 71, 32, 10, 6, 2, 1]`, on peut chiffrer `[1, 0, 1, 1, 1, 1, 0, 1]` par $568 \times 1 + 205 \times 0 + 71 \times 1 + 32 \times 1 + 10 \times 1 + 6 \times 1 + 2 \times 0 + 1 \times 1 = 688$.

13. Recopier et compléter la ligne 4 du code de la classe `Cle_symetrique` ci-après.

```
1 import outils
2 class Cle_symetrique():
3   def __init__(self):
4     ... = ...
```

14. Recopier et compléter les lignes 8 à 11 du code de la méthode `chiffrer` ci-après.

```
 1 def chiffrer(self, octet):
 2   '''
 3   renvoie le chiffrement de octet
 4   : paramètre octet (list) un tableau de huit 0 ou 1
 5   : return (int)
 6   '''
 7   tab_produits = []
 8   for i in range(...):
 9     produit = ... * ...
10     tab_produits.append(produit)
11   s = ...
12   return s
```

Afin de déchiffrer un entier et de retrouver l'octet auquel il correspond, on utilise la méthode `dechiffrer` ci-après.

Pour chaque indice `i` allant de 0 à 7 tous deux inclus :

- si l'entier à déchiffrer est supérieur ou égal à l'entier d'indice `i` du `sac` de l'instance de `Cle_symetrique` alors :
  - le bit d'indice `i` vaut 1 ;
  - on soustrait à l'entier à déchiffrer l'entier d'indice `i` du sac.
- sinon le bit d'indice `i` vaut 0.

15. Justifier qu'avec la méthode `dechiffrer`, le déchiffrement du nombre 688 est `[1, 0, 1, 1, 1, 1, 0, 1]`.

16. Écrire le code de la méthode `dechiffrer` de la classe `Cle_symetrique` qui prend en paramètre un `entier` à déchiffrer et renvoie un tableau représentant l'octet correspondant à cet `entier`.

    Exemple :

```
>>> ma_cle = Cle_symetrique()
>>> ma_cle.sac
[568, 205, 71, 32, 10, 6, 2, 1]
>>> ma_cle.dechiffrer(688)
[1, 0, 1, 1, 1, 1, 0, 1]
```