

SESSION 2022

---

**CAPES ET CAFEP/CAPES**

**CONCOURS EXTERNE  
TROISIEME CONCOURS**

Section

**NUMERIQUE ET SCIENCES INFORMATIQUES**

**ÉPREUVE ÉCRITE DISCIPLINAIRE**

Durée : 5 heures

---

*L'usage de tout ouvrage de référence, de tout dictionnaire et de tout matériel électronique est rigoureusement interdit.*

- *Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence.*
- *De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.*

**NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.**

**Tournez la page S.V.P.**

# Épreuve disciplinaire

## Étude de protocoles de traçage de contacts numériques

Dans le domaine de la santé, le *traçage de contacts* (dénommé TC par la suite) est un processus permettant d'identifier les individus qui ont pu être en contact avec une personne infectée. L'objectif du TC au cours d'une épidémie est de réussir à identifier rapidement les personnes ayant pu être contaminées afin de les tester, les traiter si besoin et/ou les isoler.

Nous nous intéressons à deux protocoles de TC automatisés (les protocoles HAMAGEN et ROBERT) qui ont été mis en œuvre durant la pandémie de la Covid19 dans différents pays. Le but est d'en proposer une implémentation et d'en discuter les propriétés en terme de sécurité.

Par la suite, il sera supposé que les individus disposent tous d'un téléphone portable, et qu'il y a une infrastructure centralisatrice (par exemple un cloud ou un serveur), appelée IC par la suite, qui est sous le contrôle d'une entité bien identifiée (par exemple le ministère de la santé).

Toutes les fonctions demandées par la suite devront être rédigées en langage Python.

Le problème est constitué de 6 exercices qui peuvent être traités de manière indépendante, en admettant le résultat des questions précédentes si nécessaire. Certains concepts sont utilisés tout au long du problème, ils sont présentés dans la première partie du sujet.

Les réponses aux questions devront être précises et rédigées avec soin.

## Concepts et notations nécessaires

Les protocoles de traçage de contacts reposent sur des concepts décrits par la suite. Nous donnons ici la manière dont les structures de données, associées à ces concepts, sont représentées en Python.

**Point de localisation :** la localisation des individus est réduite à un point, représenté en Python par un couple de réels (`lat`, `long`) désignant la latitude et la longitude de l'individu sur le globe terrestre.

**Temps :** le protocole nécessite de mémoriser la position passée des individus sur leur téléphone avec une périodicité  $P$  exprimée en minutes. Pour plus de simplicité, on se donne une date origine  $D_0$  commune à tous les individus et on compte les dates de mémorisation  $D_e$  à partir de cette date d'origine, de la manière suivante :

$$D_e = D_0 + e \times P$$

où l'entier naturel  $e$  (dénommé *époque* par la suite) désigne le nombre d'intervalles écoulés depuis  $D_0$ .

Dans la suite, nous supposons que la périodicité est de 15 minutes ( $P = 15$ ).

**Trace :** le téléphone de chaque utilisateur contient un dictionnaire `trace` dont la clé est une époque (un entier naturel) et la valeur est un point de localisation (un couple de réels (`lat`, `long`)) désignant la position de l'individu à l'époque considérée. Pour des raisons de simplicité, on ne considèrera ici que des latitudes Nord et longitudes Est. Ainsi le dictionnaire  $\{4: (10.6, 12.3), 5: (7.2, 1.0)\}$  signifie qu'à l'époque 4, l'individu était localisé à la latitude  $10^\circ 6'N$  et la longitude  $12^\circ 3'E$  tandis qu'à l'époque 5, il était situé à la latitude  $7^\circ 2'N$  et la longitude  $1^\circ 0'E$ .

**Durée de contagiosité :** on fait l'hypothèse qu'il existe une durée maximale pendant laquelle les personnes sont contagieuses. Cette durée est notée *Cont* et est exprimée en nombre de jours.

**Traces à risques :** l'infrastructure centralisatrice maintient de son côté une liste `listeAR` mémorisant les traces considérées à risque. Une trace à risque correspond à une trace d'une personne qui a été testée positive. La structure de données `listeAR` est donc d'une liste de dictionnaires correspondant aux traces de mobilité des personnes ayant été testées positivement à la Covid19. La durée des traces correspond à la durée où les personnes étaient potentiellement contagieuses. Cette liste peut ensuite être téléchargée librement par n'importe quel individu. Les personnes sont identifiées par leur position (arbitraire) dans la liste.

Ainsi la liste  $[\{4: (10.6, 12.3), 5: (7.2, 1.0)\}, \{1440: (243.0, 12.1)\}]$  indique les traces de deux individus avec leurs positions aux époques mémorisées par le système de traçage et pour lesquelles ils sont considérés à risque. Les données du premier individu concernent les époques 4 et 5, les données du 2e individu ne concernent que l'époque 1440.

## Exercice 1 : Protocole *HaMagen*

Le protocole de traçage de contacts *HaMagen* repose uniquement sur les notions présentées juste avant dans la partie introductive. Dans cette partie nous nous intéressons à son implémentation en Python dans une version simplifiée.

### Question 1.1

Avec les structures de donnée Python choisies, est-il possible de stocker la position de l'individu aux époques 4 et 6, sans connaître sa position à l'époque 5 ? Justifier brièvement.

### Principe du protocole *HaMagen*.

Le fonctionnement du protocole *HaMagen* repose sur le principe suivant : la trace d'un individu est stockée sur son téléphone portable. Si jamais l'utilisateur est testé positif à la Covid19 à une époque  $e$ , alors les 14 derniers jours de sa trace sont téléversés sur l'infrastructure centralisatrice ( $Cont = 14$ ). La liste des traces à risque, `listeAR`, est mise à jour en ajoutant cette nouvelle trace à la fin de `listeAR`.

Indépendamment, toute personne peut télécharger régulièrement la liste des traces à risque, `listeAR`, depuis l'infrastructure centralisatrice sur son téléphone. Ceci lui permet de comparer les traces à risque téléchargées avec sa propre trace stockée sur son téléphone. Si jamais certains points de localisation de sa trace sont proches d'une ou plusieurs traces à risque, alors la personne est identifiée comme cas contact.

### Question 1.2

Écrire une fonction python `debutAR` qui, étant donnée une époque  $e$  à laquelle un individu est testé positif, renvoie l'époque correspondant au début de la période à risque, c'est-à-dire l'entier naturel désignant l'époque 14 jours avant  $e$ .

**Remarque :** On rappelle que la période d'enregistrement des localisations est supposée de 15 minutes et que la valeur de l'époque retournée doit être supérieure ou égale à 0.

### Question 1.3

En déduire une fonction python `extraitTraceAR` qui, étant données la trace `tr` d'un individu et une époque  $e$  à laquelle il est détecté positif, renvoie un dictionnaire construit à partir de `tr` contenant uniquement les époques et localisations considérées comme à risque.

### Question 1.4

Montrer que la taille du dictionnaire construit avec la fonction `extraitTraceAR` est bornée par une constante qui ne dépend que de  $P$  et  $Cont$ . Dans quel cas cette borne est-elle atteinte ?

### Implémentation du protocole *HaMagen*.

Chaque personne peut donc à tout moment vérifier si elle est devenue une personne à risque via l'algorithme 1 décrit ci-dessous.

Dans cet algorithme, on suppose implémentées les fonctions `epoque()`, `posGPS()` et `recupTraces()` qui permettent respectivement d'obtenir l'époque courante, la position courante du téléphone portable ainsi que la liste, récupérée depuis l'infrastructure centrale, contenant les traces considérées à risque.

La dernière fonction, `aCroise(t1, t2, dContact)`, reste à implémenter. Elle permet de savoir si l'individu ayant produit la trace `t1` a croisé l'individu ayant produit la trace `t2` à une distance inférieure ou égale à `dContact` à au moins une époque par le passé.

---

**Algorithme 1** : Protocole HaMagen simplifié

---

**Entrée** : `dContact` : réel exprimant la distance, en mètre, en dessous de laquelle deux personnes sont considérées être en contact

**Structure de données** : `maTrace` : dictionnaire (au départ vide) contenant la trace de l'utilisateur

```
1 maTrace = dict() ;
2 aRisque = False ;
3 Tant que not aRisque faire
4   maTrace[epoque()] = posGPS() ;
5   listeAR = recupTraces() ;
6   Pour tr in listeAR faire
7     Si aCroise(tr, maTrace, dContact) alors
8       aRisque = True;
9   Fin
10 Fin
11 attendre 15 minutes ;
12 Fin
13 print('Vous venez d'être identifié comme cas contact.');
```

---

### Question 1.5

Soient deux points  $p_1 = (\text{lat}_1, \text{long}_1)$  et  $p_2 = (\text{lat}_2, \text{long}_2)$  sur la Terre, tous deux localisés par rapport au Nord et à l'Est, avec des angles compris entre  $0^\circ$  et  $90^\circ$ . La distance entre ces deux points peut se calculer en première approximation à l'aide du Théorème de Pythagore (pour des points assez proches) par :

$$d = k \times \sqrt{x^2 + y^2}$$

avec  $x = \text{lat}_1 - \text{lat}_2$ ,  $y = (\text{long}_1 - \text{long}_2) \times \cos\left(\frac{\text{lat}_1 + \text{lat}_2}{2}\right)$  et  $k = 111\,120$  un facteur multiplicatif permettant d'exprimer le résultat en mètres.

Écrire une fonction Python `proximite` qui, étant donnés deux points `p1` et `p2` ainsi qu'une distance `dContact` exprimée en mètres, renvoie `True` si les points sont à une distance inférieure ou égale à `dContact` et `False` sinon.

### Question 1.6

En déduire l'écriture d'une fonction Python `aCroise(t1, t2, dContact)` qui, étant données deux traces `t1` et `t2` ainsi qu'une distance `dContact` exprimée en mètres, renvoie `True` s'il existe au moins une époque durant laquelle un point  $p_1$  de `t1` et un point  $p_2$  de `t2` sont à une distance inférieure à la distance `dContact`. Elle renvoie `False` sinon.

### Question 1.7

Quels sont les paramètres nécessaires pour exprimer la complexité de la fonction `aCroise` ? Quelle est la complexité de la fonction dans le pire des cas ?

## Exercice 2 : Requêtes SQL : stockage des données sur l'infrastructure centralisatrice

La fonction `recupTraces()` utilisée dans l'algorithme 1 effectue un appel à l'infrastructure centralisatrice (IC) pour récupérer les traces de tous les utilisateurs positifs à la Covid19 qui ont accepté de téléverser leurs données sur le serveur. Nous faisons l'hypothèse que ces informations sont stockées sur l'IC dans une base de données relationnelle, dont le schéma est le suivant (clés primaires soulignées, clés étrangères indiquées par une contrainte REFERENCES) :

```
- users (id, numTel)
- traces (id, epoque, latitude, longitude)
- traces.id REFERENCES users.id
```

La base de données est donc composée de deux tables : `users` qui fait le lien entre un numéro de téléphone et un identifiant arbitraire généré par l'IC (`id`), et `traces` qui contient les informations de position GPS pour un `id` spécifique et une `epoque` donnée. Le numéro de téléphone ne sera pas transmis à d'autres utilisateurs de l'application et reste simplement sur l'IC. Il sert à vérifier, au moment du téléversement, que toutes les traces d'un même utilisateur sont bien affectées au bon `id`.

L'algorithme de téléversement est donc le suivant : un utilisateur détecté positif à la Covid19 télécharge ses positions GPS une par une, en associant son numéro de téléphone. Lorsque le serveur reçoit une position, il vérifie si le numéro de téléphone existe déjà. S'il n'existe pas, il génère un nouvel identifiant `id`, insère le couple (`id`, `numTel`) dans la table `users`, puis insère la position dans la table `traces` en utilisant ce nouvel identifiant. Si le numéro de téléphone existe déjà, l'IC retrouve l'id correspondant, n'insère rien dans la table `users`, et insère la position dans la table `traces` avec le bon identifiant.

Dans les questions qui suivent, on ne s'intéresse pas à la manière dont les traces sont échangées entre le téléphone et l'IC, mais simplement à la manipulation de ces traces sur le serveur.

### Question 2.1

Donner la requête SQL qui retourne l'id correspondant au numéro de téléphone 0123456789, si celui-ci est présent dans la base.

### Question 2.2

Donner la requête SQL qui retourne combien d'utilisateurs (identifiés de manière unique par leur numéro de téléphone) sont présents dans la base.

### Question 2.3

Donner la requête SQL qui retourne l'ensemble des positions (latitude et longitude) de l'utilisateur correspondant au numéro de téléphone 0123456789.

### Question 2.4

Donner la requête SQL qui retourne le nombre de positions enregistrées sur le serveur pour chaque utilisateur, identifié de manière unique par son numéro de téléphone.

### Question 2.5

Donner la requête SQL qui retourne combien d'utilisateurs ont laissé plus de 1000 enregistrements de position sur le serveur.

### Question 2.6

On suppose la base de données vide, et qu'un `commit` est exécuté après chaque insertion. L'IC exécute les instructions suivantes :

1. `INSERT INTO traces VALUES(1, 0, 1.52, 1.55)`
2. `INSERT INTO traces VALUES(1, 1, 1.53, 1.55)`
3. `INSERT INTO users VALUES(1, 0123456789)`

Indiquer le contenu de la base de données une fois ces trois instructions exécutées. Expliquer.

### Question 2.7

On suppose la base de données vide, et qu'un `commit` est exécuté après chaque insertion. L'IC exécute les instructions suivantes :

1. `INSERT INTO users VALUES(1, 0123456789)`
2. `INSERT INTO users VALUES(2, 0987654321)`
3. `INSERT INTO traces VALUES(1, 0, 1.52, 1.55)`
4. `INSERT INTO traces VALUES(1, 1, 1.53, 1.55)`
5. `INSERT INTO traces VALUES(2, 0, 1.52, 1.55)`
6. `INSERT INTO traces VALUES(2, 0, 1.53, 1.55)`

Indiquer le contenu de la base de données une fois ces six instructions exécutées. Expliquer.

## Exercice 3 : Sécurité et vie privée : confidentialité des données sur le serveur

La requête proposée à la question 2.3 montre que l'administrateur de la base de données est capable de retrouver toutes les traces de tous les individus dont il connaît le numéro de téléphone. On cherche donc à modifier l'infrastructure pour résoudre ce problème en chiffrant l'information sensible qu'est le numéro de téléphone.

### Question 3.1

Est-ce que chiffrer les communications entre les individus et l'IC, comme le fait le protocole HTTPS pour le Web, permettrait de résoudre le problème mentionné ci-dessus ? Justifier.

### Chiffrement

On décide désormais de ne pas stocker le numéro de téléphone `numTel`, mais plutôt sa valeur chiffrée  $E(\text{numTel})$ , en utilisant un algorithme de chiffrement symétrique, paramétré par une clé de chiffrement  $KS$ , que l'on supposera inaccessible à quiconque, y compris les administrateurs de l'infrastructure centralisée. Ainsi un ajout dans la base de données se fera en rajoutant un appel à une fonction  $E$  de chiffrement avec la commande `INSERT INTO users VALUES(1, E('0123456789'))`.

Pour vérifier si un numéro de téléphone est déjà présent dans la base, l'IC cherche si la valeur chiffrée de ce numéro de téléphone est présent dans la table `users`.

On suppose par ailleurs que la fonction de chiffrement  $E$  est une fonction qui peut être appelée depuis n'importe quelle requête SQL, sans avoir besoin de connaître la clé  $KS$ .

### Question 3.2

On considère que  $E$  est à valeurs dans un ensemble fini. Est-il possible d'avoir la propriété suivante, correspondant à une *collision* : il existe  $X, Y$  deux numéros de téléphone tels que  $E(X) = E(Y)$  et  $X \neq Y$  si  $E$  est à valeurs dans un ensemble fini de taille  $2^{16}$  ? Que se passerait-il si  $E$  était à valeurs dans un ensemble fini de taille  $2^{64}$  ? On ne demande pas de calculer la probabilité de collision.

### Question 3.3

Est-il possible pour l'administrateur de la base de données de retrouver les traces d'un individu dont il connaît le numéro de téléphone, maintenant que les numéros de téléphone sont chiffrés dans la base ? Justifier.

## Sécurité du protocole *HaMagen*.

Tel qu'il est défini dans l'algorithme 1, le protocole présente la vulnérabilité suivante : les points de localisation correspondant à un individu peuvent tous être liés à la même trace (celle de l'individu). Ainsi, si on est capable de retrouver la position d'un individu à un moment donné, on sera capable de connaître l'ensemble de sa trace (sur les  $Cont = 14$  derniers jours).

## Principe de l'attaque du protocole *HaMagen*.

Mme X (une espionne par exemple) achète un téléphone et installe l'application. Elle allume ensuite le GPS et l'application pendant  $k$  époques  $T_{on}$  à  $T_{on+k-1}$ , durant lesquelles elle se retrouve seule à proximité d'un individu cible (Mr Y) qu'elle souhaite pister. Elle éteint ensuite son GPS et son téléphone, pour ne le rallumer que lorsqu'elle est sûre d'être seule, afin de récupérer l'ensemble des traces à risque depuis le serveur. Si à un moment donné, dans les traces récupérées sur son téléphone, elle retrouve une trace qui a croisé la sienne, alors elle peut être sûre de deux choses : 1) Mr Y a été testé positif à la Covid19 et a envoyé sa trace à l'IC, et 2) tous les points de la trace qu'elle a croisés sont les points de Mr Y. Mme X sera donc en mesure de retracer les déplacements des 14 derniers jours de Mr Y.

### Question 3.4

En vous inspirant de l'algorithme 1, proposer un algorithme renvoyant la trace de l'individu ciblé en réalisant l'attaque décrite ci-dessus.

## Exercice 4 : Détection de "hot spots"

L'autorité gérant l'IC possède de nombreuses données sur les personnes ayant été testées positives à la Covid19 et s'étant déclarées sur l'application *HaMagen*.

L'IC souhaite détecter des *hot spots*, c'est-à-dire des endroits où de nombreuses personnes testées positives à la Covid19 se sont trouvées. On fait l'hypothèse que si une trace est présente dans la base de données de l'IC, alors la personne était positive à la Covid19, et donc que l'IC ne contient que des traces de personnes positives à la Covid19 (il n'y a donc pas de faux positifs).

**Remarque :** Dans toute ce qui suit, vous pouvez, si besoin, vous servir des fonctions proposées dans l'exercice 1.

### Question 4.1

Ecrire une fonction Python `estHotspotEpoque(p, listeAR, k, dContact, e)` qui, étant donné un point de localisation `p`, une liste de dictionnaires `listeAR` contenant les traces à risques, un entier `k`, un réel `dContact` et un entier `e`, renvoie `True` si au moins `k` individus présents dans la liste `listeAR` sont présents à l'époque `e` à une distance inférieure ou égale à `dContact` du point `p`.

### Question 4.2

Donner la complexité de votre fonction dans le pire des cas. Justifier.

### Question 4.3

En déduire une fonction Python `estHotspot(p, listeAR, k, dContact)` qui renvoie `True` si le point `p` de localisation est un hotspot, c'est-à-dire si au moins `k` individus de la liste `listeAR` sont présents à une même époque, à une distance inférieure ou égale à `dContact` de `p`.

### Question 4.4

Donner la complexité de cette fonction dans le pire des cas. Justifier.

On s'interroge maintenant sur les points de localisation à tester avec la fonction `estHotspot`. On pourrait par exemple tester toutes les positions de France avec une certaine granularité mais ce serait peu efficace.

### Question 4.5

Dans un premier temps, on se propose d'utiliser la fonction `mystere(listeAR, k, dContact)` suivante, dont le programme est donné en Python :

```
def mystere(listeAR, k, dContact):  
    IHP = []  
    for tr in listeAR:  
        for e in tr:  
            p = tr[e] # prochain point a tester  
            if estHotspot(p, listeAR, k, dContact):  
                IHP.append(p)  
    return IHP
```

Expliquer ce que fait cette fonction et ce qu'elle renvoie.

### Question 4.6

Quelle est la complexité de la fonction `mystere(listeAR, k, dContact)` dans le pire des cas ?

On se propose d'éviter, par la suite, de considérer comme hotspots des points qui sont trop proches les uns des autres.

## Question 4.7

Proposer une amélioration de la fonction précédente en écrivant une fonction `listHotspotsDistincts(listeAR, k, dContact)` qui renvoie une liste contenant les points de localisation de `listeAR` qui sont des hotspots mais pour lesquels aucun autre point de la liste renvoyée n'est à une distance inférieure ou égale à `dContact`.

## Question 4.8

Compléter les blocs `***(1)***` et `***(2)***` de la fonction Python `recupTraces()` donnée ci-dessous. Cette fonction retourne une liste `listeAR` à partir des données stockées dans la base de données de l'IC. Le bloc `***(1)***` représente une requête SQL, et le bloc `***(2)***` peut contenir plusieurs instructions Python.

**Note :** étant donnée une requête SQL de la forme `SELECT A0, A1, ... AN FROM T WHERE ...`, l'instruction `ligne[i]` permet d'accéder à la valeur de l'attribut  $A_i$  de la clause `SELECT` si on l'utilise dans une boucle `for ligne in liste:`.

```
import sqlite3
def recupTraces():

    conn = sqlite3.connect('baseDonnees.db')
    cur = conn.cursor()
    cur.execute(***(1)**)
    liste = cur.fetchall()
    ***(2)***
    conn.close()
    return listeAR
```

## Exercice 5 : Algorithmique de graphes

On souhaite maintenant étudier la structure relationnelle établie entre les personnes contaminées afin d'identifier celles qui sont les plus centrales. Pour cela, on construit un graphe social non orienté et non pondéré `GraphCovid` dont les sommets sont des personnes (identifiées par un entier) et dont les arêtes relient deux personnes qui ont été en contact à une même époque.

On supposera par la suite que le graphe est décrit par une liste d'adjacence, implémentée en Python par un dictionnaire dont les clés sont les identifiants des personnes et les valeurs sont les listes de sommets voisins dans le graphe. On supposera par ailleurs le graphe non vide et connexe.

On considérera par la suite le graphe `GraphCovid` décrit par la structure suivante :

```
GraphCovid = {
    1: [2, 3, 4, 5],
    2: [1],
    3: [1],
    4: [1],
    5: [1, 6],
    6: [5, 7, 8],
    7: [6, 9, 10],
    8: [6, 10, 11],
    9: [7],
    10: [7, 8],
    11: [8]
}
```

## Question 5.1

Donner une représentation graphique de la structure `GraphCovid`.

Pour identifier les personnes les plus centrales dans un graphe social  $G = (V, E)$ , plusieurs métriques peuvent être utilisées comme :

**Degré :** le degré d'un sommet  $s$  correspond à son nombre de voisins dans le graphe.



**Centralité de proximité :** la centralité de proximité d'un sommet  $s$  est définie par l'inverse de la somme des distances de ce sommet vers tous les autres sommets du graphe :

$$C_{prox}(s) = \frac{1}{\sum_{x \in V \setminus \{s\}} dist(s, x)}$$

où  $dist(s, x)$  correspond à la longueur d'un plus court chemin entre  $s$  et  $x$  dans  $G$ .

### Question 5.2

Sur le graphe **GraphCovid** précédent, quel sommet a le plus fort degré et quelle est sa valeur ? Quel sommet a la plus forte centralité de proximité et quelle est sa valeur parmi les sommets 1, 5 et 6 ?

### Question 5.3

Écrire une fonction Python `plusFortDegré(G)` qui, étant donné un graphe  $G$ , renvoie l'identifiant du sommet de plus fort degré.

On souhaite maintenant déterminer un sommet de plus forte centralité de proximité. Pour cela nous allons nous appuyer sur l'algorithme de parcours en largeur d'un graphe, qui est donné ci-dessous :

---

#### Algorithme 2 : Parcours en largeur d'un graphe

---

**Entrée :**  $G$ , un graphe

**Entrée :**  $s$ , identifiant du sommet de départ

```
1 F = FileVide();
2 Enfiler(F,s);
3 Marquer(s);
4 Tant que  $F$  non vide faire
5   | x=Defiler(F);
6   | Pour  $y$  voisin de  $x$  dans  $G$  faire
7   |   | Si  $y$  non marqué alors
8   |   |   | Enfiler(F,y);
9   |   |   | Marquer(y);
10  |   | Fin
11  | Fin
12 Fin
```

---

### Question 5.4

Pourquoi l'algorithme de parcours en largeur est utile pour calculer la centralité de proximité d'un sommet ?

### Question 5.5

En adaptant l'algorithme 2, écrire un algorithme `Dist` qui, étant donné un graphe  $G$  et un sommet de départ  $s$ , permet de calculer et de renvoyer les distances entre  $s$  et tous les autres sommets de  $G$ .

### Question 5.6

En déduire un algorithme `SommetProxMax` qui, étant donné un graphe  $G$ , renvoie un sommet de centralité de proximité maximale dans  $G$ .

### Question 5.7

Que peut-on dire de l'intérêt des deux métriques proposées dans le contexte de la propagation de proche en proche d'une épidémie ? L'une des métriques semble-t-elle plus pertinente que l'autre pour identifier les individus cruciaux dans la dynamique de propagation ? Pourquoi ?

## Exercice 6 : Protocole ROBERT

Le principal problème qu'on retrouve dans l'algorithme HaMagen est que les données collectées sur les personnes atteintes de la Covid19 peuvent faire fuiter des données privées, *i.e.* leur position à une époque donnée. Afin de ne pas collecter des données aussi précises que la position GPS, d'autres algorithmes respectant le RGPD ont été proposés, notamment par un consortium d'industriels (Apple et Google), mais aussi par des équipes de recherche (avec le protocole ROBERT). Ces deux protocoles (celui proposé par des industriels et le protocole ROBERT) n'utilisent pas la position GPS des individus, mais la technologie de communication *Bluetooth*, qui va permettre de détecter d'autres appareils à proximité, sans pour autant connaître la position où ils se trouvent.

### Question 6.1

Qu'est-ce que le RGPD et quel est le principe appliqué ici ?

### Protocole ROBERT simplifié

1. Un utilisateur  $u$  s'inscrit au protocole, en contactant l'IC. L'IC génère un identifiant unique  $ID_u$  pour l'utilisateur  $u$  et lui communique cet identifiant. L'IC génère également une clé symétrique  $SK_u$  et envoie  $SK_u$  à l'utilisateur  $u$ . Pour simplifier, on considèrera que  $SK_u$  permet à la fois de chiffrer et d'authentifier les messages entre l'IC et l'utilisateur  $u$ . L'application de l'utilisateur  $u$  conserve au secret la clé  $SK_u$ . L'IC connaît également une clé secrète  $K_S$ .
2. Pour chaque époque  $i$ , l'IC calcule un identifiant dit "éphémère" déduit de  $ID_u$  par  $i$  applications d'une fonction  $\Phi$  :

$$EphID_i(u) = \Phi^i(ID_u \oplus K_S)$$

$\oplus$  étant l'opérateur logique XOR.

La fonction  $\Phi$  est une fonction de *hachage cryptographique* à valeurs dans un ensemble de taille  $2^{64}$ , et utilisant une clé secrète  $K_S$  de même longueur que  $ID_u$ . Ces identifiants sont communiqués par groupe de  $d$  identifiants (par exemple une fois par jour l'IC envoie à chaque utilisateur  $d = 96$  identifiants à utiliser pendant la journée correspondant aux identifiants éphémères qui seraient calculés aux 96 époques de la journée). L'envoi de ces identifiants est chiffré par le serveur avec la clé  $SK_u$ .

3. Entre chaque époque  $i$  et chaque époque  $i + 1$ , le téléphone de l'utilisateur  $u$  émet régulièrement (par exemple chaque seconde)  $EphID_i(u)$  via la technologie de communication Bluetooth.
4. Chaque téléphone stocke l'ensemble des  $EphID_i$  distincts qu'il capte à proximité via Bluetooth pour chaque époque  $i$ .
5. Si jamais l'utilisateur  $v$  est diagnostiqué positif à la Covid19, alors le téléphone de l'utilisateur  $v$  envoie à l'IC un message incluant l'ensemble  $\epsilon_v$  contenant tous les  $EphID$  qu'il a reçus et les époques associées au cours des  $Cont = 14$  derniers jours. L'infrastructure centralisatrice est alors capable de retrouver l'ensemble des utilisateurs qui ont été exposés, en retrouvant les  $ID_u$  à partir des listes envoyées par les individus ayant été testés positifs à la Covid19.
6. Régulièrement (par exemple une fois par jour), le téléphone de chaque utilisateur se connecte et demande s'il a été identifié parmi la liste des contacts possibles. Si c'est le cas, l'IC l'en informe, et l'utilisateur est invité à se faire tester.

On rappelle qu'une fonction de hachage est une fonction dont le code est public et qui est à valeurs dans un ensemble fini de taille  $\Omega$  (par exemple  $\Omega = 2^{64}$ ). Comme il faut utiliser le résultat de l'application de  $\Phi$  comme paramètre d'entrée de  $\Phi$ , les  $ID_u$  peuvent donc être codés sur  $\log_2(\Omega)$  bits au plus. On fait également ici l'hypothèse que  $\Phi$  est bijective sur  $\Omega$ .

### Question 6.2

Comme la fonction  $\Phi$  est publique et que l'utilisateur connaît son  $ID_u$ , est-ce qu'un utilisateur  $u$  est capable de calculer  $EphID_i(u)$  étant donné  $i$  ?

### Question 6.3

On note  $n$  le nombre d'utilisateurs au système de traçage ROBERT. Donner, en fonction de  $n, \Omega$  et du nombre d'époques qui se sont écoulées (qu'on notera  $\mathcal{E}$ ), la probabilité  $P_{dist}$  de ne générer que des valeurs distinctes pour les identifiants éphémères des  $n$  utilisateurs pour les  $\mathcal{E}$  applications successives de  $\Phi$ .

### Question 6.4

Si on se donne  $n = 2$  et  $\Omega = 2^{10}$ ,  $P_{dist} \approx 0.994$  pour  $\mathcal{E} = 2$  et  $P_{dist} \approx 0.985$  pour  $\mathcal{E} = 3$ . Que pouvez-vous en déduire ? Justifier l'intérêt de ce calcul de probabilité.

### Question 6.5

On fait l'hypothèse que  $w_1$  et  $w_2$  ont croisé au même moment une personne contaminée (utilisateur  $v$ ). Est-ce qu'il est possible pour l'IC de savoir que  $w_1$  et  $w_2$  ont été en contact ? Est-il possible pour l'IC de savoir que  $w_1$ ,  $w_2$  et  $v$  ont été en contact ?

### Question 6.6

Le protocole ROBERT complet impose des contraintes supplémentaires lors du téléversement des données par un utilisateur sur l'IC :

1. la liste  $\epsilon_v$  n'est pas envoyée en une seule fois, mais chaque *EphID* de la liste  $\epsilon_v$ , avec l'époque associée, est envoyé par un message indépendant  $m_{v,j}$  (pour  $j \in [1; |\epsilon_v|]$ ) ;
2. chaque message  $m_{v,j}$  est envoyé via un *mixnet*.

Un *mixnet* est une structure d'anonymisation qui s'intercale entre un expéditeur et un récepteur et qui ne permet pas au récepteur de connaître l'adresse IP de l'expéditeur. D'autre part, un *mixnet* n'envoie pas forcément tous les messages dès qu'il les reçoit. Par exemple, un *mixnet* peut attendre avant d'envoyer un message, ou encore inverser l'ordre d'envoi des messages provenant d'utilisateurs différents ou du même utilisateur.

Expliquer comment ces deux contraintes supplémentaires participent à la sécurité du protocole ROBERT.

## INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.  
Ces codes doivent être reportés sur chacune des copies que vous remettrez.

### Concours externe du CAPES

- Enseignement public :

| Concours | Section/option | Epreuve | Matière |
|----------|----------------|---------|---------|
| E B E    | 6 2 0 0 E      | 1 0 1   | 9 3 1 1 |

- Enseignement privé :

| Concours | Section/option | Epreuve | Matière |
|----------|----------------|---------|---------|
| E B F    | 6 2 0 0 E      | 1 0 1   | 9 3 1 1 |

### Troisième concours du CAPES

- Enseignement public :

| Concours | Section/option | Epreuve | Matière |
|----------|----------------|---------|---------|
| E B V    | 6 2 0 0 E      | 1 0 1   | 9 3 1 1 |