EPITA

LSE Équipe Sécurité

RECRUTEMENT 2020-2021)

LSE RECRUTEMENT:
Épreuve de progammation
Cycles et PSeudo Cycles dans DES

Enseignants
Équipe Sécurité — LSE

Semestre S5

Décembre 2020

Table des matières

1 Recommendations

Grâce à Python et à ses nombreuses bibliothèques, il n'est pas nécessaire de comprendre en détails les algorithmes dont on parle ici. Néanmoins, si vous désirez comprendre par exemple DES, n'importe quel bon livre de cryptographie le présente, et wikipedia (en Anglais) le présente très bien. Pour le travail demandé ici, il suffit de comprendre que DES chiffre avec une clé de 56 bits un message de 64 bits.

2 Date limite et rendu ?

2.1 Date Limite

On fixera la date limite début janvier 2021.

2.2 Rendu

Vous devrez déposer votre *unique* fichier notebook python sur Teams. **Merci d'indiquer votre nom ÃăÃă la fois dans le fichier et dans le nom svp !**

2.3 Que rendre ?

— Vous devrez rendre (exclusivement) un notebook jupyter en python 3.
— Le modèle est sur Teams. Merci de **mettre dans le nom du fichier votre nom svp**.
Bon courage à toutes et à tous.
et

JOYEUX RÉVEILLON !

3 Votre travail

3.1 Code python

Vous devez choisir un projet, fixer les différents paramètres et *coder* en python.

3.2 Projet

Soit la problèmatique suivante :

1. Choisir un algorithme $\mathcal{A}$ parmi : DES (mode ECB en priorité mais on peut aussi considérer les autres modes), AES-128 (idem pour le mode), AES-256.
   — Choisir une valeur initiale $m = x0$, adaptée à l'algorithme en nombre de bits, aléatoire (ou pas : à vous de choisir).
   — Choisir une valeur de la clé K, adaptée à l'algorithme en nombre de bits, aléatoire(ou pas : à vous de choisir).
2. Considérer l'application $\mathcal{A}$ et rechercher un cycle *approché*.
3. Choisir une distance du style
   — (pour tester votre code) : distance$(Tortue, Lievre)$ = nombre de bits différents
   — puis, dans un second temps :

   $$\text{distance}(Tortue, Lievre) = \text{nombre de caractères différents}$$

   — ou (le plus intéressant)

   $$\text{Distance}(Tortue, Lievre) = \text{longueur de la plus } \textit{longue sous-chaine commune}.$$

4. Et enfin, choisir une valeur de Borne *judicieuse* en fonction de Distance.

Précisions :

1. Le test le plus intéressant à faire, que ce soit avec un algorithme de chiffrement ou un algorithme de hashage est évidemment lorsqu'on utilise Distance$(Tortue, Lievre)$ = longueur de la plus *longue sous-chaine commune* qu'il faut évidemment **normaliser**. En effet, tel quel ce n'est pas une distance, supposons par exemple qu'on travaille avec des mots de 128 bits, soit 16 caractères, il faut donc utiliser :

   $$\text{Distance}(Tortue, Lievre) = 16 - \text{longueur de la plus } \textit{longue sous-chaine commune}$$

   Ainsi, on aura donc Distance$(Tortue, Tortue) = 0$.
2. On pourrait aussi utiliser

   $$\text{distance}(Tortue, Lievre) = \text{nombre de bits différents}$$

   dans ce cas il **ne faudrait pas** normaliser, on aurait ainsi Distance$(Tortue, Tortue) = 0$ mais je me dois de préciser qu'une haute valeur de cette distance n'est pas très intéressante. Cela devient intéressant pour une chaine de 128 bits par exemple si on a Distance$(Tortue, Tortue) \leq 64$ (ce qui à mon avis est probablement difficile à atteindre mais sait-on jamais).
3. Qu'est ce qui est vraiment intéressant alors ? Le *graal* serait de trouver un schéma, un *pattern*. Je reprends l'exemple donnée par un étudiant : si on choisit $x0 =$ "*toto*" et on calcule un cycle approché pour un *algorithme de hachage* (donc : pas de clé) comme MD4, MD5, SHA-1 etc. Ok, pourquoi pas. Mais alors des questions se posent :

- Le calcul donne t-il la même chose ou pas pour "toute" chaine $x0 =$ "$xyxy$", avec x et y deux caractères différents ?
- Par exemple qu'observe t'on pour la chaine miroir "otot" ?
- Le calcul donne t-il la même chose ou pas pour "toute" chaine $x0 =$ "$xyxyxy$", avec x et y deux caractères différents ?
- etc.
- Et qu'en est-il au niveau des différences entre le code ascii de "t" et celui de "o". En clair, qu'observe t'on pour la chaine décalée à droite "upup" ? (schéma : $t- > u$ et $o- > p$) ?
- ou encore qu'observe t-on pour la chaine décalée à gauche "snsn" ? (schéma : $t- > s$ et $o- > n$) ?
- etc.

4. On peut se poser le même genre de questions (existence de schémas) sur la chaine initiale et/ou sur la clé !
5. À vous de choisir ce que vous voulez mettre en évidence (présence ou absence de schémas).
6. **Par contre**, je rappelle que ce projet est une des épreuves, donc on ne vous demande pas de tout chercher, tout tester. Restez raisonnable.

Dans un premier temps soyez disons *prudent.e.s*, si vous chercher un cycle avec une borne trop précise, cela risque de prendre du temps, voire beaucoup beaucoup de temps.

3.3 Quelques questions/réponses

1. **Q :** À partir de quelle longueur de sous-chaine commune le "cycle" trouvé est-il pertinent ? **R :** Bonne question mais cela dépend vraiment de l'algorithme choisi (chiffrement ou hachage ?) de l'entropie de votre chaine initiale ($x0 =$ "$AAAAAAAA$" ou $x_0 =$ "$1z6t9hkl$" ?), de l'entropie de la clé etc. Disons que ce n'est pas la qualité de ce que vous trouvez qui m'intéresse mais le fait que votre code puisse, avec du temps, avoir la possibilité de trouver.
2. **Q :** Doit-on prendre plusieurs fonctions de distances possibles ? **R :** Oui, si vous avez le temps, sinon une cela me suffit mais avec plusieurs tests.
3. **Q :** Doit-on prendre tous les algos de hashage ? **R :** Non ! Non ! Non ! Restons *raisonnable*.
4. **Q :** Doit-on prendre tous les cas de bornes possibles ? **R :** Dans l'idéal oui, mais la réalité c'est que vous n'aurez pas le temps ...
5. **Q :**Doit-on essayer Floyd et Floyd approché ? **R :** Si possible oui, bien sÃżr. Mais un seul des deux est demandé.
6. **Q :** [...] les sous chaines commune ne sont pas aux même emplacement pour la tortue et le lièvre. Est-ce quand même pertinent ? **R :** Pas grave du tout, c'est même probablement le cas le plus fréquent, par contre, là cela devient **très très très** intéressant de voir si le schéma se répète avec d'autres chaines similaires dans leur construction (cf remarques sur projet dans la section 3.2).

4 Contexte

4.1 Cycles

Soit f une application (non linéaire) d'un ensemble *fini* X (de taille n) dans lui-même : $f : X \to X$. Si on choisit $x_0 \in X$ et que l'on considère la suite $\{x_0, x_1, x_2 \cdots\}$ définie par :

$$x_{i+1} = f(x_i). \tag{1}$$

alors cette suite est *cyclique*. On dit *ultimement cyclique* car au début il y a un "temps" de démarrage pourrait-on dire.

4.2 *Cyclus* ?

Cycle : (Nom commun 1) Du latin cyclus emprunté au grec ancien kyklos ("cercle, rond, ronde").

Toute suite construite de manière déterministe avec une équation du type (1) sur un ensemble *fini* X, par construction, possède un cycle, *i.e.* il existe un entier n tel que :

$$x_{i+n} = f(x_n). \tag{2}$$

L'image (5.1) donne une illustration graphique de l'existence d'un cycle.

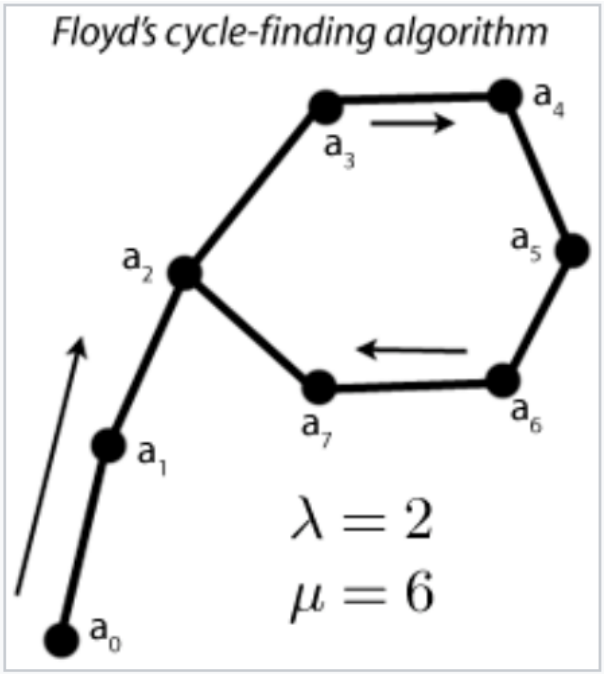

FIGURE 1 – Pseudo-code de l'algorithme de Floyd (Wikipedia)

5 L'algorithme de recherche de cycle de Floyd

5.1 Algorithme du Lièvre et de la Tortue

Appelé aussi l'*algorithme du Lièvre et de la Tortue*, l'algorithme de détection de cycle de Floyd, est un algorithme pour détecter un cycle (ou disons un multiple de la longueur ce cycle) dans une suite récurrente engendrée par toute fonction f définie d'un ensemble fini X dans lui-même.

On vous laisse le soin de voir sur Wikipedia (ou autre) la variante de cet algorithme qui, en plus de trouver un multiple de la longueur ce cycle, calcule la longueur exacte de ce cycle.

```
entrée : une fonction f, une valeur initiale a_0
sortie : indice m avec a_m = a_2m
détection-cycle-floyd(f, a_0)
          m = 1
          tortue = f(a_0)
          lièvre = f(f(a_0))

          tant que tortue != lièvre faire
                m = m + 1
                tortue = f(tortue)
                lièvre = f(f(lièvre))

          retourner m
```

FIGURE 2 – Pseudo-code de l'algorithme de Floyd (Wikipedia)

5.2 Un exemple

Soient $p = 101$, et f l'application de $X = \mathbb{Z}_{101}$ dans lui même ($f : X \to X$) définie par :

$$f(x) = x^2 + 1 \bmod p \tag{3}$$

On choisit $x_0 = 15 \in X$ et considère la suite $\{x_0, x_1, x_2 \cdots\}$ définie par :

$$x_{i+1} = f(x_i) = x_i^2 + 1 \bmod p. \tag{4}$$

On obtient la suite périodique (la période est en gras, indiquée deux fois et séparée par ||) :

$$\{\mathbf{15}, \mathbf{24}, \mathbf{72}, \mathbf{34}, \mathbf{46}, \mathbf{97}, \mathbf{17}, \mathbf{88}, \mathbf{69} || \mathbf{15}, \mathbf{24}, \mathbf{72}, \mathbf{34}, \mathbf{46}, \mathbf{97}, \mathbf{17}, \mathbf{88}, \mathbf{69}, \mathit{15}, \mathit{24}, \mathit{72} \cdots\}$$

6 L'algorithme de recherche de cycle *approché* de Floyd

Nous allons nous intéresser à une variante du problème de recherche d'un cycle d'une application. Au lieu de chercher un cycle exact nous allons modifier le critère d'arrêt avec un critère *d'approximation* au lieu d'un critère d'égalité :

```
Tant Que Distance(Tortue,Lievre) <= Borne
      m = m + 1
Tortue = f(Tortue)
Lievre = f(f(Lievre))
Fin du Tant Que
```

oÃź *Distance* donne une mesure de la proximité entre *Tortue* et *lievre* et *Borne* donne une mesure de la précision qu'on cherche.

Donnons un exemple avec la suite définie dans $\mathbb{Z}_{101}$, si on choisit $Borne = 2$ et :

$$\text{Distance}(Tortue, Lievre) = |Tortue - Lievre|.$$

On a :

$$\{15, 24, 72, 34, 46, 97, 17, 88 \cdots\}$$

alors, 17 vérifie : $\text{Distance}(Tortue, Lievre) = |15 - 17| \leq 2$.

7 Plus longue sous-chaine commune

Soient deux chaine de caractères, par exemple "xyzabcdef" et "abcdefg", la plus *longue sous-chaine commune* est **"abcdef"** car on ne s'intéresse qu'aux sous-chaines de caractères consécutives. On peut généraliser le problème en recherchant :

1. Soit la plus *longue sous-chaine commune* de caractères
2. Soit la plus *longue sous-chaine commune* de bits.

8 Outils

8.1 Module crypto ?

Il est vivement conseillé de choisir le module *cryptography* qui est très bien documenté. Mais il vous est possible de préférer et donc d'utiliser *cryptdodom*. Cela permettra de faire des comparaisons.

8.2 Module *difflib* ?

Le module *difflib* [1] contient a priori tout ce qu'il vous faut, mais vous pouvez tout recoder si vous le voulez.

1. https ://docs.python.org/3/library/difflib.html