ESIEA

PROJETS "Cryptologie & Appplication"

Enseignants
R.ERRA

MS SIS

2022-2023

Commentaires :
Date limite: 02 Janvier 2023
2) Ce document sera mis à jour à chaque question pertinente qui sera posée.
(3) Chaque nouvelle version sera "datée" avec la date dans le nom du fichier.

Table des matières

1 PROJET=EXAMEN

Il n'y aura pas d'examen. Il y aura une seule note : une note de projet !

2 Date limite et rendu ? (*Deadline etc.*)

2.1 Date limite/Deadline

1. Elle est fixée au **Lundi 02 Janvier 2022 : 23h42**
2. En cas de changements on avisera.

3 Rendre votre travail (*Return*)

Tout sera remis sur gitlab.esiea.fr. Quelques éléments :

1. Vous devrez déposer ce qui vous est demandé (exemple : votre fichier "zip" (qui contiendra votre Notebook Jupyter Python et le reste ... ainsi que tous vos codes et fichiers résultats) sur `gitlab.esiea.fr`
2. En me donnant l'accès [et en laissant privé votre répertoire)]
3. Pas d'emails !
4. Des questions ? Par Teams svp !
5. Mais nous nous autorisons à ne plus répondre les 15 jours précédents la date limite.
6. *Bon courage à toutes et à tous.*

4 Votre travail

4.1 Code python

Vous devez choisir un projet, choisir une bibliothèque crypto, fixer les différents paramètres et *coder* en python, si possible dans un notebook Jupyter.

4.2 Comment travailler ?

```
# Introduction
* Les travaux  sont à réaliser:
    * soit par deux pour le projet #1
    * soit individuellement  p#2 et #3our les autres projets
    * Toujours sur Moodle.
* Un _seul_ sujet est à choisir.
* Il y aura uhe (petite) soutenance.
* Le Projet remplace l'examen.
* Nous notons la qualité scientifique du travail, la propreté du code
et la qualité  de la documentation ainsi que la qualité de la présentation.
* Une "métrique" vous sera proposée pour le rendu
(nombre de caractères minimum/maximum, etc.).
* Pour toutes questions, nous vous invitons à nous contacter par mail
(ou par Teams!) ou en direct.
```

5 Projet #1 - Certificats, web et recherche de clés "compromises"

5.1 Version en Français

- Récupérer au moins n millions de certificats X509 en utilisant par exemple _Certificate Transparency_ ($n \geq 1$)
- Vous pouvez aussi écrire un crawler et en télécharger les certificats de _Let's Encrypt_ ou en utiliser leurs scripts (un peu long)
- Créer une base de données pour stocker les clés RSA/DSS/DH et leur origine
- Trier les clés par taille et par Algorithme (RSA, DSS, Diffie-Hellman)
- Recherche de doublons (clés identiques pour RSA et pour DH)
- Recherche de clés différentes mais ayant un facteur commun
  - Lancer Batch GCD ([REF-11], [REF-12]) sur les autres clés,
  - il est possible de trouver soit p soit q commun a une clé sachant que n=p q
  - Bacth GCD existe en Python et en C++, ne pas le recoder!
- Collecter le plus possible de certificats.

5.2 Où trouver les certificats ?

1. Un indice : si vous allez sur
2. https://crt.sh/?Identity=%25&iCAID=1000
3. vous aurez des certificats
4. et il est facile de comprendre que
5. sur
6. https://crt.sh/?Identity=%25&iCAID=1001
7. il ya aussi des certificats, etc.
8. et ça commence à https://crt.sh/?Identity=%25&iCAID=0001
9. ensuite, c'est de la programmation web ...et il y a quelque part des outils déjà prêt (ort probablement) pour extraire les certificats, ouis les clés ...
10. Rappel : ce sont les **clés RSA** qui nous intéressent
11. Sinon, vous avez aussi https://certstream.calidog.io/ qui a priori permet aussi de télécharger des certificats ...

5.3 RFC 5280

https://www.ietf.org/rfc/rfc5280.txt

4.1.2.7. Subject Public Key Info

> This field is used to carry the public key and identify the algorithm with which the key is used (e.g., RSA, DSA, or Diffie-Hellman). The algorithm is identified using the AlgorithmIdentifier structure specified in Section 4.1.1.2. The object identifiers for the supported algorithms and the methods for encoding the public key materials (public key and parameters) are specified in [RFC3279], [RFC4055], and [RFC4491].

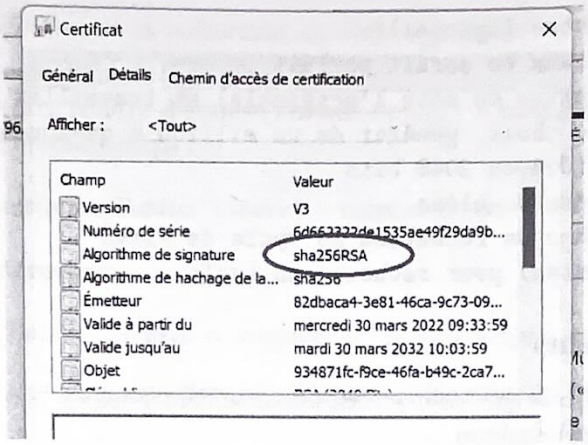

FIGURE 1 – An exemple of a X509 certificate

Et

4.1.1.2. signatureAlgorithm

The signatureAlgorithm field contains the identifier for the cryptographic algorithm used by the CA to sign this certificate. [RFC3279], [RFC4055], and [RFC4491] list supported signature algorithms, but other signature algorithms MAY also be supported.

An algorithm identifier is defined by the following ASN.1 structure:

```
AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

The algorithm identifier is used to identify a cryptographic algorithm. The OBJECT IDENTIFIER component identifies the algorithm (such as DSA with SHA-1). The contents of the optional parameters field will vary according to the algorithm identified.

This field MUST contain the same algorithm identifier as the signature field in the sequence tbsCertificate (Section 4.1.2.3).

5.4 Bonus

Si vous trouvez des facteurs communs, regarder s'il ont un pattern ...

6 Projet #2 - Génération de clés, VM et entropie

6.1 Version en Français

* Télécharger une VM Linux (avec ce qu'il faut pour générer des clés RSA,

```
donc OpenSSL ou un autre logiciel)
* Une "vieille" VM Linux ce serait parfait ...
* Faire une copie (mettre de côté l'originale) et travailler avec la copie
* Dans la VM de votre choix, générer de un million à quelques millions de clés RSA [pensez à
de 256, puis 512 ou 1024 ou 2048 bits
* Vérifier s'il y a des doublons.
* Utilisez l'algorithme de recherche de cycle de Floyd
(voir sections suivantes) pour recheche un cycle.
```

6.2 Liens intéressants

1. VM Ubuntu 2010 : https://old-releases.ubuntu.com/releases/lucid/ : download the version PC (Intel x86) desktop
2. A very interesting website for "old" linux : https://soft.lafibre.info/

7 #1 et #2 Combien de clés ?

```
* Pour le projet  #1:
    - Il est fort possible qu'en dessous de quelques millions de clés, on ne trouve rien.
    - Donc, ne pas paniquer !
    - En fait ce serait une bonne nouvelle.


* Pour le projet  #2:
    - Tester combien de temps faut-il sur votre VM pour générer 2000 clés
    - Tester combien de temps faut-il sur votre VM pour générer 4000 clés
    - Tester combien de temps faut-il sur votre VM pour générer 8000 clés
    - Extrapoler : combien de temps faut-il sur votre VM pour générer 1000000 clés
    - Si c'est _trop long_ : réduire le nombre de clés
```

8 Projet #3 - Cycles et Pseudo cycles et Collisions pour algorithmes de chiffrement et de hash (MD5, SHA1, DES, AES, etc.)

8.1 Introduction

Soit la problèmatique suivante :

1. Choisir un algorithme $\mathcal{A}$ parmi : MD5, SHA1, SHA256, DES, etc.
2. Choisir une valeur de départ x_0 et itérez ...i.e. $x_1 = H(x_0)$, $x_2 = H(x_1)$ etc.
3. Et recherchez un cycle (ou un quasi-cycle), ce qui peut donner une vraie "collision". si $x_k = x_j$, avec $k > j$ alors $H(x_{k-1}) = H(x_{j-1})$ et si $x_{k-1} \neq H(x_{j-1}$
4. On peut aussi choisir (mais cela risque de prendre du temps) pour un algorithme comme MD5 une valeur de de départ x_0 de longueur inférieur à 8 octest, et, en plus de rechercher un cycle, on peut comparer x_k avec x_0 (ce qui augmente la probabilité d'avoir une vraie collision)
5. En pratique : choisir pour valeur de départ (x_0) adaptée à l'algorithme en nombre de d'octets ou de bits, aléatoire (ou pas : à vous de choisir).

6. Considérer l'application $\mathcal{A}$ et rechercher un cycle *approché*.
7. 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)

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

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

Précisions :

1. Le test le plus intéressant à faire avec un algorithme de hashage est évidemment lorsqu'on utilise Distance($Tortue, Lievre$) = longueur de la plus *longue sous-chaîne 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 :

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

   Ainsi, on aura donc Distance($Tortue, Tortue$) = 0.
2. Si vous désirez utiliser

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

   dans ce cas il **ne faut pas** normaliser, on aura 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 chaîne de 128 bits par exemple si on a Distance($Tortue, Tortue$) ≤ 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 : on choisit $x0 =$ "*toto*" et on calcule un cycle approché pour un algorithme de hachage (donc : pas de clé). Ok, pourquoi pas. Mais alors des questions se posent :
   — Le calcul donne t-il la même chose ou pas pour "toute" chaîne $x0 =$ "$xyxy$", 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" chaîne $x0 =$ "$xyxyxy$", 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 chaîne décalée à droite "upup" ? (schéma : $t- > u$ et $o- > p$) ?
   — ou encore qu'observe t-on pour la chaîne décalée à gauche "snsn" ? (schéma : $t- > s$ et $o- > n$) ?
   — etc.
4. Pour celles et ceux qui désirent plutôt étudier un algorithme de chiffrement, on peut se poser le même genre de questions (existence de schémas) sur la chaîne initiale et/ou sur la clé !

5. À vous de choisir ce que vous voulez mettre en évidence (présence ou absence de schémas) **mais** par contre, je rappelle que ce projet remplace un examen d'1h30, donc je ne vous demande pas de tout chercher, tout tester.

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.

8.2 Quelques questions/réponses

1. **Q :** À partir de quelle longueur de sous-chaîne commune le "cycle" trouvé est-il pertinent ? **R :** Bonne question mais cela dépend vraiment de l'algorithme choisi, de l'entropie de votre chaîne 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 sûr. 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 ça devient **très très très** intéressant de voir si le schéma se répète avec d'autres chaînes similaires dans leur construction (cf remarques sur projet 1 dans la section 8).

8.3 Contexte

8.3.1 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.

8.3.2 Cycles ?

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 (8.3.2) donne une illustration graphique de l'existence d'un cycle.

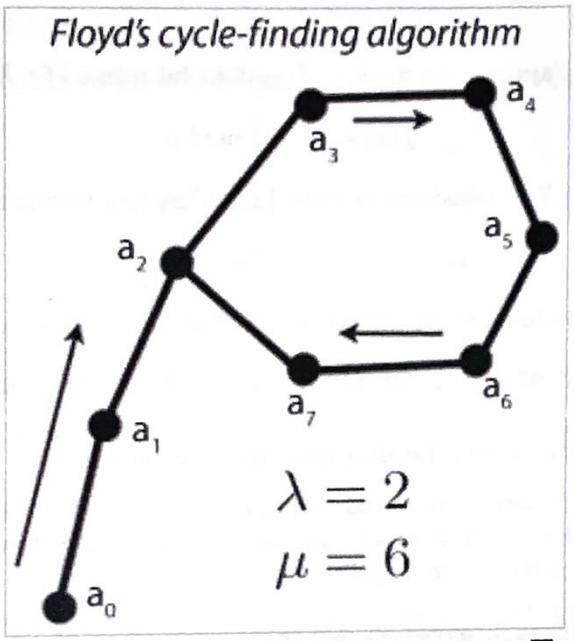

FIGURE 2 – Pseudo-code de l'algorithme de Floyd (Wikipedia)

8.4 L'algorithme de recherche de cycle de Floyd

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.

```
entrée : une fonction f, une valeur initiale a0
sortie : indice m avec am = a2m
détection-cycle-floyd(f, a0)
        m = 1
        tortue = f(a0)
        lièvre = f(f(a0))

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

        retourner m
```

FIGURE 3 – Pseudo-code de l'algorithme de Floyd (Wikipedia)

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.

8.4.1 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 ||) :

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

8.4.2 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
```

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

8.4.3 Plus longue sous-chaîne commune

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

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

8.5 Outils

8.5.1 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.

8.5.2 Module *difflib* ?

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

9 Que devez vous rendre ?

1. Votre code
2. Les fichiers nécessaires à l'exécution de votre code
3. Un ficheir readme.md sur votre gitlab.esiea.fr
4. Un (petit) fichier résumant vos résultats.
5. Si vous faites un notebook Jupyter, vous pouvez inclure la description de vos résultats dans le fichier en utlisant des cellules en Markdown.

10 Références/Webography

[REF-1] : [https://cryptography.io/en/latest/] (https://cryptography.io/en/latest/)
[REF-2] : [https://github.com/dieggoluis/rsa-attack] (https://github.com/dieggoluis/rsa-attack)
[REF-3] : [http://images.math.cnrs.fr/Cryptris-1-2-Comprendre-une-des-techniques-les-plus-sophi
[REF-4] : [http://images.math.cnrs.fr/Cryptris-2-2-Les-dessous-geometriques-de-Cryptris-la-cryp
[REF-11] https://facthacks.cr.yp.to/batchgcd.html
[REF-12] https://github.com/zugzwang/batchgcd
[REF-13] https://letsencrypt.org/fr/docs/

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