Séance 6 - Travaux Pratiques
Cryptanalyse

Yann Rotella

2026

Seul.e 45 minutes

Exercice 1. *Génération de clefs, recherche exhaustive et code.*
Un étudiant du cours a voulu réaliser une demande d'amélioration de sa note au contrôle continu. Pour se faire bien voir du professeur de cryptographie, cet étudiant a chiffré son message avec AES et a donné la clef secrète au professeur.

Votre objectif final est de trouver la réponse exacte du professeur.

Pour ce faire, on vous donne un bout de code `Python`. Ce bout de code utilise la librairie `cryptography`, probablement à télécharger avant de commencer.

(1) Récupérer le bout de code et faites-le tourner.

Pour simuler l'observation d'un autre étudiant malicieux qui aurait écouté sur le canal de communication, on vous donne (dans le code, en commentaire) ce que le code a donné avec la bonne clef : les message réellement transmis.

(2) Lire le code. Combien d'entropie y'a t'il dans la clef secrète ?

(3) À quoi sert le Nonce ? Quel problème y'a t'il dans ce code à ce sujet ?

(4) Quelle est la taille de la réponse ? Expliquer. Est-ce un problème ? Pouvons-nous le gérer ?

(5) Retrouver la réponse du professeur.

En groupe - cryptanalyse

Exercice 2. *Mode compteur.*
L'objectif est le même qu'au premier exercice. Avec le bout de code de l'exercice 2 (`exo2.py`), retrouvez le message renvoyé et les éléments secrets. Pouvez-vous identifier des identifications de la personne qui a écrit le message ?

Exercice 3. *Fonctions de hachage.*
Dans cet exercice, nous allons regarder les attaques génériques et essayer de chercher des collisions, de manière générique sur les foncions de hachage. Prenez le code correspondant à l'exercice 3 `exo3.py`. On utilise SHAKE qui est une eXtandable Output Function (XOF) et on va regarder la complexité de résolution et le temps de recherche d'une collision en fonction de la taille de la sortie.

(1) On utilise SHAKE128. N'hésitez pas à regarder sur internet et à comprendre ce que c'est !

(2) En utilisant des listes contenant des `digest` de chaînes de caractères aléatoires, écrire une fonction qui renvoie une collision sur un `digest` de taille donnée en paramètre.

(3) Tracer en python le temps d'éxécution moyen de cette fonction en fonction de la taille du digest.

Il est possible de réaliser la recherche de collisions sans mémoire. Pour cela, on exploite généralement un algorithme très pratique de détection de cycles dans une suite définie par $u_n = f(u_{n-1})$ avec $u_0 = x$ pris aléatoirement et $f : X \to X$ un espace de taille arbitraire.

(4) Lire et regarder ces algorithmes sur les pages wikipedia : `https://fr.wikipedia.org/wiki/Algorithme_du_li%C3%A8vre_et_de_la_tortue` ou en anglais `https://en.wikipedia.org/wiki/Cycle_detection`

(5) Expliquer pourquoi trouver le cycle de $(u_n)_{n\geq 0}$ permet de retrouver aussi une collision.

(6) Ici on cherche une collision avec une fonction qui n'a pas le même espace de départ et d'arrivée. Si c'était le cas on pourrait exploiter l'algorithme du dessus. Mais... pouvez-vous trouver une solution à cela ?

(7) Faites une fonction de recherche de collision sans mémoire qui exploite l'algorithme précédant.

(8) Chercher des collisions sur SHAKE128, en augmentant pas à pas la taille de sortie. Quels temps obtenez-vous ? Est-ce cohérent par rapport à la recherche avec mémoire ?

(9) Tracer les courbes de la même manière que précédemment et regardez jusqu'où vous pouvez aller pour trouver des collisions.

Exercice complémentaire

Exercice 4. *Les multi-collisions.*

Une idée pour améliorer la sécurité de fonctions de hachage peut consister à concaténer la sortie de deux fonctions de hachages, permettant alors une sortie de plus grande taille. En 2004, A. Joux a cependant montré que ce type de construction n'apportait pas nécessairement la sécurité attendue.

(1) Lire l'article des multicollisions intitulé « Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions ». Disponible à l'url suivante : `https://link.springer.com/content/pdf/10.1007/978-3-540-28628-8_19.pdf`

(2) Programmer la recherche de multicollisions sur la concaténation de deux fonctions de hachage bien choisies.