Pablo Rauzy <pr@up8.edu>
https://pablo.rauzy.name/teaching/is

Introduction à la sécurité

TP 4 : Cryptanalyse linéaire

Ce TP n'est pas à rendre.

Dans ce TP :
— Cassage d'un cryptosystème jouet par cryptanalyse linéaire.

Exercice 0.

Démarrage et recommandations.

1. Récupérez les fichiers nécessaires à ce TP sur `https://pablo.rauzy.name/teaching/is/08-tp-cryptanalyse-lineaire.tgz`.
2. Ce TP est à faire en Python.
3. Faites les questions dans l'ordre et pensez à tester votre code.
4. N'hésitez jamais à rajouter de la sortie de debug pour comprendre tout ce qui se passe.

Exercice 1.

Étude de notre cryptosystème jouet.

1. Le but de cet exercice est de comprendre le fonctionnement du cryptosystème ToyCipher implémenté dans le fichier `toycipher.py`.
   → Ouvrez ce fichier dans votre éditeur de texte favori (pourvu qu'il soit libre).
2. → À quel famille ce cryptosystème appartient-il ?
3. → Quel type de construction de cryptosystème est utilisé ?
4. Analysons donc ce cryptosystème plus en détail.
   (a) → Quelle est la taille du bloc ?
   (b) → Quelle est la taille de la clef ?
   (c) → Combien de tour effectue-t-il ?
   (d) → En quoi consiste chaque tour ?
   (e) → En quoi consiste l'étape de substitution (i.e., quelle est la boîte S) ?
   (f) → En quoi consiste l'étape de permutation ?

Exercice 2.

Comprendre la nécessité de la cryptanalyse linéaire.

1. Pour commencer, réduisons le nombre de tour de ToyCipher à un.
   (a) → Quelle type d'attaque permet de casser trivialement le chiffrement dans ce cas là ?
   (b) → Comment ?
2. Repassons donc à deux tours.
   Il ne suffit pas pour casser l'algorithme de chiffrement de répéter deux fois l'opération de la question précédente.
   → Pourquoi ?
3. Si on connaissait la clef du premier tour, on pourrait calculer la sortie du premier tour — qu'on appellera dorénavant t — et ensuite faire notre attaque de la question 1 sur le second tour.
   (a) Il y a un nombre raisonnable de clef de tour possible.
   → Combien ?
   (b) → Est-il pour autant possible de simplement toutes les essayer pour trouver la clef du premier tour ? Pourquoi ?

4. Sur notre exemple jouet, il serait raisonnable de faire une attaque par force brute sur la clef complète, mais ce n'est pas possible sur un vrai cryptosystème qui utilise une clef plus grande.
   → Si on définit "une étape" comme l'exécution d'un tour de notre algorithme de chiffrement, combien d'étapes sont nécessaires au maximum pour attaquer ToyCipher par force brute ? Justifiez votre réponse.
5. C'est important d'être rigoureu·x/ses ! Mettons donc en pratique la méthode scientifique.
   À la question précédente, nous avons formulé une *hypothèse réfutable* concernant le nombre d'étapes nécessaires au maximum pour attaquer ToyCipher par force brute.
   Cette hypothèse, qu'il nous est maintenant interdit de changer, correspond en fait à une méthode d'attaque : si on met en œuvre la méthode à laquelle vous avez pensé pour formuler votre hypothèse, on fera bien ce nombre là d'étapes (ou alors vous vous êtes complètement mélangé les pinceaux !).
   La question est : est-ce que cette hypothèse est la bonne ? C'est à dire : si on met en œuvre la méthode correspondante, est-ce qu'on est sûr de trouver la bonne clef ?
   On peut y répondre car l'hypothèse est réfutable : il existe un moyen de tester si elle est fausse, et dans notre cas il s'agit d'exhiber un cas où il faut strictement plus que le nombre d'étapes supposé maximum par l'hypothèse pour trouver la clef (ou alternativement, qu'on ne trouve pas la clef en ce nombre d'étape supposé maximum).
   (a) → Mettez rapidement en œuvre la méthode à laquelle vous avez pensé pour formuler votre hypothèse.
   (b) → Que remarquez vous ?
   (c) → Si jamais votre hypothèse est réfutée, pourquoi ?
   (d) → Si nécessaire, rerépondez à la question 4.
6. La cryptanalyse linéaire est une attaque à clair connue (KPA). On aura donc besoin d'un certains nombres de paires message et chiffré correspondant.
   → Écrivez une fonction qui étant donné une clef et un nombre n de paire à générer, renvoie une liste de n paires de message aléatoirement choisi et du chiffré correspondant avec la clef donnée.

Exercice 3.

Approximation linéaire de la boîte S.

1. Supposons maintenant qu'il n'est pas raisonnable d'attaquer ToyCipher par force brute. Il nous faut donc trouver un moyen plus intelligent d'attaquer notre algorithme de chiffrement.
   En l'occurrence, on voudrait un moyen de deviner intelligemment des valeurs probables pour k_0, puisqu'on a vu que cela nous aiderait beaucoup à retrouver k_1.
   L'idée de la cryptanalyse linéaire est d'approximer aussi bien que possible ce qu'on cherche par une fonction linéaire pour pouvoir faire des estimations éclairées par notre approximation.
   Grosso-modo (en ce qui nous concerne, en tant qu'informaticien·nes), une fonction linéaire, c'est une fonction suffisamment "simple" pour nous permettre par observation de deviner ce qui se passe dedans, au sens où on est capable de dire que si l'entrée de la fonction a une propriété X, alors on sait que la sortie a une propriété Y.
   Par exemple, la fonction $f : x \mapsto x+1$ même si on ne pouvait l'utiliser qu'en boite noire, on pourrait en extraire la propriété que si son entrée en paire, alors sa sortie est impaire, et vice-versa.
   Cependant, on sait que sauf faille majeure dans la conception de notre cryptosystème, notre fonction de tour n'est pas linéaire.
   → Pourquoi ?
2. On cherche donc à approcher notre fonction de tour par une approximation linéaire. La propriété que l'on va utiliser est la parité binaire des nombres en entrée et en sortie de la boîte S. La parité binaire d'un nombre est la parité du nombre de bits à 1.
   Si notre boîte S n'est pas trop mal faite (et c'est le cas), on aura pas de propriété intéressante (comme une égalité, par exemple) directement. En revanche, on peut regarder si ce n'est pas le cas pour un sous-ensemble des bits.
   On cherche donc deux valeurs $mask_i$ et $mask_o$ telles que si on masque l'entrée de la boîte S avec $mask_i$ (avec un **and** bit à bit, pour sélectionner seulement un sous-ensemble des bits), la parité du résultat soit la même que celle de la sortie de la boîte S masquée avec $mask_o$.
   → Écrivez une fonction qui calcule pour chaque couple $(mask_i, mask_o)$, pour combien des 16 paires entrée/-sortie de la boîte S on a une égalité de parité.
3. On veut maintenant choisir la meilleure paire $(mask_i, mask_o)$. Cette paire doit évidemment faire partie de celles qui ont eu le plus haut score à la question précédente, et on veut choisir celle qui prend le plus de bits en compte (donc avec le plus de 1 dans l'écriture binaire des masques, et on exclue évidemment les masques à 0).
   → Écrire une fonction qui trouve la meilleure paire $(mask_i, mask_o)$ de masques en se basant sur le tableau de score de la question précédente.

Exercice 4.

Trouver des candidats pour k_0.

1. Maintenant que nous avons une approximation linéaire et notre boîte S, nous allons nous en servir pour trouver des valeurs potentielles de k_0.

   Pour cela on va avoir besoin d'un certains nombre de paires de message et chiffré correspondant.

   → En utilisant votre fonction de la question 6 de l'exercice 2, générez 16 paires de message/chiffré connues.

2. On veut maintenant attribuer un score à chacun des k_0 possibles.

   Pour chaque k_0 possible, pour chaque paire (m, c) de message/chiffré connue, on calcule t le résultat du premier tour de chiffrement de m avec ce k_0, et on vérifie ensuite si l'approximation linéaire de notre boîte S permet de valider le second tour. C'est à dire si notre égalité de parité binaire tient entre t et c.

   Cependant attention, c a été `xor`é (avec le vrai k_1) avant de passer dans la boîte S. Le `xor` étant une opération linéaire il ne changera pas fondamentalement notre propriété, mais il peut inverser la parité binaire et on peut donc se retrouver a devoir compter les inégalités plutôt que les égalités.

   Par exemple, si notre approximation linéaire marche dans 14 cas sur 16, il se peut qu'avec l'ajout de la clef, elle ne marche plus que dans 2 cas sur 16.

   On cherche donc une fonction de score qui maximise dans les deux cas (celui où on a beaucoup de cas qui marche, et celui ou on en a très peu).

   → Avez vous une idée de telle fonction ?

3. Les candidats qu'on retient pour k_0 sont tous ceux qui obtiennent le plus grand score de la question précédente.

   → Écrivez une fonction de sélection de candidats pour k_0. Cette fonction prend en entrée la liste des paires message/chiffré connues, et les $mask_i$ et $mask_o$ sélectionnés précédemment.

Exercice 5.

Attaque par cryptanalyse linéaire.

1. Nous avons maintenant tous les ingrédients pour monter notre attaque.

   Pour chacun des candidats k_0, on peut facilement retrouver un potentiel k_1 correspondant.

   → Comment ?

2. On a maintenant une paire (k_0, k_1) candidate à être la clef de chiffrement.

   → Écrivez la fonction qui, étant donné les k_0 candidats, trouve la clef (si possible).

3. → Branchez maintenant tout ce que vous avez écrit ensemble et vérifiez que votre attaque fonctionne.

Exercice 6.

Analyse.

1. → Combien d'étapes de calcul sont nécessaires avec l'attaque par cryptanalyse linéaire (pour la même définition d'"étape" qu'à la question 4 de l'exercice 1).

2. Comparez ce nombre avec l'attaque par force brute.

   → Qu'en dites-vous ?

3. → Pensez-vous que les calculs nécessaires pour l'approximation linéaire de la boîte S doivent être compté dans la complexité de l'attaque ? Pourquoi ?

4. → Pouvez-vous proposer une boîte S mieux conçue qui résisterait mieux à cette attaque ? Essayez avec votre code.