ENAC
ECOLE NATIONALE DE L'AVIATION CIVILE

STIC – Introduction à la cryptographie

Jean-Christophe Deneuville
jean-christophe.deneuville@enac.fr

7 décembre 2023

En cours, nous avons vu comment un cryptosystème sûr pouvait être cassé s'il était mal implémenté (analyse de la consommation de temps, de courant, injection de fautes, ...). Dans ce TP, nous nous intéresserons à un autre aspect de la sécurité : celui où l'adversaire (vous) a en sa possession un algorithme, et cherche à en découvrir les vulnérabilités.

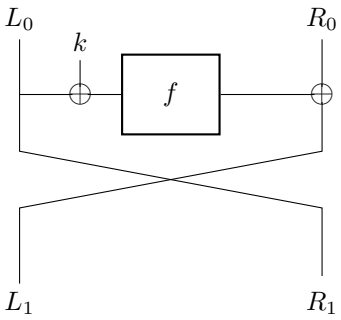
Vous pouvez utiliser le langage de programmation qui vous convient, je recommande Python pour son agilité/flexibilité.

Warm-Up

Le but de cet exercice d'échauffement est de se familiariser aux schémas symétriques de chiffrement par blocs utilisant des réseaux de Feistel. Si vous ne comprenez pas cette phrase c'est normal :)

Schéma de Feistel
On considère deux registres de n bits (4 par exemple) L_0 et R_0 en entrée, une clé k de même longueur et une fonction f de $\{0,1\}^n$ dans $\{0,1\}^n$.
Les registres de sortie L_1 et R_1 sont obtenus comme illustré ci-contre :

$$L_1 \leftarrow R_0 \oplus f\left(L_0 \oplus k\right) \quad \text{et} \quad R_1 \leftarrow L_0.$$



Un réseau de Feistel est un enchaînement de plusieurs schémas de Feistel, les fonctions f et les clés k dites "de tour" peuvent être différentes.

Questions

1. Comment inverser un schéma de Feistel (c'est-à-dire retrouver L_0 et R_0 à partir de L_1 et R_1 et k) ?
2. Cela nécessite-t-il que la fonction f ait des propriétés particulières ? Si oui, lesquelles ? Si non, pourquoi ?
3. Comment inverser un réseau de Feistel à 2 tours (retrouver L_0 et R_0 à partir de L_2, R_2, k_0 et k_1) ?

Exercice 1

Les réseaux de Feistel sont largement utilisés dans les cryptosystèmes symétriques, car en impliquant suffisamment de tours, ils permettent d'obtenir de bonnes propriétés de "confusion-diffusion".

La confusion consiste à rendre une information inintelligible (substitution d'une lettre par une autre), et la diffusion permet de propager la première propriété à tout le bloc.

Considérons un cryptosystème simplifié $\mathcal{S}$ à un seul registre de 4 bits, admettant une clé $k = (k_0, k_1)$ de 8 bits, et utilisant pour fonction f celle définie par la table suivante :

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$f(x)$	9	b	c	4	a	1	2	6	d	7	3	8	f	e	0	5

À chaque tour, l'entrée est xorée avec la clé de tour, puis la fonction f est appliquée afin de produire la sortie du tour.

1. Implémentez la fonction de tour f : `sbox(x)`. Elle prend en entrée un entier x sur 4 bits et retourne $f(x)$ sur 4 bits conformément à la table ci-dessus.
2. Implémentez la fonction de dérivation de clé : `split(key)`. À partir d'une clé k de 8 bits, retourne 2 sous-clés de tour k_0 et k_1 sur 4 bits chacune.
3. Implémentez la fonction de tour : `round(state, roundKey)`. À partir d'un état `state` et d'une clé de tour `roundKey` en entrée, modifie l'état conformément à la description ci-dessus.

4. Implémentez la fonction de chiffrement : `encrypt(plaintext, key)`. Assemblez-les briques élémentaires implémentées aux questions précédentes.

Sanity-check : vérifiez que `encrypt(0x5, 0xb3) = 0x4`.

Exercice 2

Il ne s'agit pas seulement d'assurer confusion et diffusion pour correctement chiffrer, encore faut-il pouvoir efficacement déchiffrer...

1. Dans le cas d'un schéma de Feistel (à 2 entrées/sorties), la fonction f ne nécessite aucune propriété. Dans notre cas simplifié, quelle propriété est-il nécessaire d'avoir sur f pour pouvoir déchiffrer ?
2. Implémentez cette propriété.
3. Implémentez un tour de la fonction de tour du déchiffrement.
4. Implémentez la fonction de déchiffrement : `decrypt(ciphertext, key)`.

Rappel de cours

Il existe différents modèles d'attaque en fonction des ressources dont dispose l'adversaire :

— attaque sur texte chiffré seul (ciphertext-only, COA),

— attaque à texte clair connu (known-plaintext attack, KPA),

— attaque à texte clair choisi (chosen-plaintext attack, CPA),

— attaque à texte chiffré choisi (chosen-ciphertext attack, CCA),

— attaque adaptative à texte chiffré choisi (adaptative chosen-ciphertext attack, CCA2),

Exercice 3

Il est généralement difficile/impossible de repérer une vulnérabilité d'un schéma sur un nombre élevé de tours (cf DES, AES, ...). Il est plus simple de réduire le nombre de tours afin d'identifier une faiblesse, puis ré-augmenter progressivement le nombre de tours afin d'essayer d'exploiter la faille trouvée.

1. Considérons un instant notre schéma $\mathcal{S}$ réduit à 1 tour. Quel est le type d'attaque le plus faible (l'adversaire a besoin d'un minimum d'information) permettant de cryptanalyser cette version réduite ? Expliquer l'attaque ?
2. Reprenons notre version sur 2 tours. Le type d'attaque identifié à la question précédente est-il toujours fonctionnel ? Si oui, pourquoi ? Si non, pourquoi ?

Soit x_0 l'entrée de l'algorithme de chiffrement, x_1 la sortie du premier tour (et donc l'entrée du second), et x_2 la sortie du second tour. En trouvant la bonne clé de tour au premier tour, nous trouverons le bon x_1, et pourrons donc mener l'attaque de la question 1.

3. Combien y a-t-il de clés de (premier) tour possibles ?
4. Pourquoi ne pouvons-nous pas pour autant réaliser une cryptanalyse par ce biais ?

Exercice 4

Bien que ça ne soit pas le cas dans notre exemple, la cryptanalyse de la clé complète par force brute est généralement hors de portée (clés trop grandes/longues).

Définissons une "étape élémentaire" comme un tour de l'algorithme de chiffrement $\mathcal{S}$.

1. Combien d'étapes élémentaires – au maximum – sont nécessaires pour bruteforcer $\mathcal{S}$ (c'est-à-dire retrouver une clé qui, lorsqu'elle est utilisée dans l'algorithme de chiffrement, fasse correspondre notre clair et notre chiffré) ?

Analysons un peu plus l'algorithme $\mathcal{S}$:

2. Combien y a-t-il de messages clairs possibles ?
3. Combien y a-t-il de clés (complètes) possibles ?
4. Combien y a-t-il de messages chiffrés possibles ?
5. Qu'en concluez-vous ?
6. Corrigez votre réponse à la question 1 de cet exercice.

Exercice 5

L'exercice précédent nous a permis de démontrer qu'un même couple clair/chiffré pouvait être obtenu avec plusieurs clés. (Si vous n'êtes pas convaincu(e), essayez de chiffrer 1 avec les clés `0x47`, `0x74`, et `0xe3`).

1. Écrivez une fonction `bruteforce(m, c)` qui étant donnés un message clair `m` et un chiffré `c` retourne l'ensemble des clés $k = (k_0, k_1)$ possibles telles que `encrypt(m,k)=c`.

Afin de déterminer avec exactitude quelle clé exacte a été utilisée, nous aurons besoin de plus qu'une paire clair/chiffré.

2. Écrivez donc une fonction `known_pairs_gen(k, n)`, qui étant donnés une clé $k = (k_0, k_1)$ et un nombre n retourne une liste de n paires de messages clairs aléatoires avec leur chiffré correspondant.

La technique de cryptanalyse que nous allons maintenant étudier s'appelle la cryptanalyse linéaire.

Exercice 6

Dans la suite, nous supposerons qu'il n'est pas raisonnable de réaliser une attaque par force brute sur notre schéma $\mathcal{S}$ sur deux tours. [1] Puisque nous savons (cf. exo 3.1) qu'il sera facile de retrouver k_1 lorsque nous aurons k_0, c'est sur cette première clé de tour que nous allons concentrer nos efforts.

Paradigme

Une fonction dite "linéaire" a pour caractéristique de préserver certaines propriétés mathématiques (de distance, de corrélation, etc...).
La fonction f étant censée assurer la *confusion* de notre schéma, c'est-à-dire le "brouillage" des informations, celle-ci ne peut donc pas être linéaire. (En termes cryptographiques on appel f une *substitution-box*, abrégé en S-Box).
Le paradigme de la cryptanalyse linéaire est de formuler des approximations par linéarisation, puis de les valider/infirmer par essais successifs pour apprendre certaines propriétés sur le schéma. L'objectif final étant que les informations apprises permettront de cryptanalyser le schéma beaucoup plus rapidement que par force brute.

Nous allons donc nous intéresser à des propriétés singulières de notre Sbox, la plus simple d'entre elles étant la parité des états manipulés.

1. Créez une fonction `parity(x)` qui retourne 1 si le nombre de 1 dans l'écriture binaire de `x` est impair, 0 sinon.

Typiquement, les Sbox de cryptosystèmes correctement conçus déjouent ce genre de tests. Nous pouvons toutefois regarder s'il n'existe pas de corrélation pour un sous-ensemble de cas (autrement dit, seulement certaines positions (ou certains bits)).

Cherchons donc deux valeurs $a = \text{mask}_{\text{input}}$ et $b = \text{mask}_{\text{output}}$ telles que si l'on masque (bitwise `and`) l'entrée de la Sbox avec a, la parité du résultat soit la même que celle de la sortie de la Sbox masquée avec b.

2. Écrivez une fonction `nb_parity(a,b)` qui, étant donnés a et b, calcule le nombre de x vérifiant :

$$\texttt{parity(x \& a)} == \texttt{parity(sbox(x) \& b)}.$$

Maintenant que nous avons un "facteur discriminant", nous cherchons à savoir quelles valeurs de a et b maximisent ce facteur.

3. Écrivez une fonction qui retourne la/les valeur(s) de (a, b) maximisant la fonction `nb_parity(a,b)` quel que soit la valeur `x` considérée.

Maintenant que nous avons des valeurs discriminantes pour a et b, attaquons nous à la recherche de la clé k_0.

Exercice 7

L'étape de détermination de a et b s'appelle une "linéarisation de Sbox" : nous avons approximé – par une fonction simple – une fonction complexe. Cette approximation a été appuyée par une corrélation par observation de couples clairs/chiffrés.

L'objectif de cet exercice est maintenant d'exploiter la brèche ouverte à l'exercice précédent.

1. Notre modèle est-il cohérent ?

(Cette dernière question est monnaie courante chez tout cryptographe/cryptanalyste qui se respecte : nous avons déduis des informations exploitables depuis des hypothèses. Comment pouvons-nous nous assure que nos hypothèses sont valides afin d'être sûr(e)s de pouvoir exploiter les conclusions ?)

1. Hypothèse outrageusement fausse à but uniquement pédagogique...

2. Utilisez la fonction `known_pairs_gen(k, n)` pour générer 16 couples clairs/chiffrés.

On veut maintenant attribuer un score à chacune des clés de tours k_0 possibles.

Pour chaque k_0 possible, pour chaque paire (x_0, x_2) de message/chiffré connue, on calcule x_1 le résultat du premier tour de chiffrement de x_0 avec la k_0 retenue, et on vérifie ensuite si l'approximation linéaire de notre Sbox permet de valider le second tour, autrement dit si notre égalité de parité binaire tient entre x_1 et x_2.

Cependant attention, x_2 a été xoré (avec le vrai k_1) avant de passer dans la Sbox. 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).

3. Comment procéderions-nous pour une telle fonction ?

Les candidats que l'on retiendra pour k_0 sont tous ceux qui maximisent le score de la question précédente.

4. Écrivez une fonction de sélection de candidats pour k_0. Cette fonction prend en entrée la liste des paires message/chiffré connues, et les a et b sélectionnés précédemment.

Exercice 8

Nous allons maintenant procéder à l'attaque à proprement parler. Mettons les éléments bouts-à-bouts.

1. Pour chaque k_0 candidat, décrivez comment retrouver le k_1 correspondant.

Nous avons maintenant une fonction qui nous prédit le candidat k_1 étant donnée l'hypothèse k_0.

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

Exercice 9

Branchez vos sous routines pour cryptanalyser le schéma $\mathcal{S}$ efficacement.

BONUS ;)

Certaines de vos réponses sont discutables. Lesquelles selon vous ?

Attardons-nous sur la question 2 de l'exo 3.

1. Rappel : Pourquoi n'est-il pas possible de cryptanalyser sur la première sous-clé seulement ?
2. Qu'avons-nous mis en œuvre pour contrecarre cela ?

En dédiant un peu de mémoire à la cryptanalyse, il est possible de significativement/drastiquement réduire les efforts dédiés à la cryptanalyse.

3. Une idée de comment ?

La réponse tient dans un compromis mémoire / temps CPU. Soit m/c un couple clair/chiffré. Soit k'_0 la clé supposée au premier tour, et k''_0 la clé en découlant. Dans ce bonus, nous ne considérerons uniquement que le couple $\bar{m}/\bar{c}$, obtenu via la clé $\bar{k}$.

Partie gauche :

4. Faire une hypothèse sur $\bar{k_0}$
5. Sauvegarder tous les `encrypt(`$\bar{m}, \bar{k_0}$`)` dans $\bar{L}$.

Partie droite :

6. Faire une hypothèse sur $\bar{k_1}$
7. Sauvegarder tous les `decrypt(`$\bar{c}, \bar{k_1}$`)` dans $\bar{R}$.

Concordance :

8. S'il existe i et j tels que $\bar{L}[i] == \bar{R}[j]$, retourner "done", else "fail".

Analyse des résultats :

9. Avez-vous des concordances ?
10. Qu'en déduisez-vous ?
