Introduction à la Cryptographie

TD 3 : Chiffrements Symétriques Modernes

Exercice 1. *(Chiffrement par flot avec des LFSR)*

Rappel mathématique. Le corps fini à deux éléments $\mathbb{F}_2$ est l'ensemble $\{0, 1\}$ muni des opérations $+$ et $\times$ correspondant respectivement à l'addition et la multiplication modulo 2.

Registre à Décalage Rétroactif Linéaire. Un LFSR (*Linear Feedback Shift Register*) génère une suite de bits $(b_t)_{t\geq 0}$ à partir d'un *état initial* et d'un *polynôme de rétroaction*. Plus précisément, un LFSR est constitué d'un *registre* à n bits initialisé à l'instant $t = 0$ avec un *état initial*. À chaque instant $t \geq 0$, le registre est décalé vers la droite (*right shift*) ; le bit le plus à droite du registre est alors supprimé du registre et constitue le bit b_t de la suite générée. L'emplacement le plus à gauche dans le registre est alors laissé vacant ; on y place un nouveau bit calculé à partir de l'état courant du registre et du *polynôme de rétroaction*.

Calcul du bit de rétroaction. Soit le polynôme de rétroaction suivant :

$$P(X) = 1 + c_1X + c_2X^2 + \cdots + c_nX^n \in \mathbb{F}_2[X]_{\leq n}$$

Le registre du LFSR à l'instant t est $(r_{n-1}, r_{n-2}, \cdots, r_0)$. Le bit b_t de la suite générée par le LFSR est alors r_0 (le bit le plus à droite dans le registre). À l'instant $t + 1$, le registre vaut :

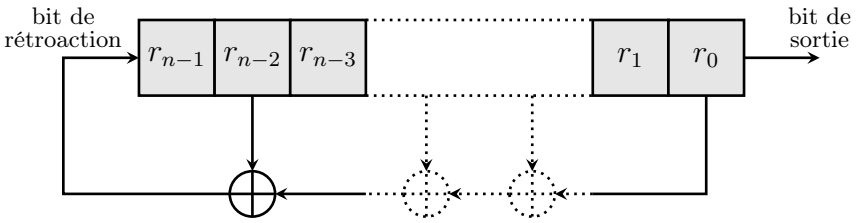
$$\left(\sum_{i=1}^{n} c_i r_{n-i}, r_{n-1}, r_{n-2}, \cdots, r_1\right)$$

Premier exemple. Soit le LFSR $\mathcal{L}_1$ de taille 8 et de polynôme de rétroaction :

$$P(X) := 1 + X + X^3 + X^4 + X^7 + X^8$$

1. Calculer les 16 premiers bits de la suite générée par le LFSR lorsque le registre est initialisé à $0x5A$ (écriture hexadécimale).
2. De façon générale, que valent les 8 premiers bits de la suite?

Représentation d'un LFSR avec un circuit logique. Les LFSR ont l'avantage d'être très faciles à implémenter et de ne demander que très peu de ressources de calcul. Pour le remarquer, on représente un LFSR de taille n à l'aide d'un circuit de la forme suivante :



À chaque instant t, le registre est décalé d'un cran vers la droite en "poussant" chaque bit dans les branches du circuit. Le bit en sortie est alors le bit b_t de la suite et le bit de rétroaction (calculé en parcourant le circuit logique) est inséré dans la case la plus à gauche du registre laissée libre après le décalage.

3. Dessiner le circuit correspondant à $\mathcal{L}_1$ à l'instant $t = 0$; on initialisera le registre avec le vecteur binaire $0x5A$.

4. Que vaut le registre à l'instant $t = 1$? $t = 4$?

5. Implémenter la fonction `LFSR` qui prend en entrée :

   – `n` : la taille du registre (et donc du polynôme de rétroaction) ;
   – `reg` : le registre courant d'un LFSR sous forme d'un entier de 64 bits (seul les $n \leq 64$ bits de poids faible seront utilisés) ;
   – `retro` : le polynôme de rétroaction sous forme d'un entier de 64 bits (les bits de poids faible seront $c_1 c_2 \cdots c_n$ et les autres bits seront 0).

   La fonction doit retourner le bit de sortie et mettre à jour le registre `reg`. Soyez astucieux dans votre implémentation ; notamment, utilisez les opérations binaires du langage que vous aurez choisi (en C ou Python, le XOR bit à bit de deux entiers peut être réalisé avec l'opérateur $\hat{}$ et le décalage des bits d'un entier peut être réalisé avec l'opérateur >>).

6. Tester votre implémentation en générant la suite calculée dans la question 1.

Chiffrement avec des LFSR. Il est proposé d'utiliser un LFSR pour générer une suite chiffrante. Le message chiffré est alors obtenu en effectuant le XOR bit à bit du message clair avec la suite chiffrante.

7. Quelle est la clé de chiffrement du crypto-système décrit juste au dessus?

8. Comment le destinataire légitime d'un message chiffré peut-il déchiffrer le message qu'il reçoit?

9. En réutilisant votre code de la question 5, implémenter les fonctions `encryptLFSR` et `decryptLFSR` qui chiffrent et déchiffrent des messages avec des LFSR.

Périodicité des LFSR. Le registre d'un LFSR ne peut prendre qu'un nombre fini d'état. Ainsi, au bout d'un certain temps, le registre sera nécessairement dans un état qu'il aura déjà eu dans le passé. Or la suite générée par un LFSR à partir d'un état donné du registre est déterminée. Ainsi, le LFSR reproduira une suite de bits déjà produite avant...

10. Que se passe-t-il lorsque le registre est initialisé avec le vecteur tout à 0? En déduire la période minimale d'un LFSR.

11. Combien de valeurs différentes peut prendre le registre d'un LFSR de taille n? En déduire la période maximale d'un LFSR. Avec une clé de 64 bits, quelle quantité de données maximum peut-on chiffrer avant que la suite chiffrante ne se répète?

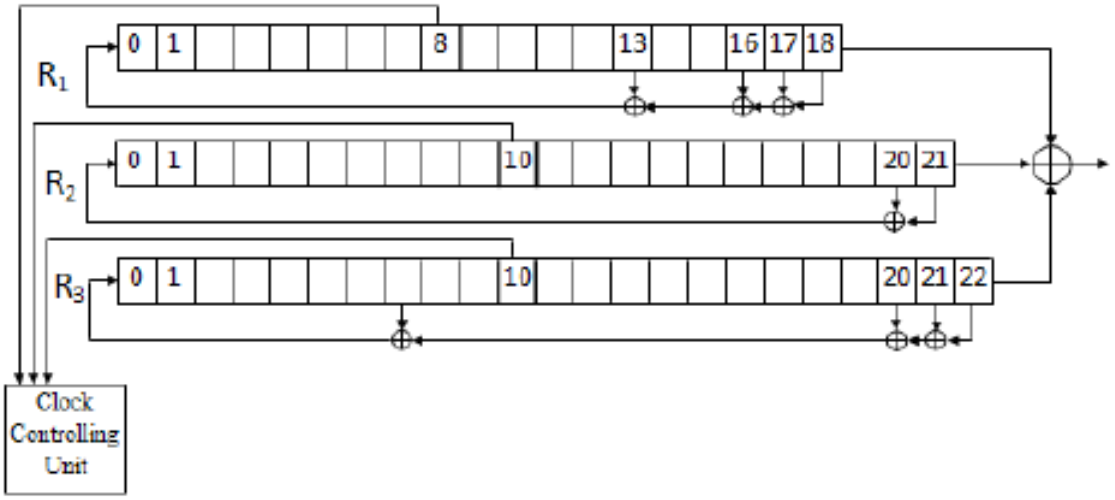
12. Soit une suite chiffrante produite avec un LFSR de taille 32 ayant comme polynôme de rétroaction :
$$P(X) := 1 + X^{32}$$
Lorsque l'état initial du registre est non nul, quelle est la période de la suite chiffrante. Justifier qu'il est important de choisir judicieusement le polynôme de rétroaction du LFSR.

Remarque. Il existe un critère mathématique sur le polynôme de rétroaction d'un LFSR pour garantir que celui-ci génère une suite de période maximale.

Attaque sur les chiffrements par LFSR. Soit un LFSR de taille n. L'algorithme de Berlekamp-Massey permet de retrouver le polynôme de rétroaction à partir de $2n$ bits de la suite générée par ce LFSR.

13. Montrer qu'un chiffrement par LFSR tel que décrit plus avant dans le TD est cassé dans le modèle KPA, CPA et CCA.

Le chiffrement A5/1 (GSM).



Le chiffrement A5/1 (utilisé dans la norme GSM) combine la sortie de 3 LFSR notés $\mathcal{L}_1$, $\mathcal{L}_2$ et $\mathcal{L}_3$ dont les polynômes de rétroaction sont respectivement :

$$\begin{aligned} P_1 &= 1 + X^{14} + X^{17} + X^{18} + X^{19} \\ P_2 &= 1 + X^{21} + X^{22} \\ P_3 &= 1 + X^{8} + X^{21} + X^{22} + X^{23} \end{aligned}$$

Les trois LFSR ne fonctionnent pas de manière synchrone : un LFSR n'est incrémenté que si son bit d'horloge est égal au bit majoritaire parmi les 3 bits d'horloge ; à chaque cycle les 3 LFSR ou seulement 2 sont donc incrémentés. Les bits d'horloge h_1, h_2 et h_3 sont respectivement le $8^{\text{ème}}$ bit du registre de $\mathcal{L}_1$, le $10^{\text{ème}}$ bit du registre de $\mathcal{L}_2$ et le $10^{\text{ème}}$ bit du registre de $\mathcal{L}_3$. On remarque alors que :

– si $(h_1, h_2, h_3) = (..., ..., ...)$ ou $(..., ..., ...)$ alors tous les LFSR sont incrémentés ;

– si $(h_1, h_2, h_3) = (..., ..., ...)$ ou $(..., ..., ...)$ alors seuls les LFSR $\mathcal{L}_1$ et $\mathcal{L}_2$ sont incrémentés ;

– si $(h_1, h_2, h_3) = (..., ..., ...)$ ou $(..., ..., ...)$ alors seuls les LFSR $\mathcal{L}_1$ et $\mathcal{L}_3$ sont incrémentés ;

– si $(h_1, h_2, h_3) = (..., ..., ...)$ ou $(..., ..., ...)$ alors seuls les LFSR $\mathcal{L}_2$ et $\mathcal{L}_3$ sont incrémentés.

Finalement, les 3 bits de sortie (les derniers bits de chaque LFSR) sont xorés pour produire un bit de la suite chiffrante.

La clé de chiffrement/déchiffrement est simplement la concaténation des trois registres initiaux. Celle-ci fait donc 64 bits. En réalité, seuls 54 bits sont utilisés dans le GSM (les 10 bits restant sont initialisés à 0).

13. En réutilisant votre code de la question 5, implémenter le chiffrement A5/1.

Remarque. Il existe un critère mathématique sur le polynôme de rétroaction d'un LFSR pour garantir que celui-ci génère une suite de période maximale.

Exercice 2. ***(Chiffrement par bloc : DES, AES).***

Nous allons utiliser la librairie `pycrypto` en Python qui implémente notamment les chiffrements symétriques classiques tels que DES et AES. Pour pouvoir l'utiliser, il faut tout d'abord l'installer. Pour cela, vous pouvez utiliser la commande :

```
pip3 install pycryptodome
```

À titre d'exemple, voici un code qui permet de créer un chiffreur DES et de chiffrer/déchiffrer un message :

```
from Crypto.Cipher import DES

def pad(text):
    n = 8 - len(text) % 8
    if n == 8:
        return text
    return text + (b' ' * n)

key = b'password'
des = DES.new(key, DES.MODE_ECB)

plaintext = pad(b'Hello world !!!')
ciphertext = des.encrypt(plaintext)

print([hex(x) for x in ciphertext])
print(des.decrypt(ciphertext))
```

1. Implémenter un générateur de clé qui retourne une clé tirée uniformément dans l'ensemble des chaines de 64 bits. La clé devra être au format `byte array` (type `bytes` en Python).

La fonction suivante calcule le xor bit à bit entre deux `byte array` :

```
def xor_bytes(s1, s2):
    if len(s1) != len(s2):
        exit("xor_bytes error : len(s1) != len(s2) !")
    return bytes([a ^ b for (a,b) in zip(s1, s2) ])
```

2. Choisir une clé secrète $\mathbf{k}$ et un message clair $\mathbf{m}$ de 64 bits chacun. Tester le chiffrement et déchiffrement DES en mode ECB sur le message $\mathbf{m}$ avec la clé $\mathbf{k}$ puis avec la clé $\mathbf{k}' := \mathbf{k} \oplus 0x0101010101010101$. Expliquer le résultat obtenu.

3. Implémenter un générateur de clé DES qui retourne une clé de 64 bits respectant la parité sur chaque octet.

4. On note $\overline{\mathbf{x}}$ le vecteur binaire complémentaire de $\mathbf{x}$. Autrement dit,

$$\overline{\mathbf{x}} := \mathbf{x} \oplus 0b111\cdots1$$

   Appliquer le chiffrement DES en mode ECB sur le message $\overline{\mathbf{m}}$ avec la clé $\overline{\mathbf{k}}$. Comparer le résultat obtenu avec le cryptogramme obtenu à la question 2. Le résultat vous parait-il normal? Est-ce une faille de sécurité du chiffrement DES?

5. Une clé secrète $\mathbf{k}$ est dite *faible* si elle génère des clés partielles $\mathbf{k}_i$ identiques. Il y a exactement 4 clés faibles pour DES. Proposer un algorithme qui permet de les trouver.

   Aide. On remarque que les clés partielles $\mathbf{k}_i$ sont toutes égales dès lors que les permutations PC-2 prennent en entrée le même vecteur $\mathbf{x}_1||\mathbf{x}_2$ où $\mathbf{x}_1$ et $\mathbf{x}_2$ sont soit le vecteur tout à 0 soit le vecteur tout à 1.

6. Choisir deux messages $\mathbf{m}_1$ et $\mathbf{m}_2$ de 64 bits tels que $\mathbf{m}_1$ et $\mathbf{m}_2$ diffèrent seulement sur un bit. Appliquer le chiffrement DES en mode ECB sur $\mathbf{m}_1$ et $\mathbf{m}_2$ avec une même clé. Comparer les résultats.

7. Appliquer le chiffrement DES en mode ECB sur un message $\mathbf{m}$ de 64 bits ; on note $\mathbf{c}$ le chiffré obtenu.

   Déchiffrer un message $\mathbf{c}'$ qui ne diffère de $\mathbf{c}$ que d'un seul bit. Comparer le message déchiffré avec le message $\mathbf{m}$. Répéter l'opération plusieurs fois en changeant le bit modifié. Que peut-on en conclure?

8. Résumer les différentes faiblesses du chiffrement DES mises en lumière depuis le début de l'exercice.

double-DES.

On voudrait étudier si le chiffrement double-DES (2-DES) améliore le niveau de sécurité par rapport au simple DES.

Le protocole 2-DES utilise deux clés secrètes, $\mathbf{k}_1$ et $\mathbf{k}_2$. Pour chiffrer un message $\mathbf{m}$, Alice calcule :

$$\mathbf{c} = Enc_{\mathbf{k}_2}(Enc_{\mathbf{k}_1}(\mathbf{m}))$$

Bob, à son tour, déchffre $\mathbf{m}$ par l'opération suivante :

$$\mathbf{m} = Dec_{\mathbf{k}_1}(Dec_{\mathbf{k}_2}(\mathbf{c}))$$

9. Chiffrer deux messages $\mathbf{m}_1$ et $\mathbf{m}_2$ par 2-DES pour obtenir deux messages chiffrés $\mathbf{c}_1$ et $\mathbf{c}_2$.

10. On a donc deux couples de messages : $(\mathbf{m}_1, \mathbf{c}_1)$ et $(\mathbf{m}_2, \mathbf{c}_2)$.

    Démontrer sur un exemple que

    $$Enc_{\mathbf{k}_1}(\mathbf{m}_i) = Dec_{\mathbf{k}_2}(\mathbf{c}_i)$$

    pour $i \in \{1, 2\}$.

11. Justifier que le nombre moyen de clés $(\mathbf{k}_1, \mathbf{k}_2)$ tels que $Enc_{\mathbf{k}_1}(\mathbf{m}_1) = Dec_{\mathbf{k}_2}(\mathbf{c}_1)$ est $2^{112-64} = 2^48$.

12. Justifier que le nombre moyen de clés $(\mathbf{k}_1, \mathbf{k}_2)$ tels que $Enc_{\mathbf{k}_1}(\mathbf{m}_1) = Dec_{\mathbf{k}_2}(\mathbf{c}_1)$ ET $Enc_{\mathbf{k}_1}(\mathbf{m}_2) = Dec_{\mathbf{k}_2}(\mathbf{c}_2)$ est 1.

13. Proposer une attaque sur le double-DES.

AES et modes de chiffrement.

14. Adapter les questions 6 et 7 au chiffrement AES-128 en mode ECB. Les faiblesses de DES vous semblent-elles corrigées avec AES?

15. Implémenter une fonction de chiffrement qui prend en entrée :

    - `key` : la clé de chiffrement ;
    - `image` : un fichier au format bitmap (`.bmp`) ;
    - `algo` : "DES" ou "AES" ;
    - `mode` : "MODE_ECB", "MODE_CBC" ou "MODE_CTR" ;
    - `iv` : un éventuel vecteur d'initialisation (nécessaire selon le mode de chiffrement considéré).

    et qui retourne un fichier au format bitmap qui représente le contenu chiffré de `image`. Attention, il est demandé de ne chiffrer que le contenu de l'image ; on ne modifiera donc pas l'entête du fichier bitmap.

16. Comparer les différents modes de chiffrement de DES et AES sur le fichier `image.bmp` (ne pas chiffrer l'entête qui est de 74 octets pour ce fichier). Quel mode vous semble le plus pertinent? Justifier.