Principe et Algorithme Cryptographiques

TP # 3 : LFSR et block-cipher

Université de Lille-1 — FIL

7 Mars 2013

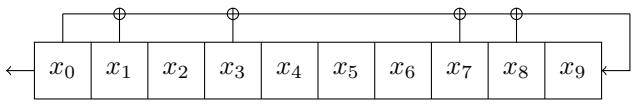
L'objectif de ce TP est de programmer des choses qui tournent autours des LFSR. Pour cela on utilisera Sage, un logiciel libre dédié aux mathématiques. Sur les machines du FIL, on peut lancer Sage avec la commande :

```
/home/share/apps/sage-5.4.1/sage
```

Vous pouvez récupérer Sage pour l'utiliser chez vous en allant sur `http://www.sagemath.org`. Sage utilise massivement le langage de programmation `Python`. Pour programmer en Sage, on programme en `Python`. La différence, c'est que dans Sage on a accès à une vaste palette de fonctions mathématiques. Vous êtes invités à jeter un coup d'oeil à la documentation de Sage, ainsi qu'à celle de Python. Dans Sage, la commande "`tutorial()`" est sûrement utile... Vous pouvez aussi vous référer au livre *Calculs mathématiques avec Sage*, qui est librement téléchargeable à `http://sagebook.gforge.inria.fr/`. De même le tutoriel `Python` (`http://docs.python.org/2/tutorial/`) vous sera sûrement utile.

Linear Feedback Shift Registers

On rappelle qu'un LFSR est constitué par un registre de n bits :



À chaque étape, le bit d'indice zéro "sort" du registre, et devient le prochain bit de la séquence générée par le LFSR. Le registre est alors décalé d'un cran vers la gauche, puis le nouveau bit fabriqué par la boucle de rétroaction entre à droite. La relation de recurrence de ce registre est :

$$P = X^{10} + X^9 + X^7 + X^3 + X^2 + 1$$

Par exemple, avec l'état initial $(0, 1, 0, 1, 0, 0, 0, 1, 1, 1)$, les premiers bits de la séquence sont :

$$1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, \ldots$$

En fait la séquence obéit à la relation de recurrence :

$$x_{n+10} = x_n + x_{n+1} + x_{n+3} + x_{n+7} + x_{n+8}$$

Pour pouvoir travailler tranquillement avec des LFSR, il nous faut manipuler des bits, donc il faut se placer dans l'anneau des entiers modulo deux. On crée cette structure de la façon suivante :

```
sage: A = GF(2)
```

Par défaut dans Sage, les nombres sont considérés comme des entiers

```
sage: 1+1
2
```

Mais les éléments de A se comportent comme prévu :

```
sage: A(1) + A(1)
0
```

Le registre d'un LFSR est simplement un tableau d'éléments de A :

```
sage: seed = [ A(1), A(0), A(1), A(1), A(1) ]
```

Comme c'est un peu lourd, on peut aussi écrire :

```
sage: seed = map(A, [0,0,1,0,1])
```

Le polynôme de rétroaction d'un LFSR est également décrit par une liste d'entiers modulo deux, de la même taille. Par exemple, le polynôme de tout à l'heure est décrit par la liste :

```
sage: seed = map(A, [1,1,0,1,0,0,0,1,1,0])
```

Exercice 1 : Fonctions de base.

1. Programmez une fonction qui prend en argument un état initial, un polynôme de rétroaction, un entier k. Cette fonction doit renvoyer les k premiers bits de la séquence générée par le LFSR.
2. Sage contient en fait déjà une fonction `lfsr_sequence`, qui est censé faire la même chose que la votre. Servez-vous en pour vérifier que vous ne vous êtes pas trompés.

Si l'état du registre s'écrit $\mathbf{x} = (x_{n-1}, \ldots, x_0)$, et que le polynôme s'écrit

$$P = a_n X^n + a_{n-1} X^{n-1} + a_{n-2} X^{n-2} + \cdots + a_1 X + 1,$$

alors après une "étape" du LFSR, le nouvel état s'écrit :

$$\mathbf{x}' = \left(\sum_{i=0}^{n-1} x_i \cdot a_{n-i},\ x_{n-1},\ x_{n-2}, \ldots, x_1 \right)$$

En fait, si on regarde le registre comme un vecteur, alors il existe une matrice M telle que $\mathbf{x}' = \mathbf{x} \times M$.

3. Déterminez cette matrice (elle a une structure simple) et écrivez une fonction de Sage qui la renvoie. Vous pouvez créer la matrice avec la fonction :

```
sage: M = zero_matrix(A, 10, 10)
sage: M
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
```

   Et vous pouvez ensuite la modifier avec :

```
M[1,4] = 1
```

4. Écrivez une fonction qui prend l'état initial d'un LFSR, le polynôme de rétroaction, un entier k, et qui renvoie l'état interne après que les k premiers bits de la sortie aient été générés. Attention, votre fonction doit avoir une complexité de l'ordre de $\mathcal{O}(\log k)$. Mais heureusement, pour calculer M^i, Sage utilise un algorithme d'exponentiation rapide !
5. Écrivez une fonction qui prend en argument l'état initial d'un LFSR, le polynôme de rétroaction, ainsi que deux entiers k et ℓ. Si $x_0, x_1, \ldots$ désigne la suite générée par le LFSR, votre fonction doit renvoyer $x_k, x_{k+1}, \ldots, x_{k+\ell-1}$. Elle doit être de complexité $\mathcal{O}(\ell + \log k)$.

La suite produite par un LFSR de taille n obéit à la relation de récurrence :

$$x_{n+k} = \sum_{i=0}^{n-1} a_{n-i} \cdot x_{k+i}$$

où les a_i sont les coefficients du polynôme de rétroaction. Il découle de ceci que si on connaît les valeurs des x_i (c'est-à-dire si on connaît la séquence générée par un LFSR), alors on peut écrire des équations linéaires en les a_i (...et les résoudre).

6. Supposons qu'on possède le début de la séquence d'un LFSR, mais qu'on ne connaisse pas le polynôme de rétroaction. De combien de bits de la séquence a-t-on besoin pour retrouver le polynôme de rétroaction ?
7. Écrivez une fonction qui prend en argument un préfixe suffisament long de la séquence générée par un LSFR, et qui renvoie une matrice M et un vecteur $\mathbf{v}$ qui satisfont la relation $M \times \mathbf{a} = \mathbf{v}$, où le vecteur $\mathbf{a}$ est formé des coefficients du polynôme de rétroaction.
8. Écrivez une fonction qui, à partir d'un préfixe suffisament long de la la séquence générée par un LSFR, retrouve le polynôme de rétroaction. Note à l'attention des petits malins : il est interdit d'utiliser les fonctions toutes faites `lfsr_connection_polynomial` et `berlekamp_massey` qui sont dans Sage.

Applications Crypto

Exercice 2 : LFSR, Bière et Crypto. Charles, Éric et Alexandre boivent des bières après le travail. ~~Complètement bourrés~~ Nettement éméchés, ils entreprennent néanmoins de concevoir un système de chiffrement par bloc à base de LFSR. Ils veulent chiffrer des blocs de 80 bits avec des clefs de 80 bits.

Charles, qui supporte très mal l'alcool, et qui commence à avoir du mal à aligner deux phrases d'affilée, propose le système suivant :

- On met bout-à-bout la clef et le message (ce qui fait donc 160 bits), et on se sert de ça comme état initial d'un LFSR (de taille 160). Plus précisément, les bits 0–79 du LFSR doivent contenir le message, et les bits 80–159 la clef. Le polynôme de rétroaction du LFSR est $X^{160}+X^5+X^3+X^2+1$ (il est irreductible).
- On génère les 160 premiers bits de la suite générée par le LFSR, et on les jette à la poubelle (en effet, ils contiennt l'état interne initial du LFSR !).
- On génère les 80 bits suivants de la suite, qui forment le message chiffré.

1. Programmez le système de chiffrement.
2. Que pensez-vous de ce système ? Est-il utilisable ? Si oui, justifiez. Sinon, donnez un exemple qui démontre de façon claire qu'il ne faut pas l'utiliser.
3. Ce système n'est pas sûr. Programmez une attaque à clair connu qui retrouve la clef, ou tout du moins des informations sur la clef.

Alexandre, qui tangue pas mal lui aussi, propose un autre système. Puisque le message a 80 bits (qu'on appelle $M_0, \ldots, M_{79}$), alors on se sert du message pour définir le polynôme :

$$P = \sum_{i=0}^{n-1} M_i \times X^i$$

La proposition d'Alexandre est la suivante :

- On se sert de la clef comme état initial d'un LFSR de 80 bits, et on se sert du message à chiffrer comme polynôme de rétroaction.
- On jette à la poubelle les 80 premiers bits de la suite chiffrée
- Les 80 bits suivants de la suite forment le message chiffré.

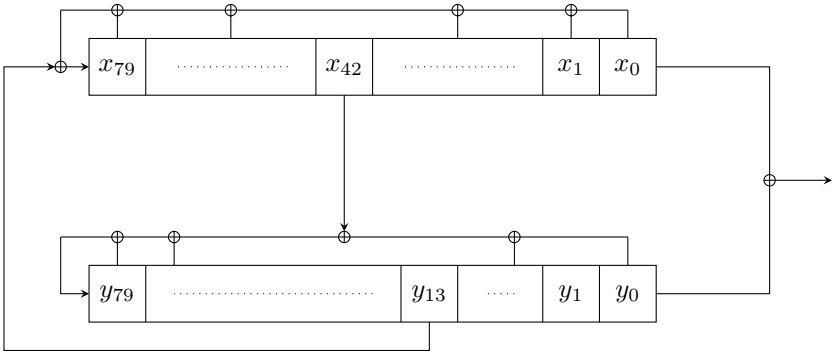
4. Répondez aux mêmes questions que précédement.

Éric, qui lui est pris de fou-rires incontrolables, propose de faire l'inverse : utiliser la clef comme polynôme de rétroaction, et le message comme état initial.

5. Répondez aux mêmes questions que précédement.

(⋆⋆⋆) 6. Qu'est-ce qui change dans les deux questions précédentes si on jette à la poubelle les 1000 premiers bits de la suite générée par le LFSR ?

Exercice 3 : Consulting. Vous travaillez dans une entreprise de sécurité informatique. Votre supérieur hiérarchique, qui a suivi un cours de LFSR, vous amène sa dernière création :



Dans ce dispositif génial, on met la clef dans $x_0, \ldots, x_{79}$, le message à chiffrer dans $y_0, \ldots, y_{79}$. On jette à la poubelle les 1000 bits de la séquence générée par ce machin, et les 80 suivants forment le chiffré. Le polynôme de rétroaction du 1er LFSR est $X^{80} + X^{11} + X^3 + X^2 + 1$ et celui du second est $X^{80} + X^9 + X^4 + X^2 + 1$. Mais à ça viennent s'ajouter les bit perturbateur en provenance de *l'autre* LFSR...

Que dites-vous à votre chef ?