TD2 - LFSR et Chiffrement par Flot

Responsable : M. Bombar

1 Introduction aux LFSR

(Q1) Écrivez une fonction `LFSR_step(P, state)` qui simule l'étape d'un LFSR. Elle prend en entrée un polynôme de rétroaction $P \in \mathbb{F}_q[X]$ ainsi que l'état S_t au temps t, et doit ressortir l'état au temps $t+1$, ainsi que la sortie à l'instant t. On ne suppose pas que le corps de base est $\mathbb{F}_2$.

(Q2) Écrivez une fonction `LFSR(P, state, N)` qui prend un polynôme de rétroaction, l'état initial S_0 ainsi qu'un entier N, et retourne les N premiers éléments de sortie du LFSR de rétroaction P, initialisé par S_0.

(Q3) Vérifiez que le LFSR de longueur 5 et de polynôme de rétroaction $P(X) \stackrel{\text{def}}{=} 1 + X^4 + X^5 \in \mathbb{F}_2[X]$, initialisé par $S_0 \stackrel{\text{def}}{=} (0,0,1,0,1)$ retourne une suite débutant par $0,0,1,0,1,0,1,\ldots$. Quelle est sa période ? La suite est-elle maximale ? Le polynôme P est-il irréductible ? Primitif ?

Remarque 1. *Pour la question précédente, vous pouvez écrire une fonction rapide qui calcule la période. Un algorithme naïf résoud aisément le problème en temps quadratique en la longueur de la liste. Notons qu'il existe un algorithme linéaire en pire cas (dérivé de l'algorithme de Knuth-Moris-Pratt pour la recherche de sous-chaîne de caractères), mais ce n'est pas nécessaire ici.*

On peut former l'anneau des séries formelles en une indéterminée avec Sage en suivant l'exemple suivant :

```
sage: F2 = GF(2)
sage: PS.<X> = PowerSeriesRing(F2)
```

(Q4) Écrivez une fonction `LFSR_minpoly(s, P)` qui prend en entrée une suite produite par un LFSR, ainsi que son polynôme de rétroaction P et retourne son polynôme de rétroaction minimal P_0.

(Q5) Vérifiez que P est bien le polynôme minimal du LFSR de la question (Q3).

(Q6) Soit L le LFSR de polynôme de rétroaction $P_1 = 1 + X + X^8$ et d'état initial $(0, 0, 1, 0, 1, 1, 1, 1)$. Déterminez son polynôme de rétroaction minimal.

Remarque 2. *Par cette méthode, on doit déjà connaître un polynôme de rétroaction.*

2 Algorithme de Berlekamp-Massey

Les LFSR semblent de prime abord fournir une solution clé en main pour générer des chaînes de bits pseudoaléatoires :

— Ils sont extrêmement rapides, surtout en hardware.

— Il est très facile de produire des suites aux périodes arbitrairement longues.

— Les suites maximales ont de bonnes propriétés statistiques.

En revanche, le but de cet exercice est de voir que ceci ne suffit pas pour fournir des chiffrements par flot sécurisés.

(Q7) Soit $\mathbf{s} \stackrel{\text{def}}{=} (s_n)_{n\in\mathbb{N}} \in \mathbb{F}_q^{\mathbb{N}}$ une suite ultimement périodique. On note $\Lambda(\mathbf{s}) \geq 1$ sa complexité linéaire. Montrez que $2\Lambda(\mathbf{s})$ termes consécutifs de la suite caractérisent entièrement $\mathbf{s}$. Plus précisément, montrez que si on connaît $\Lambda(s)$, ainsi que $2\Lambda(s)$ termes **consécutifs** de $\mathbf{s}$, alors il est possible de calculer son polynôme de rétroaction minimal P.
Indication : Écrire les coefficients de P comme les inconnues d'un système linéaire.

(Q8) On suppose que l'on connaît une borne supérieure $\Lambda(s) \leq \ell$ sur la complexité linéaire de $\mathbf{s}$. Déterminez un algorithme qui prend en entrée $\mathbf{s}$ et retourne $\Lambda(\mathbf{s})$ ainsi que le polynôme de rétroaction minimal en temps $O(\ell^4)$. Quelle est sa complexité en mémoire ?

La Question (Q8) nous donne déjà un algorithme **polynomial** pour caractériser la suite $\mathbf{s}$ en n'observant qu'un nombre restreint de coefficients. En particulier, même si $\mathbf{s}$ a d'excellentes propriétés statistiques, elle n'est absolument pas **imprédictible**. Cependant, la

complexité $O(\ell^4)$ peut parfois être assez grosse pour monter des attaques en pratique, surtout lorsque les LFSR sont combinés à d'autres opérations, ou lorsque la borne supérieure ℓ n'est pas connue. Cependant, on va voir qu'il est possible de faire beaucoup mieux étant donnée la **structure** des systèmes linéaires mis en jeu.

Pour cela, on va décrire, et implémenter, un algorithme dû à James Massey [Mas69], qui a montré comment une procédure proposée deux ans plus tôt par Elwyn Berlekamp [Ber68] afin de décoder une certaine famille de codes correcteurs d'erreurs appelés codes BCH, pouvait être utilisée pour déterminer le LFSR minimal qui génère une suite donnée. Étant donnée une suite $\mathbf{s} = (\mathbf{s}_0, \ldots, \mathbf{s}_{n-1})$ de longueur n, ou plus précisément la suite prolongée par périodicité, l'algorithme de Berlekamp-Massey va faire n itérations. À l'itération t, l'algorithme de Berlekamp-Massey détermine un LFSR de longueur minimale qui produit les t premiers symboles de $\mathbf{s}$.

Algorithm 1: Algorithme de Berlekamp-Massey

Input: $\mathbf{s}^{(n)} = (s_0, \ldots, s_{n-1})$ une suite de n éléments de $\mathbb{F}_q$.
Output: $\Lambda(\mathbf{s}^{(n)})$ la complexité linéaire de $\mathbf{s}^{(n)}$ et P, le polynôme de rétroaction de degré minimal qui engendre $\mathbf{s}^{(n)}$.

```
/* Initialisation */
P(X) ← 1
Q(X) ← 1
Λ ← 0
m ← −1
d′ ← 1
/* Algorithme */
for t ∈ {0, ..., n − 1} do
    d ← s_t + Σ_{i=1}^{Λ} p_i s_{t−i}
    if d ≠ 0 then
        T(X) ← P(X)                          ▷ Polynôme temporaire
        P(X) ← P(X) − d(d′)^{−1} Q(X) X^{t−m}
        if 2Λ ≤ t then
            Λ ← t + 1 − Λ
            m ← t
            Q(X) ← T(X)
            d′ ← d
return Λ et P(X)
```

(Q9) Décrire les étapes de l'algorithme de Berlekamp-Massey appliqué à la suite binaire de longueur 7 et définie par

$$\mathbf{s}^{(7)} \stackrel{\text{def}}{=} (0,1,1,1,1,0,0).$$

(Q10) Implémentez l'algorithme de Berlekamp-Massey avec Sage.

(Q11) Quelle est sa complexité ?

(Q12) Montrez que l'algorithme est correct, c'est-à-dire qu'à la fin de l'algorithme on a bien $\Lambda = \Lambda(s^{(n)})$ et que P est bien le polynôme de rétroaction minimal de $\mathbf{s}^{(n)}$.

3 Challenge

Remarque 3. *Ce challenge est inspiré par le France Cybersecurity Challenge organisé par l'ANSSI.*

Lors d'un test d'intrusion, vous découvrez qu'une équipe de développeurs a mis en place leur propre système de chiffrement maison pour « protéger » des fichiers sensibles.

Ils vous expliquent que leur chiffrement repose sur un LFSR original, dont le polynôme de rétroaction vous est inconnu, mais dont l'état interne est de 64 bits, et est changé tous les jours.

Pour vous prouver que leur solution est inviolable, ils ont chiffré une image PNG contenant un flag :

— Le fichier chiffré `FunWithFlags.png.enc`

— Le code du chiffrement `challenge.sage`

Ils vous mettent au défi d'extraire le flag contenu dans l'image originale.

Remarque 4.
— *Pour exécuter `challenge.sage` vous aurez besoin des fonctions de base implémentées au début du TD.*

— *Vous aurez aussi besoin d'importer le module `bitarray`. Normalement, il est déjà installé sur les machines du CREMI, mais sinon `sage -pip install bitarray` devrait faire l'affaire.*

— *Vous pouvez également installer le module `tqdm`. Il permet de faire un rendu visuel de parcours d'une boucle.*

— *Tel quel, le chiffrement est relativement lent. C'est parce que je manipule des*

objets Sagemath comme des polynômes et des éléments de corps finis. Normalement, il faudrait faire directement des opérations bit à bit, mais ça rend le fonctionnement moins lisible.

(Q13) Retrouvez le flag. Vous fournirez un fichier `NOM_solver.sage` ainsi que `NOM_flag.txt` contenant uniquement le flag, et `NOM_FunWithFlags.png` contenant l'image originale.

(Q14) Bonus : réimplémentez l'algo de chiffrement avec des véritables opérations bit à bit pour obtenir un chiffrement efficace.

Références

[Ber68] Elwyn R. Berlekamp. « Factoring Polynomials over finite fields ». In : **Algebraic Coding Theory**. Sous la dir. d'E. R. Berlekamp. McGraw-Hill, 1968. Chap. 6.

[Mas69] James L. Massey. « Shift-Register Synthesis and BCH Decoding ». In : **IEEE Trans. Inform. Theory** 15.1 (1969), p. 122-127.