UNIVERSITÉ DE LIMOGES
ÉCOLE DOCTORALE Sciences et Ingénierie pour l'Information
Faculté des Sciences et Techniques
Département de Mathématiques et Informatique – Laboratoire XLIM

Thèse n^o 68 - 2010

Thèse

pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ DE LIMOGES
Discipline : Mathématiques et applications

soutenue et présentée par
Benjamin POUSSE

le jeudi 2 décembre 2010, à 14h

Design et cryptanalyse de chiffrements à flot

Jury

Président		
	Louis Goubin	Université de Versailles
Directeurs		
	Thierry Berger	Université de Limoges
	Marine Minier	INSA Lyon
Rapporteurs		
	Anne Canteaut	INRIA Paris-Rocquencourt
	Henri Gilbert	Orange Labs
Examinateur		
	François Arnault	Université de Limoges

Remerciements

J'ai (enfin !) fini ma thèse. Avant la longue énumération des personnes ayant contribué à la réalisation de ces travaux, je souhaite faire le bilan de ma thèse. Cette thèse, c'est :

- d'un point de vue universitaire, quatre publications dans des conférences internationales ;
- d'un point de vue linguistique, l'amélioration (sensible) de mon anglais ;
- d'un point de vue humain, la rencontre de personnes devenues des amis, et la rencontre de Julie dans la salle des séminaires !

Mais cette thèse, c'est surtout, d'un point de vue touristique, cinq grands voyages en Australie, en Tunisie, au Canada, en Inde et en Malaisie. Soit environ 90 000 km parcourus. Ainsi, comme le dit si bien Pierre-Louis, cette thèse, c'est également un CAP tourisme !

Je tiens en premier lieu à remercier mes directeurs de thèse : Thierry Berger et Marine Minier. Ainsi que François Arnault, à qui je donne le statut de directeur de thèse de manière officieuse.

Je remercie les membres de mon jury Anne Canteaut, Henri Gilbert et Louis Goubin d'avoir accepté d'évaluer mes travaux.

Je n'oublie pas mes coauteurs Cédric Lauradoux et Raphael Phan. Ainsi que les membres de l'ANR Rapide, en particulier Céline Blondeau, Pascale Charpin, Yann Laigle-Chapuy, Maria Naya-Plasencia, Andrea Röck et Marion Videau.

Je remercie aussi chaleureusement mon tuteur pédagogique Abdelkader Necer, ainsi que Julien Iguchi-Cartigny qui m'a aidé à m'intégrer dans le département informatique.

Je n'oublie pas les doctorants passés par le « bureau vert » : Ainhoa Aparicio-Monforte, Elsa Bousquet, Carole El Bacha et, évidemment, Pierre-Louis Cayrel. Mais également les doctorants de Limoges Boussad Ait Salem, Julien Angeli, Christophe Chabot, Daouda Diatta, Mickaël Pouchol et Romain Validire. Ainsi que Aurore Bernard, Sandrine Jean, Delphine Poingt, Adrien Poteaux et Hassan Saoud.

Les secrétaires du département de Mathématiques Aurélie Doucet, Odile Duval, Sylvie Laval, Nadia Rossi, Patricia Vareille et Yolande Vicelli. Ainsi que l'ensemble du département de Mathématiques. Plus particulièrement Philippe Gaborit, François Laubie, Henri Massias, Olivier Ruatta et Jacques-Arthur Weil.

Enfin, je remercie évidemment ma famille sans qui je ne serais pas arrivé là : mes parents et Mathieu. Et surtout Julie de son soutien tout au long de cette thèse.

Table des matières

Table des figures

Introduction

Les chiffrements à flot, les chiffrements par bloc et les fonctions de hachage sont les trois primitives utilisées en cryptographie symétrique. Dans cette thèse, nous nous intéressons essentiellement aux chiffrements à flot. La recherche dans ce domaine a été stimulée par le projet eSTREAM, qui s'est clos en 2008, visant à fournir de nouveaux standards. Ces chiffrements sont généralement basés sur des générateurs pseudo-aléatoires : ils doivent, à partir d'une clef de quelques centaines de bits, produire une suite très longue (plusieurs méga-octets, voire giga-octets). La suite produite doit « sembler » aléatoire : elle doit posséder de bonnes propriétés statistiques, avoir une longue période, ne pas permettre de retrouver la clef utilisée, etc. Les LFSRs ont longtemps été utilisés pour cela, ces automates produisent des séquences que l'on peut décrire en utilisant l'algèbre liée aux séries formelles sur un anneau. Mais l'avènement des attaques algébriques et par corrélation rend maintenant leur utilisation non-sûre. Lors de la compétition eSTREAM, le chiffrement à flot F-FCSR-H propose de remplacer les LFSRs par une nouvelle primitive : les FCSRs. Les suites qu'ils produisent sont également des objets mathématiques connus. Cependant, ces séquences sont par construction non-linéaires sur $\mathbb{F}_q$, à la différence de celles produites par les LFSRs.

Dans cette thèse, nous utilisons cette approche, et présentons un cadre théorique pour généraliser ces automates. Pour cela, nous présentons d'abord la topologie π-adique. Cette théorie est un puissant outil pour décrire simplement des suites respectant la structure d'un anneau. Nous introduisons ensuite la théorie des automates en toute généralité.

Nous présentons l'état de l'art des automates produisant des séquences algébriques. Plus précisément, nous nous intéressons aux LFSRs qui sont des automates dont les séquences produites s'expriment dans l'anneau des séries formelles $\mathbb{F}_q[[X]]$, aux FCSRs dont les séquences s'expriment dans l'anneau des entiers N-adiques $\mathbb{Z}_N$, et aux AFSRs dont les séquences s'expriment dans un anneau π-adique.

Nous combinons ensuite la théorie π-adique et la théorie des automates pour proposer une nouvelle approche des AFSRs. Notre approche est plus globale et permet, entre autres, de considérer des AFSRs possédant des entrées.

Nous nous attachons ensuite à spécifier ces résultats dans les cas où l'anneau π-adique est l'anneau des séries formelles $\mathbb{F}_q[[X]]$. Dans ce cas, cela revient à considérer les LFSRs. Ces automates sont connus et utilisés depuis très longtemps. Notre approche permet de donner une description plus concise de ces automates, et de donner explicitement des contraintes liées aux implémentations. En particulier, nous donnons des algorithmes permettant de construire des automates dont l'implémentation logicielle et/ou matérielle est efficace. Ces résultats forment un article actuellement soumis au journal « IEEE-TC ».

Nous étudions ensuite le cas des FCSRs, *i.e.* des AFSRs sur l'anneau des entiers N-adiques $\mathbb{Z}_N$. Alors qu'actuellement deux classes de ces automates sont connues : les FCSRs en mode Galois et en mode Fibonacci ; cette approche permet de définir une plus grande variété d'automates, en utilisant une description matricielle. De plus, elle autorise à considérer des automates dont l'implémentation utilise, en plus des classiques additionneurs à retenue, des soustracteurs à retenues. Ceci permet dans certains cas de baisser le coût lors de l'implémentation de ces

automates. Ces résultats forment un article actuellement soumis au journal « *Cryptography and Communications Discrete Structures, Boolean Functions and Sequences* ».

Nous présentons enfin deux nouvelles versions de chiffrements à flot basés sur des FCSRs construites grâce à ces travaux. En effet, l'utilisation de FCSR en mode Galois dans des chiffrements à flot permet de les attaquer. Grâce à nos travaux, il est possible de remplacer les FCSRs en mode Galois par des FCSRs « génériques » pour contrer ces attaques. De plus, les nouveaux FCSRs choisis dans ces chiffrement ont des implémentations efficaces. Ces chiffrements ont été présentés à « SAC » en 2009, et à « Indocrypt » en 2009.

En appendice, nous présentons des travaux supplémentaires liés aux chiffrements par bloc et aux fonctions de hachage. Les fonctions de hachage concentrent actuellement une grande attention de la communauté cryptographique, grâce à la tenue de la compétition SHA-3 depuis 2007. La fonction de hachage qui sera sélectionnée à la fin de cette compétition deviendra un nouveau standard. Plusieurs fonctions soumises utilisent un chiffrement par bloc dans leur spécification. Le fait de « détourner » un chiffrement par bloc pour une telle utilisation permet de considérer des nouvelles attaques. En particulier, la « clef » utilisée par le chiffrement par bloc n'est plus secrète : elle peut être connue, ou même choisie. Notre approche se fonde sur les propriétés intégrales de chiffrement par blocs et sur les distingueurs à clef connue et à clef choisie. En combinant ces deux objets, il est possible de construire des distingueurs pour les fonctions de hachage. En particulier, nous présentons les résultats obtenus sur les versions réduites des fonctions de hachage Hamsi-256, LANE-256 et Grøstl-512. Deux de ces travaux ont été présentés en conférence (à « Africacrypt 2009 » et à « CANS 2010 »), et un est actuellement soumis au journal « IEEE-IT ».

Notations

Notations d'arithmétique

Soient un anneau $\mathbb{A}$ et un idéal I de $\mathbb{A}$. Nous rappelons les notations suivantes :

- Pour tout $n \in \mathbb{N}$, la puissance n-ième de I, noté I^n, est l'idéal composé des sommes finies de produits de n éléments de I, *i.e.* un élément de I^n s'écrit

$$\sum_{0 \leq i < k} x_{i,0} \dots x_{i,n-1}$$

avec $k \in \mathbb{N}$ et $x_{i,j} \in I$ pour $0 \leq i < k$ et $0 \leq j < n$. Par convention, $I^0 = \mathbb{A}$. De plus, on a $I^{n+1} \subseteq I^n$.

- L'anneau quotient $\mathbb{A}/I$ est construit en quotientant l'anneau $\mathbb{A}$ par la relation d'équivalence $\mathcal{R}_I$ définie par :

$$\forall x, y \in \mathbb{A},\ (x \mathcal{R}_I y) \Leftrightarrow (x - y) \in I$$

Lorsque I est principal engendré par $\pi \in \mathbb{A}$, noté $I = \langle \pi \rangle$, l'anneau quotient $\mathbb{A}/\langle \pi \rangle$ sera noté simplement $\mathbb{A}/\pi$.

- On considère $\mathcal{S}$ un système de représentants de l'anneau quotient $\mathbb{A}/I$ dans $\mathbb{A}$, *i.e.* :
  - $\mathcal{S} \subset \mathbb{A}$;
  - Pour tout $a \in \mathbb{A}$, il existe $s \in \mathcal{S}$ tel que $a - s \in I$, *i.e.* s et a sont dans la même classe modulo I ;
  - Pour tout $s \neq s' \in \mathcal{S}$, $s - s' \notin I$, *i.e.* s et s' ne sont pas dans la même classe modulo I

  Nous noterons alors :
  - Étant donné $a, b \in \mathbb{A}$, le fait que a et b sont congrus modulo I, *i.e.* $a - b \in I$, sera noté $a \equiv b[I]$.
  - Étant donné $a \in \mathbb{A}$, l'unique élément de $\mathcal{S}$ congru à a modulo I sera noté $a \bmod I$.

  En particulier, la notation $b = a \bmod I$ équivaut à $a \equiv b[I]$ et $b \in \mathcal{S}$.

Notations d'algèbre linéaire

Nous rappelons ici des relations liées à l'inverse d'une matrice sur un anneau $\mathbb{A}$. Soit $M = (m_{i,j})_{0 \leq i,j < n}$ une matrice $n \times n$ inversible sur $\mathbb{A}$. On a la relation

$$M^{-1} = \frac{1}{\det M} \operatorname{Adj} M$$

où $\operatorname{Adj} M$ est la matrice adjointe de M. C'est-à-dire la transposée de la matrice des cofacteurs, *i.e.*

$$\begin{pmatrix} Cof_{0,0}(M) & Cof_{1,0}(M) & \dots & Cof_{n-1,0}(M) \\ \vdots & \vdots & & \vdots \\ Cof_{0,n-1}(M) & Cof_{1,n-1}(M) & \dots & Cof_{n-1,n-1}(M) \end{pmatrix}$$

avec

$$Cof_{i,j}(M) = (-1)^{i+j} \begin{vmatrix} m_{0,0} & \dots & m_{0,j-1} & m_{0,j+1} & \dots & m_{0,n-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ m_{i-1,0} & \dots & m_{i-1,j-1} & m_{i-1,j+1} & \dots & m_{i-1,n-1} \\ m_{i+1,0} & \dots & m_{i+1,j-1} & m_{i+1,j+1} & \dots & m_{i+1,n-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ m_{n-1,0} & \dots & m_{n-1,j-1} & m_{n-1,j+1} & \dots & m_{n-1,n-1} \end{vmatrix}$$

En particulier, étant donnés $\mathbb{B}$ un sous-anneau de $\mathbb{A}$, lorsque M est à coefficients dans $\mathbb{B}$, alors :

- $\det M$ est dans $\mathbb{B}$,
- $\operatorname{Adj} M$ est à coefficients dans $\mathbb{B}$.

En particulier, M^{-1} est à coefficients dans

$$\mathcal{Q} = \left\{ \frac{a}{b},\ a, b \in \mathbb{B} \text{ avec } b \text{ inversible} \right\} \subseteq \mathbb{A}$$

Notations dans les schémas

Dans les différents schémas, on respectera les conventions suivantes :

- Un rectangle représente une cellule mémoire,
- Un rectangle avec les angles arrondis représente une fonction,
- Pour marquer qu'un ensemble de mémoires/fonctions forment un bloc, on les encadrera avec un rectangle pointillé.

Notations du chapitre 1

$\mathcal{A}$	Un alphabet
n_X	La longueur du mot X de $\mathcal{A}$
$M = (m_0, \dots, m_{n_M-1})$	Un message clair sur $\mathcal{A}$
$C = (c_0, \dots, c_{n_C-1})$	Un message chiffré sur $\mathcal{A}$
$S = (s_0, \dots, s_{n_S-1})$	Une suite chiffrante sur $\mathcal{A}$
$K = (k_0, \dots, k_{n_K-1})$	Une clef sur $\mathcal{A}$
$IV = (i_0, \dots, i_{n_I-1})$	Un vecteur d'initialisation sur $\mathcal{A}$
$(\cdot\|\cdot)$	L'opérateur de concaténation
χ_a	Une fonction inversible de $\mathcal{A}$ dans $\mathcal{A}$ paramétrée par $a \in \mathcal{A}$

Notations du chapitre 2

$s_{k,k_0}\left((e_i)_{i\in\mathbb{N}}\right)$	La k-décimation de la suite $(e_i)_{i\in\mathbb{N}}$, *i.e.* $(e_{i\cdot k+k_0})_{i\in\mathbb{N}}$
$\mathbb{A}$	Un anneau intègre
π	Un élément de $\mathbb{A}$ tel que $\bigcap_{i\in\mathbb{N}}\langle\pi^i\rangle = \{0\}$
$\mathbb{A}_\pi$	L'anneau π-adique construit par complétion de $\mathbb{A}$ en utilisant la métrique π-adique
$\mathcal{S}$	Un système de représentants fondamental de $\mathbb{A}/\pi$ dans $\mathbb{A}$
$a = \sum_{i\in\mathbb{N}} \alpha_i\pi^i \in \mathbb{A}_\pi$ avec $\alpha_i \in \mathcal{S}$	Un élément de $\mathbb{A}_\pi$ et son développement de Hensel
$\mathrm{seq}_{\mathcal{S}}(a)$	La suite $(\alpha_i)_{i\in\mathbb{N}}$
$a \bmod \pi$	α_0
$a \operatorname{div} \pi$	$\sum_{i\in\mathbb{N}} \alpha_{i+1}\pi^i$

Notations du chapitre 4

$\Sigma(s, i_0)$	$\sum_{i\in\mathbb{N}} s_{i+i_0}\pi^i$ avec $s = (s_i)_{i\in\mathbb{N}}$ une suite de $\mathcal{S}$

Notations des chapitres 6, 7, 8 et 9

$\ll j$	L'opération de décalage à gauche de j positions
$\gg j$	L'opération de décalage à droite de j positions
$\lll j$	L'opération de rotation à gauche de j positions
$\ggg j$	L'opération de rotation à droite de j positions

Notations du chapitre 7

$k = (k_{n-1}\dots k_0)_2$	$k = \sum_{i=0}^{n-1} k_i 2^i$

Première partie

Préliminaires

Chapitre 1

Chiffrement à flot

Sommaire

1.1 Introduction

La cryptographie s'attache à protéger l'**information**, c'est-à-dire les données (textes, images, bases de données, etc). Cela revêt trois aspects distincts et complémentaires :

La confidentialité : l'information ne doit être accessible qu'aux personnes autorisées. Par exemple, les chaînes de télévision payantes ne sont accessibles qu'aux abonnés. En particulier, tout le monde a accès au signal « crypté », mais seuls les abonnés possèdent la clef pour accéder aux programmes en clair.

L'intégrité : l'information ne doit pas être altérée ni effacée (de façon volontaire ou accidentelle). Par exemple, sur un chèque bancaire, on ne peut pas modifier le montant indiqué.

L'authentification : Chaque entité doit être identifiable de façon certaine. Par exemple, en utilisant un système biométrique.

Ces trois branches se basent sur l'utilisation de secrets appelés **clefs**.

Les chiffrements traitent la confidentialité. Leur but est de permettre à des personnes de communiquer de façon privée via un canal non-sûr. Ils s'utilisent de la manière suivante : supposons qu'Alice veuille envoyer un message à Bob sur un canal écouté par un espion Oscar :

– Alice chiffre le message M avec sa clef K_A et la fonction de chiffrement E :

$$C = E_{K_A}(M)$$

– Alice envoie C à Bob ;
– Bob déchiffre C avec sa clef K_B et la fonction de déchiffrement D :

$$M = D_{K_B}(C)$$

Oscar écoute leur conversation sur le canal, il récupère donc C qui est incompréhensible sans la clef.

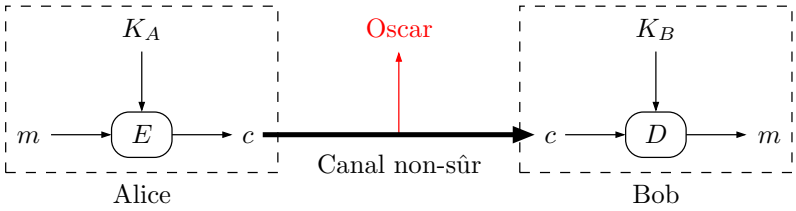


FIGURE 1.1 – Principe du chiffrement

Deux grandes familles de chiffrements existent :

Les chiffrements à clef privée (ou symétrique) : les clefs utilisées par Alice et Bob sont identiques : $K_A = K_B$. Cela a pour inconvénient que la clef doit rester secrète pour toute autre personne : il faut assurer la confidentialité de la clef pour assurer la confidentialité du message. En particulier, la clef doit être transmise de façon sûre entre Alice et Bob.

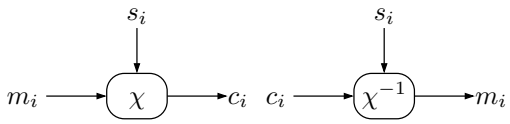
Les chiffrements à clef publique (ou asymétrique) : les clefs utilisées par Alice et Bob sont différentes. Dans le schéma 1.1, Bob construit K_A (appelé clef publique) et K_B (appelé clef secrète). Il met à disposition de tout le monde K_A, mais garde K_B secrète. En particulier, contrairement aux chiffrements symétriques, il n'y a pas de problème pour diffuser K_A. Ces chiffrements se basent sur des problèmes complexes tels que la factorisation de grands nombres ou le logarithme discret dans un groupe. L'inconvénient majeur de ces chiffrements est leur lenteur et leur coût.

Bien qu'il existe des chiffrements à flot à clef publique (par exemple [BG85]), nous traitons les chiffrements à flot à clef privée. Ils chiffrent un caractère du message à la fois, via une transformation qui varie au cours du temps. En comparaison, les chiffrements par blocs chiffrent un groupe de caractères du message simultanément, en utilisant une transformation fixée. Les chiffrements à flot sont plus rapides et requièrent des circuits plus simples que les chiffrements par blocs. Ils sont particulièrement utiles lorsque les ressources pour le chiffrement sont limitées, ou encore quand les caractères chiffrés peuvent être traités individuellement lors de la réception (par exemple dans les télécommunications). On les formalise ainsi : un chiffrement à flot est constitué :

– d'un algorithme qui produit une suite chiffrante $s_0, s_1, \ldots, s_{n-1} \in \mathcal{A}$ à partir de la clef K (et éventuellement du message),
– d'un ensemble de fonctions χ_a indexées par $a \in \mathcal{A}$ inversibles de $\mathcal{A}$ dans $\mathcal{A}$.

Alors, étant donné un message $m_0, m_1, \ldots, m_n$ et le message chiffré $c_0, c_1, \ldots, c_n$ correspondant, on a :

$$c_i = \chi_{s_i}(m_i) \qquad \text{(Chiffrement)}$$
$$m_i = \chi_{s_i}^{-1}(c_i) \qquad \text{(Déchiffrement)}$$



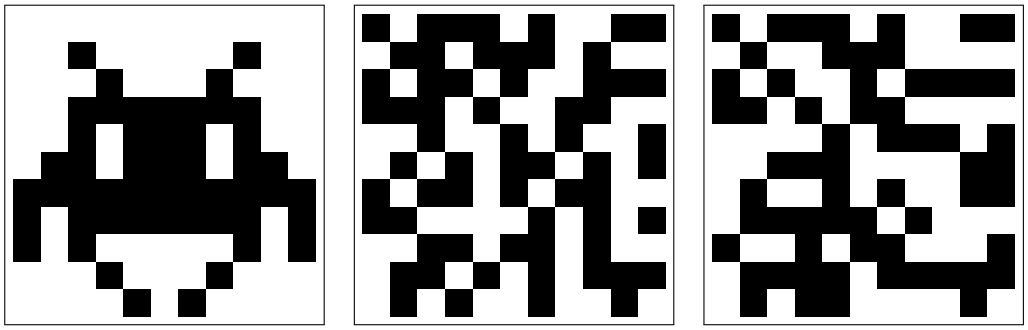
(a) Chiffrement (b) Déchiffrement

FIGURE 1.2 – Principe des chiffrements à flot

La figure 1.2 représente ce fonctionnement.

Lorsque l'alphabet $\mathcal{A}$ est un groupe (noté additivement), la fonction χ_a peut être remplacée par l'addition de a dans le groupe : $\chi_a(\cdot) = a + \cdot$. Dans ce cas, le chiffrement est dit **additif**.

La figure 1.3 présente un triplet message clair/suite chiffrante/message chiffré construit en utilisant l'addition binaire.



(a) Message clair (b) Suite chiffrante (c) Message chiffré

FIGURE 1.3 – Exemple de Message clair/Suite chiffrante/Message chiffré

Il existe trois classes de chiffrement à flot : le **masque jetable**, les **chiffrements synchrones** et les **chiffrements auto-synchronisants**. Le masque jetable a une clef de même longueur que le message et qui ne peut être utilisée qu'une seule fois. Les deux autres classes utilisent la même clef pour chiffrer plusieurs messages. Si le chiffrement ne dépend que de la clef, chaque chiffrement produit alors la même suite chiffrante, ce qui conduit à des attaques. C'est pourquoi les chiffrements à flot utilisent une donnée supplémentaire publique, appelée **vecteur d'initialisation** (noté IV), qui change à chaque chiffrement pour construire l'état initial et donc la suite chiffrante.

Dans ce qui suit, on considère que :
- les messages clairs $M = (m_0, \ldots, m_{n_M-1})$,
- les messages chiffrés $C = (c_0, \ldots, c_{n_C-1})$,
- les suites chiffrantes $S = (s_0, \ldots, s_{n_S-1})$,
- les clefs $K = (k_0, \ldots, k_{n_K-1})$,
- les vecteurs d'initialisation $IV = (i_0, \ldots, i_{n_I-1})$

sont des mots sur un alphabet $\mathcal{A}$.

1.1.1 Masque jetable

Ce chiffrement a été proposé en 1917 par Gilbert Vernam [Ver19]. On le présente ici dans une version généralisée. Pour $a \in \mathcal{A}$, soit χ_a une fonction inversible de $\mathcal{A}$ dans $\mathcal{A}$. Étant donné une clef K sur $\mathcal{A}$, un message M sur $\mathcal{A}$ et C le chiffré correspondant, on a la relation suivante :

$$c_i = \chi_{k_i}(m_i) \qquad \text{(Chiffrement)}$$
$$m_i = \chi_{k_i}^{-1}(m_i) \qquad \text{(Déchiffrement)}$$

En général, ce chiffrement est présenté avec $\mathcal{A} = \{0, 1\}$ et pour $a, b \in \mathcal{A}$, $\chi_a(b) = a \oplus b$ où $\oplus$ est l'addition binaire. L'idée de ce chiffrement est que la suite chiffrante s_i est égale à la clef k_i.

Sa sécurité est parfaite si l'on utilise une clef aléatoire une seule fois. Ce chiffrement fut utilisé pour chiffrer le « Téléphone rouge » entre Moscou et Washington.

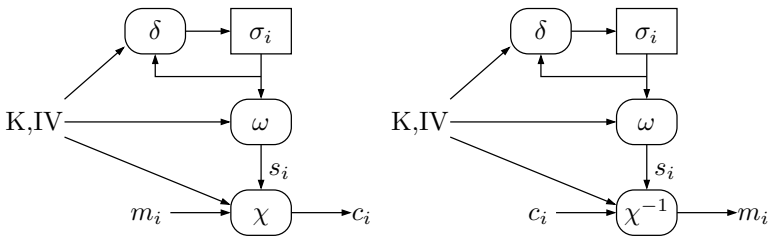
Cependant, le fait que la clef doive être aussi longue que le message le rend quasiment inutilisable en pratique. Les chiffrements synchrones et auto-synchronisants se basent sur ce principe, mais en générant une suite chiffrante de grande longueur à partir de la clef et du vecteur d'initialisation.

1.1.2 Chiffrements synchrones

Les chiffrements synchrones sont modélisés grâce à :
- un ensemble d'états Σ,
- une fonction d'initialisation (ou *key/IV setup*) de $\mathcal{A}^{n_K} \times \mathcal{A}^{n_I}$ dans Σ,
- une fonction de transition δ de Σ dans Σ,
- une fonction d'extraction ω de Σ dans $\mathcal{A}$,
- un ensemble de fonctions inversibles de combinaisons χ_a de $\mathcal{A}$ dans $\mathcal{A}$ paramétrées par $a \in \mathcal{A}$.

Grâce au *key/IV setup*, on choisit $\sigma_0 \in \Sigma$ l'état initial du chiffrement. Pour chaque nouveau caractère à chiffrer, l'état interne est mis à jour avec δ. Un élément de la suite chiffrante est extrait avec ω. La suite chiffrante est combinée avec le message clair par le biais de χ. Pour le déchiffrement, χ^{-1} est utilisé. Les fonctions δ, ω et χ dépendent du couple (K, IV). On a, comme présenté dans la figure 1.4, pour $i \in \mathbb{N}$:

$$\sigma_0 = \text{Key/IV Setup}(K|IV) \quad \text{(Initialisation)}$$
$$\sigma_{i+1} = \delta(\sigma_i) \quad \text{(Transition)}$$
$$s_i = \omega(\sigma_i) \quad \text{(Extraction)}$$
$$c_i = \chi_{s_i}(m_i) \quad \text{(Chiffrement)}$$
$$m_i = \chi_{s_i}^{-1}(c_i) \quad \text{(Déchiffrement)}$$



(a) Chiffrement (b) Déchiffrement

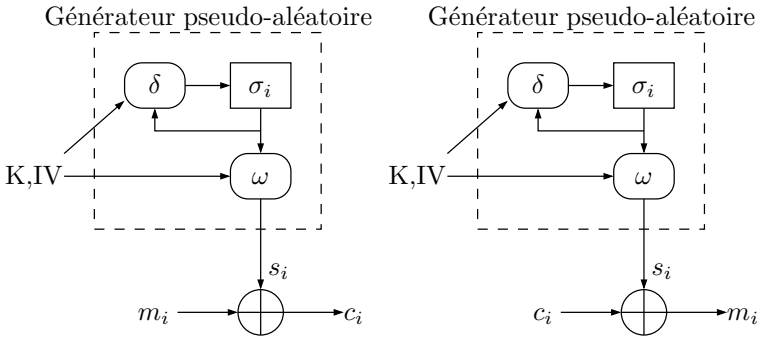
FIGURE 1.4 – Principe des chiffrements à flot synchrones

Dans le cas d'un chiffrement synchrone additif, l'état interne, la fonction de transition et la fonction d'extraction forment un générateur pseudo-aléatoire (figure 1.5).

1.1.3 Chiffrements auto-synchronisants

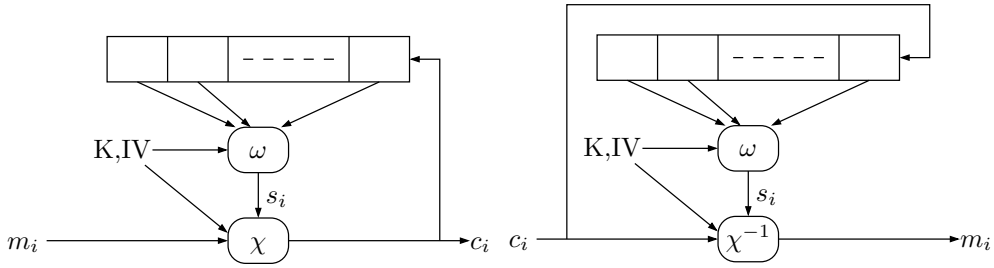
Les chiffrements auto-synchronisants produisent la suite chiffrante à partir du couple clef/vecteur d'initialisation et d'un nombre fixé d'éléments de la suite chiffrante déjà produits (figure 1.6).

Ainsi, même si certains éléments du message chiffré sont perdus, le chiffrement est capable de se resynchroniser après un petit nombre d'itérations. Les chiffrements auto-synchronisants sont



(a) Chiffrement (b) Déchiffrement

FIGURE 1.5 – Principe des chiffrements à flot synchrones additifs



(a) Chiffrement (b) Déchiffrement

FIGURE 1.6 – Principe des chiffrements à flot auto-synchronisants

cependant très vulnérables aux attaques à clair choisi [JM06b, DP05, ZWFB05]. En pratique, on utilise plutôt des chiffrements synchrones avec un mécanisme de resynchronisation externe.

1.2 Générateurs et suites pseudo-aléatoires

Nous présentons ici les générateurs et suites pseudo-aléatoires. Nous avons vu que les chiffrements synchrones additifs sont basés sur de tels générateurs pour produire la suite chiffrante. Le concept de générateur pseudo-aléatoire a été formalisé principalement par Blum, Micali et Yao [BM84, Yao82]. Leur approche donne un cadre formel qui est cependant inatteignable en pratique. Nous présenterons alors quelques tests classiques que l'on applique à des suites et générateurs.

1.2.1 Aspect formel : définitions

Nous présentons ici les générateurs et les suites pseudo-aléatoires. Ces objets ont été étudiés particulièrement par Blum, Micali et Yao [BM84, Yao82]. Nous ne nous attarderons pas ici sur ces travaux, essentiellement basés sur les probabilités. Nous donnons cependant les définitions suivantes :

Définition 1.2.1

Soient X et Y deux variables aléatoires. X et Y sont dites ***indistinguables en temps polynomial*** *si pour tout algorithme probabiliste A, pour tout polynôme p et pour n suffisamment grand, on a :*

$$\left|Pr\big(A(x)=1|x\leftarrow X\big)-Pr\big(A(y)=1|y\leftarrow Y\big)\right|<\frac{1}{p(n)}$$

Définition 1.2.2

Soient E un ensemble, $g\colon E^k \to E^l$ avec $k < l$. On note U_{E^k} (respectivement U_{E^l}) la distribution uniforme sur E^k (respectivement sur E^l). g est appelé ***générateur pseudo-aléatoire*** *si $g(U_{E^k})$ et U_{E^l} sont indistinguables en temps polynomial. Les éléments de E^k sont alors appelés les* ***graines****, et les éléments de E^l des* ***suites pseudo-aléatoires****.*

En d'autres termes, un générateur pseudo-aléatoire est un algorithme déterministe qui, étant donné une entrée aléatoire de longueur k appelée graine, produit une suite de longueur l, appelée suite pseudo-aléatoire, qui « semble » aléatoire.

1.2.2 Aspect pratique : Tests statistiques

La notion d'indistinguabilité en temps polynomial est fondamentale pour les suites et générateurs pseudo-aléatoires. Cependant, elle est impraticable car, par la définition 1.2.1, pour dire qu'une suite est indistinguable en temps polynomial, il faut qu'aucun algorithme probabiliste ne puisse la distinguer d'une suite aléatoire. En particulier, il faudrait donc lui appliquer tous les algorithmes probabilistes. En pratique, on a recours, en premier lieu, à des batteries de tests statistiques pour détecter les faiblesses des générateurs. Les tests du NIST [RSN+01] et de Marsaglia [Mar96] sont les plus utilisés.

Chaque test statistique détermine si la séquence soumise présente, ou pas, les mêmes caractéristiques qu'une suite réellement aléatoire vis-à-vis d'une certaine propriété. Plus précisément, chaque test calcule une donnée statistique liée à la séquence soumise, compare cette valeur à la valeur théorique (qui suit le plus souvent une loi normale ou du χ^2). Si ces valeurs sont trop éloignées, la séquence est rejetée, *i.e.* vis-à-vis de la propriété testée, elle ne se comporte pas comme une séquence aléatoire ; sinon, elle est acceptée, *i.e.* vis-à-vis de la propriété testée, elle se comporte comme une séquence aléatoire.

Les postulats de Golomb

Historiquement, S.W. Golomb est le premier à énoncer des critères qu'une suite pseudo-aléatoire doit satisfaire [GW82]. Ces postulats nécessitent trois définitions préalables :

Définition 1.2.3 (*[GK09]*)

Soit s une suite strictement T-périodique sur $\mathcal{A}$. s sera dite équidistribuée à l'ordre r si, pour tout $1 \leq k \leq r$ et pour tout mot b de longueur k, le nombre d'occurrences $N(b)$ de b sur une période de s vérifie :

$$\left\lfloor \frac{T}{|\mathcal{A}|^k} \right\rfloor \leq N(b) \leq \left\lceil \frac{T}{|\mathcal{A}|^k} \right\rceil$$

Définition 1.2.4 (*[GK09]*)

Soit $s = (s_i)_{\in\mathbb{N}}$ une suite strictement T-périodique sur $\mathcal{A}$.

- *On appelle* **run de longueur** *k un mot de longueur k constitué de symboles identiques,*

qui n'est pas contenu dans un mot de longueur $k+1$ constitué de symboles identiques, i.e. $(s_i, \ldots, s_{i+k-1})$ est un run *de longueur k si :*

$$(s_i = \cdots = s_{i+k-1}) \text{ et } (s_{i-1} \neq s_i) \text{ et } (s_{i+k} \neq s_{i+k-1})$$

– *s possède la propriété des* runs *si, pour tout k, le nombre $N(k)$ de* runs *de longueur k sur une période vérifie :*

$$\left\lfloor \frac{T \cdot (|\mathcal{A}| - 1)^2}{|\mathcal{A}|^{k+1}} \right\rfloor \leq N(k) \leq \left\lceil \frac{T \cdot (|\mathcal{A}| - 1)^2}{|\mathcal{A}|^{k+1}} \right\rceil$$

Définition 1.2.5

*Soit $s = (s_n)_{n\in\mathbb{N}}$ une suite binaire de période N. **La fonction d'auto-corrélation** C de s est définie pour $0 \leq t \leq N-1$ par :*

$$C(t) = \frac{1}{N} \sum_{i=0}^{N-1} (-1)^{s_i + s_{i+t}}$$

La fonction d'auto-corrélation mesure la distance entre la suite s et la suite obtenue en la décalant de t positions, c'est-à-dire $(s_{n+t})_{n\in\mathbb{N}}$. En effet, $C(t)$ correspond à la différence entre le nombre de positions en lesquelles ces deux suites coïncident et le nombre de positions en lesquelles elles diffèrent.

S.W. Golomb propose les critères suivants pour distinguer les suites pseudo-aléatoires binaires :

Définition 1.2.6 (*Postulats de Golomb*[GW82])

Soit s une suite périodique binaire de période T. Les postulats suivants sont connus sous le nom de postulats de Golomb :

1. *s est équidistribuée à l'ordre 1 ;*
2. *s possède la propriété des* runs *;*
3. *La fonction d'auto-corrélation C ne prend que deux valeurs, i.e. il existe $K \in \mathbb{Z}$ tel que*

$$T \cdot C(t) = \begin{cases} T & si \quad t = 0 \\ K & si \quad 1 \leq t \leq T-1 \end{cases}$$

Depuis 1967, l'utilisation des suites pseudo-aléatoires s'est répandue, et les propriétés d'aléa requises sont en même temps devenues plus sophistiquées. En particulier, les postulats de Golomb ne sont plus utilisés car ils ne sont pas « flexibles » : la suite considérée doit vérifier ces critères précisément, sans marge de manœuvre. Par exemple, soit une suite s vérifiant les postulats de Golomb. On note $(s_0, \ldots, s_{T-1})$ une période de s. Soit la suite s' construite à partir de s en rajoutant un 0 : une période de s' est égale à $(s_0, \ldots, s_{i-1}, 0, s_i, \ldots, s_{T-1})$ pour un certain i, $1 \leq i \leq T-1$. Alors s' ne vérifie pas les postulats de Golomb. Cependant, si s est pseudo-aléatoire, s' l'est moralement tout autant.

Tests classiques

Les tests que l'on utilise sont des tests probabilistes. On présente ici quelques tests classiques du NIST [RSN+01] :

Test de fréquence : le nombre de 0 et le nombre de 1 doivent être proches.

Test de fréquence par blocs : les nombres d'occurrences de chacun des 2^k blocs de longueur k doivent être proches.

Test du rang de la matrice : à partir de la suite, on construit des matrices 32×32. On vérifie alors si les rangs de ces matrices suivent une loi du χ^2.

Test spectral : ce test se base sur la transformée de Fourier discrète de la suite.

Test universel de Maurer : une suite aléatoire ne doit pas pouvoir être significativement compressée.

Test de complexité linéaire : la suite ne doit pas suivre une récurrence linéaire.

Test de la marche aléatoire : on se place dans un espace à une dimension, et on interprète chaque élément de la suite comme étant un pas en avant ou en arrière à partir de l'origine. Le test vérifie alors, qu'en suivant ce chemin, le nombre de passages à un endroit suit une loi du χ^2.

Les tests de la batterie DIEHARD [Mar96] se basent sur des propriétés plus profondes, et ont donc des descriptions plus complexes. Nous en décrivons seulement deux :

Distances entre les anniversaires : ce test se base sur le fait suivant : on considère m anniversaires dans une année de n jours. On liste les distances entre les anniversaires. Soit j le nombre de valeurs qui apparaissent plus d'une fois dans cette liste. Alors j suit une loi de Poisson de moyenne $m^3/(4n)$. Le test consiste alors à construire $m = 2^9$ anniversaires à partir de la suite, et à vérifier que cette propriété est respectée.

Test du Craps : le Craps est un jeu de dés populaires aux États-Unis. Ce test consiste à jouer 200 000 parties de Craps en utilisant la suite pour simuler les lancers de dés, et on vérifie alors que le nombre de parties gagnées et le nombre de lancers nécessaires respectent les statistiques.

1.3 Générateurs pseudo-aléatoires en cryptographie

1.3.1 Modèle

Le concept de générateur pseudo-aléatoire peut se formaliser ainsi [GK09] :

Définition 1.3.1

Un générateur pseudo-aléatoire est la donnée d'un quadruplet $(\Sigma, \mathcal{O}, \delta, \omega)$ *:*
- Σ *: un ensemble d'états ;*
- $\mathcal{O}$ *: un alphabet de sortie ;*
- δ *: une fonction de transition de* Σ *dans* Σ *;*
- ω *: une fonction d'extraction de* Σ *dans* $\mathcal{O}$*.*

Étant donné un état initial $\sigma_0 \in \Sigma$*, le générateur produit la suite :*

$$(\omega(\sigma_0), \omega(\delta(\sigma_0)), \omega(\delta^2(\sigma_0)), \dots) = (\omega(\delta^i(\sigma_0)))_{i \in \mathbb{N}}$$

Cette définition est compatible avec la description d'un chiffrement synchrone. En pratique, cette définition est interprétée ainsi :
- Un générateur est une « machine » avec un état interne dans Σ ;
- Son premier état est σ_0 ;
- Le générateur met à jour son état en utilisant la fonction δ :

$$\sigma_{i+1} = \delta(\sigma_i), \ \forall i \in \mathbb{N}$$

– Le générateur extrait de chaque état un caractère :

$$s_i = \omega(\sigma_i)$$

Ce mécanisme, dans le cas des chiffrements à flot, peut être représenté comme dans la figure 1.7. Dans ce cas particulier, comme présenté plus haut, l'état initial est choisi avec une procédure de *key/IV setup*.

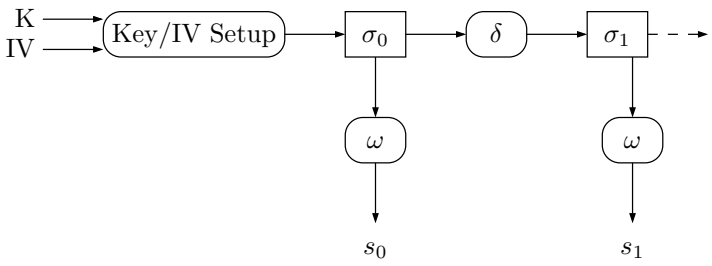


FIGURE 1.7 – Fonctionnement d'un générateur pseudo-aléatoire utilisé dans un chiffrement à flot.

1.3.2 Exemple de générateur sûr

Des générateurs pseudo-aléatoires prouvés sûrs existent. Par exemple le générateur Blum-Blum-Shub (ou BBS) [BBS86], sous l'hypothèse que la factorisation est un problème dur, est sûr, *i.e.* il n'existe pas d'algorithme en temps polynomial capable de distinguer la suite produite d'une suite aléatoire. Le générateur BBS est décrit par l'algorithme I.1.

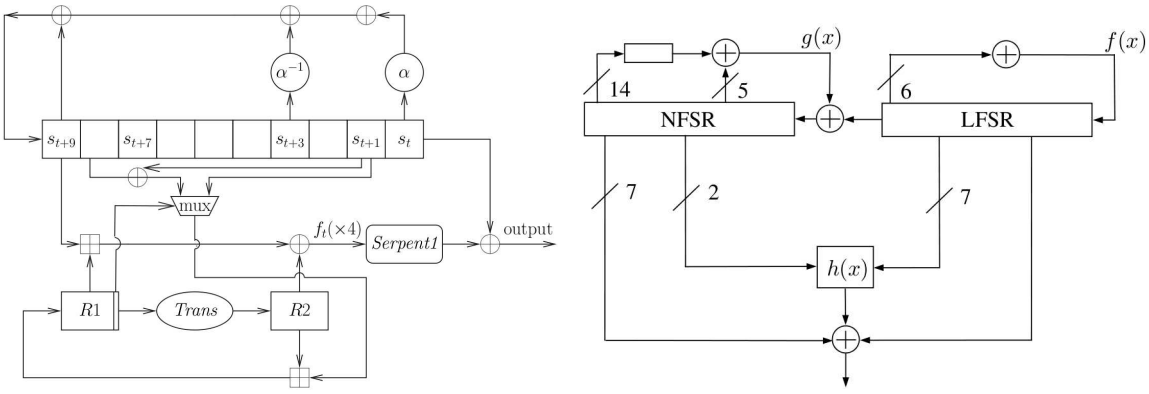
Algorithme I.1 : Générateur pseudo-aléatoire Blum-Blum-Shub

Données : p et q deux grands entiers premiers congrus à 3 modulo 4, $n = pq$.
Entrées : La graine $1 \leq s \leq n-1$ telle que s et n sont premiers entre eux ; l la longueur de la suite voulue.
Sorties : $z_1, \ldots, z_l$ une suite pseudo-aléatoire.
début
  $x_0 \leftarrow s^2 \bmod n$;
  pour $i = 1$ **à** l **faire**
    $x_i \leftarrow x_{i-1}^2 \bmod n$;
    $z_i \leftarrow x_i \bmod 2$;
  retourner $z_1, \ldots, z_l$;
fin

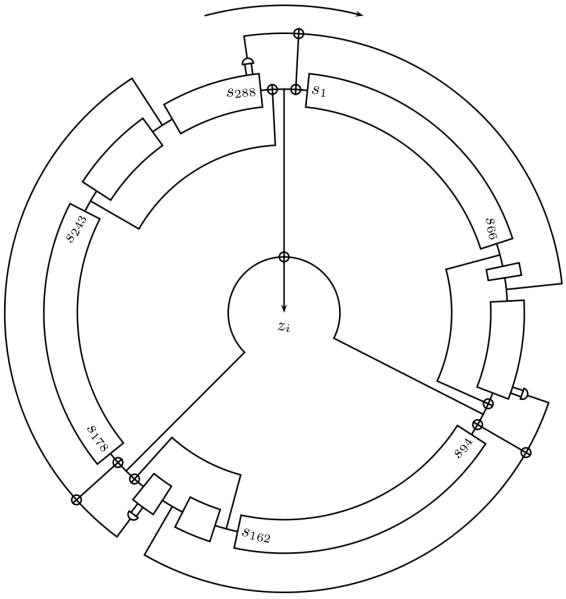
Cet exemple est représentatif des inconvénients liés aux générateurs prouvés sûrs : ils sont coûteux, car ils se basent généralement sur des problèmes de théorie des nombres. Pour le générateur BBS, chaque bit produit nécessite le calcul d'un carré modulo n, avec n de taille 1024 bits pour une application standard.

1.3.3 Générateurs basés sur des LFSRs

En pratique, pour du chiffrement à flot où le débit doit être important, on utilise des générateurs pseudo-aléatoires qui ne sont pas prouvés sûrs, mais qui ont été soumis à la communauté cryptographique et ont résisté aux attaques. Différents exemples de chiffrements à flot sont présentés dans la figure 1.8.



(a) Sosemanuk (b) Grain



(c) Trivium

FIGURE 1.8 – Trois finalistes de eSTREAM

L'un des éléments communément utilisé pour construire de tels générateurs est le registre à décalage linéaire (*Linear Feedback Shift Register* ou LSFR). Il est constitué de mémoire et a une fonction de transition linéaire. Dans le cas binaire, il ne nécessite que des portes `OU exclusif` pour être implémenté (figure 1.9). Il produit des suites avec une large période et de bonnes propriétés statistiques.

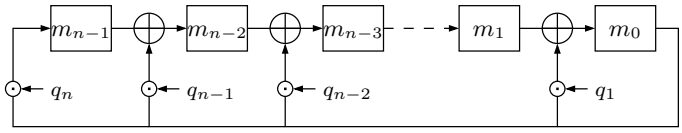


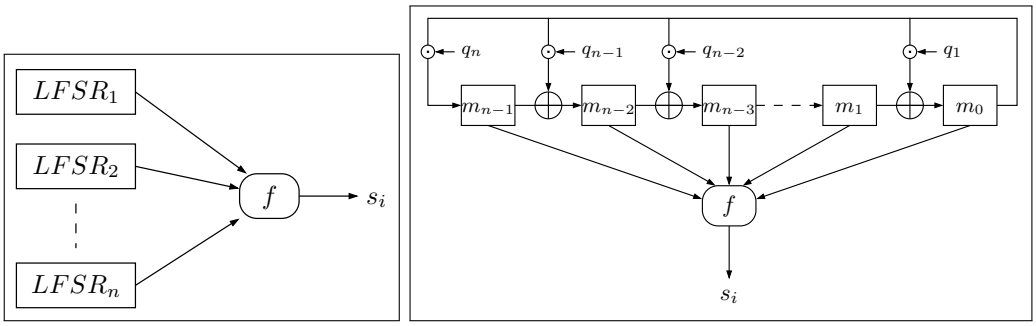
FIGURE 1.9 – Exemple de LFSR binaire.

Cependant, les LFSRs ne peuvent pas être utilisés seuls, car l'algorithme de Berlekamp-Massey [Mas69] permet de retrouver l'état d'un LFSR composé de n mémoires et sa fonction de transition à partir de $2n$ éléments consécutifs de la suite. Trois constructions classiques existent pour casser la structure linéaire (figure 1.10) :

Les générateurs à combinaison non-linéaire : Ces générateurs sont composés de plusieurs LFSRs en parallèle, et d'une fonction de combinaison f non-linéaire. La fonction f doit vérifier différentes propriétés comme avoir un haut degré, être équilibrée, etc.

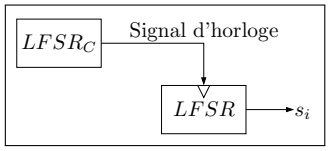
Les générateurs à filtre non-linéaire : Ces générateurs appliquent une fonction non-linéaire f à l'ensemble de l'état d'un LFSR. Comme pour les générateurs à combinaison linéaire, la fonction f doit avoir un haut degré, être équilibrée, etc.

Les générateurs à horloge contrôlée : Ces générateurs utilisent au moins deux LFSRs. $LFSR_C$ est un LFSR binaire classique qui contrôle la façon dont le second est *clocké*, *i.e.* en fonction du bit produit par $LFSR_C$, le second LFSR met à jour son état ou pas.



(a) Générateurs à combinaison non-linéaire

(b) Générateurs à filtre non-linéaire



(c) Générateurs à horloge controlée

FIGURE 1.10 – Trois constructions classiques basées sur des LFSRs

Les générateurs à combinaison non-linéaire et à filtre non-linéaire sont les plus courants. Dans ces deux cas, la fonction f est souvent coûteuse à implémenter, car elle nécessite de nombreuses portes logiques et une grande profondeur de circuit. De plus, ces générateurs ne sont maintenant plus utilisés, car les attaques algébriques [CM03], par corrélation [Sie84], par corrélation rapide [MS89, CJS00] et par compromis temps-mémoire [Bab95] permettent de les casser.

1.3.4 Types d'attaques

La sécurité des chiffrements à flot est évaluée dans le contexte d'une attaque à clair connu. Nous donnons ici les types d'attaques contre les chiffrements à flot, de la plus faible à la plus forte :

Les attaques par distingueur : Ces attaques déterminent si, étant donné une suite, elle a été produite par un générateur pseudo-aléatoire donné.

Les attaques par prédiction du bit suivant : Ces attaques consistent, à partir de la connaissance des n premiers bits d'une suite engendrée par un générateur, à prédire le bit suivant.

Les attaques par recouvrement de l'état initial : Ces attaques visent à retrouver l'initialisation du générateur, *i.e.* l'état du générateur après le *key/IV setup*.

Les attaques par recouvrement de clef : Ces attaques visent à recouvrer la clef elle-même.

1.3.5 Attaques génériques

De plus, il existe de nombreuses attaques génériques sur les chiffrements à flot telles que les attaques par corrélation [Sie84], par corrélation rapide [MS89, CJS00], par compromis temps-mémoire [Bab95] et les attaques algébriques [CM03]. Ces attaques donnent des contraintes pour le design des chiffrements à flot :

- La période du générateur doit être longue et la fonction de filtre équilibrée pour que la suite chiffrante soit pseudo-aléatoire ;
- L'état interne doit être au moins deux fois plus grand que la clef pour éviter les attaques par compromis temps-mémoire [Bab95] ;
- La fonction de transition et/ou la fonction d'extraction doivent avoir un haut degré sur $\mathbb{F}_2$ pour éviter les attaques algébriques [CM03].

1.4 eSTREAM

En 2004, suite à l'échec du concours lancé par le projet NESSIE pour recommander un chiffrement à flot, Shamir demande à la communauté « *Stream Ciphers : Dead or Alive ?* » [Sha04]. Dans ce contexte, le projet eSTREAM est lancé pour stimuler la recherche dans ce domaine. Après trois ans, sept candidats sont sélectionnés parmi les 34 soumissions initiales [BDCC+08] (Tableau 1.1). Les soumissions se divisaient en deux catégories en fonction de l'application visée : le profil matériel pour des chiffrements avec des ressources très contraintes, et le profil logiciel pour des chiffrements avec de haut débits. Les allgorithmes retenus pour le profil matériel étaient quatre, mais le générateur F-FCSR-H v2 [ABL08] a été retiré après une attaque de Hell et Johansson [HJ08].

Profil matériel	Profil logiciel
Grain v1	HC-128
MICKEY v2	Rabbit
Trivium	Salsa20/12
	SOSEMANUK

TABLE 1.1 – Algorithmes recommandés à l'issue de eSTREAM

1.5 Conclusion

Les chiffrements à flot sont un vaste sujet. Dans cette thèse, nous nous intéressons aux chiffrements à flot synchrones additifs. En particulier, nous étudions les générateurs pseudo-aléatoires basés sur des suites algébriques. Les LFSRs entrent dans cette catégorie car ils produisent des suites respectant la structure de l'anneau $\mathbb{F}_q[[X]]$. Nous étudierons aussi, comme alternative aux LFSRs, les FSCRs qui produisent des suites respectant la structure des entiers N-adiques $\mathbb{Z}_N$.

Chapitre 2

Suites algébriques

Sommaire

2.1 Introduction

Dans ce chapitre, nous présentons les suites algébriques basées sur la topologie π-adique. Celle-ci permet de construire des anneaux dont les éléments s'écrivent de façon unique sous forme de série. En considérant les coefficients de ces séries, on construit des suites. Dans certains cas, ces suites présentent des propriétés intéressantes telles qu'une longue période, une bonne répartition, etc. Nous traiterons alors en détail les anneaux $\mathbb{F}_q[[X]]$ et $\mathbb{Z}_N$.

2.2 Suites

2.2.1 Définitions

Nous rappelons dans cette section les définitions et les notations qui nous serons utiles.

Définition 2.2.1

Soit E un ensemble. On appelle suite de E tout élément de $E^{\mathbb{N}}$, i.e. tout ensemble dénombrable et ordonné d'éléments de E. Nous utiliserons la notation classique $(e_i)_{i\in\mathbb{N}}$ pour noter une telle suite.

Définition 2.2.2

Soit $e = (e_i)_{i\in\mathbb{N}}$ une suite de E. e est dite :

- ***périodique** ou T-**périodique**, s'il existe $T \in \mathbb{N}^*$ et $P \in \mathbb{N}$ tels que :*

$$\forall i \geq P,\ e_i = e_{i+T}$$

*P est appelé **une pré-période** de e ; la plus petite pré-période de e sera appelée **la pré-période** de e.*

- ***strictement périodique** ou **strictement** T-**périodique**, s'il existe $T \in \mathbb{N}^*$ tel que :*

$$\forall i \in \mathbb{N},\ e_i = e_{i+T}$$

*Dans les deux cas, T est appelé **une période** de e ; la plus petite période de e sera appelée **la période** de e.*

Étant donnée une suite $e = (e_i)_{i\in\mathbb{N}}$ strictement T-périodique, on introduit la notation suivante :

$$e = (\underline{e_0, e_1, \ldots, e_{T-1}})$$

De même, e étant T-périodique avec une pré-période P, on la notera :

$$e = (e_0, e_1, \ldots, e_{N-1}, \underline{e_P, e_{P+1}, \ldots, e_{P+T-1}})$$

Définition 2.2.3

*Soit $e = (e_i)_{i\in\mathbb{N}}$ une suite de E. Soient $k_0 \in \mathbb{N}$ et $k \in \mathbb{N}^*$. La suite de E $(e_{i\cdot k+k_0})_{i\in\mathbb{N}}$ est appelée **une k-décimation** de e, et est notée $s_{k,k_0}(e)$. Lorsque $k_0 = 0$, on notera simplement $s_k(e)$.*

2.2.2 Suites de De Bruijn

Nous présentons brièvement les suites de De Bruijn [DB69]. Ces séquences sont caractérisées par l'occurrence une seule fois de chaque mot de longueur k sur une période. Elles présentent de bonnes propriétés statistiques (équidistribuées, *runs* bien répartis, etc).

Définition 2.2.4

*Soit $s = (s_i)_{i\in\mathbb{N}}$ une suite strictement périodique de période T sur un alphabet $\mathcal{A}$. s est une **suite de De Bruijn** de largeur k sur $\mathcal{A}$ si chaque mot de longueur k apparaît exactement une fois sur une période de s, i.e.*

$$\forall (a_0, \ldots, a_{k-1}) \in \mathcal{A}^k, \exists! i \in \{0, \ldots, T-1\},\ (a_0, \ldots, a_{k-1}) = (s_i, \ldots, s_{i+k-1})$$

Par exemple, la séquence $(s_i)_{i\in\mathbb{N}} = (\underline{0, 1, 0, 1, 1, 1, 0, 0})$ est une suite de De Bruijn sur $\mathcal{A} = \{0, 1\}$ de largeur 3. En effet :

- le mot $(0, 0, 0)$ est le mot (s_6, s_7, s_8) ;
- le mot $(0, 0, 1)$ est le mot (s_7, s_8, s_9) ;
- le mot $(0, 1, 0)$ est le mot (s_0, s_1, s_2) ;
- le mot $(0, 1, 1)$ est le mot (s_2, s_3, s_4) ;
- le mot $(1, 0, 0)$ est le mot (s_5, s_6, s_7) ;
- le mot $(1, 0, 1)$ est le mot (s_1, s_2, s_3) ;
- le mot $(1, 1, 0)$ est le mot (s_4, s_5, s_6) ;
- le mot $(1, 1, 1)$ est le mot (s_3, s_4, s_5) ;

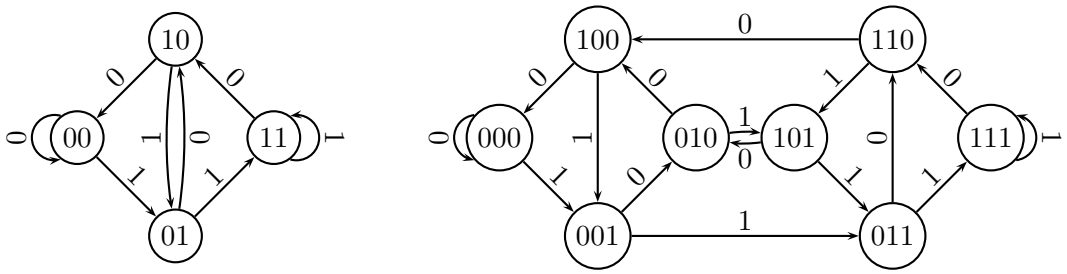
De même, $(\underline{0,1,1,1,0,1,0,0})$ est aussi une suite de De Bruijn sur $\mathcal{A} = \{0,1\}$ de largeur 3.

Ces suites sont liées à la théorie des graphes. Plus précisément, on définit le graphe de De Bruijn de dimension k sur un alphabet $\mathcal{A}$ ainsi :

- les sommets sont les mots de longueur k sur $\mathcal{A}$;
- il existe une arête, étiquetée $a \in \mathcal{A}$, du sommet $(s_1 \dots s_k)$ vers le sommet $(s'_1 \dots s'_k)$ si

$$s'_1 = s_2, s'_2 = s_3, \dots, s'_{k-1} = s_k \text{ et } s'_k = a$$

La figure 2.1 présente deux exemples de graphes de De Bruijn. Une méthode pour construire une séquence de De Bruijn de largeur k est de rechercher un cycle eulérien, *i.e.* un cycle passant par chaque arête exactement une fois, dans un graphe de De Bruijn de dimension $k-1$. Le mot composé par les étiquettes des arêtes empruntées constitue une suite de De Bruijn.



(a) Graphe de De Bruijn sur $\{0,1\}$ de dimension 2

(b) Graphe de De Bruijn sur $\{0,1\}$ de dimension 3

FIGURE 2.1 – Exemples de graphes de De Bruijn

On a les propriétés suivantes :

Proposition 2.2.5 (*[GK09]*)

Soit s une suite de De Bruijn de largeur k sur un alphabet $\mathcal{A}$. On a alors :

- *La période de s est $|\mathcal{A}|^k$.*
- *Pour tout $t \leq k$, chaque mot de longueur t apparaît $|\mathcal{A}|^{k-t}$ fois dans une période de s. En particulier, s est équidistribuée à l'ordre k.*
- *Le nombre de* runs *de longueur m dans une période de s est :*

$$\begin{cases} |\mathcal{A}|^{k-m-1} \cdot (|\mathcal{A}|-1)^2 & si \quad m \leq k-2 \\ |\mathcal{A}| \cdot (|\mathcal{A}|-2) & si \quad m = k-1 \\ |\mathcal{A}| & si \quad m = k \\ 0 & si \quad m > k \end{cases}$$

On définit aussi les suites de De Bruijn poinçonnées :

Définition 2.2.6

Soit $\mathcal{A}$ un alphabet et $a \in \mathcal{A}$. Une suite s sur $\mathcal{A}$ est une ***suite de De Bruijn poinçonnée*** *de largeur k si chaque mot de longueur k apparaît exactement une fois sur une période de s, excepté le mot $(\underbrace{a, a, \dots, a}_{k \text{ fois}})$.*

On déduit de la proposition précédente :

Proposition 2.2.7 (*[GK09]*)

Soit s une suite de De Bruijn poinconnée de largeur k sur un alphabet $\mathcal{A}$. On a alors :

– *La période de s est $|\mathcal{A}|^k - 1$.*
– *Pour tout $t \leq k$, chaque mot de longueur t apparaît $|\mathcal{A}|^{k-t}$ fois dans une période de s, excepté le mot poinçonné qui n'apparaît jamais. En particulier, s est équidistribué à l'ordre k.*
– *Le nombre de* runs *de longueur m dans une période de s est :*

$$\begin{cases} |\mathcal{A}|^{k-m-1} \cdot (|\mathcal{A}| - 1)^2 & si \quad m \leq k-1 \\ |\mathcal{A}| - 1 & si \quad m = k \\ 0 & si \quad m > k \end{cases}$$

En particulier, s possède la propriété des runs.

2.3 Topologie I-adique

Dans cette section, nous présentons la construction des anneaux I-adiques. Ces anneaux se construisent par complétion d'un anneau $\mathbb{A}$ muni d'une métrique d_I définie par un idéal I.

2.3.1 Espace métrique et suites de Cauchy

Nous rappelons ici des résultats classiques sur les espaces métriques, ainsi que sur leurs complétions. Le lecteur pourra consulter [SZ91] pour plus de détails.

Définition 2.3.1

*On appelle **espace métrique** un ensemble E muni d'une fonction **distance**, i.e. une application d de $E \times E$ dans $\mathbb{R}_+$ vérifiant :*

Symétrie : $\forall x, y \in E,\ d(x,y) = d(y,x)$.

Séparation : $\forall x, y \in E,\ d(x,y) = 0 \Leftrightarrow x = y$.

Inégalité triangulaire : $\forall x, y, z \in E,\ d(x,z) \leq d(x,y) + d(y,z)$

*De plus, cette distance sera dite **ultra-métrique** si elle vérifie l'inégalité ultra-triangulaire :*

Inégalité ultra-triangulaire : $\forall x, y, z \in E,\ d(x,z) \leq \max(d(x,y), d(y,z))$

Un tel espace peut être muni d'une topologie en considérant les boules ouvertes comme base de voisinages.

On introduit maintenant le concept de suite convergente. Une suite convergente vers un élément $l \in E$ est un ensemble d'éléments de E tel que, à partir d'un certain rang, les termes de la suite sont aussi proches que l'on veut de l.

Définition 2.3.2

*Soient (E,d) un espace métrique et $(x_n)_{n\in\mathbb{N}}$ une suite d'éléments de E. On dit que cette suite est **convergente** vers $l \in E$ si :*

$$\forall \epsilon > 0, \exists N \in \mathbb{N}, \forall n \in \mathbb{N},\ (n \geq N \Rightarrow d(x_n, l) < \epsilon)$$

La définition précédente a pour inconvénient qu'il faut connaître a priori la limite de la suite. Pour pallier ce problème, on introduit les suites de Cauchy.

Définition 2.3.3

*Soient (E,d) un espace métrique et $(x_n)_{n\in\mathbb{N}}$ une suite d'éléments de E. On dit que cette suite est **de Cauchy** dans (E,d) si :*

$$\forall \epsilon > 0, \exists N \in \mathbb{N}, \forall n, m \in \mathbb{N},\ (n, m \geq N \Rightarrow d(x_n, x_m) < \epsilon)$$

En d'autre termes, une suite sera dite de Cauchy si, à partir d'un certain rang, les termes sont aussi proches que l'on veut les uns des autres.

Une suite convergente est naturellement de Cauchy, mais la réciproque n'est pas vraie. Sauf dans un espace complet :

Définition 2.3.4

Soit (E, d) un espace métrique. On dit que cet espace est ***complet*** *si toute suite de Cauchy est convergente dans E.*

On va maintenant construire à partir d'un espace métrique quelconque (E, d), un nouvel espace métrique (E', d') qui sera complet et qui contiendra E comme sous-espace dense.

Proposition 2.3.5

Soit (E, d) un espace métrique. Il existe un espace métrique complet (E', d') tel que E se plonge isométriquement dans E' et E est dense dans E'.

Démonstration : On rappelle ici les idées de la preuve : on note $\mathcal{C}$ l'ensemble des suites de Cauchy de (E, d). On considère la relation d'équivalence $\mathcal{R}$ définie sur $\mathcal{C}$:

$$(u_n)_{n\in\mathbb{N}}\mathcal{R}(v_n)_{n\in\mathbb{N}} \Leftrightarrow \lim_{n\to+\infty} d(u_n, v_n) = 0$$

On considère E' le quotient de $\mathcal{C}$ par $\mathcal{R}$. De plus, on munit E' de la distance d' suivante :

$$d'\left(\overline{U}, \overline{V}\right) = \lim_{n\to+\infty} d(u_n, v_n)$$

où $(u_n)_{n\in\mathbb{N}} \in \mathcal{C}$ (respectivement $(v_n)_{n\in\mathbb{N}} \in \mathcal{C}$) est un représentant de la classe $\overline{U}$ (respectivement $\overline{V}$). Alors (E', d') est un espace métrique vérifiant :

- E s'injecte isométriquement dans E' ;
- E est dense dans E' ;
- E' est complet. ■

Définition 2.3.6

Soit (E, d) un espace métrique. L'espace métrique (E', d') construit dans la proposition précédente est appelé ***complété*** *de (E, d).*

2.3.2 Topologie I-adique

Nous allons définir dans cette section une topologie sur un anneau à partir d'un idéal.

Nous donnons ici une définition métrique de cette topologie. La définition classique de la topologie I-adique se fonde sur une base de voisinages définie à partir de l'idéal I, ou plus précisément sur le filtre engendré par I. Puis on définit la métrique associée dans le cas où la topologie est séparée, *i.e.* $\bigcap_{n\in\mathbb{N}} I^n = \{0\}$.

Nous privilégions l'approche métrique car celle-ci permet de ne pas introduire de notions inutiles par la suite. Pour une approche alternative, le lecteur pourra lire [GK09].

Proposition 2.3.7

Soient $\mathbb{A}$ un anneau et I un idéal de $\mathbb{A}$ vérifiant $\bigcap_{n\in\mathbb{N}} I^n = \{0\}$. L'application de $\mathbb{A}\times\mathbb{A}$ dans $\mathbb{R}_+$

$$d_I(x, y) = \begin{cases} 2^{-k} \text{ où } k = \max\{n \in \mathbb{N},\ x - y \in I^n\} & si \quad x \neq y \\ 0 & si \quad x = y \end{cases}$$

est une distance ultra-métrique.

Remarque

Soient $x, y \in \mathbb{A}$. Alors $d_I(x,y) = 2^{-k}$ est équivalent à :

$$\forall n \in \mathbb{N}, \begin{cases} x - y \in I^n & \text{si} \quad n \leq k \\ x - y \notin I^n & \text{si} \quad n > k \end{cases}$$

car on a $I^n \supseteq I^{n+1}$.

Démonstration : Cette application vérifie clairement les axiomes de symétrie et de séparation. Montrons qu'elle est bien définie et qu'elle vérifie l'inégalité ultra-triangulaire :

Bien définie : Soit $x, y \in \mathbb{A}$.

- Si $x = y$, l'application est clairement bien définie.
- Si $x \neq y$, on a $x - y \notin \{0\}$. Or puisque $\mathbb{A}$ est séparé, on a $\{0\} = \bigcap_{n\in\mathbb{N}} I^n$. Donc il existe $k \in \mathbb{N}$ tel que $x - y \in I^k$ (car $I^0 = \mathbb{A}$) et $x - y \notin I^{k+1}$. De plus, k est unique car pour tout $n \in \mathbb{N}$, $I^n \supseteq I^{n+1}$.

Inégalité ultra-triangulaire : Soient $x, y, z \in \mathbb{A}$ distincts. On note $d_I(x,z) = 2^{-k}$. Supposons

$$d_I(x,y) \leq 2^{-(k+1)} \text{ et } d_I(y,z) \leq 2^{-(k+1)}, \textit{ i.e. } x - y, y - z \in I^{k+1}.$$

Alors $(x-y)+(y-z) = x-z \in I^{k+1}$. Contradiction car $d_I(x,z) = 2^{-k}$. Donc $d_I(x,y) \geq 2^{-k}$ ou $d_I(y,z) \geq 2^{-k}$. D'où $\max(d_I(x,y), d_I(y,z)) \geq 2^{-k} = d_I(x,z)$. ■

Définition 2.3.8

Soient $\mathbb{A}$ un anneau et I un idéal de $\mathbb{A}$ tel que $\bigcap_{n\in\mathbb{N}} I^n = \{0\}$. On appelle ***topologie I-adique*** *la topologie induite sur l'espace métrique $(\mathbb{A}, d_I)$.*
De plus, on appelle anneau I-adique sur $\mathbb{A}$ (ou simplement anneau I-adique) le complété de $(\mathbb{A}, d_I)$. Cet anneau est noté $\mathbb{A}_I$. Un élément de $\mathbb{A}_I$ sera appelé un entier I-adique.

2.3.3 Premières propriétés

L'ensemble des entiers I-adiques étant construit par complétion de $(\mathbb{A}, d_I)$, nous donnons d'abord une propriété sur l'équivalence de distances I-adiques :

Proposition 2.3.9

Soient $\mathbb{A}$ un anneau, I et I' deux idéaux de $\mathbb{A}$ tels que $\bigcap_{n\in\mathbb{N}} I^n = \{0\} = \bigcap_{n\in\mathbb{N}} I'^n$. Supposons de plus qu'il existe $k \in \mathbb{N}$ tel que $I^k \subseteq I'$. Alors,

1. *Toute suite de Cauchy pour d_I est aussi de Cauchy pour $d_{I'}$;*
2. *$\mathbb{A}_I$ est contenu dans $\mathbb{A}_{I'}$.*

Démonstration :

1. Soit $(a_n)_{n\in\mathbb{N}}$ de Cauchy pour d_I. Soit $\epsilon > 0$. On considère $q \in \mathbb{N}$ tel que $2^{-q} \leq \epsilon$. $(a_n)_{n\in\mathbb{N}}$ étant de Cauchy pour d_I, il existe N_I tel que pour tout $n, m \geq N_I$, on a $d_I(a_n, a_m) \leq 2^{-kq}$. En particulier, $a_n - a_m \in I^{kq}$. Or $I^k \subseteq I'$, donc $a_n - a_m \in I'^q$, d'où $d_{I'}(a_n, a_m) \leq 2^{-q} \leq \epsilon$. Donc $(a_n)_{n\in\mathbb{N}}$ est de Cauchy pour $d_{I'}$.
2. Puisque $\mathbb{A}_I$ est le complété de $\mathbb{A}$ pour d_I, pour tout élément a de $\mathbb{A}_I$, il existe une suite de Cauchy $(a_n)_{n\in\mathbb{N}}$ de $\mathbb{A}$ pour d_I tel que $a_n \longrightarrow a$. Or on vient de voir qu'alors $(a_n)_{n\in\mathbb{N}}$ est aussi de Cauchy pour $d_{I'}$, donc $(a_n)_{n\in\mathbb{N}}$ converge dans $\mathbb{A}_{I'}$, le complété de $\mathbb{A}$ pour $d_{I'}$. Donc $\mathbb{A}_I$ se projette dans $\mathbb{A}_{I'}$. ■

Corollaire 2.3.10

Soient $\mathbb{A}$ un anneau, I et I' deux idéaux de $\mathbb{A}$ tels que $\bigcap_{n\in\mathbb{N}} I^n = \{0\} = \bigcap_{n\in\mathbb{N}} I'^n$. Supposons de plus qu'il existe $k, k' \in \mathbb{N}$ tel que $I^k = I'^{k'}$. Alors, être de Cauchy pour d_I est équivalent à être de Cauchy pour $d_{I'}$. En particulier, $\mathbb{A}_I$ et $\mathbb{A}_{I'}$ sont isomorphes.

Démonstration : Il suffit de remarquer que $I^k = I'^{k'}$ implique $I^k \subseteq I'$ et $I'^{k'} \subseteq I$ car $I^k \subseteq I$ et $I'^{k'} \subseteq I'$. Il suffit alors d'appliquer la proposition précédente. ■

2.4 Topologie π-adique

On considère maintenant le cas où I est principal, engendré par π. On note alors $\mathbb{A}_\pi$ l'anneau π-adique sur $\mathbb{A}$ plutôt que $\mathbb{A}_{\langle\pi\rangle}$. Dans ce cas, les éléments de $\mathbb{A}_\pi$ s'écrivent, grâce au théorème de Hensel, sous la forme de séries en π à coefficients dans $\mathbb{A}$.

2.4.1 Développement de Hensel

Théorème 2.4.1 (*Développement de Hensel*)

Soit $\mathbb{A}_\pi$ un anneau π-adique. En particulier, π est un élément de $\mathbb{A}^*$ tel que $\bigcap_{i\in\mathbb{N}} \langle\pi^i\rangle = \{0\}$.
On considère $\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$ dans $\mathbb{A}$, i.e. :
- *$\mathcal{S} \subset \mathbb{A}$;*
- *Pour tout $a \in \mathbb{A}$, il existe $s \in \mathcal{S}$ tel que $a \equiv s[\pi]$;*
- *Pour tout $s \neq s' \in \mathcal{S}$, $s \not\equiv s'[\pi]$.*

Alors tout élément $a \in \mathbb{A}_\pi$ s'écrit de façon unique :

$$a = \sum_{i\in\mathbb{N}} \alpha_i \pi^i \text{ avec } \alpha_i \in \mathcal{S}$$

La suite $(\alpha_i)_{i\in\mathbb{N}}$ sera noté $\mathrm{seq}_{\mathcal{S}}(a)$, ou plus simplement $\mathrm{seq}(a)$ lorsqu'il n'y a pas d'ambiguïté.

Démonstration : Existence : Soit $a \in \mathbb{A}_\pi$. Puisque $\mathbb{A}$ est dense dans $\mathbb{A}_\pi$, on considère $(a_i)_{i\in\mathbb{N}}$ une suite de Cauchy de $\mathbb{A}$ convergente vers a. On va construire les α_i par récurrence. L'hypothèse de récurrence en rang k est :

$$\exists \alpha_0, \dots, \alpha_k \in \mathcal{S}, \exists n_0 \leq \cdots \leq n_k \in \mathbb{N},\ \forall 0 \leq i \leq k, \forall n \geq n_i,\ d\left(a_n, \sum_{0\leq j\leq i} \alpha_j \pi^j\right) \leq 2^{-(i+1)}$$

Cas $k = 0$: Soit n_0 tel que pour tout $n \geq n_0$, $d(a_n, a_{n_0}) \leq 2^{-1}$. Soit $\alpha_0 \in \mathcal{S}$ tel que :

$$\alpha_0 \equiv a_{n_0}[\pi]$$

En particulier, on a $d(a_{n_0}, \alpha_0) \leq 2^{-1}$. D'où, pour tout $n \geq n_0$:

$$d(a_n, \alpha_0) \leq \max(d(a_n, a_{n_0}), d(a_{n_0}, \alpha_0)) \leq 2^{-1}$$

Cas $k = 1$: Soit $n_1 \geq n_0$ tel que pour tout $n \geq n_1$, $d(a_n, a_{n_1}) \leq 2^{-2}$. Puisque $n_1 \geq n_0$, on a :

$$d(a_{n_1}, \alpha_0) \leq 2^{-1}, \text{ i.e. } a_{n_1} \equiv \alpha_0[\pi]$$

Soit $\alpha_1 \in \mathcal{S}$ tel que :

$$\alpha_1 \equiv \frac{a_{n_1} - \alpha_0}{\pi}[\pi]$$

On a alors $a_{n_1} \equiv \alpha_0 + \alpha_1\pi[\pi^2]$, d'où :

$$d(a_n, \alpha_0 + \alpha_1\pi) \leq \max(d(a_n, a_{n_1}), d(a_{n_1}, \alpha_0 + \alpha_1\pi)) \leq 2^{-2}$$

Supposons l'hypothèse vraie au rang $k-1$: Soit $n_k \geq n_{k-1}$ tel que pour tout $n \geq n_k$, $d(a_n, a_{n_k}) \leq 2^{-(k+1)}$. Puisque $n_k \geq n_{k-1}$, on a :

$$a_{n_k} \equiv \sum_{0\leq j\leq k-1} \alpha_j\pi^j[\pi^k]$$

Soit $\alpha_k \in \mathcal{S}$ tel que $\alpha_k \equiv \dfrac{a_{n_k} - \sum_{0\leq j\leq k-1}\alpha_j\pi^j}{\pi^k}[\pi]$. On a alors :

$$d\left(a_n, \sum_{0\leq j\leq k}\alpha_j\pi^j\right) \leq \max\left(d\left(a_n, a_{n_k}\right), d\left(a_{n_k}, \sum_{0\leq j\leq k}\alpha_j\pi^j\right)\right) \leq 2^{-(k+1)}$$

On considère alors la suite $\alpha = \left(\sum_{0\leq i\leq n}\alpha_i\pi^i\right)_{n\in\mathbb{N}}$. Cette suite est clairement de Cauchy, car on a :

$$d\left(\sum_{0\leq i\leq n}\alpha_i\pi^i, \sum_{0\leq i\leq n+1}\alpha_i\pi^i\right) \leq 2^{-(n+1)}$$

De plus, $d\left(a_n, \sum_{0\leq i\leq n}\alpha_i\pi^i\right)$ tend vers 0 par construction. Donc α est la limite de $(a_n)_{n\in\mathbb{N}}$, *i.e.* $\alpha = a$.

Unicité : Soient $\sum_{i\in\mathbb{N}}\alpha_i\pi^i = \sum_{i\in\mathbb{N}}\alpha'_i\pi^i$ deux écritures d'un élément de $\mathbb{A}_\pi$ avec $\alpha_i, \alpha'_i \in \mathcal{S}$. Alors on a $\alpha_0 \equiv \alpha'_0[\pi]$, d'où $\alpha_0 = \alpha'_0$ car $\alpha_0, \alpha'_0 \in \mathcal{S}$. On en déduit, puisque $\mathbb{A}$ est intègre, $\sum_{i\geq 1}\alpha_i\pi^{i-1} = \sum_{i\geq 1}\alpha'_i\pi^{i-1}$. Ainsi, de proche en proche, on obtient $\alpha_i = \alpha'_i$. ■

Définition 2.4.2

Soient $\mathbb{A}_\pi$ un anneau π-adique, $\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$ dans $\mathbb{A}$. Pour tout $a = \sum_{i\in\mathbb{N}}\alpha_i\pi^i \in \mathbb{A}_\pi$ avec $\alpha_i \in \mathcal{S}$, on note :

$$\begin{cases} a \bmod \pi := \alpha_0 \\ a \operatorname{div} \pi := \sum_{i\in\mathbb{N}}\alpha_{i+1}\pi^i \end{cases}$$

Remarque

Concrètement :
- $a \bmod \pi$ est l'unique élément de $\mathcal{S}$ congru à a modulo π ;
- $a \operatorname{div} \pi$ est l'élément $\dfrac{a - (a \bmod \pi)}{\pi}$.

Plus précisément, $a \bmod \pi$ et $a \operatorname{div} \pi$ sont uniquement définis par la relation

$$a = (a \bmod \pi) + \pi(a \operatorname{div} \pi)$$

avec $(a \bmod \pi) \in \mathcal{S}$.

Le développement de Hensel d'un élément $a \in \mathbb{A}_\pi$ peut alors se calculer avec l'algorithme I.2.

Algorithme I.2 : Calcul du développement de Hensel dans $\mathbb{A}_\pi$

Données : $\mathbb{A}_\pi$ un anneau π-adique, $\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$ dans $\mathbb{A}$
Entrées : $a \in \mathbb{A}_\pi$
Sorties : $\mathrm{seq}_{\mathcal{S}}(a) = (\alpha_i)_{i\in\mathbb{N}}$
début
  $a_0 \leftarrow a$;
  pour $i = 0$ **à** ∞ **faire**
    $\alpha_i \leftarrow a_i \bmod \pi$;
    $a_{i+1} \leftarrow a_i \operatorname{div} \pi$;
  retourner $(\alpha_i)_{i\in\mathbb{N}}$;
fin

Démonstration de l'algorithme I.2 : On va montrer par récurrence sur n l'invariant $a = \sum_{i=0}^{n-1} \alpha_i \pi^i + a_n \pi^n$:

Cas $n = 0$: Clair.

Supposons la propriété vraie au rang n. Alors :

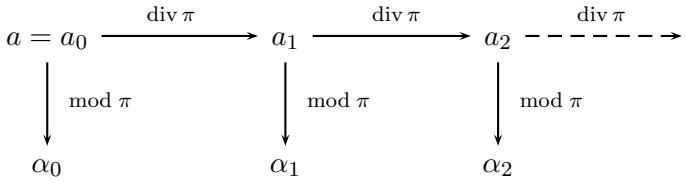
$$a = \sum_{i=0}^{n-1} \alpha_i \pi^i + a_n \pi^n = \sum_{i=0}^{n-1} \alpha_i \pi^i + (\alpha_n + \pi a_{n+1})\pi^n = \sum_{i=0}^{n} \alpha_i \pi^i + a_{n+1}\pi^{n+1}$$

La propriété est donc vraie au rang $n+1$.

Par unicité du développement de Hensel, on a donc $\mathrm{seq}_{\mathcal{S}}(a) = (\alpha_i)_{i\in\mathbb{N}}$. ■

Remarque

L'algorithme I.2 implémente le diagramme suivant :



Cet algorithme a un fonctionnement analogue à un générateur pseudo-aléatoire (définition 1.3.1) :
- Les états σ_i sont ici les $a_i \in \mathbb{A}_\pi$;
- L'alphabet de sortie est $\mathcal{S}$;
- La fonction de transition δ est ici $(\cdot \operatorname{div} \pi)$;
- La fonction d'extraction ω est ici $(\cdot \bmod \pi)$.

2.4.2 Exemples

Nous considérons ici des exemples classiques d'anneaux π-adiques.

Cas $\mathbb{A} = \mathbb{F}_2[X]$, $\pi = X$:

Soit $\mathbb{A}_0$ l'anneau π-adique construit avec $\mathbb{A} = \mathbb{F}_2[X]$ et $\pi = X$. Dans ce cas, le quotient $\mathbb{A}/\pi$ est isomorphe à $\mathbb{F}_2$. En considérant donc $\mathcal{S}_0 = \{0, 1\}$ comme système de représentants, on a facilement que $\mathbb{A}_X = \mathbb{F}_2[[X]]$, *i.e.* la topologie X-adique construit l'anneau des séries formelles. En effet, $\mathbb{A}_X = \left\{ \sum_{i\in\mathbb{N}} a_i X^i,\ a_i \in \mathcal{S}_0 \right\}$ par le théorème 2.4.1.

Considérons un second système de représentants $\mathcal{S}_1 = \{X^3, X+1\}$. On donne dans la table 2.1 les séquences associés à des éléments de $\mathbb{A}_X$ pour $\mathcal{S}_0$ et $\mathcal{S}_1$.

x	$\mathrm{seq}_{\mathcal{S}_0}(x)$	$\mathrm{seq}_{\mathcal{S}_1}(x)$
0	$(\underline{0})$	$(\underline{X^3, X^3, X^3, X+1, X^3, X+1, X+1})$
1	$(\underline{1})$	$(\underline{X+1, X^3, X+1, X^3, X^3, X+1, X+1})$
$\frac{X^3}{X+1}$	$(0,0,0,\underline{1})$	$(\underline{X^3})$
1	$(1,\underline{0})$	$(\underline{X+1})$

TABLE 2.1 – Représentations d'éléments de $\mathbb{A}_X$ avec $\mathcal{S}_0$ et $\mathcal{S}_1$

Cas $\mathbb{A} = \mathbb{F}_2[X]$, $\pi = X^3$:

Soit $\mathbb{A}_1$ l'anneau π-adique construit avec $\mathbb{A} = \mathbb{F}_2[X]$ et $\pi = X^3$. Ici, le quotient $\mathbb{A}/\pi$ est isomorphe à l'ensemble des polynômes sur $\mathbb{F}_2$ de degré strictement inférieur à 3. On choisit donc comme système de représentants $\mathcal{S} = \left\{a_0 + a_1 X + a_2 X^2,\ a_0, a_1, a_2 \in \{0,1\}\right\}$.

De plus, par le corollaire 2.3.10, $\mathbb{A}_0$ et $\mathbb{A}_1$ sont isomorphes. En effet :

– Un élément de $\mathbb{A}_0$ s'écrit $\sum_{i\in\mathbb{N}} a_i X^i$ avec $a_i \in \{0,1\}$.

– Un élément de $\mathbb{A}_1$ s'écrit $\sum_{i\in\mathbb{N}} a'_i X^{3\cdot i}$ avec $a'_i \in \left\{0, 1, X, X+1, X^2, X^2+1, X^2+X, X^2+X+1\right\}$.

Cas $\mathbb{A} = \mathbb{Z}$, $\pi = 2$:

Soit $\mathbb{A}_2$ l'anneau π-adique construit avec $\mathbb{A} = \mathbb{Z}$ et $\pi = 2$. Le quotient $\mathbb{A}/\pi$ est isomorphe à $\mathbb{F}_2$. On considère dons $\mathcal{S} = \{0,1\}$. Les éléments de $\mathbb{A}_2$ s'écrivent alors $\sum_{i\in\mathbb{N}} a_i 2^i$ avec $a_i \in \mathcal{S}$.

Cet anneau est l'anneau des entiers 2-adiques. Une autre construction de cet anneau est possible est considérant la valeur absolue 2-adique sur $\mathbb{Q}$ et en construisant le complété de $\mathbb{Q}$ associé à cette métrique.

Cas $\mathbb{A} = \mathbb{Z}$, $\pi = 10$:

Soit $\mathbb{A}_3$ l'anneau π-adique construit avec $\mathbb{A} = \mathbb{Z}$ et $\pi = 10$. Le quotient $\mathbb{A}/\pi$ est $\mathbb{Z}/10\mathbb{Z}$. On considère $\mathcal{S} = \{0, 1, \ldots, 9\}$. Les éléments de $\mathbb{A}_3$ s'écrivent alors $\sum_{i\in\mathbb{N}} a_i 10^i$ avec $a_i \in \mathcal{S}$.

Contrairement à l'exemple $\mathbb{A}_2$, cet anneau ne peut pas être construit via la définition d'une « valeur absolue 10-adique » sur $\mathbb{Q}$ car de telles valeurs absolues n'existent que pour les nombres premiers par le théorème d'Ostrowski.

2.4.3 Premières propriétés

Nous donnons ici des propriétés que nous utiliserons par la suite :

– une propriété donnant une écriture rationnelle aux éléments ayant un développement de Hensel périodique,

– une propriété caractérisant les éléments inversibles de $\mathbb{A}_\pi$,

– une propriété pour construire un système de représentants de $\mathbb{A}/\pi^n$ à partir d'un système de représentants de $\mathbb{A}/\pi$.

Proposition 2.4.3

Soit $\mathbb{A}_\pi$ un anneau π-adique. Pour tout $T \in \mathbb{N}$, on a l'égalité suivante :

$$\sum_{i=0}^{\infty} \pi^{i\cdot T} = \frac{-1}{\pi^T - 1}$$

Remarque

La relation $\sum_{i=0}^{\infty} \pi^{i\cdot T} = \frac{-1}{\pi^T-1}$ ne signifie pas $\mathrm{seq}_{\mathcal{S}}\left(\frac{-1}{\pi^T-1}\right) = (\underbrace{1, 0, \ldots, 0}_{T-1 \text{ fois}})$ car 0 et 1 peuvent ne pas appartenir à $\mathcal{S}$.

Démonstration : Clair grâce à la relation suivante :

$$(\pi^T - 1)\cdot\left(\sum_{i=0}^{\infty}\pi^{i\cdot T}\right) = \left(\sum_{i=0}^{\infty}\pi^{(i+1)\cdot T}\right) - \left(\sum_{i=0}^{\infty}\pi^{i\cdot T}\right) = \left(\sum_{i=1}^{\infty}\pi^{i\cdot T}\right) - \left(1 + \sum_{i=1}^{\infty}\pi^{i\cdot T}\right) = -1$$

■

Proposition 2.4.4

Soit $\mathbb{A}_\pi$ un anneau π-adique. Un élément $q \in \mathbb{A}$ est inversible dans $\mathbb{A}_\pi$ si et seulement si q et π sont premiers entre eux.

Démonstration : Par définition, q et π sont premiers entre eux si $\langle q\rangle + \langle\pi\rangle = \mathbb{A}$. Ce qui est équivalent au fait que q soit inversible dans $\mathbb{A}/\pi$. Avec cette caractérisation, on a :

$\Rightarrow$: Supposons q inversible dans $\mathbb{A}_\pi$, il existe donc $u \in \mathbb{A}_\pi$ tel que $qu = 1$. En réduisant modulo π cette relation, on a q inversible dans $\mathbb{A}_\pi/\pi \simeq \mathbb{A}/\pi$. q et π sont donc premiers entre eux.

$\Leftarrow$: Soit $\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$. On pose $q = \sum_{i=0}^{\infty} q_i\pi^i$ avec $q_i \in \mathcal{S}$. On cherche $u = \sum_{i=0}^{\infty} u_i\pi^i$ avec $u_i \in \mathcal{S}$ tels que $qu = 1$, *i.e.*

$$\begin{cases} q_0u_0 \equiv 1[\pi] \\ q_0u_n + q_1u_{n-1} + \cdots + q_nu_0 \equiv 0[\pi] \text{ pour } n \geq 1 \end{cases}$$

Ce système d'équations se résout par récurrence sur u_n en utilisant le fait que q_0 est inversible dans $\mathbb{A}/\pi$. ■

Proposition 2.4.5

Soit $\mathbb{A}_\pi$ un anneau π-adique. Soient $\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$, $n \in \mathbb{N}$. Alors, $\mathcal{S} + \pi\mathcal{S} + \cdots + \pi^{n-1}\mathcal{S}$ est un système de représentants de $\mathbb{A}/\pi^n$.

Démonstration : Ce résultat est direct en utilisant le développement de Hensel.

– Tout d'abord, tout élément de $\mathbb{A}$ est dans la classe d'un élément de $\mathcal{S} + \pi\mathcal{S} + \cdots + \pi^{n-1}\mathcal{S}$ modulo π^n. En effet, soit $a \in \mathbb{A}$. Puisque $\mathbb{A} \subset \mathbb{A}_\pi$, soit $\mathrm{seq}_{\mathcal{S}}(a) = (\alpha_i)_{i\in\mathbb{N}}$. En particulier, on a :

$$\pi^n \left| a - \sum_{i=0}^{n-1} \alpha_i\pi^i \right.$$

Donc $a \equiv \sum_{i=0}^{n-1} \alpha_i\pi^i [\pi^n]$, avec $\sum_{i=0}^{n-1} \alpha_i\pi^i \in \mathcal{S} + \cdots + \pi^{n-1}\mathcal{S}$.

– De plus, les éléments de $\mathcal{S} + \pi\mathcal{S} + \cdots + \pi^{n-1}\mathcal{S}$ sont tous distincts. En effet, supposons $\sum_{i=0}^{n-1} \alpha_i\pi^i = \sum_{i=0}^{n-1} \beta_i\pi^i$ avec $\alpha_i, \beta_i \in \mathcal{S}$. Alors, $\alpha_0 \equiv \beta_0[\pi]$, donc $\alpha_0 = \beta_0$ car $\mathcal{S}$ est un système de représentants de $\mathbb{A}/\pi$. Par le même raisonnement, de proche en proche, on a que $\alpha_i = \beta_i$ pour $i = 0, \ldots, n-1$. ■

2.4.4 Limites de l'approche théorique

Nous avons présenté la construction des anneaux π-adiques. Cette construction est générique à partir d'un anneau $\mathbb{A}$ et d'un élément π, mais peut conduire à des anneaux avec des propriétés très distinctes. Par exemple, étant donné p premier, on peut construire :

- l'anneau des séries formelles sur le corps à p éléments $\mathbb{F}_p[[X]]$, avec $\mathbb{A} = \mathbb{F}_p[X]$ et $\pi = X$.
- l'anneau des entiers p-adiques $\mathbb{Z}_p$, avec $\mathbb{A} = \mathbb{Z}$ et $\pi = p$.

Dans les deux cas, en choisissant $\mathcal{S} = \{0, \dots, p-1\}$ comme système de représentants, à chaque élément de $\mathbb{A}_\pi$ est associé une suite de $\mathcal{S}$. Cependant :

- dans $\mathbb{F}_p[[X]]$, les opérations sont sans propagation de retenues :

$$\forall a, b \in \mathbb{F}_p[[X]],\ \mathrm{seq}_{\mathcal{S}}(a+b) = (a_i + b_i)_{i\in\mathbb{N}} \text{ avec } \mathrm{seq}_{\mathcal{S}}(a) = (a_i)_{i\in\mathbb{N}}, \mathrm{seq}_{\mathcal{S}}(b) = (b_i)_{i\in\mathbb{N}}$$

- tandis que dans $\mathbb{Z}_p$, ce n'est pas le cas :

$$\mathrm{seq}_{\mathcal{S}}(p) = (0, 1, \underline{0}) \text{ avec } \mathrm{seq}_{\mathcal{S}}(1) = (1, \underline{0}), \mathrm{seq}_{\mathcal{S}}(p-1) = (p-1, \underline{0})$$

De même, l'approche générique recouvrant des anneaux très différents, on ne peut pas caractériser les développements de Hensel des éléments de $\mathbb{A}_\pi$. Plus précisément, on ne peut même pas caractériser les développements de Hensel des éléments de $\mathbb{A}$:

- Avec $\mathbb{A} = \mathbb{Z}$, $\pi = p$ et $\mathcal{S} = \{0, \dots, p-1\}$. Les éléments a de $\mathbb{A}$ ont un développement de Hensel fini (si $a \geq 0$) ou périodique (si $a < 0$).
- Avec $\mathbb{A} = \mathbb{Z}[X]$, $\pi = X$ et $\mathcal{S} = \{0, i - (i+1)X,\ \forall i \in \mathbb{Z}^*\}$. Certains éléments de $\mathbb{A}$ ont un développement de Hensel non-périodique. Par exemple :

$$1 = \sum_{i\in\mathbb{N}^*} (i - (i+1)X)X^{i-1}$$

On peut même construire un cas « pathologique » : un anneau π-adique construit de deux façons différentes. Avec la première méthode, les éléments de $\mathbb{A}$ ont une écriture finie ; tandis qu'avec la seconde méthode, certains éléments de $\mathbb{A}$ ont une écriture non-périodique.

- Avec $\mathbb{A} = \mathbb{F}_p[X]$, $\pi = X$ et $\mathcal{S} = \{0, \dots, p-1\}$, on a $\mathbb{A}_\pi = \mathbb{F}_p[[X]]$. Les éléments de $\mathbb{A}$ ont un développement de Hensel fini.
- Avec $\mathbb{A} = \mathbb{F}_p[[X]]$, $\pi = X$ et $\mathcal{S} = \{0, \dots, p-1\}$, on a $\mathbb{A}_\pi = \mathbb{F}_p[[X]]$. Certains éléments de $\mathbb{A}$ ont un développement de Hensel non-périodique. Par exemple :

$$\sum_{i\in\mathbb{N}} X^{2^i}$$

Toutes ces considérations justifient le fait que l'on spécifie les anneaux π-adiques que l'on construit pour pouvoir exploiter des propriétés plus profondes.

2.5 Anneau des séries formelles sur un corps fini : $\mathbb{F}_q[[X]]$

Dans cette section, nous étudions l'anneau des séries formelles sur un corps fini $\mathbb{F}_q$. Cet anneau, et en particulier ses séquences périodiques, a été étudié dans [GW82, Sto73, GK09].

Nous avons déjà vu que cet anneau peut être construit comme un anneau π-adique avec $\mathbb{A} = \mathbb{F}_q[X]$ et $\pi = X$. Nous choisissons comme système de représentants de $\mathbb{A}/\pi$ $\mathcal{S} = \{0, \dots, q-1\}$.

La première propriété de cet anneau permet de caractériser les séquences périodiques.

Théorème 2.5.1 (*[GW82],[Sto73] et [GK09]*)

Soient $S(X) \in \mathbb{F}_q[[X]]$ et $T \in \mathbb{N}$. Les assertions suivantes sont équivalentes :

1. $\mathrm{seq}(S(X))$ *est* T*-périodique.*
2. *Il existe* $A(X) \in \mathbb{F}_q[X]$ *tel que* $S(X) = A(X)/\left(X^T - 1\right)$.
3. *Il existe* $P(X), Q(X) \in \mathbb{F}_q[X]$*, avec* $Q(X) \left| \left(X^T - 1\right)\right.$*, tels que* $S(X) = P(X)/Q(X)$.

De plus, si T *est la période de* $\mathrm{seq}(S(X))$*, ces assertions sont équivalentes à :*

4. *Il existe* $P(X), Q(X) \in \mathbb{F}_q[X]$*, avec* $PGCD(P(X), Q(X)) = 1$ *et l'ordre de* X *dans* $\mathbb{F}_q[X]/Q(X)$ *est* T*, tels que* $S(X) = P(X)/Q(X)$.

Les suites périodiques sont donc liées aux éléments inversibles de $\mathbb{F}_q[[X]]$. Par la proposition 2.4.4, $Q(X)$ est inversible dans $\mathbb{F}_q[[X]]$ si et seulement si $Q(0)$ est inversible dans $\mathbb{F}_q$, *i.e.* $Q(0) \neq 0$. De plus, on peut caractériser les suites strictement périodiques :

Théorème 2.5.2 (*[GW82]*)

Soient $P(X), Q(X) \in \mathbb{F}_q[X]$ *avec* $Q(0) \neq 0$*. Alors :*
- *Si* $\deg P < \deg Q$, $\mathrm{seq}(P(X)/Q(X))$ *est strictement périodique.*
- *Si* $\deg P \geq \deg Q$, $\mathrm{seq}(P(X)/Q(X))$ *possède une pré-période de longueur* $\deg P - \deg Q$.

On introduit les définitions suivantes, en accord avec le point 4 du théorème 2.5.1, pour caractériser les séquences de période maximale :

Définition 2.5.3

Soit $Q(X) \in \mathbb{F}_q[X]$ *de degré* n.
- *Si* X *est primitif dans* $\mathbb{F}_q[X]/Q(X)$, *i.e. l'ordre de* X *est* $q^n - 1$ *dans* $\mathbb{F}_q[X]/Q(X)$*, on dira que* $Q(X)$ *est* ***primitif****.*
- *Étant donné* $Q(X)$ *primitif, les séquences* $\mathrm{seq}(P(X)/Q(X))$*, avec* $P(X) \in \mathbb{F}_q[X]$ *et* $\deg(P(X)) < n$*, seront appelées* m**-sequences**.

Les *m-sequences*, en plus d'avoir une période maximale, ont de bonnes propriétés statistiques :
- Elles remplissent les postulats de Golomb.
- Ce sont des suites de De Bruijn poinçonnées.

Nous donnons une expression du développement de Hensel d'un élément $P(X)/Q(X)$ avec $\deg P < \deg Q =: n$ et $Q(0) \neq 0$. On choisit comme système de représentants de $\mathbb{F}_q[X]/Q(X)$ l'ensemble des polynômes de degré strictement inférieur à n. Alors :

$$\mathrm{seq}\left(\frac{P(X)}{Q(X)}\right) = \Big(\big(P(X) \cdot X^{-i} \bmod Q(X)\big) \bmod X\Big)_{i \in \mathbb{N}}$$

i.e. le i-ème terme de la suite se calcule ainsi :
- On considère le polynôme $R(X)$ de degré strictement inférieur à n congru à $P(X) \cdot X^{-i}$ modulo $Q(X)$;
- $R(0)$ est le i-ème terme du développement de Hensel.

Cette approche, appelée représentation exponentielle des séquences dans [GK09], permet de calculer le i-ème terme du développement de Hensel d'un quotient, sans calculer les précédents.

2.6 Anneau des entiers N-adiques : $\mathbb{Z}_N$

Dans cette section, nous étudions l'anneau des entiers N-adiques. Cet anneau, et en particulier ses séquences périodiques, a été étudié dans [KG97, GK09, Kob97].

Nous avons déjà vu que cet anneau peut être construit comme un anneau π-adique avec $\mathbb{A} = \mathbb{Z}$ et $\pi = N$. Nous choisissons comme système de représentants de $\mathbb{A}/\pi$ $\mathcal{S} = \{0, \ldots, N-1\}$.

La première propriété de cet anneau permet de caractériser les séquences périodiques.

Théorème 2.6.1 ([Kob97, KG97])

Soient $s \in \mathbb{Z}_N$ et $T \in \mathbb{N}$. Les assertions suivantes sont équivalentes :

1. $\text{seq}(s)$ *est T-périodique.*
2. *Il existe $a \in \mathbb{Z}$ tel que $s = a/\left(N^T - 1\right)$.*
3. *Il existe $b, c \in \mathbb{Z}$, avec $c \left| \left(N^T - 1\right)\right.$, tels que $s = b/c$.*

De plus, si T est la période de $\text{seq}(s)$, ces assertions sont équivalentes à :

4. *Il existe $b, c \in \mathbb{Z}$, avec $PGCD(b, c) = 1$ et l'ordre de N dans $\mathbb{Z}/c\mathbb{Z}$ est T, tels que $s = b/c$.*

De plus, on peut caractériser les suites strictement périodiques :

Proposition 2.6.2 ([GK09])

Soient $p, q \in \mathbb{Z}_N$ avec q inversible dans $\mathbb{Z}/N\mathbb{Z}$. Alors, si $pq < 0$ et $0 < |p| < |q|$, $\text{seq}(p/q)$ est strictement périodique.

On introduit les définitions suivantes, en accord avec le point 4 du théorème précédent, pour caractériser les séquences de période maximale :

Définition 2.6.3

Soit $q \in \mathbb{N}$.

- *Si N est primitif dans $\mathbb{Z}/q$, i.e. l'ordre de N est $\varphi(q)$ dans $\mathbb{Z}/q\mathbb{Z}$, on dira que q est* ***primitif****.*
- *Étant donné q primitif, les séquences $\text{seq}(a/q)$, avec $aq < 0$ et $0 < |a| < |q|$, seront appelées* ***l*-sequences**.

Pour que l'ordre de N modulo q soit $\varphi(q)$, il faut que q soit la puissance d'un premier $q = p^e$. En cryptographie, nous sommes intéressés par les séquences de période maximale. Cependant, si $q = p^e$ avec $e > 1$, les *l-sequences* ne seront pas toutes de même période. En effet, si $PGCD(a, q) > 1$, la période de la séquence est égale à l'ordre de 2 modulo q divisé par $PGCD(a, q)$:

$$\frac{\text{ord}_q(2)}{PGCD(a,q)} < \text{ord}_q(2)$$

C'est pourquoi, il est plus intéressant de considérer les *l-sequences* avec q premier.

Les *l-sequences* ont de bonnes propriétés statistiques. En particulier, ce sont « presque » des suites de De Bruijn :

Proposition 2.6.4 ([GK09])

Soit $q = p^t$, avec $p > 2$ premier, primitif. Soit s une l-sequence associée à q. Alors, le nombre $N(b)$ d'occurrences de chaque mot b de longueur k sur une période de s vérifie :

$$\begin{array}{rcll} \left\lfloor \frac{q}{N^k} \right\rfloor & \leq N(b) \leq & \left\lfloor \frac{q}{N^k} \right\rfloor + 1 & si \quad t = 1 \\ \left\lfloor \frac{q}{N^k} \right\rfloor - \left\lfloor \frac{p^{t-1}}{N^k} \right\rfloor - 1 & \leq N(b) \leq & \left\lfloor \frac{q}{N^k} \right\rfloor - \left\lfloor \frac{p^{t-1}}{N^k} \right\rfloor + 1 & si \quad t > 1 \end{array}$$

Nous donnons une expression du développement de Hensel d'un élément p/q avec $0 < p < -q$. On choisit comme système de représentants de $\mathbb{Z}/q\mathbb{Z}$ l'ensemble des entiers strictement inférieurs à $-q$. Alors :

$$\text{seq}\left(\frac{p}{q}\right) = \left(\left(p \cdot N^{-i} \bmod q\right) \bmod N\right)_{i \in \mathbb{N}}$$

i.e. le i-ème terme de la suite se calcule ainsi :

– On considère l'entier r strictement inférieur à $-q$ congru à $p \cdot N^{-i}$ modulo q ;
– r modulo N est le i-ème terme du développement de Hensel.

Comme pour les séries formelles, cette approche permet de calculer le i-ème terme du développement de Hensel d'un quotient, sans calculer les précédents.

Chapitre 3

Automates

Sommaire

3.1 Modèles de Mealy et de Moore

Nous présentons ici la théorie des automates, au sens de Moore et Mealy [Moo56, Mea55]. Ces approches sont deux modèles pour décrire des automates avec entrées et sorties. Ces modèles, fruits de recherches distinctes, ne diffèrent que sur la façon de construire les sorties des automates.

3.1.1 Modèle de Moore

Définition 3.1.1 (*[Moo56]*)

Une machine déterministe dans le ***modèle de Moore*** *est la donnée d'un sextuplet* $(\Sigma, \mathcal{I}, \mathcal{O}, \sigma_0, \delta, \omega)$ *:*

- Σ *: un ensemble d'états ;*
- $\mathcal{I}$ *: un ensemble fini, appelé alphabet des mots d'entrée ;*
- $\mathcal{O}$ *: un ensemble fini, appelé alphabet des mots de sortie ;*
- σ_0 *dans* Σ *: un état initial ;*
- δ *: une fonction de transition de* $\mathcal{I} \times \Sigma$ *dans* Σ *;*

– ω : *une fonction d'extraction de* Σ *dans* $\mathcal{O}$.

Étant donné le mot $(i_1 \dots i_n)$ *en entrée de l'automate, le mot produit par l'automate est :*

$$(\omega(\sigma_0) \dots \omega(\sigma_n)) \text{ avec } \sigma_t = \delta(i_t, \sigma_{t-1}),\ 1 \leq t \leq n$$

Une machine de Moore peut être représentée sous la forme d'un graphe orienté étiqueté où :
– chaque sommet représente un état,
– il existe une arête étiquetée i du sommet σ vers le sommet σ' si $\delta(i, \sigma) = \sigma'$.

De plus, l'état initial est représenté avec une flèche entrante, et, pour chaque état σ, l'élément $\omega(\sigma)$ est écrit sous le sommet σ.

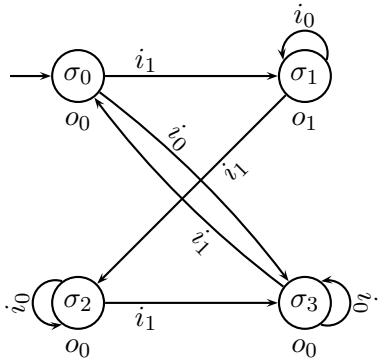
Une machine de Moore peut aussi être décrite en utilisant un tableau liant chaque état σ avec la sortie $\omega(\sigma)$ et les différentes transitions $\delta(\cdot, \sigma)$.

Par exemple, la figure 3.1 présente les deux représentations de la machine de Moore suivante :

$$(\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}, \mathcal{I} = \{i_0, i_1\}, \mathcal{O} = \{o_0, o_1\}, \sigma_0, \delta, \omega)$$

avec

$$\begin{cases} \delta(i_0, \sigma_0) = \sigma_3 \\ \delta(i_1, \sigma_0) = \sigma_1 \\ \delta(i_0, \sigma_1) = \sigma_1 \\ \delta(i_1, \sigma_1) = \sigma_2 \\ \delta(i_0, \sigma_2) = \sigma_2 \\ \delta(i_1, \sigma_2) = \sigma_3 \\ \delta(i_0, \sigma_3) = \sigma_3 \\ \delta(i_1, \sigma_3) = \sigma_0 \end{cases} \text{ et } \begin{cases} \omega(\sigma_0) = o_0 \\ \omega(\sigma_1) = o_1 \\ \omega(\sigma_2) = o_0 \\ \omega(\sigma_3) = o_0 \end{cases}$$



(a) Diagramme de transition

État actuel s	Transition $\delta(i_0, s)$	Transition $\delta(i_1, s)$	Sortie $\omega(s)$
σ_0	σ_3	σ_1	o_0
σ_1	σ_1	σ_2	o_1
σ_2	σ_2	σ_3	o_0
σ_3	σ_3	σ_0	o_0

(b) Tableau de transition

FIGURE 3.1 – Exemple de machine de Moore

3.1.2 Modèle de Mealy

Définition 3.1.2 (*[Mea55]*)

Une machine déterministe dans le ***modèle de Mealy*** *est la donnée d'un sextuplet* $(\Sigma, \mathcal{I}, \mathcal{O}, \sigma_0, \delta, \omega)$ *:*
– Σ *: un ensemble d'états ;*
– $\mathcal{I}$ *: un ensemble fini, appelé alphabet des mots d'entrée ;*
– $\mathcal{O}$ *: un ensemble fini, appelé alphabet des mots de sortie ;*
– σ_0 *dans* Σ *: un état initial ;*

– δ : *une fonction de transition de* $\mathcal{I} \times \Sigma$ *dans* Σ ;
– ω : *une fonction d'extraction de* $\mathcal{I} \times \Sigma$ *dans* $\mathcal{O}$.
Étant donné le mot $(i_0 \dots i_n)$ *en entrée de l'automate, le mot produit par l'automate est :*

$$(\omega(i_0, \sigma_0) \dots \omega(i_n, \sigma_n)) \text{ avec } \sigma_t = \delta(i_{t-1}, \sigma_{t-1}),\ 1 \leq t \leq n$$

Comme pour les machines de Moore, les machines de Mealy peuvent être représentées sous la forme d'un graphe orienté étiqueté où :
– chaque sommet représente un état,
– il existe une arête étiquetée $i/\omega(i, \sigma)$ du sommet σ vers le sommet σ' si $\delta(i, \sigma) = \sigma'$.
De plus, l'état initial est représenté avec une flèche entrante.

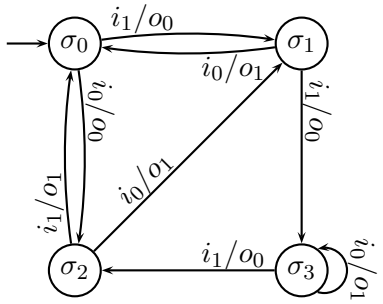
Une machine de Mealy peut aussi être décrite en utilisant un tableau liant chaque état σ avec les différentes sorties $\omega(\cdot, \sigma)$ et les différentes transitions $\delta(\cdot, \sigma)$.

Par exemple, la figure 3.2 présente les deux représentations de la machine Mealy suivante :

$$(\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}, \mathcal{I} = \{i_0, i_1\}, \mathcal{O} = \{o_0, o_1\}, \sigma_0, \delta, \omega)$$

avec

$$\begin{cases} \delta(i_0, \sigma_0) = \sigma_2 \\ \delta(i_1, \sigma_0) = \sigma_1 \\ \delta(i_0, \sigma_1) = \sigma_0 \\ \delta(i_1, \sigma_1) = \sigma_3 \\ \delta(i_0, \sigma_2) = \sigma_1 \\ \delta(i_1, \sigma_2) = \sigma_0 \\ \delta(i_0, \sigma_3) = \sigma_3 \\ \delta(i_1, \sigma_3) = \sigma_2 \end{cases} \text{ et } \begin{cases} \omega(i_0, \sigma_0) = o_0 \\ \omega(i_1, \sigma_0) = o_0 \\ \omega(i_0, \sigma_1) = o_1 \\ \omega(i_1, \sigma_1) = o_0 \\ \omega(i_0, \sigma_2) = o_1 \\ \omega(i_1, \sigma_2) = o_1 \\ \omega(i_0, \sigma_3) = o_1 \\ \omega(i_1, \sigma_3) = o_0 \end{cases}$$



(a) Diagramme de transition

État actuel	Transition		Sortie	
s	$\delta(i_0, s)$	$\delta(i_1, s)$	$\omega(o_0, s)$	$\omega(o_1, s)$
σ_0	σ_2	σ_1	o_0	o_0
σ_1	σ_0	σ_3	o_1	o_0
σ_2	σ_1	σ_0	o_1	o_1
σ_3	σ_3	σ_2	o_1	o_0

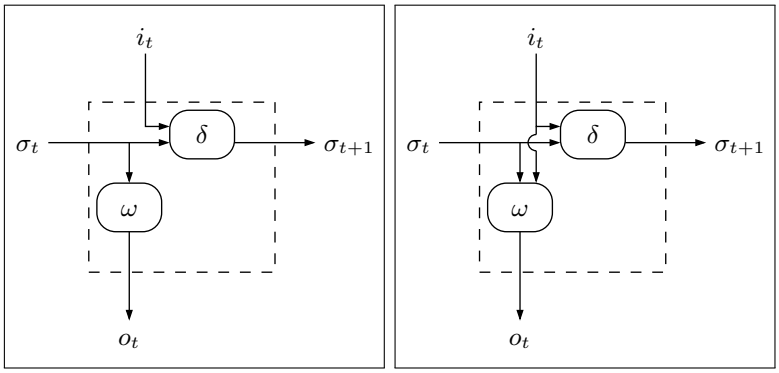
(b) Tableau de transition

FIGURE 3.2 – Exemple de machine de Mealy

3.1.3 Comparaison des deux modèles

La différence entre ces deux modèles est que le symbole émis au temps t dépend :
– uniquement de l'état au temps t pour Moore.
– de l'état au temps t et du symbole reçu à l'instant t pour Mealy.
En particulier, les machines de Moore sont un cas particulier des machines de Mealy. Celles-ci permettant de décrire une plus grande variété de machines, nous les préférerons aux machines de Moore. De plus, le fait que la sortie à l'instant t dépend de l'entrée à l'instant t permet un traitement sans délai.

La figure 3.3 présente schématiquement les modèles de Moore et de Mealy.



(a) Moore (b) Mealy

FIGURE 3.3 – Fonctionnement des machines de Moore et de Mealy

3.1.4 Automates linéaires

Nous définissons ici les automates linéaires ([Kag06] dans le cas des LFSMs). Ceux-ci utilisent des transformations « linéaires » pour leurs fonctions de transition et d'extraction. Plus précisément, l'ensemble des états Σ, l'alphabet d'entrée $\mathcal{I}$ et l'alphabet de sortie $\mathcal{O}$ étant munis de structures de groupe, les applications δ et ω sont des morphismes de groupe, *i.e.* , dans le cas d'une machine de Moore et en notant additivement les opérations dans les groupes Σ, $\mathcal{I}$, $\mathcal{O}$, on a, pour tout $i, i' \in \mathcal{I}, \sigma, \sigma' \in \Sigma$:

$$\begin{aligned} \delta(i+i', \sigma+\sigma') &= \delta(i,\sigma) + \delta(i',\sigma') \\ \omega(\sigma+\sigma') &= \omega(\sigma) + \omega(\sigma') \end{aligned}$$

Définition 3.1.3

Soient Σ un groupe, $\mathcal{I}$, $\mathcal{O}$ deux groupes finis. On note additivement les opérations. Soit $(\Sigma, \mathcal{I}, \mathcal{O}, \sigma_0, \delta, \omega)$ une machine de Moore (respectivement de Mealy). Elle est dite ***linéaire*** *si :*

- *δ est un morphisme de groupes de $\mathcal{I} \times \Sigma$ dans Σ,*
- *ω est un morphisme de groupes de Σ (respectivement $\mathcal{I} \times \Sigma$) dans $\mathcal{O}$.*

En particulier, étant donné un anneau $\mathbb{A}$, une machine de Moore linéaire peut être construite ainsi :

- L'ensemble des états est $\mathbb{A}^n$;
- L'alphabet d'entrée est $\mathbb{A}^l$;
- L'alphabet de sortie est $\mathbb{A}^k$;
- $\delta(i, \sigma)$ est défini par $A \cdot \sigma + B \cdot i$ où A et B sont des matrices dans $\mathbb{A}$ de tailles respectives $n \times n$ et $n \times l$;
- $\omega(\sigma)$ est défini par $C \cdot \sigma$ où C est une matrice dans $\mathbb{A}$ de taille $k \times n$.

De même, une machine de Mealy peut être construite de la même manière, et en définissant $\omega(i, \sigma)$ par $C \cdot \sigma + D \cdot i$ où C et D sont des matrices dans $\mathbb{A}$ de tailles respectives $k \times n$ et $k \times l$.

Dans ce cas, la différence entre le modèle de Moore et celui de Mealy est uniquement porté par la matrice D : si elle est nulle, on est dans le modèle de Moore ; sinon c'est le modèle de Mealy.

3.1.5 Automates finis

Le concept d'automates présenté jusqu'ici est formel : certains automates ne peuvent être construits physiquement. La principale contrainte pour pouvoir implémenter les automates est

la mémoire requise : celle-ci doit être finie. La première contrainte pour construire un automate est qu'il ait un nombre fini d'états. Ainsi il ne nécessitera qu'une quantité de mémoire finie pour stocker son état :

Définition 3.1.4

Soit $(\Sigma, \mathcal{I}, \mathcal{O}, \sigma_0, \delta, \omega)$ une machine de Moore (respectivement de Mealy). Elle est dite ***à états finis*** *si Σ est fini.*

3.1.6 Automates autonomes

Les automates dans les modèles de Moore et de Mealy sont des machines traitant des entrées, possédant un état interne, et produisant des sorties. Lorsqu'un automate ne traite aucune entrée, *i.e.* $\mathcal{I} = \varnothing$, on dira qu'il est **autonome**. En particulier, les générateurs pseudo-aléatoires définis dans la définition 1.3.1 sont en fait des automates sans entrées.

3.2 Aspect matériel

Cette section traite des différentes contraintes liées aux implémentations matérielles des automates. Ces problèmes classiques sont généralement traités en cours d'électronique numérique. Le lecteur peut lire [DOJ] pour une approche plus détaillée. Nous présentons les circuits synchrones. Ces circuits sont caractérisés par trois éléments :

- Un **signal d'horloge** : ce signal peut être considéré comme le métronome du circuit. C'est un signal binaire oscillant régulièrement. Les éléments du circuit réagissent lorsque ce signal présente un front montant. La période du signal est appelée **cycle d'horloge**.
- Des **éléments mémorisants** : chacun de ces éléments possède un état et agit ainsi :
  - Durant un cycle d'horloge, elle émet son état.
  - À l'apparition d'un front d'horloge, son état est mis à jour grâce à ses entrées.
- Des **opérateurs combinatoires** : ceux-ci traitent les signaux binaires durant chaque cycle d'horloge.

Nous présentons les portes logiques, éléments fondamentaux des circuits électroniques. Elles permettent de construire la partie combinatoire des automates. Puis nous présentons les bascules D qui sont les éléments mémorisants de base.

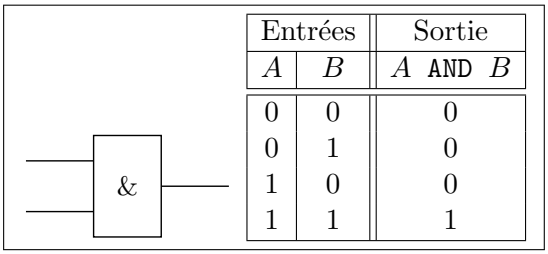
3.2.1 Portes logiques

Les portes logiques sont les éléments de base pour implémenter les fonctions logiques en électronique. Ce sont des circuits simples qui traitent des données binaires. Elles servent à implémenter l'algèbre de Boole ; les trois principales sont `ET`, `NON` et `OU exclusif`. On utilisera les notations anglaises `AND`, `NOT` et `XOR`. La figure 3.4 présente leur symbole (norme européenne) ainsi que leur table de vérité.

Ces portes servent à implémenter les opérations classiques de l'algèbre de Boole (addition, multiplication, complément). Plus précisément :

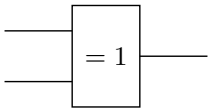
- l'addition dans l'algèbre de Boole correspond à une porte `XOR` : A `XOR` $B = A \oplus B$.
- la multiplication dans l'algèbre de Boole correspond à une porte `AND` : A `AND` $B = A \cdot B$.
- le complément dans l'algèbre de Boole correspond à une porte `NOT` : `NOT` $A = \overline{A}$

L'algèbre de Boole est un outil puissant qui permet une approche mathématique des circuits électroniques. En particulier, l'algèbre de Boole permet, à partir d'une table de vérité donnée, de construire un circuit correspondant. Nous présentons ici cette propriété pour synthétiser un circuit réalisant l'addition de trois bits. La table 3.1 présente la table de vérité de ce circuit.



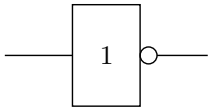
Entrées		Sortie
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

(a) ET



Entrées		Sortie
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

(b) OU exclusif



Entrée	Sortie
A	NOT A
0	1
1	0

(c) NON

FIGURE 3.4 – Symboles et tables de vérité des portes logiques

Entrées			Sorties	
A	B	C	R	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

TABLE 3.1 – Table de vérité d'un additionneur 3 bits

Une méthode pour calculer une relation booléenne entre les entrées et les sorties se base sur l'interpolation de fonctions. Pour cela, on se base (dans le cas de fonctions de trois variables) sur les fonctions suivantes, appelées fonctions **indicatrices** :

$$f_{a,b,c}(A,B,C) := (A \oplus \overline{a}) \cdot (B \oplus \overline{b}) \cdot (C \oplus \overline{c}) \text{ avec } a,b,c \in \{0,1\}$$

Celles-ci valent 1 en (a,b,c), et 0 sinon.

Par exemple ici, on considère la sortie R. Pour construire la fonction $R(A,B,C)$, on écrit la relation suivante en utilisant la table de vérité :

$$R = (0 \cdot f_{0,0,0}) \oplus (0 \cdot f_{0,0,1}) \oplus (0 \cdot f_{0,1,0}) \oplus (1 \cdot f_{0,1,1}) \oplus (0 \cdot f_{1,0,0}) \oplus (1 \cdot f_{1,0,1}) \oplus (1 \cdot f_{1,1,0}) \oplus (1 \cdot f_{1,1,1})$$

Après simplification, on obtient l'expression $R(A,B,C) = (A \cdot B) \oplus (B \cdot C) \oplus (A \cdot C)$. Ce qui s'écrit encore :

$$R(A,B,C) = (A \cdot B) \oplus C \cdot (A \oplus B)$$

Cette dernière expression nécessite une opération de moins, et donc une porte logique de moins lors de l'implémentation.

De même, en suivant cette méthode, on obtient $S(A,B,C) = A \oplus B \oplus C$.

La figure 3.5 présente une implémentation possible d'un additionneur trois bits.

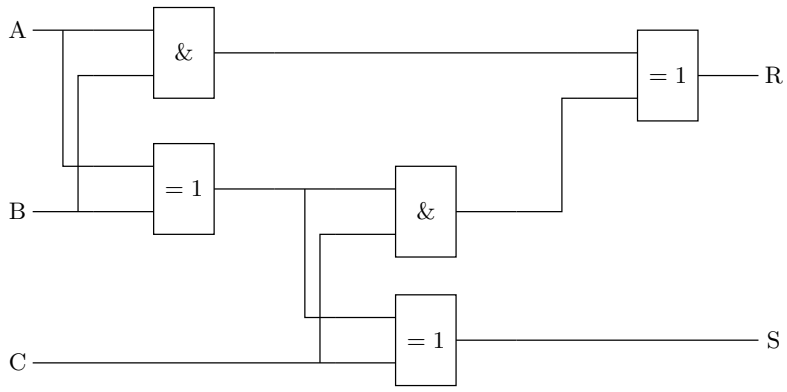
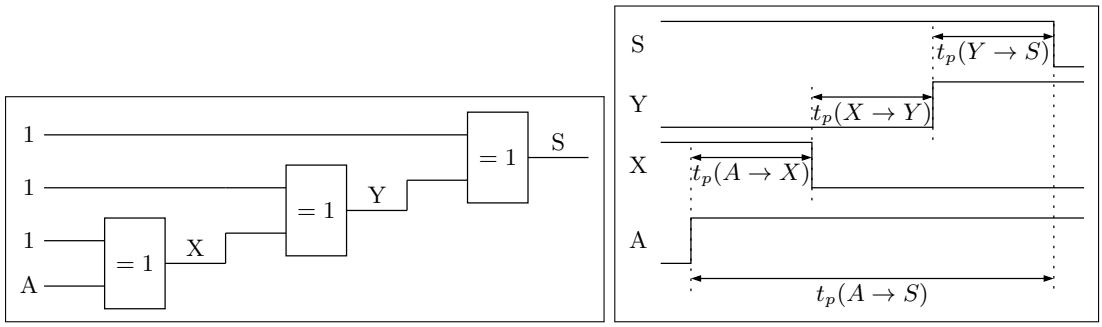


FIGURE 3.5 – Implémentation d'un additionneur 3 bits

3.2.2 Première contrainte liée aux circuits logiques

Les circuits logiques sont construits en utilisant des portes logiques. Celles-ci sont construites en utilisant des composants électroniques classiques (résistances, transistors, condensateurs, etc). En conséquence, une porte logique ne réagit pas instantanément : il existe un délai entre l'application d'une valeur en entrée et l'apparition du résultat en sortie. Ce délai entre l'entrée A et la sortie S est appelé temps de propagation de A à S et est noté $t_p(A \to S)$. La figure 3.6 présente un circuit ainsi que son chronogramme.



(a) Circuit (b) Chronogramme

FIGURE 3.6 – Exemples de temps de propagation dans un circuit logique.

3.2.3 Bascules D

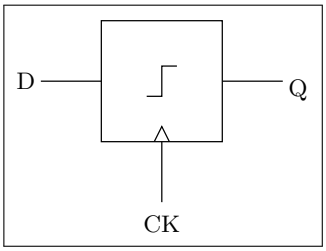
Une bascule D permet de stocker un bit Q_0 de façon synchrone. Plus précisément, une bascule D a :

- un état Q_0 : un bit ;
- un signal D en entrée : sa valeur détermine le prochain bit stocké par la bascule ;
- un signal CK d'horloge en entrée : lorsque ce signal présente un front montant, l'état de la bascule est mis à jour grâce au signal D ;
- un signal Q en sortie : ce signal est égal au bit stocké par la bascule.

Autrement dit, avec le point de vue de la programmation impérative, le comportement d'une bascule D peut être décrit ainsi :

- Si $CK = 0$ ou $CK = 1$, alors $Q \leftarrow Q_0$.
- Si CK présente un front montant (noté $\lrcorner\!\!\urcorner$), alors $Q_0 \leftarrow D$.

La figure 3.7 contient la représentation symbolique d'une bascule D, ainsi que sa table de transition.



(a) Représentation symbolique

D	CK	Q
X	0	Q_0
X	1	Q_0
0	⎍	0
1	⎍	1

(b) Table de transition

FIGURE 3.7 – Représentation et table de transition d'une bascule D

Comme précédemment, les bascules D étant des circuits logiques, il y a un délai entre les entrées et les sorties. Pour que la bascule change d'état sur un front d'horloge, il faut que le signal d'entrée soit stable avant (**temps de prépositionnement** ou ***setup time*** t_{su}) et après (**temps de maintien** ou ***hold time*** t_h) le front. De plus, la sortie changera après le temps d'accès t_a. La figure 3.8 présente les caractéristiques temporelles d'une bascule D.

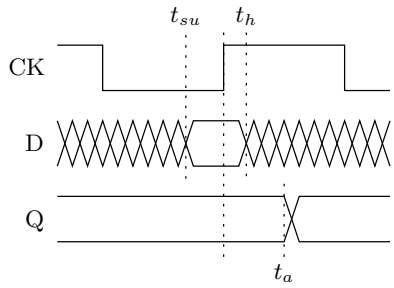


FIGURE 3.8 – Chronogramme d'une bascule D

Dans un circuit synchrone, les bascules D sont connectées ensemble. Ainsi, le signal Q d'une bascule est, après passage dans un bloc combinatoire, un signal D d'une autre bascule. De plus, on a la relation $t_h \leq t_a$. Ainsi, le signal D est en pratique stable pendant t_a après le front.

3.2.4 Seconde contrainte liée aux circuits synchrones

Nous rappelons qu'un circuit synchrone est composé d'éléments mémorisants synchronisés par une horloge, et de blocs combinatoires qui traitent les données mémorisées pour les mettre à jour. La figure 3.9 présente une portion d'un circuit synchrone.

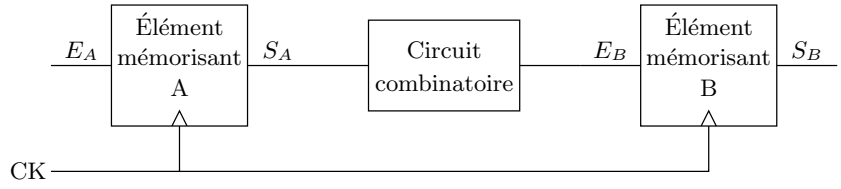


FIGURE 3.9 – Schéma d'un circuit synchrone

De nouvelles contraintes apparaissent sur le temps des signaux. Pour que l'élément A ait l'effet voulu sur l'élément B, il faut que :

– Le signal S_A soit valide, *i.e.* $t_a(A)$ doit s'être écoulé depuis le front montant,

- Le signal S_A doit être traité par le circuit combinatoire. Cela nécessite un temps égal à $t_p(S_A \to E_B)$,
- Le signal E_B doit être maintenu pendant $t_{su}(B)$ avant le front montant pour que la valeur soit stockée.

Ainsi, la période de l'horloge T doit vérifiée

$$T \geq t_a(A) + t_p(S_A \to E_B) + t_{su}(B) \tag{3.1}$$

Cette relation permet de déterminer la fréquence de fonctionnement maximale d'un circuit.

3.2.5 Critères

Étant donné un automate, différentes implémentations sont possibles. Nous donnons ici trois critères caractéristiques d'un circuit, permettant ainsi de choisir la meilleure implémentation.

- Tout d'abord, le circuit doit avoir un coût minimal, *i.e.* posséder le moins de portes logiques possibles. On choisira donc l'implémentation qui en utilise le moins.
- Il doit avoir une fréquence d'horloge élevée, permettant ainsi des débits élevés. Par la relation 3.1, la fréquence d'un circuit dépend des délais d'écritures des mémoires et du temps de propagation des blocs combinatoires. On considère que les mémoires utilisées sont identiques dans tous nos circuits. Pour augmenter la fréquence, il nous faut donc diminuer le temps de propagation des blocs combinatoires.
  Pour mesurer ce paramètre, on définit le **chemin critique** d'un circuit comme étant le chemin de propagation qui limite sa fréquence de fonctionnement, *i.e.* c'est le chemin comportant le plus de portes logiques entre deux éléments mémorisants. On l'exprime en nombre de portes logiques.
- Enfin, un critère lié à la construction du circuit : le ***fan-out*** (arité sortante). Le *fan-out* d'un port de sortie est le nombre de blocs connecté à ce port. Cette valeur doit être la plus basse possible, sinon on devra dupliquer des portes pour implémenter le circuit.

3.2.6 Automates de Moore et de Mealy

Nous donnons ici la représentation classique des automates de Moore et de Mealy d'un point de vue matériel (figure 3.10). L'état de l'automate est stocké grâce à un élément mémorisant, tandis que les fonctions de transition et d'extraction sont implémentées par des opérateurs combinatoires.

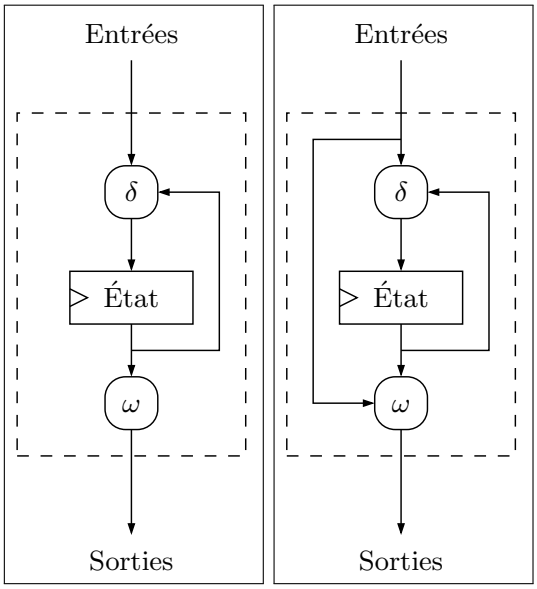
3.3 Aspect logiciel

Alors que l'implémentation matérielle d'un automate consiste à construire physiquement la machine désirée, l'implémentation logicielle consiste à exécuter une série d'instructions grâce à une machine déjà existante : le processeur. Nous présentons d'abord rapidement l'architecture d'un processeur, pour ensuite décrire les spécificités de la programmation logicielle.

3.3.1 Processeur

Un processeur est un circuit électronique complexe implémentant :

des registres : Ce sont des mémoires de taille identique : souvent 32 bits, mais éventuellement 8, 16, 64, 128 ou 256 bits. Les registres sont les mémoires les plus rapidement accessibles : ils ne nécessitent pas de cycle d'horloge supplémentaire pour être chargés. Cependant, ils sont peu nombreux (généralement 8 ou 16).



(a) Automate de Moore (b) Automate de Mealy

FIGURE 3.10 – Vision matériel des automates de Moore et de Mealy

un jeu d'instructions : un ensemble d'opérations que le processeur peut effectuer comme « charger une valeur », « additionner », « stocker en mémoire », etc. Elles s'appliquent à des valeurs qui sont soit stockées dans un registre, soit stockées à une adresse mémoire, soit des constantes. Les instructions « de base » présentes dans la plupart des processeurs sont :

Déplacement : Lecture, écriture.

Calcul : Addition, soustraction, opération booléenne (`AND`, `XOR`, `NOT`, décalage, etc.).

Comparaison : Égal, inférieur, supérieur, nul.

Modification de l'exécution : Saut dans le programme, revenir d'un saut.

Il existe différents jeux d'instructions, certains avec très peu d'instructions (le processeur est alors moins cher à produire), d'autres très fournis. De plus, les jeux d'instructions sont régulièrement enrichis en fonction des nouveaux besoins, par exemple le jeu d'instruction x86, à l'origine constitué de moins de cent instructions sur huit registres de 16 bits, est maintenant, avec les évolutions MMX, x64, SSE, SSE2, SSE3, SSE4, SSE5, constitué de plus de 200 instructions agissant sur seize registres de 64 bits et seize registres de 128 bits.

Un programme est alors une suite d'instructions que le processeur va exécuter de façon séquentielle. Le programme peut être écrit :

- dans un langage de bas niveau (comme l'assembleur) : directement avec le jeu d'instructions du processeur. Le programme est alors très rapide lors de l'exécution. Cependant, utiliser un tel langage nécessite une grande connaissance de l'architecture cible, beaucoup de temps de programmation et le programme ne sera pas portable (*i.e.* il ne fonctionnera que sur les processeurs possédant ce jeu d'instructions).
- dans un langage de plus haut niveau (comme le C) : avec un niveau d'abstraction qui permet d'ignorer de gérer les registres, la mémoire, etc. Le code sera alors traduit dans le jeu d'instructions du processeur par un compilateur. Le programme est alors plus facile à écrire et est portable. Mais il est généralement plus lent qu'un programme écrit dans un langage de bas niveau.

Nous choisissons d'écrire nos programmes en C pour une plus grande compréhension. De plus, les compilateurs pour langage C sont très performants pour optimiser le programme final : l'exécution est quasiment aussi rapide qu'avec un programme en assembleur.

3.3.2 Mémoire cache

Il existe plusieurs types de mémoires utilisées en informatique :

Les registres : ils sont intégrés au processeur et leur nombre et leur taille dépendent du jeu d'instructions du processeur.

La mémoire cache : elle aussi est intégrée au processeur. Elle permet d'accélérer l'exécution des programmes, car elle stocke les données nécessaires au programme (les données à proprement parler et les instructions à venir). Sa taille (de l'ordre du mégaoctet) dépend du processeur et résulte d'un compromis coût/performance.

La mémoire centrale couramment appelée mémoire vive : elle fait plusieurs gigaoctets. Elle est plus lente que la mémoire cache.

La mémoire de masse : elle atteint maintenant le téraoctet. Elle est constitué des disques durs, DVDs, etc.

Celles-ci sont ordonnées selon la **hiérarchie mémoire** présentée dans la figure 3.11

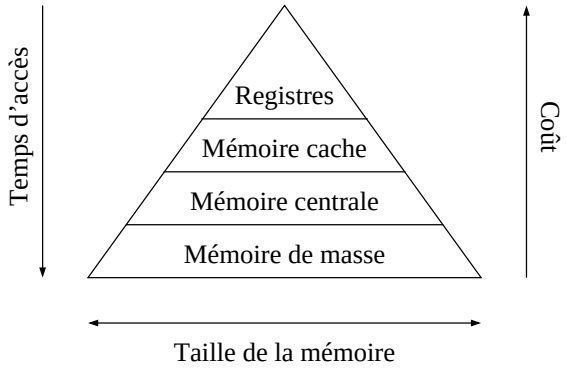


FIGURE 3.11 – Hiérarchie mémoire en informatique

Cette hiérarchie existe pour que le processeur accède le plus rapidement possible aux données les plus utilisées. Ainsi, le programme actuellement exécuté est copié (tout ou partie) dans la mémoire cache. La mémoire centrale stocke les données plus grosses nécessaires au programme, comme une base de données par exemple. Enfin la mémoire de masse stocke les données encore plus grosses, et les données actuellement non-nécessaires au programme.

Ainsi, pour accélérer l'exécution d'un programme, il faut que celui-ci puisse tenir dans la mémoire cache, pour cela il faut qu'il soit le plus petit possible. De plus, il ne doit pas nécessiter de gros volumes de données, pour que celles-ci soient également dans le cache. En pratique, un programme (avec ces données) doit avoir une taille de moins d'un mégaoctet pour être exécuter entièrement depuis la mémoire cache. Cependant, ceci n'est pas une règle stricte, puisque la taille de la mémoire cache dépend du processeur ; et que l'espace libre de celle-ci dépend aussi des autres applications « simultanément » exécutées (avec un système d'exploitation multitâche, un processeur exécute plusieurs tâches concourantes).

3.3.3 Contraintes

La programmation logicielle a deux contraintes essentielles :

– Les accès à la mémoire étant lents, on doit avoir le moins d'accès possible à la mémoire. On utilisera en priorité les registres, ainsi que le cache. Pour cela, le programme devra être petit, et traiter un petit nombre de données simultanément.
– Les instructions étant implémentées pour s'appliquer sur des valeurs de 8, 16, 32, 64, 128 ou 256 bits en fonction du processeur, on manipulera des données de cette taille.

Deuxième partie

Automates algébriques

Introduction

Dans cette partie, nous utilisons les anneaux π-adiques (section 2.4) et les automates linéaires (définition 3.1.3) pour construire des automates :

- respectant la structure algébrique des anneaux π-adiques,
- avec des entrées et des sorties,
- définis par des matrices.

Ces automates seront dits algébriques.

- Lorsque l'anneau π-adique considéré est l'anneau des séries formelles sur un anneau, les automates algébriques sont déjà connus : les LFSRs.
- Lorsque l'anneau π-adique considéré est l'anneau des entiers N-adiques, on sait construire des automates respectant cette structure : les FCSRs.
- Lorsque l'anneau π-adique n'est pas spécifié, on sait construire des automates respectant cette structure : les AFSRs.

Nous conserverons ces noms pour les automates algébriques que nous définirons.

Nous appliquons d'abord cette approche aux AFSRs (chapitre 5), puis aux cas particuliers des LFSRs (chapitre 6) et des FCSRs (chapitre 7). Dans ces deux derniers cas, nous étudierons en particulier les implémentations possibles liées à l'utilisation de matrices pour définir les automates. Enfin, dans les chapitres 8 et 9, nous appliquons ces résultats dans le cas des FCSRs pour construire des chiffrements à flot.

Chapitre 4

État de l'art

Sommaire

Nous présentons ici l'état de l'art sur les **LFSRs** (*Linear Feedback Shift Registers*), les **FCSRs** (*Feedback with Carry Shift Registers*) et les **AFSRs** (*Algebraic Feedback Shift Registers*). Les LFSRs et les FCSRs sont des cas particuliers d'AFSRs, cependant nous les présentons dans cet ordre qui est l'ordre chronologique : les LFSRs [GW82] (première publication en 1967), puis les FCSRs [KG93a], enfin les AFSRs [KX99].

En conséquence, les connaissances sur ces différents automates ne sont pas le mêmes. Dans les trois cas, on sait caractériser les séquences produites en utilisant l'anneau π-adique adéquat. Pour le reste :

- Dans le cas des LFSRs, on sait donner une description matricielle des automates. Cela permet, en particulier, de construire des automates avec de bonnes implémentations matérielles ou logicielles. De plus, on a une compréhension totale des LFSRs avec des entrées.
- Dans le cas des FCSRs, il existe deux types d'automates : les FCSRs en mode Galois, et les FCSRs en mode Fibonacci. Ceux-ci s'implémentent aisément de façon matérielle. En outre, ces deux familles d'automates sont sans entrées.
- Dans le cas des AFSRs, il n'existe qu'un type d'automate dont la structure est analogue aux LFSRs et FCSRs en mode Fibonacci.

4.1 LFSR / LFSM

Comme on le verra au chapitre 6, les registres à décalage linéaire (LFSR) sont des AFSRs sur $(\mathbb{A}[X], X, \mathbb{A})$. Cependant, ces registres étant connus depuis longtemps ([GW82, Sto73]), ils

ont été très étudiés, et ce, sans utiliser la théorie des AFSRs, et ce, par différentes communautés (allant des mathématiques formelles [Sto73] à l'électronique [MRT04]). On va présenter ici les résultats sur les LFSRs, mais aussi sur les LFSMs qui en sont une généralisation.

4.1.1 Définition

La littérature étant très fournie sur les LFSRs, il existe différentes approches possibles. Nous choisissons de présenter une approche basée sur les automates linéaires (définition 3.1.3).

Une LFSM sur un anneau $\mathbb{A}$ est un automate dont l'ensemble des états est $\mathbb{A}^n$, ayant une fonction de transition linéaire, et en entrée des éléments de $\mathbb{A}^k$ et produisant des éléments de $\mathbb{A}^l$. On peut donner une description matricielle d'un tel automate :

Définition 4.1.1

*Une **LFSM** (*Linear Finite State Machine*) avec k entrées, l sorties et de longueur n sur un anneau $\mathbb{A}$ est un quadruplet de matrices (A, B, C, D) sur $\mathbb{A}$ de tailles $n \times n$, $n \times k$, $l \times n$, $l \times k$ respectivement. Étant donné l'état à l'instant t $m(t) \in \mathcal{M}_{n,1}(\mathbb{A})$ et le vecteur en entrée $u(t) \in \mathcal{M}_{k,1}(\mathbb{A})$. L'état à l'instant $t+1$ $m(t+1) \in \mathcal{M}_{n,1}(\mathbb{A})$ et le vecteur sortie $v(t) \in \mathcal{M}_{l,1}(\mathbb{A})$ sont calculés par :*

$$\begin{cases} m(t+1) &= Am(t) + Bu(t) \\ v(t) &= Cm(t) + Du(t) \end{cases}$$

*La matrice A est appelée **matrice de transition** de l'automate. De plus, quand $B = 0$ et $D = 0$, la LFSM est dit **autonome**.*

Comme vu précédemment, lorsque $D = 0$, la LFSM est une machine de Moore ; sinon c'est une machine de Mealy.

4.1.2 Séquences produites

On peut caractériser les séquences produites en utilisant le théorème donné dans [GW82]. Plus précisément, on sait qu'il existe des relations linéaires entre les séquences produites par une LFSM par construction. Cependant, ces relations ne sont pas souvent écrites explicitement.

Nous donnons ici une expression complète de ces relations en utilisant l'anneau des séries formelles $\mathbb{A}[[X]]$ (qui peut être vu comme le complété de la topologie X-adique associé à $\mathbb{A}[X]$). Étant donné $s = (s(t))_{t\in\mathbb{N}}$ une suite de $\mathbb{A}$ (respectivement $\mathbb{A}^i$), nous noterons $\Sigma(s, t_0)$ l'élément (respectivement le vecteur de taille i) de $\mathbb{A}$ défini par :

$$\Sigma(s, t_0) = \sum_{t\in\mathbb{N}} s(t + t_0)X^t$$

Théorème 4.1.2

Soit $\mathcal{L} = (A, B, C, D)$ une LFSM sur un anneau $\mathbb{A}$. Les vecteurs $\Sigma(m, t_0)$ et $\Sigma(v, t_0)$ vérifient :

$$\begin{cases} \Sigma(m, t_0) = \dfrac{\mathrm{Adj}(I - X \cdot A)}{\det(I - X \cdot A)} \cdot (m(t_0) + X \cdot B \cdot \Sigma(u, t_0)) \\ \Sigma(v, t_0) = C \cdot \dfrac{\mathrm{Adj}(I - X \cdot A)}{\det(I - X \cdot A)} \cdot (m(t_0) + X \cdot B \cdot \Sigma(u, t_0)) + D\Sigma(u, t_0) \end{cases}$$

Démonstration : Avec les notations précédentes, on a :

$$\Sigma(m, t_0 + 1) = A \cdot \Sigma(m, t_0) + B \cdot \Sigma(u, t_0) \quad (4.1)$$

$$\Sigma(m, t_0) = X \cdot \Sigma(m, t_0 + 1) + m^{(t_0)} \quad (4.2)$$

La relation 4.1 vient de la définition 4.1.1. La relation 4.2 vient de la définition de $\Sigma(m, t_0)$. D'où la relation :

$$\det(I - X \cdot A) \cdot \Sigma(m, t_0) = \mathrm{Adj}(I - X \cdot A) \cdot (m(t_0) + X \cdot B \cdot \Sigma(u, t_0))$$

On remarque alors que $\det(I - X \cdot A)$ est inversible dans $\mathbb{A}[[X]]$, ce qui mène à $\Sigma(m, t_0) = \mathrm{Adj}(I - X \cdot A) \cdot m^{(t_0)} / \det(I - X \cdot A) + X \cdot B \cdot \Sigma(u, t_0))$ dans $\mathbb{A}[[X]]$.

On conclut en utilisant la relation :

$$\Sigma(v, t_0) = C\Sigma(m, t_0) + D\Sigma(u, t_0)$$

■

Le polynôme $\det(I - X \cdot A)$ jouant un rôle central d'après le théorème précédent, on le nomme :

Définition 4.1.3

*Soit $\mathcal{L} = (A, B, C, D)$ une LFSM. Le polynôme $\det(I - X \cdot A)$ est appelé **polynôme de connexion** de $\mathcal{L}$. Il est noté $Q_{\mathcal{L}}(X)$, ou $Q(X)$ quand il n'y a pas d'ambiguïté.*

Remarque

Le degré de $Q(X)$ est inférieur ou égal à n ; avec égalité si et seulement si $\det A \neq 0$. De plus, $Q(0) = 1$ car $Q(0) = \det(I)$ par définition.

4.1.3 Familles classiques

Dans la suite, nous considérerons que $\mathbb{A}$ est un corps fini $\mathbb{F}_q$, car c'est le cas utilisé en pratique. Cette particularisation n'a pas de conséquence sur les résultats présentés, mais permet de considérer des automates réalisables en pratique, car nécessitant une mémoire finie. Ces automates ont été plus largement étudiés car ils s'implémentent facilement, en particulier sur le corps $\mathbb{F}_2$. En effet, comme vu dans le chapitre 3, l'addition dans $\mathbb{F}_2$ s'implémente grâce à une porte `XOR` ; tandis que la multiplication dans $\mathbb{F}_2$ s'implémente grâce à une porte `AND`.

Les LFSMs sont connues depuis longtemps. Il existe donc des familles particulières d'automates en fonction de l'application voulue. Ces automates sont définis par la forme de leur matrice de transition.

De plus, comme des communautés différentes ont travaillé sur ces objets, il existe plusieurs terminologies. Par exemple, il existe des différences de notations entre la communauté électronique [Sto73, Kag06, CM98] et la communauté cryptographique [GW96, LSRQ05, JD06, GK09]. Nous choisissons d'utiliser les notations de la communauté cryptographique.

Tout d'abord, il existe deux implémentations duales pour les LFSMs :

- Les **LFSRs en mode Fibonacci**, aussi appelés **External-XOR LFSRs**, ou simplement **LFSRs** ;
- Les **LFSRs en mode Galois**, aussi appelés **Internal-XOR LFSRs**, ou **LFSRs canoniques**.

Les LFSRs en mode Galois et Fibonacci sont définis de façon unique par le polynôme de connexion de la LFSM. Les matrices de transition sont décrites dans la figure 4.1, et les implémentations correspondantes dans le figure 4.2.

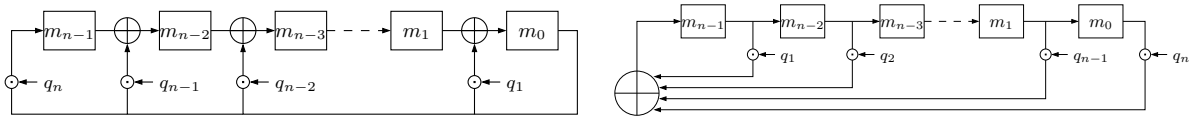
Une autre famille de LFSMs est celle des automates cellulaires (CA) [CM98, CM96a, CM96b, Kag06]. Nous présentons ici les automates cellulaires avec 3 voisins car leur implémentation est la plus simple. Ces automates sont caractérisés par une matrice de transition (figure 4.3) présentant des 1 sur la sur-diagonale et la sous-diagonale, les éléments de la diagonale sont quelconques, et le reste de la matrice est nul.

Nous introduisons aussi les ***ring* LFSRs**. Leurs mémoires forment un registre à décalage cyclique. Cela s'exprime de la façon suivante sur leur matrice de transition :

$$T_G = \begin{pmatrix} q_1 & 1 & & & \\ q_2 & & 1 & (0) & \\ \vdots & & (0) & \ddots & \\ q_{n-1} & & & & 1 \\ q_n & 0 & 0 & \cdots & 0 \end{pmatrix} \qquad T_F = \begin{pmatrix} 0 & 1 & & & \\ 0 & & 1 & (0) & \\ \vdots & & (0) & \ddots & \\ 0 & & & & 1 \\ q_n & q_{n-1} & \cdots & q_2 & q_1 \end{pmatrix}$$

(a) LFSR en mode Galois (b) LFSR en mode Fibonacci

FIGURE 4.1 – Matrices de transition des LFSRs en mode Galois et Fibonacci associées au polynôme de connexion $Q(X) = q_n X^n + \cdots + q_1 X + 1$

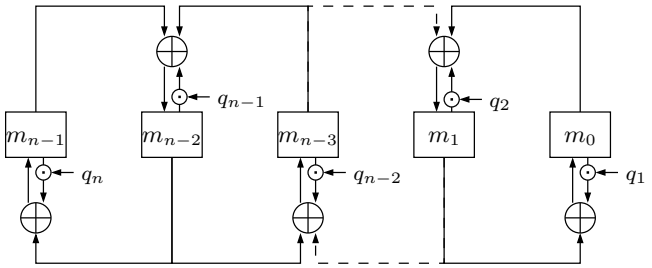


(a) LFSR en mode Galois (b) LFSR en mode Fibonacci

FIGURE 4.2 – Implémentation des LFSRs en mode Galois et Fibonacci avec polynôme de connexion $Q(X) = q_n X^n + \cdots + q_1 X + 1$

$$T_{CA} = \begin{pmatrix} q_1 & 1 & & & \\ 1 & q_2 & 1 & (0) & \\ & \ddots & \ddots & \ddots & \\ & (0) & 1 & q_{n-1} & 1 \\ & & & 1 & q_n \end{pmatrix}$$

(a) Matrice de transition d'un CA



(b) Implémentation d'un CA

FIGURE 4.3 – Matrice de transition et implémentation d'un automate cellulaire à 3 voisins

Définition 4.1.4

Soit une LFSM $\mathcal{L}$ avec matrice de transition A. $\mathcal{L}$ est appelée ring *LFSR si $A = (a_{i,j})_{0 \leq i,j < n}$ vérifie :*

$$\begin{cases} a_{i,i+1} = 1 \text{ pour } 0 \leq i < n-1 \\ a_{n-1,0} = 1 \end{cases}$$

i.e. ,

$$A = \begin{pmatrix} & 1 & & (*) & \\ & & \ddots & & \\ & (*) & & \ddots & \\ & & & & 1 \\ 1 & & & & \end{pmatrix}$$

En particulier, les LFSRs en mode Galois et Fibonacci sont des cas particuliers de *ring* LFSRs

On donne ici des exemples de tels automates. On considère le polynôme de connexion primitif $Q(X) = X^8 + X^6 + X^5 + X^3 + 1$. On note $\mathcal{L}_0$ le LFSR en mode Galois correspondant, $\mathcal{L}_1$ le LFSR en mode Fibonacci et $\mathcal{L}_2$ un *ring* LFSR avec polynôme de connexion $Q(X)$. Les matrices

de transition correspondantes T_0, T_1 et T_2 sont décrites dans la figure 4.4. On donne dans la figure 4.5 les implémentations de $\mathcal{L}_0$, $\mathcal{L}_1$ et $\mathcal{L}_2$, et, dans la table 4.1, les états successifs de $\mathcal{L}_0$, $\mathcal{L}_1$ et $\mathcal{L}_2$ pendant 8 clocks à partir du même état initial.

$$T_0 = \begin{pmatrix} 0 & 1 & & & & & & \\ 0 & & 1 & & & (0) & & \\ 1 & & & 1 & & & & \\ 0 & & & & 1 & & & \\ 1 & & & & & 1 & & \\ 1 & & (0) & & & & 1 & \\ 0 & & & & & & & 1 \\ 1 & & & & & & & \end{pmatrix}$$

(a) LFSR en mode Galois

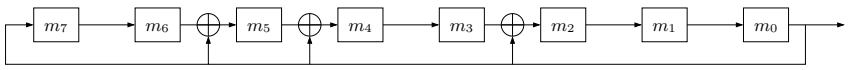
$$T_1 = \begin{pmatrix} & 1 & & & & & & \\ & & 1 & & & & (0) & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & (0) & & & & 1 & \\ & & & & & & & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

(b) LFSR en mode Fibonacci

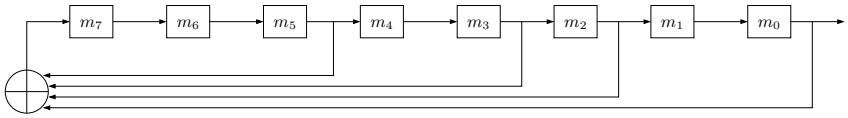
$$T_2 = \begin{pmatrix} & 1 & & & & & & \\ & & 1 & & & (0) & & \\ & & & 1 & & & & \\ & & & & 1 & & & 1 \\ & & & & & 1 & & \\ & & (0) & & & & 1 & \\ & & & & & & & 1 \\ 1 & & 1 & & & & & \end{pmatrix}$$

(c) *Ring* LFSR

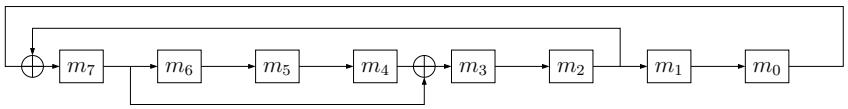
FIGURE 4.4 – Matrices de transition de $\mathcal{L}_0$, $\mathcal{L}_1$ et $\mathcal{L}_2$



(a) LFSR en mode Galois $\mathcal{L}_0$



(b) LFSR en mode Fibonacci $\mathcal{L}_1$



(c) *Ring* LFSR $\mathcal{L}_2$

FIGURE 4.5 – Trois LFSRs avec polynôme de connexion $Q(X) = X^8 + X^6 + X^5 + X + 1$

Remarque

À partir du même état initial `00000001`, les séquences produites (par exemple par la cellule m_0) sont différentes. Cependant, ce sont toutes des *m-sequences* définies par $Q(X) = X^8+X^6+X^5+X+1$ d'après le théorème 4.1.2, *i.e.* , il existe trois polynômes $P_0(X), P_1(X), P_2(X)$ de degré inférieur à 8 tels que les séquences produites par $\mathcal{L}_0$, $\mathcal{L}_1$ et $\mathcal{L}_2$ soient $P_0(X)/Q(X)$, $P_1(X)/Q(X)$ et $P_2(X)/Q(X)$.

Clock	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$
	Cellules	*Cellules*	*Cellules*
	7654321 0	7654321 0	7654321 0
0	0000000 1	0000000 1	0000000 1
1	1011010 0	1000000 0	1000000 0
2	0101101 0	0100000 0	0100100 0
3	0010110 1	0010000 0	0010010 0
4	1010001 0	1001000 0	1001001 0
5	0101000 1	0100100 0	0100000 1
6	1001110 0	1010010 0	1010000 0
7	0100111 0	0101001 0	0101100 0
8	0010011 1	0010100 1	0010110 0

TABLE 4.1 – États de $\mathcal{L}_0$, $\mathcal{L}_1$ et $\mathcal{L}_2$ pendant 8 clocks.

4.1.4 Implémentations matérielles efficaces

Nous présentons dans cette sous-section les différentes propositions existantes pour construire des LFSMs ayant une bonne implémentation matérielle, *i.e.* , d'après la sous-section 3.2.5, avec un coût, un chemin critique et un *fan-out* minimums. Ces implémentations concernent les LFSRs sur $\mathbb{F}_2$.

Dans [WM88], les auteurs proposent la famille des *top-bottom LFSR* : ce sont des *ring* LFSR divisés en deux parties : une partie Fibonacci et une partie Galois. Plus précisément, ces automates ont des matrices de transition de la forme :

$$A = \begin{pmatrix} g_1 & 1 & & & & & & \\ g_2 & & 1 & & & & (0) & \\ \vdots & & & \ddots & & & & \\ g_{i-1} & & & & \ddots & & & \\ & & & & & \ddots & & \\ & & (0) & & & & \ddots & \\ & & & & & & & 1 \\ 1 & & & & f_i & f_{i+1} & \dots & f_n \end{pmatrix}$$

Cette approche est un compromis entre les LFSRs en mode Galois et en mode Fibonacci. En particulier, le chemin critique, le *fan-out* et le coût d'un tel automate sont des valeurs intermédiaires aux cas Fibonacci et Galois. De plus, cette construction porte aussi les inconvénients de ces deux modes : une diffusion lente. Plus précisément, la modification de la valeur d'une cellule nécessite, dans le pire des cas, n transitions pour atteindre toutes les cellules de l'automate.

Dans [MRT04], les auteurs présentent une méthode pour, à partir d'un LFSR donné, construire un LFSR équivalent avec un chemin critique et un *fan-out* plus faible. Pour cela, ils modifient étape par étape la matrice de transition du LFSR original, en utilisant des décalages à droite et à gauche, et ce sans modifier le polynôme de connexion. Pour un polynôme de connexion donné, leur méthode donne un nouveau LFSR avec un chemin critique de 2, un *fan-out* de 3 et un coût constant, en considérant le LFSR original en mode Galois. Plus précisément, leur construction est efficace sur les polynômes dont les coefficients sont uniformément répartis, *i.e.* des polynômes avec le même écart entre deux coefficients non-nuls. En fait, leur méthode produit des LFSRs

avec une matrice de transition de la forme :

$$A = \begin{pmatrix} & 1 & & & & & \\ & & 1 & & & (0) & \\ & & & \ddots & & & \\ & & & & 1 & & \\ (0) & & & & h_1 & 1 & \\ & & & \iddots & h_2 & \ddots & \\ & & h_{n-4} & \iddots & & & \ddots \\ & h_{n-2} & h_{n-3} & & & & 1 \\ 1 & h_{n-1} & & & (0) & & \end{pmatrix}$$

pour le polynôme de connexion $X^n + h_{n-1}X^{n-1} + \cdots + h_1X + 1$ et n pair (la forme est comparable pour n impair).

Les auteurs donnent aussi une méthode générique, basée sur deux transformations appelées SDL et SDR qui préservent le polynôme de connexion, pour diminuer le coût du LFSR. Pour cela, ils présentent une technique à appliquer étape par étape.

4.1.5 Implémentations logicielles efficaces

Comme présenté dans la sous-section 3.3.3, la principale différence entre les implémentations logicielles et matérielles est la taille des données manipulées. En matériel on traite les bits un par un, tandis qu'en logiciel ils sont regroupés en mots dont la taille dépend de l'architecture du processeur. Pour profiter de cette architecture, les LFSRs doivent agir sur des mots. Dans ce qui suit, w est la taille d'un mot sur l'architecture visée. De plus, ces implémentations concernent les LFSRs sur $\mathbb{F}_2$.

Tout d'abord, les *Generalized Feedback Shift Registers* ont été proposés dans [LP73] pour augmenter le débit. L'idée est de paralléliser w LFSRs en mode Fibonacci. Les matrices de transition sont de la forme :

$$A = \begin{pmatrix} 0 & I_w & & & & \\ & 0 & I_w & & (0) & \\ & & 0 & I_w & & \\ & (0) & & \ddots & \ddots & \\ & & & & 0 & I_w \\ I_w & a_{n-2}I_w & \dots & a_2I_w & a_1I_w & a_0I_w \end{pmatrix}$$

où I_w est la matrice identité de taille $w \times w$ sur $\mathbb{F}_2$ et où a_i pour $0 \leq i \leq n-2$ sont des coefficients binaires.

En 1992, Matsumoto dans [MK92] généralise cette approche en considérant des LFSRs utilisant une transformation linéaire L comme rétroaction. Cette représentation est appelée *Twisted Generalized Feedback Shift Register*. Le même genre de construction est décrite dans [MN98] et est appelée *Mersenne Twister*. Plus précisément, dans ces approches, les LFSRs considérés sont en mode Fibonacci au niveau des mots, avec une rétroaction linéaire unique. La matrice de transition correspondante est de la forme :

$$A = \begin{pmatrix} 0 & I_w & & & & \\ & 0 & I_w & & (0) & \\ & & 0 & I_w & & \\ & (0) & & \ddots & \ddots & \\ & & & & 0 & I_w \\ I_w & 0 & 0 & L & 0 & 0 \end{pmatrix}$$

où I_w est la matrice identité de taille $w \times w$ et L est une matrice binaire de taille $w \times w$.

Dans [Nie95], l'idée est généralisée avec la *Multiple-Recursive Matrix Method* et est appliquée dans les *Xorshift Generators* décrits dans [Mar03] et étudiés dans [PL05]. Ici, les LFSRs sont en mode Fibonacci avec plusieurs rétroactions linéaires. La matrice correspondante est de la forme :

$$A = \begin{pmatrix} 0 & I_w & & & & \\ & 0 & I_w & & (0) & \\ & & 0 & I_w & & \\ & (0) & & \ddots & \ddots & \\ & & & & 0 & I_w \\ A_r & A_{r-1} & A_{r-2} & \dots & A_2 & A_1 \end{pmatrix}$$

avec I_w la matrice identité et les matrices A_i sont des transformations efficacement implémentables telles que les décalages à droite et à gauche. De plus, le polynôme de rétroaction se calcule en utilisant la relation $P(X) = \det\left(I + \sum_{j=1}^{r} X^j A_j\right)$.

Un cas particulier des *Multiple-Recursive Matrix Method* est étudié dans [TV03]. Les auteurs considèrent des matrices A_i de la forme $a_i \cdot T$ où T est une matrice $w \times w$ et a_i un scalaire. Dans ce cas, les auteurs donnent un algorithme pour construire des LFSMs avec un polynôme de connexion primitif.

Une autre technique pour construire des LFSRs implémentés de façon efficace en logiciel est de travailler avec des LFSRs sur $\mathbb{F}_{2^w}$ comme proposé dans [EJ02, BBC+08]. Cette technique se base alors sur l'utilisation de tables pré-calculées pour accélérer les calculs. Ces automates peuvent être considérés comme des automates sur $\mathbb{F}_2$ grâce à l'isomorphisme d'espaces vectoriels entre $\mathbb{F}_{2^w}$ et $(\mathbb{F}_2)^w$.

4.2 FCSR

Les registres à décalage avec retenue (FCSRs) sont des AFSRs sur $(\mathbb{Z}, N \in \mathbb{Z}, \mathcal{S} = \{0, 1, \dots, N-1\})$ [KG93b, KG97]. Les séquences produites sont donc des nombres N-adiques qui possèdent de nombreuses bonnes propriétés statistiques [Kob97, KG97]. Ces automates ont été introduits avec les deux modes Galois et Fibonacci [GK02].

Alors que les LFSMs sont implémentées (dans le cas d'automates binaires) avec des portes `XOR`, les FCSRs se fondent sur les additionneurs à retenue (figure 4.6). Comme leur nom l'indique, ces additionneurs nécessitent une mémoire pour stocker une retenue. Cependant, nous ne représenterons pas toujours cette cellule mémoire sur les schémas, pour ne pas les surcharger.

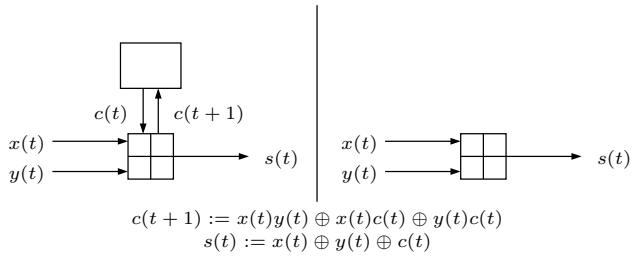
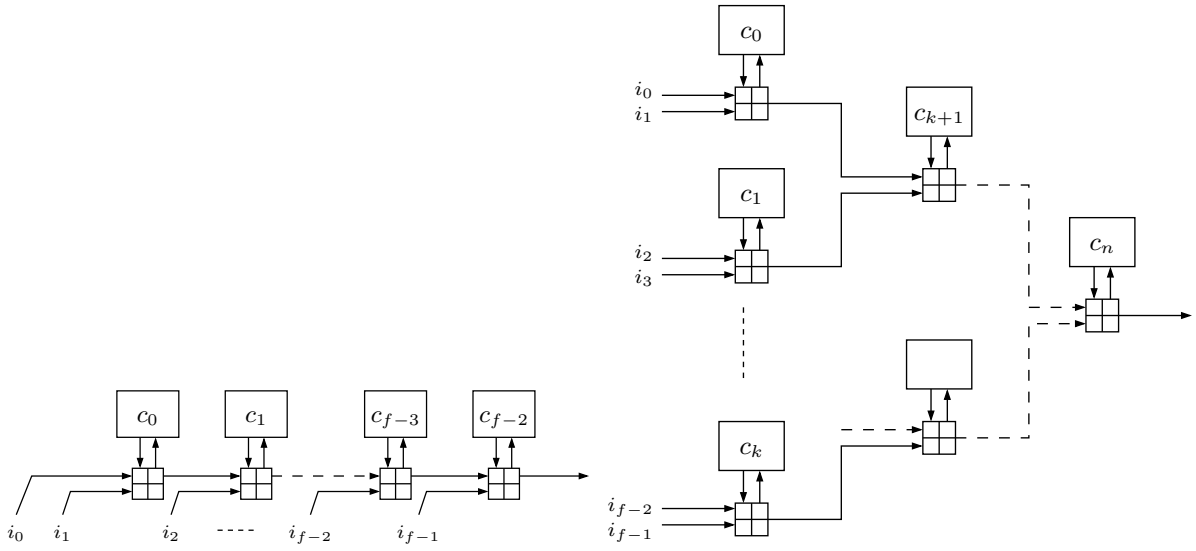


FIGURE 4.6 – Deux représentations d'un additionneur à retenue.

Alors que pour les LFSMs, l'implémentation ne consiste qu'à placer des portes `XOR`, pour les FCSRs, il faut aussi placer des cellules mémoires. De plus, lorsque l'on doit construire un additionneur avec plus de deux entrées, plusieurs implémentations sont possibles :

– Construire un circuit avec des additionneurs à retenue à deux entrées placés en série (figure 4.7(a)).
– Construire un circuit avec des additionneurs à retenue à deux entrées placés en parallèle (figure 4.7(b)).

Ces deux implémentations différentes d'un additionneur à f entrées n'ont pas les mêmes propriétés électroniques (*fan-out*, chemin critique et coût).



(a) Implémentation série (b) Implémentation parallèle

FIGURE 4.7 – Implémentations d'additionneurs à retenue avec plus de deux entrées.

4.2.1 FCSRs en mode Galois

Un FCSR en mode Galois (figure 4.8) est constitué de :
– Un registre principal de n éléments de $\mathcal{S}$: $m = (m_0, \ldots, m_{n-1})$;
– Un registre de retenue de n éléments de $\mathcal{S}$: $c = (c_0, \ldots, c_{n-1})$;
– n multiplicateurs dans $\mathcal{S}$: $(d_0, \ldots, d_{n-1})$;
– n additionneurs avec retenue.

Étant donné l'état à l'instant t $(m(t), c(t))$, l'état à l'instant $t+1$ est :

$$\forall 0 \leq i < n, \begin{cases} \sigma_i(t) &= m_{i+1}(t) + c_i(t) + d_i m_0(t) \text{ avec la convention } m_n(t) = 0 \\ m_i(t+1) &= \sigma_i \bmod N \\ c_i(t+1) &= \sigma_i \operatorname{div} N \end{cases}$$

La figure 4.8 est une représentation d'un FCSR en mode Galois.

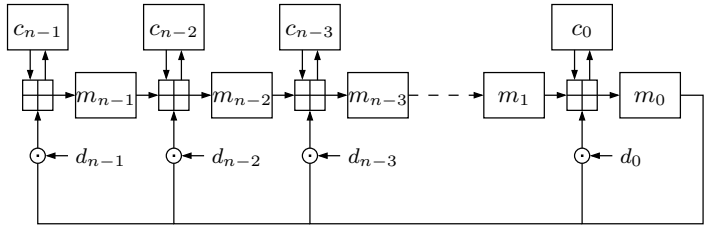


FIGURE 4.8 – Un FCSR en mode Galois

[GK02] présente une description profonde d'un tel automate. Nous rappelons ici la propriété principale des FCSRs [ABM08] :

Proposition 4.2.1

Soit $q = 1 - N\sum_{i=0}^{n-1} d_i N^i$ *et* $M_i(t_0) = \sum_{t=0}^{\infty} m_i(t+t_0)N^t$ *pour* $0 \leq i < n$, *i.e.* $M_i(t_0)$ *est l'entier N-adique correspondant à la séquence observée dans la cellule* i *du registre principal à partir de l'instant* t_0. *Alors, pour* $0 \leq i < n$, *il existe* $p_i(t_0) \in \mathbb{Z}$ *tel que* $M_i(t_0) = p_i(t_0)/q$.

4.2.2 FCSRs en mode Fibonacci

Un FCSR en mode Fibonacci (figure 4.9) est constitué de :
- Un registre principal de n éléments de $\mathcal{S}$: $m = (m_0, \ldots, m_{n-1})$;
- Une retenue $c \in \mathbb{N}$;
- n multiplicateurs dans $\mathcal{S}$: $(d_0, \ldots, d_{n-1})$;
- Un additionneur avec retenue.

Étant donné l'état à l'instant t $(m(t), c(t))$, l'état à l'instant $t+1$ est :

$$\begin{cases} \sigma(t) &= c + \sum_{i=0}^{n-1} m_i d_{n-1-i} \\ m(t+1) &= (m_1(t), \ldots, m_{n-1}(t), \sigma \bmod N) \\ c(t+1) &= \sigma \operatorname{div} N \end{cases}$$

La figure 4.9 est une représentation d'un FCSR en mode Fibonacci.

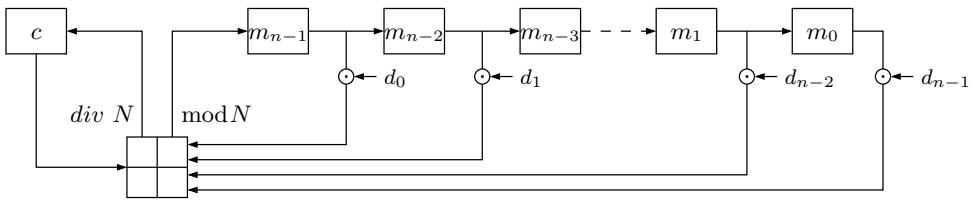


FIGURE 4.9 – Un FCSR en mode Fibonacci

Comme pour les FCSRs en mode Galois, la propriété 4.2.1 est vraie pour les FCSRs en mode Fibonacci [GK02].

4.2.3 Chiffrements à flot basés sur des FCSRs

Les FCSRs produisent des séquences non-linéaires et sont facilement implémentables. Le fait que les FCSRs produisent des séquences non-linéaires les protègent par construction contre les attaques algébriques. Ils ont donc été proposés comme une alternative aux LFSRs [AB05a, AB05b, ABL06, ABM08].

La famille de chiffrement F-FCSR, orienté matériel, est construite en utilisant un FCSR dont on extrait la suite chiffrante en appliquant un filtre linéaire sur l'état actuel.

Il existe aussi une famille de chiffrement orienté logiciel : X-FCSR [ABLM07]. Ces chiffrements se basent sur deux FCSRs, et utilisent une fonction d'extraction basée sur un chiffrement par bloc.

Cependant les premières versions de ces deux chiffrements à flot ont été cassées grâce à une faiblesse liée aux FCSRs en mode Galois [HJ08, SHJ09]. En effet, en mode Galois, il existe une corrélation entre les retenues et l'unique rétroaction. En particulier, il est possible que les

retenues soient toutes nulles sauf une pendant plusieurs cycles : la fonction de transition du FCSR devient alors affine. Cette faiblesse se produit avec une probabilité non-négligeable de par les bonnes propriétés statistiques des FCSRs. Ceci permet de construire des attaques contre les chiffrements à flot basés sur des FCSRs en mode Galois.

4.3 AFSR

Un AFSR est un registre à décalage qui opère en respectant la structure algébrique d'un anneau. Ce cadre théorique pour les générateurs pseudo-aléatoires est étudié dans [GK09]. Nous présentons ici les résultats essentiels de cette théorie.

4.3.1 Définition

Pour décrire le comportement d'un AFSR, on va utiliser les anneaux π-adiques présentés dans la section 2.4. Cependant, on peut définir un AFSR sans cette notion. Les éléments nécessaires pour construire un AFSR sont :

- Un anneau $\mathbb{A}$,
- Un élément $\pi \in \mathbb{A}$,
- Un système de représentants $\mathcal{S} \subset \mathbb{A}$ de $\mathbb{A}/\pi$.

De plus, $\mathcal{S}$ ayant été choisi, chaque élément $a \in \mathbb{A}$ s'écrit de façon unique $a = a_0 + b\pi$ avec $a_0 \in \mathcal{S}$. Par abus de notations, on notera :

$$a_0 = a \bmod \pi \text{ et } b = a \operatorname{div} \pi = \frac{a - a_0}{\pi}$$

Définition 4.3.1

Soient $q_0, \ldots, q_m \in \mathbb{A}$ tels que q_0 est inversible dans $\mathbb{A}/\pi$. Un ***registre à décalage algébrique****, ou* ***AFSR****, sur $(\mathbb{A}, \pi, \mathcal{S})$ de longueur m avec coefficients $q_0, \ldots, q_m$ est une machine dont les états sont de la forme*

$$(a_0, \ldots, a_{m-1}, z) \text{ avec } a_i \in \mathcal{S} \text{ et } z \in \mathbb{A}$$

La fonction de transition est :

$$\begin{array}{ccc} \mathcal{S}^m \times \mathbb{A} & \to & \mathcal{S}^m \times \mathbb{A} \\ (a_0, \ldots, a_{m-1}, z) & \mapsto & (a_1, \ldots, a_{m-1}, -q_0^{-1}\sigma \bmod \pi, \sigma \operatorname{div} \pi) \end{array} \quad \text{avec } \sigma = \sum_{i=1}^{m} q_i a_{m-i} + z$$

La fonction d'extraction est :

$$\begin{array}{ccc} \mathcal{S}^m \times \mathbb{A} & \to & \mathcal{S} \\ (a_0, \ldots, a_{m-1}, z) & \mapsto & a_0 \end{array}$$

La figure 4.10 représente un AFSR.

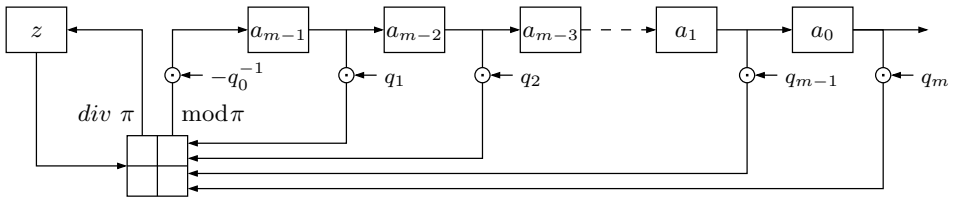


FIGURE 4.10 – Représentation d'un AFSR de longueur m avec coefficients $q_0, \ldots, q_m$

4.3.2 Théorème fondamental des AFSRs

On considère un AFSR sur $(\mathbb{A}, \pi, \mathcal{S})$ de longueur m avec coefficients $q_0, \ldots, q_m$. De plus, on suppose que π vérifie

$$\bigcap_{i \in \mathbb{N}} \langle \pi^i \rangle = 0$$

On peut alors considérer la topologie π-adique de $\mathbb{A}$.

Théorème 4.3.2 (*[KX99]*)

La séquence produite par un AFSR initialisé dans l'état $(a_0, \ldots, a_{m-1}, z)$ est la séquence $\mathrm{seq}_\pi(u/q)$ *où*

$$\begin{cases} q = \displaystyle\sum_{i=0}^{m} q_i \pi^i \\ u = \displaystyle\sum_{n=0}^{m-1} \sum_{i=0}^{n} q_i a_{n-i} \pi^n - z\pi^m \end{cases}$$

Remarque

- L'élément u/q a un sens dans $\mathbb{A}_\pi$ car, par la proposition 2.4.4, q est inversible dans $\mathbb{A}_\pi$.
- Si les q_i appartiennent à $\mathcal{S}$, alors l'écriture de q est unique. Cependant cette restriction n'est pas obligatoire.

4.3.3 Conclusion

Nous ne développerons pas plus ici la théorie des AFSRs car, pour obtenir des résultats significatifs, il faut faire des hypothèses supplémentaires sur $\mathbb{A}$ et la structure de $\mathbb{A}_\pi$. Le lecteur pourra se référer à [GK09] pour plus de détails.

Chapitre 5

AFSR

Sommaire

5.1 Introduction

Nous présentons dans ce chapitre une approche globale des AFSRs, *i.e.* nous allons construire des automates linéaires sur un anneau π-adique. En particulier, ces automates pourront avoir des entrées et des sorties (les AFSRs présentés dans le chapitre 4 n'ont pas d'entrées, et une seule sortie). Cette approche est un travail nouveau, *i.e.* ne faisant l'objet d'aucune publication actuellement.

Notre approche se fonde sur les automates linéaires (définition 3.1.3). Cependant, nous verrons qu'il est impossible d'appliquer directement cette définition, car elle mène à considérer des automates dont la transition nécessite un temps infini. Pour s'affranchir de cette limite, nous décomposerons les opérations utilisées par ces méta-automates, afin de construire des automates réalisant ces opérations simples. Cette approche permet de mieux comprendre les opérations effectuées par ces automates.

Nous étudierons ensuite ces automates en détail. En particulier, nous caractériserons les séquences produites et nous nous intéresserons aux automates définis par des matrices à coefficients dans $\mathcal{S}$, et non pas dans $\mathbb{A}_\pi$ tout entier.

Comme nous l'avons présenté dans l'état de l'art, les AFSRs sont une approche théorique recouvrant, entre autre, les LFSRs et les FCSRs. C'est pourquoi ce chapitre a une nature très théorique. En particulier, nous ne donnerons pas d'exemples concrets d'automates π-adiques.

Dans la suite, on considère la topologie π-adique (introduite dans la section 2.4) associée à un anneau $\mathbb{A}$, *i.e.* on considère :

- un anneau intègre $\mathbb{A}$;
- un élément non-nul π de $\mathbb{A}$ vérifiant $\bigcap_{i\in\mathbb{N}} \langle \pi^i \rangle = \{0\}$;
- un système de représentants $\mathcal{S}$ de $\mathbb{A}/\pi$ dans $\mathbb{A}$.

Pour tout $a \in \mathbb{A}_\pi$, on note :
- $a \bmod \pi$ l'unique élément de $\mathcal{S}$ congru à a modulo π ;
- $a \operatorname{div} \pi$ l'élément $\dfrac{a - (a \bmod \pi)}{\pi}$.

Par le théorème 2.4.1, chaque élément a de $\mathbb{A}_\pi$ s'écrit de façon unique sous la forme $\sum_{i\in\mathbb{N}} \alpha_i \pi^i$ avec $\alpha_i \in \mathcal{S}$. On a alors :

$$\begin{cases} a \bmod \pi = \alpha_0 \\ a \operatorname{div} \pi = \sum_{i\in\mathbb{N}} \alpha_{i+1}\pi^i \end{cases}$$

De plus, on note $\mathrm{seq}(a) = (\alpha_i)_{i\in\mathbb{N}}$ la suite π-adique associée à un nombre. Réciproquement, étant donné une suite d'éléments $s = (s_i)_{i\in\mathbb{N}}$ de $\mathcal{S}$, on note $\Sigma(s, i_0) = \sum_{i\in\mathbb{N}} s_{i+i_0}\pi^i$.

5.2 Combinaisons linéaires dans $\mathbb{A}_\pi$

Dans le but de construire des automates linéaires (définition 3.1.3), on doit être capable de calculer le produit entre une matrice et un vecteur dans $\mathbb{A}_\pi$. On s'intéresse donc au calcul de combinaisons linéaires dans $\mathbb{A}_\pi$. L'algorithme II.1 calcule k combinaisons linéaires de l entiers π-adiques, *i.e.* le produit d'une matrice $k \times l$ par un vecteur colonne dans $\mathbb{A}_\pi$.

Algorithme II.1 : Calcul du développement de Hensel de combinaisons linéaires d'éléments de $\mathbb{A}_\pi$

Données : $\mathbb{A}_\pi$ un anneau π-adique ;
$\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$ dans $\mathbb{A}$;
A une matrice $k \times l$ dans $\mathbb{A}_\pi$.
Entrées : $X = (x_1, \ldots, x_l) \in \mathbb{A}_\pi^l$ avec
$(\mathrm{seq}_\mathcal{S}(x_1), \ldots, \mathrm{seq}_\mathcal{S}(x_l)) = (x_1(t), \ldots, x_l(t))_{t\in\mathbb{N}} = (x(t))_{t\in\mathbb{N}}$.
Sorties : $Y = (y_1, \ldots, y_k) \in \mathbb{A}_\pi^l$ avec
$(\mathrm{seq}_\mathcal{S}(y_1), \ldots, \mathrm{seq}_\mathcal{S}(y_k)) = (y_1(t), \ldots, y_k(t))_{t\in\mathbb{N}} = (y(t))_{t\in\mathbb{N}}$.
début
  $c(0) \leftarrow (0, \ldots, 0) \in \mathbb{A}_\pi^k$;
  pour $t = 0$ **à** ∞ **faire**
    $y(t) \leftarrow Ax(t) + c(t) \bmod \pi$;
    $c(t+1) \leftarrow Ax(t) + c(t) \operatorname{div} \pi$;
  retourner $(y(t))_{t\in\mathbb{N}}$;
fin

Démonstration de l'algorithme II.1 : On va montrer par récurrence sur n l'invariant

$$A\sum_{t=0}^{n} x(t)\pi^t = \sum_{t=0}^{n} y(t)\pi^t + c(n+1)\pi^{n+1}$$

Cas $n = 0$: Clair par définition de $\bmod\pi$ et $\operatorname{div}\pi$.

Supposons la propriété vraie au rang n. Alors :

$$\begin{aligned} A\sum_{t=0}^{n+1} x(t)\pi^t &= A\sum_{t=0}^{n} x(t)\pi^t + Ax(n+1)\pi^{n+1} \\ &= \sum_{t=0}^{n} y(t)\pi^t + c(n+1)\pi^{n+1} + Ax(n+1)\pi^{n+1} \end{aligned}$$

$$= \sum_{t=0}^{n} y(t)\pi^t + (y(n+1) + \pi c(n+2))\pi^{n+1}$$
$$= \sum_{t=0}^{n+1} y(t)\pi^t + c(n+2)\pi^{n+2}$$

La propriété est donc vraie au rang $n+1$.

Par unicité du développement de Hensel, on a donc $(\text{seq}_{\mathcal{S}}(y_1), \ldots, \text{seq}_{\mathcal{S}}(y_k)) = (y_1(t), \ldots, y_k(t))_{t\in\mathbb{N}}$ avec $Y = AX$. ■

On remarque que cet algorithme produit les développements de Hensel des combinaisons linéaires, caractère par caractère. De plus, le calcul de k combinaisons linéaires nécessite un vecteur c « de retenues » à coefficients dans $\mathbb{A}_\pi$, *i.e.* , le vecteur c propage les retenues des calculs à l'itération t aux calculs suivants.

5.3 Première discussion

Puisque l'on sait calculer des combinaisons linéaires dans $\mathbb{A}_\pi$, il nous est possible de construire des automates linéaires sur $\mathbb{A}_\pi$ comme décrits par la définition 3.1.3. Plus précisément, ils sont définis par (dans le modèle de Mealy) :

- L'ensemble des états est $\mathbb{A}_\pi^n$;
- L'alphabet d'entrée est $\mathbb{A}_\pi^l$;
- L'alphabet de sortie est $\mathbb{A}_\pi^k$;
- La fonction de transition $\delta(U, M)$ définie par :

$$A \cdot M + B \cdot U$$

où A et B sont des matrices à coefficients dans $\mathbb{A}_\pi$ de tailles respectives $n \times n$ et $n \times l$;

- La fonction d'extraction $\omega(U, M)$ définie par :

$$C \cdot M + D \cdot U$$

où C et D sont des matrices à coefficients dans $\mathbb{A}_\pi$ de tailles respectives $k \times n$ et $k \times l$.

Plus précisément, pour $M \in \mathbb{A}_\pi^n$ un état et $U \in \mathbb{A}_\pi^l$ une entrée, on note $(m(t))_{t\in\mathbb{N}} := \text{seq}_{\mathcal{S}}(M)$ et $(u(t))_{t\in\mathbb{N}} := \text{seq}_{\mathcal{S}}(U)$. Alors, en notant

$$\begin{cases} N := A \cdot M + B \cdot U \\ V := C \cdot M + D \cdot U \end{cases}$$

le corps de la boucle de l'algorithme II.1 est :

$$\begin{aligned} n(t) &\leftarrow Am(t) + Bu(t) + c(t) \bmod \pi \\ c(t+1) &\leftarrow Am(t) + Bu(t) + c(t) \operatorname{div} \pi \\ v(t) &\leftarrow Cm(t) + Du(t) + c'(t) \bmod \pi \\ c'(t+1) &\leftarrow Cm(t) + Du(t) + c'(t) \operatorname{div} \pi \end{aligned}$$

avec $c(t) \in \mathbb{A}_\pi^n$ et $c'(t) \in \mathbb{A}_\pi^k$ deux vecteurs de retenues, et $c(0) = 0$ et $c'(0) = 0$. On a alors $\text{seq}_{\mathcal{S}}(N) = (n(t))_{t\in\mathbb{N}}$ et $\text{seq}_{\mathcal{S}}(V) = (v(t))_{t\in\mathbb{N}}$.

Cependant, il faut remarquer que l'algorithme II.1 est composé d'une boucle infinie, *i.e.* formellement l'algorithme ne finit jamais. Les automates ainsi construits sont des automates irréalisables en pratique. On va les concevoir comme étant des méta-automates : on va considérer les automates permettant de réaliser le corps de la boucle de l'algorithme II.1. En itérant ces automates, on construira, pas à pas, les entiers π-adiques désirés.

5.4 Définition et premières propriétés

À partir de la discussion précédente, on donne la définition suivante :

Définition 5.4.1

Soient $\mathbb{A}_\pi$ un anneau π-adique, $\mathcal{S} \subset \mathbb{A}$ un système de représentants de $\mathbb{A}/\pi$. Un AFSR avec k entrées et l sorties est la donnée de :

- *deux matrices A, B, de tailles respectives $n \times n$ et $n \times k$, à coefficients dans $\mathbb{A}_\pi$. Ces matrices servent à définir la fonction de transition de l'automate.*
- *deux matrices C, D, de tailles respectives $l \times n$ et $l \times k$, à coefficients dans $\mathbb{A}_\pi$. Ces matrices servent à définir la fonction d'extraction de l'automate.*

L'état de l'automate à l'instant t est constitué de :

- *$m(t) \in \mathcal{M}_{n,1}(\mathcal{S})$;*
- *$c(t) \in \mathcal{M}_{n,1}(\mathbb{A}_\pi)$, le vecteur de retenues nécessaire à la fonction de transition ;*
- *$c'(t) \in \mathcal{M}_{l,1}(\mathbb{A}_\pi)$, le vecteur de retenues nécessaire à la fonction d'extraction.*

De plus, on note :

- *$u(t) \in \mathcal{M}_{n,1}(\mathcal{S})$ les entrées à l'instant t ;*
- *$v(t) \in \mathcal{M}_{l,1}(\mathcal{S})$ les sorties à l'instant t.*

On a alors les relations suivantes :

$$\begin{cases} m(t+1) &= Am(t) + Bu(t) + c(t) \bmod \pi \\ c(t+1) &= Am(t) + Bu(t) + c(t) \operatorname{div} \pi \\ v(t) &= Cm(t) + Du(t) + c'(t) \bmod \pi \\ c'(t+1) &= Cm(t) + Du(t) + c'(t) \operatorname{div} \pi \end{cases}$$

Remarque

Les vecteurs de retenues sont définis comme étant à coefficients dans $\mathbb{A}_\pi$. Cependant, cette définition est large. En pratique, comme vu dans l'algorithme II.1, les retenues dépendent des matrices A, B, C, D. Pour s'en convaincre, il suffit de regarder leur comportement lorsque $c(0)$ et $c'(0)$ sont nuls : leurs valeurs seront construites à partir des produits $Am(0) + Bu(0)$ et $Cm(0) + Du(0)$ avec $m(0)$ et $u(0)$ à coefficients dans $\mathcal{S}$.

En particulier, considérer des valeurs de retenues sans rapport avec les matrices A, B, C, D revient à, artificiellement, rajouter $n + l$ entrées à l'automate. En effet, notons $\text{seq}(c(0)) = (\gamma(t))_{t\in\mathbb{N}}$ et $\text{seq}(c'(0)) = (\gamma'(t))_{t\in\mathbb{N}}$, alors, l'AFSR à k entrées défini par les matrices A, B, C, D, et avec l'état initial $(m(0), c(0), c'(0))$ et les entrées $(u(t))_{t\in\mathbb{N}}$ produira les mêmes séquences que l'AFSR défini par :

- les matrices $A, \hat{B}, C, \hat{D}$, où $\hat{B}$ et $\hat{D}$ sont définis par blocs par

$$\hat{B} = \begin{pmatrix} B & I_{n,n} & 0_{n,l} \end{pmatrix}, \hat{D} = \begin{pmatrix} D & 0_{l,n} & I_{l,l} \end{pmatrix}$$

- l'état initial $(m(0), 0, 0)$,
- les entrées $(u(t)|\gamma(t)|\gamma'(t))_{t\in\mathbb{N}}$.

On donne la proposition suivante qui donne une première approche avec la topologie π-adique d'un AFSR :

Proposition 5.4.2

Avec les notations de la définition 5.4.1, on a les relations :

$$\begin{cases} \Sigma(m, t+1) = A \cdot \Sigma(m, t) + B \cdot \Sigma(u, t) + c(t) \\ \Sigma(v, t) = C \cdot \Sigma(m, t) + D \cdot \Sigma(u, t) + c'(t) \end{cases}$$

Démonstration : Tout d'abord, on a la relation suivante pour tout $t \in \mathbb{N}$:

$$m(t+1) + \pi c(t+1) = Am(t) + c(t) + Bu(t)$$

En multipliant cette relation par π^i et en sommant pour $i \in \mathbb{N}$, on obtient :

$$\sum_{i\in\mathbb{N}} (m(t+1+i) + \pi c(t+1+i)) \cdot \pi^i = \sum_{i\in\mathbb{N}} (Am(t+i) + c(t+i) + Bu(t+i)) \cdot \pi^i$$

D'où,

$$\Sigma(m, t+1) + \pi \sum_{i\in\mathbb{N}} c(t+1+i)\pi^i = A \cdot \Sigma(m,t) + \sum_{i\in\mathbb{N}} c(t+i)\pi^i + B \cdot \Sigma(u,t)$$

Ce qui donne la première relation annoncée. La seconde relation s'obtient de manière analogue. ■

On donne maintenant le théorème fondamental des AFSRs :

Théorème 5.4.3

Avec les notations de la définition 5.4.1, on a les relations :

$$\begin{cases} \Sigma(m,t) = (I - \pi A)^{-1}(m(t) + \pi c(t) + \pi \cdot B \cdot \Sigma(u,t)) \\ \Sigma(v,t) = C \cdot (I - \pi A)^{-1}(m(t) + \pi c(t) + \pi \cdot B \cdot \Sigma(u,t)) + D \cdot \Sigma(u,t) + c'(t) \end{cases}$$

Démonstration : Par la proposition 5.4.2 et la relation $\Sigma(m,t) = \pi\Sigma(m,t+1) + m(t)$, on a :

$$\Sigma(m,t) = \pi(A \cdot \Sigma(m,t) + c(t) + B \cdot \Sigma(u,t)) + m(t)$$

D'où,

$$(I - \pi A)\sigma(m,t) = m(t) + \pi c(t) + \pi \cdot B \cdot \Sigma(u,t)$$

Or $(I - \pi A)$ est inversible dans $\mathbb{A}_\pi$ car $\det(I - \pi A) \equiv \det(I) \equiv 1[\pi]$.

La seconde relation s'obtient en remplaçant l'expression obtenue pour $\Sigma(m,t)$ dans la relation de la proposition 5.4.2. ■

5.5 Opérations élémentaires

On s'intéresse maintenant à l'implémentation des AFSRs (en particulier d'un point de vue matériel). Pour cela, on doit s'intéresser aux opérations élémentaires utilisées dans l'algorithme II.1, à savoir :

- l'addition de deux éléments de $\mathbb{A}_\pi$,
- la multiplication de deux éléments de $\mathbb{A}_\pi$, l'un étant constant (nous l'appellerons donc scalaire).

Les algorithmes II.2 et II.3 présentent les méthodes de calcul pour la somme de deux éléments de $\mathbb{A}_\pi$ et la multiplication par un scalaire d'un élément $\mathbb{A}_\pi$. On les présente côte-à-côte pour mettre en lumière leurs comportements analogues.

Algorithme II.2 : Calcul du développement de Hensel dans $\mathbb{A}_\pi$ de la somme de deux éléments : $x + y$

Données : $\mathbb{A}_\pi$ un anneau π-adique ;
$\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$ dans $\mathbb{A}$.
Entrées : $x \in \mathbb{A}_\pi$ avec $\text{seq}_\mathcal{S}(x) = (x(t))_{t\in\mathbb{N}}$;
$y \in \mathbb{A}_\pi$ avec $\text{seq}_\mathcal{S}(y) = (y(t))_{t\in\mathbb{N}}$.
Sorties : $\text{seq}_\mathcal{S}(x + y) = (z(t))_{t\in\mathbb{N}}$.
début
  $c(0) \leftarrow 0$;
  pour $t = 0$ **à** ∞ **faire**
    $z(t) \leftarrow x(t) + y(t) + c(t) \bmod \pi$;
    $c(t+1) \leftarrow x(t) + y(t) + c(t) \operatorname{div} \pi$;
  retourner $(z(t))_{t\in\mathbb{N}}$;
fin

Algorithme II.3 : Calcul du développement de Hensel dans $\mathbb{A}_\pi$ du produit d'un élément par un scalaire : $\alpha \cdot x$

Données : $\mathbb{A}_\pi$ un anneau π-adique ;
$\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$ dans $\mathbb{A}$.
Entrées : $x \in \mathbb{A}_\pi$ avec $\text{seq}_\mathcal{S}(x) = (x(t))_{t\in\mathbb{N}}$;
$\alpha \in \mathbb{A}_\pi$.
Sorties : $\text{seq}_\mathcal{S}(\alpha x) = (z(t))_{t\in\mathbb{N}}$
début
  $c(0) \leftarrow 0$;
  pour $t = 0$ **à** ∞ **faire**
    $z(t) \leftarrow \alpha x(t) + c(t) \bmod \pi$;
    $c(t+1) \leftarrow \alpha x(t) + c(t) \operatorname{div} \pi$;
  retourner $(z(t))_{t\in\mathbb{N}}$;
fin

Démonstration de l'algorithme II.2 : On va montrer par récurrence sur n l'invariant

$$\sum_{t=0}^{n} x(t)\pi^t + \sum_{t=0}^{n} y(t)\pi^t = \sum_{t=0}^{n} z(t)\pi^t + \pi^{n+1}c(n+1)$$

Cas $n = 0$: Clair.

Supposons la propriété vraie au rang n.
Alors :

$$\begin{aligned}
&\sum_{t=0}^{n+1} x(t)\pi^t + \sum_{t=0}^{n+1} y(t)\pi^t \\
&= \sum_{t=0}^{n} x(t)\pi^t + \sum_{t=0}^{n} y(t)\pi^t \\
&\quad +(x(n+1) + y(n+1))\pi^{n+1} \\
&= \sum_{t=0}^{n} z(t)\pi^t \\
&\quad +\pi^{n+1}(c(n+1) + x(n+1) + y(n+1)) \\
&= \sum_{t=0}^{n} z(t)\pi^t \\
&\quad +\pi^{n+1}(z(n+1) + \pi c(n+2))
\end{aligned}$$

La propriété est donc vraie au rang $n + 1$.

Par unicité du développement de Hensel, on a donc $\text{seq}_\mathcal{S}(x + y) = (z_t)_{t\in\mathbb{N}}$. ■

Démonstration de l'algorithme II.3 : On va montrer par récurrence sur n l'invariant

$$\alpha \sum_{t=0}^{n} x(t)\pi^t = \sum_{t=0}^{n} z(t)\pi^t + \pi^{n+1}c(n+1)$$

Cas $n = 0$: Clair.

Supposons la propriété vraie au rang n.
Alors :

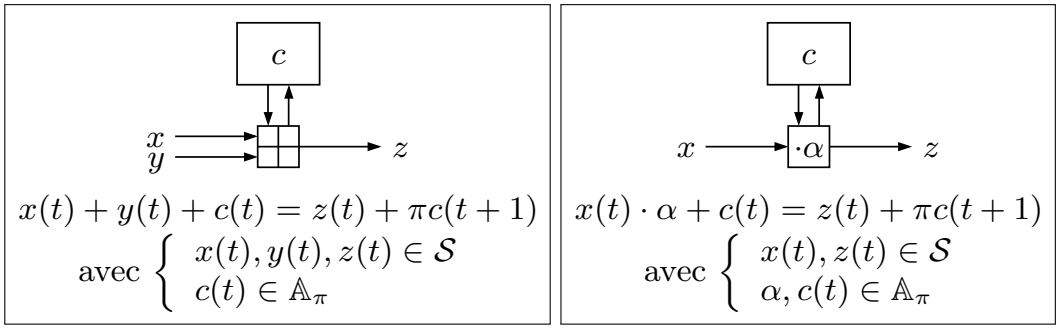
$$\begin{aligned}
&\alpha \sum_{t=0}^{n+1} x(t)\pi^t \\
&= \alpha \sum_{t=0}^{n} x(t)\pi^t \\
&\quad +\alpha x(n+1)\pi^{n+1} \\
&= \sum_{t=0}^{n} z(t)\pi^t \\
&\quad +\pi^{n+1}(c(n+1) + \alpha x(n+1)) \\
&= \sum_{t=0}^{n} z(t)\pi^t \\
&\quad +\pi^{n+1}(z(n+1) + \pi c(n+2))
\end{aligned}$$

La propriété est donc vraie au rang $n + 1$.

Par unicité du développement de Hensel, on a donc $\text{seq}_\mathcal{S}(\alpha x) = (z(t))_{t\in\mathbb{N}}$. ■

Comme le montrent ces algorithmes, ces opérations se traitent de façon séquentielle sur le développement de Hensel des éléments en ayant recours à une variable pour stocker les retenues.

La figure 5.1 présente les schémas de ces opérations. Pour ne pas surcharger les schémas, nous ne représenterons pas toujours les cellules de retenue de ces opérations.



(a) Additionneur dans $\mathbb{A}_\pi$ (b) Multiplicateur dans $\mathbb{A}_\pi$

FIGURE 5.1 – Opérateurs élémentaires dans $\mathbb{A}_\pi$

On verra que ces deux opérations s'implémentent simplement dans le cas des LFSRs et des FCSRs.

5.6 Deuxième discussion

La définition 5.4.1 permet de considérer des AFSRs irréalisables en pratique d'un point de vue de la mémoire requise. En effet, une des premières conditions pour que les automates soient constructibles est qu'ils nécessitent une mémoire finie, *i.e.* un état $(m(t), c(t), c'(t))$ doit nécessiter une mémoire finie pour être stocké.

De façon équivalente, puisque un AFSR manipule des éléments dans $\mathbb{A}_\pi$, il convient de se demander quand un élément de $\mathbb{A}_\pi$ est représentable avec une mémoire finie.

Soient $\mathbb{A}_\pi$ un anneau π-adique et $\mathcal{S}$ un système de représentants de $\mathbb{A}/\pi$. On peut représenter un nombre π-adique de deux façons :

De manière explicite : un nombre π-adique est identifié avec son développement de Hensel. C'est la manière traditionnelle utilisée en informatique, par exemple un entier est stocké explicitement grâce à son écriture binaire.

De manière descriptive : un nombre π-adique est identifié par une expression permettant de le construire. C'est la manière formelle de décrire les éléments, par exemple avec cette méthode on peut représenter des éléments tels que

$$\sum_{i \in \mathbb{N}} X^{2^i} \text{ ou encore } \sqrt{2}$$

qui nécessitent une mémoire infinie avec la manière explicite.

Pour notre part, nous nous intéressons au développement de Hensel des entiers π-adiques. La manière descriptive semble donc peu adaptée car pour calculer le développement de Hensel d'un élément, il faudra évaluer l'expression le représentant, ce qui peut nécessiter de nombreuses opérations. On choisit donc d'utiliser la manière explicite. Dans ce cas, un élément de $\mathbb{A}_\pi$ est représentable avec une mémoire finie si son développement de Hensel est périodique. Cela implique que l'élément correspondant dans $\mathbb{A}_\pi$ soit rationnel :

Proposition 5.6.1

Soit $\text{seq}_{\mathcal{S}}(u) = (u_i)_{i \in \mathbb{N}}$ *une suite périodique. Alors, il existe* $a, b \in \mathbb{A}$ *tel que* $u = a/b$.

Démonstration : Soit N la pré-période et T la période de la suite. On a alors :

$$\begin{aligned} u &= \sum_{i=0}^{N-1} u_i\pi^i + \sum_{i=0}^{\infty}\left(\sum_{t=0}^{T-1} u_{N+t}\pi^t\right)\pi^{N+i\cdot T} \\ &= \sum_{i=0}^{N-1} u_i\pi^i + \left(\sum_{t=0}^{T-1} u_{N+t}\pi^t\right)\sum_{i=0}^{\infty}\pi^{N+i\cdot T} \end{aligned}$$

Or, par la proposition 2.4.3, $\sum_{i=0}^{\infty}\pi^{N+i\cdot T} = -\dfrac{\pi^N}{\pi^T-1}$. On en déduit :

$$u = \frac{\left(\sum_{i=0}^{N-1} u_i\pi^i\right)(\pi^T-1) - \pi^N\sum_{t=0}^{T-1} u_{N+t}\pi^t}{\pi^T-1}$$

D'où le résultat puisque $\pi \in \mathbb{A}$ et $u_i \in \mathcal{S} \subset \mathbb{A}$. ■

De plus, pour s'assurer a priori que la mémoire requise sera finie, on impose que $\mathbb{A}/\pi$ soit fini.

Cependant ces conditions ne suffisent pas à garantir que la mémoire requise sera finie. C'est là encore la limite de l'approche théorique : nous ne pouvons pas traiter tous les anneaux π-adiques de la même façon. Les AFSRs doivent donc être vus comme des objets théoriques. Pour les construire, il faudra spécifier l'anneau choisi, et alors on pourra évaluer la mémoire requise.

5.7 Automate élémentaire de multiplication/division

Nous présentons ici un automate qui permet de multiplier le nombre qu'il prend en entrée par un nombre rationnel fixé. Cet objet est important pour la suite, car il permettra, dans certains cas discutés ensuite, à partir d'un AFSR défini avec des matrices à coefficients rationnels, de considérer un AFSR similaire mais avec des matrices à coefficients dans $\mathcal{S}$. En d'autres termes, les séquences produites par un AFSR défini avec des matrices à coefficients rationnelles sont constructibles avec un AFSR à coefficients dans $\mathcal{S}$. Ceci facilitera en particulier l'implémentation.

Nous présentons ici les **pales en mode Galois**. Elles ont été introduites d'abord pour les LFSRs [SC88], puis appliquées aux FCSRs [Lau09]. Nous les appelons « pales » car elles sont utilisés dans [SC88] pour construire des *windmill generators* : des « générateurs moulin à vent ». Les pales en mode Galois sont des AFSRs avec une entrée et une sortie, et qui sont définies par des matrices de la forme suivante :

$$A = \begin{pmatrix} a_1 & 1 & & & \\ a_2 & & 1 & (0) & \\ \vdots & & (0) & \ddots & \\ a_{d-1} & & & & 1 \\ a_d & 0 & 0 & \cdots & 0 \end{pmatrix}, B = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{d-2} \\ b_{d-1} \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & \ldots & 0 \end{pmatrix}, D = \begin{pmatrix} d_0 \end{pmatrix}$$

La figure 5.2 présente une pale en mode Galois. Dans ce schéma, pour ne pas le surcharger, nous ne représentons pas les cellules de retenue présente pour effectuer les additions.

Nous utilisons alors le théorème 5.4.3 pour donner une expression de la sortie $\Sigma(v,t)$. Pour cela, on note $a = 1-\pi\sum_{i=1}^{d} a_i\pi^i$ et $b = \sum_{i=0}^{d-1} b_i\pi^i$. Alors, $C\cdot(I-\pi A)^{-1}$ est égale à $\frac{1}{a}\cdot(1,\pi,\ldots,\pi^{d-1})$ et $C\cdot(I-\pi A)^{-1}\cdot B$ est égale à $\frac{b}{a}$. D'où :

$$\Sigma(v,t) = \frac{(1,\pi,\ldots,\pi^{d-1})\cdot(m(t)+\pi c(t))}{a} + \pi\frac{b}{a}\Sigma(u,t) + d_0\cdot\Sigma(u,t) + c'(t)$$

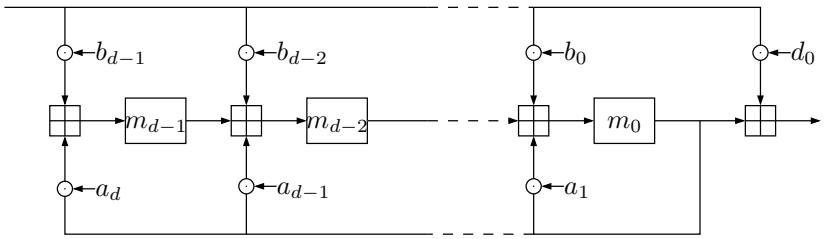


FIGURE 5.2 – Une pale en mode Galois

En particulier, lorsque $m(t) = c(t) = 0$ et $c'(t) = 0$, *i.e.* on considère la pale avec ses mémoires vides, la relation devient :

$$\Sigma(v,t) = \left(d_0 + \pi\frac{b}{a}\right)\Sigma(u,t)$$

Cette expression exprime qu'une pale multiplie son entrée par la fraction $\frac{d_0 \cdot a + \pi \cdot b}{a}$. Cette propriété nous permettra dans le cas des LFSRs et des FCSRs de définir l'automate avec des coefficients rationnels, et de l'implémenter en utilisant des opérations de multiplication dans $\mathcal{S}$ seulement. En particulier, on verra qu'il est intéressant d'utiliser :

- une représentation faisant intervenir des rationnels pour décrire de manière concise un automate ;
- une représentation faisant intervenir des éléments de $\mathcal{S}$ pour décrire l'implémentation de l'automate.

5.8 Troisième discussion

On a vu dans la deuxième discussion sur les automates π-adiques que la représentabilité des éléments π-adiques est un problème. En effet, ces éléments, si on les représente par le développement de Hensel, sont en bijection avec les suites de $\mathcal{S}$. Or, si représenter un élément de $\mathbb{A}_\pi$ est difficile, il est alors encore plus difficile de calculer des combinaisons linéaires à coefficients dans $\mathbb{A}_\pi$. Or, c'est précisément ce que doit faire un AFSR comme défini dans la définition 5.4.1. Pour pallier ce problème, on va se restreindre à des coefficients dans $\mathcal{S}$.

En effet, $\mathcal{S}$ est a priori un « petit » ensemble :

- dans le cas usuel des LFSRs sur $\mathbb{F}_q[[X]]$, c'est un ensemble de q éléments : $\{0,\dots,q-1\}$ généralement ;
- dans le cas des FCSRs sur $\mathbb{Z}_N$, c'est un ensemble de N éléments : $\{0,\dots,N-1\}$ généralement.

De plus, les AFSRs définis avec des matrices à coefficients dans $\mathcal{S}$ permettent de générer l'essentiel des séquences rationnelles. En effet, dans la relation du théorème 5.4.3

$$\Sigma(v,t) = C\cdot(I-\pi A)^{-1}(m(t)+\pi c(t)) + (D + \pi\cdot C\cdot(I-\pi A)^{-1}\cdot B)\cdot\Sigma(u,t) + c'(t)$$

on remarque que $(I-\pi\cdot A)^{-1}$ est une matrice à coefficients de la forme a/b avec $a,b\in\mathbb{A}$. Plus précisément, avec les notations introduites en début de thèse, on a la relation

$$(I-\pi\cdot A)^{-1} = \frac{1}{\det(I-\pi\cdot A)}\operatorname{Adj}(I-\pi\cdot A)$$

En particulier, ici, puisque $(I-\pi\cdot A)$ est une matrice à coefficients dans $\mathbb{A}$ (puisque $\mathcal{S}\subset\mathbb{A}$) :

- $\operatorname{Adj}(I-\pi\cdot A)$ est à valeur dans $\mathbb{A}$,
- $\det(I-\pi\cdot A)$ appartient aussi à $\mathbb{A}$.

Plus précisément, puisque $\det(I - \pi \cdot A) \equiv \det(I) \equiv 1[\pi]$, $(I - \pi \cdot A)^{-1}$ est à coefficients dans

$$\mathcal{Q} = \left\{ \frac{a}{b},\ a, b \in \mathbb{A} \text{ avec } b \equiv 1[\pi] \right\} \subseteq \mathbb{A}$$

Ainsi, la relation du théorème 5.4.3 exprime que les sorties d'un AFSR sont des combinaisons linéaires à coefficients de la forme a/b, avec $a, b \in \mathbb{A}$, de l'état interne et des entrées. Plus précisément :

$$\Sigma(v,t) = \frac{1}{\det(I - \pi \cdot A)} \cdot C \cdot \mathrm{Adj}(I - \pi A)\Big((m(t) + \pi c(t) + \pi \cdot B \cdot \Sigma(u,t)\Big) + D \cdot \Sigma(u,t) + c'(t)$$

Les automates à coefficients dans $\mathcal{S}$ permettent donc de générer les séquences de la forme a/b avec $a, b \in \mathbb{A}$ et $b \equiv 1[\pi]$. Le déterminant $\det(I - \pi \cdot A)$ caractérisant les séquences produites, on le nomme :

Définition 5.8.1

On appelle ***entier de connexion*** *le déterminant* $\det(I - \pi \cdot A)$ *dans le cas où* A *est à valeur dans* $\mathcal{S}$.

5.9 Conclusion

Nous avons présenté dans ce chapitre les AFSRs avec une approche basée sur les automates linéaires. Cette nouvelle approche permet de considérer des AFSRs avec des entrées et de multiples sorties. Nous avons aussi donné une expression explicite des séquences produites par ces objets.

Cependant, le caractère théorique des AFSRs pose rapidement problème pour décrire des propriétés plus profondes. En effet, comme discuté dans le chapitre 2, les anneaux π-adiques peuvent être très différents, et ne permettent pas une approche en profondeur sans faire de nombreuses hypothèses.

Nous avons cependant aussi présenté les pales en mode Galois. Ces automates prendront tout leur sens avec les chapitres suivants, dans lesquels on va spécifier les anneaux π-adiques considérés.

Chapitre 6

LFSMs

Sommaire

Dans ce chapitre, nous appliquons l'approche basée sur les AFSRs vu comme automates linéaires aux LFSMs, *i.e.* des AFSRs sur l'anneau π-adique $\mathbb{F}_q[[X]]$ construit avec $\mathbb{A} = \mathbb{F}_q[X]$ et $\pi = X$. Nous choisissons comme système de représentants de $\mathbb{A}/\pi$ $\mathcal{S} = \{0, \ldots, q-1\}$.

Nous présentons une représentation plus concise des LFSMs en utilisant des matrices à coefficients polynomiaux. Nous étudierons aussi les contraintes liées à l'implémentation des LFSRs, d'un point de vue matériel et logiciel. Ces travaux forment un article, actuellement soumis au journal « IEEE-IT ».

6.1 Représentation polynomiale

Nous avons vu dans l'état de l'art, définition 4.1.1, qu'une LFSM est définie par des matrices à coefficients dans $\mathbb{F}_q$. Nous allons généraliser ici cette définition en autorisant les coefficients à appartenir à $\mathbb{F}_q[[X]]$. Pour cela, il suffit d'appliquer la définition 5.4.1 :

Définition 6.1.1

Une LFSM avec k entrées et l sorties est la donnée de :
- *deux matrices A, B, de tailles respectives $n \times n$ et $n \times k$, à coefficients dans $\mathbb{F}_q[[X]]$. Ces matrices servent à définir la fonction de transition de l'automate.*
- *deux matrices C, D, de tailles respectives $l \times n$ et $l \times k$, à coefficients dans $\mathbb{F}_q[[X]]$. Ces matrices servent à définir la fonction d'extraction de l'automate.*

L'état de l'automate à l'instant t est constitué de :
- $m(t) \in \mathcal{M}_{n,1}(\mathbb{F}_q)$;
- $c(t) \in \mathcal{M}_{n,1}(\mathbb{F}_q[[X]])$, *le vecteur de retenues nécessaire à la fonction de transition ;*
- $c'(t) \in \mathcal{M}_{l,1}(\mathbb{F}_q[[X]])$, *le vecteur de retenues nécessaire à la fonction d'extraction.*

De plus, on note :

– $u(t) \in \mathcal{M}_{n,1}(\mathbb{F}_q)$ *les entrées à l'instant* t ;
– $v(t) \in \mathcal{M}_{l,1}(\mathbb{F}_q)$ *les sorties à l'instant* t.

On a alors les relations suivantes :

$$\begin{cases} m(t+1) &= Am(t) + Bu(t) + c(t) \bmod X \\ c(t+1) &= Am(t) + Bu(t) + c(t) \operatorname{div} X \\ v(t) &= Cm(t) + Du(t) + c'(t) \bmod X \\ c'(t+1) &= Cm(t) + Du(t) + c'(t) \operatorname{div} X \end{cases}$$

Comme présenté dans le chapitre 5, certains des automates correspondant à cette description ne sont pas implémentables car ils requièrent une mémoire infinie. On va d'abord considérer la cas (classique) où les matrices définissant la LFSM sont à valeur dans $\mathbb{F}_q$. À partir de là, on utilisera ces automates pour en construire avec des matrices à valeur dans $\mathbb{F}_q[[X]]$.

6.1.1 Automates à valeur dans $\mathbb{F}_q$

Nous considérons ici des LFSMs comme définies dans la définition 6.1.1, et avec la contrainte supplémentaire que A, B, C, D ont leurs coefficients dans $\mathbb{F}_q$. Ceci correspond au cas décrit dans l'état de l'art.

Remarquons tout d'abord que, pour $a, b \in F_q$, $a + b$ et $a \cdot b$ sont dans $\mathbb{F}_q$. Cela signifie que les opérations arithmétiques s'effectuent sans propagation de retenue. En particulier, dans la définition 6.1.1, si les vecteurs de retenues c et c' sont nuls, alors ils resteront toujours nuls. En effet, les expressions $Am(t) + Bu(t) + c(t)$ et $Cm(t) + Du(t) + c'(t)$ utilisées pour décrire la mise-à-jour et la fonction d'extraction d'une LFSM sont alors égales à des vecteurs à valeur dans $\mathbb{F}_q$. Nous considérons donc nuls les vecteurs c et c'.

Tout d'abord, nous allons utiliser le théorème 5.4.3 pour exprimer le lien qu'il existe entre les entrées, l'état et les sorties d'une LFSM. Cette approche est celle décrite dans la section 5.8. On a :

$$\Sigma(v, t) = C \cdot (I - X \cdot A)^{-1} m(t) + (D + X \cdot C \cdot (I - X \cdot A)^{-1} \cdot B) \cdot \Sigma(u, t)$$

On note $Q(X) = \det(I - X \cdot A)$ le polynôme de connexion de la LFSM. Alors :

– $C \cdot (I - X \cdot A)^{-1}$ est une matrice $l \times n$ dont les coefficients sont de la forme $P_{i,j}(X)/Q(X)$ avec $\deg P_{i,j} < n$ par définition de la matrice adjointe. D'où, le terme $C \cdot (I - X \cdot A)^{-1} m(t)$ est un vecteur colonne de l éléments de la forme

$$C \cdot (I - X \cdot A)^{-1} m(t) =: \begin{pmatrix} P_0(t)(X)/Q(X) \\ \vdots \\ P_{l-1}(t)(X)/Q(X) \end{pmatrix}$$

avec, pour $0 \leq i < l$, $\deg P_i(t)(X) < n$.

– De même, $(D + X \cdot C \cdot (I - X \cdot A)^{-1} \cdot B)$ est une matrice $l \times k$ de la forme $R_{i,j}(X)/Q(X)$ avec $\deg R_{i,j} \leq n$.

Ceci mène à une représentation polynomiale des LFSMs. Le lien entre les entrées et les sorties d'une LFSM est caractérisé par :

– La matrice $(R_{i,j}(X)/Q(X))_{0 \leq i < l, 0 \leq j < k}$ qui décrit la transformation linéaire appliquée aux entrées ;
– Le vecteur $(P_0(t)(X), \ldots, P_{l-1}(t)(X))^t$ qui représente l'état interne de la LFSM.

Proposition 6.1.2

Soit une LFSM de taille n, avec k entrées et l sorties, définie par les matrices A, B, C, D à coefficients dans $\mathbb{F}_q$. On note :

– $Q(X) := \det(I - X \cdot A)$ *son polynôme de connexion,*
– $P(t)(X) := C \cdot \mathrm{Adj}(I - X \cdot A) \cdot m(t)$,
– $R := (D + X \cdot C \cdot (I - X \cdot A)^{-1} \cdot B)$

Alors, la relation entre les entrées et la sortie est :

$$\Sigma(v,t) = \frac{P(t)(X)}{Q(X)} + R \cdot \Sigma(u,t)$$

Cette proposition permet de voir l'intérêt de l'approche polynomiale : on ne se soucie pas de la façon dont les séquences sont calculées par l'automate. En particulier, il existe différents automates qui possèdent la même relation entre les entrées et les sorties.

De plus, cette proposition décrit le fait que les automates, définis par des matrices à coefficients dans $\mathbb{F}_q$ et vus comme des automates sur $\mathbb{F}_q[[X]]$, appliquent une relation affine à coefficients rationnels à leurs entrées.

6.1.2 Contrainte sous-jacente

Nous reconsidérons maintenant la définition 6.1.1. Cette définition permet de décrire des automates non implémentables car les coefficients de A, B, C, D, c, c' appartiennent à $\mathbb{F}_q[[X]]$. Or un élément de $\mathbb{F}_q[[X]]$ peut nécessiter une mémoire infinie pour être représenté en mémoire, comme par exemple $\sum_{i \in \mathbb{N}} X^{2^i}$. Pour éviter ces cas, on doit se restreindre à un sous-ensemble de $\mathbb{F}_q[[X]]$. L'ensemble qui semble naturel à considérer est celui des séries rationnelles :

$$\mathcal{Q} = \left\{ \frac{P(X)}{Q(X)} \in \mathbb{F}_q[[X]],\ P(X), Q(X) \in \mathbb{F}_q[X], PGCD(P(X), Q(X)) = 1, Q(0) = 1 \right\}$$

En effet, la proposition 6.1.2 montre que les automates à coefficients dans $\mathbb{F}_q$ peuvent produire de tels éléments.

6.1.3 Intérêt

Comme nous venons de le voir, les automates définis par des matrices à coefficients dans $\mathcal{S}$ suffisent à construire les LFSRs générant des séquences rationnelles, en particulier, les *m-sequences*. La représentation polynomiale ne permet donc pas d'accéder à de nouveaux automates, et n'est donc pas intéressante d'un point de vue constructif. Toutefois, elle permet de décrire les automates de manière concise, et a donc un intérêt d'un point de vue descriptif.

Cependant, cette représentation peut poser problème lorsque l'on souhaite implémenter l'automate décrit. On considère pour s'en convaincre la LFSM $\mathcal{L}^1$ autonome définie par les matrices

$$A = \begin{pmatrix} \frac{X^2}{X^3+1} & \frac{X}{X^2+X+1} \\ 1 & 0 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Nous calculons $(I - X \cdot A)^{-1}$ pour caractériser les séquences produites :

$$(I - X \cdot A)^{-1} = \begin{pmatrix} \frac{X^3+1}{X^4+X^3+1} & \frac{X^3+X^2}{X^4+X^3+1} \\ \frac{X^4+X}{X^4+X^3+1} & \frac{1}{X^4+X^3+1} \end{pmatrix}$$

Une première idée pour implémenter cet automate est d'utiliser des pales en mode Galois pour construire chaque LFSM décrite par les coefficients de A. Cette réalisation est présentée dans la figure 6.1.

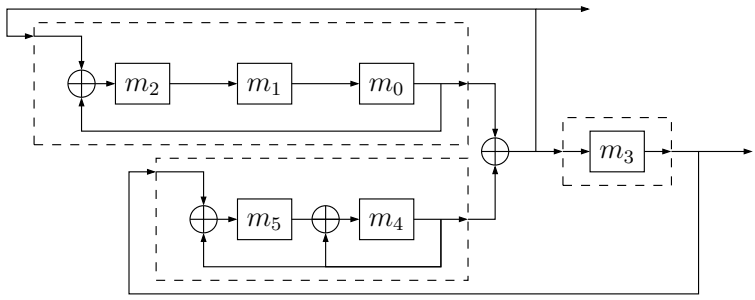


FIGURE 6.1 – Première implémentation de $\mathcal{L}^1$.

Cette implémentation n'est pas optimale car elle requiert six cellules alors que quatre peuvent suffire (par exemple en utilisant un LFSR en mode Galois), puisque l'automate produit des séquences de la forme $P(X)/(X^4+X^3+1)$ avec $\deg P \leq 3$.

Il est possible de construire une meilleure implémentation en construisant l'automate avec une pale en mode Galois pour chaque ligne de A. Pour cela, il suffit de réécrire les fractions $\dfrac{X^2}{X^3+1}$ et $\dfrac{X}{X^2+X+1}$ avec le même dénominateur : $\dfrac{X}{X^2+X+1} = \dfrac{X^2+X}{X^3+1}$. Cela donne l'implémentation présentée dans la figure 6.2. Cela donne une implémentation avec quatre cellules, ce qui est

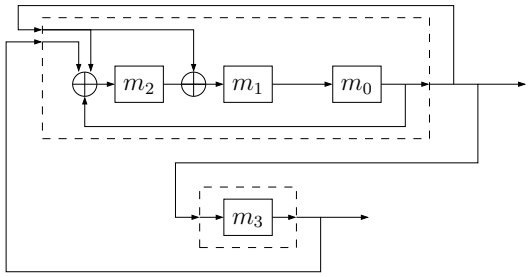


FIGURE 6.2 – Deuxième implémentation de $\mathcal{L}^1$.

optimal ici.

Cependant, le fait de considérer une pale en mode Galois par ligne ne fournit pas toujours une implémentation optimale. Soit la LFSM autonome $\mathcal{L}^2$ définie par les matrices :

$$A = \begin{pmatrix} \dfrac{X+1}{X^3+X+1} & \dfrac{X}{X^2+X+1} & 0 \\ X^3+X^2 & X^2 & 1 \\ 0 & \dfrac{X+1}{X^2+X+1} & 0 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La figure 6.3 présente l'implémentation construite en considérant une pale en mode Galois pour chaque coefficient de A.

Comme précédemment, cette implémentation n'est pas optimale car elle requiert quinze mémoires, alors que neuf suffisent. En effet, les séquences rationnelles produites ont pour dénominateur $Q(X) = X^9+X^7+X^6+X^2+X+1$.

On construit donc une nouvelle implémentation en utilisant les dénominateurs communs sur chaque ligne de A. Cela conduit à l'implémentation présentée dans le figure 6.4.

Cette implémentation n'est toujours pas optimale car elle requiert onze mémoires au lieu de neuf. Cela vient du fait que dans la matrice A, deux coefficients avec le même dénominateur apparaissent dans la même colonne : $\dfrac{X}{X^2+X+1}$ et $\dfrac{X+1}{X^2+X+1}$.

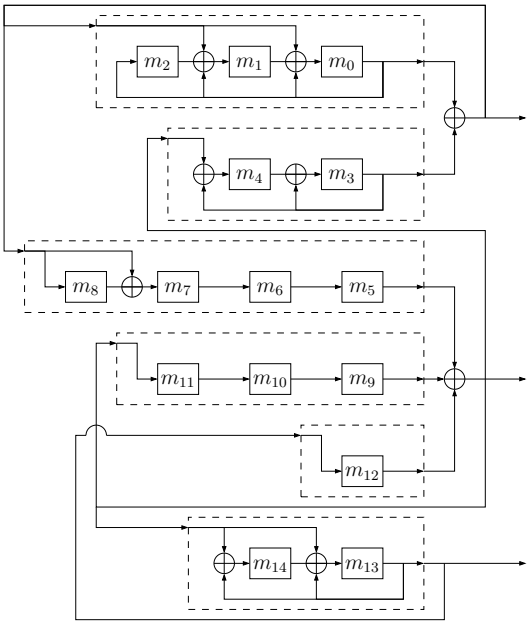


FIGURE 6.3 – Première implémentation de $\mathcal{L}^2$.

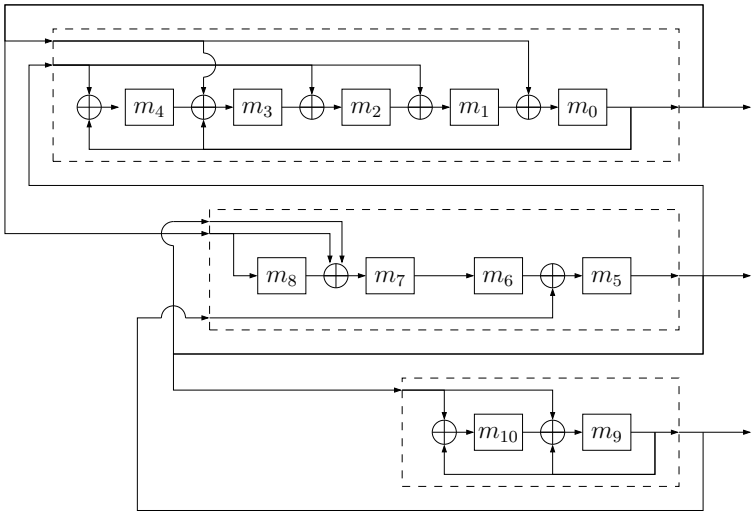


FIGURE 6.4 – Deuxième implémentation de $\mathcal{L}^2$.

En d'autres termes, la représentation polynomiale pose problème lorsque l'on souhaite implémenter la LFSM décrite. Une condition (forte) pour que l'implémentation soit aisément réalisable, est qu'un seul coefficient soit non-nul sur chaque ligne et chaque colonne de A.

6.1.4 Application aux *windmill* LFSRs

Dans [SC88], les *windmill* LFSRs sont définis comme étant une LFSM composée de $v \geq 1$ LFSMs connectées pour former un cycle. Chaque LFSM est appelée une pale du *windmill*. Nous choisissons de présenter les pales en mode Galois, comme présenté dans la figure 5.2.

Les *windmill* LFSRs sont construits pour générer une séquence en parallèle : Soit une séquence $S = (s_n)_{n\in\mathbb{N}}$.

- Un automate classique la produira caractère par caractère : d'abord s_0, puis s_1, etc.
- Un automate parallèle produira lui v caractères à la fois : d'abord $(s_0, s_1, \ldots, s_{v-1})$, puis

$(s_v, \ldots, s_{2v-1})$, etc.
Plus précisément, un automate parallèle est un automate avec v sorties et qui produit les séquences $s_{v,i}(s)$ pour $0 \leq i < v$, *i.e.* des séquences décimées de s. Avec notre approche, nous caractériserons la production de ces séquences décimées, et non la séquence reconstruite.

On considère comme exemple le *windmill* présenté dans le figure 6.5 qui est utilisé dans le chiffrement à flot E0 [Blu01]. Il est constitué d'une pale de longueur 7 et de trois pales identiques de longueur 6.

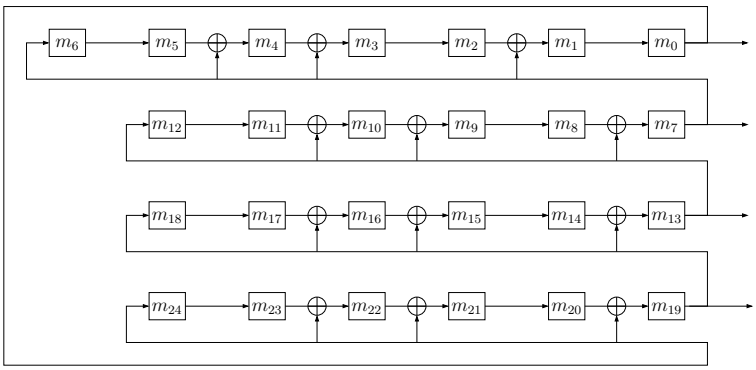


FIGURE 6.5 – Un *windmill* de E0

Jusqu'à maintenant, les *windmill* LFSRs étaient construits en utilisant des pales identiques. Avec l'approche polynomiale, nous allons donner une description plus compacte des *windmill*. De plus, cette approche permettra de considérer des pales différentes lors de l'implémentation. Dans le cas du *windmill* présentée dans la figure 6.5, nous nous intéressons aux séquences générées par les cellules m_0, m_7, m_{13} and m_{19}. En appliquant la proposition 6.1.2, nous pouvons donner une interprétation polynomiale de chaque pale. Cela conduit à l'interprétation présentée dans la figure 6.6.

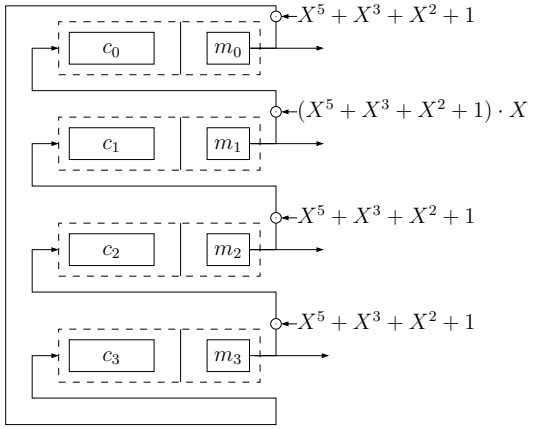


FIGURE 6.6 – Représentation polynomiale du *windmill* de la figure 6.5

Avec cette approche, ce *windmill* a la matrice de transition suivante :

$$X^5 + X^3 + X^2 + 1 \cdot \begin{pmatrix} 0 & X & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Nous donnons dans la table 6.1, les valeurs de $m(t)$ et $c(t)$ durant 9 cycles.

t	$m_0(t)$	$m_1(t)$	$m_2(t)$	$m_3(t)$	$c_0(t)$	$c_1(t)$	$c_2(t)$	$c_3(t)$
0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	X^4+X^2+X
2	0	0	1	0	0	0	X^4+X^2+X	X^3+X+1
3	0	1	0	1	0	X^4+X^2+X	X^3+X+1	X^2+1
4	0	0	0	1	$X^5+X^3+X^2+1$	X^3+X+1	X^4+X+1	X
5	1	1	0	0	X^4+X^2+X	X^2+1	$X^4+X^3+X^2+X+1$	1
6	0	1	1	0	X^5+X^2+X	X	X^3+X^2+X+1	X^4+X^2+X
7	0	1	1	0	$X^5+X^4+X^3+X^2+X$	X^4+X^2+X+1	X^2+X+1	X^3+X+1
8	0	0	1	1	X^5+X^4+X	$X^4+X^3+X^2+1$	$X+1$	X^2+1

TABLE 6.1 – État du *windmill* de la figure 6.6 durant 9 cycles.

Formellement, les *windmill* LFSRs définis par Smeets et Chambers dans [SC88] correspondent à la définition suivante :

Définition 6.1.3

Un windmill *LFSR défini par les polynômes $\alpha(X), \beta(X)$ avec $\beta(0) \neq 0$ et v pales est une LFSM de longueur v avec pour matrice de transition sur $\mathbb{F}_q[[X]]$:*

$$\begin{pmatrix} 0 & \frac{\alpha(X)}{\beta(X)} \cdot X^{i_0} & & (0) \\ \vdots & \ddots & \ddots & \\ 0 & (0) & \ddots & \frac{\alpha(X)}{\beta(X)} \cdot X^{i_{v-2}} \\ \frac{\alpha(X)}{\beta(X)} \cdot X^{i_{v-1}} & 0 & \dots & 0 \end{pmatrix}$$

où $0 \leq i_0, \dots, i_{v-1}$.

Avec cette représentation, chaque ligne représente une pale du *windmill*. En particulier, la longueur de la pale j est égale à $\max(\deg(\alpha(X) \cdot X^{i_j}), \deg(\beta(X)))$. Ainsi, en utilisant cette représentation, nous avons généralisé la notion de *windmill* LFSRs en utilisant une définition polynomiale de l'automate.

6.2 Implémentation

Dans cette section, nous nous intéressons aux implémentations des LFSMs sans entrée. En effet, comme présenté dans les chapitres 1 et 4, ces automates sont largement utilisés en cryptographie, et par conséquent, ont été très étudiés [GW82, Rog89, SC88, GK09, PL05, Lau09]. Nous utilisons la description classique des LFSMs, *i.e.* avec des matrices à coefficients dans $\mathbb{F}_q$, pour construire des LFSMs autonomes possédant une implémentation efficace matérielle et/ou logicielle. Nous présentons d'abord un nouveau concept, la diffusion, qui sera un des critères pour le design des LFSMs.

6.2.1 Diffusion

Nous avons déjà vu dans le chapitre 2 que les *m-sequences* ont de bonnes propriétés statistiques. Nous construirons donc des automates produisant de telles séquences.

Nous présentons maintenant un critère lié à l'implémentation de l'automate. Ce paramètre mesure le temps nécessaire à l'automate pour « mélanger » le contenu de ses mémoires. Plus

précisément, il correspond au nombre minimal de cycles nécessaires pour que chaque mémoire ait été influencée par toutes les autres.

Définition 6.2.1

Soit $\mathcal{L}$ une LFSM avec matrice de transition A de taille n. On note $\mathcal{G}$ le graphe défini par la matrice d'adjacence A^t, i.e. :

- *$\mathcal{G}$ possède n sommets nommés $0, 1, \ldots, n-1$. Ces sommets représentent les cellules de l'automate.*
- *si $a_{i,j} \neq 0$, alors il existe une arête orientée du sommet j au sommet i. Les arêtes représentent les liens entre les cellules.*

*Nous définissons la **diffusion** comme étant égale au diamètre du graphe $\mathcal{G}$, i.e. la plus grande des distances entre deux sommets de $\mathcal{G}$.*

Plus la diffusion sera petite, meilleur sera l'automate. En effet, ce paramètre mesure le temps nécessaire à l'automate pour qu'une différence sur une cellule se répande sur toutes les autres cellules. Plus précisément, on considère une LFSM de taille n avec une diffusion de δ. Remplacer la valeur de la cellule m_i au temps t par $m_i(t) + e$, avec $e \in \mathbb{F}_q$, aura une influence sur chaque cellule après au plus δ cycles. En d'autres termes, la diffusion mesure aussi la corrélation entre les cellules : après δ cycles, le comportement de chaque cellule est corrélé à toutes les autres cellules.

Par exemple, les LFSRs en mode Galois, Fibonacci (figure 4.2) et les automates cellulaires (figure 4.3) de taille n ont une diffusion de $n-1$, car la cellule m_{n-1} requiert $n-1$ cycles pour influencer m_0. Autrement, les *ring* LFSRs peuvent avoir une diffusion plus basse car le graphe qui leur est associé est plus proche d'un graphe aléatoire. Or le diamètre d'un graphe aléatoire à n sommets est en moyenne $\sqrt{n}$. Cependant, cette valeur n'est qu'une moyenne qui n'est pas toujours accessible, en particulier car nous chercherons des *ring* LFSRs avec des matrices de transition creuses, *i.e.* le graphe associé aura peu d'arêtes.

Ce paramètre est important en cryptographie où les différences doivent avoir un large impact sur le comportement d'un automate. Il peut aussi être utilisé pour diminuer le *dimension gap* dans le cas des générateurs pseudo-aléatoires comme présenté dans [L'E96, PL05].

6.2.2 Implémentation matérielle

Nous présentons dans cette section une méthode pour construire un automate avec une implémentation matérielle efficace, *i.e.* respectant les contraintes présentées dans le chapitre 3 :

- avec un chemin critique le plus court possible ;
- avec un *fan-out* minimal ;
- avec un coût en terme de portes logiques faible.

On rappelle que baisser ces différents valeurs permet d'augmenter la fréquence du circuit, et donc d'augmenter le débit de la suite produite. Nous présentons nos résultats dans le cas des automates binaires, *i.e.* $\mathbb{F}_q = \mathbb{F}_2$, car c'est le cas usuel. Mais ces approches se généralisent à $\mathbb{F}_q$.

Des méthodes efficaces existent déjà pour baisser ces valeurs. En particulier, comme présenté dans le chapitre 4, l'article [MRT04] permet à partir d'un LFSR donné de construire un LFSR équivalent avec un chemin critique et un *fan-out* plus bas. Cependant, cette méthode ne permet pas de diminuer la diffusion de l'automate : étant donné un LFSR en mode Galois de taille n, la méthode conduit à un LFSR avec une diffusion de $n-1$, ce qui est la valeur maximale, *i.e.* le pire cas.

De plus, cette méthode s'attache à transformer un LFSR existant pour trouver une meilleure implémentation sans changer son polynôme de connexion. Or dans la plupart des applications, on ne cherche pas à ce que le polynôme de connexion ait une valeur précise, seulement qu'il soit primitif. C'est le cœur de notre approche : nous allons tirer aléatoirement des matrices de

	Galois	Fibonacci	Automate cellulaire	[MRT04]	*Ring* LFSR
Chemin critique	1	$\lceil \log_2(w-1) \rceil$	2	≤ 2	$\max\lceil \log_2(w_H(Row_i)) \rceil$
Fan-out	$w-1$	2	3	≤ 3	$\max w_H(Col_i)$
Coût	$w-2$	$w-2$	$\in [n-1; 3(n-1)]$	$\leq w-2$	$w_H(A)-n$
Diffusion	$n-1$	$n-1$	$n-1$	$n-1$	$\leq n-1$

TABLE 6.2 – Chemin critique, *fan-out*, coût et diffusion de différents LFSRs.

	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$
Chemin critique	1	2	1
Fan-out	4	2	2
Coût	3	3	2
Diffusion	7	7	6

TABLE 6.3 – Chemin critique, *fan-out*, coût et diffusion des LFSRs présentés dans le figure 4.5

transition associées à une implémentation efficace, et nous vérifierons ensuite que le polynôme de connexion est primitif. Pour cela, nous devons d'abord exprimer les contraintes sur le chemin critique, le *fan-out* et le coût d'un automate en terme de matrice de transition d'un *ring* LFSR.

Le tableau 6.2 rassemble ces contraintes avec les notations suivantes : soit $\mathcal{L}$ un *ring* LFSR de taille n avec A pour matrice de transition avec $\det(A) \neq 0$ (cela assure que son polynôme de connexion est de degré n) :

- On note $Q(X)$ son polynôme de connexion, et w le nombre de coefficients non-nuls de $Q(X)$: $w := w_H(Q(X))$.
- On note $Col_0, \ldots, Col_{n-1}$ les colonnes de A.
- On note $Row_0, \ldots, Row_{n-1}$ les lignes de A.

On présente aussi dans ce tableau les valeurs atteintes en appliquant la méthode présentée dans [MRT04].

Les LFSRs en mode Galois sont optimaux pour le chemin critique ; ceux en mode Fibonacci le sont pour le *fan-out*. Un *ring* LFSR peut être construit pour atteindre ces deux valeurs. Plus précisément, un *ring* LFSR avec un poids de Hamming de au plus 2 pour chaque ligne et chaque colonne de sa matrice de transition sera optimal. Nous donnons dans la table 6.3 les valeurs associées aux LFSRs présentés dans la figure 4.5. En particulier, le *ring* LFSR $\mathcal{L}_2$ est meilleur que les LFSRs en mode Galois et Fibonacci, et ce pour tous les critères.

Cependant, comme annoncé, nous allons tirer aléatoirement des matrices respectant les contraintes données. En particulier, les matrices considérées ne présenteront pas de structure particulière, nous ne pourrons donc pas calculer le polynôme de connexion autrement qu'avec les algorithmes classiques.

L'algorithme II.4 tire aléatoirement des positions de rétroaction et calcule le polynôme de connexion correspondant. Cet algorithme est probabiliste. Nous supposons que tirer une matrice aléatoire de taille n et calculer son polynôme de connexion est équivalent à tirer aléatoirement un polynôme de degré n. Plus précisément, nous savons que le polynôme de connexion a son terme de plus haut degré et son terme constant égaux à 1, donc le nombre de polynômes que l'on peut construire avec l'algorithme est 2^{n-2}. Le nombre de polynômes primitifs de degré n sur $\mathbb{F}_2$ est $\dfrac{\varphi(2^n-1)}{n}$ où φ est la fonction d'Euler. L'algorithme II.4 finira donc en moyenne après $\dfrac{2^{n-2}}{\varphi(2^n-1)/n}$ essais. Cette valeur théorique semble valide au regard de nos tests (figure 6.7).

Algorithme II.4 : Tirage aléatoire de *ring* LFSR respectant les contraintes matérielles.

Entrées : n la longueur du LFSR. $f \leq n$ le nombre de rétroactions à placer.
Sorties : Une matrice de transition avec un chemin critique de 1, un *fan-out* de 2 et un coût de f, et telle que le polynôme de connexion est primitif de degré n

début
  répéter
    $A \leftarrow (a_{i,j})_{0 \leq i,j < n}$ avec $a_{i,j} = \begin{cases} 1 \text{ if } j \equiv i+1[n] \\ 0 \text{ sinon} \end{cases}$;
    tant que $w_H(A) < n + f$ **faire**
      $(i,j) \leftarrow Random([0,n] \times [0,n])$;
      si $w_H(Row_i) = 1$ *et* $w_H(Col_j) = 1$ **alors**
        $a_{i,j} \leftarrow 1$;
  jusqu'à $Q(X)$ *est primitif* ;
  retourner A;
fin

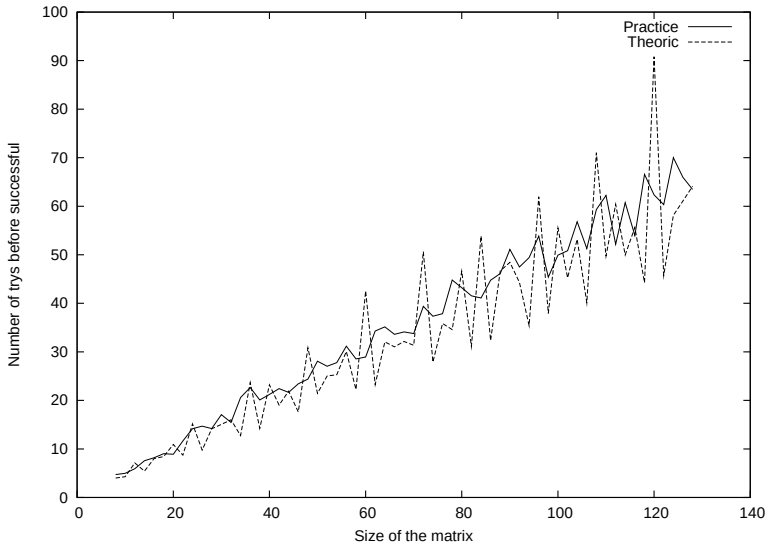


FIGURE 6.7 – Nombres d'essais théorique et empirique nécessaires pour l'algorithme II.4.

Le temps nécessaire à cet algorithme est dominé par le temps de calcul du polynôme de connexion $\det(I - X \cdot A)$. Ce calcul a une complexité en $\mathcal{O}(n^3)$. Or, l'algorithme II.4 place les rétroactions une par une. Nous allons utiliser cette propriété pour diminuer la complexité de l'algorithme. Nous rappelons d'abord la propriété suivante liée à la matrice des cofacteurs :

Proposition 6.2.2

Soit A une matrice sur un anneau $\mathbb{A}$ de taille $n \times n$. On note $E_{i,j}$ la matrice constituée d'un unique 1 en position i,j. Alors, pour $\lambda \in \mathbb{A}$, $\det(A + \lambda E_{i,j}) = \det(A) + \lambda Cof_{i,j}$ où $Cof_{i,j}$ est le cofacteur i,j de la matrice A

De plus, la matrice des cofacteurs est égale à la transposée de la matrice adjointe, que l'on calcule avec un algorithme d'inversion. On utilise cette proposition pour améliorer la complexité de l'algorithme, comme présenté par l'algorithme II.5. L'idée est que, au lieu de placer les f rétroactions, de calculer $Q(X)$, puis de jeter la matrice si celui-ci n'est pas primitif, ici on place $f-1$ rétroactions, puis on teste tous les polynômes de connexion construits en plaçant la dernière rétroaction.

Algorithme II.5 : Tirage aléatoire de *ring* LFSR respectant les contraintes matérielles.

Entrées : n la longueur du LFSR. $f \leq n$ le nombre de rétroactions à placer.
Sorties : Une matrice de transition avec un chemin critique de 1, un *fan-out* de 2 et un coût de f, et telle que le polynôme de connexion est primitif de degré n

début
  Répéter
    $A \leftarrow (a_{i,j})_{0 \leq i,j < n}$ avec $a_{i,j} = \begin{cases} 1 \text{ if } j \equiv i+1[n] \\ 0 \text{ sinon} \end{cases}$;
    tant que $w_H(A) < n + f - 1$ **faire**
      $(i,j) \leftarrow Random([0,n] \times [0,n])$;
      si $w_H(Row_i) = 1$ *et* $w_H(Col_j) = 1$ **alors**
        $a_{i,j} \leftarrow 1$;
    $C \leftarrow$ Matrice des cofacteurs de $I - X \cdot A$;
    $Q_0(X) \leftarrow \det(I - X \cdot A)$;
    pour $0 \leq i,j < n$ **faire**
      si $w_H(Row_i) = 1$ *et* $w_H(Col_j) = 1$ *et* $a_{i,j} = 0$ **alors**
        $Q(X) \leftarrow Q_0(X) - X \cdot C_{i,j}$;
        si $Q(X)$ *est primitif* **alors**
          retourner $A + E_{i,j}$;
fin

La complexité de cet algorithme est dominé par le calcul de la matrice des cofacteurs et du déterminant. Ces deux opérations, avec les algorithmes classiques, sont calculées simultanément. Mais maintenant, pour une matrice de cofacteur calculée, nous testons $n^2 - n \cdot f$ polynômes. La complexité en moyenne de cet algorithme est donc de $\mathcal{O}(n)$.

6.2.3 Implémentation logicielle

Nous venons de traiter le cas des implémentations matérielles. Nous nous intéressons maintenant aux implémentations logicielles. On a vu dans le chapitre 3 que les contraintes dans ce cas sont :

- travailler sur des données de la longueur des mots du processeur : 8, 16, 32, 64, 128 ou 256 bits.
- utiliser des opérations rapides sur un mot.

Comme pour le cas matériel, notre approche se base sur la construction de matrice de transition d'une LFSM dont l'implémentation logicielle sera efficace. Pour cela, nous considérons des matrices par blocs. En effet, en prenant des matrices avec des blocs de taille w égale à la taille des mots processeurs, les données seront traitées par mot entier. Un LFSR possédant une matrice de transition définie par bloc sera appelé **LFSR sur des mots**.

De plus, nous utiliserons les opérations de décalage à droite et à gauche (notées $\gg$ et $\ll$), car elles sont rapides et implémentées sur des mots. Étant donnés des mots de taille w, nous notons L la matrice de décalage à gauche, *i.e.* L est une matrice $w \times w$ avec sa sur-diagonale remplie de 1. De même, nous notons R la matrice de décalage à droite, *i.e.* R est une matrice $w \times w$ avec sa sous-diagonale remplie de 1. On a alors :

$$\begin{aligned} L \cdot (x_0, x_1, \ldots, x_{w-1})^t &= (x_1, \ldots, x_{w-1}, 0)^t \\ R \cdot (x_0, x_1, \ldots, x_{w-1})^t &= (0, x_0, x_1, \ldots, x_{w-2})^t \end{aligned}$$

Nous présentons l'algorithme II.6 qui découle de la discussion précédente.

Algorithme II.6 : Tirage aléatoire de *ring* LFSR respectant les contraintes logicielles.

Entrées : w la taille des mots. n la longueur du LFSR avec $w|n$. $f \leq n/k$ le nombre de rétroactions (sur des mots) à placer.

Sorties : Une matrice de transition définie par blocs avec une implémentation nécessitant f décalages et f `OU exclusif`, et telle que le polynôme de connexion est primitif de degré n

début

  répéter

    $A \leftarrow (a_{i,j})_{0 \leq i,j < n/w}$ avec $a_{i,j} = \begin{cases} I_k \text{ if } j \equiv i+1 [n/w] \\ 0 \text{ sinon} \end{cases}$;

    $From \leftarrow Random([0, n/w]^f)$;

    $To \leftarrow Random([0, n/w]^f)$;

    $Shift \leftarrow Random\left(([-w/2, w/2] \setminus \{0\})^f\right)$;

    pour $0 \leq l \leq f-1$ **faire**

      $a_{To[l],From[l]} \leftarrow a_{To[l],From[l]} + \begin{cases} L^{Shift[l]} \text{ if } Shift[l] > 0 \\ R^{-Shift[l]} \text{ otherwise} \end{cases}$;

    $Q(X) \leftarrow \det(I - X \cdot A)$

  jusqu'à $Q(X)$ *est primitif* ;

  retourner A;

fin

Cet algorithme tire aléatoirement la position des rétroactions ainsi que la valeur du décalage associé, et calcule le polynôme de connexion associé. La complexité de cet algorithme est comparable à celle de l'algorithme II.4, car nous ne sommes pas capables d'utiliser la structure par bloc de la matrice pour diminuer le coût du calcul du déterminant.

L'algorithme II.6 renvoie une LFSM qui remplit les contraintes logicielles, mais pas matérielles. Cependant, il est facile de combiner toutes ces contraintes pour construire un automate avec des implémentations logicielles et matérielles efficaces. Il suffit de ne placer qu'une seule rétroaction sur des mots par ligne et par colonne.

Nous donnons dans la figure 6.8 un exemple de LFSR avec une implémentation logicielle efficace avec $n = 40$ et $w = 8$, et un polynôme de connexion primitif (produisant donc des *m-sequences*). L'implémentation matérielle de ce LFSR est aussi bonne (*fan-out* de 2, chemin critique de 1 et coût de 19 portes logiques) car il respecte aussi les contraintes matérielles. Sa diffusion est de 27.

Nous comparons maintenant un LFSR sur des mots avec le LFSR utilisé dans SNOW2.0 défini dans [EJ02].

Exemple de LFSR sur des mots de taille 512 bits

Nous donnons dans la figure 6.9 la matrice de transition d'un LFSR de taille 512 bits construits sur des mots de 32 bits. Cet LFSR produit des *m-sequences* car son polynôme de connexion est primitif.

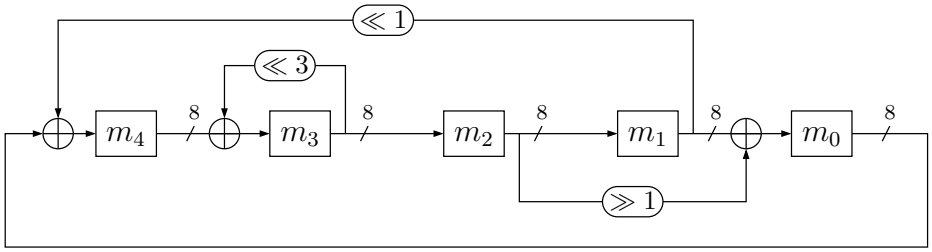
Description du LFSR de SNOW2.0

Nous donnons ici une description du LFSR utilisé dans SNOW2.0 [EJ02] vu sur $\mathbb{F}_2$.

Ce LFSR est un LFSR en mode Fibonacci sur $\mathbb{F}_{2^{32}}$. Le corps $\mathbb{F}_{2^{32}}$ est défini comme une extension de $\mathbb{F}_{2^8}$ pour une implémentation efficace et pour contrer des attaques [HR02].

$$A = \begin{pmatrix} & I_8 & R^1 & & \\ & & I_8 & & \\ & & & I_8 & \\ & & & L^3 & I_8 \\ I_8 & L^1 & & & \end{pmatrix}$$

(a) Matrice de transition



(b) Représentation

FIGURE 6.8 – Un LFSR avec une implémentation logicielle efficace.

$$\left(\begin{array}{ccccc|ccc|ccc|ccc|c} & I_{32} & & & & & & & & & & & & & \\ & & I_{32} & & & & & R^{14} & & & & & & & \\ & & & I_{32} & & & & & L^8 & & & & & & \\ \hline & & & & & I_{32} & & & & & (0) & L^{12} & & & \\ & & & & & & I_{32} & & & & & & & & \\ & & & & & & & L^2 & I_{32} & & & & & & \\ \hline & & & & & & & I_{32} & & & & & & & \\ & & & & & & & & & I_{32} & & & & & R^{11} \\ L^{13} & & & & & & & & & & I_{32} & & & & \\ \hline & & & & & (0) & & & & & & I_{32} & & & \\ & & & & & & & & & & & & I_{32} & & \\ & & & & & & & & & & & & & I_{32} & \\ \hline & & & & & & & & & & & & & & I_{32} \\ & & & & & & & & & & & R^{13} & & & \\ & & & & & & & & & & & & & I_{32} & \\ \hline I_{32} & & R^{10} & & & & & & & & & & & & \end{array}\right)$$

FIGURE 6.9 – Matrice de transition d'un LFSR sur des mots

L'implémentation est basée sur la multiplication par $\alpha \in \mathbb{F}_{2^{32}}$ vérifiant $\alpha \cdot (c_3\alpha^3 + c_2\alpha^2 + c_1\alpha^1 + c_0) = (c_2\alpha^3 + c_1\alpha^2 + c_0\alpha) + c_3 \cdot V$ avec V un élément de $\mathbb{F}_{2^{32}}$. Nous notons M_α la matrice de cette application linéaire sur $\mathbb{F}_2^{32}$:

$$M_\alpha = \left(\begin{array}{ccc|c|c|c|c} 0 & 0 & 0 & & & & \\ I_8 & & (0) & V_0 & V_1 & \dots & V_7 \\ & I_8 & & & & & \\ (0) & & I_8 & & & & \end{array}\right)$$

où

$$\begin{cases} V_0 = (\texttt{0xE19FCF13})^t \\ V_1 = (\texttt{0x6B973726})^t \\ V_2 = (\texttt{0xD6876E4C})^t \\ V_3 = (\texttt{0x05A7DC98})^t \\ V_4 = (\texttt{0x0AE71199})^t \\ V_5 = (\texttt{0x1467229B})^t \\ V_6 = (\texttt{0x28CE449F})^t \\ V_7 = (\texttt{0x50358897})^t \end{cases}$$

La matrice de transition du LFSR de SNOW2.0 est présentée dans la figure 6.10.

$$\begin{pmatrix} & I_{32} & & & & & & & & & & & & & & \\ & & I_{32} & & & & & & & & & & & & & \\ & & & I_{32} & & & & & & & & & & & & \\ & & & & I_{32} & & & & & & & (0) & & & & \\ & & & & & I_{32} & & & & & & & & & & \\ & & & & & & I_{32} & & & & & & & & & \\ & & & & & & & I_{32} & & & & & & & & \\ & & & & & & & & I_{32} & & & & & & & \\ & & & & & & & & & I_{32} & & & & & & \\ & & & & (0) & & & & & & I_{32} & & & & & \\ & & & & & & & & & & & I_{32} & & & & \\ & & & & & & & & & & & & I_{32} & & & \\ & & & & & & & & & & & & & I_{32} & & \\ & & & & & & & & & & & & & & I_{32} & \\ & & & & & & & & & & & & & & & I_{32} \\ M_\alpha & 0 & I_{32} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (M_\alpha)^{-1} & 0 & 0 & 0 & 0 \end{pmatrix}$$

FIGURE 6.10 – Matrice de transition du LFSR de SNOW2.0

Ces 2 LFSRs produisent des *m-sequences* de degré 512. Nous comparons la diffusion et le débit de ces LFSRs :

- La diffusion pour SNOW2.0 est de 49 ; et de 33 pour le LFSR sur des mots.
- Le coût pour une transition est de 8 cycles pour SNOW2.0 en utilisant l'implémentation basée sur la méthode *sliding window* comme proposée dans [EJ02] (cette technique ne s'applique que aux LFSRs en mode Fibonacci). Avec une implémentation classique, le coût est de 20 cycles. Le coût du LFSR sur des mots est de 33 cycles.

La diffusion est meilleure pour le LFSR sur des mots, mais le coût en cycles est plus élevé. Cela provient du fait que le LFSR de SNOW2.0 est creux (trois rétroactions) tandis que le LFSR sur des mots en a huit. De plus, l'implémentation de SNOW2.0 utilise des tables précalculées pour baisser le coût en cycles. Cependant, le LFSR sur des mots remplit les contraintes pour une bonne implémentation matérielle, alors que le LFSR de SNOW2.0 ne peut pas être implémenté aussi efficacement.

Chapitre 7

FCSRs

Sommaire

Dans ce chapitre, nous appliquons l'approche basée sur les AFSRs, vus comme automates linéaires, aux FCSRs, *i.e.* des AFSRs sur l'anneau π-adique $\mathbb{Z}_N$ construit avec $\mathbb{A} = \mathbb{Z}$ et $\pi = N$. Nous choisissons comme système de représentants de $\mathbb{A}/\pi$ $\mathcal{S} = \{0, \ldots, N-1\}$.

Cette approche permet, comme dans le cas des LFSMs, de considérer des FCSRs définis par des matrices à coefficients dans $\mathbb{Z}_N$. Mais dans ce cas, l'intérêt est double :

- Un FCSR pourra maintenant être défini par des matrices, et non plus seulement en mode Galois et Fibonacci.
- Le fait de permettre aux matrices d'avoir des coefficients dans $\mathbb{Z}_N$ permettra de décrire de manière concise les automates considérés (comme dans le cas des LFSMs), et permettra en particulier d'utiliser des matrices à coefficients dans $\{-1, 0, 1\}$ dont l'implémentation est facile.

Nous étudierons aussi les contraintes liées à l'implémentation des FCSRs, d'un point de vue matériel et logiciel. Ces travaux forment un article, actuellement soumis au journal « *Cryptography and Communications Discrete Structures, Boolean Functions and Sequences* ».

7.1 Approche matricielle

Nous avons vu dans l'état de l'art, qu'un FCSR est un automate autonome associé à un entier de connexion $q = 1 - N\sum_{i=0}^{n-1} d_i N^i$ avec les notations des figures 4.8 et 4.9. Étant donné un entier de connexion q, deux implémentations sont possibles : le mode Galois et le mode Fibonacci. Les séquences produites sont alors de la forme p/q avec $p \in \mathbb{Z}$.

Nous allons généraliser cette approche en considérant les AFSRs définis sur les entiers N-adiques. Pour cela, on applique la définition 5.4.1 :

Définition 7.1.1

Un FCSR avec k entrées et l sorties est la donnée de :

– *deux matrices A, B, de tailles respectives $n \times n$ et $n \times k$, à coefficients dans $\mathbb{Z}_N$. Ces matrices servent à définir la fonction de transition de l'automate.*
– *deux matrices C, D, de tailles respectives $l \times n$ et $l \times k$, à coefficients dans $\mathbb{Z}_N$. Ces matrices servent à définir la fonction d'extraction de l'automate.*

L'état de l'automate à l'instant t est constitué de :
– $m(t) \in \mathcal{M}_{n,1}(\{0, \ldots, N-1\})$;
– $c(t) \in \mathcal{M}_{n,1}(\mathbb{Z}_N)$, *le vecteur de retenues nécessaire à la fonction de transition ;*
– $c'(t) \in \mathcal{M}_{l,1}(\mathbb{Z}_N)$, *le vecteur de retenues nécessaire à la fonction d'extraction.*

De plus, on note :
– $u(t) \in \mathcal{M}_{n,1}(\{0, \ldots, N-1\})$ *les entrées à l'instant t ;*
– $v(t) \in \mathcal{M}_{l,1}(\{0, \ldots, N-1\})$ *les sorties à l'instant t.*

On a alors les relations suivantes :

$$\begin{cases} m(t+1) &= Am(t) + Bu(t) + c(t) \bmod N \\ c(t+1) &= Am(t) + Bu(t) + c(t) \operatorname{div} N \\ v(t) &= Cm(t) + Du(t) + c'(t) \bmod N \\ c'(t+1) &= Cm(t) + Du(t) + c'(t) \operatorname{div} N \end{cases}$$

Avec ces notations, le FCSR en mode Galois associé à l'entier de connexion $q = 1 - N\sum_{i=0}^{n-1} d_i N^i$ est défini par les matrices

$$A = \begin{pmatrix} d_0 & 1 & & & \\ d_1 & & 1 & (0) & \\ \vdots & & (0) & \ddots & \\ d_{n-2} & & & & 1 \\ d_{n-1} & 0 & 0 & \cdots & 0 \end{pmatrix}, B = (0), C = \begin{pmatrix} 1 & 0 & \ldots & 0 \end{pmatrix}, D = (0)$$

De même, le FCSR en mode Fibonacci est défini par les matrices

$$A = \begin{pmatrix} 0 & 1 & & & \\ 0 & & 1 & (0) & \\ \vdots & & (0) & \ddots & \\ 0 & & & & 1 \\ d_{n-1} & d_{n-2} & \cdots & d_1 & d_0 \end{pmatrix}, B = (0), C = \begin{pmatrix} 1 & 0 & \ldots & 0 \end{pmatrix}, D = (0)$$

En particulier, cette définition permet de considérer des FCSRs avec des rétroactions placées entre n'importe quelles cellules, contrairement aux modes Galois et Fibonacci. Cette liberté permettra de décorréler les retenues, et donc de se prémunir contre l'attaque par LFSRization des FCSRs présentée dans le chapitre 4. Cette attaque, ainsi que la résistance des FCSRs en mode ni Galois ni Fibonacci, est détaillée dans le chapitre 8.

Cependant, comme présenté dans les chapitres 5 et 6, certains des automates décrits par la définition 7.1.1 ne sont pas implémentables car ils requièrent une mémoire infinie. Nous allons d'abord traiter le cas des matrices à coefficients dans $\{0, \ldots, N-1\}$, avant de considérer des matrices à coefficients dans $\mathbb{Z}_N$.

7.1.1 Automates à valeur dans $\{0, \ldots, N-1\}$

Nous appliquons ici l'approche décrite dans la section 5.8, *i.e.* nous considérons des FCSRs définis avec des matrices à coefficients dans $\{0, \ldots, N-1\}$.

Étudions dans ce cas les opérations élémentaires présentées dans la section 5.5. Nous nous intéressons plus particulièrement aux valeurs prises par les retenues. Soient $\alpha, \beta \in \mathbb{Z}_N$, avec $\text{seq}(\alpha) = (a(t))_{t\in\mathbb{N}}$ et $\text{seq}(\beta) = (b(t))_{t\in\mathbb{N}}$:

- Lors des itérations de l'algorithme II.2, la retenue $c(t)$ appartient à $\{0, 1\}$. En effet, dans ce cas, $0 \leq a(t) + b(t) + c(t) \leq 2N - 1$, donc $c(t + 1) = (a(t) + b(t) + c(t) \text{ div } N) \in \{0, 1\}$.
- Lors des itérations de l'algorithme II.3, la retenue $c(t)$ appartient à $\{0, \ldots, N-1\}$. En effet, dans ce cas, $0 \leq a(t) \cdot b(t) + c(t) \leq N \cdot (N - 1)$, donc $c(t + 1) = (a(t) \cdot b(t) + c(t) \text{ div } N) \in \{0, \ldots, N - 1\}$.

Ainsi, les opérations élémentaires nécessitent des mémoires finies pour stocker les retenues.

Nous utilisons le théorème 5.4.3 pour exprimer le lien qu'il existe entre les entrées, l'état et les sorties d'un FCSR. On a :

$$\Sigma(v, t) = C \cdot (I - N \cdot A)^{-1}(m(t) + N \cdot c(t)) + (D + N \cdot C \cdot (I - N \cdot A)^{-1} \cdot B) \cdot \Sigma(u, t) + c'(t)$$

On note $q = \det(I - N \cdot A)$ l'entier de connexion du FCSR. Alors, comme pour le cas des LFSRs, on a :

- Le terme $C \cdot (I - N \cdot A)^{-1}(m(t) + N \cdot c(t))$ est un vecteur colonne de l éléments de la forme
$$\begin{pmatrix} p_0(t)/q \\ \vdots \\ p_{l-1}(t)/q \end{pmatrix}$$
avec $p_0, \ldots, p_{l-1} \in \mathbb{Z}$.
- De même, $(D + N \cdot C \cdot (I - N \cdot A)^{-1} \cdot B)$ est une matrice $l \times k$ de la forme $r_{i,j}/q$ avec $r_{i,j} \in \mathbb{Z}$.

Comme dans le cas des LFSRs, ceci mène à une représentation rationnelle des FCSRs. Le lien entre les entrées et les sorties d'un FCSR est caractérisé par :

- La matrice $(r_{i,j}/q)_{0\leq i<l, 0\leq j<k}$ qui décrit la transformation linéaire appliquée aux entrées ;
- Le vecteur $(p_0(t), \ldots, p_{l-1}(t))^t$ qui représente l'état interne du FCSR.

Proposition 7.1.2

Soit un FCSR de taille n, avec k entrées et l sorties, défini par les matrices A, B, C, D à coefficients dans $\{0, \ldots, N - 1\}$. On note :

- *$q := \det(I - N \cdot A)$ son entier de connexion,*
- *$p(t) := C \cdot \text{Adj}(I - N \cdot A) \cdot (m(t) + N \cdot c(t))$,*
- *$r := (D + N \cdot C \cdot (I - N \cdot A)^{-1} \cdot B)$*

Alors, la relation entre les entrées et les sorties est :

$$\Sigma(v, t) = \frac{p(t)}{q} + r \cdot \Sigma(u, t)$$

Comme pour les LFSRs, cette approche ne se soucie pas de la façon dont les séquences sont calculées par l'automate. En particulier, il existe différents automates qui possèdent la même relation entre les entrées et les sorties.

De plus, cette proposition décrit le fait que les automates, définis par des matrices à coefficients dans $\{0, \ldots, N - 1\}$ et vus comme des automates sur $\mathbb{Z}_N$, appliquent une relation affine à coefficients rationnels à leurs entrées.

7.1.2 Contrainte sous-jacente

Nous reconsidérons maintenant la définition 7.1.1. Cette définition permet de décrire des automates non implémentables car les coefficients de A, B, C, D, c, c' appartiennent à $\mathbb{Z}_N$. Or un

élément de $\mathbb{Z}_N$ peut nécessiter une mémoire infinie pour être représenté en mémoire, par exemple $\sum_{i\in\mathbb{N}} N^{2^i}$. Pour éviter ces cas, on doit se restreindre à un sous-ensemble de $\mathbb{Z}_N$. L'ensemble qui semble naturel à considérer est celui des séries rationnelles :

$$\mathcal{Q} = \left\{ \frac{p}{q} \in \mathbb{Z}_N,\ p, q \in \mathbb{Z}_N, PGCD(p,q) = 1, q \equiv 1[N] \right\}$$

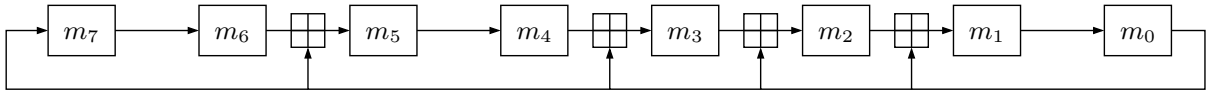
En effet, la proposition 7.1.2 montre que les automates à coefficients dans $\mathbb{Z}_N$ peuvent produire de tels éléments.

7.1.3 Intérêt

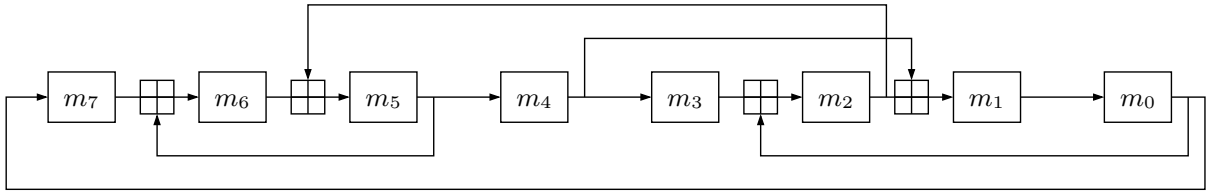
Dans le cas des FCSRs, seuls les modes Galois et Fibonacci étaient connus, l'approche matricielle permet de s'affranchir de cette limite. En particulier, alors que :

- dans le cas Galois, la cellule m_0 a un rôle important puisqu'elle contrôle toutes les rétroactions ;
- dans le cas Fibonacci, la cellule m_{n-1} a un rôle important puisqu'elle est la seule à être en sortie d'un additionneur à retenue ;

il est maintenant possible de considérer des FCSRs sans ces contraintes. Par exemple, la figure 7.1 présente un FCSR en mode Galois, et un *ring* FCSR avec le même entier de connexion, mais avec des rétroactions librement placées.



(a) FCSR en mode Galois



(b) *Ring* FCSR

FIGURE 7.1 – Deux FCSRs avec entier de connexion -347.

Comme pour les LFSRs, les automates définis par des matrices à coefficients dans $\mathcal{S}$ suffisent à construire les FCSRs générant des séquences rationnelles, en particulier, les *l-sequences*. La représentation rationnelle semble donc peu intéressante, car elle ne permet pas d'accéder à une nouvelle classe d'automates. En particulier, des problèmes semblables à ceux des LFSRs se posent lors de l'implémentation (recherche de dénominateur commun, automate non-optimal).

Cependant, dans le cas des FCSRs, il est intéressant de considérer les FCSRs définis par des matrices à coefficients dans $\{-(N-1), \ldots, 0, \ldots, N-1\}$. Ces FCSRs sont un cas particulier des automates à coefficients dans $\mathbb{Q}$. L'intérêt de ces automates résident dans leur implémentation facile.

Nous présentons cela dans le cas des FCSRs sur $\mathbb{Z}_2$ pour une meilleurs compréhension, et car c'est le cas usuel. Nous considérons donc des FCSRs à coefficients dans $\{-1, 0, 1\}$. Nous appellerons ces automates des automates ternaires. Puisque le coefficient -1 apparaît dans les matrices du FCSR, l'automate doit implémenter en particulier la soustraction sur $\mathbb{Z}_2$, de façon analogue à l'addition présentée dans la figure 5.5 pour les AFSRs, ou la figure 4.6 dans le cas des FCSRs. Nous donnons d'abord la proposition suivante qui permet de donner un lien entre le développement de Hensel d'un élément 2-adique, et son complémentaire :

Proposition 7.1.3

Soit $s = \sum_{i\in\mathbb{N}} s_i 2^i$ *avec* $s_i \in \{0,1\}$. *On note* $s' = \sum_{i\in\mathbb{N}} \overline{s_i} 2^i$, *i.e. l'entier 2-adique avec un développement de Hensel complémentaire à s. Alors :* $s' = -1 - s$.

Démonstration : Nous avons la relation $s + s' = \sum_{i\in\mathbb{N}} (s_i + \overline{s_i}) 2^i$. Or $s_i + \overline{s_i} = 1$ pour tout $i \in \mathbb{N}$. On a donc :

$$s + s' = \sum_{i\in\mathbb{N}} 2^i = -1$$

■

En utilisant cette proposition, nous présentons dans la figure 7.2 un soustracteur à retenue construit à partir d'un additionneur à retenue et d'une porte `NOT`. En effet, on a la proposition

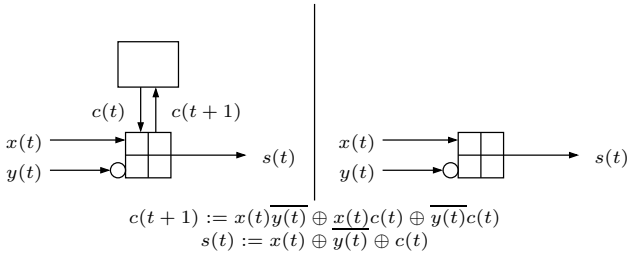


FIGURE 7.2 – Soustracteur à retenue

suivante :

Proposition 7.1.4

Soit un soustracteur à retenue comme présenté dans la figure 7.2. Soit $X = \sum_{t=0}^{\infty} x(t)2^t$, $Y = \sum_{t=0}^{\infty} y(t)2^t$ *les entiers 2-adiques en entrée du soustracteur, et* $S = \sum_{t=0}^{\infty} s(t)2^t$ *l'entier en sortie. On a :*

$$S = X - Y + (c(0) - 1)$$

avec $c(0) \in \{0,1\}$.

Démonstration : En effet, on a la relation suivante pour tout $t \in \mathbb{N}$:

$$x(t) + \overline{y(t)} + c(t) = s(t) + 2 \cdot c(t+1)$$

Ce qui donne en multipliant par 2^t et en sommant pour tout $t \in \mathbb{N}$:

$$\sum_{t\in\mathbb{N}} x(t)2^t + \sum_{t\in\mathbb{N}} \overline{y(t)}2^t + c(0) + \sum_{t\in\mathbb{N}^*} c(t)2^t = \sum_{t\in\mathbb{N}} s(t)2^t + 2\sum_{t\in\mathbb{N}} c(t+1)2^t$$

Or, par la proposition 7.1.3, $\sum_{t\in\mathbb{N}} \overline{y(t)}2^t = -1 - Y$. D'où : $X - 1 - Y + c(0) = S$. ■

Remarque

La valeur contenue dans la retenue d'un soustracteur à retenue implémenté par la figure 7.2 est différente de la valeur du vecteur de retenue dans le définition 7.1.1. En effet :

- lorsque l'implémentation stocke 0, la valeur interprétée est -1 ;
- lorsque l'implémentation stocke 1, la valeur interprétée est 0.

Ces soustracteurs permettent donc d'implémenter, avec un coût matériel quasiment identique à celui d'un additionneur, les matrices à coefficients ternaires. Dans certains cas, ces matrices permettent d'implémenter un FCSR avec un entier de connexion fixé avec un coût inférieur. Cela, grâce à la représentation binaire signée qui est une généralisation de la représentation binaire des entiers.

Définition 7.1.5 (*[EH07]*)

Soient $n \in \mathbb{N}$ *et* $k \in [-(2^n-1), 2^n-1]$. *Une* ***représentation binaire signée*** *de* k *est un* n*-uplet de* $\{-1,0,1\}$ *tel que* $k = \sum_{i=0}^{n-1} k_i 2^i$. *On notera une telle représentation* $(k_{n-1}\dots k_0)_2$, *et* $\tilde{1} := -1$.

Un entier peut avoir plusieurs représentations binaires signées (seul 0 à une représentation unique). Par exemple, 1 peut être exprimé comme $(1\tilde{1})_2$ ou $(1)_2$. De même, 23 s'écrit $(10111)_2$ et $(1100\tilde{1})_2$.

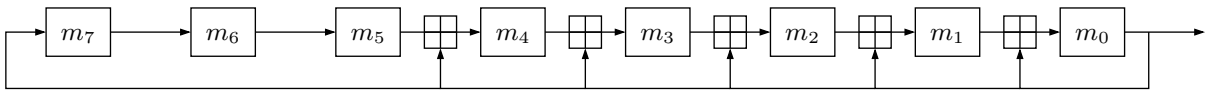
Cette représentation est utilisée pour diminuer le nombre d'additions/soustractions requis par un calcul dans un anneau où le coût de l'addition est (quasiment) celui d'une soustraction. Cette technique est, par exemple, utilisée lors de calcul arithmétique sur des courbes elliptiques, car, dans ce cas, le calcul de l'opposé d'un élément est gratuit [HVM04].

Il existe pour chaque entier une représentation binaire signée particulière : la *non-adjacent form* ou NAF [HVM04]. Cette représentation est telle qu'il n'y a pas deux éléments k_i consécutifs et non-nuls. La NAF a les propriétés suivantes :

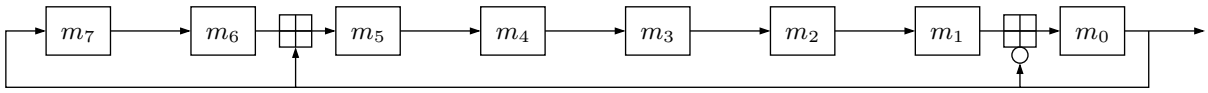
- Un entier k a une unique NAF, noté $NAF(k)$.
- $NAF(k)$ est la représentation binaire signée de k avec le moins de digits non-nuls.

De plus, un algorithme pour calculer $NAF(k)$ existe et a une complexité linéaire en la taille de k : $\mathcal{O}(n)$.

En utilisant la NAF d'un entier de connexion, il est possible de construire un FCSR avec un coût inférieur, par exemple en utilisant le mode Galois. La figure 7.3(a) présente un FCSR en mode Galois associé à l'entier $q = -317$. Pour construire ce FCSR, nous considérons $d = (1-q)/2 = 159 = (10011111)_2$. En particulier, $NAF(d) = (10100001\tilde{1})_2$. La figure 7.3(b) présente alors l'implémentation ternaire de ce FCSR.



(a) FCSR binaire



(b) FCSR ternaire

FIGURE 7.3 – Deux implémentations d'un FCSR avec entier de connexion -317

Le coût du FCSR ternaire est de deux additionneurs, alors que le FCSR binaire en requiert cinq. La table 7.1 présente des états successifs de ces deux automates.

7.2 Implémentation

Nous nous intéressons aux implémentations des FCSRs sans entrée sur $\mathbb{Z}_2$. Nous utilisons la description classique des FCSRs avec des matrices à coefficients binaires ou ternaires.

(a) États successifs du FCSR de la figure 7.3(a)

t	$m(t)$	$c(t)$
0	00000001	00000000
1	10011111	00000000
2	11010000	00001111
3	01100111	00001000
4	10100100	00011011
5	01001001	00010010
6	10101001	00010110
7	11011101	00010110
8	11100111	00011110
9	11110010	00011111
10	01100110	00011001

(b) États successifs du FCSR de la figure 7.3(b)

t	$m(t)$	$c(t)$
0	00000001	00000000
1	10100001	0000000-1
2	11110000	0000000-1
3	01111001	0000000-1
4	10011100	0010000-1
5	01101111	0000000-1
6	10010111	0010000-1
7	11001011	0010000-1
8	11100101	0010000-1
9	11110010	0010000-1
10	01011000	00100000

TABLE 7.1 – États successifs des FCSR des figures 7.3(a) et 7.3(b)

L'implémentation matérielle et logicielle des FCSRs pose les mêmes problème que pour les LFSRs, en remplaçant les portes `OU exclusif` des LFSRs par des additionneurs à retenue dans le cas des FCSRs. Et ces problèmes ont les mêmes solutions, à savoir la construction de matrices de transition avec des formes particulières.

La différence essentielle entre les cas des LFSRs et des FCSRs est la présence de retenues pour les FCSRs. Cependant, cela n'a pas d'influence sur la transposition des résultats liés aux LFSMs aux FCSRs, car nous allons simplement évaluer le coût de l'implémentation matérielle d'un FCSR en nombre d'additionneurs à retenue. Les retenues seront donc évaluées à ce moment.

La seconde différence entre les cas des LFSRs et des FCSRs est que l'état de l'art des FCSRs se limite aux modes Galois et Fibonacci. Nous présenterons donc ici succinctement les résultats, l'idée étant simplement de faire avec les FCSRs ce qui existe déjà avec les LFSRs. Pour cela, nous définissons les *ring* FCSRs [ABL+09] :

Définition 7.2.1

Soit un FCSR $\mathcal{F}$ avec matrice de transition A. $\mathcal{F}$ est appelée ring *FCSR si $A = (a_{i,j})_{0 \leq i,j < n}$ vérifie :*

$$\begin{cases} a_{i,i+1} = 1 \text{ pour } 0 \leq i < n-1 \\ a_{n-1,0} = 1 \end{cases}$$

i.e. ,

$$A = \begin{pmatrix} & 1 & & (*) \\ & & \ddots & \\ & (*) & & \ddots \\ & & & & 1 \\ 1 & & & & \end{pmatrix}$$

Comme pour les LFSRs, ces FCSRs, pour l'implémentation matérielle, sont construits à partir d'un registre à décalage. De plus, les cellules formant un cycle de longueur n, un changement de valeur d'une cellule influencera toutes les autres après n itérations maximum.

7.2.1 Diffusion

Nous adaptons ici le concept de diffusion aux FCSRs. Nous nous intéressons à la diffusion entre les cellules du registre principal (*i.e.* pas entre les retenues). La définition est identique

aux cas des LFSMs :

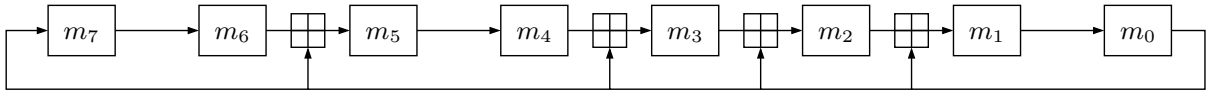
Définition 7.2.2

Soit $\mathcal{F}$ un FCSR avec matrice de transition A de taille n. On note $\mathcal{G}$ le graphe défini par la matrice d'adjacence A^t, i.e. :

- *$\mathcal{G}$ possède n sommets nommés 0, 1,..., $n-1$. Ces sommets représentent les cellules de l'automate.*
- *si $a_{i,j} \neq 0$, alors il existe une arête orientée du sommet j au sommet i. Les arêtes représentent les liens entre les cellules.*

*Nous définissons la **diffusion** comme étant égale au diamètre du graphe $\mathcal{G}$, i.e. la plus grande des distances entre deux sommets de $\mathcal{G}$.*

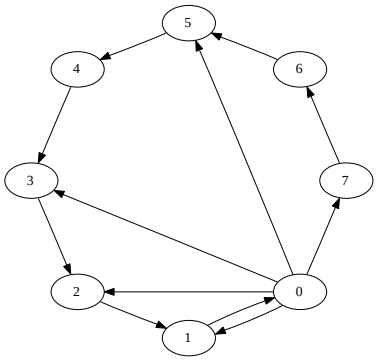
Les figures 7.4 et 7.5 présentent des FCSRs ainsi que le graphe associé. Le FCSR de la figure 7.4 a une diffusion égale à 7 (le sommet 7 est à une distance 7 du sommet 0). Le FCSR de la figure 7.5 a une diffusion égale à 5.



(a) Implémentation

	1						
1		**1**		(0)			
1			**1**				
1				**1**			
					1		
1		(0)				**1**	
							1
1							

(b) Matrice de transition



(c) Graphe associé

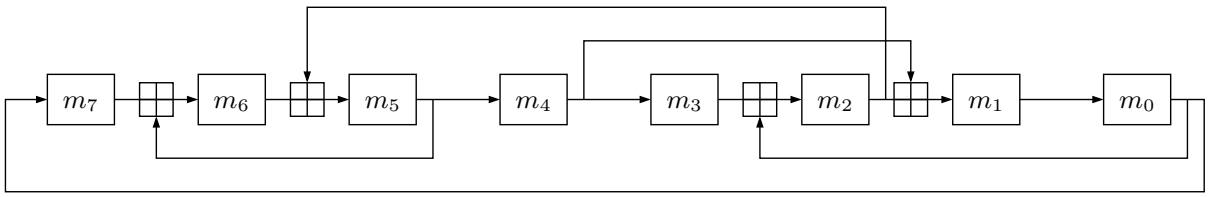
FIGURE 7.4 – Implémentation, matrice de transition et graphe associé à un FCSR en mode Galois

On rappelle que plus la diffusion sera petite, meilleur sera l'automate. Comme pour les LFSRs, on verra que les *ring* FCSRs peuvent avoir une diffusion plus basse que les FCSRs en mode Galois et Fibonacci.

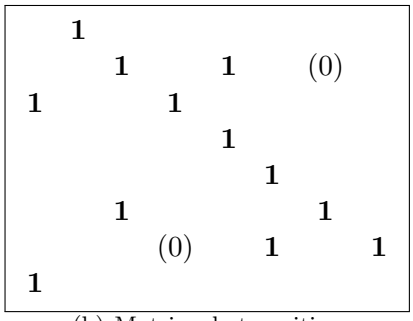
7.2.2 Implémentation matérielle

Cette section donne des critères pour construire des FCSRs avec une bonne implémentation matérielle. Une partie de ces résultats est présentée dans [ABL+09], où nous donnons une nouvelle version du chiffrement à flot F-FCSR-H. Cette application est détaillée dans le chapitre 8.

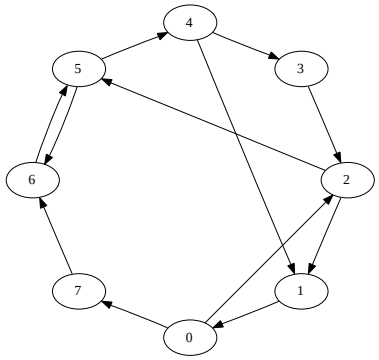
Comme annoncé, la différence par rapport aux cas des LFSRs, est que ici nous allons évaluer le coût de l'implémentation en nombre d'additionneurs à retenue, et non en nombre de portes logiques. Nous comparons les implémentations de FCSRs avec le même entier de connexion en mode Galois, Fibonacci et *ring*. Les valeurs des chemins critiques, *fan-outs* et coûts se calculent à partir de la matrice de transition A du FCSR. Plus précisément, pour des FCSRs de taille n, en notant Row_i la i-ème ligne de A, Col_j la j-ème colonne, et w le poids de Hamming de l'écriture binaire de $d = (1 - |q|)/2$, on a le tableau 7.2.



(a) Implémentation



(b) Matrice de transition



(c) Graphe associé

FIGURE 7.5 – Implémentation, matrice de transition et graphe associé à un *ring* FCSR

	Galois	Fibonacci	*Ring* FCSR
Chemin critique	1	$\lceil \log_2(w-1) \rceil$	$\max \lceil \log_2(w_H(Row_i)) \rceil$
Fan-out	$w-1$	2	$\max w_H(Col_i)$
Coût	$w-2$	$w-2$	$w_H(A)-n$
Diffusion	$n-1$	$n-1$	$\leq n-1$

TABLE 7.2 – Chemin critique, *fan-out*, coût et diffusion de différents FCSRs.

Nous conjecturons que pour un entier de connexion q de taille n, il existe une matrice de transition correspondant à un chemin critique de 1 et un *fan-out* de 2. En effet, le nombre de FCSRs ternaires de taille n avec un chemin critique de 1 et un *fan-out* de 2 est $\mathcal{O}(3^{n^2})$, tandis que le nombre d'entier de connexions de taille n est 2^{n-1}.

Comme pour les LFSRs, on s'intéresse donc à la construction de matrices de transition respectant ces contraintes. Nous utilisons alors le même algorithme, en le modifiant légèrement pour chercher des FCSRs ternaires (Algorithme II.7).

La complexité de cet algorithme est dominée par le calcul de la matrice des cofacteurs et du déterminant. Ces deux opérations, avec les algorithmes classiques, sont calculées simultanément. Mais maintenant, pour une matrice de cofacteur calculée, nous testons $n^2 - n \cdot f$ polynômes. La complexité en moyenne de cet algorithme est donc de $\mathcal{O}(n)$.

Le coût pour tester si un entier q est primitif peut être diminué en utilisant deux tests rapides pour rejeter la plupart des candidats :

- Si q a de petits facteurs, il ne peut pas être primitif.
- Si $2^{\frac{|q|-1}{2}} \equiv 1[q]$, il ne peut pas être primitif.

Seulement un petit nombre d'entiers passent ces tests, alors seulement nous calculons l'ordre de 2 modulo q (en utilisant l'algorithme implémenté dans Magma).

Algorithme II.7 : Tirage aléatoire de *ring* FCSRs ternaires respectant les contraintes matérielles.

Entrées : n la longueur du FCSR. $f \leq n$ le nombre de rétroactions à placer.

Sorties : Une matrice de transition avec un chemin critique de 1, un *fan-out* de 2 et un coût de f, et telle que l'entier de connexion est primitif

début

  Répéter

    $A \leftarrow (a_{i,j})_{0 \leq i,j < n}$ avec $a_{i,j} = \begin{cases} 1 \text{ if } j \equiv i+1[n] \\ 0 \text{ sinon} \end{cases}$;

    tant que $w_H(A) < n + f - 1$ **faire**

      $(i,j) \leftarrow Random([0,n] \times [0,n])$;

      si $w_H(Row_i) = 1$ *et* $w_H(Col_j) = 1$ **alors**

        $a_{i,j} \leftarrow Random(\{-1,1\})$;

    $C \leftarrow$ Matrice des cofacteurs de $I - 2 \cdot A$;

    $q_0 \leftarrow \det(I - 2 \cdot A)$;

    pour $0 \leq i,j < n$ **faire**

      si $w_H(Row_i) = 1$ *et* $w_H(Col_j) = 1$ *et* $a_{i,j} = 0$ **alors**

        $q \leftarrow q_0 - 2 \cdot C_{i,j}$;

        si q *est primitif* **alors**

          retourner $A + E_{i,j}$;

        $q \leftarrow q_0 - 2 \cdot C_{i,j}$;

        si q *est primitif* **alors**

          retourner $A - E_{i,j}$;

fin

7.2.3 Implémentation logicielle

Cette section donne des critères pour construire des FCSRs avec une bonne implémentation logicielle. Une partie de ces résultats est présentée dans [BMP09], où nous donnons une nouvelle version du chiffrement à flot X-FCSR-128. Cette application est détaillée dans le chapitre 9.

L'idée est de travailler avec des matrices définies par blocs et d'utiliser les opérations de décalage à droite et à gauche sur les mots. Cette partie étant identique à celle des LFSRs, nous donnons directement un exemple de FCSR efficace pour l'implémentation logicielle dans la figure 7.6.

L'entier de connexion du FCSR de la figure 7.6 est égal à 1497813390989, qui est premier et primitif. Ce FCSR produit donc des *l-sequences* et est efficacement implémenté en logiciel.

Les *ring* FCSRs sur des mots sont évidemment plus efficaces que les *ring* FCSRs génériques. Plus précisément, pour mettre à jour un FCSR générique, il faut :

- construire un vecteur contenant les rétroactions. Chaque rétroaction est construite en appliquant un masque sur l'état, et en décalant le bit ainsi isolé jusqu'à sa place correcte.
- Une fois cette opération effectuée, on applique une rotation au registre principal.
- Ensuite, on effectue les additions avec retenues.

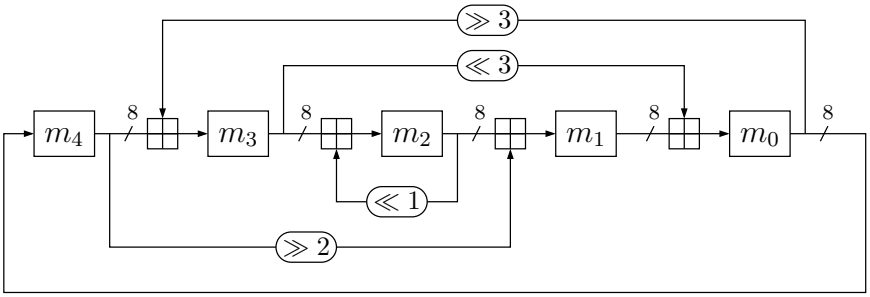
Pour un FCSR sur des mots, il faut :

- construire les rétroactions. Ceci est rapide car cette opération ne nécessite que des décalages.
- Ensuite, on applique une rotation au registre principal, mais cela au niveau des mots. Cette opération ne nécessite que des copies en mémoire.
- Enfin, on effectue les additions avec retenues.

La première et la deuxième étape sont plus efficaces avec les FCSRs sur des mots qu'avec des

$$A = \begin{pmatrix} & I_8 & & L^3 & \\ & & I_8 & & R^2 \\ & & L^1 & I_8 & \\ R^3 & & & & I_8 \\ I_8 & & & & \end{pmatrix}$$

(a) Matrice de transition



(b) Représentation

FIGURE 7.6 – Un FCSR avec une implémentation logicielle efficace.

FCSRs génériques. Le gain entre ces deux modes est d'environ 200 cycles pour une transition d'un FCSR de 256 bits (25 cycles pour un FCSR sur des mots, contre 221 pour un FCSR générique).

Nous présentons ici le gain lié à l'utilisation d'un FCSR sur des mots par rapport à un FCSR générique utilisé dans des chiffrements à flot. Pour cela, nous allons comparer les performances de F-FCSR-16 v3 [ABL+09] et F-FCSR-32 [BMP09]. Ces deux chiffrements font partie de la famille des F-FCSRs qui est détaillée dans le chapitre 8. Ces chiffrements supportent tous les deux des clefs et des IVs de 128 bits. Ils sont basés chacun sur un FCSR :

- pour F-FCSR-32, ce FCSR est de longueur 512, défini sur des mots de 32 bits. Sa matrice de transition est donnée dans la figure 9.2.
- pour F-FCSR-16 v3, ce FCSR est de longueur 256. C'est un *ring* FCSR possédant une implémentation matérielle efficace. Plus précisément, sa matrice de transition est :
  - Pour $0 \leq i < 256$, $a_{i,i+1 \bmod 256} = 1$;
  - Pour $(i,j) \in S$, $a_{i,j} = 1$, où $S = \{$ (0, 52) ; (2, 150) ; (3, 2) ; (5, 169) ; (6, 89) ; (8, 100) ; (9, 1) ; (11, 156) ; (12, 9) ; (13, 46) ; (19, 146) ; (20, 206) ; (26, 204) ; (31, 254) ; (32, 151) ; (38, 144) ; (40, 108) ; (46, 167) ; (47, 198) ; (48, 70) ; (49, 98) ; (50, 213) ; (53, 214) ; (56, 87) ; (57, 55) ; (58, 162) ; (62, 160) ; (63, 13) ; (64, 192) ; (65, 59) ; (66, 12) ; (67, 207) ; (68, 209) ; (71, 229) ; (73, 84) ; (74, 199) ; (77, 168) ; (78, 122) ; (79, 35) ; (80, 154) ; (82, 153) ; (85, 188) ; (87, 51) ; (89, 4) ; (90, 49) ; (93, 231) ; (95, 224) ; (97, 249) ; (101, 208) ; (102, 120) ; (104, 218) ; (105, 8) ; (108, 77) ; (109, 68) ; (110, 250) ; (113, 237) ; (115, 252) ; (116, 17) ; (118, 73) ; (119, 182) ; (123, 29) ; (124, 234) ; (127, 138) ; (132, 190) ; (134, 244) ; (136, 219) ; (141, 228) ; (142, 205) ; (143, 58) ; (144, 230) ; (145, 210) ; (146, 44) ; (147, 137) ; (148, 130) ; (150, 79) ; (152, 111) ; (153, 172) ; (154, 141) ; (156, 78) ; (157, 131) ; (158, 110) ; (159, 127) ; (170, 189) ; (171, 112) ; (174, 217) ; (175, 7) ; (176, 187) ; (177, 40) ; (179, 118) ; (181, 195) ; (184, 48) ; (186, 64) ; (189, 246) ; (190, 47) ; (191, 37) ; (192, 211) ; (193, 85) ; (194, 181) ; (195, 61) ; (196, 54) ; (198, 222) ; (199, 83) ; (203, 105) ; (204, 201) ; (205, 43) ; (206, 139) ; (208, 20) ; (210, 242) ; (211, 124) ; (213, 253) ; (215, 243) ; (216, 69) ; (218, 176) ; (220, 30) ; (222, 19) ; (223, 232) ; (224, 239) ; (225, 220) ; (227, 102) ; (231, 185) ; (232, 15) ; (234, 152) ; (236, 62) ; (238, 245) ; (242, 197) ; (245, 235) ; (246, 171) ; (247, 67) ; (253, 26) ; (254, 202) $\}$;
  - Sinon, $a_{i,j} = 0$.

Pour extraire la suite chiffrante, ces chiffrements utilisent un filtre linéaire :

– pour F-FCSR-32, en notant $(M_0(t), \ldots, M_{15}(t))$ les 16 mots de 32 bits du registre principal du FCSR, l'automate produit le mot de 32 bits

$$M_1(t) \oplus (M_2(t) \ggg 3) \oplus (M_4(t) \ggg 5) \oplus (M_5(t) \ggg 7) \\ \oplus (M_6(t) \ggg 11) \oplus (M_7(t) \ggg 13) \oplus (M_8(t) \ggg 17) \\ \oplus (M_9(t) \ggg 19) \oplus (M_{10}(t) \ggg 23)$$

– pour F-FCSR-16 v3, le filtre est construit de la même manière que pour F-FCSR-H v3, décrit dans le chapitre 8.

Nous présentons dans le tableau 7.3 les performances de F-FCSR-32, F-FCSR-16 v3 et de l'AES en mode compteur. Ces valeurs ont été obtenues grâce au *benchmark* d'eSTREAM [dC05].

	cycles/byte				cycles/key	cycles/IV
Chiffrement	Keystream speed	40 octets	576 octets	1500 octets	Key setup	IV setup
F-FCSR-32	11.92	104.23	18.05	14.63	13.42	3717.39
AES-CTR (128)	12.4	18.27	12.64	12.52	336.54	16.73
F-FCSR-16 v3	130.81	1682.56	243.12	170.56	43.44	61719.00

TABLE 7.3 – Performances calculées en utilisant le *benchmark* d'eSTREAM [dC05].

F-FCSR-32 est toujours plus efficace que F-FCSR-16 v3, que ce soit pour chiffrer de petits fichiers (40 ou 576 octets), ou des plus gros (1500 octets). Lors de la génération de suite chiffrante uniquement, F-FCSR-32 a une vitesse équivalente à l'AES.

Chapitre 8

Applications cryptographiques des FCSRs orientées matérielles

Sommaire

Dans ce chapitre, nous présentons les chiffrements à flot orientés matériels existants basés sur des FCSRs : la famille des FCSRs filtrés F-FCSRs [AB05a, AB05b], et en particulier le chiffrement soumis à eSTREAM F-FCSR-H v2 [ABL06].

Nous présenterons aussi l'attaque par LFSRization des FCSRs [HJ08] qui permet de casser la famille des F-FCSRs basés sur des FCSRs en mode Galois.

Nous présenterons alors comment, grâce à l'approche matricielle présentée dans le chapitre 7, il est possible de se prémunir de cette attaque. Plus précisément, nous présentons une nouvelle version de F-FCSR-H basée sur un *ring* FCSR. Ce chiffrement est présenté dans l'article [ABL+09].

8.1 F-FCSR-H v2

8.1.1 Famille F-FCSR

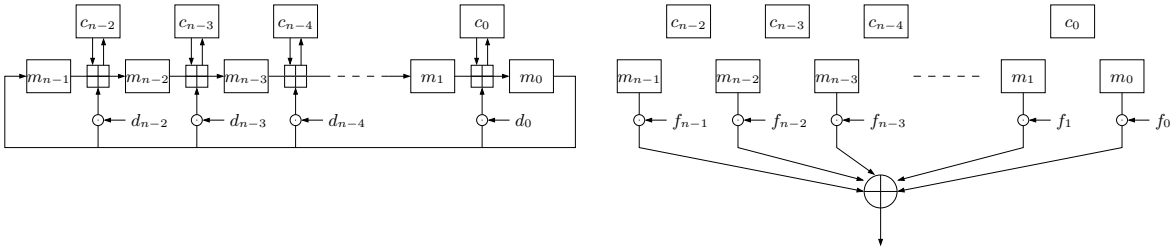
Les chiffrements à flot de la famille F-FCSR [AB05a, AB05b] sont constitués :

– d'un FCSR binaire en mode Galois de longueur n. Plus précisément, il est constitué de :
  – Un registre principal binaire de longueur n : $m = (m_0, \ldots, m_{n-1})$.
  – Un registre de retenue binaire de longueur $n-1$: $c = (c_0, \ldots, c_{n-2})$.

  On note q son entier de connexion. De plus, on note $d = \sum_{i=0}^{n-1} d_i 2^i = \frac{1+|q|}{2}$ avec $d_i \in \{0,1\}$.

  Ce FCSR est utilisé comme fonction de transition de l'automate (figure 8.1(a)).

– d'un filtre linéaire représenté par un vecteur binaire de longueur n : $f = (f_0, \ldots, f_{n-1})$. Ce vecteur est utilisé pour extraire un bit de suite chiffrante (figure 8.1(b)). Plus précisément, le bit extrait est égal à $\sum_{i=0}^{n-1} f_i \cdot m_i$.



(a) Transition (b) Extraction

FIGURE 8.1 – Famille F-FCSR basée sur un FCSR en mode Galois

Cette famille de chiffrement peut être vue comme l'analogue des générateurs à filtre non-linéaire dans le cas de LFSRs :

– Dans le cas des LFSRs, on construit un chiffrement à flot en appliquant un filtre non-linéaire à un LFSR (qui a une fonction de transition linéaire) (figure 1.10(b)).
– Dans le cas des F-FCSRs, on construit un chiffrement à flot en appliquant un filtre linéaire à un FCSR (qui a une fonction de transition non-linéaire).

Ainsi, le FCSR assure que les séquences produites sont non-linéaires et ont de bonnes propriétés statistiques. Le filtre linéaire permet de masquer la structure 2-adique, est facilement implémenté, et empêche les attaques par corrélation [ABL08].

8.1.2 F-FCSR-H v2

Nous détaillons maintenant spécifiquement le chiffrement F-FCSR-H v2 [ABL06]. Par rapport à la première version [AB05a], seul la procédure de *key/IV setup* a été modifiée pour contrer l'attaque par distingueur présentée dans [JM06a]. Ce nouveau *key/IV setup* est construit pour résister aussi aux attaques algébriques présentées dans [BM05].

Ce chiffrement utilise des clefs de 80 bits, et des IVs de longueur v, $32 \leq v \leq 80$. Le FCSR est de longueur $n = 160$, et est caractérisé par l'entier de connexion

$$q = -1993524591318275015328041611344215036460140087963$$

En particulier, 82 rétroactions sont nécessaires pour construire le FCSR (et donc 82 additionneurs avec retenue).

La fonction d'extraction produit 8 bits en appliquant 8 filtres sur le registre principal. Plus précisément, on note

$$\begin{aligned} F_0 &= (0011\ 0111\ 0100\ 1010\ 1010)_2, & F_1 &= (1001\ 1010\ 1101\ 1100\ 0001)_2, \\ F_2 &= (1011\ 1011\ 1010\ 1110\ 1111)_2, & F_3 &= (1111\ 0010\ 0011\ 1000\ 1001)_2, \\ F_4 &= (0111\ 0010\ 0010\ 0011\ 1100)_2, & F_5 &= (1001\ 1100\ 0100\ 1000\ 1010)_2, \\ F_6 &= (1111\ 0010\ 0011\ 1000\ 1001)_2, & F_7 &= (1101\ 0011\ 1011\ 1011\ 0100)_2 \end{aligned}$$

Alors, le i-ème bit b_i (avec $0 \leq i \leq 7$) de l'octet extrait est

$$b_i = \bigoplus_{j=0}^{19} f_i^j m_{8j+i} \text{ avec } F_i = \sum_{j=0}^{19} f_i^j 2^j$$

La procédure de *key/IV setup* est présentée dans l'algorithme II.8

Algorithme II.8 : *Key/IV setup* de F-FCSR-H v2.

Entrées : K la clef, IV l'IV de longueur $32 \leq v \leq 80$.
début
  $m \leftarrow (0^{80-v}|IV|K)$;
  $c \leftarrow (0^{160})$;
  pour $i = 0$ *à* 19 **faire**
    Mettre à jour le FCSR;
    Extraire S_i un octet en utilisant le filtre;
  $m \leftarrow (S_{19}|\dots|S_1|S_0)$;
  pour $i = 0$ *à* 161 **faire**
    Mettre à jour le FCSR;
fin

8.2 LFSRization des FCSRs

Dans [HJ08], les auteurs présentent une attaque contre la famille des F-FCSRs en utilisant une faiblesse liée au mode Galois du FCSR. Nous présentons ici l'idée clef de l'attaque, appelée LFSRization des FCSRs, appliquée au F-FCSR-H v2.

Tout d'abord, les auteurs remarquent que le vecteur de retenues du FCSR n'a pas un comportement aléatoire. En effet, la valeur de chaque retenue est liée à la rétroaction, *i.e.* la valeur de la cellule m_0. Si l'on suppose que cette rétroaction est nulle, alors :

- Si une retenue vaut 0, alors elle reste nulle.
- Si une retenue vaut 1, alors elle devient nulle avec probabilité 1/2 (en supposant que l'entrée de l'additionneur aléatoire).

En particulier, si la rétroaction est nulle plusieurs fois consécutives, les retenues deviennent nulles avec une forte probabilité. Plus précisément, à chaque transition où la rétroaction est nulle, le nombre de retenues non-nul est environ divisé par deux. Ce comportement est confirmé de manière expérimentale en observant le vecteur de retenues. Ainsi, le registre de retenues de F-FCSR-H v2 étant consititué de 82 retenues actives, il faut environ $\log_2 82 \approx 7$ rétroactions consécutives nulles pour que le registre de retenues soit nul.

Une fois que le vecteur de retenues est nul, si la rétroaction est nulle pendant encore 19 fois, le générateur extrait 160 bits obtenues par un filtre linéaire. Il est donc possible de retrouver l'état de l'automate puisqu'il est constitué de 160 bits pour le registre principal (et 82 bits pour le registre de retenues, que l'on sait nul).

En fait, l'événement « le vecteur de retenues est nul et la rétroaction est nulle plusieurs fois consécutives » a lieu avec une probabilité négligeable [ABM08]. Les auteurs remarquent alors que pour que la rétroaction soit nulle plusieurs fois consécutives, il faut que la dernière retenue (c_1 dans le cas de F-FCSR-H v2) soit égale à 1. Cela modifie légèrement l'approche : on s'intéresse donc à l'événement suivant dans le cas de F-FCSR-H v2 :

$$c(t) = c(t+1) = \cdots = c(t+19) = (0, 0, \dots, 0, 1, 0)$$

Cet événement se produit avec une forte probabilité quand 20 rétroactions consécutives sont nulles. Alors, on est capable de reconstruire la valeur du registre principal grâce aux 160 bits produits en utilisant une relation affine et non plus linéaire (car le vecteur de retenue n'est plus nul). De plus, les 8 filtres sont construits tels que le filtre i n'utilise que les cellules m_j avec $j \equiv i[8]$. Cela permet de séparer les dépendances, et de pré-calculer des tables pour résoudre le système affine.

On a vu qu'il faut environ $\log_2 82 \approx 7$ rétroactions consécutives nulles pour que le vecteur de retenue soit dans cet état. En supposant que la valeur de la rétroaction est uniformément

distribuée, cet événement arrive avec une probabilité 2^{-26} environ. Des mesures empiriques confirment cette valeur.

Ainsi, cette méthode permet de retrouver l'état de l'automate avec 2^{26} octets de suite chiffrante environ. Dans [HJ08], des améliorations sont apportées pour descendre la longueur de suite chiffrante nécessaire jusqu'à $2^{24,7}$ octets.

8.3 F-FCSR-H v3

Dans [ABL+09], nous proposons une nouvelle version du chiffrement F-FCSR-H qui résiste à cette attaque. Pour cela, on remplace le FCSR utilisé par un *ring* FCSR. Ainsi, les retenues ne sont plus contrôlées par une seule mémoire ; elles ne sont donc plus corrélées.

8.3.1 Le FCSR

Pour F-FCSR-H v3, nous avons cherché un FCSR répondant aux contraintes suivantes :
- F-FCSR-H v3 utilise des clefs et des IVs de 80 bits. Nous avons donc cherché un FCSR de 160 bits, pour assurer une entropie suffisante.
- Le nombre de rétroactions doit être « élevé » (compris entre 75 et 85) pour assurer au FCSR un comportement non-linéaire, même localement.
- Le FCSR doit avoir une implémentation matérielle efficace. En particulier, avec un chemin critique et un *fan-out* minimal.

Nous avons donc cherché un *ring* FCSR de longueur 160 vérifiant ces conditions. De plus, nous avons ajouté la condition que la matrice de transition A devait être inversible : $\det(1) \neq 0$.

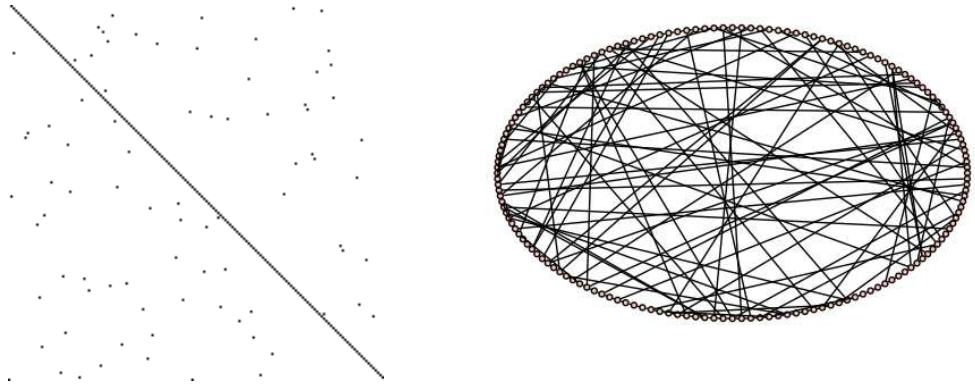
Nous avons choisi le FCSR avec la matrice de transition $A = (a_{i,j})_{0 \leq i,j < 160}$ suivante :
- Pour $0 \leq i < 160$, $a_{i,i+1 \bmod 160} = 1$;
- Pour $(i,j) \in S$, $a_{i,j} = 1$, où $S = \{$ (1, 121) ; (2, 133) ; (4, 44) ; (5, 82) ; (9, 38) ; (11, 40) ; (12, 54) ; (14, 105) ; (15, 42) ; (16, 63) ; (18, 80) ; (19, 136) ; (20, 2) ; (21, 35) ; (23, 28) ; (25, 137) ; (28, 131) ; (31, 102) ; (36, 41) ; (39, 138) ; (40, 31) ; (42, 126) ; (44, 127) ; (45, 77) ; (46, 110) ; (47, 86) ; (48, 93) ; (49, 45) ; (51, 17) ; (54, 8) ; (56, 7) ; (57, 150) ; (59, 25) ; (62, 51) ; (63, 129) ; (65, 130) ; (67, 122) ; (73, 148) ; (75, 18) ; (77, 46) ; (79, 26) ; (80, 117) ; (81, 1) ; (84, 72) ; (86, 60) ; (89, 15) ; (90, 89) ; (91, 73) ; (93, 12) ; (94, 84) ; (102, 141) ; (104, 142) ; (107, 71) ; (108, 152) ; (112, 92) ; (113, 83) ; (115, 23) ; (116, 32) ; (118, 50) ; (119, 43) ; (121, 34) ; (124, 13) ; (125, 74) ; (127, 149) ; (128, 90) ; (129, 57) ; (130, 103) ; (131, 134) ; (132, 155) ; (134, 98) ; (139, 24) ; (140, 61) ; (141, 104) ; (144, 48) ; (145, 14) ; (148, 112) ; (150, 59) ; (153, 39) ; (156, 22) ; (157, 107) ; (158, 30) ; (159, 78) $\}$;
- Sinon, $a_{i,j} = 0$.

Ce FCSR possède 82 rétroactions. Puisque c'est un *ring* FCSR, il a une meilleure diffusion, égale à 24 (159 pour F-FCSR-H v2). La figure 8.2 représente la matrice de transition A, ainsi que le graphe $\mathcal{G}$ ayant pour matrice d'adjacence A^t.

8.3.2 Le filtre

Comme pour F-FCSR-H v2, nous utilisons un filtre linéaire pour extraire la suite chiffrante, de façon à casser la structure 2-adique du FCSR. Le filtre agit sur les cellules du registre principal recevant une rétroaction. De plus, le fait que le filtre soit linéaire permet de contrer les attaques par corrélation. En effet, ici, le filtre possède 82 entrées, il est donc résilient d'ordre 81, *i.e.* équilibré et sans corrélation entre sa sortie et tout ensemble d'au plus 81 entrées [ABL08]. De plus, pour que le filtre soit difficilement inversible, nous choisissons de produire 8 bits à chaque extraction.

La structure périodique du filtre dans F-FCSR-H v2 a été utilisée dans [HJ08] pour accélérer l'attaque. Nous choisissons donc pour F-FCSR-H v3 une structure non-périodique :



(a) Matrice de transition (b) Graphe associé à A

FIGURE 8.2 – Représentations de F-FCSR-H v3

– Soit $f_0 < f_1 < \cdots < f_{81}$ les indices des cellules du registre principal recevant une rétroaction, *i.e.* les lignes f_i de la matrice A ayant un poids égal à 2.
– Les 8 bits $z_0, \ldots, z_7$ extraits sont :

$$\forall 0 \leq i < 8,\ z_i = \bigoplus_{j \equiv i[8]} m_{f_j}$$

8.3.3 *Key/IV setup*

Comme présenté dans [HJ08], lorsque un FCSR est synchronisé (*i.e.* dans un état dans lequel, après un nombre fini d'itérations, il revient), on peut calculer ses états précédents en utilisant seulement des multiplications dans $\mathbb{Z}/q\mathbb{Z}$. De plus, un FCSR en mode Galois est synchronisé en au plus $n + 4$ itérations [ABM08], mais en réalité, quelques itérations suffisent. Pour éviter la faiblesse du *key/IV setup* utilisé dans [HJ08], nous décidons de maintenir le FCSR dans un état non-synchronisé durant le *key/IV setup*. Ainsi, ce nouveau *key/IV setup* est très difficile à inverser, de façon à empêcher le recouvrement de la clef.

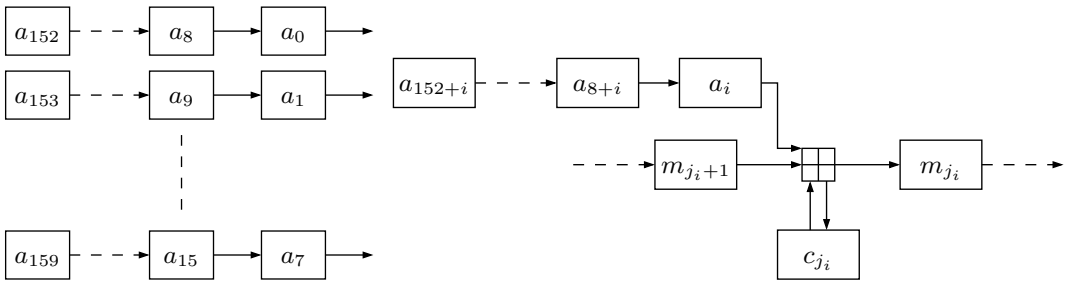
Cependant, le fait d'utiliser un *ring* FCSR pose un nouveau problème : nous ne pouvons pas garantir l'entropie de l'automate. Dans le cas de F-FCSR basé sur un FCSR en mode Galois, mettre les retenues à zéro permet d'éviter les collisions (*i.e.* deux initialisations différentes produisant la même suite chiffrante) et garantit une entropie constante. Cela provient de la structure particulière de la matrice $\mathrm{Adj}(I - 2A)$, dont la première ligne est constituée des puissances de 2 successives. Avec un *ring* FCSR, cette matrice n'a pas de structure particulière. Cependant, dans ce cas la recherche de collisions est une instance du problème de la somme de sous-ensembles (*subset sum problem*), avec une complexité de $2^{n/2}$ dans le cas où les retenues sont nulles ; $2^{3n/2}$ dans le cas général.

En conséquence, le nouvel *key/IV setup* maintient le FCSR dans un état non-synchronisé aussi longtemps que possible. Pour cela, nous connectons 8 registres à décalage de longueur 20. Ces 160 nouvelles cellules $a_0, \ldots, a_{159}$ sont connectées comme présenté dans le figure 8.3(a).

Les positions où nous connectons ces registres sont $J = \{3, 22, 43, 64, 83, 103, 123, 143\}$. Elles ont été choisies telles qu'il n'existait pas d'additionneur entre les cellules m_{j_i+1} et m_{j_i} (où $j_i \in J$). Chaque registre est connecté en utilisant un additionneur à retenue comme présenté dans la figure 8.3(b).

La procédure de *key/IV setup* est présenté dans l'algorithme II.9

La première boucle « pour » injecte la clef et l'IV depuis les registres additionnels dans le FCSR. Les 24 clocks suivants assurent une diffusion de la clef et de l'IV sur l'ensemble du FCSR



(a) Disposition des cellules (b) Injection

FIGURE 8.3 – Disposition des registres à décalage pour le *key/IV setup*

Algorithme II.9 : *Key/IV setup* de F-FCSR-H v3.

Entrées : K la clef, IV l'IV de longueur $32 \leq v \leq 80$.
début
  $(a_0, \ldots, a_{159}) \leftarrow (K|0^{160-v}|IV)$;
  $m \leftarrow (0^{160})$;
  $c \leftarrow (0^{160})$;
  pour $i = 0$ *à* 19 **faire**
    Mettre à jour le FCSR;
    Extraire $z_0, \ldots, z_7$ un octet en utilisant le filtre;
    $(a_{152}, a_{153}, \ldots, a_{159}) \leftarrow (z_0, \ldots, z_7)$;
  pour $i = 0$ *à* 27 **faire**
    Mettre à jour le FCSR;
fin

(on rappelle que la diffusion du FCSR est de 24). Ainsi, si un attaquant est capable de retrouver l'état à la fin du *key/IV setup*, il ne sera pas en mesure d'utiliser cette information pour retrouver la clef car les états non-synchronisés ont plusieurs antécédents. Plus précisément, soit une cellule m_{k+1} du registre principal recevant une rétroaction de la cellule m_j. La valeur $m_{k+1}(t)$ dépend des valeurs $c_k(t-1)$, $m_k(t-1)$ et $m_j(t-1)$:

– si $(c_k(t-1), m_k(t-1), m_j(t-1)) \in \{(0,0,0),(0,1,1),(1,0,1),(1,1,0)\}$, alors $m_{k+1}(t) = 0$.
– si $(c_k(t-1), m_k(t-1), m_j(t-1)) \in \{(0,0,1),(0,1,0),(1,0,0),(1,1,1)\}$, alors $m_{k+1}(t) = 1$.

La connaissance de $m_{k+1}(t)$ ne donne qu'une information très partielle sur les valeurs $c_k(t-1)$, $m_k(t-1)$ et $m_j(t-1)$. Le nombre de possibilités pour l'état précédent grandit de façon combinatoire.

Comme mentionné plus haut, le *key/IV setup* n'est pas une bijection, et se comporte plutôt comme une fonction aléatoire. De ce point de vue, deux attaques sont possibles : la recherche de collisions directe et la recherche de collisions en utilisant un compromis temps/mémoire basé sur la perte d'entropie. Comme nous l'avons déjà dit, la recherche de collisions directe a un coût de 2^{80} si l'attaquant peut rendre nul le vecteur de retenues. Or, puisque l'on utilise un *ring* FCSR, les retenues ne sont pas contrôlées par une unique rétroaction. En particulier, la probabilité que toutes les retenues soient nulles est d'environ 2^{-82}. En conséquence, ce type d'attaque est plus coûteux que la recherche exhaustive de la clef. Si l'attaquant ne peut pas forcer le vecteur de retenues à 0, la complexité de l'attaque est 2^{240}, ce qui est trop coûteux.

Les attaques par compromis temps/mémoire sont possibles si suffisamment d'entropie est perdue. Comme étudié dans [Röc08], en considérant la clef et l'IV aléatoire, la perte d'entropie d'un FCSR est d'environ 1 bit. Dans notre cas, l'entropie après le *key/IV setup* est proche de 159 bits. Est-ce possible d'utiliser cette perte d'entropie pour une recherche de collisions

par compromis temps/mémoire ? Un cas d'étude comparable est l'attaque proposée dans [HK05] contre le chiffrement MICKEY. Même si cette attaque semble fonctionner, A. Röck a montré dans [Röc08] que la complexité liée à l'ensemble des états initiaux ne peut pas être significativement réduite, et que les attaques basées sur le problème de perte d'entropie sont moins efficaces que ce que l'on pouvait espérer. En conséquence, nous conjecturons que notre *key/IV setup* se comporte comme une fonction aléatoire, et que la perte d'entropie n'est pas suffisante pour monter une recherche de collisions par compromis temps/mémoire, en particulier en terme de complexité.

8.3.4 Résistance contre les attaques connues

Dans [BM05], les auteurs montrent que le chiffrement à flot F-FCSR-H v2 est résistant aux attaques algébriques. En particulier, ils étudient le degré et le nombre de monômes des équations algébriques décrivant un FCSR. Comme présenté dans [ABL06, MP08], le nombre de monômes dans ces équations semble être un facteur limitant pour construire une attaque algébrique. Dans le cas d'un FCSR en mode Galois de taille 128, la construction de l'équation décrivant l'état du FCSR après 13 itérations ne semble pas atteignable d'un point de vue calculatoire. L'utilisation d'un *ring* FCSR rend la construction de ces équations encore plus ardue. En effet, la diffusion étant ici de 24, le degré des équations augmente plus rapidement. Pour ces raisons, F-FCSR-H v3 semble hors d'atteinte des attaques algébriques.

Comme présenté dans [ABL08], la famille F-FCSR est résistante aux attaques par corrélation pour deux raisons :

- la fonction de transition d'un FCSR est non-linéaire sur $\mathbb{F}_2$. Il semble donc difficile de construire des dépendances linéaires entre les cellules du registre principal. Plus précisément, la non-linéarité provient des cellules de retenue. Or, le filtre ne prend en entrées que des cellules recevant une rétroaction, *i.e.* dont le comportement est lié à une retenue ;
- comme évoqué précédemment, le filtre étant linéaire avec 82 entrées, il est résilient d'ordre 81. Ce qui rend la recherche de corrélation très difficile.

F-FCSR-H v3 apparaît donc résistant aux attaques par corrélation.

Dans [JM06a], une attaque est proposée, basée sur le *IV setup* de F-FCSR-H v1. Cette attaque utilise le fait que l'introduction d'une différence dans un FCSR en mode Galois a un impact très localisé après un petit nombre d'itérations. Or, dans F-FCSR-H v1, l'IV sert à initialiser le registre de retenues lors de l'*IV setup*. Cette attaque a été mise en défaut dans F-FCSR-H v2 en changeant le *key/IV setup*, en particulier en augmentant le nombre d'itérations du FCSR avant de produire la suite chiffrante [ABL06]. Ceci permet de diffuser une différence sur une cellule à l'ensemble du FCSR. Ici, le fait d'utiliser un FCSR avec une diffusion de 24 et d'effectuer 28 itérations avant de produire la suite chiffrante permet de se prémunir contre cette attaque.

Dans [FMS08], les auteurs présentent une attaque contre un F-FCSR basé sur un FCSR en mode Fibonacci. Ils remarquent que dans ce cas, une seule cellule a un comportement non-linéaire. Ils proposent de considérer à chaque itération cette nouvelle valeur comme une inconnue. Étant donné qu'un F-FCSR utilise un filtre linéaire produisant plusieurs bits à chaque extraction, il est possible de résoudre le système linéaire ainsi construit. Cette attaque est possible car, dans le cas d'un FCSR en mode Fibonacci, la non-linéarité ne concerne qu'une seule cellule. L'utilisation d'un *ring* FCSR (et même d'un FCSR en mode Galois) suffit à contrer cette attaque.

Les attaques par compromis temps/mémoire ont été traitées dans la sous-section précédente.

Nous nous intéressons ici à l'attaque présentée dans [HJ08]. Cette attaque contre F-FCSR-H v2, basé sur un FCSR en mode Galois, se fonde sur l'existence de corrélations entre les retenues et la valeur de la rétroaction. Plus précisément, le contrôle de la cellule m_0 permet de contrôler la rétroaction. Et si cette rétroaction est nulle pendant t itérations consécutives, le comportement du FCSR devient linéaire. On peut alors retrouver son état facilement. Ce comportement linéaire

se produit avec une probabilité 2^{-t} pour un FCSR en mode Galois. Dans le cas d'un *ring* FCSR, cette probabilité devient $2^{-t \cdot k}$ où k est le nombre de cellules contrôlant une rétroaction (dans le cas de F-FCSR-H v3 : $k = 82$). En particulier ici, ce comportement linéaire a une probabilité tellement petite, que la complexité de l'attaque devient supérieure à la complexité de la recherche exhaustive. De plus, l'attaque présentée dans [HJ08] se fonde sur le fait que le vecteur de retenues reste constant pendant plusieurs itérations. De façon expérimentale sur F-FCSR-H v3, sur 2^{38} états, nous avons trouvé seulement 41 états différents pour lesquels le registre de retenues restent inchangé après une itération. Et nous n'avons trouvé aucun état tel que le registre de retenues reste inchangé après deux itérations.

Chapitre 9

Applications cryptographiques des FCSRs orientées logicielles

Sommaire

Dans ce chapitre, nous présentons un chiffrement à flot basé sur des FCSRs orienté logiciel : le chiffrement X-FCSR-128 v1 [ABLM07]. Ce chiffrement utilise deux FCSRs en mode Galois et et produit 128 bits à chaque extraction. Cependant, dans [SHJ09], les auteurs montrent que la LFSRization des FCSRs est applicable : dans le cas de X-FCSR-128 v1, cela ne conduit pas à une attaque ; mais dans le cas de X-FCSR-256 (qui est un chiffrement semblable, mais produisant 256 bits à chaque extraction), cela conduit à une attaque.

Nous montrons alors comment, grâce à l'approche matricielle présentée dans le chapitre 7, il est possible de se prémunir de cette menace. Plus précisément, nous présentons la deuxième version de X-FCSR-128 basée sur un *ring* FCSR [BMP09].

9.1 X-FCSR-128 v1

X-FCSR-128 v1 [ABLM07] est un chiffrement à flot dédié à une application logicielle. Il produit 128 bits de suite chiffrante à la fois. Il est construit autour de deux primitives :

- Deux FCSRs en mode Galois de longueur 256 bits. Les séquences produites par chacun des FCSRs sont des *l-sequences*. De plus, ils sont mis à jour dans le sens opposé : le premier est basé sur un registre à décalage dont la fonction de transition glisse les valeurs vers la droite ; tandis que le second décale ses valeurs vers la gauche.
- Une fonction $Round_{128}$ inspirée des chiffrements par bloc. Plus précisément, étant donné un vecteur de 128 bits a, on le représente par une matrice 4×4 d'octets que l'on note $(a_{i,j})_{0 \leq i,j < 4}$. $Round_{128}$ se décompose en trois opérations :
  - Une substitution des octets via une boîte S (présentée dans la sous-section 9.2.3).

– Une opération agissant sur les lignes de la matrice, correspondant au $ShiftRows$ de l'AES.
– Une opération agissant sur les colonnes de la matrice, semblable au $MixColumns$ de l'AES. Plus précisément, $MixColumns$ est décrite dans [GLP06] : la j-ième colonne de a, $0 \leq j < 4$, est remplacée par :

$$\begin{pmatrix} a_{3,j} \oplus a_{0,j} \oplus a_{1,j} \\ a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \\ a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \\ a_{2,j} \oplus a_{3,j} \oplus a_{0,j} \end{pmatrix}.$$

Même si cette fonction n'a pas une diffusion optimale, son *branch number* vaut 4, et son évaluation est réellement plus rapide que le $MixColumns$ de l'AES : elle peut être implémentée avec seulement 6 `XOR` sur des mots de 32 bits.

La figure 9.1 présente son fonctionnement. Plus précisément, l'extraction de 128 bits se

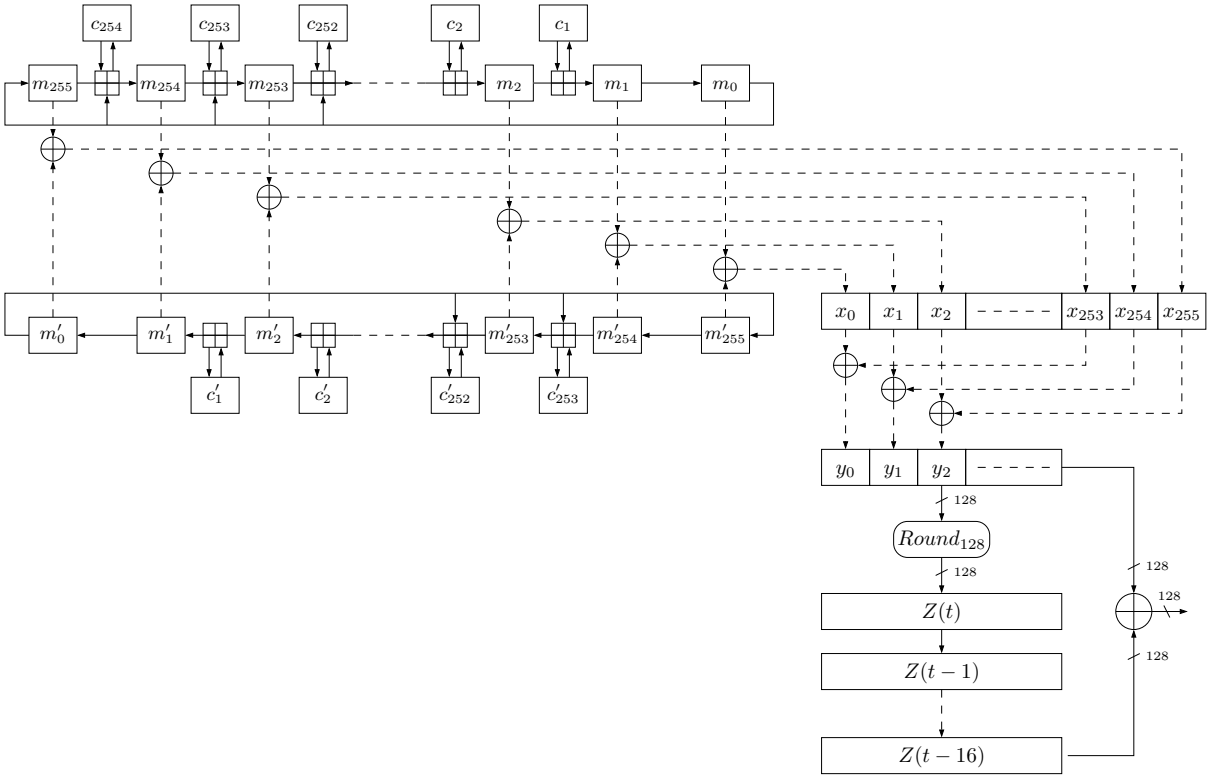


FIGURE 9.1 – X-FCSR-128 v1

déroule ainsi :

– Les registres principaux des FCSRs sont `XOR`és :

$$X(t) = (x_0(t), \ldots, x_{255}(t)) = (m_0(t) \oplus m'_{255}(t), m_1(t) \oplus m'_{254}(t), \ldots, m_{255}(t) \oplus m'_0(t))$$

Le fait que les FCSRs soient mis à jour en sens opposé permet d'éviter l'apparition de corrélation entre des valeurs $x_i(t)$ et $x_{i+1}(t-1)$.

– Le vecteur $X(t)$ est « replié » sur lui-même :

$$Y(t) = (x_0(t) \oplus x_{128}(t), x_1(t) \oplus x_{129}(t), \ldots, x_{127}(t) \oplus x_{255}(t))$$

– On applique $Round_{128}$ à $Y(t)$: $Z(t) = Round_{128}(Y(t))$.

– $Z(t)$ est conservé en mémoire pendant 16 itérations.
– La sortie de l'automate est : $Y(t) \oplus Z(t-16)$.

Il existe une version produisant 256 bits de suite chiffrante à chaque itération : X-FCSR-256. Les différences avec X-FCSR-128 v1 sont :

– l'utilisation d'une fonction $Round_{256}$ de $\{0,1\}^{256}$ dans lui-même, construite de la même façon que $Round_{128}$;
– le vecteur $X(t)$ n'est pas « replié » sur lui-même pour être traité lors de l'extraction : l'automate produit $X(t) \oplus Round_{256}(X(t-16))$.

Cette version a été attaquée [SHJ09] en se basant sur la LFSRization des FCSRs. Cependant, cette attaque, adaptée à X-FCSR-128 v1, a une complexité trop élevée pour être réalisable (elle nécessiterait une table de 2^{72} valeurs et 2^{74} blocs de suite chiffrante). Malgré tout, une nouvelle version de X-FCSR-128 a été proposée, à la fois pour prévenir la LFSRization des FCSRs, mais aussi pour améliorer les performances du chiffrement.

9.2 X-FCSR-128 v2

Comme pour les autres versions de X-FCSR, le design se base sur des opérations (boîte S et opérations linéaires) utilisées dans les chiffrements par bloc pour augmenter l'efficacité du chiffrement. X-FCSR-128 v2 utilise des clefs de 128 bits, et des IVs allant de 64 à 128 bits, et produit 128 bits à chaque itération.

9.2.1 Le FCSR

Le cœur de X-FCSR-128 v2 est un *ring* FCSR de longueur 512 agissant sur des mots de 32 bits. La matrice de transition du FCSR est présentée dans la figure 9.2. Elle a été construite en utilisant l'algorithme II.6 adapté aux FCSRs.

$$\left(\begin{array}{c|c|c|c|c|c}
I & L^{10} & & & & \\
\quad I & \quad I \quad R^{7} & & & & \\
\hline
 & \quad I & & (0) & & \\
 & \quad\quad I & & & I & \\
L^{7} & & I & & & \\
\hline
R^{10} & & I & & & \\
 & & \quad I & & R^{6} & \\
 & & & I \quad L^{3} & & \\
\hline
 & & & \quad I & & L^{8} \\
R^{4} & & & \quad\quad I & & \\
 & (0) & & & I & \\
\hline
 & & & & I & \\
 & & & & \quad I & \\
 & & & & & I \\
\hline
I & & & & &
\end{array}\right)$$

FIGURE 9.2 – Matrice de transition du FCSR sur des mots de 32 bits de X-FCSR-128 v2

Ce FCSR est constitué de neuf rétroactions. L'entier de connexion q associé est tel que :
– q est premier,
– $\log_2(q) \approx 512.43$,

– 2 est primitif dans $\mathbb{Z}/q\mathbb{Z}$.

De plus, la diffusion de ce FCSR est de 37, ce qui est beaucoup plus faible que dans le cas d'un FCSR en mode Galois (dans ce cas, la diffusion est de 511).

9.2.2 L'extraction

On note $(M_0(t), \ldots, M_{15}(t))$ le contenu du registre principal, constitué de 16 mots de 32 bits, à l'instant t. La fonction d'extraction est constituée de :

– la fonction $Round_{128}$ (déjà utilisée dans X-FCSR-128 v1) qui prend en entrée et produit des mots de 128 bits ;

– une mémoire de 16 mots de 128 bits pour stocker la sortie de $Round_{128}$ qui sera utilisée 16 itérations plus tard.

Plus précisément, l'extraction de 128 bits se déroule ainsi :

– On extrait $Y(t) = (Y_0(t)|Y_1(t)|Y_2(t)|Y_3(t))$, un vecteur de 128 bits, du registre principal du FCSR :

$$\begin{aligned} Y_0(t) &= M_0(t) \oplus (M_4(t) \ggg 11) \oplus (M_8(t) \ggg 19) \oplus (M_{12}(t) \ggg 23) \\ Y_1(t) &= M_1(t) \oplus (M_5(t) \ggg 11) \oplus (M_9(t) \ggg 19) \oplus (M_{13}(t) \ggg 23) \\ Y_2(t) &= M_2(t) \oplus (M_6(t) \ggg 11) \oplus (M_{10}(t) \ggg 19) \oplus (M_{14}(t) \ggg 23) \\ Y_3(t) &= M_3(t) \oplus (M_7(t) \ggg 11) \oplus (M_{11}(t) \ggg 19) \oplus (M_{15}(t) \ggg 23) \end{aligned}$$

– On calcule $Z(t) = Round_{128}(Y(t))$,

– On stocke $Z(t)$ pendant 16 itérations,

– La sortie à l'instant t est le vecteur de 128 bits $Y(t) \oplus Z(t-16)$.

La fonction $Round_{128}$ est la même que pour X-FCSR-128 v1 [ABLM07]. C'est une fonction de $\{0,1\}^{128}$ dans lui-même. Étant donné un vecteur de 128 bits a, on le représente par une matrice 4×4 d'octets que l'on note $(a_{i,j})_{0\leq i,j<4}$. Alors, $Round_{128}(a) = MixColumns(ShiftRows(Sbox(a)))$ où :

– $Sbox$ applique la boîte S (présentée dans la sous-section 9.2.3) sur les octets $a_{i,j}$, *i.e.* l'octet $a_{i,j}$ est transformé en l'octet $b_{i,j} = S(a_{i,j})$.

– $ShiftRows$ correspond à l'opération du même nom dans l'AES, *i.e.* la première ligne de la matrice est fixe ; la seconde subie une rotation d'un cran vers la gauche ; la troisième de deux crans ; la quatrième de trois crans.

– $MixColumns$ est décrite dans [GLP06]. Plus précisément, la j-ième colonne de a, $0 \leq j < 4$, est remplacée par :

$$\begin{pmatrix} a_{3,j} \oplus a_{0,j} \oplus a_{1,j} \\ a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \\ a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \\ a_{2,j} \oplus a_{3,j} \oplus a_{0,j} \end{pmatrix}.$$

Même si cette fonction n'a pas une diffusion optimale, son *branch number* vaut 4, et son évaluation est réellement plus rapide que le $MixColumns$ de l'AES : elle peut être implémenté avec seulement 6 `XOR` sur des mots de 32 bits.

La figure 9.3 présente un schéma de l'extraction de X-FCSR-128 v2.

9.2.3 La boîte S

La boîte S a été choisie selon les contraintes décrites dans [GLP06], sauf qu'ici les étapes sont effectuées de $\mathbb{F}_2^8$ dans lui-même. La table 9.1 présente la boîte S en notation hexadécimale. Elle a été choisie pour avoir :

– une bonne résistance aux attaques différentielles et à la cryptanalyse linéaire,

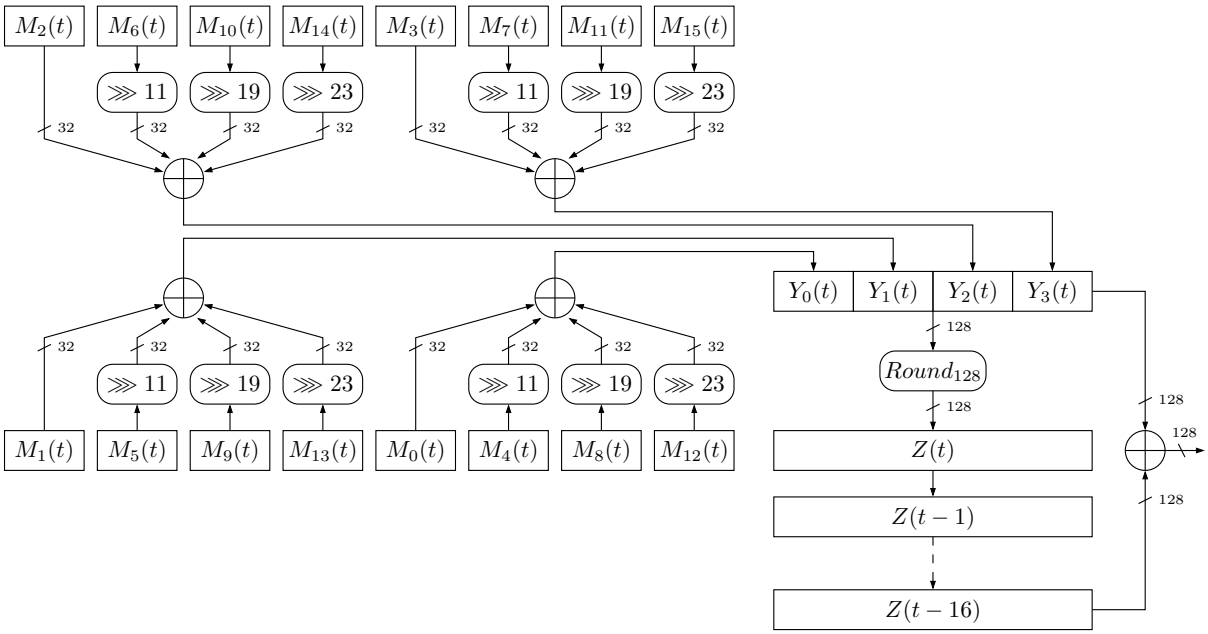


FIGURE 9.3 – Principe de l'extraction dans X-FCSR-128 v2

- un haut degré algébrique,
- un ordre de non-linéarité élevé,
- un degré entre les entrées et les sorties égales à 3.

Elle a été générée en utilisant le *key schedule* KSA de RC4 ([Riv92]) initialisé avec la clef de 26 octets « `To design our streamcipher` », puis en l'itérant 48574 fois.

52	c3	45	ce	9	cf	a8	f8	fd	ab	b8	6d	95	2	31	8
56	f4	cb	40	61	7	12	39	62	bb	ef	5d	3a	a9	fb	2c
78	ad	75	77	10	ca	55	66	9e	65	7b	9b	13	76	c7	1c
71	d	18	3f	50	6c	28	64	a3	b7	d0	be	e6	9c	b9	94
fc	bc	a1	cd	3b	48	4c	99	cc	3e	79	24	f2	c1	da	d8
de	f	e8	67	2e	16	53	c4	9d	57	c0	4f	f0	d6	4e	81
69	8a	ae	f9	8b	ee	43	3d	e4	23	97	68	b	32	e1	b2
ec	e9	59	1	c2	34	b5	1f	2a	29	d7	d5	b0	96	11	c6
7d	91	2d	72	8f	87	1d	e7	ba	19	25	15	5e	d9	98	70
4a	ed	51	a6	88	86	58	c5	5f	eb	49	0	ff	1b	2f	6a
82	1a	af	9f	8c	6b	a2	f1	e	5	7f	73	92	3c	f5	d2
54	14	ac	83	20	90	c9	22	fa	74	d3	27	37	38	a5	33
85	6	4	b3	e2	5b	e3	47	1e	8d	4b	b1	36	46	bd	35
dc	6e	d1	7c	a7	41	c	42	a0	aa	26	5a	4d	e5	5c	80
21	3	f3	63	ea	44	dd	89	8e	7e	b4	30	a	a4	60	f6
bf	fe	e0	f7	c8	d4	9a	db	84	7a	6f	2b	b6	17	93	df

TABLE 9.1 – La boîte S de X-FCSR-128

Elle a les caractéristiques suivantes :

- son meilleur chemin différentiel est $DP(S) = \max_{a,b \in (\mathbb{F}_2^8)^2 \setminus \{0,0\}} \#\{x,\ S(x \oplus a) \oplus S(x) = b\} = 10$.
- son meilleur chemin linéaire est $LP(S) = \max_{a,b \in (\mathbb{F}_2^8)^2 \setminus \{0,0\}} \left|\#\{x,\ a \cdot S(x) = b \cdot x\} - 128\right| = 32$.
- son degré algébrique vaut 7.

– son ordre de non-linéarité vaut 6.
– le degré entre les entrées et les sorties vaut 3. Il n'y a pas de relations de degré 2.

9.2.4 *Key/IV setup*

Comme présenté pour F-FCSR-H v3, le fait d'utiliser un *ring* FCSR ne permet pas d'assurer que l'automate ne va pas perdre de l'entropie [Röc08]. Cependant, les mêmes arguments permettent d'assurer que la perte d'entropie (estimée à un bit [Röc08]) ne permet pas de construire une attaque par compromis temps/mémoire.

Nous avons divisé le *key/IV setup* en deux étapes (comme pour X-FCSR-128 v1) pour accélérer l'injection de l'IV :
– la préparation des clefs (*key schedule*) qui dépend de la clef mais pas de l'IV ;
– l'injection de l'IV qui utilise la sortie du *key schedule* et l'IV.

Ainsi, l'injection d'IV pour une clef fixée est moins coûteuse qu'un *key/IV setup* entier. Ceci améliore le design car, en pratique, les changements d'IVs sont plus fréquents que les changements de clefs.

Key schedule

Nous choisissons d'utiliser un *key schedule* classique de chiffrement par blocs. Nous nous inspirons de celui du DES [Eps03] car il résiste aux attaques par clefs apparentées (*related key attacks*) [Bih94] et aux attaques rectangles (*related key rectangle attacks*) [HKLP05]. L'expansion de clef produit 25 sous-clefs de 128 bits notées $K_0, \ldots, K_{24}$. Elle se déroule ainsi :
– La sous-clef K_0 est déduite de la clef maître K :

$$K_0 = Round_{128}(K) \lll 23$$

où $\lll j$ est la rotation de j positions vers la gauche d'un mot de 128 bits.
– Pour $0 < i \leq 24$, la sous-clef K_i est déduite de la clef K_{i-1} :

$$K_i = Round_{128}(K_{i-1} \lll j) \text{ avec } j = \begin{cases} 23 & \text{si } i \equiv 3[4] \\ 1 & \text{sinon} \end{cases}$$

Injection d'IVs

Si nécessaire, l'IV est étendue pour faire 128 bits en ajoutant des 0. On considère alors le mot de 128 bits constitué en `XOR`ant l'IV et la sous-clef K_0. Puis, on applique $Round_{128}$ à cette valeur, et on ajoute la sous-clef suivante. Cette opération est répétée 24 fois. Plus précisément :
– $V_0 = IV \oplus K_0$;
– Pour $i = 1$ à 24, $V_i = Round_{128}(V_{i-1}) \oplus K_i$

Les valeurs V_{12}, V_{16} V_{20} et V_{24} sont alors utilisées pour initialiser le registre principal du FCSR : $M(0) = (V_{12}|V_{20}|V_{16}|V_{24})$. Le registre des retenues est initialisé à 0. Le FCSR est alors mis à jour 16 fois pour remplir les seize registres de mémoire nécessaires à l'extraction.

9.2.5 Justification du design

Dans [BG07], les auteurs prouvent que le *key/IV setup* (paramétré par la clef) d'un chiffrement à flot dépendant de l'IV doit être une fonction pseudo-aléatoire pour atteindre un niveau de sécurité suffisant. Nous avons essayé de respecter cette contrainte dans notre *key/IV setup* en le construisant comme un chiffrement par bloc en utilisant la fonction $Round_{128}$. Sous cette condition, la clef secrète du chiffrement ne peut pas être retrouvée facilement à partir de l'état initial du générateur. Si un attaquant récupère l'état initial, il est seulement capable de générer

la suite chiffrante correspondante à cette clef et cette IV. De plus, l'utilisation d'un *ring* FCSR implique qu'il puisse y avoir des collisions après le *key/IV setup* à cause de la perte d'entropie. Dans X-FCSR-128 v2, trouver une collision est aussi difficile que de cryptanalyser un chiffrement par bloc (*i.e.* inverser un chiffrement par bloc sans connaître la clef).

Les bonnes propriétés statistiques (période, séquences équilibrées, etc) sont assurées par la structure 2-adique du FCSR. De plus, l'utilisation d'un *ring* FCSR rend l'attaque présentée dans [SHJ09] inefficace : la LFSRization du FCSR n'est pas possible, comme expliqué précédemment.

La fonction $Round_{128}$ a été choisie pour sa bonne diffusion et ses propriétés de non-linéarité. L'utilisation de 16 registres de mémoires lors de l'extraction est un bon compromis entre une meilleure sécurité et un impact limité sur les performances. De plus, cela introduit une grande non-linéarité sur la sortie. En effet, même s'il existe des dépendances entre $Y(t)$ et $Y(t-16)$, il est pratiquement impossible de déterminer les valeurs du registre principal au temps t à partir des valeurs au temps $t-16$, sans connaître le vecteur de retenues.

9.2.6 Résistance contre les attaques connues

Grâce à l'utilisation de la fonction $Round_{128}$ et des 16 registres de mémoires lors de l'extraction, les attaques par corrélation et algébriques sont clairement hors d'atteinte. Tout d'abord l'utilisation d'un FCSR, dont la fonction de transition est non-linéaire sur $\mathbb{F}_2$, est un obstacle pour monter une attaque par corrélation. De plus, comme vu dans le chapitre 8, les attaques par corrélation et algébriques ne sont pas efficaces contre F-FCSR-H v3 qui utilise un simple filtre linéaire lors de l'extraction. Nous utilisons ici une extraction bien plus complexe. En particulier :

- la fonction $Round_{128}$ ayant une bonne diffusion, une différence locale dans l'état du FCSR produit une différence sur l'ensemble des bits produits. En particulier, les éventuelles corrélations existantes entre les cellules du FCSR sont détruites par cette fonction ;
- l'utilisation des 16 registres de mémoires augmente considérablement le degré et le nombre d'inconnues dans les équations algébriques décrivant l'automate.

Pour ces raisons, ces attaques ne sont pas applicables à X-FCSR-128 v2.

Les attaques par compromis temps/mémoire ont été traitées dans la sous-section traitant du *key/IV setup*.

Nous nous intéressons ici à l'attaque présentée dans [SHJ09]. Comme pour [HJ08], cette attaque se base sur la corrélation qui existe entre les retenues d'un FCSR en mode Galois. L'utilisation d'un *ring* FCSR décorrèle les retenues comme décrit pour F-FCSR-H v3. Pour résumer les analyses précédentes, l'utilisation d'un *ring* FCSR empêche les attaques précédentes, car les dépendances qui existaient précédemment dans un FCSR en mode Galois n'existent plus. Plus précisément, les dépendances ne sont plus localisées, mais sont réparties sur toutes les cellules, produisant ainsi un meilleur comportement non-linéaire. Nous pensons que les attaques traditionnelles exploitant des relations linéaires de la fonction de transition ne sont pas réalistes ici. Ainsi, cryptanalyser les *ring* FCSRs nécessite un nouveau type d'attaque utilisant d'autres relations.

9.2.7 Performances

Nous avons intégré X-FCSR-128 v2 dans le *benchmark* d'eSTREAM [dC05]. La table 9.2 présente les performances des chiffrements X-FCSR-128, ainsi que celles des finalistes d'eSTREAM orientés logiciels.

Le gain entre X-FCSR-128 v1 et v2 est d'environ 4 cycles par octet produit. X-FCSR-128 v2 est actuellement le chiffrement à flot basé sur un FCSR le plus rapide. Cependant, il est encore trois fois plus lent que les chiffrements retenues par eSTREAM [BDCC+08].

	cycles/octet				cycles/key	cycles/IV
Chiffrement	Keystream speed	40 octets	576 octets	1500 octets	Key setup	IV setup
Rabbit	2.35	17.94	3.06	2.81	412.04	347.57
HC-128	2.38	502.69	36.84	15.70	54.40	19851.28
Salsa 20/12	2.56	12.67	2.80	3.02	27.27	16.47
Sosemanuk	3.43	25.16	6.07	5.08	793.65	651.30
X-FCSR-128 v2	7.54	114.70	16.16	11.59	1478.94	3968.20
X-FCSR-128 v1	11.21	78.37	15.61	15.17	1256.70	2954.88
AES-CTR (128)	12.4	18.27	12.64	12.52	336.54	16.73

TABLE 9.2 – Performances calculées en utilisant le *benchmark* d'eSTREAM [dC05].

Conclusion et perspectives

Cette thèse m'a permis d'aborder les chiffrements à flot avec un concept très théorique : la topologie π-adique. Notre approche a consisté à combiner cette théorie avec celle des automates linéaires. Ainsi, nous sommes capables de décrire des AFSRs de façon matricielle. En particulier, cette description permet de ne plus être limité au seul AFSR en mode Fibonacci précédemment connu. Cependant, cette approche nécessite assez rapidement d'être spécifiée en choisissant un anneau π-adique.

Dans le cas des LFSRs, l'application de notre approche permet de décrire les automates avec des matrices à coefficients polynomiaux. Cette description permet une écriture plus concise des automates et permet de considérer plusieurs implémentations liées à un même automate. Cependant, l'implémentation d'un automate défini par des matrices à coefficients polynomiaux se révèle peu pratique, car elle conduit parfois à des implémentations avec plus de cellules mémoires que nécessaires. Nous avons également étudié les contraintes liées à l'implémentation des LFSRs « classiques », *i.e.* définis par des matrices à coefficients binaires.

Dans le cas FCSRs, cette approche permet de décrire des automates inconnus auparavant. En effet, les FCSRs étaient limités aux mode Galois et Fibonacci. La représentation matricielle accorde plus de libertés dans le design des FCSRs. Nous avons également proposé de considérer des FCSRs définis par des matrices à coefficients ternaires : 0, 1 ou -1. Ces automates s'implémentent en utilisant des soustracteurs à retenue. Ceux-ci s'obtiennent simplement en adjoignant une porte `NOT` à l'entrée d'un additionneur à retenue. Ces FCSRs peuvent permettre de réduire le coût de certains FCSRs, en particulier les FCSRs en mode Galois et Fibonacci. Nous avons également étudié le coût des implémentations matérielles et logicielles des FCSRs en représentation matricielle.

Nous avons appliqué nos travaux sur les FCSRs pour décrire deux chiffrements à flot : F-FCSR-H v3 et X-FCSR-128 v2. Ces nouvelles versions de chiffrements à flot utilisent des *ring* FCSRs possédant une implémentation matérielle (respectivement logicielle) efficace pour F-FCSR-H v3 (respectivement X-FCSR-128 v2). De plus, le fait de ne plus utiliser de FCSRs en mode Galois permet de se prémunir contre les attaques par LFSRization.

De nombreux aspects de cette approche n'ont pas été développé dans cette thèse. Par exemples :

- Puisque l'on construit maintenant des AFSRs avec des entrées, il pourrait être intéressant d'utiliser ces automates pour débiaiser une séquence réellement aléatoire, *i.e.* , est-il possible, grâce à un AFSR prenant en entrée une suite aléatoire biaisée, de produire une suite équilibrée ?
- Les LFSRs sont des AFSRs sur l'anneau des séries formelles. Que se passe-t-il lorsque l'on munit cet anneau du produit de Hadamard (produit terme à terme) ? En particulier, dans le cas binaire, le produit de Hadamard s'implémente grâce à une porte `AND`. Est-il alors possible de caractériser le comportement des circuits utilisant des portes `XOR` et des portes `AND` ?
- Dans le cas des FCSRs, il est maintenant possible de construire des automates possédant n cellules dans le registre principal, et produisant des séquences avec une période plus grande

que 2^{n+1} (dans le cas des FCSRs en mode Galois et Fibonacci, cela n'est pas possible). Cela signifie que l'on stocke plus d'informations dans les retenues. Jusqu'où est-il possible d'aller dans ce sens ?

De plus, cette nouvelle approche pose aussi des problèmes actuellement sans réponse. En particulier, nous ne connaissons pas d'algorithme pour construire un automate avec une implémentation matérielle et/ou logicielle efficace avec un entier de connexion donné. Un tel algorithme permettrait de trouver un automate plus rapidement qu'avec les algorithmes probabilistes. De plus, il permettrait aussi, étant donné un automate existant, de proposer une implémentation alternative plus efficace, sans pour autant changer les séquences produites.

« *Le hasard n'est que la mesure de notre ignorance.* »
Alfred Capus

Bibliographie

[AB05a] F. Arnault et T.P. Berger : Design and Properties of a New Pseudorandom Generator Based on a Filtered FCSR Automaton. *IEEE Transaction on Computers*, 54(11):1374–1383, 2005. 4.2.3, 8, 8.1.1, 8.1.2

[AB05b] F. Arnault et T.P. Berger : F-FCSR : design of a new class of stream ciphers. *In Fast Software Encryption*, pages 83–97. Springer, 2005. 4.2.3, 8, 8.1.1

[ABL06] F. Arnault, T.P. Berger et C. Lauradoux : Update on F-FCSR Stream Cipher. ECRYPT - Network of Excellence in Cryptology, Call for stream Cipher Primitives - Phase 2 2006. http://www.ecrypt.eu.org/stream/. 4.2.3, 8, 8.1.2, 8.3.4

[ABL08] F. Arnault, T. Berger et C. Lauradoux : F-FCSR stream ciphers. *New Stream Cipher Designs*, pages 170–178, 2008. 1.4, 8.1.1, 8.3.2, 8.3.4

[ABL+09] F. Arnault, T.P. Berger, C. Lauradoux, M. Minier et B. Pousse : A New Approach for FCSRs. *In* M.J. Jacobson Jr., V. Rijmen et R. Safavi-Naini, éditeurs : *Selected Areas in Cryptography*, volume 5867 de *Lecture Notes in Computer Science*, pages 433–448. Springer, 2009. 7.2, 7.2.2, 7.2.3, 8, 8.3

[ABLM07] F. Arnault, T.P. Berger, C. Lauradoux et M. Minier : X-FCSR - A New Software Oriented Stream Cipher Based Upon FCSRs. *In* K. Srinathan, C. Pandu Rangan et Moti Yung, éditeurs : *INDOCRYPT*, volume 4859 de *Lecture Notes in Computer Science*, pages 341–350. Springer, 2007. 4.2.3, 9, 9.1, 9.2.2

[ABM08] F. Arnault, T.P. Berger et M. Minier : Some Results on FCSR Automata With Applications to the Security of FCSR-Based Pseudorandom Generators. *IEEE Transactions on Information Theory*, 54(2):836–840, 2008. 4.2.1, 4.2.3, 8.2, 8.3.3

[Bab95] S. Babbage : Improved "exhaustive search" attacks on stream ciphers. *In Security and Detection, 1995., European Convention on*, pages 161–166, 1995. 1.3.3, 1.3.5

[BBC+08] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin et H. Sibert : Sosemanuk, a fast software-oriented stream cipher. *In* M. J. B. Robshaw et O. Billet, éditeurs : *The eSTREAM Finalists*, volume 4986 de *Lecture Notes in Computer Science*, pages 98–118. Springer, 2008. 4.1.5

[BBS86] L. Blum, M. Blum et M. Shub : A simple unpredictable pseudo-random number generator. *SIAM Journal on computing*, 15:364, 1986. 1.3.2

[BDCC+08] S. Babbage, C. De Canniere, A. Canteaut, C. Cid, H. Gilbert, T. Johansson, M. Parker, B. Preneel, V. Rijmen et M. Robshaw : The eSTREAM Portfolio (rev. 1), 2008. http://www.ecrypt.eu.org/stream/. 1.4, 9.2.7

[BG85] M. Blum et S. Goldwasser : An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information. Advances in Cryptology : CRYPTO'84. *Lecture Notes in Computer Science*, 196:289–299, 1985. 1.1

[BG07] C. Berbain et H. Gilbert : On the security of IV dependent stream ciphers. *In Fast Software Encryption*, pages 254–273. Springer, 2007. 9.2.5

[Bih94] E. Biham : New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 7(4):229–246, 1994. 9.2.4

[BKN09] A. Biryukov, D. Khovratovich et I. Nikolić : Distinguisher and related-key attack on the full AES-256. *Advances in Cryptology-CRYPTO 2009*, pages 231–249, 2009. 9.2.7

[Blu01] Bluetooth : Specification of the bluetooth system, volume 1 : Core, v1.1. Bluetooth SIG, February 2001. 6.1.4

[BM84] M. Blum et S. Micali : How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. Comput.*, 13(4):850–864, 1984. 1.2, 1.2.1

[BM05] T. Berger et M. Minier : Two algebraic attacks against the F-FCSRs using the IV mode. *Progress in Cryptology-INDOCRYPT 2005*, pages 143–154, 2005. 8.1.2, 8.3.4

[BMP09] T.P. Berger, M. Minier et B. Pousse : Software Oriented Stream Ciphers Based upon FCSRs in Diversified Mode. *In* B.K. Roy et N. Sendrier, éditeurs : *INDOCRYPT*, volume 5922 de *Lecture Notes in Computer Science*, pages 119–135. Springer, 2009. 7.2.3, 7.2.3, 9

[CJS00] V. Chepyzhov, T. Johansson et B. Smeets : A simple algorithm for fast correlation attacks on stream ciphers. *In Fast Software Encryption*, pages 124–135. Springer, 2000. 1.3.3, 1.3.5

[CM96a] K. Cattell et J.C. Muzio : Analysis of One-Dimensional Linear Hybrid Cellular Automata over GF(q). *IEEE Trans. Comput.*, 45(7):782–792, 1996. 4.1.3

[CM96b] K. Cattell et J.C. Muzio : Synthesis of One-Dimensional Linear Hybrid Cellular Automata. *IEEE Trans. Computer-Aided Design*, 15:325–335, 1996. 4.1.3

[CM98] K. Cattell et J.C. Muzio : An Explicit Similarity Transform between Cellular Automata and LFSR Matrices. *Finite Fields and Their Applications*, 4(3):239 – 251, 1998. 4.1.3, 4.1.3

[CM03] N. Courtois et W. Meier : Algebraic Attacks on Stream Ciphers with Linear Feedback. Advances in cryptology–EUROCRYPT 2003. *Lecture Notes in Computer Science*, 2656:345–359, 2003. 1.3.3, 1.3.5

[DB69] NG De Bruijn : A combinatorial problem. *Kibern. Sb., Nov. Ser.*, 6:33–40, 1969. 2.2.2

[dC05] C. de Canniere : eSTREAM Optimized Code HOWTO, 2005. 7.2.3, 7.3, 9.2.7, 9.2

[DOJ] C. Douillard, G. Ouvradou et M. Jézéquel : Cours d'électronique numérique. *Telecom Bretagne*. http://public.enst-bretagne.fr/~douillar/ELP304/ELP304_poly2.pdf. 3.2

[DP05] J.L.J. Daemen et B. Preneel : Chosen Ciphertext Attack on SSS. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/044, 2005. 1.1.3

[EH07] N.M. Ebeid et A. Hasan : On binary signed digit representations of integers. *Des. Codes Cryptography*, 42(1):43–65, 2007. 7.1.5

[EJ02] P. Ekdahl et T. Johansson : A new version of the stream cipher SNOW. *In Selected Areas in Cryptography – SAC 2002*, volume 2295 de *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2002. 4.1.5, 6.2.3, 6.2.3, 6.2.3

[Eps03] M. Epstein : DES encryption system, février 25 2003. US Patent 6,526,505. 9.2.4

[FMS08] S. Fischer, W. Meier et D. Stegemann : Equivalent representations of the F-FCSR keystream generator. *In ECRYPT Network of Excellence-SASC Workshop*, pages 87–94, 2008. 8.3.4

[GK02] M. Goresky et A. Klapper : Fibonacci and Galois representations of feedback-with-carry shift registers. *IEEE Transactions on Information Theory*, 48(11):2826–2836, 2002. 4.2, 4.2.1, 4.2.2

[GK09] M. Goresky et A. Klapper : Algebraic Shift Register Sequences, 2009. http://cs.engr.uky.edu/~klapper/algebraic.html. 1.2.3, 1.2.4, 1.3.1, 2.2.5, 2.2.7, 2.3.2, 2.5, 2.5.1, 2.5, 2.6, 2.6.2, 2.6.4, 4.1.3, 4.3, 4.3.3, 6.2

[GLP06] L. Granboulan, É. Levieil et G. Piret : Pseudorandom permutation families over abelian groups. *In Fast Software Encryption*, pages 57–77. Springer, 2006. 9.1, 9.2.2, 9.2.3

[GW82] S.W. Golomb et L.R. Welch : *Shift register sequences*. Aegean Park Press Laguna Hills, CA, 1982. Première publication en 1967. 1.2.2, 1.2.6, 2.5, 2.5.1, 2.5.2, 4, 4.1, 4.1.2, 6.2

[GW96] I. Goldberg et D. Wagner : Architectural considerations for cryptanalytic hardware. 1996. 4.1.3

[HJ08] M. Hell et T. Johansson : Breaking the F-FCSR-H stream cipher in real time. *Advances in Cryptology-ASIACRYPT 2008*, pages 557–569, 2008. 1.4, 4.2.3, 8, 8.2, 8.3.2, 8.3.3, 8.3.4, 9.2.6

[HK05] J. Hong et W.-H. Kim : TMD-Tradeoff and State Entropy Loss Considerations of Streamcipher MICKEY. *In* Subhamoy Maitra, C. E. Veni Madhavan et Ramarathnam Venkatesan, éditeurs : *INDOCRYPT*, volume 3797 de *Lecture Notes in Computer Science*, pages 169–182. Springer, 2005. 8.3.3

[HKLP05] S. Hong, J. Kim, S. Lee et B. Preneel : Related-key rectangle attacks on reduced versions of SHACAL-1 and AES-192. *In Fast Software Encryption*, pages 368–383. Springer, 2005. 9.2.4

[HR02] P. Hawkes et G.G. Rose : Guess-and-determine attacks on snow. *In* K. Nyberg et H. M. Heys, éditeurs : *Selected Areas in Cryptography*, volume 2595 de *Lecture Notes in Computer Science*, pages 37–46. Springer, 2002. 6.2.3

[HVM04] D. Hankerson, S. Vanstone et A. Menezes : *Guide to elliptic curve cryptography*. Springer-Verlag New York Inc, 2004. 7.1.3

[JD06] A. Joux et P. Delaunay : Galois LFSR, embedded devices and side channel weaknesses. *Progress in Cryptology-INDOCRYPT 2006*, pages 436–451, 2006. 4.1.3

[JM06a] E. Jaulmes et F. Muller : Cryptanalysis of the F-FCSR stream cipher family. *In Selected Areas in Cryptography*, pages 20–35. Springer, 2006. 8.1.2, 8.3.4

[JM06b] A. Joux et F. Muller : Chosen-ciphertext attacks against Mosquito. *In Fast Software Encryption*, pages 390–404. Springer, 2006. 1.1.3

[Kag06] D. Kagaris : A similarity transform for linear finite state machines. *Discrete Applied Mathematics*, 154(11):1570–1577, 2006. 3.1.4, 4.1.3, 4.1.3

[KG93a] A. Klapper et M. Goresky : 2-adic shift registers. *In Fast Software Encryption*, pages 174–178. Springer, 1993. 4

[KG93b] A. Klapper et M. Goresky : 2-adic shift registers. *In Fast Software Encryption - FSE'93*, Lecture Notes in Computer Science 809, pages 174–178. Springer-Verlag, 1993. 4.2

[KG97] A. Klapper et M. Goresky : Feedback shift registers, 2-adic span and combiners with memory. *Journal of Cryptology*, 10(2):111–147, 1997. 2.6, 2.6.1, 4.2

[Kob97] N. Koblitz : *p-adic numbers, p-adic analysis and Zeta-Functions*. Springer-Verlag, 1997. 2.6, 2.6.1, 4.2

[KR07] L.R. Knudsen et V. Rijmen : Known-key distinguishers for some block ciphers. *In Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, pages 315–324. Springer-Verlag, 2007. 9.2.7

[KW02] L. Knudsen et D. Wagner : Integral cryptanalysis. *In Fast Software Encryption*, pages 629–632. Springer, 2002. 9.2.7

[KX99] A. Klapper et J. Xu : Algebraic feedback shift registers. *Theoretical Computer Science*, 226(1):61–92, 1999. 4, 4.3.2

[Lau09] C. Lauradoux : Extended windmill polynomials. *In ISIT'09 : Proceedings of the 2009 IEEE international conference on Symposium on Information Theory*, pages 1120–1124, Piscataway, NJ, USA, 2009. IEEE Press. 5.7, 6.2

[L'E96] P. L'Ecuyer : Maximally equidistributed combined Tausworthe generators. *Math. Comput.*, 65(213):203–213, 1996. 6.2.1

[LP73] T. G. Lewis et W. H. Payne : Generalized feedback shift register pseudorandom number algorithm. *J. ACM*, 20(3):456–468, 1973. 4.1.5

[LSRQ05] P. Leglise, F.X. Standaert, G. Rouvroy et J.J. Quisquater : Efficient implementation of recent stream ciphers on reconfigurable hardware devices. *In 26th Symposium on Information Theory in the Benelux*, pages 261–268, 2005. 4.1.3

[Mar96] G. Marsaglia : DIEHARD : a battery of tests of randomness. 1996. http://stat.fsu.edu/~geo/diehard.html. 1.2.2, 1.2.2

[Mar03] G. Marsaglia : Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003. 4.1.5

[Mas69] J. Massey : Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969. 1.3.3

[Mea55] G. H. Mealy : A Method for Synthesizing Sequential Circuits. *Bell Systems Technical Journal*, 34:1045–1076, 1955. 3.1, 3.1.2

[MK92] M. Matsumoto et Y. Kurita : Twisted GFSR generators. *ACM Trans. Model. Comput. Simul.*, 2(3):179–194, 1992. 4.1.5

[MN98] M. Matsumoto et T. Nishimura : Mersenne twister : A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998. 4.1.5

[Moo56] E. F. Moore : Gedanken-experiments on Sequential Machines. *Automata Studies,Annals of Mathematical Studies (Princeton, N.J. : Princeton University Press)*, 34:129–153, 1956. 3.1, 3.1.1

[MP08] M. Minier et B. Pousse : Construction of FCSR algebraic equations and empirical analysis. *In ECRYPT Network of Excellence-SASC Workshop*, 2008. 8.3.4

[MRT04] G. Mrugalski, J. Rajski et J. Tyszer : Ring generators-new devices for embedded test applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(9):1306–1320, 2004. 4.1, 4.1.4, 6.2.2

[MS89] W. Meier et O. Staffelbach : Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989. 1.3.3, 1.3.5

[Nie95] H. Niederreiter : The multiple-recursive matrix method for pseudorandom number generation. *Finite Fields Appl.*, 1(1):3–30, 1995. 4.1.5

[PL05] F. Panneton et P. L'Ecuyer : On the xorshift random number generators. *ACM Trans. Model. Comput. Simul.*, 15(4):346–361, 2005. 4.1.5, 6.2, 6.2.1

[Riv92] R.L. Rivest : The RC4 Encryption Algorithm. RSA Data Security. *Inc., March*, 12, 1992. 9.2.3

[Röc08] A. Röck : Stream ciphers using a random update function : Study of the entropy of the inner state. *In* Serge Vaudenay, éditeur : *AFRICACRYPT*, volume 5023 de *Lecture Notes in Computer Science*, pages 258–275. Springer, 2008. 8.3.3, 9.2.4

[Rog89] Y. Roggeman : Varying feedback shift registers. *In EUROCRYPT*, volume 434 de *Lecture Notes in Computer Science*, pages 670–679. Springer-Verlag, 1989. 6.2

[RSN+01] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, BOOZ-ALLEN et HAMILTON INC MCLEAN VA : A statistical test suite for random and pseudorandom number generators for cryptographic applications. 2001. 1.2.2, 1.2.2

[SC88] B.J.M. Smeets et W.G. Chambers : Windmill Generators : A Generalization and an Observation of How Many There Are. *In EUROCRYPT*, pages 325–330, 1988. 5.7, 6.1.4, 6.1.4, 6.2

[Sha04] A. Shamir : Stream Ciphers : Dead or Alive ? *In Advances in Cryptology-ASIACRYPT*, 2004. 1.4

[SHJ09] P. Stankovski, M. Hell et T. Johansson : An efficient state recovery attack on X-FCSR-256. *In Fast Software Encryption*, pages 23–37. Springer, 2009. 4.2.3, 9, 9.1, 9.2.5, 9.2.6

[Sie84] T. Siegenthaler : Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30(5):776–, 1984. 1.3.3, 1.3.5

[Sto73] HS Stone : Discrete Mathematical Structures and their Applications. Sci. Res. *Associates, Chicago*, 1973. 2.5, 2.5.1, 4.1, 4.1.3

[SZ91] L. Schwartz et K. Zizi : *Analyse. 1. Théorie des ensembles et topologie*. Hermann, 1991. 2.3.1

[TV03] B. Tsaban et U. Vishne : Efficient linear feedback shift registers with maximal period. *CoRR*, cs.CR/0304010, 2003. 4.1.5

[Ver19] G.S. Vernam : Secret signaling system, 1919. US Patent 1,310,719. 1.1.1

[WM88] L.-T. Wang et E. J. McCluskey : Hybrid designs generating maximum-length sequences. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 7(1):91–99, 1988. 4.1.4

[Yao82] A.C.C. Yao : Theory and Applications of Trapdoor Functions (Extended Abstract). *In FOCS*, pages 80–91. IEEE, 1982. 1.2, 1.2.1

[ZWFB05] B. Zhang, H. Wu, D. Feng et F. Bao : Chosen ciphertext attack on a new class of self-synchronizing stream ciphers. *Progress in Cryptology-INDOCRYPT 2004*, pages 73–83, 2005. 1.1.3

Appendices

Travaux liés aux chiffrements par bloc et aux fonctions de hachage

Dans cette partie, nous incorporons les articles publiés et soumis liés aux chiffrements par bloc et aux fonctions de hachage. Ces travaux se fondent sur deux concepts :

– les propriétés intégrales des chiffrements par bloc [KW02] ;
– les distingueurs à clef connue [KR07], clef choisie [BKN09] de chiffrements par bloc.

Les propriétés intégrales sont des propriétés structurelles des chiffrements par bloc. Un chiffrement par bloc, pour une clef fixée, doit se comporter comme une fonction aléatoire. En considérant un ensemble de messages clairs et les chiffrés correspondant, il est possible de mettre en défaut ce comportement. Plus précisément, les chiffrements par bloc traitent souvent les données par octet. Alors, en faisant prendre les 256 valeurs possibles à certains octets (on dit qu'on les sature, ou encore que ces octets sont actifs), la somme des chiffrés n'est pas toujours aléatoire.

Les distingueurs à clef connue et choisie sont utiles lorsque les chiffrements par bloc sont utilisés pour construire une autre primitive cryptographique, les fonctions de hachage en particulier. Dans ce cas, il n'y a plus de clef secrète et l'on peut donc exploiter ces distingueurs.

De plus, ce domaine est actuellement très actif grâce à la tenue de la compétition SHA-3 [1]. Cette compétition, débutée en 2007, a pour but de choisir une nouvelle fonction de hachage comme standard. Plusieurs fonctions proposées utilisent des chiffrements par bloc comme permutation interne. Dans ce cas, l'utilisation des propriétés intégrales et des distingueurs à clef connue ou choisie peut permettre de construire des attaques contre ces fonctions.

Dans l'article *Distinguishers for Ciphers and Known Key attack against Rijndael with Large Blocks* présenté à « Africacrypt 2009 », nous formalisons la notion de distingueurs à clefs connues pour les chiffrements par bloc. Nous présentons également de tels distingueurs pour le chiffrement Rijndael, basés sur de nouvelles propriétés intégrales.

Dans l'article *Some results on Distinguishers with applications to SPNs* soumis au journal « IEEE-IT », nous formalisons les distingueurs à clef inconnue, connue et choisie. Nous présentons aussi une nouvelle classe de distingueurs dans le modèle des clefs connues et liées. Ce dernier type de distingueur est utilisé pour améliorer les distingueurs intégrales contre 7 tours de l'AES.

Dans *Integral distinguishers of some SHA-3 candidates* présenté à « CANS 2010 », nous étudions les propriétés intégrales des fonctions de compression réduites de candidats à la compétition SHA-3 : Hamsi-256, LANE-256 et Grøstl-512.

1. http ://csrc.nist.gov/groups/ST/hash/sha-3/index.html

Résumé

Dans cette thèse, nous nous intéressons à l'étude des chiffrements à flot basés sur l'utilisation de séquences algébriques. Plus précisément, nous utilisons dans notre approche la topologie π-adique et les automates linéaires.

Dans un premier temps, nos travaux présentent des résultats généraux basés sur l'utilisation d'un anneau π-adique quelconque. Puis nous traitons deux cas particuliers : lorsque l'anneau considéré est l'anneau des séries formelles, et lorsque celui-ci est l'anneau des entiers N-adiques. Dans ces deux cas, nous nous intéressons particulièrement aux contraintes liées à l'implémentation de tels automates, d'un point de vue matériel et logiciel.

Nous présentons ensuite comment il est possible d'améliorer la sécurité de chiffrements à flot basés sur des automates 2-adiques. Pour cela, nous appliquons nos travaux aux chiffrements à flot F-FCSR-H et X-FCSR-128.

Enfin, nous présentons des travaux réalisés au cours de cette thèse dans le domaine des chiffrements par bloc et des fonctions de hachage. Ces travaux se fondent sur les propriétés intégrales des chiffrements par bloc, ainsi que sur les notions de distingueurs à clef connue et choisie. La combinaison de ces deux approches a été appliquée aux fonctions de hachage Hamsi-256, LANE-256 et Grøstl-512, toutes trois candidates à la compétition SHA-3.

Mots clefs : cryptologie, chiffrement à flot, anneau π-adique, automate, AFSR, LFSR, FCSR, chiffrement par bloc, fonction de hachage, propriété intégrale, distingueur à clef connue, distingueur à clef choisie.

Abstract

In this thesis, we are interested in the study of stream ciphers based upon algebraic sequences. More precisely, our approach used π-adic topology and linear automata.

First, general results based on π-adic ring are presented. Then we specify the ring considered. We focus on the ring of formal power series, and on the ring of N-adic integers. In both case, constraints of implementation are especially studied, for hardware and software purpose.

Next, we present how these results may be used to enhance the security of stream ciphers based upon 2-adic automata. We focus on the stream ciphers F-FCSR-H and X-FCSR-128.

In last part, this thesis deals with additional works in block ciphers and hash functions. These results are based upon integral properties of block ciphers, and upon known key and chosen key distinguishers. The combination of these properties and distinguishers have been applied to the hash functions Hamsi-256, LANE-256 and Grøstl-512, which are submitted to the SHA-3 competition.

Keywords : cryptology, stream ciphers, π-adic ring, automaton, AFSR, LFSR, FCSR, block cipher, hash function, integral property, known key distinguisher, chosen key distinguisher.