DGA Maîtrise
de l'Information
Bruz (35)

Master Cryptologie et
Sécurité Informatique
Université Bordeaux 1

Rapport de stage

Reconstruction de brasseurs synchrones et auto-synchronisants

Marion Candau

Maître de stage : Pierre Loidreau

Tuteur universitaire: Gilles Zémor

— Du 31 mars au 16 septembre 2011 —

Table des matières

Introduction

En cryptologie, lorsque l'on veut assurer la confidentialité des données, il nous paraît indispensable de les chiffrer avant de les envoyer sur le réseau. Ainsi seuls la source et le destinataire peuvent avoir accès au contenu de ces données. On peut donc imaginer qu'un attaquant écoutant une communication non chiffrée a accès aux données qu'il écoute. Cependant, si l'attaquant n'a pas connaissance des spécificités de la chaîne de transmission, les données qu'il capture ne sont exploitables que s'il retrouve comment elles ont été transformées avant d'être envoyées par la source. En effet, la chaîne de transmission des données entre une source et un destinataire peut être modélisée par le schéma 1.

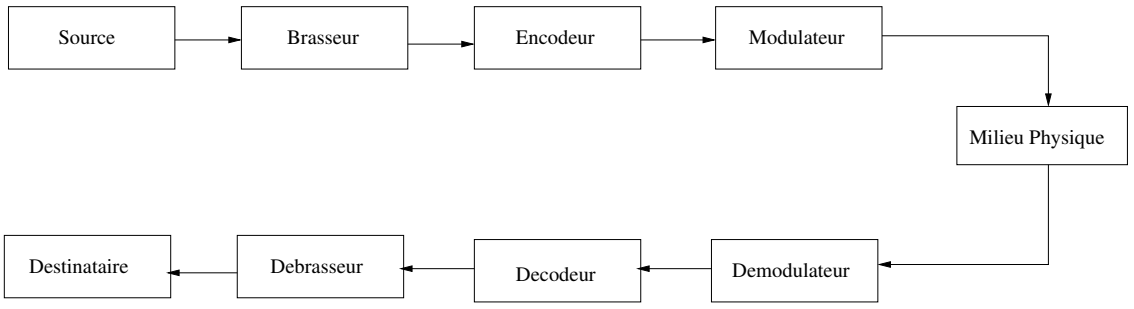

FIG. 1 – Schéma d'une chaîne de transmission

Sur ce schéma, on a différentes opérations qui sont effectuées sur le message à transmettre. Tout d'abord, la source doit délivrer des symboles appartenant à un alphabet, le plus souvent l'alphabet binaire. Le message numérique que la source va transmettre est donc la suite des symboles de cet alphabet. On peut caractériser la source par son nombre de symboles par secondes appelé débit alphabétique. De plus, en théorie de l'information, on définit également un débit d'information. Le lien entre ces deux débits dépend des propriétés statistiques de la source. Pour une source binaire idéale, le débit d'information est égal au débit alphabétique, c'est-à-dire que la source émet des symboles indépendants dont les valeurs sont équiprobables. Pour transmettre un message, il faut d'abord le transformer en une suite de symboles appartenant à l'alphabet de la source. Après cette opération, il se peut que le message n'ait rien d'aléatoire et donc ne convienne pas au modèle de la source idéale. Pour pallier à ce problème, on utilise l'opération de brassage du message numérique. Le rôle de cette opération est de réduire les modifications des caractéristiques du message émis, et par exemple, de diminuer la longueur des paquets de bits identiques qui pourraient occasionner des problèmes de synchronisation lors de la phase suivante qui est la phase d'encodage des données. Celle-ci est utilisée pour corriger les éventuelles erreurs qui peuvent survenir lors de la transmission. Enfin, la phase de modulation permet de transformer le message brassé et encodé en une forme adaptée à la transmission dans un milieu physique donné. A la réception du message, les opérations inverses sont effectuées

par le récepteur et le message est transmis au destinataire.

L'attaquant écoutant la communication doit donc connaître les paramètres de la modulation, de l'encodage et du brassage afin de remonter au message original. L'objectif de ce stage est de se mettre à la place de l'attaquant qui connaît les paramètres de la modulation et de l'encodage et qui cherche à retrouver ceux du brassage. On suppose, de plus, qu'il existe un biais dans le message à brasser. Cette hypothèse est réaliste puisque tous les schémas de codage utilisés en pratique satisfont cette hypothèse comme par exemple l'ASCII.

Dans ce rapport, nous allons tout d'abord parler du contexte dans lequel s'est déroulé ce stage. Puis nous verrons la reconstruction des paramètres d'un brasseur synchrone. Cette partie sur les brasseurs synchrones est divisée en deux sous-parties. La première sera consacrée à l'état de l'art c'est-à-dire à la technique de reconstruction du polynôme de rétroaction puis à celle de l'état initial décrite dans la thèse [1] de Mathieu Cluzeau. Dans la deuxième sous-partie sera abordée notre proposition de reconstruction des paramètres d'un brasseur synchrone basée sur la recherche exhaustive. Pour cela, nous verrons tout d'abord le calcul de tous les polynômes primitifs ou irréductibles candidats à être le polynôme de rétroaction du brasseur, et ensuite le test statistique permettant de déterminer si un état initial et un polynôme sont les paramètres d'un brasseur ou non. Enfin, nous parlerons de la reconstruction des paramètres d'un brasseur auto-synchronisant.

Contexte du stage

Ce stage s'est déroulé à la DGA Maîtrise de l'Information à Bruz dans l'agglomération rennaise. Mon maître de stage était Pierre Loidreau, membre du département Services Crypto (SCY) de la division Sécurité des Systèmes d'Information (SSI).

Ce site de la DGA à Bruz est un centre technique de très haut niveau qui apporte une expertise technique à la plupart des grands programmes d'armement dans une large gamme de domaine. La DGA-MI a hérité des domaines du CELAR auquel elle succède, c'est-à-dire la cryptologie, la sécurité des systèmes d'information, les composants électroniques, les radars et l'optronique. De plus, elle a élargi son expertise aux systèmes de navigation et aux missiles tactiques et stratégiques.

Ce centre, implanté depuis 1968, compte 900 personnes dont 60% d'ingénieurs et intervient sur des produits qui vont du composant électronique au « système de systèmes » et contribue aux fonctions stratégiques suivantes :

- la connaissance et l'anticipation (renseignement spatial, drônes, connaissance de la menace, ...)
- la protection (défense anti-missile balistique, alerte avancée, ...)
- l'intervention (avion de combat Rafale, missiles tactiques, ...)
- la dissuasion (missiles balistiques stratégiques, sous-marin nucléaire lanceur d'engins, ...)

Chapitre 1

Brasseur synchrone

Avant de parler de la reconstruction d'un brasseur synchrone, nous allons voir la définition et les différentes propriétés d'un tel brasseur. Son élément principal est un registre à décalage à rétroaction linéaire (LFSR) dont on additionne modulo 2 chaque bit s_t avec chaque bit x_t du message entrant comme représenté par le schéma 1.1.

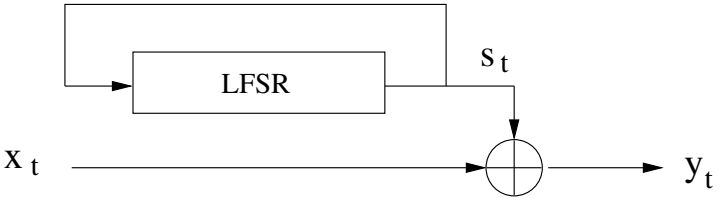

FIG. 1.1 – Brasseur synchrone

Nous allons donc d'abord introduire les registres à décalage à rétroaction linéaire avec des définitions et propriétés tirées de [1].

1.1 Définition et propriétés d'un LFSR

Définition 1. Un registre à décalage à rétroaction linéaire binaire de longueur L est composé d'un registre à décalage contenant L bits au temps t notés $(s_t, s_{t+1}, \ldots, s_{t+L-1})$, mis à jour par une fonction de rétroaction linéaire.

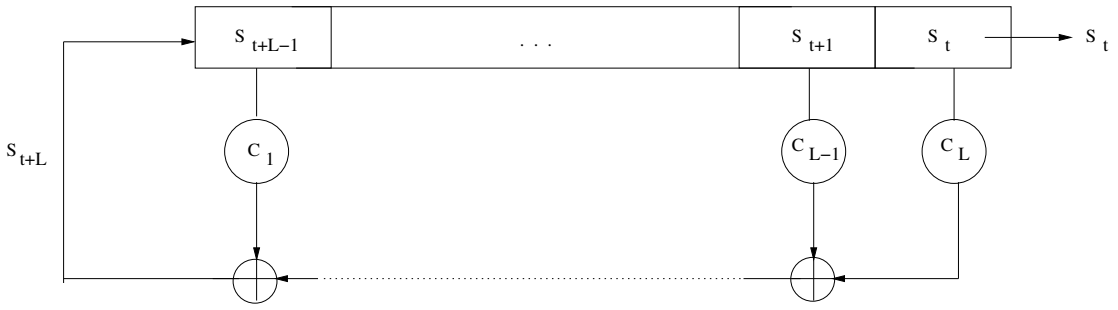

FIG. 1.2 – LFSR

A chaque bit d'horloge, on sort le bit t, on décale le contenu d'origine vers la droite et

dans la cellule de gauche, on met :

$$s_{t+L} = c_L s_t + c_{L-1} s_{t+1} + \ldots + c_1 s_{t+L-1}$$

avec $(c_1, c_2, \ldots, c_L) \in \mathbb{F}_2^L$, les cœfficients de rétroaction.

Définition 2. Les bits $(s_0, \ldots, s_{L-1})$ sont appelés l'état initial du registre et déterminent entièrement la suite.

Proposition 1. Toute suite binaire produite par un LFSR de longueur L est ultimement périodique, c'est-à-dire qu'il existe un entier t_0 tel que la suite $(s_t)_{t \geqslant t_0}$ est périodique. De plus, sa plus petite période T est inférieure ou égale à $2^L - 1$ et si $c_L = 1$ alors la suite est périodique.

Définition 3. On peut associer à une suite binaire $(s_n)_{n \geqslant 0}$ la série génératrice :

$$s(X) = \sum_{n \geqslant 0} s_n X^n$$

Si $(s_n)_{n \geqslant 0}$ est produite par un LFSR de longueur L alors son polynôme de rétroaction est défini par :

$$P(X) = 1 + c_1 X + c_2 X^2 + \ldots + c_L X^L$$

avec $(c_1, c_2, \ldots, c_L) \in \mathbb{F}_2^L$. De plus, son polynôme caractéristique est le polynôme réciproque du polynôme de rétroaction et est donc défini sur $\mathbb{F}_2[X]$ par :

$$P_c(X) = X^L + c_1 X^{L-1} + \ldots + c_L$$

Proposition 2. Une suite s est produite par un LFSR de polynôme de rétroaction $P(X) = 1 + c_1 X + c_2 X^2 + \ldots + c_L X^L$ si, et seulement si, son développement en série formelle est :

$$s(X) = \frac{g(X)}{P(X)}$$

avec $g \in \mathbb{F}_2[X]$, un polynôme de degré inférieur ou égal au degré de P. De plus, g est entièrement déterminé par l'état initial du registre :

$$g(X) = \sum_{t=0}^{L-1} X^t \times \left(\sum_{j=0}^{t} c_j s_{t-j} \right)$$

Proposition 3. Soit s une suite binaire produite par un LFSR de longueur L, son polynôme de rétroaction minimal est l'unique polynôme unitaire $P_0 \in \mathbb{F}_2[X]$ tel qu'il existe $g \in \mathbb{F}_2[X]$ avec $\deg(g) < \deg(P_0)$ et $PGCD(g, P_0) = 1$ tel que $s(X) = \dfrac{g(X)}{P_0(X)}$.
La complexité linéaire de s, notée $\Lambda(s)$ est égale au degré de P_0. C'est la longueur du plus petit LFSR qui produit s.

En conséquence, en prenant un polynôme irréductible sur $\mathbb{F}_2[X]$, la fraction ne pourra pas être réduite et on est certain de ne pas pouvoir générer la même suite avec un registre plus court. Un autre paramètre essentiel d'une suite générée par un LFSR est sa période qui est déterminée par l'ordre du polynôme de rétroaction.

Définition 4. Soit P un polynôme de $\mathbb{F}_2[X]$. Son ordre, noté $o(P)$ est le plus petit entier $r > 0$ tel que :

$$X^r = 1 \mod P(X)$$

Proposition 4. Soit $(s_t)_{t\geqslant 0}$ une suite binaire issue d'un LFSR de polynôme de rétroaction minimal P_0 et dont l'état initial est non nul. Alors sa plus petite période est égale à l'ordre de P_0.

Définition 5. Soit P un polynôme irréductible de $\mathbb{F}_2[X]$, de degré L. Il est dit primitif s'il est d'ordre $2^L - 1$.

Donc si on veut utiliser un LFSR optimal, on doit alors s'assurer que le polynôme de rétroaction de degré L est primitif. De plus, les suites générées par un LFSR avec de tels polynômes vérifient de bonnes qualités statistiques comme on le voit dans la proposition 5.

Proposition 5. Soit s une suite générée par un LFSR de longueur L. Si le polynôme de rétroaction P est primitif et si l'état initial est non nul alors :
- toutes les suites de longueur $l < L$ se retrouvent avec la même fréquence dans s
- toutes les suites binaires non nulles de longueur L apparaissent aussi avec la même fréquence.

Par conséquent, les suites produites par des LFSR avec des polynômes de rétroaction primitifs sont intéressantes et en pratique on choisit toujours un polynôme de rétroaction de ce type.

Un brasseur combine donc une suite issue d'un LFSR $(s_t)_{t\geqslant 0}$ et le message d'entrée $(x_t)_{t\geqslant 0}$ additionnés bit à bit modulo 2. Les éléments binaires brassés sont donc :

$$y_t = s_t \oplus x_t$$

L'opération de débrassage est identique à celle de brassage, elle réalise l'addition bit à bit du train binaire démodulé $(y'_t)_{t\geqslant 0}$ et d'une séquence issue du même LFSR que celle de l'émission. Les générateurs des LFSR du brasseur et du débrasseur doivent donc être identiques et donner la même suite avec la même phase. La synchronisation des deux générateurs est donc indispensable pour les faire fonctionner en phase.
Lorsque les deux générateurs sont synchronisés, on a l'égalité :

$$x_t \oplus x'_t = y_t \oplus y'_t$$

Une erreur sur un bit de la séquence à débrasser se reporte directement sur le bit correspondant après l'opération de débrassage, et uniquement sur celui-ci si les deux générateurs sont synchronisés.

Un brasseur synchrone est donc entièrement caractérisé par l'état initial et le polynôme de rétroaction du LFSR qui le compose, c'est donc ces deux paramètres que l'on va chercher à retrouver à partir de la suite de bits démodulée.

1.2 État de l'art de la reconstruction

L'état de l'art de la reconstruction d'un brasseur repose essentiellement sur la thèse de Mathieu Cluzeau [1], intitulée *Reconnaissance d'un schéma de codage*. Pour retrouver les paramètres du brasseur, il recherche dans un premier temps le polynôme de rétroaction du LFSR et dans un second temps l'état initial. Nous allons donc voir en premier lieu, la recherche du polynôme de rétroaction du brasseur synchrone.

1.2.1 Recherche du polynôme de rétroaction du brasseur

Nous rappelons l'hypothèse faite sur le message à brasser qui est l'existence d'un biais dans ce message. Faire une hypothèse sur le message est nécessaire car toute suite binaire est engendrée par une infinité de brasseurs linéaires. En effet, toute suite $(x_t)_{t\geqslant 0}$ est trivialement produite par tout brasseur synchrone à partir de l'entrée $(x_t \oplus s_t)_{t\geqslant 0}$ où $(s_t)_{t\geqslant 0}$ est la suite produite par le LFSR.
Cette hypothèse est :

$$\forall t \geqslant 0,\ P[x_t = 0] = \frac{1}{2} + \varepsilon,\ \varepsilon \neq 0$$

Dans l'exemple du codage en ASCII, chaque caractère est codé sur un octet et le bit de poids fort vaut toujours 0. Donc on a en moyenne $\frac{1}{2} + \frac{1}{16}$ zéros ce qui fait un biais de 0.0625. Pour les schémas classiques, les valeurs de biais sont de l'ordre de 0.1 ou 0.05.

1.2.1.1 Principe de l'algorithme

La technique de reconstruction du polynôme de rétroaction du brasseur présentée par Mathieu Cluzeau est similaire à la technique d'Anne Canteaut et Eric Filiol présentée dans [8] dans le cadre de la cryptanalyse d'une combinaison de LFSR. En effet, elle s'appuie sur le détection de n'importe quel multiple du polynôme de rétroaction, détection possible grâce au biais statistique de la suite de sortie. Cette technique s'appuie sur les théorèmes suivants dont toutes les démonstrations sont faites dans [1]. On va s'intéresser plus particulièrement au cas des multiples du polynôme de rétroaction de poids 3 car c'est le plus performant et donc celui utilisé en pratique.

Théorème 1. Soient $(z_t)_{t\geqslant 0}$ des variables aléatoires bornées telles que z_t et $z_{t'}$ sont indépendantes dès que $|t' - t| > d$ pour un certain d fixé. Considérons alors la variable aléatoire

$$S = \sum_{t=0}^{N} z_t$$

On note M sa moyenne et V sa variance. Alors $\frac{S-M}{\sqrt{V}}$ tend vers une loi normale centrée réduite lorsque N tend vers l'infini.

Théorème 2. Soit $(y_t)_{t\geqslant 0}$ la sortie d'un brasseur synchrone biaisé dont l'entrée $(x_t)_{t\geqslant 0}$ vérifie :

$$\forall t \geqslant 0,\ P[x_t = 0] = \frac{1}{2} + \varepsilon,\ \varepsilon \neq 0$$

Soit $Q \in \mathbb{F}_2[X]$ un trinôme

$$Q(X) = 1 + X^v + X^u, \text{ avec } v < u$$

Soit

$$z_t = y_t + y_{t-v} + y_{t-u} \quad \forall t \geqslant u$$

Notons

$$Z_N = \sum_{t=u}^{N-1} (-1)^{z_t}$$

et

$$\mu = 8(N-u)\varepsilon^3$$

$$\sigma^2 = (N-u)\left(1-64\varepsilon^6\right) + N_1 \times \left((2\varepsilon)^4 - (2\varepsilon)^6\right) + N_2 \times \left((2\varepsilon)^2 - (2\varepsilon)^6\right)$$

où N_1 désigne le nombre de monômes communs entre $Q(X)$ et $XQ(X)$ et N_2, le nombre de monômes communs entre $Q(X)$ et $X^2Q(X)$.
Si Q est un multiple du polynôme de rétroaction du LFSR, alors la variable aléatoire

$$Z = \frac{Z_N - \mu}{\sigma}$$

converge en loi vers une loi normale centrée réduite lorsque N tend vers l'infini.

On fait ensuite une approximation sur la variance afin de rendre la formule plus maniable.

Théorème 3. Avec les mêmes notations que dans le théorème précédent, on a la borne supérieure suivante :

$$\frac{\sigma^2}{N-u} \leqslant 13 \times \left(1-(2\varepsilon)^6\right)$$

Grâce à ces théorèmes, on va pouvoir détecter les multiples de poids 3 du polynôme de rétroaction P. En effet, si un polynôme Q n'est pas un multiple de P alors la variable aléatoire Z_N suit une loi normale centrée de variance $N-u$. On va donc tester la moyenne de Z_N et distinguer deux hypothèses :

- H_0 : $|Z_N|$ suit une loi normale de moyenne 0 et de variance $N-u$, c'est-à-dire que $Q(X) = 1 + X^v + X^u$ n'est pas un multiple de P.
- H_1 : $|Z_N|$ suit une loi normale de moyenne μ et de variance σ^2 c'est-à-dire que Q est un multiple de P.

On va utiliser un seuil de décision noté T avec $T > 0$ pour déterminer quelle hypothèse est vérifiée. Si $|Z_N| > T$ alors on accepte l'hypothèse H_1 sinon on accepte H_0. Le nombre de bits minimum de la suite requis dépend des probabilités de fausse-alarme et de non-détection qui correspondent aux deux types d'erreurs de décision décrits dans le tableau 1.3.

décision / réalité	H_0 acceptée	H_1 acceptée
H_0 vraie	OK	fausse alarme
H_1 vraie	non-détection	OK

FIG. 1.3 – Types d'erreurs associées aux décisions

Calculons maintenant ces probabilités. On a, pour cela, besoin de ϕ la fonction de répartition associée à la densité de la loi normale :

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt$$

On note P_f la probabilité de fausse-alarme et P_n celle de non-détection. On a :

$$\begin{aligned} P_f &= P[|Z_N| \geqslant T \mid H_0 \text{ vraie}] \\ &= 2 \times P\left[Z_N \leqslant -T \mid Z_N \sim \mathcal{N}(0, N-u)\right] \\ &= 2 \times \phi\left(\frac{-T}{\sqrt{N-u}}\right) \end{aligned}$$

et

$$\begin{aligned} P_n &= P\left[|Z_N| \leqslant T \mid H_1 \text{ vraie}\right] \\ &= \frac{1}{\sqrt{2\pi}} \int_{\frac{-T-\mu}{\sigma}}^{\frac{T-\mu}{\sigma}} e^{-\frac{x^2}{2}} dx \\ &= \phi\left(\frac{T-\mu}{\sigma}\right) - \phi\left(\frac{-T-\mu}{\sigma}\right) \\ &\simeq \phi\left(\frac{T-\mu}{\sigma}\right) \end{aligned}$$

car, dans la plupart des cas, $\phi\left(\frac{-T-\mu}{\sigma}\right)$ est négligeable devant $\phi\left(\frac{T-\mu}{\sigma}\right)$.

On a donc

$$\phi^{-1}(P_n) = \frac{T-\mu}{\sigma}$$

$$\phi^{-1}\left(1 - \frac{P_f}{2}\right) = \frac{T}{\sqrt{N-u}}$$

Notons $a = \phi^{-1}\left(1 - \frac{P_f}{2}\right)$ et $b = -\phi^{-1}(P_n)$ avec $a > 0$ et $b > 0$.
En utilisant les dernières équations et la borne sur la variance σ^2 et après plusieurs calculs intermédiaires détaillés dans [1], on a :

$$T \leqslant \frac{a^2 + ab\sqrt{13(1-64\varepsilon^6)}}{8\varepsilon^3} \lesssim \frac{a^2 + ab\sqrt{13}}{8|\varepsilon|^3}$$

$$N - u = \left(\frac{T}{a}\right)^2$$

On a simplifié cet algorithme de reconstruction en constatant que :

$$\begin{aligned} Z_N &= \sum_{t=u}^{N-1} (-1)^{z_t} \\ &= \sum_{t=u}^{N-1} (-1)^{y_t + y_{t-v} + y_{t-u}} \\ &= \sum_{t=0}^{N-u-1} (-1)^{y_{t+u} + y_{t+u-v} + y_t} \\ &= \sum_{t=0}^{n-1} (-1)^{y_{t+u} + y_{t+u-v} + y_t} \quad avec\ n = N-u \\ Z_N &= n - 2\sum_{i=0}^{n-1} y_i + y_{i+u-v} + y_{i+u} \end{aligned}$$

Par conséquent, on a l'algorithme 1.2.1.

Algorithme 1.2.1 Algorithme de recherche du polynôme de rétroaction d'un brasseur

Entrées: Une suite de N bits : $y_0, \ldots, y_{N-1}$,
trois réels $0 \leqslant P_f \leqslant 1$,
$0 \leqslant P_n \leqslant 1$
et $0 \leqslant \varepsilon < 0.5$.

Sorties: Soit échec si la taille de la suite d'entrée est trop petite
- soit un polynôme Q comme polynôme de rétroaction du brasseur et une liste de trinômes Tri
- soit une liste de trinômes Tri éventuellement vide.

1: $a = \phi^{-1}\left(1 - \frac{P_f}{2}\right)$ et $b = -\phi^{-1}(P_n)$
2: $T = \frac{a^2 + ab\sqrt{13}}{8\varepsilon^3}$ et $n = \frac{T^2}{a^2}$.
3: $Tri = \emptyset$
4: **Si** $N < n$ **Alors**
5: **Retourner** échec
6: **Sinon**
7: $u = 2,\ z = 0$
8: **Tant que** $u \leqslant N - n$ et $z = 0$ **Faire**
9: **Pour** $v = 1$ à $u - 1$ **Faire**
10: **Pour** $i = 0$ à $n - 1$ **Faire**
11: $z = z + ((y_i + y_{i+v} + y_{i+u}) \mod 2)$
12: **Fin pour**
13: **Si** $|n - 2z| > T$ **Alors**
14: $Tri = Tri \cup \{x^u + x^{u-v} + 1\}$
15: **Pour tout** $P, P' \in Tri$ **Faire**
16: $Q = PGCD(P, P')$
17: **Si** $Q \neq 1$ **Alors**
18: **Retourner** (Q, Tri)
19: **Fin si**
20: **Fin pour**
21: **Fin si**
22: $z = 0$
23: **Fin pour**
24: $u = u + 1$
25: **Fin tant que**
26: **Fin si**
27: **Retourner** Tri

1.2.1.2 Implémentation et résultats

Dans cet algorithme, on cherche à calculer tous les polynômes multiples de P de poids 3 et de degré $< N$. Or le nombre moyen m de multiples de P qui sont de poids 3 et de degré plus petit que D peut être approximé [6] asymptotiquement par :

$$m = \frac{D^2}{2^{L+1}} \tag{1.1}$$

Pour un polynôme de rétroaction de degré L, notre algorithme trouvera un multiple de degré inférieur ou égal à D dès que $m = 2$. La longueur minimale de la suite de sortie nécessaire est donc :

$$\begin{aligned}
N &= \left(\frac{T}{a}\right)^2 + u \\
N &\leqslant u + \frac{\left(a + b\sqrt{13(1-64\varepsilon^6)}\right)^2}{64\varepsilon^6} \\
&\leqslant u + \frac{a^2 + 2ab\sqrt{13(1-64\varepsilon^6)} + b^2 13(1-64\varepsilon^6)}{64\varepsilon^6} \\
&\lesssim u - b\sqrt{13}(a + b\sqrt{13}) + \frac{(a+b\sqrt{13})^2}{64\varepsilon^6} \\
&\lesssim 2^{\frac{L+2}{2}} + \frac{(a+b\sqrt{13})^2}{64\varepsilon^6}
\end{aligned}$$

Les paramètres P_f et P_n doivent donc être choisis tels que le nombre de bits de sortie soit supérieur à N. Le nombre d'opérations effectués par cet algorithme est :

$$W_P \lesssim 2^{L+1} \frac{\left(a + b\sqrt{13}\right)^2}{64\varepsilon^6}$$

J'ai implémenté cet algorithme en langage C puis je l'ai testé pour différents polynômes. On constate dans les tests que, pour avoir un taux de succès de plus de 90%, P_f doit être dans $[10^{-7}, 10^{-6}]$. Le tableau 1.4 présente quelques unes des simulations sur un Pentium 4 à 2.8 GHz avec comme paramètres : $\varepsilon = 0.1$, $P_f = 2.10^{-7}$ et $P_n = 0.5$.

polynôme de rétroaction	polynôme détecté	temps de calcul
$X^8 + X^4 + X^3 + X^2 + 1$	$X^8 + X^4 + X^3 + X^2 + 1$	3 s
$X^{10} + X^8 + X^7 + X^6 + X^5 + X^2 + 1$	$X^{10} + X^8 + X^7 + X^6 + X^5 + X^2 + 1$	13 s
$X^{13} + X^7 + X^6 + X^5 + X^4 + X + 1$	$X^{13} + X^7 + X^6 + X^5 + X^4 + X + 1$	1 min 45 s
$X^{15} + X^{13} + X^6 + X^5 + X^3 + X + 1$	$X^{15} + X^{13} + X^6 + X^5 + X^3 + X + 1$	6 min 56 s
$X^{17} + X^6 + X^4 + X^2 + 1$	$X^{17} + X^6 + X^4 + X^2 + 1$	27 min 47 s
$X^{21} + X^{14} + X^7 + X^2 + 1$	$X^{21} + X^{14} + X^7 + X^2 + 1$	7h 23 min 44 s

FIG. 1.4 – Performances de l'algorithme

Ces performances correspondent bien aux résultats théoriques car si l'on incrémente le degré du polynôme de rétroaction, le temps de calcul est bien multiplié par 2.

1.2.2 Décodage de la suite de bits reçue

Après avoir retrouvé le polynôme de rétroaction de la suite, il nous reste à retrouver l'état initial du registre utilisé. Le principe est de modéliser le problème par un problème de correction d'erreur. En effet, on identifie la sortie du brasseur à la sortie d'un LFSR qui serait passé dans un canal binaire symétrique bruité alors que les erreurs proviennent en réalité du message. Le schéma 1.5 permet de visualiser le problème.

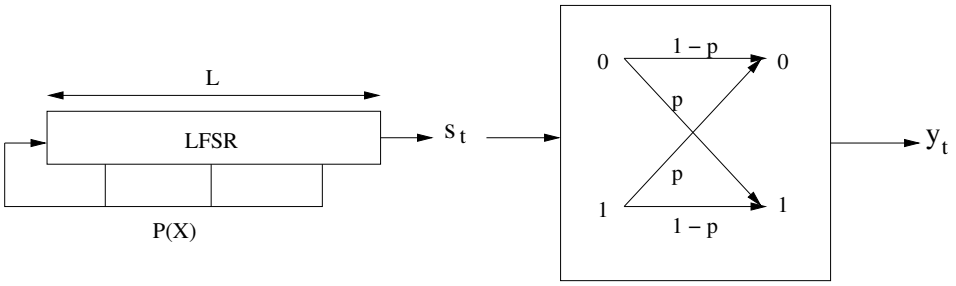

FIG. 1.5 – Modèle du problème de reconstruction de l'état initial d'un brasseur synchrone

La suite $(s_t)_{t\geqslant 0}$ vérifie l'équation de récurrence linéaire définie par le polynôme de rétroaction P. $(s_0, \ldots, s_{N-1})$ est donc un mot du code de longueur N et de dimension L que nous allons décoder. On va utiliser l'algorithme de décodage des codes LDPC dû à Gallager [7].

L'attaque que l'on va mener se partage en deux parties :
- Une phase qui consiste à déterminer des équations de parité de poids 3 pour la suite $(s_t)_{t\geqslant 0}$. Mathieu Cluzeau a choisi 3 car cette valeur conduit à l'algorithme le plus performant compte tenu du nombre de bits nécessaires pour reconstruire le polynôme de rétroaction du brasseur.
- La phase de décodage qui permet de retrouver l'état initial du brasseur

1.2.2.1 Détermination des équations de parité

On va chercher des équations de la forme :

$$s_t + s_{t-i} + s_{t-j} = 0 \ \forall t$$

De telles équations correspondent exactement aux trinômes multiples de P de la forme $1 + X^i + X^j$. On va donc rechercher ces trinômes et pour cela utiliser l'algorithme 1.2.2.

Dans cet algorithme, une donnée est inconnue, la valeur de D le degré maximal des trinômes à calculer. Cependant, il est fort possible que l'on sache évaluer m le nombre d'équations de parité qui seront nécessaires pour décoder la suite. On peut ainsi utiliser l'équation 1.1 qui est une bonne approximation asymptotique de m en fonction de D. A partir de ces équations de parité, on va pouvoir utiliser l'algorithme de décodage des codes LDPC.

Algorithme 1.2.2 Algorithme de recherche des trinômes multiples de P

Entrées: P le polynôme de rétroaction, D le degré maximal des trinômes à calculer
Sorties: les trinômes multiples de P de degré plus petit que D

1: Calculer $q_i(X) = X^i \mod P,\ 1 \leqslant i < D$ et les stocker dans un tableau T défini par : $\forall\ 0 \leqslant a < 2^L,\ T[a] = \{i,\ q_i(X) = a\}$
2: **Pour** $i = 1$ à D **Faire**
3: $A = 1 + q_i(X)$
4: **Pour** $j \in T[A]$ **Faire**
5: $1 + X^i + X^j$ est un multiple de $P(X)$ ayant la forme souhaitée
6: **Fin pour**
7: **Fin pour**

1.2.2.2 Décodage des codes LDPC et algorithme de Gallager

1.2.2.2.1 Codes LDPC

Les codes LDPC (Low Density Parity Check codes) ont été inventés par Robert Gallager lors de sa thèse en 1962 [7]. Comme leur nom l'indique, ce sont des codes de parité qui se caractérisent par une matrice de parité très creuse, ce qui permet un décodage efficace. Comme tous les codes, on peut les représenter par un graphe bi-partite comme dans le schéma 1.6. Comme la matrice est creuse, sa densité est faible, par conséquent, on a un graphe avec peu d'arêtes.

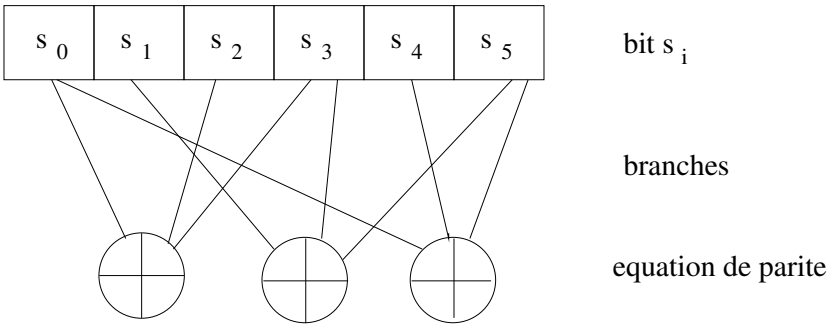

Fig. 1.6 – Exemple de codes LDPC

Dans cet exemple, on a la matrice de parité suivante :

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

On voit que le nombre de colonnes représente le nombre de bits et le nombre de lignes le nombre d'équations de parité. De plus, le nombre de 1 sur la ligne i est le nombre de bits dans l'équation de parité i et le nombre de 1 dans la colonne j est le nombre d'équations de parité associées au bit j. Ainsi :

$$\begin{aligned} (s_0, s_1, \ldots, s_5) \text{ mot du code} &\Leftrightarrow \text{ toutes les équations de parité sont respectées} \\ &\Leftrightarrow H.\mathbf{s} = 0 \end{aligned}$$

Un code LDPC est dit régulier et on le note (i, j) si :
- le nombre de 1 dans chaque colonne de H est i
- le nombre de 1 dans chaque ligne de H est j.

1.2.2.2.2 Graphe de Tanner et cycles

Le graphe de l'exemple ci-dessus est appelé graphe de Tanner. On appelle chemin d'un graphe, une séquence de nœuds et d'arêtes, qui commence et finit par des nœuds tel que chaque arête est incidente avec les nœuds qui la précèdent et la suivent et aucun nœud n'apparaît plus d'une fois. La longueur du chemin est donnée par le nombre d'arêtes composant le chemin. Un cycle est un chemin qui commence et finit par le même nœud. Afin de décoder efficacement un code LDPC avec le décodage itératif, il est important qu'il n'y ait aucun cycle de longueur 4 (et plus généralement de cycle court) dans le graphe de Tanner associé au code.

1.2.2.2.3 Principe de l'algorithme de Gallager

Robert Gallager a introduit dans [7], un algorithme de décodage des codes LDPC. Il s'agit d'un algorithme itératif et probabiliste dans lequel on associe à chaque symbole la probabilité qu'il vaille 1. A chaque itération de l'algorithme, on met à jour ces probabilités en utilisant les équations de parité. Nous allons voir l'algorithme dans le cas d'une transmission sur un canal binaire symétrique de probabilité d'erreur τ car c'est celui qui va être utilisé dans la reconstruction de l'état initial du brasseur comme on le voit sur le schéma 1.5.
L'information dont on dispose à la réception d'un mot est celle qu'on observe sur ce mot reçu. Donc si on reçoit un bit y_t égal à 1, alors $P[s_t = 1] = 1 - \tau$, alors que si y_t égal à 0, on a $P[s_t = 1] = \tau$.

On va utiliser des équations de parité de poids 3 sur les bits du mot de code, donc on aura :

$$(E) \ : \ s_t + s_v + s_u = 0$$

Cette équation permettra de calculer une nouvelle probabilité pour le bit s_t en utilisant les autres bits s_u et s_v. Donc, pour mettre à jour la probabilité pour un certain bit s_t, on doit combiner les différentes informations dont on dispose c'est-à-dire les informations provenant de toutes les équations de parité impliquant s_t ainsi que l'information provenant de l'observation.

1.2.2.2.4 Algorithme proposé par A.Canteaut et M.Trabbia

Dans [6], un algorithme de décodage des codes LDPC utilisant les log-vraisemblances des probabilités est proposé. La log-vraisemblance d'une variable aléatoire X notée $g(X)$ est définie par :

$$g(X) = \log\left(\frac{P[X=0]}{P[X=1]}\right)$$

Le signe de $g(X)$ correspond à une décision sur X, et la valeur absolue $g(X)$ nous donne une idée de la fiabilité de la décision. On notera toujours dans la suite, $(y_t)_{t<N}$ le mot de code reçu et $(s_t)_{t<N}$ la suite issue du LFSR, donc la suite émise dans notre modèle du schéma 1.5. Avant toute itération de l'algorithme, tous les bits ont la même fiabilité, on a donc :

$$g(s_t) = (-1)^{y_t} \log\left(\frac{1-\tau}{\tau}\right)$$

Puis, pour chaque équation de parité, on peut mettre à jour $g(s_t)$ de la façon suivante :

$$g(s_t) = g(s_t) + g(s_u + s_v)$$

avec $g(s_u + s_v)$ qui peut être approximé par

$$g(s_u + s_v) = (-1)^{s_u+s_v} \min\left(|g(s_u)|, |g(s_v)|\right)$$

Chaque itération de l'algorithme de décodage prend en entrée l'estimation de la suite $(s_t)_{t<N}$ obtenue à l'itération précédente et la fiabilité de chacun de ses bits donnée par un tableau contenant les $|g(s_t)|$. Après avoir utilisé les équations de parité, on en déduit la nouvelle valeur des $g(s_t)$, dont le signe fournit une nouvelle estimation de s_t et dont la valeur absolue correspond à la fiabilité de cette valeur. On a donc l'algorithme 1.2.3.

Algorithme 1.2.3 Algorithme de reconstruction de l'état initial proposé par A.Canteaut et M.Trabbia

Entrées: $(y_0, \ldots, y_{N-1})$ la suite reçue
Sorties: $(s_0, \ldots, s_{N-1})$ la suite décodée

1: **Pour** $t = 0$ à $N-1$ **Faire**
2: $s_t = y_t$
3: $g[t] = \log\left(\frac{1-\tau}{\tau}\right)$
4: **Fin pour**
5: **Tant que** il n'y a pas convergence **Faire**
6: **Pour** $t = 0$ à $N-1$ **Faire**
7: $g'[t] = (-1)^{s_t} g[t]$
8: **Pour** chaque équation de parité contenant s_t, $s_t = s_u + s_v$ **Faire**
9: $g'[t] = g'[t] + (-1)^{s_u+s_v} \min(g[u], g[v])$
10: **Fin pour**
11: **Fin pour**
12: **Pour** $t = 0$ à $N-1$ **Faire**
13: **Si** $g'[t] > 0$ **Alors**
14: $s_t = 0$
15: **Sinon**
16: $s_t = 1$
17: **Fin si**
18: $g[t] = |g'[t]|$
19: **Fin pour**
20: **Fin tant que**
21: **Retourner** s_t, $\forall t$

Chaque trinôme $1 + X^u + X^v$ avec $u < v$ multiple de $P(X)$ fournit les équations de parité suivantes :

$$\begin{aligned} s_t &= s_{t-u} + s_{t-v} \quad \text{si } n - v \geqslant 0 \\ s_t &= s_{t+u} + s_{t+u-v} \quad \text{si } n + u < N \text{ et } n - v + u \geqslant 0 \\ s_t &= s_{t+v} + s_{t+v-u} \quad \text{si } n + v < N \end{aligned}$$

L'algorithme converge quand la suite $(s_t)_{t\geqslant 0}$ n'est plus modifiée par de nouvelles itérations. Dans l'algorithme classique présenté dans [6], chaque trinôme multiple de P donne une seule équation de parité (c'est-à-dire qu'à l'étape 8 de l'algorithme, dès qu'on trouve une équation de parité contenant s_t, on arrête la boucle) et le nombre de ces trinômes est entièrement déterminé par la valeur de N puisqu'il correspond au nombre de trinômes de degré au plus N, multiples du polynôme de rétroaction.

J'ai implémenté un autre algorithme, optimisation de ce dernier, avec le fait que la boucle de l'étape 8 peut être exécutée au maximum 3 fois et qu'ainsi, un trinôme peut donner jusqu'à 3 équations de parité. Une autre optimisation est le fait que le nombre de trinômes m n'est pas déterminé par N mais par une fonction de ε que l'on verra par la suite.

1.2.2.2.5 Algorithme proposé par Mathieu Cluzeau

Dans [1], un algorithme différent est proposé. La principale différence est la remise à jour de la table des valeurs absolues des vraisemblances des probabilités. Cet algorithme est détaillé dans 1.2.4 avec $Obs(y_t)$ la vraisemblance de la probabilité résultant de l'observation du bit y_t, $APP^{(i)}(y_t)$ le tableau des vraisemblances des probabilités de y_t à l'itération i et $Eq(i)$ l'équation de parité dans laquelle le bit y_i apparaît.

Algorithme 1.2.4 Algorithme de recherche de l'état initial du brasseur

Entrées: La suite reçue $(y_0, \ldots, y_{N-1})$
Sorties: l'état initial de cette suite décodée

Pour $t = 0$ à $N - 1$ **Faire**
  Si $y_t = 1$ **Alors**
    $Obs(y_t) = \log\left(\frac{1-\tau}{\tau}\right)$
  Sinon
    $Obs(y_t) = \log\left(\frac{\tau}{1-\tau}\right)$
  Fin si
  $APP^{(0)}(y_t) = Obs(y_t)$
Fin pour
$j = 1$
Tant que il n'y a pas convergence **Faire**
  Pour $t = 0$ à $N - 1$ **Faire**

$$APP^{(j)}(y_t) = Obs(y_t) + \sum_i -Sign\left(\prod_{k \in I_i \setminus \{t\}} APP^{(j-1)}(y_k)\right) \times \min_{k \in I_i \setminus \{t\}} \left|APP^{(j-1)}(y_k)\right|$$

    avec $I_i = \{k \mid y_k \in Eq(i)\}$
  Fin pour
  $j = j + 1$
Fin tant que
Pour $t = 0$ à L **Faire**
  Si $APP(y_t) > 0$ **Alors**
    $y_t = 1$
  Sinon
    $y_t = 0$
  Fin si
Fin pour
Retourner $(y_0, \ldots, y_L)$

De même que pour l'algorithme 1.2.3, il y a convergence de la suite lorsqu'elle n'est plus modifiée par de nouvelles itérations. Pour tous ces algorithmes, le nombre d'itérations moyen est 10.

1.2.2.3 Implémentations et résultats

J'ai implémenté en langage C les 3 algorithmes, celui d'A.Canteaut et de M.Trabbia, une version optimisée de ce dernier et l'algorithme de M.Cluzeau. On va maintenant voir la valeur des différents paramètres intervenant dans ces algorithmes. Comme il a été dit précédemment, dans le premier algorithme, le nombre de trinômes multiples de P construisant les équations de parité, est entièrement déterminé par le nombre de bits N de la suite. Dans [6], on trouve que la valeur moyenne m du nombre de trinômes nécessaire pour la convergence est de :

$$m \geqslant \frac{2}{C(\tau)}$$

avec $C(\tau)$ la capacité du canal binaire symétrique de probabilité d'erreur τ, c'est-à-dire :

$$C(\tau) = 1 + \tau \log_2(\tau) + (1-\tau)\log_2(1-\tau)$$

De plus, si $\varepsilon = \tau - \frac{1}{2}$ est petit, on peut approximer [1] $C(\tau)$ par :

$$C(\tau) \simeq \frac{2\varepsilon^2}{\ln(2)}$$

On en déduit :

$$m \gtrsim \frac{\ln(2)}{\varepsilon^2}$$

Et avec l'équation 1.1 et $D = N$, on a :

$$N \simeq \frac{\sqrt{\ln(2)}}{\varepsilon} 2^{\frac{L+1}{2}}$$

Nous rappelons que cette méthode implique qu'un trinôme multiple de P ne donne qu'une seule équation de parité.

Si on choisit d'avoir environ 3 équations par trinômes, comme dans la deuxième méthode, optimisation de la première, on peut donc raisonnablement penser qu'il nous faudra 3 fois moins de trinômes. Ce qui entraîne :

$$m \geqslant \frac{2}{3C(\tau)}$$

Donc, avec l'approximation de $C(\tau)$ on a :

$$m \gtrsim \frac{\ln(2)}{3\varepsilon^2}$$

D'où d'après l'équation 1.1 :

$$D \simeq \frac{\sqrt{\ln(2)}}{\sqrt{3}\varepsilon} 2^{\frac{L+1}{2}}$$

On pourrait également choisir $D = N$ et ainsi avoir un nombre de bits minimal.

Dans la troisième méthode, celle de M.Cluzeau, il est écrit dans [1] que :

$$m \geqslant \frac{1}{3C(\tau)}$$

[1] L'explication de cette approximation est donnée dans [1]

On constate dans les simulations que cette formule semble exacte si on a un nombre de bits bien supérieur à celui donné dans [1]. Ce nombre de bits est de l'ordre de

$$N \simeq \frac{\sqrt{\ln(2)}}{\varepsilon} 2^{\frac{L+1}{2}}$$

Cependant, on constate en pratique, que le nombre de trinômes nécessaire et le nombre de bits minimal (avec $D = N$) sont les mêmes que dans la deuxième méthode c'est-à-dire :

$$m \gtrsim \frac{\ln(2)}{3\varepsilon^2}$$

et

$$N \simeq \frac{\sqrt{\ln(2)}}{\sqrt{3}\varepsilon} 2^{\frac{L+1}{2}}$$

Cependant si on a un nombre de bits de l'ordre de $\frac{\sqrt{\ln(2)}}{\varepsilon} 2^{\frac{L+1}{2}}$, seulement $\frac{m}{2}$ trinômes suffisent.

Nous allons maintenant voir la complexité de ces algorithmes. Tout d'abord le nombre d'opérations de la phase de précalcul est de D. Donc :

- si $m \geqslant \frac{2}{C(\tau)}$ alors le nombre d'opérations est de $\frac{\sqrt{\ln(2)}}{\varepsilon} 2^{\frac{L+1}{2}}$
- si $m \geqslant \frac{2}{3C(\tau)}$ alors le nombre d'opérations est de $\frac{\sqrt{\ln(2)}}{\sqrt{3}\varepsilon} 2^{\frac{L+1}{2}}$
- si $m \geqslant \frac{1}{3C(\tau)}$ alors le nombre d'opérations est de $\frac{\sqrt{\ln(2)}}{\sqrt{6}\varepsilon} 2^{\frac{L+1}{2}}$

Dans le tableau 1.7, on trouve les valeurs de m pour les biais de 0.2, 0.1 et 0.05.

biais	0.2			0.1			0.05		
$m \geqslant$	$\frac{2}{C(\tau)}$	$\frac{2}{3C(\tau)}$	$\frac{1}{3C(\tau)}$	$\frac{2}{C(\tau)}$	$\frac{2}{3C(\tau)}$	$\frac{1}{3C(\tau)}$	$\frac{2}{C(\tau)}$	$\frac{2}{3C(\tau)}$	$\frac{1}{3C(\tau)}$
m	17	6	3	70	24	12	277	92	47

FIG. 1.7 – Valeurs de m en fonction du biais

En ce qui concerne le décodage, le nombre moyen d'opérations du premier algorithme est donné dans [6] :

$$10 \left(\frac{\ln(2)}{\varepsilon}\right)^3 2^{\frac{L+1}{2}}$$

On peut ainsi en déduire le nombre d'opérations de l'optimisation de cet algorithme et de l'algorithme de M.Cluzeau avec un nombre de bits minimal :

$$10 \left(\frac{\ln(2)}{\sqrt{3}\varepsilon}\right)^3 2^{\frac{L+1}{2}}$$

Pour l'algorithme de M.Cluzeau, si on considère un nombre de bits $N = \frac{\sqrt{\ln(2)}}{\varepsilon} 2^{\frac{L+1}{2}}$, on a un nombre d'opérations de :

$$\frac{5}{3} \left(\frac{\ln(2)}{\varepsilon}\right)^3 2^{\frac{L+1}{2}}$$

Le nombre d'opérations de l'algorithme de Cluzeau avec un nombre de trinômes deux fois plus petit est légèrement inférieur au nombre d'opérations de l'algorithme de Canteaut-Trabbia optimisé. Il est important de noter que toutes ces valeurs sont asymptotiques, et que, pour le décodage des codes LDPC, il existe peu de résultats théoriques, alors que l'on constate une bonne efficacité de décodage en pratique.

Nous avons effectué quelques tests avec le polynôme de rétroaction $X^{21} + X^{14} + X^{12} + X^{11} + X^{10} + X^5 + X^3 + X^2 + 1$. Les résultats se trouvent dans le tableau 1.8 avec 0 qui représente la méthode de Canteaut-Trabbia, 1 cette dernière optimisée et 2 celle de Cluzeau.

méthode	biais	N	m	% de réussite	temps de calcul
0	0.2	8526	19	100	2 s
1		5000	6	100	2 s
2		5000	6	100	1 s
0	0.1	17051	79	100	9 s
1		9845	24	99	2 s
2		9845	24	90	2 s
0	0.05	34102	296	100	59 s
1		19689	92	100	13 s
2		19689	92	100	15 s

FIG. 1.8 – Tests de décodage avec $X^{21} + X^{14} + X^{12} + X^{11} + X^{10} + X^5 + X^3 + X^2 + 1$

On constate que le pourcentage est proche de 100% avec les valeurs de N calculées à partir des formules. On ne constate pas de différences entre la méthode de Canteaut-Trabbia optimisée et celle de M.Cluzeau dans les pourcentages de réussite et dans les temps de calcul. Selon M.Cluzeau, la différence se situe au niveau des valeurs des vraisemblances des bits. En effet, lorsque la méthode de Canteaut-Trabbia ne converge pas, les valeurs des vraisemblances se comportent de manière assez aléatoire, alors que, dans la méthode de Cluzeau, on pourrait tirer de l'information de la valeur des vraisemblances lorsque l'algorithme ne converge pas. Pour résumer, la méthode de reconstruction des paramètres

ε	L	N_I	N_P
0.05	10	435	87 874 292
	20	13 921	87 876 276
	30	445 500	87 939 764
	40	14 255 978	89 971 380
0.1	10	217	1 373 098
	20	6960	1 375 083
	30	222 750	1 438 571
	40	7 127 988	3 470 186
0.2	10	109	21 517
	20	3 480	23 501
	30	111 374	86 989
	40	3 563 994	2 118 605

FIG. 1.9 – Nombre de bits nécessaires pour reconstruire l'état initial (N_I) et le polynôme (N_P) en fonction de L et de ε

du brasseur de M.Cluzeau nécessite un grand nombre de bits pour retrouver le polynôme de rétroaction comme on le voit dans le tableau 1.9. On peut se demander s'il n'est pas possible de retrouver ces paramètres avec un nombre de bits inférieur. De plus, la complexité dépend d'un facteur $\frac{1}{\varepsilon^6}$ ce qui fait que les temps de calcul grandissent très vite quand le biais se rapproche de zéro.

1.3 Recherche globale des paramètres du brasseur

On a vu dans l'état de l'art que la reconstruction du brasseur synchrone était effectuée en deux étapes, d'abord celle de recherche du polynôme de rétroaction du brasseur, puis le décodage de la suite reçue pour retrouver l'état initial du brasseur. La première étape nécessite un nombre de bits de sortie du brasseur plus important que la seconde. L'idée serait donc de combiner ces deux étapes en une seule et ainsi de tenter d'avoir un nombre de bits nécessaires moins important. On a ainsi imaginé un autre algorithme basé sur la recherche exhaustive qui est succinctement décrit dans l'algorithme 1.3.1. Dans cet algo-

Algorithme 1.3.1 Algorithme de recherche exhaustive de l'état initial et du polynôme de rétroaction du brasseur

Entrées: la suite reçue $(y_0, \ldots, y_{N-1})$
Sorties: l'état initial et le polynôme de rétroaction de cette suite

Précalcul : Rechercher tous les polynômes candidats à être polynôme de rétroaction du LFSR
Pour tous ces polynômes P **Faire**
  Pour tous les états initiaux i possibles et non nuls **Faire**
    Générer une suite de longueur N notée $(s_t)_{t<N}$ à partir du LFSR engendré par i et P
    Pour $t = 0$ à $N - 1$ **Faire**
      $w_t = y_t \oplus s_t$
    Fin pour
    Tester la suite $(w_t)_{t<N}$
    Si $(w_t)_{t<N}$ passe le test **Alors**
      Retourner le polynôme de rétroaction et l'état initial de $(s_t)_{t<N}$
    Fin si
  Fin pour
Fin pour

rithme, on doit précalculer tous les polynômes candidats à être polynôme de rétroaction du brasseur ce que l'on va voir dans la partie suivante.

1.3.1 Calcul des polynômes candidats

On a vu qu'en pratique, le LFSR composant le brasseur avait comme paramètre un polynôme primitif afin d'avoir la plus grande période possible. Il est cependant possible de n'avoir qu'un polynôme irréductible comme polynôme de rétroaction du registre. Nous allons donc voir dans un premier temps le calcul de tous les polynômes primitifs de degré donné, puis dans un second temps, le calcul de tous les polynômes irréductibles.

1.3.1.1 Calcul de tous les polynômes primitifs

Pour calculer tous les polynômes primitifs d'un certain degré m, on a testé trois algorithmes. Le premier et le plus naturel est de tester l'irréductibilité de tous les polynômes de degré m et ensuite de tester la primitivité de ces polynômes irréductibles. Le deuxième est basé sur la factorisation du $(2^m - 1)$-ème polynôme cyclotomique car il est le produit de tous les polynômes primitifs de degré m. Enfin, le troisième calcule simplement tous les polynômes minimaux des éléments primitifs du corps F_{2^m} construit à l'aide d'un polynôme primitif trouvé aléatoirement et dont on a testé l'irréductibilité et la primitivité. Nous allons voir ces trois algorithmes en détails.

1.3.1.1.1 Test d'irréductibilité et de primitivité de tous les polynômes

Test d'irréductibilité

Définition 6. Un polynôme $P \in \mathbb{F}_2[X],\ P \neq 0$ est dit irréductible si ses seuls diviseurs sont le polynôme $Q = 1$ et lui-même.

Pour tester le caractère irréductible d'un polynôme, nous allons utiliser l'algorithme 1.3.2 de Ben-Or [4].

Algorithme 1.3.2 Algorithme de Ben-Or

Entrées: Un polynôme P de degré m
Sorties: faux, s'il n'est pas irréductible et vrai sinon

1: **Pour** $i = 1$ à $\lfloor \frac{m}{2} \rfloor$ **Faire**
2: $\quad g_i = PGCD\left(x^{2^i} - x, P\right)$
3: $\quad$ **Si** $g_i \neq 1$ **Alors**
4: $\quad\quad$ **Retourner** faux
5: $\quad$ **Fin si**
6: **Fin pour**
7: **Retourner** vrai

Cet algorithme est basé sur le théorème suivant.

Théorème 4. Un polynôme $P \in \mathbb{F}_2[X]$ de degré $m \geqslant 1$ est irréductible si, et seulement si

1. P divise $x^{2^m} - x$
2. $PGCD\left(x^{2^{\frac{m}{t}}} - x, P\right) = 1$ pour tous les diviseurs premiers t de m.

La démonstration de ce théorème est donnée page 383 de [4]. De plus, un polynôme réductible possède un facteur irréductible de degré au moins $\frac{m}{2}$ qui divise un des $PGCD\left(x^{2^i} - x, P\right)$ car le polynôme $x^{2^d} - x \in \mathbb{F}_2[X],\ \forall d \geqslant 1$, est le produit de tous les polynômes irréductibles de $\mathbb{F}_2[X]$ dont le degré divise d.

Test de primitivité

Pour définir la notion de primitivité d'un polynôme, nous avons besoin de quelques définitions et théorèmes préalables.

Définition 7. Soit $\alpha \in \mathbb{F}_{2^m}^*$, l'ordre de l'élément α, noté $o(\alpha)$ est le plus petit entier $n > 0$ tel que $\alpha^n = 1$.

Théorème 5. Soit $\mathbb{F}_{2^m}$ le corps à 2^m éléments et $\alpha \in \mathbb{F}_{2^m}^*$ alors $o(\alpha)$ divise $2^m - 1$.

Définition 8. Un élément $\alpha \in \mathbb{F}_{2^m}^*$ est dit primitif si $o(\alpha) = 2^m - 1$.

Définition 9. Soit $\alpha \in \mathbb{F}_{2^m}$, le polynôme minimal P_α associé à α est le polynôme de $\mathbb{F}_{2^m}$ de plus faible degré possible dont α est une racine. Si α est primitif alors son polynôme minimal P_α est dit aussi primitif.

Donc pour tester si un polynôme P est primitif, il suffit de calculer l'ordre de $X \mod P(X))$ et voir s'il est égal à $2^m - 1$. On peut aussi tester que $(X \mod P(X))^i \neq 1$, $\forall i \leqslant d$ avec d le plus grand diviseur de $2^m - 1$. On a ainsi l'algorithme 1.3.3.

Algorithme 1.3.3 Test de la primitivité d'un polynôme de $\mathbb{F}_{2^m}$

Entrées: P un polynôme de degré m
Sorties: faux s'il n'est pas primitif et vrai sinon

Calculer d le plus grand diviseur de $2^m - 1$
Pour $i = m$ à d **Faire**
  Si $X^i = 1 \mod P$ **Alors**
    Retourner faux
  Fin si
Fin pour
Retourner vrai

Par conséquent, pour trouver tous les polynômes primitifs de degré m, on va combiner ces deux algorithmes ce qui nous donne l'algorithme 1.3.4

Algorithme 1.3.4 Algorithme de test de l'irréductibilité et de la primitivité de tous les polynômes de degré m

Entrées: m le degré du polynôme souhaité
Sorties: Tous les polynômes primitifs de degré m

Calculer d, le plus grand diviseur de $2^m - 1$
Pour tous les polynômes P de degré m **Faire**
  Si poids de $P = 0 \mod 2$ **Ou** $P(0) = 0$ **Alors**
    Passer au polynôme suivant
  Fin si
  Pour i de 1 à $\dfrac{m}{2}$ **Faire**
    $g = PGCD(P, x^{2^i} - x \mod P)$
    Si $g \neq 1$ **Alors**
      Passer au polynôme suivant
    Fin si
  Fin pour
  Pour i de m à d **Faire**
    Si $X^i = 1 \mod P$ **Alors**
      Passer au polynôme suivant
    Fin si
  Fin pour
  Ajouter P à la liste des polynômes primitifs de degré m
Fin pour

1.3.1.1.2 Factorisation du polynôme cyclotomique

La deuxième méthode est basée sur la factorisation du polynôme cyclotomique qui est défini comme suit :

Définition 10. Soit le corps fini $\mathbb{F}_2$, d un nombre positif impair, et α une racine primitive d-ème de l'unité sur $\mathbb{F}_2$ (c'est-à-dire tel que α est d'ordre d). Alors le polynôme

$$Q^{(d)}(X) = \prod_{\substack{s=1 \\ PGCD(s,d)=1}}^{d} (X - \alpha^s)$$

est appelé le d-ème polynôme cyclotomique sur $\mathbb{F}_2$.

On constate que le $(2^m - 1)$-ème polynôme cyclotomique est bien le produit de tous les polynômes primitifs de degré m.

Définition 11. L'indicatrice d'Euler notée φ est une fonction définie comme suit :

$$\begin{array}{rcl} \varphi : \mathbb{N}^* & \to & \mathbb{N}^* \\ n & \mapsto & Card\{m \in \mathbb{N}^* \mid m \leqslant n,\ PGCD(m,n) = 1\} \end{array}$$

On peut la calculer avec la formule suivante :

$$\varphi(n) = \prod_{i=1}^{m} p_i^{e_i - 1}(p_i - 1)$$

avec $n = \prod_{i=1}^{m} p_i^{e_i}$ où p_i est premier et $e_i \in \mathbb{N}^*$.

Définition 12. Avec la décomposition de n précédente, on peut définir la fonction de Möbius $\mu(n)$ par :

$$\mu(n) = \begin{cases} 1 & \text{si } n = 1 \\ 0 & \text{s'il existe } i \text{ tel que } e_i \geqslant 2 \\ (-1)^m & \text{si } e_1 = e_2 = \ldots = e_m = 1 \end{cases}$$

De plus, on a les formules suivantes appelées formules d'inversion de Möbius, si $G(n) = \sum_{d|n} F(d)$ alors

$$F(n) = \sum_{d|n} \mu(d) G\left(\frac{n}{d}\right) = \sum_{d|n} \mu\left(\frac{n}{d}\right) G(d)$$

et si $G(n) = \prod_{d|n} F(d)$

$$F(n) = \prod_{d|n} G\left(\frac{n}{d}\right)^{\mu(d)} = \prod_{d|n} G(d)^{\mu\left(\frac{n}{d}\right)}$$

Théorème 6. Soit $\alpha \in \mathbb{F}_{2^m}^*$ alors si $d|(2^m - 1)$, il existe $\varphi(d)$ éléments d'ordre d où φ est l'indicatrice d'Euler.

Démonstration. Nous savons que $\mathbb{F}_{2^m}^*$ est un groupe cyclique fini. Pour démontrer ce théorème, nous allons avoir besoin du théorème suivant :

Théorème 7. Dans $\mathbb{F}_{2^m}^*$, l'élément α^i est d'ordre $\frac{2^m-1}{PGCD(2^m-1,i)}$.

Démonstration. Soit $k = PGCD(i, 2^m - 1)$. L'ordre de α^i est le plus petit entier n tel que $\alpha^{in} = 1$. Ce qui implique que $(2^m - 1)|in$ et donc que $\frac{2^m-1}{k}|n$ car $\frac{2^m-1}{k}$ ne divise pas $\frac{i}{k}$ (car ils sont premiers entre eux). Le plus petit entier positif n avec cette propriété est $\frac{2^m-1}{k}$. □

Si $d|(2^m - 1)$ alors on a $2^m - 1 = kd$. D'après le théorème précédent, α^i est d'ordre d si et seulement si $PGCD(i, 2^m - 1) = k$. Le nombre d'éléments d'ordre d est égal au nombre d'entiers i, avec $1 \leqslant i \leqslant 2^m - 1$ et $PGCD(i, 2^m - 1) = k$. On écrit $i = kh$ avec $1 \leqslant h \leqslant d$ et ainsi $PGCD(i, 2^m - 1) = d \Leftrightarrow PGCD(h, d) = 1$. Le nombre de ces h est égal à $\varphi(d)$. □

Remarque :
$Q^{(d)}(X)$ est de degré $\varphi(d)$ puisque le nombre d'éléments d'ordre d est $\varphi(d)$.

Théorème 8.

$$X^{2^m} - X = \prod_{\alpha \in \mathbb{F}_{2^m}} (X - \alpha)$$

Démonstration. Tout élément $\alpha \in \mathbb{F}_{2^m}$ d'ordre d est un zéro de $X^d - X$ car $d|2^m - 1$ donc $2^m - 1 = dt,\ t \in \mathbb{N}$. On a alors :

$$\alpha^{2^m} = \alpha^{2^m-1} \times \alpha = \alpha \times \left(\alpha^d\right)^t = \alpha$$

D'où :

$$\prod_{\alpha \in \mathbb{F}_{2^m}} (X - \alpha) \mid X^{2^m} - X$$

Comme de plus, les polynômes sont de même degré, ils sont égaux. □

Théorème 9. Le polynôme cyclotomique d'ordre $2^m - 1$ est égal à :

$$Q^{(2^m-1)}(X) = \prod_{d|2^m-1} (X^d - 1)^{\mu\left(\frac{2^m-1}{d}\right)}$$

Démonstration. On a :

$$\begin{aligned} X^{2^m-1} - 1 &= \prod_{\alpha \in \mathbb{F}_{2^m}^*} (X - \alpha) \\ &= \prod_{d|2^m-1} \prod_{\substack{o(\alpha)=d \\ \alpha \in \mathbb{F}_{2^m}^*}} (X - \alpha) \\ &= \prod_{d|2^m-1} Q^{(d)}(X) \end{aligned}$$

Il suffit maintenant d'appliquer une des formules d'inversion de Möbius et on a :

$$Q^{(2^m-1)}(X) = \prod_{d|(2^m-1)} (X^d - 1)^{\mu\left(\frac{2^m-1}{d}\right)}$$

□

On en déduit que le nombre de polynômes primitifs de degré m est de :

$$\frac{\varphi\left(2^m-1\right)}{m}$$

Calculer la fonction de Möbius n'étant pas très pratique, il existe un algorithme permettant de calculer efficacement le polynôme cyclotomique d'ordre 2^m-1. Il repose sur le théorème suivant :

Théorème 10. Soient $m,k \in \mathbb{N}\backslash\{0\}$. Alors :

1. Si m est premier alors $Q^{(m)}(X) = X^{m-1}+X^{m-2}+\ldots+X+1$
2. Si m est impair alors $Q^{(2m)}(X) = Q^{(m)}(-X)$
3. Si k est premier et k ne divise pas m alors $Q^{(km)}(X)Q^{(m)}(X) = Q^{(m)}(X^k)$
4. Si chaque diviseur premier de k divise m alors $Q^{(mk)}(X) = Q^{(m)}(X^k)$

Démonstration. 1 : On a $Q^{(m)}(X) = \frac{X^m-1}{X-1}$ car $\mu(1)=1$ et $\mu(m)=-1$ car m est premier. D'où $Q^{(m)}(X) = X^{m-1}+X^{m-2}+\ldots+X+1$.
2 :

$$\begin{aligned}
Q^{(2m)}(X) &= \prod_{d|2m}(X^d-1)^{\mu\left(\frac{2m}{d}\right)} \\
&= \prod_{d|m}(X^d-1)^{\mu\left(\frac{2m}{d}\right)}(X^{2d}-1)^{\mu\left(\frac{m}{d}\right)} \\
&= \prod_{d|m}(X^d-1)^{\mu(2)\mu\left(\frac{m}{d}\right)}(X^{2d}-1)^{\mu\left(\frac{m}{d}\right)}
\end{aligned}$$

car $\mu(uv)=\mu(u)\mu(v)$ si $PGCD(u,v)=1$ et $PGCD(2,\frac{m}{d})=1$ car $\frac{m}{d}$ impair.

$$\begin{aligned}
Q^{(2m)}(X) &= \prod_{d|m}(X^d-1)^{-\mu\left(\frac{m}{d}\right)}(X^{2d}-1)^{\mu\left(\frac{m}{d}\right)} \text{ car } \mu(2)=-1 \\
&= \prod_{d|m}\left(\frac{X^{2d}-1}{X^d-1}\right)^{\mu\left(\frac{m}{d}\right)} \\
&= \prod_{d|m}\left(\frac{(X^d-1)(X^d+1)}{X^d-1}\right)^{\mu\left(\frac{m}{d}\right)} \\
&= \prod_{d|m}\left(X^d+1\right)^{\mu\left(\frac{m}{d}\right)} \\
&= \prod_{d|m}\left(-(X^d-1)\right)^{\mu\left(\frac{m}{d}\right)} \\
&= \prod_{d|m}\left((-X)^d-1\right)^{\mu\left(\frac{m}{d}\right)} \text{ car } d \text{ est impair puisque } m \text{ est impair} \\
&= Q^{(m)}(-X)
\end{aligned}$$

3 :

$$\begin{aligned} Q^{(km)}(X) &= \prod_{d|km}(X^d-1)^{\mu\left(\frac{km}{d}\right)} \\ &= \prod_{d|m}(X^d-1)^{\mu\left(\frac{km}{d}\right)}(X^{kd}-1)^{\mu\left(\frac{m}{d}\right)} \\ &= \prod_{d|m}(X^d-1)^{\mu(k)\mu\left(\frac{m}{d}\right)}(X^{kd}-1)^{\mu\left(\frac{m}{d}\right)} \end{aligned}$$

car $\mu(uv)=\mu(u)\mu(v)$ si $PGCD(u,v)=1$ et $PGCD(k,\frac{m}{d})=1$ car $PGCD(m,k)=1$

$$\begin{aligned} Q^{(km)}(X) &= \prod_{d|m}(X^d-1)^{-\mu\left(\frac{m}{d}\right)}(X^{kd}-1)^{\mu\left(\frac{m}{d}\right)} \text{ car } \mu(k)=-1 \text{ car } k \text{ premier} \\ &= \prod_{d|m}\left(\frac{X^{kd}-1}{X^d-1}\right)^{\mu\left(\frac{m}{d}\right)} \\ &= \frac{Q^{(m)}(X^k)}{Q^{(m)}(X)} \end{aligned}$$

4 :

$$Q^{(km)}(X) = \prod_{d|km}(X^d-1)^{\mu\left(\frac{km}{d}\right)}$$

Or, pour d ne divisant pas k, on a $\frac{km}{d}$ qui n'est pas sans facteur carré car les diviseurs premiers de k divisent m et ainsi les diviseurs premiers de k apparaissent au moins à la puissance 2 dans km, donc $\forall d \nmid k,\ \mu\left(\frac{km}{d}\right)=0$. Donc on a :

$$\begin{aligned} Q^{(km)}(X) &= \prod_{d'|m}(X^{kd'}-1)^{\mu\left(\frac{m}{d'}\right)} \text{ avec } d=kd' \\ &= Q^{(m)}(X^k) \end{aligned}$$

□

L'algorithme 1.3.5 permet donc de calculer le polynôme cyclotomique d'ordre m.

Cet algorithme calcule bien le polynôme cyclotomique d'ordre m. En effet, notons $n=p_1\dots p_r$. Nous avons que $f_i=Q^{(p_1\cdots p_i)}$ pour $0\leqslant i\leqslant r$. Cela est évident pour $i=0$ et découle ensuite du point 3 du théorème. Nous avons ensuite :

$$f_r\left(x^{\frac{m}{n}}\right)=Q^{(n)}\left(x^{\frac{m}{n}}\right)=Q^{(m)}(x)$$

par le point 4 du théorème car chaque diviseur de $\frac{m}{n}$ divise m.

Le polynôme cyclotomique étant calculé, il faut maintenant le factoriser pour obtenir tous les polynômes primitifs de degré m. Pour cela, on va utiliser l'algorithme de factorisation dû à Berlekamp et décrit dans l'algorithme 1.3.6. On peut utiliser cet algorithme

Algorithme 1.3.5 Calcul du polynôme cyclotomique

Entrées: $m = \sum_{i=1}^{r} p_i^{e_i}$ l'ordre du polynôme cyclotomique
Sorties: Le polynôme cyclotomique d'ordre m

1: $f_0 = x - 1$
2: **Pour** $i = 1$ à r **Faire**
3: $\quad f_i = \dfrac{f_{i-1}(x^{p_i})}{f_{i-1}}$
4: **Fin pour**
5: **Retourner** $f_r\left(x^{\frac{m}{p_1 \cdots p_r}}\right)$

Algorithme 1.3.6 Algorithme de factorisation de Berlekamp

Entrées: $P \in \mathbb{F}_2[X]$ un polynôme unitaire sans facteur carré de degré n
Sorties: N'importe lequel des facteurs g de P ou échec

1: **Pour** $i = 0$ à $n - 1$ **Faire**
2: $\quad$ Calculer $x^{qi} \mod P = \sum_{j=0}^{n-1} q_{ij} x^j$
3: **Fin pour**
4: $Q = (q_{ij})_{0 \leqslant i,j < n}$
5: Calculer une base $(b_1, \ldots, b_r)$ du noyau de la matrice $Q - I \in \mathbb{F}_2^{n \times n}$ avec I la matrice identité de taille $n \times n$
6: Choisir aléatoirement $c_1, \ldots, c_r \in \mathbb{F}_2$
7: $a = c_1 b_1 + \ldots + c_r b_r$
8: $g_1 = PGCD(a, P)$
9: **Si** $g_1 \neq 1$ **Alors**
10: $\quad$ **Retourner** g_1
11: **Fin si**
12: $g_2 = PGCD(a - 1, P)$
13: **Si** $g_2 \neq 1$ et $g_2 \neq P$ **Alors**
14: $\quad$ **Retourner** g_2
15: **Sinon**
16: $\quad$ **Retourner** échec
17: **Fin si**

car la condition que le polynôme à factoriser soit unitaire et sans facteur carré est respectée dans le cas du polynôme cyclotomique. Et pour avoir la factorisation complète du polynôme cyclotomique, il suffit de calculer la base du noyau de $Q - I$ une seule fois et d'appliquer récursivement le processus des lignes 6 à 17 de l'algorithme 1.3.6 à g et $\frac{P}{g}$. On obtient ainsi tous les polynômes primitifs de degré m recherchés.

1.3.1.1.3 Calcul des polynômes minimaux de tous les éléments primitifs

On va maintenant voir la troisième et dernière méthode, celle classique du calcul des polynômes minimaux de tous les éléments primitifs de $\mathbb{F}_{2^m}$. Pour cela, il faut déjà construire le corps $\mathbb{F}_{2^m}$, et donc calculer un premier polynôme primitif. On va tester l'irréductibilité et la primitivité de polynômes de degré m tirés au hasard grâce à l'algorithme 1.3.7.

Ainsi, on va avoir $\mathbb{F}_{2^m} = \dfrac{\mathbb{F}_2[X]}{(P)}$.

Théorème 11. Si P est un polynôme primitif avec comme racine $a = x$ alors tous les

Algorithme 1.3.7 Algorithme de recherche d'un polynôme primitif de degré m

Entrées: m le degré du polynôme souhaité
Sorties: Un polynôme primitif de degré m

1: Calculer d, le plus grand diviseur de $2^m - 1$
2: Tirer aléatoirement P, un polynôme de degré m
3: *Test de Ben-Or :*
4: **Si** poids de $P = 0 \mod 2$ **Alors**
5: Retourner en 2
6: **Fin si**
7: **Si** $P(0) = 0$ **Alors**
8: Retourner en 2
9: **Fin si**
10: **Pour** i de 1 à $\frac{m}{2}$ **Faire**
11: $g = PGCD(P, x^{2^i} - x \mod P)$
12: **Si** $g \neq 1$ **Alors**
13: Retourner en 2
14: **Fin si**
15: **Fin pour**
16: *Test d'un polynôme primitif :*
17: **Pour** i de m à d **Faire**
18: **Si** $X^i = 1 \mod P$ **Alors**
19: Retourner en 2
20: **Fin si**
21: **Fin pour**
22: **Retourner** P

$\frac{\varphi(2^m-1)}{m}$ polynômes primitifs sont donnés par l'ensemble :

$$B = \left\{ f_s(x) = (x - a^s)\left(x - a^{s2}\right) \ldots \left(x - a^{s2^{m-1}}\right) \mid s \leqslant 2^m - 1 \text{ et } PGCD\left(s, 2^m - 1\right) = 1 \right\}$$

Définition 13. On appelle classe cyclotomique binaire modulo m de l'entier i, l'ensemble des entiers modulo m de la forme $i2^j$.

Si on calcule l'ensemble B avec tous les $s \leqslant 2^m - 1$ tel que $PGCD(s, 2^m - 1)$, on va avoir $\varphi\left(2^m - 1\right)$ polynômes donc certains vont être égaux entre eux. En effet, $f_s = f_{s2^i}$, $\forall i \leqslant 2^m - 1$. On doit donc pour calculer seulement $\frac{\varphi(2^m-1)}{m}$ polynômes primitifs, calculer les polynômes minimaux des a^s avec $PGCD(s, 2^m - 1) = 1$ et les s pris dans des classes cyclotomiques différentes. On va donc calculer un représentant de chaque classe cyclotomique ce qui va se faire grâce à l'algorithme 1.3.8.

Nous allons maintenant voir comment calculer le polynôme minimal de a^s avec s tel que $PGCD(s, 2^m - 1) = 1$. Cette méthode de calcul du polynôme minimal a été proposée dans [9].
Notons $g_s(x) = x^m + \sum_{i=0}^{m-1} c_i x^i$ avec $c_i \in \mathbb{F}_2$ le polynôme minimal de a^s qui est donc primitif.
D'après la définition d'un polynôme minimal, on a :

$$g_s(x) = a^{sm} + c_{m-1}a^{s(m-1)} + \ldots + c_0 = 0$$

Algorithme 1.3.8 Calcul des représentants de chaque classe cyclotomique

Entrées: m le degré souhaité

Sorties: le tableau contenant les représentants k de chaque classe cyclotomique tel que $PGCD(k, 2^m - 1) = 1$.

1: Initialiser deux tableaux tab et c de taille $2^m - 1$
2: **Pour** $k = 1$ à $2^m - 1$ **Faire**
3: **Si** $PGCD(k, 2^m - 1) = 1$ **Alors**
4: **Si** $tab[k] = 0$ **Alors**
5: $tab[k] = 1$ et $c[k] = 1$
6: $e = 2k \mod 2^m - 1$
7: **Tant que** $e \neq k$ **Faire**
8: $tab[e] = 1$
9: $e = 2e \mod 2^m - 1$
10: **Fin tant que**
11: **Fin si**
12: **Fin si**
13: **Fin pour**
14: **Retourner** c

En multipliant cette équation par a^{us}, $u = 0, 1, 2 \ldots, m - 1$, on a le système :

$$a^{s(m+u)} = \sum_{t=1}^{m} c_{m-t} a^{s(m+u-t)}$$

Si on ne garde que le cœfficient constant on a :

$$a_0^{(s(m+u))} = \sum_{t=1}^{m} c_{m-t} a_0^{(s(m+u-t))}$$

Si $v_{m+u} = a_0^{(s(m+u))}$, on a $v_{m+u} = \sum_{t=1}^{m} e_t v_{m+u-t}$ avec $e_t = c_{m-t}$.
Cette dernière équation constitue un système de m équations linéaires indépendantes d'inconnues e_t. On peut résoudre ce système en utilisant l'algorithme 1.3.9 de Berlekamp-Massey.

Algorithme 1.3.9 Algorithme de Berlekamp-Massey

Entrées: une suite $s_0, s_1, \ldots, s_n$

Sorties: le polynôme de rétroaction engendrant cette suite

1: $P(X) = 1$, $L = 0$, $k = -1$, $g(X) = 1$
2: **Pour** N de 0 à n **Faire**
3: Calculer $d = s_N + \sum_{i=1}^{L} c_i s_{N-i}$ avec c_i le $i$ème cœfficient de P.
4: **Si** $d = 1$ **Alors**
5: $t(X) = P(X)$, $P(X) = P(X) + g(X)X^{N-k}$
6: **Si** $2L \leqslant N$ **Alors**
7: $L = N + 1 - L$, $k = N$, $g(X) = t(X)$
8: **Fin si**
9: **Fin si**
10: **Fin pour**

Pour arriver à ce système linéaire à résoudre, il faut tout d'abord avoir les cœfficients constants de a^{si} pour des i donnés. Cependant, on va parcourir tous les représentants des classes cyclotomiques, donc il est plus judicieux de précalculer les éléments du corps $\mathbb{F}_{2^m}$ et de conserver dans un tableau la valeur de leur cœfficient constant. Cela est calculé grâce à un petit système dont les périphériques de stockage contiennent $(1\ 0\ 0\ \ldots 0)$ initialement, représenté sur le schéma 1.10 dont l'algorithme est 1.3.10.

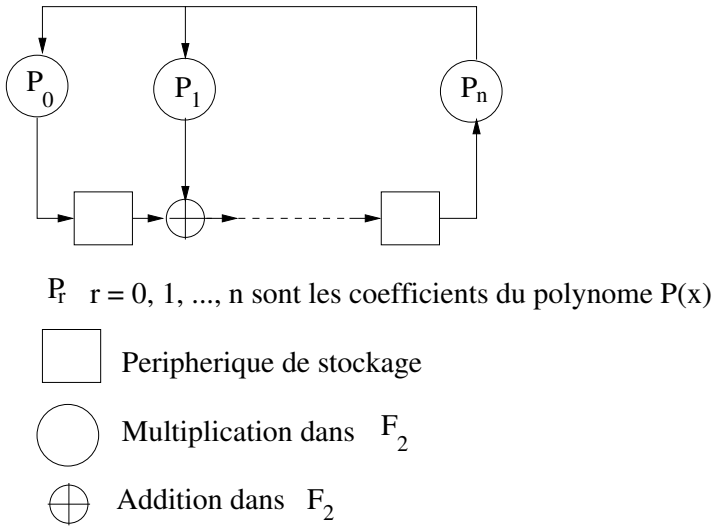

FIG. 1.10 – Schéma de calcul des éléments du corps

Algorithme 1.3.10 Calcul des éléments du corps $\frac{\mathbb{F}_2}{(P)}$

Entrées: m le degré souhaité et P un polynôme primitif
Sorties: Le tableau des cœfficients constants de tous les éléments du corps

1: $e[0] = 1$
2: $elt = 1$
3: **Pour** $i = 1$ à $2^m - 1$ **Faire**
4: **Si** $\deg(elt) = m - 1$ **Alors**
5: $p = 1$
6: **Fin si**
7: **Pour** $j = 1$ à m **Faire**
8: $t = elt_{j-1} + elt_{m-1} \times P_j \mod 2$
9: **Si** $t = 1$ **Alors**
10: $p = p + X^j$
11: **Fin si**
12: **Fin pour**
13: $e[i] = p_0$
14: $elt = p$
15: **Fin pour**
16: **Retourner** e

Pour résumer, on va appliquer l'algorithme 1.3.11. Cet algorithme effectue le calcul des polynômes réciproques des polynômes Q trouvés, ce qui en pratique n'est pas nécessaire.

En effet, le polynôme réciproque de tout polynôme primitif est également primitif. On va donc trouver tous les polynômes réciproques des polynômes primitifs mais qui seront primitifs eux aussi, donc on les aura tous également.

Algorithme 1.3.11 Algorithme de calcul des polynômes primitifs de degré m

Entrées: m, le degré souhaité

Sorties: Tous les polynômes primitifs de degré m

1: Calculer un représentant s par classe cyclotomique tel que $PGCD(s, 2^m - 1) = 1$ (voir algorithme 1.3.8).
2: Chercher un polynôme primitif noté P (voir algorithme 1.3.7)
3: Calculer tous les éléments du corps $\frac{\mathbb{F}_2[X]}{(P)}$ et ne garder que leur cœfficient constant dans un tableau e (voir algorithme 1.3.10).
4: **Pour** j de 1 à $\frac{\varphi(2^m - 1)}{m}$ **Faire**
5: $k = classe[j]$ le représentant de la j-ème classe cyclotomique.
6: calculer $s_i = e[ki]$, i allant de 0 à $2m - 1$.
7: Résoudre le système linéaire formé par la séquence s_i à l'aide de l'algorithme 1.3.9 de Berlekamp-Massey et trouver ainsi un polynôme Q
8: Calculer son polynôme réciproque Q' et l'ajouter à la liste des polynômes primitifs.
9: **Fin pour**

1.3.1.1.4 Polynômes primitifs de petits poids

Les trinômes et pentanômes étant, en pratique, des polynômes souvent utilisés comme polynômes de rétroaction d'un brasseur synchrone, il peut être intéressant de calculer les polynômes primitifs de petits poids. Pour cela, on utilise l'algorithme 1.3.4 de test d'irréductibilité et de primitivité des polynômes et on l'applique uniquement aux polynômes de poids d donné.

1.3.1.1.5 Comparaison de ces algorithmes

Complexité

Notons L le degré des polynômes primitifs recherchés. Nous allons donner la complexité pour la recherche de tous les polynômes primitifs de degré L.
Tout d'abord, dans la méthode de factorisation du polynôme cyclotomique, il y a deux étapes, celle de la construction de ce polynôme, puis celle de la factorisation. L'algorithme 1.3.5 construisant le polynôme cyclotomique a une complexité de

$$O\left(\sqrt{L} + 3\log(L)\right)$$

et l'algorithme 1.3.6, factorisant ce polynôme, a une complexité de

$$O\left(\varphi\left(2^L - 1\right)^3 + \varphi\left(2^L - 1\right)\log\left(\frac{\varphi\left(2^L - 1\right)}{L}\right)\log(2)\right)$$

Ces deux complexités sont expliqués aux pages 389 et 379 de [4]. Comme la complexité de l'algorithme 1.3.5 est négligeable devant celle de 1.3.6, on en déduit que cet algorithme a une complexité de

$$O\left(\varphi\left(2^L - 1\right)^3 + \varphi\left(2^L - 1\right)\log\left(\frac{\varphi\left(2^L - 1\right)}{L}\right)\log(2)\right)$$

Puis, dans la méthode de test de l'irréductibilité et de la primitivité de tous les polynômes de degré L, on a deux algorithmes, celui de Ben-Or pour tester le caractère irréductible d'un polynôme (1.3.2), et celui qui teste le caractère primitif (1.3.3).
On va tester l'irréductibilité de tous les polynômes, donc on va effectuer 2^L tests, et chaque test va nécessiter environ au plus $2L$ opérations. L'algorithme 1.3.2 va donc avoir une complexité de $2^L \times (2L) = 2^{L+1}L$.
Ensuite, pour les polynômes irréductibles trouvés, on va tester leur primitivité. D'après le Lemme 14.38 page 384 de [4], on va effectuer moins de $\dfrac{2^L}{L}$ tests, et chaque test va nécessiter au plus $\frac{2^L-1}{3} - L + 1$ opérations car, d'une part, $2^L - 1$ est impair et son plus grand diviseur sera inférieur ou égal à $\frac{2^L-1}{3}$, et d'autre part, on ne va pas effectuer le test pour les $L-1$ premiers exposants de $X \mod P(X)$ car ils seront nécessairement différents de 1. Cet algorithme effectuera donc un nombre d'opérations inférieur à :

$$\frac{2^L}{L}\left(\frac{2^L-1}{3} - L + 1\right)$$

Par conséquent, la méthode de test de l'irréductibilité et de la primitivité de tous les polynômes de degré L aura une complexité de

$$O\left(\frac{2^{2L}}{3L}\right)$$

Enfin, dans la méthode de calcul de tous les polynômes minimaux, on lance principalement trois algorithmes : celui de calcul des représentants des classes cyclotomiques (1.3.8), celui de calcul des éléments du corps (1.3.10) et celui de Berlekamp-Massey (1.3.9). L'algorithme 1.3.8 nécessite $2^L - 1 + \varphi\left(2^L - 1\right)$ opérations. L'algorithme 1.3.10 effectue $2L(2^L-1)$ opérations. Et l'algorithme de Berlekamp-Massey 1.3.9 s'exécute avec un nombre d'opérations de $\varphi\left(2^L - 1\right) L$. On aura donc une complexité totale de :

$$O\left(\varphi\left(2^L - 1\right) L + 2L\left(2^L - 1\right)\right)$$

Testons le nombres d'opérations de chaque algorithme pour $L = 10$.

- La méthode de factorisation du polynôme cyclotomique nécessite théoriquement environ 216 001 703 opérations.
- La méthode de test nécessite environ 60 320 opérations.
- La méthode de calcul des polynômes minimaux nécessite environ 28 083 opérations.

La méthode théoriquement la moins coûteuse est la dernière méthode ce que nous allons vérifier en pratique.

Temps de calcul

On s'est fixé au départ comme objectif de calculer tous les polynômes primitifs de degré inférieur ou égal à 30, car il y en a déjà un certain nombre, ce qui limite au niveau de la mémoire. L'algorithme 1.3.5 nous permet d'aller seulement jusqu'au degré 10 en temps et mémoire raisonnable, il n'est donc pas assez efficace. L'algorithme de test est très rapide lorsque 2^L-1 est premier car son plus grand diviseur strict est 1. Il est utilisable en pratique jusqu'au degré 20. Le dernier algorithme, le plus rapide théoriquement, est également le plus rapide en pratique et va nous permettre d'atteindre le degré 30. Le tableau 1.11 donne les différents temps de calcul sur un Intel Pentium 4 à 2.8 GHz. Tous ces polynômes sont d'abord précalculés et écrits dans un fichier car la mémoire vive ne peut pas tous les stocker. On a donc un fichier contenant 49 098 768 polynômes en commençant au degré 5 (car en pratique, les brasseurs n'ont pas de polynômes de rétroaction de degré inférieur à 5) et jusqu'au degré 30.

degré	nombre de polynômes primitifs	temps de calcul
17	7 710	< 1 s
18	7 776	1 s
19	27 594	1 s
20	24 000	3 s
21	84 672	6 s
22	120 032	12 s
23	356 960	24 s
24	276 480	42 s
25	1 296 000	1 min 59 s
26	1 719 900	3 min 34 s
27	4 202 496	7 min 58 s
28	4 741 632	13 min 37 s
29	18 407 808	38 min 6 s
30	17 820 000	57 min 5 s

FIG. 1.11 – Temps de calcul des polynômes primitifs en fonction du degré

1.3.1.2 Calcul de tous les polynômes irréductibles

Si jamais le polynôme de rétroaction utilisé par le brasseur n'est pas primitif, il est intéressant de calculer tous les polynômes irréductibles de degré m afin de le retrouver. On pourrait tester l'irréductibilité de tous les polynômes mais il existe un algorithme plus rapide, basé sur le même principe que celui du calcul des polynômes minimaux des éléments du corps pour le calcul des polynômes primitifs.

On a vu dans le théorème 4 qu'un polynôme P est irréductible si et seulement si P divise $X^{2^m} - X$ et que $PGCD\left(X^{2^{\frac{m}{t}}} - X, P\right) = 1$ pour tous les diviseurs premiers t de m. Donc tous les polynômes irréductibles de degré m divisent $X^{2^m} - X$ et donc aussi $X^{2^m-1} - 1$. Or, d'après [11], il y a une correspondance bijective entre facteurs irréductibles de $X^{2^m-1} - 1$ et classes cyclotomiques modulo $2^m - 1$. En particulier le nombre de facteurs irréductibles de $X^{2^m-1} - 1$ égale le nombre de classes cyclotomiques. Le degré d'un facteur irréductible égale le cardinal de la classe cyclotomique associée. Donc toutes les classes cyclotomiques de cardinal m sont associées à un polynôme irréductible de degré m. Et pour calculer ces polynômes, il suffit d'appliquer l'algorithme 1.3.11, en remplaçant l'étape de calcul d'un représentant s par classe cyclotomique tel que $PGCD(s, 2^m - 1) = 1$ par le calcul d'un représentant par classe cyclotomique tel que $\#(\text{classe cyclotomique}) = m$. On aura donc l'algorithme 1.3.12 de calcul des représentants des classes cyclotomiques.

Il est également intéressant de calculer seulement les trinômes et pentanômes irréductibles de degré m. Pour cela, il suffit de tester l'irréductibilité de tous les polynômes de poids 3 ou 5.

Algorithme 1.3.12 Calcul des représentants de chaque classe cyclotomique

Entrées: m le degré souhaité

Sorties: le tableau contenant les représentants k de chaque classe cyclotomique tel que $\#(\text{classe cyclotomique}) = m$.

```
1: Initialiser deux tableaux tab et c de taille 2^m − 1
2: Pour k = 1 à 2^m − 1 Faire
3:    nombre = 0
4:    Si tab[k] = 0 Alors
5:       tab[k] = 1
6:       nombre = nombre + 1
7:       e = 2k mod 2^m − 1
8:       Tant que e ≠ k Faire
9:          tab[e] = 1
10:         nombre = nombre + 1
11:         e = 2e mod 2^m − 1
12:      Fin tant que
13:      Si nombre = m Alors
14:         c[k] = 1
15:      Fin si
16:   Fin si
17: Fin pour
18: Retourner c
```

1.3.2 Test statistique

On va donc tester tous les polynômes trouvés précédemment comme polynômes de rétroaction possibles de la suite, et pour chaque polynôme testé, tous les états initiaux possibles. Comme cela est écrit dans l'algorithme 1.3.1, on va construire une suite $(s_n)_{n<N}$ qui sera la suite issue du LFSR engendré par l'état initial et le polynôme de rétroaction testés. Puis, cette suite va être additionnée bit à bit avec la suite d'entrée, pour donner une suite $(w_n)_{n<N}$ que l'on va tester.

On aura deux cas :

1. Soit la suite d'entrée $(y_n)_{n<N}$ n'a pas pour paramètres le polynôme de rétroaction ni l'état initial qui ont construit $(s_n)_{n<N}$
2. Soit la suite d'entrée $(y_n)_{n<N}$ a pour paramètres ceux de $(s_n)_{n<N}$

Notons X_N la variable aléatoire définie par :

$$X_N = \sum_{n=0}^{N-1} w_n = \omega(w_n)$$

avec ω la fonction calculant le poids d'une suite binaire.

Dans le premier cas, X_N va suivre une loi binomiale de paramètres N et $\frac{1}{2}$ car chaque bit de w_n suit une loi de Bernoulli de paramètre $\frac{1}{2}$. En effet, $P[X_t = 1] = P[X_t = 0] = \frac{1}{2}$, $\forall t < N$. Par le théorème central limite, on peut approcher cette loi par la loi normale d'espérance $\frac{N}{2}$ et de variance $\frac{N}{4}$.
Dans le second cas, X_N va suivre une loi binomiale de paramètres N et $\frac{1}{2} - \varepsilon$ car $\forall t < N,\ P[X_t = 0] = \frac{1}{2} + \varepsilon$ et donc $\forall t < N,\ P[X_t = 1] = 1 - \left(\frac{1}{2} + \varepsilon\right) = \frac{1}{2} - \varepsilon$. De

même, on peut approcher cette loi par la loi normale d'espérance $\left(\frac{1}{2}-\varepsilon\right)N$ et de variance $\left(\frac{1}{4}-\varepsilon^2\right)N$.

Les deux cas peuvent donc être traduits en termes de lois de probabilité par :
- H_0 : X_N suit la loi normale d'espérance $\frac{N}{2}$ et de variance $\frac{N}{4}$, si $(y_n)_{n<N}$ n'a pas pour paramètres le polynôme de rétroaction ni l'état initial qui ont construit $(s_n)_{n<N}$.
- H_1 : X_N suit la loi normale d'espérance $\left(\frac{1}{2}-\varepsilon\right)N$ et de variance $\left(\frac{1}{4}-\varepsilon^2\right)N$, si $(y_n)_{n<N}$ a pour paramètres ceux de $(s_n)_{n<N}$

Comme dans le cas de la recherche du polynôme de rétroaction de la thèse de M. Cluzeau, on va utiliser un seuil $T > 0$ pour faire un choix entre H_0 et H_1. Comme pour le même N et avec $\varepsilon > 0$, on a $\left(\frac{1}{2}-\varepsilon\right)N \leqslant \frac{N}{2}$, on va accepter H_1 si $X_N < T$. Et on va accepter H_0 dans le cas contraire. Le nombre de bits N requis de la suite $(y_n)_{n<N}$ va dépendre des probabilités de fausse alarme P_f et de non-détection P_n définies dans le tableau 1.3. Afin de simplifier l'écriture, on va renommer la variable aléatoire X_N en X'_N quand elle suit la loi normale correspondant à l'hypothèse H_0. On a donc :

$$X_N \sim \mathcal{N}\left(\left(\frac{1}{2}-\varepsilon\right)N, \left(\frac{1}{4}-\varepsilon^2\right)N\right)$$

et

$$X'_N \sim \mathcal{N}\left(\frac{N}{2}, \frac{N}{4}\right)$$

Comme $E(X_N) \leqslant E(X'_N)$, on va choisir $T = E(X_N) + a\sigma(X_N)$ avec $\sigma(X_N)$ l'écart-type de la variable aléatoire X_N. On a donc :

$$\begin{aligned} P_f &= P[X'_N \leqslant T] \\ &= P[X'_N \leqslant E(X_N) + a\sigma(X_N)] \\ &= P[X'_N \leqslant E(X'_N) + b\sigma(X'_N)] \text{ avec } b \leqslant 0 \\ &= \phi(b) \text{ avec } \phi \text{ la fonction de répartition associée à la densité de la loi normale} \end{aligned}$$

D'où :

$$b = \phi^{-1}(P_f)$$

Et :

$$\begin{aligned} P_n &= P[X_N > T] \\ &= P[X_N > E(X_N) + a\sigma(X_N)] \\ &= 1 - P[X_N \leqslant E(X_N) + a\sigma(X_N)] \\ &= 1 - \phi(a) \end{aligned}$$

D'où :

$$a = \phi^{-1}(1 - P_n)$$

En calculant P_f on a utilisé l'égalité $P[X'_N \leqslant E(X_N) + a\sigma(X_N)] = P[X'_N \leqslant E(X'_N) +$

$b\sigma(X'_N)]$. Pour que cette égalité soit vraie, il faut que :

$$\begin{aligned}
E(X_N) + a\sigma(X_N) &= E(X'_N) + b\sigma(X'_N) \\
\left(\frac{1}{2} - \varepsilon\right) N + a\sqrt{\left(\frac{1}{4} - \varepsilon^2\right) N} &= \frac{1}{2}N + \frac{b\sqrt{N}}{2} \\
-\varepsilon\sqrt{N} + a\sqrt{\frac{1}{4} - \varepsilon^2} &= \frac{b}{2} \\
\sqrt{N} &= \frac{a\sqrt{\frac{1}{4} - \varepsilon^2} - \frac{b}{2}}{\varepsilon} \\
N &= \frac{\left(a\sqrt{\frac{1}{4} - \varepsilon^2} - \frac{b}{2}\right)^2}{\varepsilon^2}
\end{aligned}$$

On a donc l'algorithme 1.3.13.

Algorithme 1.3.13 Algorithme de recherche des paramètres du brasseur synchrone

Entrées: la suite $(y_t)_{t<N}$ reçue, la liste L_P des polynômes à tester, les probabilités de fausse-alarme P_f et de non-détection P_n, le biais ε

Sorties: l'état initial et le polynôme de rétroaction du brasseur ou échec

$a = \phi^{-1}(1 - P_n)$, $b = \phi^{-1}(P_f)$
$n = \frac{\left(a\sqrt{\frac{1}{4}-\varepsilon^2} - \frac{b}{2}\right)^2}{\varepsilon^2}$
Si $N < n$ **Alors**
  Retourner échec
Fin si
$T = \left(\frac{1}{2} - \varepsilon\right) N + a \times \sqrt{\left(\frac{1}{4} - \varepsilon^2\right) N}$
Pour tout polynôme P dans L_P **Faire**
  $L = \deg(P)$
  Pour tout état initial i de longueur L non nul **Faire**
    Calculer $\forall t < N,\ s_t = LFSR(i, P)$
    $X = 0$
    Pour $t = 0$ à $N - 1$ **Faire**
      $w_t = s_t + y_t \mod 2$
      Si $w_t = 1$ **Alors**
        $X = X + 1$
      Fin si
    Fin pour
    Si $X < T$ **Alors**
      Retourner P, i
    Fin si
  Fin pour
Fin pour

1.3.3 Implémentation et résultats

1.3.3.1 Complexité

1.3.3.1.1 Dans le cas des polynômes primitifs

Cet algorithme de recherche exhaustive va tester tous les polynômes primitifs jusqu'au bon polynôme de rétroaction de degré L, et pour chacun des polynômes P, il va tester tous les états initiaux de longueur $\deg(P)$ non nuls. Pour chaque état initial, il va effectuer $O(N)$ opérations. Et le nombre d'états initiaux testés va être au maximum (en commençant par les polynômes de degré 5) de

$$\sum_{i=5}^{L} \frac{\varphi\left(2^i-1\right)}{i} \times (2^i-1)$$

Donc la complexité de cet algorithme sera de

$$\frac{\left(a\sqrt{\frac{1}{4}-\varepsilon^2}-\frac{b}{2}\right)^2}{\varepsilon^2} \sum_{i=5}^{L} \frac{\varphi\left(2^i-1\right)(2^i-1)}{i}$$

1.3.3.1.2 Dans le cas des polynômes irréductibles

On va, dans ce cas, tester tous les polynômes irréductibles jusqu'au bon polynôme de rétroaction de degré L, ce qui revient à tester moins de $\frac{2^L}{L}$ polynômes. Donc la complexité sera de :

$$\frac{\left(a\sqrt{\frac{1}{4}-\varepsilon^2}-\frac{b}{2}\right)^2}{\varepsilon^2} \sum_{i=5}^{L} \frac{2^i(2^i-1)}{i}$$

1.3.3.2 Implémentation

Cet algorithme a été implémenté en langage C et testé sur différents brasseurs synchrones. On a calculé le temps moyen de cet algorithme pour un L donné, c'est-à-dire si, par exemple, $L = 8$ et qu'on teste les polynômes primitifs, on va tester tout d'abord $\frac{\varphi(2^5-1)}{5}+\frac{\varphi(2^6-1)}{6}+\frac{\varphi(2^7-1)}{7}$ polynômes, puis au maximum $\frac{\varphi(2^8-1)}{8}$ polynômes si le polynôme de rétroaction est le dernier testé, et $\frac{\varphi(2^8-1)}{16}$ en moyenne, si le polynôme de rétroaction se trouve au milieu des autres. Le tableau 1.12 donne les temps de calculs moyens sur un Intel Pentium 4 à 2.8 GHz en fonction de L et avec des probabilités de fausse-alarme et de non détection de 2.10^{-7} et de 0.1 respectivement ainsi qu'un biais de 0.1.

polynôme de rétroaction	nombre moyen de polynômes testés	temps
$X^8+X^7+X^6+X^5+X^4+X^2+1$	38	0 s
$X^{10}+X^6+X^5+X^3+X^2+X+1$	124	2 s
$X^{13}+X^{11}+X^{10}+X^8+X^4+X^3+1$	789	3 min 24 s
$X^{15}+X^{13}+X^6+X^5+X^3+X+1$	2760	49 min 25 s

FIG. 1.12 – Temps de calcul du programme de recherche exhaustive avec des polynômes primitifs

1.4 Comparaison des deux algorithmes

1.4.1 Comparaison du nombre de bits nécessaire

Dans l'algorithme de M.Cluzeau, la partie nécessitant le plus de bits de suite est la reconstruction du polynôme de rétroaction. Elle nécessite au moins

$$n \geqslant \frac{(a+b\sqrt{13})^2}{64\varepsilon^6}$$

bits de suite avec $a = \phi^{-1}\left(1-\frac{P_f}{2}\right)$ et $b = -\phi^{-1}(P_n)$.

Dans l'algorithme de recherche exhaustive, on a besoin d'au moins

$$n \geqslant \frac{\left(a\sqrt{\frac{1}{4}-\varepsilon^2}-\frac{b}{2}\right)^2}{\varepsilon^2}$$

bits de la suite d'entrée avec $a = \phi^{-1}(1-P_n)$ et $b = \phi^{-1}(P_f)$.

On constate que le nombre de bits dépend d'un coté d'ε^6 et de l'autre d'ε^2 et que le nombre de bits de la recherche exhaustive va donc être très inférieur avec le même ε.
Cela nous donne en pratique les valeurs du tableau 1.13 pour une probabilité de fausse-alarme de 2.10^{-7} et une probabilité de non-détection de 0.1.

biais	Méthode Cluzeau	Méthode exhaustive
0.3	2067	104
0.2	23 543	243
0.1	1 506 768	1 000
0.05	96 433 175	4 025

FIG. 1.13 – Tableau du nombre de bits nécessaire en fonction du biais

On peut donc en conclure que si le nombre de bits de suite à notre disposition est limité, il vaut mieux utiliser la méthode de recherche exhaustive, puisque l'autre méthode n'aboutira pas.

1.4.2 Complexité

La complexité de l'algorithme de recherche du polynôme de rétroaction est de :

$$2^{L+1}\frac{\left(a+b\sqrt{13}\right)^2}{64\varepsilon^6}$$

La complexité de l'algorithme de recherche de l'état initial basé sur le décodage des codes LDPC est négligeable devant celle de la recherche du polynôme de rétroaction.
La complexité de l'algorithme de recherche exhaustive est de :

$$\frac{\left(a\sqrt{\frac{1}{4}-\varepsilon^2}-\frac{b}{2}\right)^2}{\varepsilon^2}\sum_{i=5}^{L}\frac{\varphi\left(2^i-1\right)\left(2^i-1\right)}{i}$$

dans le cas des polynômes primitifs et de :

$$\frac{\left(a\sqrt{\frac{1}{4}-\varepsilon^2}-\frac{b}{2}\right)^2}{\varepsilon^2}\sum_{i=5}^{L}\frac{2^i(2^i-1)}{i}$$

dans le cas des polynômes irréductibles. Ces complexités nous laissent supposer que plus ε va être proche de zéro, plus l'algorithme de recherche exhaustive va être plus efficace que celui de M.Cluzeau. Nous allons voir si comment cela se passe en pratique.

1.4.3 Comparaison pratique des temps de calcul

On va comparer ces deux méthodes avec une probabilité de fausse-alarme de 2.10^{-7} car elle permet d'avoir un bon pourcentage de réussite dans le cas de la recherche du polynôme de rétroaction, et une probabilité de non-détection de 0.1 car c'est la probabilité maximale pour avoir un bon pourcentage de détection dans le cas de la recherche exhaustive.

1.4.3.1 Dans le cas des polynômes primitifs

1.4.3.1.1 Si on ne possède aucune information sur le polynôme de rétroaction

Avec un biais de 0.2

Avec un biais de 0.2, on a besoin d'environ 60 000 bits de suite pour reconstruire le polynôme de rétroaction avec la méthode de M.Cluzeau et de seulement 300 bits avec la recherche exhaustive. Dans le tableau 1.14, on constate que la méthode de M.Cluzeau est toujours plus efficace que la recherche exhaustive.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^8+X^7+X^6+X^5+X^2+X+1$	<1 s	<1 s
$X^9+X^6+X^4+X^3+X^2+X+1$	<1 s	<1 s
$X^{10}+X^9+X^6+X^5+X^4+X^3+X^2+X+1$	1 s	<1 s
$X^{11}+X^9+X^8+X^7+X^2+X+1$	1 s	3 s
$X^{12}+X^{10}+X^9+X^8+X^7+X^5+X^4+X^3+X^2+X+1$	1 s	10 s
$X^{13}+X^{11}+X^{10}+X^9+X^8+X^7+X^6+X^5+X^4+X^3+X^2+X+1$	1 s	51 s

FIG. 1.14 – Temps de calcul des deux méthodes avec $\varepsilon = 0.2$

Avec un biais de 0.1

Avec $\varepsilon = 0.1$, le nombre de bits nécessaire pour retrouver le polynôme de rétroaction avec la méthode de M.Cluzeau est d'environ 2 000 000, et d'environ 1 200 pour la recherche exhaustive. En ce qui concerne l'efficacité de ces deux méthodes, on constate que jusqu'au degré 14, c'est la recherche exhaustive qui est la plus efficace et que cela s'inverse pour les polynômes de rétroaction de degré supérieur à 14. On voit tout cela dans le tableau 1.15.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^8+X^7+X^6+X^5+X^2+X+1$	12 s	<1 s
$X^{10}+X^9+X^6+X^5+X^4+X^3+X^2+X+1$	49 s	3 s
$X^{13}+X^7+X^6+X^5+X^4+X+1$	6 min 26 s	3 min 20 s
$X^{14}+X^{13}+X^{12}+X^7+X^5+X^4+X^3+X+1$	12 min 50 s	11 min 24 s
$X^{15}+X^{10}+X^6+X^5+X^3+X^2+1$	25 min 47 s	44 min 13 s
$X^{16}+X^{15}+X^{11}+X^{10}+X^7+X^4+X^2+X+1$	51 min 31 s	2h 13 min 3 s

FIG. 1.15 – Temps de calcul des deux méthodes avec $\varepsilon = 0.1$

Avec un biais de 0.05

Le nombre de bits nécessaire pour la méthode de M.Cluzeau est d'environ 110 000 000, alors qu'il est de seulement 5 000 bits pour la recherche exhaustive. Jusqu'aux polynômes de degré 19, on constate dans le tableau 1.16 que c'est la méthode de recherche exhaustive qui est la plus efficace. Pour les polynômes de degré supérieur, la méthode de M.Cluzeau est plus rapide.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^8+X^7+X^6+X^5+X^2+X+1$	13 min 52 s	1 s
$X^{10}+X^9+X^6+X^5+X^4+X^3+X^2+X+1$	55 min 28 s	13 s
$X^{13}+X^7+X^6+X^5+X^4+X+1$	7h 24 min 22 s	15 min 58 s
$X^{14}+X^{13}+X^{12}+X^7+X^5+X^4+X^3+X+1$	14h 48 min 32 s	42 min 59 s
$X^{15}+X^{10}+X^6+X^5+X^3+X^2+1$	$\approx$[2]1 jour 5h 35 min	2h 17 min 37 s
$X^{16}+X^{15}+X^{11}+X^{10}+X^7+X^4+X^2+X+1$	$\approx$ 2 jours 11h	6h 52 min 51 s
$X^{17}+X^{14}+X^{13}+X^{10}+X^4+X^3+X^2+X+1$	$\approx$ 4 jours 22h	$\approx$ 1 jour 10 h
$X^{18}+X^{17}+X^{15}+X^{14}+X^{13}+X^{12}+X^9+X^7+X^3+X+1$	$\approx$ 9 jours 20h	$\approx$ 4 jours 10h
$X^{19}+X^{15}+X^{14}+X^{13}+X^{12}+X^{11}+X^9+X^8+X^4+X^3+X^2+X+1$	$\approx$ 19 jours 17h	$\approx$ 20 jours 19h

FIG. 1.16 – Temps de calcul des deux méthodes avec $\varepsilon = 0.05$

1.4.3.1.2 Si on connaît la longueur du registre

Avec un biais de 0.2

Si on connaît la longueur du registre, on obtient les mêmes résultats que si on ne la

[2]Le signe $\approx$ signifie que le test n'a pas réellement été lancé sur la machine, mais que le temps de calcul a été calculé théoriquement grâce au nombre d'opérations de l'algorithme

connaît pas, c'est-à-dire que la méthode de M.Cluzeau est plus efficace comme le voit dans le tableau 1.17.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^8+X^7+X^6+X^5+X^2+X+1$	<1 s	<1 s
$X^9+X^6+X^4+X^3+X^2+X+1$	<1 s	<1 s
$X^{10}+X^9+X^6+X^5+X^4+X^3+X^2+X+1$	1 s	<1 s
$X^{11}+X^9+X^8+X^7+X^2+X+1$	1 s	2 s
$X^{12}+X^{10}+X^9+X^8+X^7+X^5+X^4+X^3+X^2+X+1$	1 s	4 s
$X^{13}+X^{11}+X^{10}+X^9+X^8+X^7+X^6+X^5+X^4+X^3+X^2+X+1$	1 s	35 s

FIG. 1.17 – Temps de calcul des deux méthodes avec $\varepsilon = 0.2$

Avec un biais de 0.1

Si on connaît la longueur du registre, le temps de recherche exhaustive sera moins long. On ne gagne cependant qu'un peu de temps, donc jusqu'au degré 14, la méthode de recherche exhaustive est la plus efficace et cela s'inverse pour les polynômes de degré supérieur à 14.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^8+X^7+X^6+X^5+X^2+X+1$	12 s	<1 s
$X^{10}+X^9+X^6+X^5+X^4+X^3+X^2+X+1$	49 s	2 s
$X^{13}+X^7+X^6+X^5+X^4+X+1$	6 min 26 s	2 min 26 s
$X^{14}+X^{13}+X^{12}+X^7+X^5+X^4+X^3+X+1$	12 min 50 s	5 min 50 s
$X^{15}+X^{10}+X^6+X^5+X^3+X^2+1$	25 min 47 s	27 min 47 s
$X^{16}+X^{15}+X^{11}+X^{10}+X^7+X^4+X^2+X+1$	51 min 31 s	1h 3 min 13 s
$X^{17}+X^{14}+X^{13}+X^{10}+X^4+X^3+X^2+X+1$	1h 43 min 16 s	7h 55 min 59 s

FIG. 1.18 – Temps de calcul des deux méthodes avec $\varepsilon = 0.1$

Avec un biais de 0.05

On constate dans le tableau 1.19 que jusqu'aux polynômes de rétroaction de degré 20, la méthode de recherche exhaustive est plus efficace que la méthode de M.Cluzeau, qui elle, est plus rapide à partir des polynômes de degré 21.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^{10}+X^9+X^6+X^5+X^4+X^3+X^2+X+1$	55 min 28 s	7 s
$X^{13}+X^7+X^6+X^5+X^4+X+1$	7h 24 min 22 s	10 min 5 s
$X^{14}+X^{13}+X^{12}+X^7+X^5+X^4+X^3+X+1$	14h 48 min 32 s	24 min 12 s
$X^{15}+X^{10}+X^6+X^5+X^3+X^2+1$	1 jour 5h 35 min	1h 55 min 15 s
$X^{16}+X^{15}+X^{11}+X^{10}+X^7+X^4+X^2+X+1$	$\approx$ 2 jours 11h 10 min	4h 22 min 16 s
$X^{17}+X^{14}+X^{13}+X^{10}+X^4+X^3+X^2+X+1$	$\approx$ 4 jours 22h	$\approx$ 1 jour 8h 54 min
$X^{18}+X^{17}+X^{15}+X^{14}+X^{13}+X^{12}+X^9+X^7+X^3+X+1$	$\approx$ 9 jours 20h	$\approx$ 2 jours 18h
$X^{19}+X^{15}+X^{14}+X^{13}+X^{12}+X^{11}+X^9+X^8+X^4+X^3+X^2+X+1$	$\approx$ 19 jours 17h	$\approx$ 19 jours 15h
$X^{20}+X^{19}+X^{18}+X^{15}+X^{13}+X^{12}+X^{11}+X^8+X^7+X^6+X^3+X+1$	$\approx$ 39 jours 10h	$\approx$ 34 jours 3h
$X^{21}+X^{20}+X^{15}+X^{14}+X^8+X^7+X^5+X^4+X^3+X+1$	$\approx$ 78 jours 20h	$\approx$ 240 jours 23h

FIG. 1.19 – Temps de calcul des deux méthodes avec $\varepsilon = 0.05$

1.4.3.1.3 Si on sait que le polynôme de rétroaction est un trinôme

Un programme testant uniquement les trinômes primitifs a également été implémenté. On va voir quelles sont les performances de ce dernier, en le comparant à l'algorithme de recherche de polynômes de M.Cluzeau lorsque le polynôme est un trinôme.

Avec un biais de 0.1

Avec un biais de 0.1, on obtient les temps de calcul du tableau 1.20. On constate qu'à partir d'un trinôme de degré 17, la méthode de Cluzeau est plus efficace que la recherche exhaustive.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
X^7+X^4+1	2 s	$<$ 1 s
X^9+X^5+1	3 s	$<$ 1 s
$X^{11}+X^9+1$	3 s	$<$ 1 s
$X^{15}+X^7+1$	6 s	3 s
$X^{17}+X^{11}+1$	9 s	21 s
$X^{18}+X^{11}+1$	10 s	43 s

FIG. 1.20 – Temps de calcul des deux méthodes avec $\varepsilon = 0.1$

Avec un biais de 0.05

Les temps de calcul des deux méthodes avec un biais de 0.05 sont dans le tableau 1.21. On constate qu'à partir du degré 21, la méthode de M.Cluzeau est plus efficace que la recherche exhaustive.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^7 + X^4 + 1$	1 min 40 s	< 1 s
$X^9 + X^5 + 1$	2 min 38 s	< 1 s
$X^{11} + X^9 + 1$	3 min 42 s	1 s
$X^{15} + X^7 + 1$	7 min 16 s	13 s
$X^{17} + X^{11} + 1$	9 min 48 s	1 min 37 s
$X^{18} + X^{11} + 1$	10 min 24 s	3 min 21 s
$X^{20} + X^{17} + 1$	14 min 35 s	8 min 14 s
$X^{21} + X^{19} + 1$	14 min 15 s	18 min 12 s
$X^{22} + X^{21} + 1$	15 min 40 s	29 min 43 s

FIG. 1.21 – Temps de calcul des deux méthodes avec $\varepsilon = 0.05$

1.4.3.1.4 Si on sait que le polynôme de rétroaction est un pentanôme

J'ai également implémenté un programme permettant d'effectuer la recherche exhaustive uniquement sur les pentanômes.

Avec un biais de 0.1

Lorsque l'on recherche un pentanôme, la recherche exhaustive est plus efficace que la méthode de M.Cluzeau jusqu'aux polynômes de rétroaction de degré 30 comme on le voit dans le tableau 1.22.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^{16} + X^{12} + X^3 + X + 1$	51 min 12 s	4 min 6 s
$X^{17} + X^{13} + X^3 + X + 1$	1h 42 min 24 s	12 min 19 s
$X^{18} + X^{14} + X^5 + X + 1$	3h 24 min 48 s	25 min 59 s
$X^{19} + X^{14} + X^3 + X + 1$	6h 49 min 36 s	1h 1 min 49 s
$X^{20} + X^{13} + X^9 + X + 1$	13h 39 min 12 s	2h 7 min 56 s
$X^{21} + X^8 + X^2 + X + 1$	$\approx$ 1 jour 3h	4h 46 min 2 s
$X^{22} + X^{11} + X^8 + X + 1$	$\approx$ 2 jours 6h	9h 47 min 27 s
$X^{23} + X^{20} + X^{17} + X^{14} + 1$	$\approx$ 4 jours 12h	$\approx$ 1 jour 3h
$X^{24} + X^8 + X^5 + X^2 + 1$	$\approx$ 9 jours	$\approx$ 2 jours 2h
$X^{25} + X^{16} + X^8 + X^2 + 1$	$\approx$ 18 jours	$\approx$ 5 jours 18h
$X^{26} + X^{11} + X^7 + X^6 + 1$	$\approx$ 36 jours	$\approx$ 12 jours 12h
$X^{27} + X^{18} + X^5 + X^3 + 1$	$\approx$ 72 jours	$\approx$ 25 jours 10h
$X^{28} + X^{16} + X^9 + X^8 + 1$	$\approx$ 144 jours	$\approx$ 47 jours 3h
$X^{29} + X^{10} + X^9 + X^4 + 1$	$\approx$ 288 jours	$\approx$ 109 jours 3h
$X^{30} + X^{14} + X^9 + X^8 + 1$	$\approx$ 576 jours	$\approx$ 213 jours 10h

FIG. 1.22 – Temps de calcul des deux méthodes avec $\varepsilon = 0.1$

Avec un biais de 0.05

On constate dans le tableau 1.23, que la méthode de M.Cluzeau n'est pas très praticable au delà du degré 20 pour un biais de 0.05 et que la méthode de recherche exhaustive sur les pentanômes est toujours plus efficace quelque soit le degré. Si le biais statistique du brasseur est de 0.05 et que le polynome de rétroaction est de degré supérieur à 20, il est

très long de le retrouver à moins d'avoir une machine très puissante.

polynôme de rétroaction	Méthode Cluzeau	Méthode exhaustive
$X^{14}+X^{12}+X^{11}+X^5+1$	14h 47 min 28 s	3 min 19 s
$X^{15}+X^{14}+X^{13}+X+1$	$\approx$ 1 jour 5h	8 min 29 s
$X^{17}+X^{13}+X^3+X+1$	$\approx$ 4 jours 20h	53 min 33 s
$X^{19}+X^{14}+X^3+X+1$	$\approx$ 19 jours 8h	4h 28 min 44 s
$X^{21}+X^8+X^2+X+1$	$\approx$ 77 jours 8h	20h 43 min 30 s
$X^{22}+X^{11}+X^8+X+1$	$\approx$ 154 jours 16h	$\approx$ 1 jour 18h
$X^{23}+X^{20}+X^{17}+X^{14}+1$	$\approx$ 309 jours	$\approx$ 4 jours 22h
$X^{24}+X^8+X^5+X^2+1$	$\approx$ 1 an 253 jours	$\approx$ 9 jours 4h
$X^{25}+X^{16}+X^8+X^2+1$	$\approx$ 3 ans 142 jours	$\approx$ 25 jours
$X^{26}+X^{11}+X^7+X^6+1$	$\approx$ 6 ans 284 jours	$\approx$ 54 jours 10h
$X^{27}+X^{18}+X^5+X^3+1$	$\approx$ 13 ans 203 jours	$\approx$ 110 jours 12h
$X^{28}+X^{16}+X^9+X^8+1$	$\approx$ 27 ans 41 jours	$\approx$ 205 jours
$X^{29}+X^{10}+X^9+X^4+1$	$\approx$ 53 ans 47 jours	$\approx$ 1 an 109 jours
$X^{30}+X^{14}+X^9+X^8+1$	$\approx$ 106 ans	$\approx$ 2 ans 197 jours

FIG. 1.23 – Temps de calcul des deux méthodes avec $\varepsilon = 0.05$

1.4.3.2 Dans le cas des polynômes irréductibles

Lorsque l'on teste tous les polynômes irréductibles, on parcourt quelques polynômes de plus, donc les temps de calcul sont plus longs, mais ils restent du même ordre de grandeur. Cependant, pour certains polynômes, la méthode de recherche exhaustive s'avère la seule à pouvoir les retrouver. En effet, on constate que la méthode de M.Cluzeau ne retrouve pas certains polynômes irréductibles et non primitifs, comme par exemple les polynômes de la forme :

$$\frac{X^k+1}{X+1} \text{ avec } k \text{ impair}$$

ni $X^{11}+X^9+X^7+X^6+X^5+X+1$, ni $X^{15}+X^{14}+X^{13}+X^{11}+X^{10}+X^8+X^7+X^6+X^5+X^4+1$. Dans ces cas-là, la méthode de recherche du polynôme de rétroaction retourne toujours un polynôme de degré plus petit que le polynôme de rétroaction qu'il fallait retrouver. Il faudrait étudier plus finement la structure de ces cas pathologiques afin de comprendre pourquoi on ne les retrouve pas.

Chapitre 2

Brasseur auto-synchronisant

Nous allons maintenant voir la reconstruction des paramètres d'un brasseur auto-synchronisant.

2.1 Définition

Un brasseur auto-synchronisant est constitué d'un registre à décalage à rétroaction linéaire défini par un polynôme de rétroaction et lors de la remise à jour, le bit de sortie est le même que le bit entrant dans le registre. Il est représenté par le schéma 2.1.

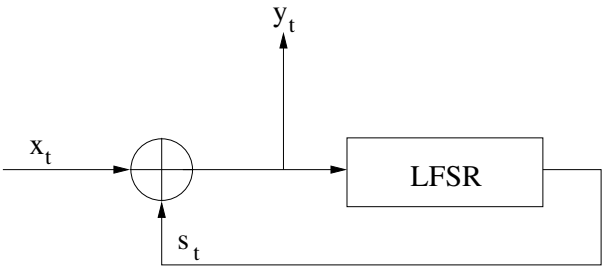

FIG. 2.1 – Brasseur auto-synchronisant

L'équation du brasseur est donc :

$$y_t = x_t + \sum_{i=0}^{L-1} c_i y_{t-(L-i)}$$

ou encore :

$$x_t = \sum_{i=0}^{L} c_i y_{t-(L-i)}$$

A l'entrée du débrasseur, si on suppose qu'il peut y avoir des erreurs, on a $y'_t = y_t + e_t$ avec $e_t = 0$ ou $e_t = 1$, et on a en sortie :

$$x'_t = \sum_{i=0}^{L} c_i y_{t-(L-i)} + \sum_{i=0}^{L} c_i e_{t-(L-i)}$$

avec c_i les cœfficients du polynôme de rétroaction du registre. Si $\forall t,\ e_t = 0$, on reconstruit les éléments x_t du message. On voit que l'inconvénient majeur du brassage auto-synchronisant est la multiplication des erreurs. En effet, une erreur sur un bit se répercute

sur tous les autres bits qui font intervenir ce bit. Ce type de brasseur a, en revanche, l'avantage de permettre la synchronisation.
Généralement, un brasseur auto-synchronisant est initialisé par l'état nul, il suffit donc de retrouver le polynôme de rétroaction du registre.

2.2 Reconstruction du polynôme de rétroaction

On présente ici la méthode de reconstruction décrite par M.Cluzeau dans [1]. Notons $(x_t)_{t\geqslant 0}$ le message, $(s_t)_{t\geqslant 0}$ la suite en sortie du LFSR et $(y_t)_{t\geqslant 0}$ le train binaire de sortie du brasseur. Dans la majorité des applications, L, la longueur du brasseur ne dépasse pas $L_{max} = 100$.

Nous allons nous placer sous la même hypothèse que dans le cas du brasseur synchrone, c'est-à-dire que l'on sait que le message d'entrée est biaisé :

$$\forall t \geqslant 0,\ P[x_t = 0] = \frac{1}{2} + \varepsilon$$

avec $\varepsilon \neq 0$. Il serait tentant d'utiliser la même technique que dans le cas du brasseur synchrone, cependant, cela est impossible car la relation liant y_t à s_t fait intervenir à la fois x_t mais aussi d'autres x_i, $i < t$. Donc, au lieu de détecter un multiple de P, on va directement étudier ce qu'il se passe pour P lui-même.

Théorème 12. Soit $(y_t)_{t\geqslant 0}$ la sortie d'un brasseur auto-synchronisant dont la suite d'entrée $(x_t)_{t\geqslant 0}$ vérifie :

$$\forall t \geqslant 0,\ P[x_t = 0] = \frac{1}{2} + \varepsilon$$

Soit $P = 1 + \sum_{j=1}^{d-1} X^{i_j}$, $(i_1 < \ldots < i_{d-1} = L)$ le polynôme de rétroaction du registre et soit :

$$z_t = y_t + \sum_{j=1}^{d-1} y_{t-i_j},\ t \geqslant L$$

Notons :

$$Z_N = \sum_{t=L}^{N-1} (-1)^{z_t}$$

$$\mu = (N-L)(2\varepsilon)$$

$$\sigma^2 = (N-L)\left(1 - (2\varepsilon)^2\right)$$

Alors la variable aléatoire

$$Z = \frac{Z_N - \mu}{\sigma}$$

tend vers une loi normale centrée réduite lorsque N tend vers l'infini.

Démonstration. Ce théorème est prouvé dans [1]. □

Nous allons donc utiliser une nouvelle fois, un test d'hypothèse avec un seuil T, mais en testant cette fois-ci tous les polynômes et pas seulement ceux de poids faible. De plus, on remarque que le nombre de bits de sortie sera ici moins important car l'écart entre les deux moyennes est de 2ε alors qu'il était de $(2\varepsilon)^3$ dans le cas synchrone. M. Cluzeau a

décidé de tester les polynômes de degré inférieur à L_{max} en faisant croître leur poids afin de minimiser le temps de calcul. La valeur du seuil est :

$$T = \frac{a^2 + ab\sqrt{1-4\varepsilon^2}}{2\varepsilon}$$

avec $a = \phi^{-1}\left(1 - \frac{P_f}{2}\right)$ et $b = -\phi^{-1}(P_n)$ avec ϕ qui est toujours la fonction de répartition associée à la densité de la loi normale. Et le nombre de bits nécessaire pour la reconstruction est de :

$$N_{min} = L_{max} + \frac{\left(a + b\sqrt{1-4\varepsilon^2}\right)^2}{4\varepsilon^2}$$

On a l'algorithme 2.2.1.

Algorithme 2.2.1 Algorithme de recherche du polynôme de rétroaction d'un brasseur autosynchronisant

Entrées: la suite y_n reçue, les probabilités de non-détection P_n et de fausse alarme P_f, le biais ε, la longueur L_{max}

Sorties: le polynôme de rétroaction du brasseur

1: $T = \frac{a^2+ab\sqrt{1-4\varepsilon^2}}{2\varepsilon}$
2: $N_{min} = L_{max} + \frac{\left(a+b\sqrt{1-4\varepsilon^2}\right)^2}{4\varepsilon^2}$
3: $d = 2$
4: **Pour tout** P de degré $< L_{max}$ et de poids d **Faire**
5: $Z = 0$
6: **Pour** $t = L_{max}$ à N_{min} **Faire**
7: $z = y_t$
8: **Pour** $i = 1$ à $deg(P)$ **Faire**
9: **Si** $c_i = 1$ **Alors**
10: $z = z + y_{t-(L-i)}$
11: **Fin si**
12: **Fin pour**
13: $Z = Z + (-1)^z$
14: **Fin pour**
15: **Si** $|Z| > T$ **Alors**
16: **Retourner** P
17: **Fin si**
18: **Fin pour**
19: $d = d + 1$
20: Aller en 4

2.3 Implémentation et résultats

Le nombre d'opérations de cet algorithme est de :

$$(N_{min} - L_{max}) \sum_{d=2}^{w} d\binom{d-1}{L_{max}} \simeq \frac{wL_{max}^{w-1}}{(w-1)!}\frac{(a+b)^2}{(2\varepsilon)^2}$$

avec w le poids du polynôme de rétroaction. J'ai implémenté cet algorithme et je l'ai testé sur différents polynômes. Les temps de calcul sur un Intel Pentium 4 à 2.8 GHz se trouvent

dans le tableau 2.2 pour un biais de 0.1 et dans le tableau 2.3 pour un biais de 0.05. On utilise une probabilité de fausse-alarme de 2.10^{-6}, une probabilité de non-détection de 10^{-5} et $L_{max} = 100$.

polynôme de rétroaction	temps
$X^8 + X^3 + X^2 + X + 1$	15 s
$X^{10} + X^6 + X^5 + X^3 + X^2 + X + 1$	6 min 4 s
$X^{15} + X^5 + X^4 + X^2 + 1$	15 s
$X^{21} + X^6 + X^5 + X^2 + 1$	16 s
$X^{29} + X^2 + 1$	< 1 s

FIG. 2.2 – Temps de recherche du polynôme de rétroaction avec $\varepsilon = 0.1$ et $N = 1002$

polynôme de rétroaction	temps
$X^8 + X^3 + X^2 + X + 1$	21 s
$X^{10} + X^6 + X^5 + X^3 + X^2 + X + 1$	24 min 48 s
$X^{15} + X^5 + X^4 + X^2 + 1$	21 s
$X^{21} + X^6 + X^5 + X^2 + 1$	22 s
$X^{29} + X^2 + 1$	< 1 s

FIG. 2.3 – Temps de recherche du polynôme de rétroaction avec $\varepsilon = 0.05$ et $N = 3734$

Le temps de calcul augmentant exponentiellement avec le poids du polynôme de rétroaction, cet algorithme nous permet seulement de reconstruire les brasseurs auto-synchronisants dont le polynôme de rétroaction est de poids faible.

Conclusion

Dans ce rapport de stage, nous avons vu comment retrouver les paramètres d'un brasseur synchrone et auto-synchronisant, lorsque le message d'entrée présente un biais statistique. Dans le cas d'un brasseur synchrone, la méthode de M.Cluzeau permettant de retrouver en deux étapes, d'abord le polynôme de rétroaction puis l'état initial est une méthode générique qui s'applique à tous les polynômes. Elle est basée sur un test statistique permettant de détecter les multiples creux du polynôme de rétroaction du registre et sur le décodage des codes LDPC. Cependant celle-ci a une complexité qui dépend d'un facteur $\frac{1}{\varepsilon^6}$ avec ε le biais du message et elle nécessite un nombre de bits important, alors qu'en pratique, lors d'une interception de données, le nombre de bits est limité.

La méthode proposée, celle de recherche exhaustive, teste pour tous les polynômes primitifs et irréductibles, tous les états initiaux possibles. Elle nécessite donc le précalcul des polynômes primitifs ou irréductibles ce qui permet d'atteindre des polynômes de degré 30. Elle a besoin de beaucoup moins de bits de sortie que la méthode de recherche du polynôme de rétroaction de M.Cluzeau. De plus, pour de petits biais, lorsque l'on connaît la longueur du registre ou des informations sur le poids du polynôme de rétroaction utilisé, elle est aussi, voire plus efficace que la méthode générique de M.Cluzeau. Enfin, pour des polynômes irréductibles et non primitifs particuliers, la méthode de M.Cluzeau est incapable de retrouver le polynôme de rétroaction, et seule la recherche exhaustive permet de le faire efficacement. Il faudrait étudier la structure de ces polynômes pour comprendre pourquoi l'algorithme de M.Cluzeau ne les retrouve pas. Pour optimiser d'autant plus cette recherche des paramètres d'un brasseur synchrone, on pourrait imaginer un algorithme hybride qui parcourt tous les polynômes primitifs ou irréductibles possibles, et qui, pour chacun d'eux, calcule leurs multiples creux et essaye de décoder la suite grâce à l'algorithme de décodage des codes LDPC.

Dans le cas du brasseur auto-synchronisant, nous avons étudié l'algorithme proposé par M.Cluzeau, qui permet de reconstruire efficacement les brasseurs dont le polynôme de rétroaction est de poids relativement faible (ce qui est souvent le cas dans les applications pratiques). Mais le temps de calcul de cet algorithme augmente exponentiellement avec le poids du polynôme de rétroaction. Une des pistes pour avoir un algorithme dont la complexité n'est pas reliée au poids, serait de pouvoir détecter les multiples creux du polynôme de rétroaction lorsque l'on débrasse avec ceux-ci.

Bibliographie

[1] Mathieu Cluzeau *Reconnaissance d'un schéma de codage*
Thèse de doctorat, École Polytechnique, 2006.
(http://www-rocq.inria.fr/codes/Mathieu.Cluzeau/pdfs/these_cluzeau.pdf)

[2] Elwin R. Berlekamp *Algebraic Coding Theory* 1968

[3] A. Menezes, I. Blake, X. Gao, R. Mullin, S. Vanstone, T. Yaghoobian *Applications of Finite Fields* 1993
(http://www.springer.com/engineering/electronics/book/978-1-4419-5130-4)

[4] J. von zur Gathen, J. Gerhard *Modern Computer Algebra* Cambridge University Press, New York, 1999.
(http://cosec.bit.uni-bonn.de/science/mca/)

[5] T. Moon *Error Correction Coding* 2005
(http://www.eyrolles.com/Sciences/Livre/error-correction-coding-9780471648000?PHPSESSID=)

[6] Anne Canteaut, Michael Trabbia *Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5* Advances in Cryptology - Eurocrypt 2000
(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.6761)

[7] Robert Gallager *Low-Density Parity-Check Codes* IRE Transactions on Information Theory, 1962
(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.143.2744)

[8] Anne Canteaut, Eric Filiol *Ciphertext Only Reconstruction of Stream Ciphers based on Combination Generators* Fast Software Encryption 2000
(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.2343)

[9] A. Di Porto, F. Guida, E. Montolivo *Fast Algorithm For Finding Primitive Polynomials Over GF(q)* 1992
(http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=118917)

[10] Rudolf Lidl et Harald Niederreiter *Finite Fields* Encyclopedia of Mathematics and its applications, Volume 20, Cambridge University Press, 1997

[11] Gilles Zémor *Master CSI, Arithmétique 1 : corps finis et applications* 2006
(http://www.math.u-bordeaux.fr/~zemor/arit06.pdf)