ECOLE NATIONALE SUPERIEURE DE TECHNIQUES AVANCEES
DEA ALGORITHMIQUE

Rapport de Stage de DEA

Application de techniques de décodage à la cryptanalyse de systèmes de chiffrement

Racem TRIKI

encadré par

Anne Canteaut et Jean-Pierre Tillich

Année 2004

INRIA
ROCQUENCOURT

Remerciements

Au terme de ce travail, je tiens à remercier Jean-Pierre Tillich et Anne Canteaut qui ont dirigé ce travail. La clarté de leurs explications, leurs idées et leur persévérance m'ont beaucoup aidé : j'espère que leurs qualités m'auront imprégné et qu'ils garderont un bon souvenir de ce travail accompli ensemble.

Des remerciements tout particuliers à Nicolas Sendrier qui m'a accepté au sein de son équipe.

Mes plus sincères remerciements s'adressent aussi à l'ensemble des membres du projet *CODES* qui m'ont chaleureusement accueilli et n'ont à aucun moment hésité à m'encadrer et à m'orienter.

Table des matières

Chapitre 1

Introduction générale

Ce projet se situe dans le cadre de l'adaptation des techniques de décodage de codes correcteurs à la cryptanalyse de systèmes de chiffrement à clé secrète.

En effet, plusieurs techniques de décodage ont montré de bonnes performances dans la protection d'information à l'égard d'erreurs de transmission mais elles n'ont pas encore été utilisées en cryptanalyse. Les deux techniques qui seront étudiées dans ce projet sont celles de décodage souple par ensemble d'information appliqué à un canal gaussien et quelques techniques avancées de décodage itératif (on expliquera à la fin de ce chapitre la raison et l'intérêt de ce choix). Avant d'aller plus loin nous nous proposons dans cette introduction d'éclaircir les lecteurs non initiés sur certains points de cryptographie et de codes correcteurs d'erreurs nécessaires à la compréhension de la suite du rapport.

1.1 Introduction au chiffrement

Le chiffrement est une activité cryptographique très ancienne qui remonte à l'antiquité (6ème siècle avant J.C) et qui consiste à coder (c'est le terme commun, mais en cryptographie, on dit plutôt "chiffrer" que coder) un message pour le rendre incompréhensible pour quiconque n'est pas doté d'une clé de déchiffrement K_D qui doit donc impérativement être gardée secrète.

Jusqu'au milieu des années 70, les seuls cryptosystèmes connus étaient *symétriques* (on dit aussi *conventionnels* ou à *clé secrète*) : la clé de chiffrement K_C était la même que la clé de déchiffrement K_D (ou du moins , la connaissance de la clé de chiffrement permettait d'en déduire la clé de déchiffrement) ce qui obligeait à garder secrète la clé K_C elle aussi. En 1976, le concept de cryptographie à *clé publique* (ou *asymétrique* fut introduit pour la première fois : dans ce type de système, chaque personne reçoit un couple de clés, dont l'une publique *i.e.* connue de tous, l'autre reste secrète et propre à chaque utilisateur. La clé publique est liée mathématiquement à la clé privée.
Mais cela n'a pas pour autant sonné la fin de la cryptographie conventionnelle car celle-ci est beaucoup plus rapide à mettre en œuvre (entre 100 et 1000 fois plus rapide selon les cryptosystèmes) et ses clés sont en général plus courtes pour un même niveau de sécurité.
Dans ce qui suit nous nous préoccupons uniquement de cryptographie symétrique.

1.2 Le chiffrement à clé secrète

Il existe deux sous-catégories d'algorithmes à clé secrète : les algorithmes de chiffrement par bloc, comme le DES, l'AES, ou les algorithmes de chiffrement à flot continu, auxquels nous nous intéressons dans toute la suite de ce document. Plus précisément, les algorithmes de chiffrement par bloc segmentent les données à chiffrer en blocs de taille donnée ; ils consistent généralement à effectuer, à partir d'un bloc de message (de 64 ou 128 bits) et d'une clé secrète, une suite d'opérations de chiffrement qui se déroulent sur plusieurs étages, appelés "rounds".
A l'inverse, les schémas de chiffrement par flot traitent l'information bit à bit, en ajoutant à chaque bit un bit d'une séquence de clé. Cet algorithme qui est désigné plus explicitement en anglais par l'expression *one-time pad* est l'un des rares algorithmes qui assure une sécurité parfaite, *i.e.* qu'il est théoriquement impossible de retrouver le message clair à partir du chiffré sans connaître la clé secrète, à condition que celle-ci soit une suite complètement aléatoire. Cependant, comme il n'est en général pas envisageable de partager une clé secrète qui soit aussi longue que le message à chiffrer, on utilise dans la pratique une clé pseudo-aléatoire, c'est à dire une clé qui ne soit pas choisie aléatoirement parmi tous les mots binaires de longueur n. Cette clé (qu'on appellera *suite pseudo-aléatoire*) est générée par différents procédés à partir d'une clé aléatoire d'une longueur juste suffisante pour résister aux attaques exhaustives (il faut prendre au moins 80 bits). Plus précisément la sécurité d'un tel système contre une attaque à clair connu est équivalente à la sécurité du générateur pseudo-aléatoire. On demande bien entendu que la connaissance de N bits consécutifs de la clé ne permette pas de calculer facilement la clé aléatoire.
En plus de leur sûreté, ces schémas sont très rapides et ils peuvent être traités avec une mémoire limitée et la propagation des erreurs de transmission du chiffré au clair est limitée (puisque le message est chiffré bit à bit).
Pour réaliser ces schémas en pratique, l'idée naturelle fut d'opter pour des générateurs produisant des séquences pseudo-aléatoires de période la plus élevée possible. Le procédé électronique pour générer de telles séquences est le registre à décalage linéaire qui va être décrit en détail dans le chapitre suivant(*linear Feedback shift Register* en anglais ; on écrira en abrégé LFSR).

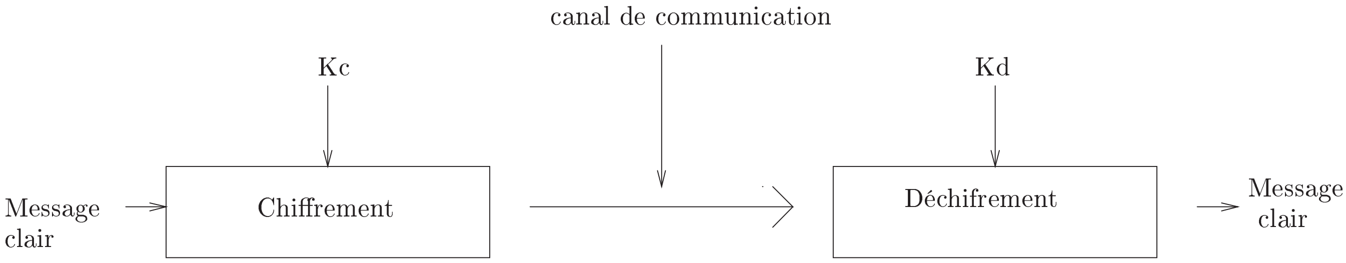


FIG. 1.1 – **Chiffrement et Déchiffrement**

1.3 Les différents types d'attaques

L'objectif de l'attaquant c'est de trouver la clé, pour cela il a à sa disposition différents méthodes d'attaques :

- L'attaque à chiffré seul : l'attaquant ne connaît que le chiffré et il cherche le clair et si possible la clé.
- L'attaque à clair connu : l'attaquant connaît un ou plusieurs couples clairs-chiffrés, et il cherche la clé.
- L'attaque à clair choisi (ou chiffré choisi), semblable à la précédente, mais pour laquelle l'attaquant peut choisir les clairs de façon que son attaque réussisse avec meilleure probabilité.

Dans ce rapport, on s'intéresse à la cryptanalyse à clair connu du modèle d'un système cryptographique qu'on explicitera dans le chapitre suivant.

1.4 Codes correcteurs d'erreurs

1.4.1 Introduction

La qualité de réception d'un message numérique est complètement définie par la probabilité d'erreur par bit, pour améliorer cette qualité une alternative possible consiste à introduire un certain taux de redondance dans le message numérique émis de façon à le rendre moins sensible aux erreurs élémentaires. Ce principe est connu sous le nom : *Codage correcteur d'erreur*. La redondance introduite par le codage se traduit par l'introduction d'une "corrélation" entre certains éléments binaires originellement totalement indépendants. Ceci permet de détecter les erreurs de transmission et, dans certains cas, de les corriger, dans la limite des possibilités du code et de la qualité de la liaison qui restent tout de même la base des performances globales.
Les deux grandes familles de code sont : les codes en bloc auxquels on s'intéresse dans la suite et les codes convolutifs. Le choix de code prend en compte la distribution des erreurs de transmission dans le temps. Nous étudierons dans ce rapport les modèles de canaux classiques que sont le canal binaire symétrique (noté CBS) et le canal gaussien (à Bruit Blanc Additif Gaussien).

1.4.2 Codes en blocs

Un code en bloc regroupe des symboles q-aires de k éléments appelés messages et associe à chacun de ces messages un n-uplet appelé mot de code. Un code en blocs est donc un sous-ensemble de q^k séquence q-aires de longueur n appelées mots de code. Dans notre cas on utilisera un code binaire : $m = 1$ (soit $q = 2$).
Le nombre n de bits qui composent un mot du code est appelé la longueur du code. La dimension k étant la longueur initiale des mots. Le rendement ou taux de codage est défini par $R = k/n$.
Nous nous intéressons maintenant aux codes en blocs linéaires construits sur le corps Galois $GF(2^m)$. Un code linéaire $\mathcal{C}(k, n)$ est un sous-espace vectoriel de dimension k de $GF(2^m)$. Les vecteurs définissant une base de ce sous-espace $\mathcal{C}$ sont utilisés pour former

une matrice G, de taille $k \times n$. La matrice G est appelée matrice génératrice du code $\mathcal{C}$. tout mot de code est une combinaison linéaire de lignes de G.
Le code $\mathcal{C}'$ dont la matrice génératrice est obtenue en permutant des colonnes et en additionnant des lignes de la matrice G est dit équivalent au code $\mathcal{C}$. Par permutations des colonnes et opérations sur les lignes, on montre donc que le code $\mathcal{C}$ est équivalent à un code $\mathcal{C}'$ ayant une matrice génératrice de la forme

$$G = [I : P]$$

où I est la matrice identité $k \times k$ et P une matrice de taille $k \times (n-k)$. Cette forme de matrice génératrice s'appelle forme systématique, avec une telle forme les k premiers symboles d'un mot de code sont les symboles d'information non modifiés. Les $(n-k)$ symboles qui restent sont appelés symboles de parité.
Après réception du mot émis, l'algorithme de décodage aura alors comme objectif de retrouver le mot de code transmis. Dans ce rapport on s'intéressera essentiellement à deux algorithmes de décodage, le premier s'appuie sur les algorithmes de décodages de Gallager, et le deuxième qu'on explicitera plus en détail dans la deuxième partie du rapport et qui est un mélange de décodage dur et souple décrit dans la thèse d'A.Valembois [2].

1.5 Contexte général et motivation

On se demande après cette brève introduction sur les systèmes de chiffrement à clé secrète puis sur les codes correcteurs d'erreurs, quel est le lien qui relie ces deux domaines. En effet, dans le cadre de ce stage, on s'intéresse à la cryptanalyse de système de chiffrement à flot et en particulier aux systèmes de chiffrement par flot fondé sur des registres filtrés.
Une des méthodes de cryptanalyse considère non pas le système tel quel mais un système de codage équivalent(on l'explicitera en détail dans le troisième chapitre), ce qui ramène l´étude de la cryptanalyse des registres filtrés à un problème de décodage de codes correcteurs d'erreurs. Mais, en analysant les résultats déjà évoqués dans la littérature, on se rend compte que ce type de cryptanalyse à base de décodage itératif dépend d'un seuil au delà duquel la cryptanalyse échoue, ce seuil est la probabilité de transition du canal de transmission introduit par la modélisation (il est lié au choix de la fonction booléenne). La question qu'on s'est posée alors est : comment peut on aller plus loin que ce seuil ?
La solution que nous avons essayé de développer tout au long de ce stage consistait alors à essayer d'exploiter le mieux l'information générée par le décodage itératif. En effet, la sortie de ce décodeur est une information souple mais on était pas sûr qu'elle soit exploitable. Pour cela, notre travail s'est structuré de la façon suivante : d'abord, l'étude du décodage itératif et la sélection de l'algorithme de Gallager qui donne la meilleure information souple (meilleure dans le sens pouvant être exploitable par un nouveau décodeur). Une fois l'information souple obtenue, la deuxième étape du travail consistait alors à employer des techniques de décodage souple appliquées à un canal

gaussien. On vérifie en effet que la distribution des erreurs à la sortie du décodeur itératif est une gaussienne.

Présentation du rapport

Après ce premier chapitre introductif, le deuxième chapitre comportera un rappel sur les registres à décalage à rétroaction linéaire et quelques notions sur les fonctions booléennes scalaires, tout en insistant sur les hypothèses que doivent satisfaire ces deux outils pour la suite de notre étude. Ensuite le troisième chapitre est un état de l'art en ce qui concerne les attaques des systèmes à base de registres filtrés. J'ai ensuite étudié le décodage itératif et comparé les résultats du décodage utilisant différentes variantes de l' algorithme de Gallager. Enfin, dans le dernier chapitre je me suis intéressé au décodage souple par ensemble d'information appliqué à un canal gaussien afin de l'appliquer à la sortie du décodage itératif.

Chapitre 2

Introduction sur les registres filtrés

2.1 Introduction

On a vu dans le chapitre précédent que l'un des procédés électroniques pour générer des clés pseudo-aléatoires ou plus précisément des suites à récurrence linéaire est le registre à décalage linéaire (*Linear Feedback Shift Register*). Ce LFSR sera suivi d'une fonction booléenne qui doit être choisie pour résister à certaines attaques (l'attaque par corrélation, attaque linéaire). Ceci nous amène à s'intéresser le long de ce chapitre à faire des rappels sur ces deux notions tout en insistant sur les hypothèses qu'on utilisera pour le système cryptographique étudié.

2.2 Rappel sur les registres à décalage à rétroaction linéaire

Un registre à décalage à rétroaction linéaire ($LFSR$) de longueur L est composé d'un registre de L bits $(s_i, \ldots, s_{i+L-1})$ et d'une fonction de rétroaction linéaire.

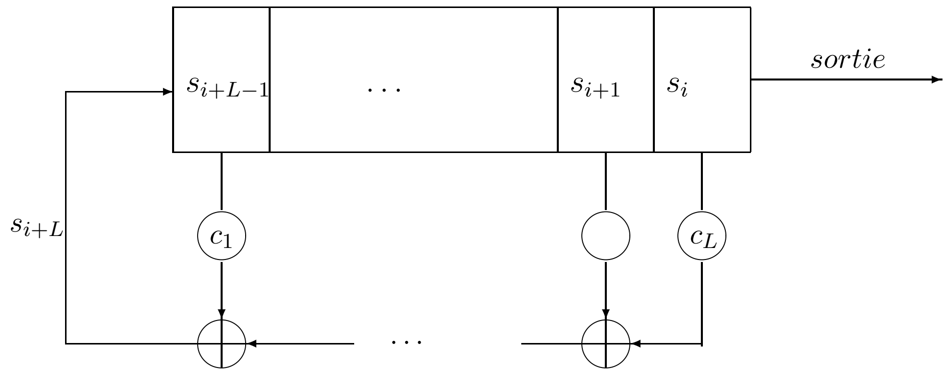


Fig. 2.1 – Fonctionnement d'un $LFSR$

A chaque top d'horloge, le bit s_i constitue la sortie et les autres bits sont décalés vers la droite. Le nouveau bit s_{i+L} placé dans la cellule de gauche du registre est donné par une fonction linéaire des bits $(s_i, \ldots, s_{i+L-1})$:

$$s_{i+L} = c_1 s_{i+L-1} + c_2 s_{i+L-2} + \cdots + c_L s_i \tag{2.1}$$

où les coefficients de rétroaction $(c_i)_{1 \leq i \leq L}$ sont des éléments de $\mathbb{F}_2$.
Les bits $(s_0, \ldots, s_{L-1})$, qui déterminent entièrement la suite, constituent *l'état initial du registre.*
Les coefficients de rétroaction sont usuellement représentés par un polynôme de $\mathbb{F}_2[X]$ de degré L, appelé *polynôme de rétroaction du registre* :

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

Enfin on doit initialiser $2L$ bits en tout pour composer un registre : les L bits coefficients de rétroaction, et les L coefficients s_i qui constituent l'état initial du registre.

Proposition 1 *Supposons fixés les coefficients de récurrence $(c_1, c_2, ..., c_L)$. Soit $s = (s_n)_{n \geq 0}$ une suite binaire vérifiant la relation de récurrence 2.1. La donnée des L premières valeurs $(s_0, s_1, ..., s_{L-1})$ détermine de façon unique la suite $s = (s_n)_{n \geq 0}$. Plus précisément, l'application qui à la suite s associe ses L premiers termes est un isomorphisme de $\mathbb{F}_2$-espaces vectoriels de l'espace des suites récurrentes pour les coefficients $(c_1, c_2, ..., c_L)$ sur $\mathbb{F}_2^L$.*

Nous allons maintenant étudier la périodicité d'une telle suite :

Proposition 2 *Dans le cas général la suite s est ultimement périodique, de période $T \leq 2^L - 1$ (i.e. il existe un entier i_0 tel que $s_i = s_{i+T}$ pour tout $i \geq i_0$). Si, de plus, $c_L = 1$, la suite s est périodique (i.e. $s_i = s_{i+T}$ pour tout $i \geq 0$).*

On supposera désormais que $c_L = 1$. Clairement, pour des applications cryptographiques, il est souhaitable que la suite s ait une périodicité aussi longue que possible, c'est à dire, d'après ce qui précède, une période $T = 2^L - 1$. Remarquons que, dans ce cas, on parcourt tous les états possibles de $\mathbb{F}_2^L$ sauf 0.

Définition 1 *Si la suite s (avec $c_L = 1$) a pour période $2^L - 1$, on dit que s est une m-suite ou m-séquence, ou une séquence* ***PN*** *ou encore qu'elle est de longueur maximale.*

En terme de polynôme de rétroaction, on utilisera dans notre cas des polynômes primitifs, ce qui assure que la période de la suite engendrée est maximale et donc de se retrouver dans la champ d'applications cryptographiques.

Citons, avant de passer à la définition de la complexité linéaire, quelques propriétés de la séquence qui nous seront très utiles dans la suite du rapport :
- Toute combinaison linéaire de décalées d'une séquence **PN** est encore une séquence **PN** qui satisfait la même récurrence.
- Décoder une combinaison linéaire de la séquence initiale nous permet facilement de remonter à la séquence initiale, puisqu'elle-même s'exprime comme une combinaison linéaire de la séquence décodée.
- Tout bit de la séquence s s'exprime comme une combinaison linéaire des L premiers bits $(s_0, s_1, ..., s_{L-1})$. Ainsi, décoder L bits quelconques linéairement indépendants nous permet d'identifier $(s_0, s_1, ..., s_{L-1})$ en inversant un système linéaire $(L * L)$.

Maintenant on va parler d'un paramètre intervenant dans plusieurs applications cryptographiques : **la complexité linéaire**

Définition 2 *La complexité linéaire d'une suite périodique s est la longueur du plus petit LFSR qui l'engendre.*

En effet la complexité linéaire d'un LFSR est donc un paramètre déterminant pour sa sécurité cryptographique : l'observation d'un petit nombre de bits seulement de la suite permet de la reconstituer entièrement ce qui s'avère évidemment catastrophique. Aussi s'atache-t-on généralement à utiliser des LFSR dont la complexité linéaire est égale à la longueur.

2.2.1 Résumé

Un LFSR de longueur maximale semble être un bon candidat pour un générateur pseudo-aléatoire. En effet, sa période est grande, et sa complexité linéaire est bien égale à L. Il a de bonnes propriétés statistiques : chaque L-uplet non nul de nombres binaires apparaît comme un registre une et une seule fois par période (les registres prennent tous les états possibles de $\mathbb{F}_2^L$ sauf 0). Malheureusement il n'en est rien à cause de la proposition suivante, qui montre que connaissant L bits consécutifs de la suite qu'il engendre, on peut calculer aisément les coefficients de rétroaction.

Proposition 3 *Soit s une suite récurrente de longueur maximale, de complexité linéaire* L*. Si on connaît* $2L$ *termes consécutifs de cette suite, alors on peut calculer les coefficients de récurrences en inversant un système linéaire de taille* $L * L$*.*

2.2.2 Conclusion

On a vu que les LFSR ne sont pas cryptographiquement sûrs à cause de leur structure linéaire. Une solution pour augmenter la complexité linéaire consiste à combiner la sortie de n LFSRs en parallèle par une fonction de combinaison f (qu'on prendra booléenne en général) :

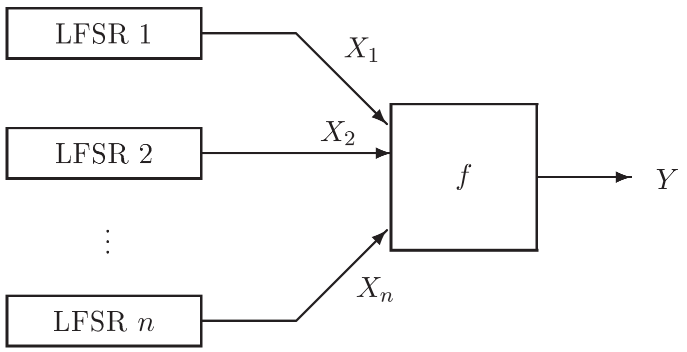


FIG. 2.2 – **Combinaison de plusieurs LFSR**

On utilise n LFSR qui à chaque top d'horloge, sortent chacun un bit x_i. Et les bits $x_1, x_2, ..., x_k$ constituent l'entrée d'une fonction booléenne qui sort un bit à chaque top d'horloge. Les polynômes de rétroaction des LFSR sont en général publics, ainsi que la fonction booléenne. Seules les initialisations des LFSR sont secrètes et constituent la clé (ce qui était impossible en utilisant un simple LFSR, puisqu'on a vu que si le polynôme de rétroaction est connu, alors en observant L bits consécutifs, on peut calculer tous les suivants).

Remarque : LFSR filtrés

On a opté pour un seul LFSR et à chaque top, l'état du registre (la suite des bits qu'il contient) ou un état partiel (une sous-suite) est l'entrée d'une fonction booléenne. On montre que ce système est équivalent au précédent : en effet, il revient à combiner n LFSR de même polynôme de rétroaction, dont les registres initiaux sont les n premiers registres de celui-ci ; mais les attaques sur ce système ne fonctionnent pas avec la même complexité que sur le système par fonction de combinaison qui lui est équivalent. Dans notre cas on fait suivre le registre d'une combinaison linéaire de bits qui choisit les bits qui passeront dans la fonction booléenne (voir figure suivante : les indices $t + \gamma_i$ définissent l'entrée de la fonction f).

$$\forall t > 0, \quad s_t = f(u_{t+\gamma_1}, u_{t+\gamma_2}, ..., u_{t+\gamma_n})$$

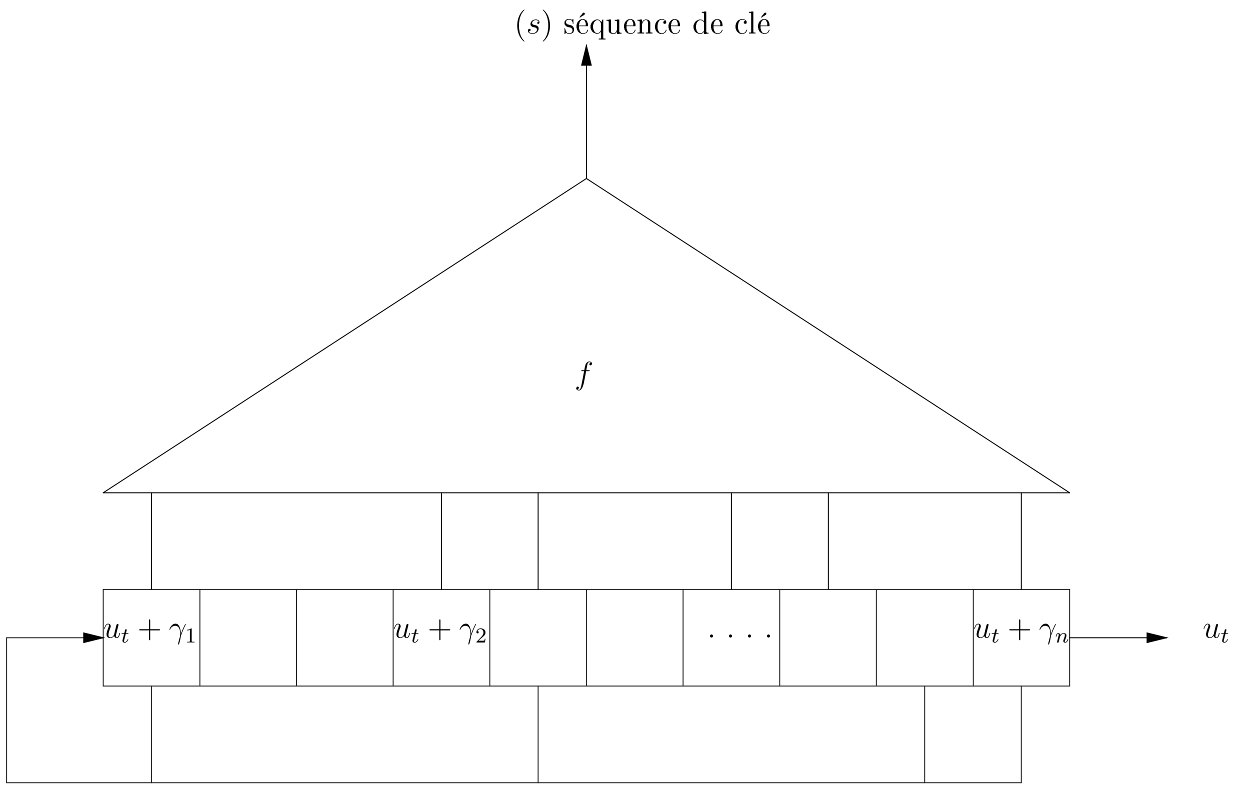


FIG. 2.3 – Registre filtré

2.3 Rappels sur les fonctions booléennes

Dans le but d'utiliser les registres à décalage combinées avec des fonctions boolénnes, on se propose de voir dans ce paragraphe quelques notions sur de telles fonctions. En effet, une fonction boolénne doit être choisie pour résister à certaines attaques (l'attaque par corrélation, attaque linéaire). Les différents critères pour les fonctions booléennes (balancement, non corrélation, non-linéarité, degré algébrique) ont été étudiés dans beaucoup de travaux. Il a été montré que certains critères ne peuvent être satisfaits simultanément. A la fin de ce paragraphe on évoquera les critères que doit satisfaire notre fonction booléenne pour assurer une bonne sécurité du cryptosystème fondé sur les registres filtrés.

Définition 3 *Soit $V_n = \mathbb{F}_2^n$. On appelle fonction booléenne sur V_n toute application de V_n dans $\mathbb{F}_2$.*

- une fonction booléenne f est dite **linéaire** si

$$\exists u \in \{0,1\}^n \; tel\, que \forall x \in \{0,1\}^n, \;\; f(x) = u.x$$

- Soit $V_n = \mathbb{F}_2^n$. On appelle fonction booléenne sur V_n toute application de V_n dans $\mathbb{F}_2$.
- Le poids $W(f)$ d'une fonction f sur V_n est le nombre de vecteurs v sur V_n tel que $f(v) = 1$.
- Une fonction est dite **balancée** ou **équilibrée** si $W(f) = W(f \oplus 1)$ (ou bien $W(f) = 2^{n-1}$ i.e. f prend autant de fois la valeur 0 que la valeur 1.
- On appelle table des valeurs de la fonction, la table qui recense pour chacun des 2^n vecteurs d'entrée possibles, les valeurs prises par la fonction.

Il est bien connu qu'une fonction f sur V_n peut être représentée d'une façon unique par un polynôme sur $\mathbb{F}_2$ de degré au plus n :

$$f(x_1, x_2, ..., x_n) = \sum_{u \in V_n} a_u (\prod_{i=1}^{n} x_i^{u_i})$$

où, pour tout u, le coefficient a_u est dans $\mathbb{F}_2$.

Cette écriture, unique, correspond à la représentation de la fonction booléenne comme fonction polynomiale à n variables dont le degré relatif à chacune des variables est au plus 1 (d'où le nom de normale) est appelée **la forme algébrique normale**. Le **degré algébrique** de f, noté $deg(f)$ ou $ord(f)$ est définie comme étant le nombre de variables dans le plus long terme de f, par exemple les fonctions booléennes linéaires sont de degré 1.

Définition 4 *La* **distance de Hamming** *$d(v_1, v_2)$ entre deux vecteurs v_1 et v_2 est le nombre de composant où les vecteurs v_1 et v_2 diffèrent. Par extension, pour deux fonctions f_1 et f_2 sur V_n, on définit leur distance par*

$$d(f_1, f_2) = \sharp\{v \in V_n | f_1(v) \neq f_2(v)\}$$

La distance minimale entre f et l'ensemble de toutes les fonctions affines (c'est l'ensemble des fonctions linéaires et leurs complentaires) est appelée la **non-linéarité** de f.

Définition 5 *La* **corrélation** *entre deux fonctions booléennes f et g est définie par :*

$$c(f,g) = 1/2^n(\sharp\{v \in V_n | f(v) \neq g(v)\} - \sharp\{v \in V_n | f(v) \neq g(v)\})$$

Une fonction booléenne f sur V_n est dite **sans corrélation** d'ordre m, avec $1 \leqslant m \leqslant n$, si la sortie de f et n'importe quelle entrée à m variables sont statistiquement indépendant. Ce concept a été introduit par Siegenthaler [4] en décrivant l'une des plus importantes attaques : **l'attaque par corrélation**. Elle concerne le cas où la sortie du système est corrélé à l'un des LFSR le composant, ou à une combinaison de ceux-ci.

Définition 6 *une fonction booléenne f est dite* **résiliente d'ordre m** *si et seulement si f est sans corrélations d'ordre m et f est équilibrée.*

2.3.1 Critères de choix de la fonction booléenne

Une fonction booléenne utilisée comme fonction de combinaison doit être équilibrée, car sinon il existe une attaque sur le système appelée attaque statistique. Enfin comme on l'a déjà évoqué dans les paragraphes précédents on cherche à avoir une complexité linéaire élevée du système étudié. On démontre que la complexité linéaire d'une fonction $f(x_1, x_2, ..., x_n) = \sum_{u \in V_n} a_u (\prod_{i=1}^{n} x_i^{u_i})$ prenant en entrée des suites générées par des LFSR de longueurs $L_1, L_2, ..., L_n$, *i.e* $\sum_{u \in V_n} a_u (\prod_{i=1}^{n} L_i^{u_i})$, où le calcul de la somme est effectué dans $\mathbb{R}$ et non modulo 2. On voit bien qu'il faut utiliser des fonctions de haut degré pour atteindre de hautes complexités linéaires.
En utilisant la notion de transformée de Walsh de f, qui est une transformée de Fourier, on montre que, si f est résiliente d'ordre m, alors le degré de f est au plus égal à $n - m$ (si on la choisit en plus équilibrée *i.e* résiliente d'ordre m, le degré de f sera au plus $n - m - 1$). Cette propriété est donc contradictoire avec la recherche de fonctions de complexité linéaire élevé.

Remarque

L'intérêt du **caractère résilient** dans le cas d'un unique registre filtré n'est, à première vue, pas évident contrairement au cas des registres par combinaison, mais nous montrerons cependant dans le chapitre suivant que ces fonctions présentent une meilleure résistance face aux attaques connues. Dans le chapitre suivant on fera un état de l'art sur la cryptanalyse des registres filtrés avant de passer au chapitre d'après à l'attaque qu'on a opté pour.

Chapitre 3

Cryptanalyse des registres filtrés

3.1 Introduction

Après ce bref rappel sur les registres à décalage à rétroaction linéaire et sur les fonctions booléennes, on va s'intéresser, dans toute la suite de ce rapport, au vif du sujet, à savoir l'application de techniques de décodage des codes correcteurs d'erreurs à la cryptanalyse des registres filtrés. Pour cela on se propose, d'expliquer l'idée de modélisation de Meier et Staffelbach qui ont été les premiers à cryptanalyser le système du registre filtré en utilisant un algorithme inspiré du décodage itératif de codes correcteurs. Mais avant cela on va s'intéresser dans un premier temps au principe des attaques par corrélation et à leurs limites de performances qui ont poussé Meier et Staffelbach à proposer *l'attaque par corrélation rapide* se fondant sur les codes correcteurs d'erreurs.

3.2 Attaques par corrélation

Les méthodes de cryptanalyse présentées dans ce chapitre sont des cryptanalyses à clair connu : nous supposons avoir intercepté une longueur N de bits de la séquence de bits de clé (en ajoutant par exemple le message clair connu à la séquence de chiffré interceptée), et l'objectif est alors de déduire l'initialisation du registre à décalage.
Une attaque par corrélation exploite l'existence d'une éventuelle corrélation entre la sortie de la fonction de combinaison f et l'une de ses entrées, ces attaques ont été introduites par Siegenthaler en 1985 [4], et elles se basent sur le principe *divide and conquer* : elle repose sur l'existence d'une corrélation entre la sortie s_k de la fonction de combinaison et la sortie x_k^i du i^{me} LFSR. L'attaque consiste à tester toutes les valeurs d'initialisation possibles pour le LFSR i, et pour chacune de ces valeurs d'initialisation, de calculer la corrélation entre la séquence pseudo-aléatoire s_k et la suite générée par ce LFSR. L'observation d'un pic de corrélation permet alors de détecter l'initialisation effectivement utilisée.

Complexité

L'intérêt de cette attaque est que l'initialisation de chacun des LFSR est inférée indépendamment des autres. Cela permet une réduction considérable de la complexité numérique de l'attaque par rapport à une recherche exhaustive. Ainsi, s'il existe une corrélation entre la sortie de chacun des LFSR et la sortie de la fonction de combinaison, la complexité de l'attaque est de l'ordre de :

$$\sum_{i=1}^{N}(2^{L_i} - 1)$$

où N est le nombre de LFSR couplés et L_i est le degré du LFSR i. A titre de comparaison, la complexité numérique d'une attaque brutale par recherche exhaustive serait de l'ordre de :

$$\prod_{i=1}^{N}(2_i^L - 1)$$

toutefois, la complexité de cet algorithme est exponentielle en la longueur du registre à décalage examiné. La cryptanalyse devient donc hors de portée dès lors que la taille des registres dépasse 50 ou 60.

3.3 Attaque par corrélation rapide

Pour éviter de passer en revue toutes les initialisations, W.Meier et O.Staffelbach [3] ont cryptanalysé le système du registre filtré en assimilant la recherche de l'initialisation d'un registre à un problème de correction d'erreur.

3.3.1 Modélisation du système

Les k bits d'initialisation du registre sont vus comme les k bits d'information d'un code (n, k), et la fonction booléenne est supposée se comporter comme un canal binaire symétrique, dont la probabilité de transition est évaluée en fonction du spectre de la fonction. Enfin la séquence (s) en sortie de la fonction est alors perçue comme une version bruitée du mot de code émis, la probabilité d'erreur sur chacun des bits de cette séquence étant la probabilité de transition du canal. Comme la suite engendrée par le LFSR est fortement redondante (suite engendrée par une relation de récurrence linéaire), on peut la corriger. Le but du reste du rapport sera alors d'exposer les algorithmes de décodage utilisés pour reconstituer ce mot de code.

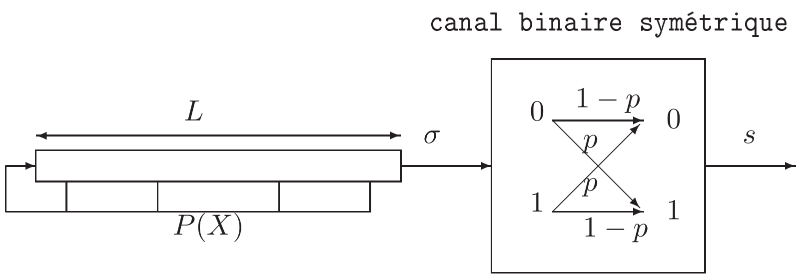


Fig. 3.1 – **Modèle de l'attaque**

La première étape de ce type d'attaque "probabiliste" consiste à trouver la probabilité qu'une erreur ait lieu, qui est évaluée à l'aide de la connaissance de la fonction booléenne (lorsque elle est équilibrée) :

$$p = \frac{1}{2^n} \times max_{u \in \{0,1\}^n} |\widehat{f}(u)| + 1/2 \tag{3.1}$$

où $\widehat{f}$ désigne la transformée de Fourier de f :

$$\forall u \in \{0,1\}^n, \widehat{f}(u) = \sum_{x \in \{0,1\}^n} f(x)(-1)^{u.x}$$

avec :

$$u.x = \sum_i u_i x_i$$

Détaillons le calcul de la probabilité de transition du canal binaire symétrique équivalent :
$\forall z_t$, $\exists X(t)$ (l'entrée de f à l'instant t) tel que $z_t = f(X(t))$. En appliquant la définition de la probabilité conditionnelle, on obtient :

$$\begin{aligned} P[f(X) = 1 | u.X = 0] &= \frac{P[f(X) = 1, u.X = 0]}{P[u.x = 0]} \\ &= 2 \times \frac{1}{2^n} \sum_X f(X) \frac{1 + (-1)^{u.X}}{2} \end{aligned}$$

ce qui implique :

$$\begin{aligned} P[f(X) = 1 | u.X = 0] &= \frac{1}{2^n} [\widehat{f}(0_n) + \widehat{f}(u)] \\ P[f(X) = 1 | u.X = 1] &= \frac{1}{2^n} [2^n - \widehat{f}(0_n) + \widehat{f}(u)] \end{aligned}$$

Ce qui justifie entre autres la nécessité d'avoir une fonction équilibrée pour que le canal équivalent soit symétrique (nécessité pour le cas de notre cryptanalyse), sinon si ce n'est pas le cas la cryptanalyse sera plus facile à effectuer avec d'autres techniques.

3.3.2 Cas des registres filtrés

Si le modèle déjà présenté au paragraphe précèdent s'avére justifié dans le cas d'un registre à combinaison (plusieurs registres éventuellement différents constituant chacun une entrée de la fonction booléenne, cf figure 2.2 page 9), il l'est moins dans le cas d'un unique registre filtré qui est intrinsèquement sujet à un effet de mémoire dont le modèle du BSC fait totalement abstraction.

D'abord, on rappelle que dans le cas des registres filtrés on s'intéresse à la séquence issue de combinaisons linéaires des bits d'entrée de la fonction plutôt qu'à la séquence telle quelle. La structure linéaire du système implique que toute forme linéaire de la séquence d'entrée vérifie exactement les mêmes équations de parité que la séquence

de départ. Une fois la séquence décodée obtenue, on peut retrouver (sachant la forme linéaire et le polynôme de rétroaction) facilement la séquence de départ.
En ce qui concerne la probabilité de transition équivalente à celle d'un BSC, on obtient le même résultat qu'avec les calculs qui ont été effectués au dessus. Cependant il faut faire attention au choix de la fonction booléenne car cette probabilité de transition suivant l'équation (3.1) n'est optimale que dans le cas d'une fonction résiliente et en plus dans ce cas précis : les espacements (liés aux choix des bits à l'entrée de la fonction f) doivent être tous premiers entre eux. Dans tous les autres cas, il faudra examiner les différents probabilités ostensibles afin de faire le choix optimal.

Conclusion

Une fois le système cryptographique défini et les paramètres de la modélisation (probabilité de transition du canal binaire symétrique), on ne tient plus compte du registre filtré et on essaye de cryptanalyser en décodant un code correcteurs d'erreurs. On commencera alors notre étude par l'exploration des techniques de décodage itératifs ce qui fera l'objet du prochain chapitre dont lequel on essayera de choisir la meilleure variante des algorithmes de Gallager qui nous donnera de l'information souple exploitable par un décodeur souple (ce qui fera l'objet du dernier chapitre).

Chapitre 4

Décodage itératif et registre filtré

4.1 Introduction :

Le principe essentiel de l'attaque par corrélation rapide, introduit dans le chapitre précédent, est d'assimiler la recherche de l'initialisation d'un registre à un problème de correction d'erreurs. Ceci nous amène à se consacrer entièrement dans ce chapitre à l'étude des performances des algorithmes de décodage, et plus précisément le décodage itératif. Vous vous demandez sûrement à quoi ça sert d'étudier en détails toutes les versions des algorithmes de Gallager ? En fait, le but de notre étude ne sera pas les performances de point de vue taux d'erreurs et complexité(en temps et en espace), mais ça sera plutôt l'information souple fournie par ce type de décodage pour voir si elle peut être exploitable ou non. En effet, les résultats sont très différents d'une version à une autre quand on dépasse un seuil de probabilité par bit (le seuil est défini par le maximum de probabilité de transition qui nous permet de décoder la bonne initialisation). Donc, dans ce chapitre on se propose de revoir les diverses versions de l'algorithme de Gallager qu'on a implémenté et testé, tout en commentant au fur et à mesure les résultats.

4.2 Les codes LDPC

Avant de parcourir les différentes méthodes de décodage itératif proposées par Gallager, il nous semble judicieux de commencer par introduire les codes LDPC. En effet, le système étudié nous fournit des équations de parité creuses et c'est le point commun avec les codes LDPC.

4.2.1 Construction des codes LDPC

Les codes LDPC "Low-Density Parity-Check", sont l'ensemble des codes, de longueur n, de dimension k et défini par une matrice de parité H creuse de dimension $((n-k) \times n)$. Un code LDPC peut aussi être représenté par son graphe biparti associé où chaque bit dans le mot de code est représenté par un noeud variable et une connexion relie un noeud variable à un noeud de parité si et seulement si le noeud de variable intervient dans l'équation de parité correspondant au noeud de parité (voir figure 4.1). Le nombre de connexions reliées à un noeud est appelé le degré du noeud.

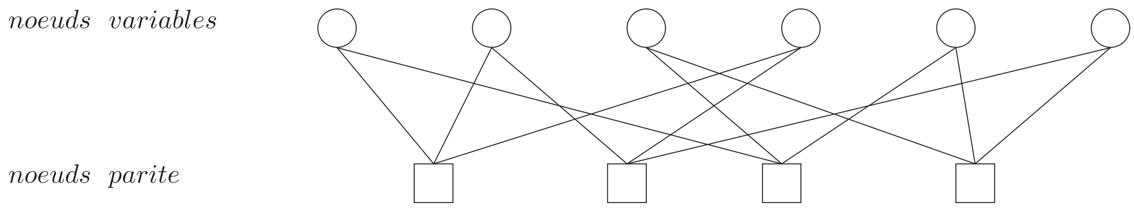


FIG. 4.1 – **Graphe biparti pour les codes LDPC**

4.3 Décodage itératif : algorithmes de Gallager

Puisque le décodage au sens du maximum de vraisemblance est optimal mais irréalisable en pratique, Gallager a introduit un algorithme de décodage itératif sous-optimal avec un nombre constant d'opérations par bit pour les codes LDPC.

Le décodage itératif des codes LDPC consiste en un algorithme à passage de messages mettant à jour itérativement le rapport de vraisemblance des noeuds dans le graphe biparti. Le point très important de l'algorithme est le suivant : à chaque itération, tous les messages sont supposés statistiquement indépendants. Cette hypothèse est raisonnable parce que la matrice de parité est creuse. C'est pourquoi, les dépendances dues à l'existence de cycles peuvent être ignorées sans dégradation significative des performances de décodage.

Dans cette présentation, nous adoptons les mêmes notations que celles introduites dans la thèse de S.Leveiller [1] :

- la séquence de bits codés en entrée du canal sera notée $(a_t)_{t=0}^{N-1}$ et la séquence reçue après passage dans le canal binaire symétrique sera notée $(z_t)_{t=0}^{N-1}$.
- À chaque instant t, soit Z_t la variable aléatoire en sortie du canal dont z_t constitue une réalisation.
- $Obs(a_t) = P(Z_t = z_t \mid a_t = 1)$: c'est *l'observation* du bit a_t, l'information sur le bit a_t dont on dispose en sortie de canal. Pour un canal binaire symétrique :

$$Obs(a_t) = p \quad si\ z_t = 0$$

$$Obs(a_t) = 1 - p \quad si\ z_t = 1$$

- $APP(a_t) = P(a_t = 1 \mid (Z_i)_{i=0}^{N-1} = (z_i)_{i=0}^{N-1})$: c'est la **probabilité a posteriori** du bit a_t.

$$APP(a_t) > 1/2 \quad \longrightarrow \quad retourne\ 1$$

$$APP(a_t) < 1/2 \quad \longrightarrow \quad retourne\ 0$$

- $S_m(t)$: ensemble d'indices de la $m-$ième équation de parité du $t-$ième bit.

– $S'_m(t)$: ensemble d'indices de la $m-$ième équation de parité du $t-$ième bit, t exclu : $S_m(t)\backslash\{t\}$.
– Le nombre moyen d'équations de parité par bit sera noté μ_b.

L'algorithme de Gallager est fondé sur le lemme suivant (le lecteur pourra se reporter à la référence [6] pour la preuve) :

Lemme 1 *Soit $(x_1, ..., x_l) \in \{0,1\}^l$, l bits statistiquement indépendants. Notons $p_t = P(x_t = 1)$ la probabilité que le $t-$ième bit soit égal à 1. Alors, la probabilité que le $l-$uplet $(x_1, ..., x_l)$ contienne un nombre impair de 1 (ou de manière équivalente la probabilité pour que $\sum_{i=1}^{l} x_i = 1 \ mod \ 2$) est donnée par :*

$$\frac{1 - \prod_{i=1}^{l}(1 - 2p_i)}{2}$$

Utilisons ce résultat pour calculer une nouvelle probabilité pour le bit a_t : $P'(a_t = 1)$ déduite à partir de la connaissance des probabilités des bits voisins (les bits $(a_i)_{i \in S'_m(t)}$ appartenant à l'équation de parité considérée d'indice m) :

$$\begin{aligned} a_t + \sum_{i=1}^{d-1} a_{t_i} = 0 \ mod \ 2 \Longleftrightarrow a_t &= \sum_{i=1}^{d-1} a_{t_i} \ mod \ 2 \\ &\Longrightarrow P'(a_t = 1) = P(\sum_{i=1}^{d-1} a_{t_i} = 1) \\ &\Longrightarrow P'(a_t = 1) = \frac{1 - \prod_{i=1}^{d-1}(1 - 2p_{t_i})}{2} \end{aligned}$$

où les t_i correspondent aux indices des bits voisins au bit d'indice t dans l'équation de parité d'indice m.

Chaque équation de parité permet alors d'évaluer une probabilité $P'(a_t)$, d'où on définit la probabilité **a posteriori** du bit a_t en tenant compte de toutes les équations de parité comme :

$$APP(a_t) = P(a_t = 1 \mid (Z_i)_{i=0}^{N-1} = (z_i)_{i=0}^{N-1})$$

En développant le calcul on aura donc :

$$\begin{aligned} APP(a_t) &= P(a_t = 1 \mid (Z_i)_{i=0}^{N-1} = (z_i)_{i=0}^{N-1}) \\ &= P(a_t = 1 \mid (Z_t) = z_t, \ (Z_i)_{i \neq t} = (z_i)_{i \neq t}) \\ &= P(Z_t = z_t \mid a_t, \ (Z_i)_{i \neq t} = (z_i)_{i \neq t}) \times \frac{P(a_t = 1, \ (Z_i)_{i \neq t} = (z_i)_{i \neq t})}{P(Z_t = z_t)} \end{aligned}$$

Cette dernière égalité a été obtenue en appliquant juste la définition de la probabilité conditionnelle, une fois arrivé à cette étape, on applique de nouveau la définition de la probabilité conditionnelle au deuxième terme du produit pour obtenir :

$$\begin{aligned} APP(a_t) = P(Z_t = z_t \mid a_t, \ (Z_i)_{i \neq t} = (z_i)_{i \neq t}) \times \\ \frac{P(a_t = 1 \mid (Z_i)_{i \neq t} = (z_i)_{i \neq t}) \times P((Z_i)_{i \neq t} = (z_i)_{i \neq t})}{P(Z_t = z_t)} \end{aligned}$$

Or comme on suppose que le canal est sans mémoire, le premier terme de ce produit se simplifie en $P(Z_t = z_t \mid a_t = 1) = Obs(a_t)$, et puis le quotient $\frac{P((Z_i)_{i \neq t} = (z_i)_{i \neq t})}{P(Z_t = z_t)}$ ne dépend que des z_i, donc on a le résultat de proportionnalité suivant :

$$APP(a_t) \propto Obs(a_t) \times P(a_t = 1 \mid (Z_i)_{i \neq t} = (z_i)_{i \neq t})$$

En utilisant le fait qu'on considère par hypothèse que toutes les équations sont indépendantes, la probabilité globale qui tient compte de toutes les équations de parité sera alors :

$$APP(a_t) \propto Obs(a_t) \times \prod_{m=1}^{\mu_b} P(\sum_{l \in S'_m(t)} a_l = 1 \mid (Z_i)_{i \neq t} = (z_i)_{i \neq t})$$

où $P(\sum_{l \in S'_m(t)} a_l = 1 \mid (Z_i)_{i \neq t} = (z_i)_{i \neq t})$ définit **l'information extrinsèque** du bit a_t dans sa $m-$ième équation (extrinsèque car c'est une information sur le bit a_t obtenue à l'aide des autres bits voisins), noté $Extr_m(a_t)$. On aura une information extrinsèque par bit et par équation.

Plus précisément :

$$\begin{aligned} Extr_m(a_t) &= P(\sum_{l \in S'_m(t)} a_l = 1 \mid (Z_i)_{i \neq t} = (z_i)_{i \neq t}) \\ &= \frac{1 - \prod_{l \in S'_m(t)} (1 - 2P(a_l = 1 \mid (Z_i)_{i \neq t} = (z_i)_{i \neq t}))}{2} \end{aligned}$$

On définit aussi **la probabilité a posteriori partielle**, qui nous sera utile dans la suite et qui correspond à la probabilité globale de a_t privée d'une information intrinsèque, celle générée par la $m-$ième équation de parité :

$$A_m(a_t) \propto Obs(a_t) \prod_{i, i \neq m} Extr_i(a_t)$$

Notation : $Extr^j_m(a_t)$ désigne la probabilité extrinsèque associée au bit a_t dans sa $m-$ième équation à la $j-$ième itération.

Conclusion :

Dans tout ce qui précède, on décrit les expressions à calculer à chaque itération de l'algorithme de Gallager, il nous reste juste à préciser l'initialisation de l'algorithme (l'étape 0) :
- $Extr^0_m(a_t) = 1/2$. Si on considère que nous n'avons pas d'information à priori sur les bits.
- Initialiser les $Obs(a_t)$ selon le canal que l'on utilise (dans notre cas c'est un canal binaire symétrique ce qui correspond à p ou $1 - p$ selon que z_t est égale à 0 ou 1).

et la terminaison :
- Calculer les $APP(a_t)$ totale pour chaque bit.
- Décoder selon que $APP(a_t) > 1/2$ (décoder en 1) ou $APP(a_t) < 1/2$ (décoder en 0).

Remarque

Dés le début de ce chapitre on a souligné l'importance des codes LDPC, et notamment l'intérêt d'avoir des équations de poids faible : par exemple dans ce cas, l'une des raisons peut s'expliquer par le fait que dans le calcul des extrinsèques on fait un produit de $(d-1)$ termes inférieurs à 1 en valeur absolue, et donc plus d est élevé, plus ce produit tend vers 0, et donc l'algorithme ne pourra pas démarrer.

4.4 Décodage itératif et registre filtré

Rappelons le modèle d'attaque (dans la figure 4.1) :

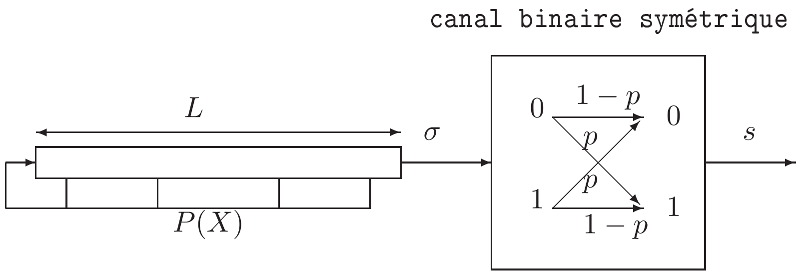


FIG. 4.2 – **Modèle de l'attaque**

La suite σ, par définition, vérifie l'équation de récurrence linéaire définie par le polynôme de rétroaction P. Le mot de N bits, $(\sigma_0, ..., \sigma_{N-1})$, est donc un mot de code de longueur N et de dimension L. Pour le décoder on va utiliser les différents algorithmes de Gallager, qui exploitent l'existence d'équations de parité creuses pour le code.
L'attaque se divise donc en 2 parties bien distinctes :

- **Une phase de précalcul**, qui ne dépend que de la fonction de combinaison et du polynôme de rétroaction P. Cette phase consiste à déterminer des équations de parité pour la suite σ : dans nos programmes on a cherché les équations de poids 3.
- **Une phase de décodage**, qui consiste à appliquer l'algorithme de Gallager afin de retrouver $(\sigma_n)_{n<N}$.

4.4.1 Phase de précalcul

Dans la phase de précalcul, on recherche toutes les équations linéaires faisant intervenir exactement 3 bits de la suite $(\sigma_n)_{n<N}$. Il s'agit des équations de la forme

$$\sigma_n + \sigma_{n-i} + \sigma_{n-j} = 0 \tag{4.1}$$

satisfaites pour tout n.

De telles équations correspondent exactement aux multiples Q de poids 3 du polynôme de rétroaction P de la forme $Q(X) = 1 + X^i + X^j$. L'étape de précalcul consiste donc à rechercher tous les multiples de P ayant cette forme particulière. Pour cela, on utilisera l'algorithme suivant.

Algorithme 7

- *Calculer tous les restes* $q_i(X) = X^i \bmod P(X)$ *pour* $1 \leq i < N$ *et les stocker dans un tableau* T *défini par*

$$\forall 0 \leq a < 2^L,\ T[a] = \{i, q_i(X) = a\}\ .$$

- *Pour tout* $i \in \{1, \dots, N\}$,
  calculer $A = 1 + q_i(X)$
  Pour $j \in T[A]$, $1 + X^i + X^j$ *est un multiple de* $P(X)$ *ayant la forme souhaitée.*

Remarque :

Dans le prochain chapitre où on parlera d'une autre technique de décodage, on aura besoin de calculer la matrice génératrice d'un code formé par un LFSR. Pour cela on utilisera le même principe que celui utilisé dans cette phase de précalcul (on utilise le polynôme de rétroaction et on fera des divisions modulo ce polynôme ce qui nous donnera les colonnes de matrice génératrice sous forme systématique).
Dans toute la suite du chapitre on décrira les différentes versions de l'algorithme de Gallager implémentés et testés.

4.5 Algorithmes de Gallager

Pour commencer on va donner l'algorithme de Gallager comme il a été introduit dans les paragraphes précédents (version α selon [1]) :

1. Initialisation :

$$Obs(a_t) = 1 - p \quad si\ z_t = 1$$
$$Obs(a_t) = p \quad si\ z_t = 0$$

$$Extr_m^{(0)}(a_t) = 1/2.$$

2. Déroulement :
à l'itération $j \geq 1$: $\forall t$, $\forall m$

$$A_m^{(j)}(a_t) \propto Obs(a_t) \prod_{i, i \neq m} Extr_i^{(j-1)}(a_t)$$
$$Extr_m^{(j)}(a_t) = 1/2(1 - \prod_{i \in S'_m(t)} (1 - 2A_i^{(j)}(a_t)))$$

3. Terminaison :
à l'itération nb_{iter} :

$$APP(a_t) \propto Obs(a_t) \prod_m Extr_m^{(nb_{iter})}(a_t)$$

$$APP(a_t) > 1/2 \implies a_t = 1,$$
$$APP(a_t) < 1/2 \implies a_t = 0$$

Remarque :

Dans ce chapitre on ne va pas faire l'étude de la complexité, elle est effectuée dans la thèse de S.Leveiller [1].

4.5.1 Algorithme de Gallager sous forme logarithmique

Dans ce paragraphe on va manipuler des quantités logarithmiques car elles présentent beaucoup d'avantages : d'une part elles permettent de remplacer les multiplications par des additions et, d'autre part ça donne lieu à des approximations qui facilitent le calcul comme on va le voir dans les versions simplifié de cet algorithme. Pour cela on commence par définir : **la vraisemblance** d'une variable aléatoire binaire X

$$L(X) = \log \frac{Pr[X=0]}{Pr[X=1]} = \log \frac{Pr[X=0]}{1 - Pr[X=0]}$$

On va réécrire la version précédante de l'algorithme de Gallager en utilisant les logarithmes

1. Initialisation :

$$L[Obs(a_t)] = \log\frac{1-p}{p} \quad si\ z_t = 1$$
$$L[Obs(a_t)] = \log\frac{p}{1-p} \quad si\ z_t = 0$$

$$L[Extr_m^{(0)}(a_t)] = 0.$$

2. Déroulement :
à l'itération $j \geq 1$: $\forall t$, $\forall m$

$$L[A_m^{(j)}(a_t)] = L[Obs(a_t)] \sum_{i,i\neq m} L[Extr_i^{(j-1)}(a_t)]$$

$$L[Extr_m^{(j)}(a_t)] = \log\frac{(1-\prod_{i\in S'_m(t)}\Lambda^{(j)}(i))}{(1+\prod_{i\in S'_m(t)}\Lambda^{(j)}(i))}$$

avec $\Lambda^{(j)}(i) = \frac{1-\exp^{L[A_m^{(j)}(a_i)]}}{1+\exp^{L[A_m^{(j)}(a_i)]}}$

3. Terminaison :
à l'itération nb_{iter} :

$$L[APP(a_t)] = L[Obs(a_t)] + \sum_m L[Extr_m^{(nb_{iter})}(a_t)].$$

$$L[APP(a_t)] > 0 \quad \Longrightarrow \quad a_t = 1,$$
$$L[APP(a_t)] < 0 \quad \Longrightarrow \quad a_t = 0$$

Résultats et simulations :

Cet algorithme que nous présentons en premier (pour des raisons de cohérence : vu que les autres versions découlent de celui là et sont plus exactement des simplifications de ce dernier), a été implémenté en dernier recours car toutes les simplifications de ce dernier ont donné des rapports de vraisemblance impropres à une utilisation ultérieure dans un décodeur à entrées souples. L'ensemble des résultats présentés sur les courbes correspondantes à chacun des algorithmes, a été obtenu à partir de 1000 tirages, effectués sur registre de mémoire $K = 23$, de polynôme de rétroaction $g(x) = x^{23} + x^{18} + x^{12} + x^{11} + x^{10} + x^5 + x^3 + x + 1$. Les équations de parité utilisées sont de poids $d = 3$ et générées à l'aide de l'algorithme de A.Canteaut. Pour introduire de l'erreur on utilise un canal binaire symétrique avec une probabilité de transition qui correspond à la valeur seuil à partir de laquelle l'algorithme n'arrive plus à converger (dans ce cas $p_{seuil} = 0.35$. Pour cet algorithme la courbe suivante représente l'histogramme des "log-vraisemblance" de l'erreur (le nombre d'itérations de l'algorithme de Gallager est fixé à 100 pour toutes les variantes). En se basant sur plusieurs simulations avec des ($p > seuil$), on remarque que la distribution des vraisemblances de l'erreur tend vers une gaussienne dont on a calculé l'espérence et la variance (voir tableau 4.1) (on remarque bien que la variance

p	Espérance	Variance
0.35	-0.3034	0.6035
0.36	-0.27628	0.4671
0.37	-0.1983	0.3843
0.39	-0.1288	0.2395
0.41	-0.0763	0.1492
0.45	-0.0205	0.041

TAB. 4.1 – Variance et espérance des gaussiennes pour différents ($p > seuil$)

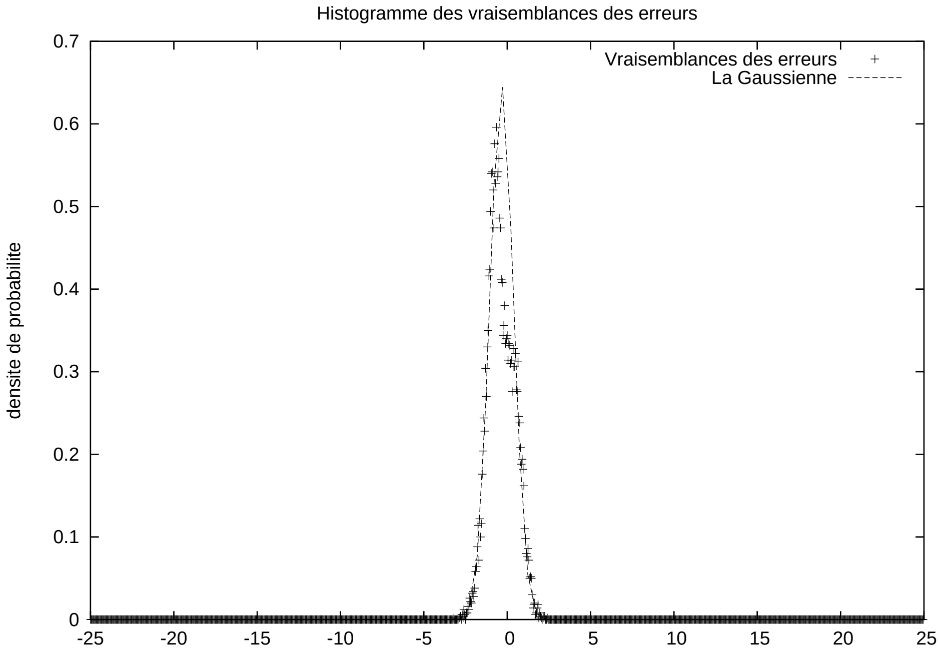


FIG. 4.3 – Superposition de la courbe théorique et de la courbe expérimentale pour moyenne = -0.3034 et variance = 0.6035.

est le double de l'espérance, ce qu'on montrera dans le chapitre suivant en explicitant l'expression de cette gaussienne).

Remarque

Les résultats expérimentaux montrent que la gaussienne commence à se détériorer au fur et à mesure qu'on augmente la probabilité de transition du canal binaire symétrique (on voit l'apparition de deux pics) : voir figure 4.4. Les raisons de cette détérioration seront étudiés durant la suite du stage..

4.5.2 Algorithme de Gallager approché (première simplification)

Cet algorithme est une simple simplification qui consiste à considérer les probabilités a posteriori globales au lieu des probabilités partielles. C'est une solution beaucoup moins coûteuse du point de vue de la mémoire (puisqu'on évite de conserver en mé-

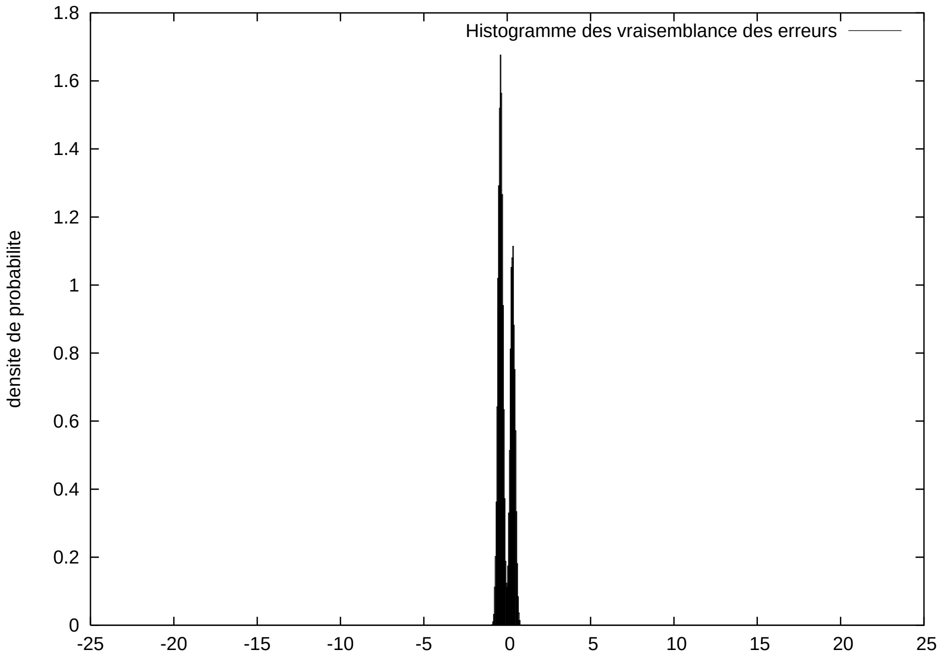


FIG. 4.4 – La distribution des vraisemblances des erreurs pour $p = 0.41$

moire toutes les APP partielles d'une itération à l'autre) et en terme des calculs effectués.

1. Initialisation :

$$L[Obs(a_t)] = \log \frac{1-p}{p} \quad si\ z_t = 1$$
$$L[Obs(a_t)] = \log \frac{p}{1-p} \quad si\ z_t = 0$$

$$L[Extr_m^{(0)}(a_t)] = 0.$$

2. Déroulement :
   à l'itération $j \geq 1$: $\forall t$, $\forall m$

$$L[APP^{(j)}(a_t)] = L[Obs(a_t)] + \sum_i L[Extr_i^{(j-1)}(a_t)]$$

$$\forall m\ L[Extr_m^{(j)}(a_t)] = \log \frac{(1 - \prod_{i \in S'_m(t)} \Lambda^{(j)}(i))}{(1 + \prod_{i \in S'_m(t)} \Lambda^{(j)}(i))}$$

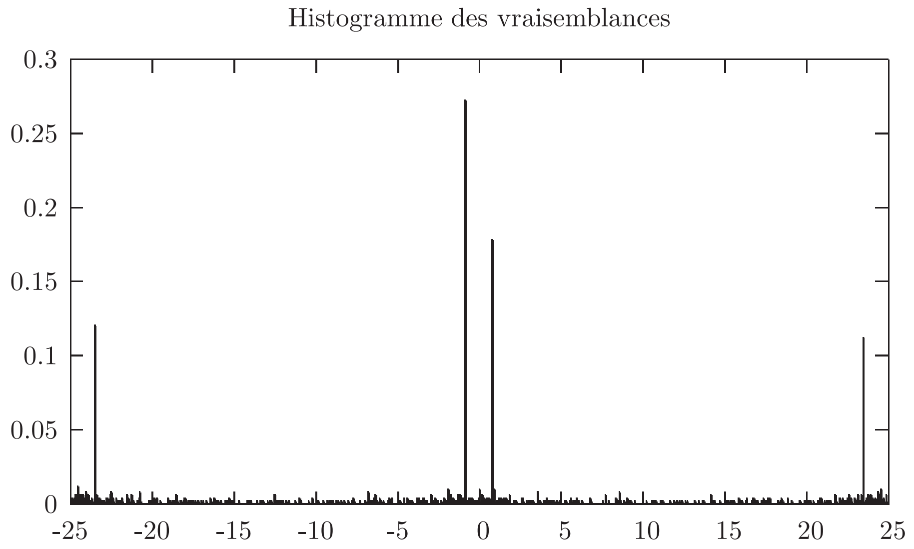
   avec $\Lambda^{(j)}(i) = \frac{1 - \exp^{L[APP^{(j)}(a_i)]}}{1 + \exp^{L[APP^{(j)}(a_i)]}}$
3. Terminaison :
   à l'itération nb_{iter} :

$$L[APP(a_t)] = L[Obs(a_t)] + \sum_m L[Extr_m^{(nb_{iter})}(a_t)].$$

$$L[APP(a_t)] > 0 \quad \Longrightarrow \quad a_t = 1,$$
$$L[APP(a_t)] < 0 \quad \Longrightarrow \quad a_t = 0$$

4.5.3 Résultats et simulations

La courbe suivante (est tracée en tenant compte des mêmes paramètres qu'on a exploité pour la courbe précédente. Mais l'inconvénient essentiel de cet algorithme -pour la suite de notre travail- était que la distribution finale des vraisemblances de l'erreur au delà du seuil ne donnait pas une gaussienne. En essayant de voir de plus près le déroulement des itérations on s'est rendu compte du fait que l'algorithme converge vers des résultats faux, par exemple il donne des vraisemblances élevées à des bits dont les APP ont un signe erroné.



Remarque d'implémentation

Lors de l'implémentation on s'est rendu compte qu'il fallait fixer des seuils aux probabilités calculés. En effet, ces méthodes de calculs présentent un inconvénient majeur : les quantités par lesquelles on divise peuvent être arbitrairement faibles en théorie. Diviser par une quantité proche de zéro comporte des risques, la précision des ordinateurs étant limitée. Ce seuillage a nettement amélioré les performances : au-dessous du seuil ($p < 0.35$) l'algorithme arrive à décoder sans erreurs.

4.5.4 Algorithme de Gallager : "MIN-SUM approché"

Dans cet algorithme on retient la même simplification qu'on a évoqué dans le paragraphe précédant, à savoir travailler avec les probabilités a posteriori globales, une deuxième simplification consiste à approximer les log-vraisemblance des extrinsèque, plus exactement on remplace $\prod \Lambda(i)$ par le minimum des termes en valeurs absolue, pondéré par le signe du produit :

$$\prod_{i \in S'_m(t)} \Lambda(i) \simeq sgn(\prod_{i \in S'_m(t)} \Lambda(i)) \times \min_{i \in S'_m(t)} |\Lambda(i)|$$

ce qui donne :

$$L[Extr_m(a_t)] = -sgn(\prod_{i \in S'_m(t)} (L[APP(a_i)])(i)) \times \min_{i \in S'_m(t)} |L[APP(a_i)]|$$

1. Initialisation :

$$L[Obs(a_t)] = \log \frac{1-p}{p} \quad si\ z_t = 1$$
$$L[Obs(a_t)] = \log \frac{p}{1-p} \quad si\ z_t = 0$$

$$L[Extr_m^{(0)}(a_t)] = 0.$$

2. Déroulement :
   à l'itération $j \geq 1$: $\forall t$

$$L[APP^{(j)}(a_t)] = L[Obs(a_t)] + \sum_i L[Extr_i^{(j-1)}(a_t)]$$

   $\forall m$

$$L[Extr_m^{(j)}(a_t)] = -sign(\prod_{i \in S'_m(t)} L[APP^{(j)}(a_i)]) \times \min_{i \in S'_m(t)} |L[APP^{(j)}(a_i)]|$$

3. Terminaison :
   à l'itération nb_{iter} :

$$L[APP(a_t)] = L[Obs(a_t)] + \sum_m L[Extr_m^{(nb_{iter})}(a_t)].$$

$$L[APP(a_t)] > 0 \implies a_t = 1,$$
$$L[APP(a_t)] < 0 \implies a_t = 0$$

Résultats et simulations

Cet algorithme a le même défaut que le précédent à savoir l'information souple fournie au delà du seuil n'est pas exploitable (n'est pas une gaussienne) mais il a l'avantage d'être beaucoup plus rapide que les deux algorithmes précédents, et il donne de très bons résultats pour une probabilité de transition du canal faible (au dessous du seuil).

Conclusion

Cette étude des variantes de l'algorithme de Gallager nous a permis de déterminer la version qui peut être exploitée par un algorithme de décodage à décisions souples. Une fois l'information souple obtenue, le but sera d'exploiter cette information pour décoder : cela sera l'objet du chapitre suivant.

Chapitre 5

Décodage souple

5.1 Introduction

L'étude menée au chapitre précédent montre que le mot de code obtenue après décodage itératif (la version logarithmique de l'algorithme de Gallager sans simplification) contient encore des erreurs si on dépasse un certain seuil de la probabilité de transition du canal binaire symétrique. Cela dit l'intérêt de ce dernier décodage réside dans l'information souple qu'il nous fournit et qui correspond à la sortie d'un canal gaussien : pour cela on a choisi d'appliquer la technique de décodage souple par ensemble d'information appliqué à un canal gaussien proposé dans la thèse d'A.Valembois [2], et comme sa technique est un mélange de décodage dur et de décodage souple, on va parler de ces deux types de décodages et exploiter le lien entre eux pour définir clairement un algorithme de décodage. Après avoir rappelé les différents types de décodage, à savoir décodage dur, décodage souple, je décrirai quelques variantes des algorithmes de décodage par ensemble d'information. Enfin, je parlerai de décodage souple sur canal gaussien, ce qui nous amènera à décrire l'algorithme principal que nous avons implémenté.

5.2 Canal gaussien

Le canal gaussien à bruit blanc additif -(AWGN en anglais *additive white gaussien noise*)- est le modèle de canal le plus répandu. Définissons ce canal dans le cas d'une modulation antipodale (BPSK *Binary Phase shift Keying*) d'énergie 1. Cette modulation correspond à émettre le signal $(-1)^b$ pour transmettre un bit b.
Pour ce type de canal, 1 ou -1 est envoyé et à la réception un signal réel y_i est mesuré et qui sera la somme d'un réel $x_i = (-1)^b$ et d'une réalisation d'une loi normale d'espérance nulle et de variance σ^2 ($y_i = x_i + N(0, \sigma^2)$). Dans ce cas la densité de probabilité conditionnelle $p(y_i|x_i)$ s'écrit :

$$p(y_i|x_i) = \frac{1}{\sqrt{2\Pi\sigma^2}} exp(\frac{-(y_i - x_i)^2}{2\sigma^2})$$

Calculons alors la vraisemblance de x_i pour un canal gaussien :

$$
\begin{aligned}
L(x_i) &= \log \frac{p(y_i|x_i = +1)}{p(y_i|x_i = -1)} \\
&= \log \frac{\frac{1}{\sqrt{2\Pi\sigma^2}} exp(\frac{-(y_i-1)^2}{2\sigma^2})}{\frac{1}{\sqrt{2\Pi\sigma^2}} exp(\frac{-(y_i+1)^2}{2\sigma^2})} \\
&= \frac{2y_i}{\sigma^2}
\end{aligned}
$$

Cela nous permet alors de calculer la variance du bruit gaussien à partir de la variance de la distribution gaussienne de la log-vraisemblance : la variance est égale à $\frac{4}{\sigma^2}$ et la moyenne $\frac{2}{\sigma^2}$ (remarquons que la variance est égale deux fois la moyenne : cela a été vérifié lors des résultats du décodage itératif voir tableau 4.1).

5.3 Décodage dur vs. décodage souple

Dans une chaîne de transmission habituelle, le démodulateur fournit au décodeur canal de l'information et selon la nature de cette information on distingue deux types de décodages :

– décodage dur : le démodulateur compare le signal à certains seuils et prend une décision quant au symbole reçu en fonction du résultat de ces comparaisons (entrée du canal discrète). Mais ce faisant, la valeur précise réellement mesurée du signal est perdue, ce qui constitue une perte d'information.

– Décodage souple : le démodulateur souple ne prend pas de décision et fournit à la couche physique qui lui succède une valeur représentant le plus fidèlement possible ce qu'il a mesuré, minimisant ainsi la perte de l'information.

Dans notre cas l'équivalent du démodulateur souple c'est le décodeur itératif, et donc on va essayer d'exploiter l'information fournie par ce dernier.

Notations

- $(c \in C)$ le mot de code émis qu'on cherche à trouver.
- $y = (y_1, y_2, ... y_n)$ l'information souple qu'on cherche à décoder.
- m le mot qui serait renvoyé par un démodulateur dur.

Avec ces notations, on peut décrire plus concrètement les deux types de décodage :

– Décodage dur : les motifs d'erreurs possibles sont $e = m \oplus c$ $(c \in C)$, et il s'agit donc de minimiser le poids de Hamming de ces motifs.

– Décodage souple : il s'agit de minimiser le poids généralisé du motif d'erreur $\sum_{i=1}^{n}(c_i \oplus m_i)|y_i|$ que nous définissons comme étant le poids généralisé du motif d'erreur..

Cette différence rend le décodage souple plus précis : en effet, les tests montrent que les performances du décodage souple dépassent largement celles du décodage dur.

Le but du reste du chapitre n'est pas d'exposer les différents algorithmes de décodage que j'ai du rencontrer durant mon stage, mais ça sera plutôt d'étudier les algorithmes qu'on a choisi d'implémenter et tester.

5.4 Décodage par ensemble d'information souple

Le décodage par ensemble d'information est une technique classique qui fut introduite par Prange [8] dans les années soixante. Contrairement aux algorithmes exhaustifs, le décodage par ensembles d'information exploite la redondance du code, son principe repose sur la constatation qu'il suffit pour décoder un vecteur x de trouver parmi ses coordonnées un ensemble de k positions d'information ne contenant aucune erreur.

Définition 8 (Ensemble d'information) : *Soit C un code linéaire de longueur n et de dimension k, et $\mathbf{G}$ une matrice génératrice de C. Un sous-ensemble I de $\{1, ..., n\}$ de taille k est un ensemble d'information pour ce code si et seulement si la restriction du code à ces k positions est un espace vectoriel de dimension k. Ceci équivaut à dire que la matrice carré G_I correspondant à la restriction de G aux positions I est inversible.*

Il y a de nombreuses variantes du décodage par ensemble d'information : celles qui autorisent certains motifs d'erreurs dans l'ensemble d'information [9] [11], celles qui utilisent un code poinçonné [10], obtenu en enlevant certaines positions de redondance du code C, et enfin celles qui unifie les deux approches tels que l'algorithme de Jacques Stern [12] et l'algorithme de Canteaut et Chabaud [13].
Toutes ces variantes offraient un principe puissant pour le décodage dur qui permettaient de réduire fortement les complexités. On peut donc s'attendre à ce qu'il en aille de même pour le décodage souple, d'autant que l'information souple devrait pouvoir permettre de choisir pertinemment les ensembles d'information successifs.

Décodage par résonance stochastique

L'algorithme qu'on va décrire maintenant est l'algorithme d'A.Valembois [2] : il s'agit de mettre au point un algorithme de décodage mixte (la variante qui autorise un petit nombre d'erreur p dans l'ensemble d'information), en choisissant des bases successives d'ensembles d'information de manière aléatoire, mais corrélée avec leur fiabilité (l'information souple), c.a.d que plus une base sera fiable (constituée des k bits les plus fiables) plus elle aura de chance d'être choisie par l'algorithme.
La variante de l'algorithme de décodage mixte choisie est celle du décodage mixte avec utilisation de codes poinçonnés(qui unifie les deux approches décrites ci-dessus), qui est le plus rapide pour la plupart des longueurs et dimensions de codes utilisables en pratique.

Choix de bases :

À chaque itération, nous ajoutons à notre information souple (de manière non cumulée, on reconsidère l'information originale à chaque fois) un bruit gaussien secondaire, crée dans l'algorithme. Si on note $(y_i)_{i=1\ldots n}$ l'information originellement reçu, à chaque itération on calcule grâce à un générateur pseudo-aléatoire, $y_i' = y_i + n_i$ où n_i est une réalisation d'une variable (pseudo-) aléatoire de loi normale, centrée et de variance σ^2 qui sera un nouveau paramètre qui réglera l'amplitude de l'exploration de l'algorithme : plus cette variance est faible, plus les bases successives seront proches de la base la plus fiable et moins cette exploration aura d'amplitude (les itérations successives sont donc d'avantage corrélées entre elles), cela veut dire que le choix de ce paramètre est crucial. Enfin, on choisit alors la base la plus fiable relativement aux y_i'. C'est à dire que l'on choisit pour base les k positions correspondant aux $|y_i'|$ les plus élevés.

Décodage mixte utilisant les codes poinçonnés :

L'idée est d'autoriser un petit nombre d'erreur p sur l'ensemble d'information : la valeur optimale de p est généralement égale à 2, et ce, indépendamment de la longueur du code. La deuxième amélioration (Dumer en 91 [15]) consiste à calculer à chaque itération les éléments du cosets de poids au plus p non pas sur l'ensemble d'information mais plutôt sur un ensemble contenant strictement un ensemble d'information. Sur chaque moitié de support des codes poinçonnés, on devra calculer les $\sum_{i=0}^{p/2} \binom{s/2}{i}$ motifs d'erreur de poids au plus $p/2$, pour cela on utilise la technique suivante :

Énumération ordonnée des motifs d'erreur :

Cet algorithme qu'on a implémenté s'inspire d'une idée de Dumer [16] : le but est d'énumérer N vecteurs de plus faibles poids généralisé. Elle consiste en n (longueur du code) étapes.

Pour chaque j on définit la liste $X_j = \{(m, w)\}$ des couples (m, w) où :
m : mot restreint aux sous-ensemble $\{1, ..., j\}$ du support et w :poids généralisé de m.

Données :

- $(y_i)_{i=1\ldots n}$ l'information souple originellement reçue.
- N : le nombre de vecteurs à énumérer.

Résultats :

- Liste des N mots de plus faible poids généralisé croissant.

1. $j \leftarrow 1$
2. $X_j = \{(0,0)\ (1, |y_0|)\}$
3. **Tant que** $j \leq [\log_2 N]$ **faire**

   //Construire X_{j+1} *à partir de* x_j
   - concaténer à tous les éléments de X_j un 0 à la fin des mots m et garder le même w.

     //le poids généralisé ne change pas quand on ajoute un 0.
   - concaténer à tous les éléments de X_j un 1 à la fin des mots m et mettre à jour w : $w = w + |y_j + 1|$.

//Après ces deux concaténations on obtient alors $2*2^j$ *éléments qui constitueront la liste* X_{j+1}.

- détruire X_j composées de 2^j éléments.
- $j \leftarrow j+1$

//A la sortie de cette boucle $j = \log_2 N$ *et la Liste* $X_{\log_2 N}$ *contient* N *mots*

4. **Tant que** $j \leq n$ **faire**
   - concaténer 0 et 1 pour avoir $2N$ mots.
   - Trier par poids généralisé croissant ces $2N$ mots.
   - Ne garder que les N premiers mots.
   - $j \leftarrow j+1$

Après la $n$ème étape on a la liste attendue des N mots de plus faible poids généralisé croissant.

Algorithme complet :

A chaque itération :

- 1ère Étape : Choix de l'ensemble d'information
  - Calculer $y'_i = y_i + n_i$ avec n_i réalisation d'une variable aléatoire de loi normale, centrée et de variance σ^2 (la valeur optimale de σ^2 est à peu près le quart de la puissance du bruit).
  - On choisit la base la plus fiable relativement aux y'_i.
- 2ème Étape : application du décodage mixte avec utilisation de codes poinçonnés
  - Choix d'un $s-$sous-ensemble I contenant strictement l'ensemble d'information choisi.
    card$(I) = s > k$, choix de s pair.
  - Calcul de la matrice de parité H_I du $[s, k]$ code poinçonné C_I.
  - Calcul et stockage pour chaque moitié de l'ensemble I des 2^{s-k} motifs d'erreurs de plus faible poids généralisé.
    $\longrightarrow$ Pour générer ces motifs on utilise l'algorithme décrit précédemment.
  - Calcul, pour chaque paire (e_1, e_2) de demi-motifs dont les syndromes partiels se somment en $\widetilde{s}$ le syndrome selon H_I de m_I restriction de m à I.
  - Réencoder la partie information de $m_I + (e_1|e_2)$ en $C(m_I + (e_1|e_2))$.
  - Calcul du poids de $m + C(m_I + (e_1|e_2))$, si ce poids est le plus petit on stocke $C(m_I + (e_1|e_2))$.

p	Espérance	Variance	N moyen
0.35	-0.3034	0.6035	5
0.36	-0.27628	0.4671	39
0.37	-0.1983	0.3843	73
0.39	-0.1288	0.2395	105
0.41	-0.0763	0.1492	388
0.45	-0.0205	0.041	2662

TAB. 5.1 – Nombre moyen d'itérations pour trouver 2 erreurs dans l'ensemble d'information

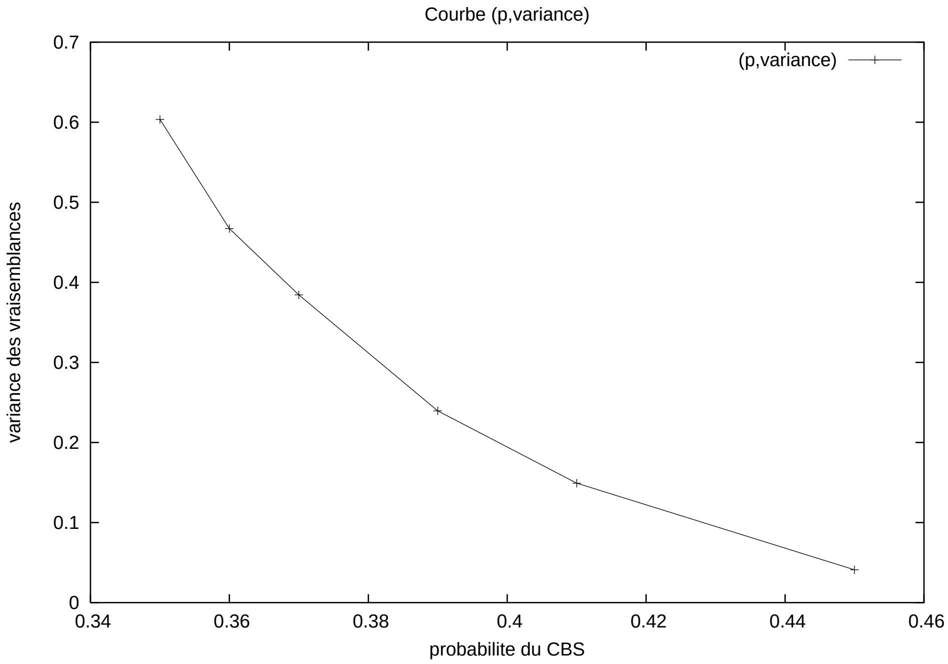


FIG. 5.1 – Courbe représentant la variance des vraisemblances des erreurs en fonction des probabilités de transitions du CBS

5.5 Résultats

La première étape de mes tests consistait à essayer de voir le nombre moyen d'itération (voir tableau 5.1) nécessaire pour trouver au maximum 2 erreurs dans l'ensemble d'information en fonction de la probabilité de transition du canal binaire symétrique : en effet, une fois on a cet ensemble, on applique l'algorithme de décodage implémenté par Anne Canteaut et on obtient le mot décodé.

La deuxième étape de mes tests était de voir les performances de l'algorithme présenté ci-dessus. Le tableau 5.3 donne une idée sur le nombre d'itérations effectués pour décoder le mot reçu en fonction de la probabilité de transition du canal binaire symétrique. Les résultats du tableau 5.2 montrent que le nombre d'itération varie selon le choix de s, mais il faut noter que la durée d'exécution d'une itération augmente énormément au fur et à mesure qu'on augmente s. Cela peut s'expliquer par le nombre de couples qui vérifient la somme de leurs syndromes est égale au syndrome du mot restreint à I (voir algorithme), qui lui augmente exponentiellement quand on augmente s.

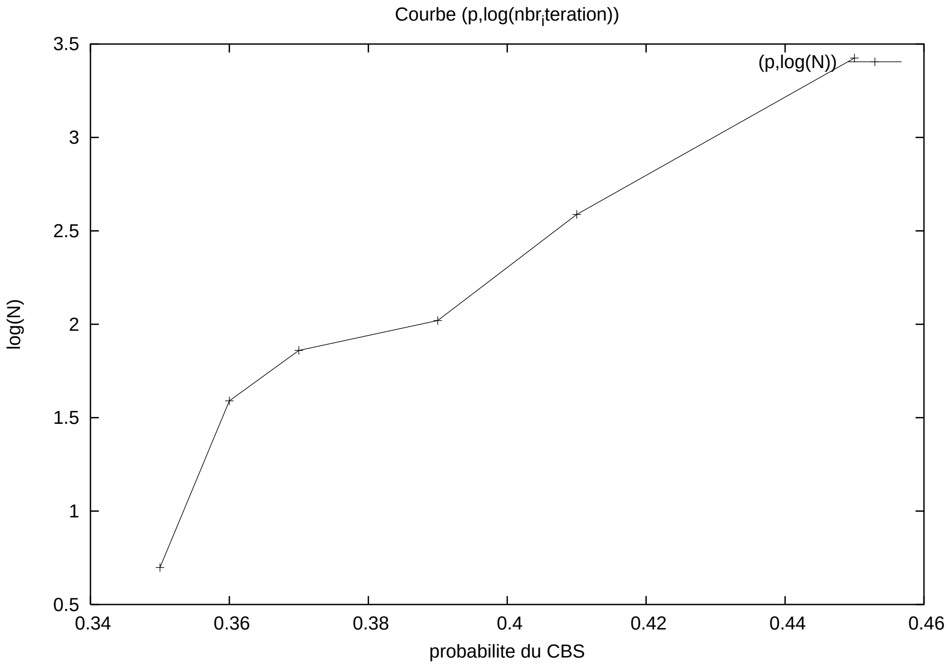


FIG. 5.2 – Courbe représentante du nombre d'itérations pour décoder en fonction des probabilités de transitions du CBS

s	Nbre d'itérations	Nbre de couples
$s = k + 3$	191	12
$s = k + 5$	ne décode pas	30
$s = k + 7$	14	200
$s = k + 9$	19	2500
$s = k + 11$	29	7000
$s = k + 13$	21	15000

TAB. 5.2 – Variation du nombre d'itérations en fonction de s pour $p = 0.35$

p	Nbre itérations
0.35	29
0.36	35
0.37	37
0.38	30
0.39	4
0.41	17
0.45	ne décode pas

TAB. 5.3 – Nombre d'itérations en fonction de p pour $s = 11$ et σ optimal

σ	Nbre itérations
0.7	62
0.8	52
0.9	127
1	14
1.1	25
1.3	29
1.4	8
1.5	8
1.7	8
1.9	8
2.2	8
3	33
4	218

TAB. 5.4 – Nombre d'itération en fonction de σ du bruit ajouté pour $s = 11$ et $p = 0.35$

Remarque :

- Tous ces tests sont effectués avec les mêmes données du chapitre précédent à savoir le polynôme de rétroaction (donc dimension du code qui est égale au degré de ce polynôme $K = 23$).
- Ces tests ont été effectués pour une valeur de σ^2 optimale du bruit ajouté permettant d'aboutir le plus rapidement à la résonance stochastique, cette valeur optimale vaut à peu près le quart de la puissance du bruit (résultat expérimental vérifié aussi dans le cadre de mes tests). Pour des tests de σ on peut voir les résultats du tableau 5.4.
- le paramètre s (longueur du code poinçonné utilisé) dans notre algorithme influe énormément sur le temps d'exécution d'une itération (voir tableau 5.2), cela dit il y a une valeur optimale de s qui vérifie plusieurs contraintes (voir la section 2.1.4 sur les codes poinçonnés dans la thèse d'A.Valembois [2]).

Conclusion et perspectives

Dans ce travail nous nous sommes attachés à améliorer les cryptanalyses de registres filtrés fondés sur des techniques de décodage itératif. Nous avons montré qu'en utilisant une version adéquate de l'algorithme de Gallager, on obtient, même quand le décodage itératif échouait à retrouver la clé secrète qui est l'initialisation du registre, une information souple sur les bits de sortie du registre qui peuvent être exploités par des algorithmes de décodage souple adaptés au canal gaussien. Parmi tous les algorithmes de décodages souple, nous avons choisi des techniques de décodage par ensemble d'information pour leur complexité réduite. Plus précisément, on a appliqué une version évoluée se basant sur un décodage par résonance stochastique d'A.valembois [2] combiné à un décodage dur mixte avec utilisation des codes poinçonnés.

Nous avons montré que les premiers résultats obtenus par cette approche ont montré que ces techniques améliorent significativement la plage des paramètres pour lesquels une cryptanalyse est possible. Cela dit, on peut envisager, dans le cadre d'une amélioration du travail, la possibilité de réduire la complexité de la version de l'algorithme de Gallager qui donne la bonne information souple et aussi la possibilité d'un raccourcissement intelligent du code avant l'étape de décodage souple.

Bibliographie

[1] S.Leveiller. *Quelques algorithmes de cryptanalyse du registre filtré.* Thèse, ENST, Paris, 2003.

[2] A.Valembois. *Décodage, Détection et Reconnaissance des Codes Linéaires binaires.* Thèse, université de Limoges, 2000.

[3] W.Meier and O.Staffelbach . *Fast correlation attack on certain stream ciphers*, Journal of Cryptography, pp. 159-176,1989.

[4] T.Siegenthaler. *Decrypting a class of stream ciphers using ciphertext only.* IEEE Transactions on Computers, vol. C-34, $n°1$, 1985, pp. 81-84.

[5] R.G.Gallager, *Low-density parity-check codes.*IRE Trans.inform.Theory, IT-8 :21-28,1962.

[6] R.G.Gallager, *Low-density parity-check codes*,MIT Press, Cambridge, MA, 1963.

[7] T.J.Richardson, M.A. Shokrollahi and R.L. Urbanke.- Design of capacity approaching irregular low-density parity-check codes, IEEE *Trans.* Inf. Theory, vol. 47, no.2,pp.619-637, February 2001.

[8] Prange (E.).- The use of information sets in decoding cyclic codes. IRE *Transaction*, vol. IT-8, 1962, pp. S5-S9.

[9] Lee (P.J.) et Brickell (E.F.).- An observation on the security of Mceliece's public-key cryptosystem. Electronics Letters, vol.26, n1, 1990, pp. 16-17.

[10] Leon (J.S.). - A probabilistic algorithm for computing minimum weight of large error-correcting codes. IEEE Transactions on Information Theory, vol.34, n5, 1988, pp.1345-1359.

[11] Kasami (T.). - A decoding procedure for multiple-error-correcting cyclic codes. IEEE Transactions on Information Theory, vol. IT-10, 1964, pp. 134-138.

[12] Stern (J.). - A method for finding codewords of small weight. In : Coding theory and Applications, éd. par Cohen (G.) et Wolfmann (J.), Lecture Notes in Computer Science, n388. pp. 106-113. - Springer-Verlag, 1989.

[13] Canteaut (A.) et Chabaud (F.). - A new algorithm for finding minimum-weight words in a linear code : application to primitive narrow-sense BCH codes of length 511. IEEE Transactions on Information Theory, 1995.

[14] Canteaut (A.) et Chabaud (F.). - A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. In : EUROCODE 94- Livre des Résumés, éd. par Charpin (P.).- INRIA,1994.

[15] I.I. Dumer. Two decoding algorithms for linear codes. In $5th$ joint Soviet-Swedish Int. Workshop Inform. Theory, pages 50-52, 1991. Moscow, USSR.

[16] I.I. Dumer.soft decision decoding using punctured codes. IEEE Transactions on Information Theory, 2001. to be published, 22 pp.