Utilisation de techniques de codage correcteur d'erreurs pour la cryptanalyse de systèmes de chiffrement à clé secrète

Benoît Gérard Le Bobinnec

Dirigé par Jean-Pierre Tillich

INRIA-projet CODES
Domaine de Voluceau
78153 Le Chesnay

Table des matières

Remerciements

Je souhaite remercier toute l'équipe du Projet CODES de l'INRIA Rocquencourt pour l'ambiance sympathique qui régne dans les locaux et plus spécialement :

- Nicolas Sendrier pour m'avoir accueilli au sein du projet,
- Christelle Guiziou pour l'aide apportée à divers niveaux.
- Jean-Pierre Tillich qui m'a proposé le sujet et m'a trouvé un financement pour poursuivre ce travail. Je tiens à le remercier pour la patience dont ila fait preuve durant ces 6 mois et le fait qu'il m'ait permit de le battre au tennis grâce à un handicap de 30 points par jeu.

Introduction

Dans le cadre du Master II en Algèbre Appliquée de l'Université de Versailles, j'ai réalisé mon stage à l'INRIA, au sein de projet CODES sous la direction de Jean-Pierre Tillich membre permanent du projet.

Ce stage de six mois m'a permis de m'intéresser à l'utilisation du codage dans la cryptanalyse linéaire à plusieurs équations. Le but du stage étant d'avoir une cryptanalyse 'compétitive' du DES à 8 tours afin de pouvoir, ultérieurement, monter une attaque sur 16 tours.

Nous allons tout d'abord présenter les notions de chiffrement symétrique par bloc et de cryptanalyse. Ensuite nous présenterons le DES et la cryptanalyse linéaire.

Comme il est précisé dans le titre du rapport, l'idée est d'utiliser des techniques de décodage afin de réaliser cette cryptanalyse, c'est pourquoi, dans un deuxième temps, nous donnerons quelques notions de théorie des codes correcteurs, les liens avec le problème de la cryptanalyse utilisant plusieurs équations ainsi que l'étude d'un algorithme de décodage pour des codes linéaires aléatoires.

La troisième et dernière partie de ce rapport présentera les résultat obtenus ainsi que le travail restant à faire et les pistes qu'il pourrait être intéressant d'explorer.

Notations

Je présente ici quelques notations récurrentes de ce rapport qui seront, pour la plupart, redéfinies le moment venu.

Codage

- $\mathcal{C}$ est un code linéaire .
- n est la longueur du code considéré.
- k est la dimension du code considéré.
- G est la matrice génératrice du code $\mathcal{C}$.
- H est la matrice de parité du code $\mathcal{C}$.
- C est la capacité du canal.
- Si X est une variable aléatoire, alors $\mathcal{H}(X)$ est son entropie.
- $\mathcal{N}(\mu,\sigma^2)$ est la loi normale d'espérance μ et de variance σ^2.

Cryptanalyse

- n est le nombre d'équations disponibles.
- k est la taille de la clé.
- m est la taille des messages.
- N est le nombre de couples de messages à disposition de l'attaquant.
- P est le message clair, $P \in \mathbb{F}_2^m$.
- C est le message chiffré, $C \in \mathbb{F}_2^m$.
- K est la clé utilisée lors du chiffrement, $K \in \mathbb{F}_2^k$.
- ε^j est le biais de la j-ième équation.
- A un élément de F_2^d, A_i est sa i-ème coordonnée.
- $\oplus$ est le XOR, c'est-à-dire l'addition dans $\mathbb{F}_2$.
- Le produit scalaire de F_2^d est : $\langle A,B\rangle = \bigoplus_{i=0}^{d} A_i.B_i$.
- $A \in \mathbb{F}_2^a$, $B \in \mathbb{F}_2^b$, la concaténation des vecteurs A et B est :

$$A||B \in \mathbb{F}_2^{a+b} \quad , \quad A||B = (A_1 \dots A_a B_1 \dots B_b)$$

Première partie

Cryptanalyse

Chapitre 1

Le DES

1.1 Schémas de Feistel

Le schéma de Feistel est une construction de chiffrement itératif par bloc. L'idée est d'appliquer un certain nombre de fois une étape rapide (on parle de tour) à un bloc de message. Cette étape dépend, en général, de la clé et de l'indice du tour dans l'exécution.

Pour chiffrer un message M de taille m, on le coupe en deux blocs de taille $m/2$ que l'on nomme L_0 et R_0 : $M = (L_0||R_0)$ puis on applique la récurrence suivante :

$$L_{i+1} = R_i \quad , \quad R_{i+1} = L_i \oplus F_{i+1}(R_i,K)$$

F_i est une fonction quelconque de $m/2$ bits vers $m/2$ bits et K est la clé.
De nombreuses études ont été faites afin de connaître le niveau de sécurité atteint en fonction du nombre de tours effectués et, actuellement, 6 tours sont nécessaires pour éviter des attaques génériques sur le schéma.

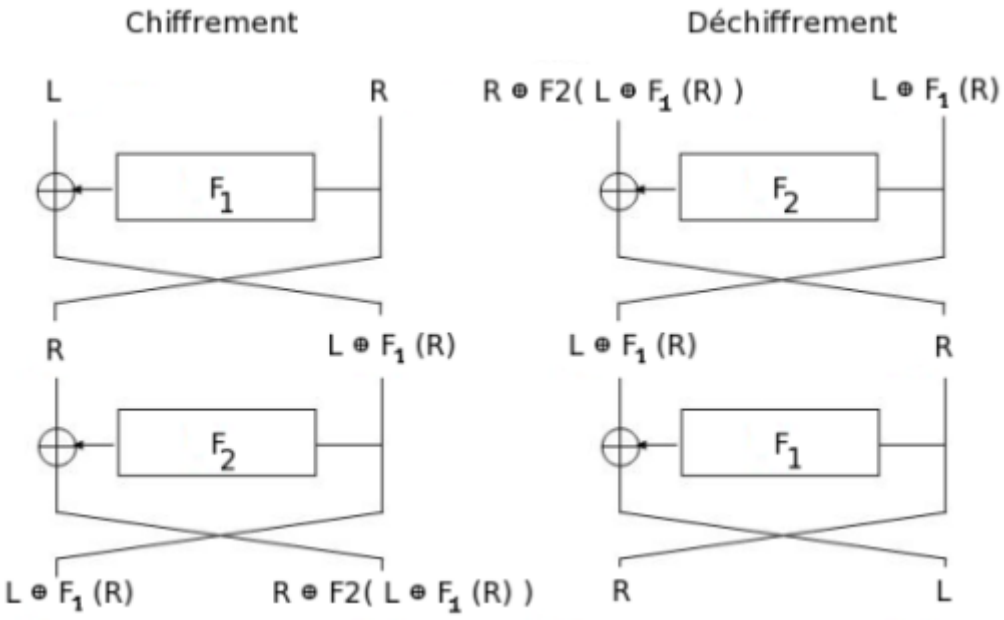
Remarque : Quelque soit la fonction F utilisée, un schéma de Feistel est une bijection mais surtout, son inverse est aussi un schéma de Feistel.
En effet, soit $\pi : \{0;1\}^{m/2} \times \{0;1\}^{m/2} \longrightarrow \{0;1\}^{m/2} \times \{0;1\}^{m/2}$
$(L,R) \longrightarrow (L \oplus F(R,K),R)$

Et $\sigma : \{0;1\}^{m/2} \times \{0;1\}^{m/2} \longrightarrow \{0;1\}^{m/2} \times \{0;1\}^{m/2}$
$(L,R) \longrightarrow (R,L)$

On voit facilement que π et σ sont des involutions.
La fonction tour d'un schéma de Feistel est $G = \sigma \circ \pi$ et donc sa fonction inverse est $G^{-1} = \pi \circ \sigma = \sigma \circ \sigma \circ \pi \circ \sigma = \sigma \circ G \circ \sigma$.
Un exemple pour un schéma à deux tours est donné dans le schéma 1.1

1.2 Spécifications du DES

Le DES est un algorithme de chiffrement à clé secrète qui fut très répandu il y a quelques années de cela. Il a été conçu afin de répondre à un appel d'offre

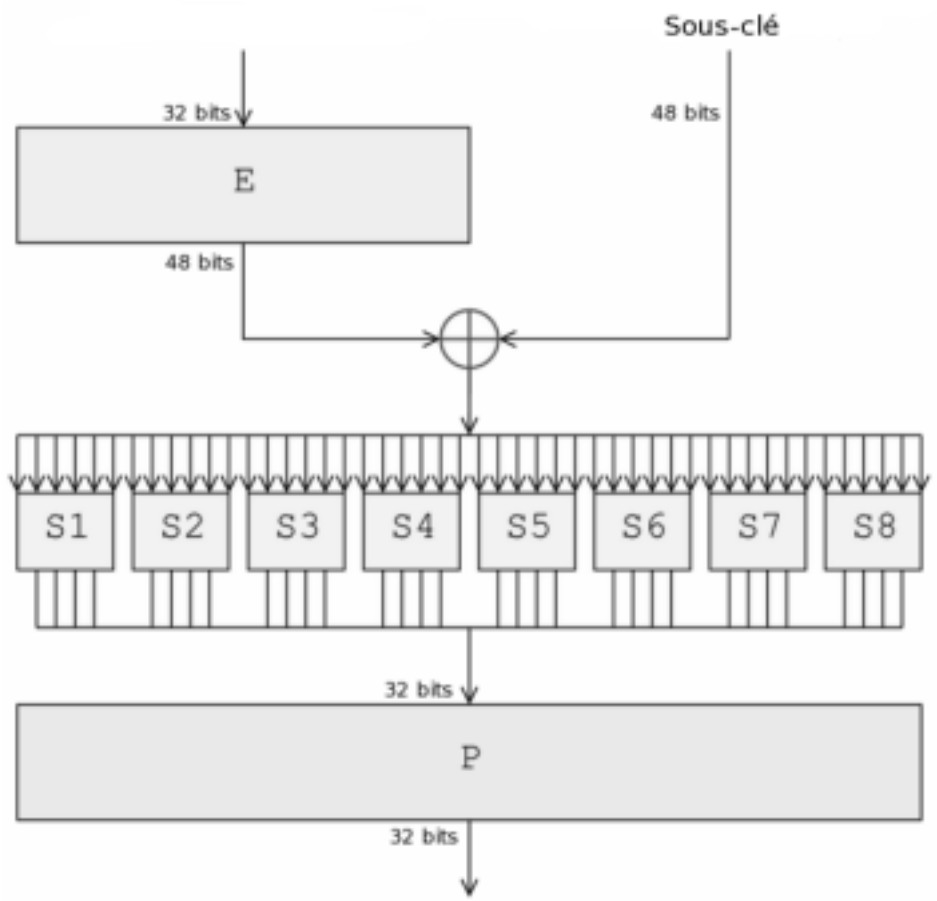


TAB. 1.1 – *Déchiffrement d'un Feistel à 2 tours*

lancé en 1972 par le National Bureau of Standards (NBS). Il a été accepté dans une version utilisant une clé de 56 bits en 1981 et a ensuite été largement utilisé dans le monde.

Cependant, avec sa clé de 56 bits et les progrès technologiques, il est devenu plus facile d'effectuer une cryptanalyse par recherche exhaustive (c'est-à-dire tester toutes les clés possibles). C'est pourquoi il a été remplacé par 3DES qui est l'application de 3 DES avec 3 clés différentes. Il existe maintenant un autre standard : l'AES mais ce n'est pas le sujet de notre étude.

Le DES est un système symétrique itératif par bloc qui suit un schéma de Feistel à 16 tours. La fonction F du DES est représentée sur la figure 1.2. Elle utilise une expansion E, 8 S-BOX S1 ... S8 (fonctions booléennes non linéaires) et une permutation P. Les sous-clés de 48 bits sont extraites de la clé de 56 bits selon des tables prédéfinies.



Tab. 1.2 – *Fonction F du DES*

Chapitre 2

Cryptanalyse linéaire de Matsui

La cryptanalyse linéaire a été inventée par Matsui en 1993 [5]. C'est une attaque à clair connu, c'est-à-dire que l'attaquant connaît un certain nombre de couples message clair/message chiffré. L'idée de Matsui est de mettre en évidence une équation linéaire liant les bits du clair, du chiffré et de la clé qui ait une forte probabilité d'être vérifiée, puis, de l'utiliser comme un distingueur.

Quelques notations :
On note un couple de message clair/chifré (P,C) (P pour 'Plaintext' et C pour 'Ciphertext'), la clé est notée K. P et C sont des vecteurs binaires de longueur m ($m = 64$ dans le cas du DES) et K un vecteur binaire de taille k ($k = 56$ pour le DES). On note P_i la i-ème coordonnée du vecteur P, de même pour C_i et K_i. Une équation linéaire faisant intervenir les bits du clair, du chiffré et de la clé peut s'écrire :

$$P_{i_1} \oplus \cdots \oplus P_{i_h} \oplus C_{i_1} \oplus \cdots \oplus C_{i_j} \oplus K_{i_1} \oplus \cdots \oplus K_{i_k} = c$$

Cette notation étant assez lourde, on utilise la notation de produit scalaire suivante.

$$\langle P,\Pi\rangle \oplus \langle C,\Gamma\rangle \oplus c = \langle K,\kappa\rangle \tag{2.1}$$

Où $b \in \mathbb{F}_2$, $\Pi,\Gamma \in \mathbb{F}_2^m$, $\kappa \in \mathbb{F}_2^k$ et $\langle P,\Pi\rangle = \bigoplus_{1\leq i\leq n} P_i\Pi_i$ et $\langle K,\kappa\rangle = \bigoplus_{1\leq i\leq n} K_i\kappa_i$. Π,Γ et κ sont appelés, respectivements,masques d'entrée, de sortie et de clé.

Supposons, donc, que l'on a trouvé une équation linéaire qui s'avère correcte dans plus de la moitié des cas, on a donc :

$$P\left(\langle P,\Pi\rangle \oplus \langle C,\Gamma\rangle \oplus c = \langle K,\kappa\rangle\right) = \frac{1}{2} + \varepsilon$$

L'attaque étant à clair connu on dispose aussi de N couples clair/chiffré $(P^{(i)},C^{(i)})_i$.

2.1 Première attaque

La première attaque consiste à retrouver un bit d'information sur la clé. Pour cela, on va utiliser un compteur :

$$T = \sum_{1 \leq i \leq N} \langle P,\Pi \rangle \oplus \langle C,\Gamma \rangle \oplus c$$

En d'autres termes, T est égal au nombre de fois que le membre gauche de l'équation (2.1) vaut 1.

- Si $T > N/2$ alors on devine que $\langle K,\kappa \rangle = 1$
- Si $T \leq N/2$ alors on devine que $\langle K,\kappa \rangle = 0$

La probabilité d'avoir raison dépend de ε (qu'on appelle biais de l'équation) et du nombre N de messages. L'algorithme échoue si $T < N/2$.

Pour plus de précision sur la probabilité d'échec, on va faire appel au théorème de Berry-Esséen.

Théorème 2.1.1 *Soient $(X_i)_{1 \leq i \leq N}$ des variables i.i.d., $\mathbb{E}(X_i) = 0$, $\mathbb{E}(X_i^2) = \sigma^2$ et $\mathbb{E}(|X_i|^3) = \rho$. Alors :*

$$\frac{\sum_{i=1}^{N} X_i}{\sigma\sqrt{N}} \underset{N \to \infty}{\longrightarrow} \mathcal{N}(0,1)$$

De plus, si on note $F_N(x)$ la fonction de répartition de la variable $\dfrac{\sum_{i=1}^{N} X_i}{\sigma\sqrt{N}}$ et $\Phi(x)$ celle de la loi normale centrée réduite, alors :

$$\exists C \, , \forall x \, , \forall N \, , \; | \, F_N(x) - \Phi(x) \, | \leq \frac{C\rho}{\sqrt{N}\sigma^3}$$

Ici, T est une somme de N variables de Bernoulli indépendantes de paramètre $p = 1/2 + \varepsilon$ on a donc

$$\frac{T - Np}{\sqrt{Np(1-p)}} \underset{N \to \infty}{\longrightarrow} \mathcal{N}(0,1)$$

La probabilité d'erreur de cet algorithme est donc :

$$P(T < N/2) = P\left(\frac{T - Np}{\sqrt{Np(1-p)}} < \frac{\sqrt{N}(1/2 - p)}{\sqrt{p(1-p)}} \right)$$

En utilisant les notations du théorème, $P(T < N/2) = F_N\left(-\dfrac{\sqrt{N}\varepsilon}{\sqrt{p(1-p)}} \right)$. On a donc un encadrement de la probabilité d'erreur :

$$\left| P(T < N/2) - \Phi\left(-\frac{\sqrt{N}\varepsilon}{\sqrt{p(1-p)}} \right) \right| \leq \frac{C\rho}{\sqrt{N}\sigma^3}$$

Comme nos variables sont des variables de Bernoulli, on a $\sigma^2 = p(1-p)$ et $\rho = p(1-p)(p^2+(1-p)^2)$ et donc :

$$P(T < N/2) = \Phi\left(-\frac{\sqrt{N}\varepsilon}{\sqrt{pq}}\right) + O\left(\frac{1}{\sqrt{N}}\right)$$

Finalement, $pq = 1/4 - \varepsilon^2$ et donc la valeur de $\Phi\left(-\frac{\sqrt{N}\varepsilon}{\sqrt{pq}}\right)$ est proche de celle de $\Phi\left(-2\sqrt{N}\varepsilon\right)$. On voit donc bien que la probabilité d'erreur est fonction du produit $\sqrt{N}\varepsilon$ comme annoncé plus haut.

2.2 Seconde attaque

La seconde attaque va nous permettre de trouver 48 bits de clé plus un bit d'information. Dans la description qui suit, on appelera C^{15} le message chiffré obtenu après 15 trous de DES et C le chiffré final. On note F^{16} la fonction F du 16-ième tour de DES, si la clé utilisée est K alors on a $C^{15} = (F^{16})^{-1}(C,K)$. On a besoin d'une équation sur les 15 premiers tours du DES, on note ε son biais. Voici l'attaque proposée (l'équation a été simplifiée : des termes en C et en F^{16} peuvent apparaître) :

ATTAQUE $2(C^{(i)},P^{(i)},\varepsilon)$

for all $K \in \{0;1\}^{48}$
  do
    $T_K = 0$
    for $i = 1 \ldots N$
      do
        calculer $C^{15} = (F^{16})^{-1}(C^{(i)},K)$
        if $\langle P^{(i)},\Pi\rangle \oplus \langle C^{15},\Gamma\rangle \oplus c = 0$
          then
            $T_K = T_K + 1$
Soit K_{max} la clé maximisant $|T_K - N/2|$
Alors on dit que les 48 bits de sous clé sont ceux de K_{max}
Et $\langle K_{max},\kappa\rangle = 0$ si $T_{K_{max}} - N/2 > 0$, 1 sinon

On n'étudiera pas en détail la probabilité de réussite mais elle dépend aussi de ε et de $\sqrt{N}$.

Dans le papier original [5]. Matsui utilise 2^{21} couples pour le DES à 8 tours et 2^{47} pour le DES à 16 tours. Ce dernier nombre représente 64 Teraoctets ce qui est énorme. Y a-t-il un moyen de réduire le nombre de messages nécessaires ? Peut-on éviter la recherche exhaustive ? C'est ce que l'on va voir dans le chapitre suivant.

Chapitre 3

Utilisations de plusieurs équations

On remarque que l'attaque de Matsui utilise un grand nombre de couples clair/chiffré mais que très peu de bits servent réellement. En utilisant plusieurs équations on a plus d'information pour le même nombre de couples. La question est la suivante, comment exploiter ces informations afin de mener une attaque plus puissante que celle de Matsui ?
Des réponses partielles ont été données dans [4] puis dans [1]. Je vais donc expliquer ce qui est proposé dans ces articles.

3.1 Extraction de l'information

Avec les mêmes notations que dans le chapitre précédent, on a n équations et N couples :

$$\langle P^{(i)},\Pi^{(j)}\rangle \oplus \langle C^{(i)},\Gamma^{(j)}\rangle \oplus c^{(j)} = \langle K^{(i)},\kappa^{(j)}\rangle \qquad , \qquad 1 \leq i \leq N, \qquad 1 \leq j \leq n$$

$$P\left[\langle P^{(i)},\Pi^{(j)}\rangle \oplus \langle C^{(i)},\Gamma^{(j)}\rangle \oplus c^{(j)} = \langle K^{(i)},\kappa^{(j)}\rangle\right] = \frac{1}{2} + \varepsilon^{(j)}$$

Pour chaque couple de messages, on va extraire et condenser l'information qui nous intéresse dans un vecteur :

$$d^{(i)} = (\langle P^{(i)},\Pi^{(1)}\rangle \oplus \langle C^{(i)},\Gamma^{(1)}\rangle \oplus c^{(1)}, \ldots, \langle P^{(i)},\Pi^{(n)}\rangle \oplus \langle C^{(i)},\Gamma^{(n)}\rangle \oplus c^{(n)})$$

Puis on va compter les apparitions des 1 :

$$D = \sum_{i=1}^{N} d^{(i)}$$

On notera **D** la variable aléatoire associée.

L'idée exposée dans [1] est de sortir une liste des clés ordonnées de la plus probable à la moins probable puis d'effectuer une recherche exhaustive dans cet ordre.

On va donc regarder la loi de probabilité du vecteur $\mathbf{D}$ en fonction des valeurs des $\langle K,\kappa^{(j)}\rangle$:

$$P\left[\mathbf{D}_j = D_j \mid \langle K,\kappa^{(j)}\rangle = 1\right] = \binom{N}{D_j} (\frac{1}{2} + \varepsilon^{(j)})^{D_j}.(\frac{1}{2} - \varepsilon^{(j)})^{N-D_j} \tag{3.1}$$

De même,

$$P\left[\mathbf{D}_j = D_j \mid \langle K,\kappa^{(j)}\rangle = 0\right] = \binom{N}{D_j} (\frac{1}{2} + \varepsilon^{(j)})^{N-D_j}.(\frac{1}{2} - \varepsilon^{(j)})^{D_j} \tag{3.2}$$

3.2 Trouver la clé

Nous avons vu comment extraire l'information des couples clair/chiffré. Comment utiliser ce vecteur afin de retrouver la clé ?
On va faire appel à la notion de vraisemblance, la vraisemblance d'une classe K est $P(\mathbf{D} = D \mid \langle K,\kappa\rangle)$ et se calcule à partir des formules 3.1 et 3.2. Les auteurs de [1] utilisent une approximation par une loi normale ce qui rend le calcul plus rapide. Pour chaque clé possible, on va donc calculer sa vraisemblance puis trier la liste et effectuer une recherche exhaustive dans cet ordre.

3.3 Complexité

Il nous faut donc :
- Calculer les valeurs de $P\left[\mathbf{D}_j = D_j \mid \langle K,\kappa^{(j)}\rangle = 0\right]$ pour chaque équation.
- Calculer la vraisemblance de chaque clé possible.
- Trier les clés selon leur vraisemblance.
- Faire la recherche exhaustive dans cet ordre.

On ne s'occuppera pas de la complexité de la dernière étape. On verra plus tard le nombre moyen de clés à tester avant d'arriver à la clé utilisée.

Calul des $P\left[\mathbf{D}_j = D_j \mid \langle K,\kappa^{(j)}\rangle = 0\right]$ Pour chacune des n équations et des N messages, il faut calculer $\langle P^{(i)},\Pi^{(j)}\rangle \oplus \langle C^{(i)},\Gamma^{(j)} \oplus c^{(j)}\rangle$ ce qui représente 4 fois le calcul du poids de Hamming (modulo 2) d'un entier de 32 bits. Il reste ensuite à déduire les probabilités souhaitées grâce aux D_j obtenus. On notera ce coût v, $v = O(n) = o(N)$ car $n = o(N)$ et donc on a un coût de :

$$n(4N + v) = O(nN)$$

Calul des vraisemblances des clés Pour chacune des 2^k clés possibles, il faut calculer la vraisemblance correspondante. Notons t le coût de cette étape, on a donc un coût de :

$$t.2^k = O(2^k)$$

Tri des clés Reste à trier les clés. Pour cela, on connaît des algorithmes de complexité $n \log_2(n)$. Comme le nombre de clés est 2^k on a donc une complexité de :

$$k.2^k = O(k2^k)$$

Au final on a une complexité de $O(nN) + O(k2^k)$. Aucun des deux termes ne peut être, à priori, négligé. En effet, k est égal à 56 pour le DES et les valeurs de n et N dépendent des conditions de l'attaque (probabilité de réussite souhaitée, biais des équations, ...). Cependant, on sait que l'attaque de Matsui avec une équation nécessitait $N = 2^{47}$ pour $n = 1$. On peut donc penser que c'est le terme en $k2^k$ qui dominera en pratique.

L'idée de recherche du mot le plus vraisemblable est une des idées de base du décodage, la question est : peut-on utiliser une technique de décodage de façon à diminuer le nombre de clés regardées? L'estimation du nombre de messages donnée dans [1] semble très pessimiste, peut-on utiliser la théorie de l'information afin d'en trouver une meilleure? C'est ce que nous allons voir après une brève présentation de la théorie des codes correcteurs.

Deuxième partie

Décodage linéaire

Chapitre 1

Introduction à la théorie des codes correcteurs

1.1 Un problème de transmission

La théorie des codes correcteurs a pour objectif de permettre à une personne recevant un message de pouvoir détecter les erreurs de transmission voire de les corriger. Le message est transmis par un canal, le message reçu en sortie peut avoir été modifié durant la traversée du canal, c'est pourquoi on dit que le canal est bruité.
Afin de pouvoir détecter les erreurs de transmission, plusieurs méthodes ont été inventées, les plus simples étant :

- le bit de contrôle : ajout d'un bit égal au XOR des bits du message (comme le dernier nombre du numéro de sécurité sociale)
- la répétition : on répète chaque bit (2n+1) fois et on décode en regardant la valeur majoritaire sur chaque paquet de (2n+1) bits en sortie.

Mais ces techniques simples ne sont pas satisfaisantes. En effet, la première ne fait que détecter un nombre impair d'erreurs et ne permet pas de les corriger. La seconde, elle, est satisfaisante du point de vue de la correction mais utilise $2n+1$ fois plus de bande passante que la transmission du message seul.
Avant d'aller plus loin, commençons par définir l'élément central de la théorie des codes correcteur : le canal.

Définition 1.1.1
Un canal est un système constitué d'un alphabet d'entrée $\mathcal{X}$, d'un alphabet de sortie $\mathcal{Y}$ et d'une famille de distributions de probabilité $(P_x)_{x\in\mathcal{X}}$ où $P_x(A) = P(y \in A \mid \boldsymbol{x} = x)$

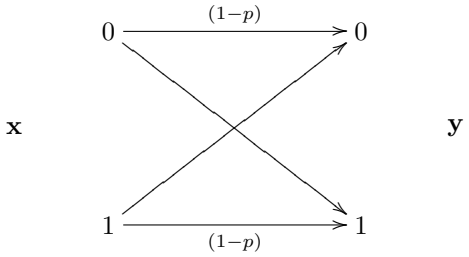
Dans ce rapport, l'alphabet d'entrée sera {0,1}, un tel canal est appelé canal binaire :

Définition 1.1.2
Un canal est dit à entrées discrètes si le cardinal de $\mathcal{X}$ est fini.
Un canal est dit discret si les alphabets $\mathcal{X}$ et $\mathcal{Y}$ sont de cardinal fini.
Un canal est dit binaire si le cardinal de $\mathcal{X}$ est deux.

Un canal est dit sans mémoire si :

$$P(Y \mid X) = \Pi_{i=1}^{n} P(Y_i \mid X_i)$$

Exemple L'exemple de canal le plus simple est le canal binaire symétrique. On dit de ce canal qu'il est symétrique car $P(\mathbf{y} = 1 \mid \mathbf{x} = 0) = P(\mathbf{y} = 0 \mid \mathbf{x} = 1)$. Cette probabilité est la probabilité d'erreur, on la note p. On peut représenter ce canal par le schéma suivant :



Par la suite, on supposera que les canaux traités sont à entrées binaires.

1.2 Codes linéaires

Un code est l'ensemble des mots que l'on s'autorise à transmettre à notre correspondant. L'idée est de les choisir suffisament différents pour pouvoir les reconnaître même avec quelques erreurs.

Définition 1.2.1
Un code (noté $\mathcal{C}$) de longueur n est un sous-ensemble de $\mathcal{X}^n$.
Le rendement R d'un code est la grandeur liée à l'exploitation de la bande passante disponible. Il correspond au nombre de bits utiles par symbole transmis.

$$R = \frac{\log_2 card(\mathcal{C})}{n}$$

Nous allons maintenant pouvoir nous intéresser au cas particulier des codes linéaires. La philosophie des codes correcteurs est d'ajouter de la redondance afin de pouvoir retrouver le mot émis même s'il y a des erreurs de transmission dues au bruit du canal. Cette opération d'ajout de redondance est appelée codage.

Définition 1.2.2
Une application ϕ de $\{0,1\}^k$ dans $\{0,1\}^n$ injective est appellée application de codage.

On peut alors schématiser une transmission ainsi :

$$Z \in \{0,1\}^k \overset{\text{codage}}{\longrightarrow} \phi(Z) = \mathbf{X} \in \mathcal{C} \subset \{0,1\}^n \overset{\text{canal}}{\longrightarrow} \mathbf{Y} \in \mathcal{Y}^n \overset{\text{décodage}}{\longrightarrow} \mathbf{X}' \in \{0,1\}^k$$

Il y a erreur si $X \neq X'$. Ceci nous amène à définir la probabilité maximale d'erreur.

Définition 1.2.3
Pour un canal donné et un code $\mathcal{C}$, on appelle probabilité maximale d'erreur :

$$P_{Err}(\mathcal{C}) = \max_{X \in \mathcal{C}} P(\boldsymbol{X}' \neq \boldsymbol{X} \mid \boldsymbol{X} = X)$$

Lors du décodage, on va regarder si le mot Y reçu appartient au code, et, si ce n'est pas le cas, chercher le mot de code le plus vraisemblable, c'est-à-dire le X' qui maximise $P(\mathbf{X} = X' \mid \mathbf{Y} = Y)$. Le simple fait de savoir si un mot appartient au code ou pas a, à priori, un coût exponentiel en k si on ne fait aucune hypothèse sur le code en question. Cependant, dans le cas où ϕ est une application linéaire, il existe un algorithme déterministe polynomial pour résoudre ce problème.

Définition 1.2.4
On appelle code (binaire) linéaire de dimension k et de longueur n un sous espace vectoriel $\mathcal{C}$ de $\mathbb{F}_2^n$ de dimension k.
Le rendement d'un code linéaire est donc $R = \frac{k}{n}$.

Un paramètre important en codage est la distance minimale d'un code. La distance minimale permet de donner des bornes sur le nombre d'erreurs pouvant être détectées/corrigées.

Définition 1.2.5
La distance minimale d'un code $\mathcal{C}$ est :

$$d(\mathcal{C}) = \min_{X,Y \in \mathcal{C}, X \neq Y} \{d_H(X,Y)\}$$

où $X,Y \in \mathbb{F}_2^n \quad , \quad X = (X_i)_{1 \leq i \leq n} \quad , \quad Y = (Y_i)_{1 \leq i \leq n}$
et d_H est la distance de Hamming :

$$d_H(X,Y) = card\{i | X_i \neq Y_i\}$$

Pour un code linéaire, la distance minimale est égale au poids de Hamming du mot de code non nul de plus petit poids car par linéarité, prendre le minimum sur la distance entre deux mots quelconques du code revient à prendre le minimum sur la distance entre un mot quelconque et le mot nul.

Intéressons nous de nouveau aux applications de codage. Si on a une application ϕ linéaire injective de $\mathbb{F}_2^k$ dans $\mathbb{F}_2^n$, alors on peut lui associer un code de longueur n et de dimension k : $\mathcal{C} = Im(\phi)$. Plusieurs applications peuvent être associées au même code.

Définition 1.2.6
Soit ϕ une application linéaire injective de $\mathbb{F}_2^k$ dans $\mathbb{F}_2^n$ et $\mathcal{C}$ le code associé à ϕ. Une matrice génératrice du code $\mathcal{C}$ est notée G, c'est la matrice associée au codage ϕ. C'est la matrice à k lignes et n colonnes telle que :

$$\phi(Z) = Z.G$$

Les lignes de G forment une base du code $\mathcal{C}$ et réciproquement, une matrice dont les lignes forment une base de $\mathcal{C}$ est une matrice génératrice du code. C'est pourquoi il existe plusieurs matrices génératrices pour un même code linéaire. ϕ étant injective, on peut, par des permutations de colonnes et combinaisons linéaires de lignes, mettre G sous la forme :

$$G = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix} \quad G'$$

On dit alors que G est sous une forme systématique.

Exemple Pour le cas du bit de contrôle, la forme systématique de G est : $\begin{pmatrix} 1 & & & 1 \\ & \ddots & & \vdots \\ & & 1 & 1 \end{pmatrix}$.

Prenons le cas $k = 2$ et donc $n = k + 1 = 3$, alors, $G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$.

Si l'on veut coder le vecteur (a,b), alors $(a,b)G = (a,b,a \oplus b)$. Sous cette forme on voit bien qu'on a d'abord le message puis de la redondance.

1.3 Décodage au maximum de vraisemblance

Maintenant que l'on voit comment ajouter de la redondance à un message se pose la question du décodage. Comment, en ayant reçu un mot bruité, retrouver le message qui a été transmis ?
On va s'intéresser au décodage par maximum de vraisemblance que j'ai déjà évoqué précédemment.

Définition 1.3.1
Un algorithme de décodage au maximum de vraisemblance renvoie le mot de code X' qui maximise la probabilité $P(\boldsymbol{X} = X' \mid \boldsymbol{Y} = Y)$.

Définition 1.3.2
Si on émet le mot $X = (X_i)_{1 \leq i \leq n} \in \mathcal{C}$ et que l'on reçoit $Y = (Y_i)_{1 \leq i \leq n}$, $Y_i \in \mathbb{R}$, la log-vraisemblance du i-ème bit est :

$$\mathcal{L}(i) = \log \left(\frac{P(X_i = 0 \mid Y_i)}{P(X_i = 1 \mid Y_i)} \right)$$

Cette valeur est simple à calculer si on se souvient des formules 3.1 et 3.2. Dans le cadre du décodage au maximum de vraisemblance, on veut pouvoir calculer rapidement la vraisemblance d'un mot de code.

Propriété 1.3.1 *Avec les notations précédentes, on a :*

$$X \text{ maximise } P(\boldsymbol{X} = X \mid \boldsymbol{Y} = Y) \Leftrightarrow X \text{ maximise } \sum_{1 \leq i \leq n} (-1)^{X_i} \mathcal{L}(i)$$

Démonstration
Pour alléger la notation, on notera $P(X \mid Y) = P(\mathbf{X} = X \mid \mathbf{Y} = Y)$. On écrit donc la formule de Bayes ainsi :

$$P(X \mid Y) = \frac{P(Y \mid X).P(X)}{P(Y)} \tag{1.1}$$

$P(Y)$ ne dépend pas de X et sous l'hypothèse que la probabilité de transmission est uniforme (c'est à dire que la probabilité d'être transmis est la même pour tous les mots du code) on a $P(X) = 1/ \mid \mathcal{C} \mid$ ($\mathcal{C}$ étant le code considéré). Donc, si on trouve un mot qui maximise $P(Y \mid X)$ alors il maximise aussi $P(X \mid Y)$.
Si l'on rajoute l'hypothèse que le canal est sans mémoire alors,

$$P(Y \mid X) = \prod_{j=1}^{n} P(Y_j \mid X_j)$$

Comme la fonction logarithme est strictement croissante, maximiser $P(Y \mid X)$ revient à maximiser $\sum \log P(Y_j \mid X_j)$.
On effectue une dernière hypothèse (qui elle non plus n'est pas abusive), $P(X_j = 0) = P(X_j = 1) = \frac{1}{2}$ et alors on peut utiliser la formule de Bayes pour dire que tout ceci est équivalent à maximiser $\sum \log P(X_j \mid Y_j)$:

$$P(X_j \mid Y_j) = \frac{P(Y_j \mid X_j).P(X_j)}{P(Y_j)}$$

La mesure de vraisemblance que nous allons utiliser est, rappelons-le,

$$\sum_{1 \leq j \leq n} (-1)^{X_j} \mathcal{L}(j) =$$

$$\sum_{j, X_j = 0} \log P(X_j = 0 \mid Y_j) - \sum_{j, X_j = 0} \log P(X_j = 1 \mid Y_j) +$$

$$\sum_{j, X_j = 1} \log P(X_j = 1 \mid Y_j) - \sum_{j, X_j = 1} \log P(X_j = 0 \mid Y_j)$$

On remarque que la somme du premier et du troisième terme donne justement $\sum_j \log P(X_j \mid Y_j)$ que l'on va noter $\mathcal{V}$. Quant aux deux autres termes; si l'on note :

$$S_1 = \sum_{1 \leq j \leq n} \log P(X_j = 0 \mid Y_j) \quad , \quad S_2 = \sum_{1 \leq j \leq n} \log P(X_j = 1 \mid Y_j)$$

Alors la somme des termes restants vaut $S_1 + S_2 - \mathcal{V}$ et donc, au final :

$$\sum_{1 \leq j \leq n} (-1)^{X_j} \mathcal{L}(j) = 2\mathcal{V} - S_1 - S_2$$

Comme S_1 et S_2 ne dépendent pas des X_j, maximiser $\mathcal{V}$ est bien équivalent à maximiser notre définition de vraisemblance.

□

1.4 Décodage par syndrome

Maintenant que l'on sait comment calculer la vraisemblance d'un mot de code, encore faut-il avoir un moyen de savoir si un vecteur donné appartient au code ou pas.

Définition 1.4.1
Le code dual du code linéaire $\mathcal{C}$ est :

$$\mathcal{C}^{\perp} = \{A \in \mathbb{F}_2^n \mid \langle A,B \rangle = 0 \, , \forall B \in \mathcal{C}\}$$

où $\langle A,B \rangle$ est le produit scalaire usuel dans $\mathbb{F}_2^n$:

$$\langle A,B \rangle = \bigoplus_i A_i B_i$$

$C^{\perp}$ est aussi un code linéaire, il est de longueur n et de dimension $n-k$, une matrice génératrice H est appelée matrice de parité du code $\mathcal{C}$. Si on a G sous forme systématique, on obtient facilement H :

$$H = \begin{pmatrix} & 1 & \\ -{}^{T}G' & & \ddots \\ & & 1 \end{pmatrix}$$

Remarquons que dans le cas binaire, $-{}^{T}G' = {}^{T}G'$

Définition 1.4.2
Le syndrome d'un élément X de $\mathbb{F}_2^n$ est :

$$S \in \mathbb{F}_2^{n-k} \quad , \quad S = H\,{}^{T}X$$

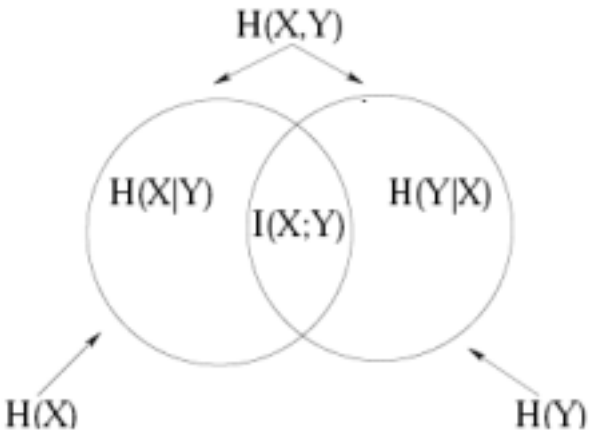
Les vecteurs ligne de H formant une base du code dual, ils sont orthogonaux à tous les mots du code $\mathcal{C}$ ce qui donne les deux propriétés suivantes :

Propriétés du syndrome
- $\forall X \in \mathcal{C}, H\,{}^{T}X = 0$
- si $Y = X + E$, alors $H\,{}^{T}Y = H\,{}^{T}E$

Le problème de décodage revient donc à trouver un motif d'erreur ayant le même syndrome que le mot reçu. En effet, le message reçu est de la forme $\phi(Z) + E$, or, $\phi(Z) \in \mathcal{C}$ donc le syndrome du message reçu est celui de E. Rechercher le mot de code le plus proche du mot reçu est équivalent à chercher le e de poids le plus faible tel que $Y + e \in \mathcal{C}$ et donc il suffit de générer les e par poids croissant et de regarder si le syndrome de $Y + e$ est nul.
Il existe plusieurs algorithmes pour effectuer un décodage par syndrome, on verra quelques astuces pour accélérer ce décodage quand on regardera l'algorithme de Valembois. Dans le cas où le e correspondant à ces critères est différent de E, on dit que le message $Y = X + E$ est indécodable.

1.5 Décodage dur/souple

La différence entre le décodage dur et le décodage souple provient essentiellement de la façon dont est reçu et traité le signal.
Dans le premier cas (décodage dur), on considère que ce qui est reçu est une suite de bits qui contient potentiellement des erreurs.
Dans le cas du décodage souple, on prend en compte la physique de la transmission. Les bits sont en fait modélisés par un signal d'amplitude A. On envoie un signal $(-1)^x A$ où x est le bit que l'on veut transmettre. Après son passage par le canal bruité, on récupère une mesure du signal qui nous permet de calculer la probabilité que le bit émis soit égal à 0 ou 1.
Dans le décodage dur, on coderait chaque mesure positive par 1 et négative par 0. Le décodage souple, lui, garde cette mesure afin d'utiliser l'information de fiabilité qu'elle apporte. En effet, si on mesure $-2A$, on voit que la probabilité que le signal envoyé soit $-A$ est bien plus grande que si l'on reçoit $-A/2$.
En utilisant le décodage souple, on peut connaître la fiabilité de chaque bit et donc la prendre en considération dans l'algorithme.



TAB. 1.1 – *Diagramme récapitulatif*

1.6 Capacité d'un canal

1.6.1 Entropie

L'entropie est une grandeur centrale en théorie de l'information. Moralement, l'entropie d'une variable aléatoire est la quantité moyenne de bits d'information que donne une réalisation de cette variable. Plus formellement :

Définition 1.6.1
Soit $\boldsymbol{X}$ *une variable aléatoire discrète à valeurs dans* $\mathcal{X}$*.*
Pour $X \in \mathcal{X}$ *, on note ,* $P(X) = \mathcal{P}(\boldsymbol{X} = X)$
L'entropie $\mathcal{H}(\boldsymbol{X})$ *de la variable aléatoire* $\boldsymbol{X}$ *est égale à :*

$$\mathcal{H}(\boldsymbol{X}) = -\sum_{X \in \mathcal{X}} P(X)\ \log_2 P(X)$$

Par exemple, si $\mathbf{X}$ est une variable de Bernoulli de paramètre 1, alors l'événement est sûr et $\mathcal{H}(\mathbf{X}) = 0$. Au contraire si $p = 1/2$, $\mathcal{H}(\mathbf{X}) = 1$; c'est pour cette valeur de p que $\mathcal{H}(\mathbf{X})$ est maximum car c'est pour $p = 1/2$ que l'incertitude est la plus grande.
On va maintenant parler de l'entropie jointe de deux variables aléatoires. Cela revient à considérer le vecteur $(\mathbf{X},\mathbf{Y})$ comme une seule variable aléatoire et lui appliquer la définition précédente.

Définition 1.6.2
Soient $\boldsymbol{X}$*et* $\boldsymbol{Y}$*deux variables aléatoires discrètes à valeurs dans* $\mathcal{X}$ *et* $\mathcal{Y}$*.*
$\forall (X,Y) \in \mathcal{X} \times \mathcal{Y}$, $P(X,Y) = P(\boldsymbol{X} = X, \boldsymbol{Y} = Y)$
L'entropie jointe $\mathcal{H}(\boldsymbol{X}, \boldsymbol{Y})$ *des variables aléatoires* $\boldsymbol{X}$ *et* $\boldsymbol{Y}$ *est égale à :*

$$\mathcal{H}(\boldsymbol{X}, \boldsymbol{Y}) = -\sum_{X \in \mathcal{X}} \sum_{Y \in \mathcal{Y}} P(X,Y) \log_2 P(X,Y)$$

On définit aussi l'entropie conditionnelle qui est la quantité d'information moyenne donnée par la réalisation d'une variable connaissant le résultat d'une autre.

Définition 1.6.3
Soient $\boldsymbol{X}$ et $\boldsymbol{Y}$ deux variables aléatoires discrètes à valeurs dans $\mathcal{X}$ et $\mathcal{Y}$. L'entropie conditionnelle $\mathcal{H}(\boldsymbol{X} \mid \boldsymbol{Y})$ est égale à :

$$\mathcal{H}(\boldsymbol{X} \mid \boldsymbol{Y}) = -\sum_{X \in \mathcal{X}} P(X)\, \mathcal{H}(\boldsymbol{X} \mid \boldsymbol{Y} = X),$$

où $\mathcal{H}(\boldsymbol{X} \mid \boldsymbol{Y} = Y) = -\sum_{Y \in \mathcal{Y}} P(\boldsymbol{Y} = Y \mid \boldsymbol{X} = X) \log(P(\boldsymbol{Y} = Y \mid \boldsymbol{X} = X))$

Remarque $\mathcal{H}(\mathbf{X} \mid \mathbf{X}) = 0$.
On peut prouver que :

$$\mathcal{H}(\mathbf{X},\mathbf{Y}) = \mathcal{H}(\mathbf{X}) + \mathcal{H}(\mathbf{Y} \mid \mathbf{X})$$

Ceci est naturel si on interpréte l'entropie comme la quantité d'information donnée par une variable.
Une dernière définition à donner est celle de l'information mutuelle :

Définition 1.6.4
Soient $\boldsymbol{X}$ et $\boldsymbol{Y}$ deux variables aléatoires discrètes à valeurs dans $\mathcal{X}$ et $\mathcal{Y}$. L'information mutuelle $I(\boldsymbol{X}; \boldsymbol{Y})$ est égale à :

$$\begin{aligned} I(\boldsymbol{X}; \boldsymbol{Y}) &= \textstyle\sum_{X \in \mathcal{X}} \sum_{Y \in \mathcal{Y}} P(X,Y)\, log_2 \frac{P(X,Y)}{P(X)P(Y)} \\ &= \mathcal{H}(\boldsymbol{X}) - \mathcal{H}(\boldsymbol{X} \mid \boldsymbol{Y}) = \mathcal{H}(\boldsymbol{Y}) - \mathcal{H}(\boldsymbol{Y} \mid \boldsymbol{X}) \\ &= \mathcal{H}(\boldsymbol{X}) + \mathcal{H}(\boldsymbol{Y}) - \mathcal{H}(\boldsymbol{X}, \boldsymbol{Y}) \end{aligned}$$

Remarque $I(\mathbf{X}; \mathbf{X}) = \mathcal{H}(\mathbf{X}) - \mathcal{H}(\mathbf{X} \mid \mathbf{X}) = \mathcal{H}(\mathbf{X})$
Ces égalités ne seront pas prouvées ici, elles sont là afin de permettre une meilleure compréhension intuitive. J'ai ajouté un diagramme de Venn 1.1 sur lequel on visualise que l'entropie $\mathcal{H}$ se comporte comme la fonction d'aire.. Il justifie parfaitement le nom donné à l'information mutuelle. En effet, on voit que plus celle-ci est élevée, plus les variables $\mathbf{X}$ et $\mathbf{Y}$ sont liées.

1.6.2 Capacité d'un canal

Maintenant que ces notions ont été abordées, on va pouvoir définir la capacité d'un canal.

$$x \in \mathcal{X} \overset{canal}{\longrightarrow} y \in \mathcal{Y}$$

Le canal est défini par une matrice que l'on note $P(x \mid y)$ où chaque ligne correspond à une distribution $P(\mathbf{y} \mid \mathbf{x} = x)$, la matrice a donc $\mid \mathcal{X} \mid$ lignes et $\mid \mathcal{Y} \mid$ colonnes.

Définition 1.6.5
On définit alors la capacité C d'un canal discret sans mémoire par :

$$C = \max_{p(\boldsymbol{x})} I(\boldsymbol{x}; \boldsymbol{y})$$

Où le maximum est pris sur toutes les distributions $p(\boldsymbol{x})$ possibles.

Pour des variables aléatoires continues, on peut définir l'entropie en remplaçant la somme par une intégrale. Le canal qui va nous intéresser est un canal binaire à bruit additif blanc gaussien à entrées binaires.

Définition 1.6.6
Rappelons qu'un canal est définit par un alphabet d'entrée $\mathcal{X}$, un alphabet de sortie $\mathcal{Y}$ et une famille de distributions de probabilité $(P_x)_{x\in\mathcal{X}}$.
Pour un canal binaire à bruit blanc additif gaussien, $\mathcal{X} = \{0,1\}$, $\mathcal{Y} = \mathbb{R}$ et P_0 (resp. P_1) est la distribution d'une loi $\mathcal{N}(1,\sigma^2)$ (resp. $\mathcal{N}(-1,\sigma^2)$).

La convention utilisée pour envoyer le bit b à travers un canal à entrées binaires est de l'encoder par $(-1)^b$. On dit de ce canal qu'il est à bruit blanc additif gaussien car $y = (-1)^x + n$ où n suit une loi normale $\mathcal{N}(0,\sigma^2)$. Le terme 'additif' vient du fait que le bit est ajouté au bit émis, le bruit est 'blanc gaussien' car il suit une loi normale centrée. On va donc pouvoir calculer la capacité de ce canal.

Théorème 1.6.1
Pour un canal à entrées binaires et à bruit blanc additif gaussien de variance σ^2 sans mémoire, la distribution $p(\boldsymbol{x})$ qui maximise $I(\boldsymbol{x},\boldsymbol{y})$ est la distribution uniforme.
On peut calculer sa capacité par l'intégrale suivante :

$$C = \int_{-1}^{1} \frac{\sigma}{\sqrt{2\pi}(1-t^2)} e^{-\frac{(1-\sigma^2 tanh^{-1}(t))^2}{2\sigma^2}} \log_2(1+t)dt \tag{1.2}$$

On a pour cela supposé que l'alphabet d'entrée est $\{-1;1\}$.

Le théorème suivant est un théorème central en théorie des codes correcteurs, il permet de comprendre pourquoi le mot capacité a été employé pour désigner la quantité définie ci-dessus.

Théorème 1.6.2 *(théorème du codage de canal)*
Soient un canal de capacité C, une suite de codes $(\mathcal{C}_n)$ de longueur n, de rendement R et de probabilité d'erreur maximum $P_{Err}(C_n)$.
Alors, on a :

1. $\lim_{n\to+\infty} P_{Err}(C_n) = 0 \Longrightarrow R \leq C$.
2. *Si $R < C$, il existe une suite de codes $(\mathcal{C}_n)$ de rendements $R_n \leq R$ telle que $\lim_{n\to+\infty} P_{Err}(C_n) = 0$.*

Chapitre 2

Utilisation du décodage en cryptanalyse

2.1 Recherche d'équations

Définition 2.1.1 *Soit v un entier positif et $\{u_i,\ 1 \leq i \leq 2^v\}$ l'ensemble des éléments de $\mathbb{F}_2^v$.*
Alors, pour $0 \leq r \leq v$ on peut définir le code de Reed-Muller d'ordre r et de longueur $n = 2^v$ par :

$$\mathcal{C} = \{(f(u_1), \ldots, f(u_n)) \mid f \in \mathbb{F}_2^{(r)}[x_1, \ldots, x_v]\}$$

où $\mathbb{F}_2^{(r)}[x_1, \ldots, x_v]$ désigne l'ensemble des polynômes de $\mathbb{F}_2$, de degré inférieur ou égal à r à v variables .

Soit f une fonction booléenne, $f : \mathbb{F}_2^v \longrightarrow \{0;1\}$, $\mathbf{u}$ le vecteur $(f(u_1), \ldots, f(u_{n-1}))$. Approximer f par un polynôme de degré au plus r revient à chercher parmi tous ces polynômes quel est celui dont la valeur coïncide avec f pour le plus grand nombre d'éléments de $\mathbb{F}_2^v$. Or les polynômes parmi lesquels il faut chercher forment exactement le code de Reed-Muller d'ordre r. Cela revient donc à trouver parmi les mots du code le mot le plus proche de $\mathbf{u}$ c'est-à-dire décoder.

La recherche d'équations vérifiées avec un bon biais va être menée en décodant un code de Reed-Muller sur le canal binaire symétrique. Pour cela nous allons choisir une combinaison linéaire de bits de clé (ie. un masque de sortie Γ). La conbinaison $\langle C,\Gamma\rangle \oplus c$ peut être vue comme une fonction booléenne $F(P,K)$. En décodant le code de Reed-Muller associé, on va trouver la combinaison linéaire de bits de clair et de clé qui approxime le mieux $\langle C,\Gamma\rangle \oplus c$. C'est-à-dire trouver les masques Π et κ qui maximisent $P\left[\langle P,\Pi\rangle \oplus \langle K,\kappa\rangle = \langle C,\Gamma\rangle \oplus c\right]$.

Bien entendu cela n'est pas réalisable en pratique car la longueur du code est bien trop grande (2^{64+56}) . Cependant, il existe des méthodes probabilistes qui, à l'aide d'un échantillon de couples de messages, réussit à obtenir de bons résultats. Je ne rentrerai pas dans les détails de ces opérations ici, pour plus d'informations à ce sujet on peu consulter [7].

2.2 Traitement de plusieurs équations

Voyons maintenant en quoi l'utilisation de techniques de décodage peut améliorer le traitement de plusieurs équations. On va d'abord regarder à quel type de canal correspond cette attaque.

2.2.1 Canal binaire à bruit blanc gaussien

On va montrer que le canal correspondant à la cryptanalyse linéaire peut être modélisé par un canal gaussien. Pour cela, on va faire appel au théorème de Berry-Esséen.

Posons $b = \langle K,\kappa^{(j)}\rangle$ alors la probabilité que $\langle P^{(i)},\Pi^{(j)}\rangle \oplus \langle C^{(i)},\Gamma^{(j)}\rangle \oplus c^{(j)} = 1$ est $p = 1/2 - (-1)^b\varepsilon^{(j)}$, on notera $q = 1 - p$. Le théorème de Berry-Esséen va nous amener à regarder la quantité : $\dfrac{Np - \mathbf{D}_j}{\sqrt{Npq}}$ mais ce n'est pas exactement le terme qui nous intéresse.

$$\frac{\sqrt{N}}{\sqrt{pq}}\left(p - \frac{\mathbf{D}_j}{N}\right) = \frac{\sqrt{N}\varepsilon^{(j)}}{\sqrt{pq}}\left(\frac{N - 2\mathbf{D}_j}{2N\varepsilon^{(j)}} - (-1)^b\right)$$

On a vu que la probabilité d'erreur de l'attaque de Matsui dépend du produit ${N\varepsilon^{(j)}}^2$. On peut s'attendre à ce qu'il en soit de même pour n'importe quelle attaque linéaire. On choisit donc N en fonction de $\varepsilon^{(j)}$ de façon à avoir ${N\varepsilon^{(j)}}^2$ constant. Notons α^2 ce produit.

On a donc :

$$\frac{Np - \mathbf{D}_j}{\sqrt{Npq}} = \frac{\sqrt{\alpha^2}}{\sqrt{pq}}\left(\frac{N - 2\mathbf{D}_j}{2N\varepsilon^{(j)}} - (-1)^b\right)$$

Notons $\phi(t)$ la fonction de répartition de la loi normale centrée réduite et $f_N(t)$ celle de la quantité $\dfrac{Np - \mathbf{D}_j}{\sqrt{Npq}}$. Le théorème de Berry-Esséen nous dit qu'il existe une constante C telle que pour tout x,

$$\left|\int_{-\infty}^{x} f_N(t)dt - \int_{-\infty}^{x}\phi(t)dt\right| \leq \frac{C\rho}{\sqrt{N}\sigma^3}$$

$$\left|\int_{-\infty}^{\sqrt{pq}x/\alpha} f_N(\alpha t/\sqrt{pq})dt - \int_{-\infty}^{\sqrt{pq}x/\alpha}\phi(\alpha t/\sqrt{pq})dt\right| \leq \frac{C\rho}{\sqrt{N}\sigma^3}$$

L'inégalité est vraie pour tout x donc :

$$\left|\int_{-\infty}^{x} f_N(\alpha t/\sqrt{pq})dt - \int_{-\infty}^{x}\phi(\alpha t/\sqrt{pq})dt\right| \leq \frac{C\rho}{\sqrt{N}\sigma^3}$$

$$\left|\int_{-\infty}^{x} \frac{\sqrt{pq}}{\alpha} f_N(\alpha t/\sqrt{pq})dt - \int_{-\infty}^{x}\frac{\sqrt{pq}}{\alpha}\phi(\alpha t/\sqrt{pq})dt\right| \leq \frac{C\rho\alpha}{\sqrt{N}\sigma^3\sqrt{pq}}$$

On reconnaît $\int_{-\infty}^{x} \dfrac{\sqrt{pq}}{\alpha} f_N(\alpha t/\sqrt{pq})dt$ fonction de répartition de

$$\frac{\sqrt{pq}}{\alpha}\frac{Np - \mathbf{D}_j}{\sqrt{Npq}} = \left(\frac{N - 2\mathbf{D}_j}{2N\varepsilon^{(j)}} - (-1)^b\right)$$

et $\int_{-\infty}^{x} \frac{\sqrt{pq}}{\alpha}\phi(\alpha t/\sqrt{pq})dt$ fonction de répartition de la loi normale $\mathcal{N}(0,\frac{pq}{\alpha^2})$.

Et donc :

$$\left(\frac{N-2\mathbf{D}_j}{2N\varepsilon^{(j)}} - (-1)^b\right) \underset{N\to\infty}{\longrightarrow} \mathcal{N}(0,\frac{pq}{\alpha^2}), \tag{2.1}$$

où la convergence est en $1/\sqrt{N}$.

Donc on peut voir la cryptanalyse comme le problème de décodage suivant :

$$K \longrightarrow \langle K,\kappa^{(j)}\rangle \longrightarrow (-1)^{\langle K,\kappa^{(j)}\rangle} \overset{canal}{\longrightarrow} \frac{N-2\mathbf{D}_j}{2N\varepsilon^{(j)}},$$

où le canal est binaire à bruit blanc gaussien de variance $\frac{pq}{N\varepsilon^{(j)^2}} \simeq \frac{1}{4N\varepsilon^{(j)^2}}$.
On remarque que le bruit n'est pas le même pour chaque bit.
On a donc un alphabet de sortie qui a la puissance du continu. C'est une situation dans laquelle il est bénéfique d'utiliser le décodage souple. On va donc utiliser un algorithme de décodage souple à maximum de vraisemblance afin de profiter au mieux de cet avantage.

2.2.2 Code linéaire et matrice génératrice

Dans le cas de la cryptanalyse, l'application de codage est l'application qui à K associe les $\langle K,\kappa^{(j)}\rangle$. On fait ensuite transiter ces valeurs par le canal en les représentant par $(-1)^{\langle K,\kappa^{(j)}\rangle}$. Un produit scalaire étant bilinéaire, l'application $\phi : K \in \mathbb{F}_2^k \longrightarrow \langle K,\kappa^{(j)}\rangle \in \mathbb{F}_2^n$ est donc linéaire et définit bien un code linéaire.
La matrice génératrice associée à ϕ est une matrice de k lignes et n colonnes. Chaque ligne correspond à un bit de clé et chaque colonne à une équation. Le coefficient situé colonne i ligne j vaut 1 si le j-ième bit de clé intervient dans l'équation i, 0 sinon. On a bien, alors, $\phi(K) = K.G$. Nous verrons plus tard les solutions pouvant être apportées au problème de l'injectivité de cette matrice.

Exemple

Les cinq équations suivantes donnent la matrice G ci-dessous :

$$P_1 = K_1 \oplus K_2 \oplus K3 \oplus K4$$

$$P_1 \oplus C_2 = K_1 \oplus K_2 \oplus K4$$

$$P_2 \oplus C_1 \oplus C_2 = K_1 \oplus K_2 \oplus K3$$

$$P_3 \oplus C_1 = K_2 \oplus K3 \oplus K4$$

$$P_2 \oplus C_1 \oplus C_3 = K_2 \oplus K4$$

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

Chapitre 3

Algorithme de Valembois

3.1 Présentation de l'algorithme

3.1.1 Ensemble d'information

Définition 3.1.1 *Soit $I \subset \{0,\ldots,n\}$, $card(I) = k$.*
Soit G une matrice génératrice du code $\mathcal{C}$, on note G_I la sous-matrice formée des colonnes de G d'indice appartenant à I.
I est un ensemble d'information si,

$$rang(G_I) = k$$

Une première idée est donc de se restreindre à un ensemble d'information contenant peu d'erreur, on ne regarde alors que les mots de code ayant au plus t erreurs sur cet ensemble . On souhaite donc que l'ensemble choisi ne comporte que peu d'erreurs. Une façon de procéder est d'effectuer un décodage souple. En effet, avec le décodage souple on a la fiabilité de l'information reçue sur chaque bit et donc on peut tester si l'ensemble des k positions les plus fiables est un ensemble d'information et le choisir le cas échéant. Malgré la fiabilité élevée des positions choisies, le nombre d'erreurs peut être supérieur à t. Le décodage échouera dans les deux cas évoqués ci-dessus. La solution consiste à effectuer le décodage sur plusieurs ensembles d'information et à garder le mot de code le plus vraisemblable.

3.1.2 Résonnance stochastique

Le challenge est donc de trouver plusieurs ensembles d'information contenant peu d'erreurs qui soient suffisamment différents les uns des autres pour que l'on ait un réel intérêt à effectuer plusieurs tours.
La solution donnée par Valembois est l'utilisation de la résonnance stochastique. C'est un phénomène connu des physiciens qui n'avait encore jamais été utilisé en théorie des codes correcteurs. Pour décoder, on va donc appliquer l'algorithme de décodage plusieurs fois pour des ensembles d'information différents. A chaque tour, on va ajouter au vecteur reçu Y un bruit blanc gaussien d'une puissance (variance) égale au quart de celle du bruit du canal.

$$Y_i' = Y_i + N_i$$

Oú N_i est une réalisation de la loi normale $\mathcal{N}(0,\sigma^2/4)$. On va ensuite trier les Y_i' et prendre pour ensemble d'information les k positions les plus fiables, c'est-à-dire les positions correspondant aux $\mid Y_i' \mid$ les plus grands.

3.1.3 Code poinçonné

On va maintenant essayer de diminuer le nombre de motifs d'erreur calculés et de mots de code regardés de façon à accélérer le décodage. Pour l'instant, notre algorithme regarde $O\binom{k}{t}$ mots de code. Une première idée est de couper l'ensemble d'information en deux et de supposer qu'il y a $t/2$ erreurs sur chaque moitié. On regarde alors $O\binom{k/2}{t/2}^2$ mots de code. Le problème c'est que l'on ne regarde plus certains motifs d'erreur qui ont pourtant une bonne probabilité d'être le bon et donc il faut augmenter le nombre de tours pour compenser cette perte. Les deux complexités étant du même ordre pour un t fixé, le coefficient gagné est perdu à cause de l'augmentation du nombre de tours.
Cependant, on peut reprendre cette idée pour faire beaucoup mieux. Pour cela on va utiliser des codes poinçonnés.

Définition 3.1.2 *Soient G la matrice génératrice d'un code $\mathcal{C}$ et $P \subset \{1,...,n\}$, $card(P) > k$. On notera $h = card(P) - k$*
Soit G_P la matrice constituée des colonnes de G dont l'indice appartient à P. Alors le code poinçonné $\mathcal{C}_P$ est le sous espace vectoriel de $\mathbb{F}_2^{k+h}$ égal à l'image de l'application de codage Φ_P définie par la matrice G_P.

Remarque Pour que $\mathcal{C}_P$ soit un code il faut Φ_P soit injective.

Définition 3.1.3 *Soit x un bit émis à travers un canal binaire à bruit blanc gaussien. y est le bit reçu. On appelle quantification de y le bit :*

$$\tilde{y} = \begin{cases} 0 \ si \ y \geq 0 \\ 1 \ si \ y < 0 \end{cases}$$

On va sélectionner plusieurs codes poinçonnés $\mathcal{C}_P$ de la même façon que les ensembles d'information puis, calculer la matrice de parité correspondante H_P. Pour chaque code poinçonné, on va quantifier le vecteur reçu, on obtient donc un mot du code poinçonné : $\tilde{Y}_P'$. L'étape suivante consiste à découper le support de chaque code poinçonné en deux parties de même taille puis chercher les motifs de $t/2$ erreurs ou moins sur chacune des deux moitiés de ce support. On aura donc plus de motifs d'erreur à regarder mais, à la différence de la méthode précédente, tous les motifs ne donneront pas un mot de code. Pour détecter si un mot appartient ou non au code, on a vu que l'on pouvait utiliser une matrice de parité afin de calculer le syndrome d'un élément de $\mathbb{F}_2^n$. Notons m_1 (resp. m_2) un motif d'erreur de poids au plus $t/2$ sur la première (resp. seconde) partie du support et de poids 0 sur le reste. Le but est de trouver des couples (m_1,m_2) tels que :

$$H^T(\tilde{Y}_P' \oplus m_1 \oplus m_2) = 0 \Leftrightarrow s = s_1 \oplus s_2$$
$$s = H^T\tilde{Y}_P' \quad , \quad s_1 = H^T m_1 \quad , \quad s_2 = H^T m_2$$

Cette recherche permet d'effectuer un compromis temps mémoire, ce qui est intéressant vu que l'on n'utilise que peu de mémoire pour le reste des opérations.

On va, pour cela, calculer tous les motifs m de poids inférieur ou égal à $t/2$ sur la première partie du support ainsi que le syndrome correspondant et stocker le motif à l'adresse $s(m) = H^T m$. Le syndrome ayant une longueur h on a donc besoin d'un tableau de taille 2^h. On va ensuite faire les mêmes calculs pour les motifs m d'au plus $t/2$ sur la seconde moitié du support et on va regarder dans la table à l'adresse $s(m) \oplus s$ si il y a un (ou plusieurs) élément(s). Si c'est le cas, alors on a un couple (m_1,m_2) tel que $s_1 \oplus s_2 = s$ et donc $\tilde{Y}'_P \oplus m_1 \oplus m_2 \in \mathcal{C}$.
Il est facile de voir que la complexité en mémoire est $O\binom{(k+h)/2}{t/2}$. Le nombre de mots de code regardé est, quant à lui, en $O\binom{(k+h)/2}{t/2}^2 . \frac{1}{2^h}$.
Ce qui est intéressant c'est d'avoir à peu près 1 élément par case du tableau, ainsi on diminue fortement le nombre de mots de code regardés et on n'augmente pas trop la taille des objets à manipuler ainsi que le nombre de motifs à calculer. C'est pourquoi on prendra h tel que $\binom{(k+h)/2}{t/2} . \frac{1}{2^h} = O(1)$ et donc le nombre de mots de codes regardés est $O\binom{(k+h)/2}{t/2}$ ce qui est donc un gain de temps considérable.

En résumé, G est la matrice génératrice du code, h tel que $card(I) = k + h$ et $s_G(a)$ le syndrome du mot a relativement à la matrice G:

ALGORITHME DE DÉCODAGE PAR RÉSONNANCE STOCHASTIQUE($y = (y_i)_i$, $y_i \in \mathbb{R}$, G , h)

Initialiser le nombre réel $vmax$ et le mot de code res à 0
for j allant de 1 au nombre de tours
  do Ajouter à y_i un bruit de variance $\sigma^2/4$
    Trier les y'_i de la plus fiable à la moins fiable
    Extraire de G les $k + h$ colonnes correspondant aux positions les plus fiables $\rightarrow G_P$
    Quantifier les $k + h$ premières positions de Y' en $\tilde{Y}'_P$
    Calculer la matrice de parité du code poinçonné H_P
    Calculer $s = s_{G_P}(\tilde{Y}'_P) = H_P^T \tilde{Y}'_P$
    Rechercher les motifs de poids 0 ou 1 sur les $\frac{k+h}{2}$ positions les moins fiables $\longrightarrow a_i$
    Stocker les a_i dans un tableau **T** à l'adresse $s_{G_P}(a_i)$
    for b parcourant tous les motifs de poids 0 ou 1 sur les $\frac{k+h}{2}$ positions les plus fiables
      do Regarder les éléments de **T** stockés à l'adresse $s_{G_P}(b) \oplus s$
        On obtient des couples (a,b) tels que $s_{G_P}(a \oplus b) = s_{G_P}(y)$
        On a donc $s_{G_P}(a \oplus b \oplus y) = 0 \Leftrightarrow (a \oplus b \oplus y) \in \mathcal{C}$
        On calcule la log-vraisemblance v de $(a \oplus b \oplus y)$
        if $v > vmax$
          then $res = a \oplus b \oplus y$
            $vmax = v$
Retourner res

3.2 Etude probabilistique du comportement des erreurs

Pour que l'algorithme retourne le bon mot, il faut que le nombre d'erreurs sur P soit au moins une fois strictement inférieur à 3 sur l'ensemble des tours. On peut alors se demander quelle est la probabilité que cela se produise en fonction du nombre de tours effectués. Calculer la probabilité de réussite du premier tour est facile (on va le voir plus loin). Mais il y a un problème : les vecteurs (Y_i') utilisés à chaque tour ne sont pas indépendants. En effet, à chaque tour on calcule les $Y_i' = Y_i + N_i$ où N_i est la réalisation d'une loi normale centrée de variance $\sigma^2/4$. On voit bien qu'il n'y a pas indépendance entre un Y_i' calculé à un tour et un Y_j' calculé à un tour ultérieur. Donc la probabilité de réussite de n tours n'est pas $1 - (1 - p_{\text{réussite d'un tour}})^n$ et risque d'être difficile à calculer.
Les tests dont nous allons parler ont tous été réalisés pour $k = 56$ (en vue de l'application à la cryptanalyse) et pour un bruit de puissance σ^2 tel que le rendement $R < C(\sigma^2)$ où $C(\sigma^2)$ est la capacité du canal gaussien (cf. 1.2).

La première étape a été de regarder comment évolue le nombre d'erreurs en fonction du rendement du code.
Dans l'algorithme de Valembois, on fait l'hypothèse qu'il y a seulement t erreurs sur l'ensemble des positions P (qui est l'ensemble des $k + h$ positions les plus fiables). Pour k et h fixés, on voit que le nombre d'erreurs suit une loi binomiale. Pour $k = 56$ et $h = 6$, les résultats sont présentés dans le tableau 3.1.

Valembois préconise de parier sur un nombre d'erreurs égal à 2, on voit bien que la probabilité de réussite devient assez vite négligeable. De plus, on parie sur une répartition équilibrée des erreurs, c'est-à-dire que l'on parie sur le fait qu'il y ait une erreur par moitié de l'ensemble. Or, les positions ont été triées par ordre de fiabilité.
C'est pourquoi, j'ai regardé la probabilité d'erreur sur le i-ème bit $\mathcal{P}(X_i \neq Y_i)$ où X est le message envoyé et Y le reçu.

Définition 3.2.1 *Soient n variables aléatoires $(A_1, \ldots, A_n)$) continues indépendantes et identiquements distribuées.*
Soient $A_{(1)}, \ldots, A_{(n)}$) les variables aléatoires obtenues en triant les réalisations des A_j par ordre croissant.
La loi de la statistique d'ordre i est la loi de la variable $A_{(i)}$.

Théorème 3.2.1 *Notons $f(x)$ la densité de la loi des A_I.*
La densité de la loi de la statistique d'ordre i est :

$$f_{X_{(i)}}(x) = nf(x) \binom{n-1}{i-1} P(X_1 \leq x)^{i-1} P(X_1 \geq x)^{n-i} \tag{3.1}$$

On rappelle que le mot émis X est un vecteur composé de 1 ou -1. Pour calculer la probabilité d'erreur, on va supposer que les bits envoyés sont tous des 1. Cette hypothèse n'influence pas le résultat car le canal est symétrique. Le mot émis traverse ensuite le canal bruité et donc, le mot reçu Y contient des éléments suivant une loi $\mathcal{N}(1,\sigma^2)$. Comme on utilise l'algorithme de résonnance stochastique, on va ajouter un bruit de variance $\sigma^2/4$ et donc les Y_i' suivent

une loi $\mathcal{N}(1,\frac{5}{4}.\sigma^2)$. On obtient ainsi les Y'_j que l'on trie par ordre décroissant selon les valeurs absolues. On note π l'application de tri. La probabilité d'erreur que nous allons calculer est la probabilité que la quantification du i-ème bit soit fausse pour un tour donné.

$$\mathcal{P}(X_{\pi^{-1}(i)} \neq \tilde{Y}'_{\pi^{-1}(i)}) = \binom{n-1}{i-1}.n.\int_{\infty}^{0} f(x).\left[1+F(x)-F(-x)\right]^{i-1}.\left[F(-x)-F(x)\right]^{n-i}dx \tag{3.2}$$

où $f(x)$ est la densité de la loi normale $\mathcal{N}(1,\frac{5}{4}.\sigma^2)$ et $F(x) = \int_{-\infty}^{x} f(t)dt$.

En effet, la densité de la loi du i-ème bit est la densité de la i-ème plus grande valeur, et donc :

$$f_{X_{(i)}}(x) = nf(x)\binom{n-1}{i-1}P(|X_1| \geq x)^{i-1}P(|X_1| \leq x)^{n-i}$$

$$P(|X_1| \geq x) = 1 + F(x) - F(-x) \text{ et } P(|X_1| \leq x) = F(-x) - F(x)$$

On a supposé que l'on a envoyé 1 et donc il y a erreur si $Y'_i < 0$, c'est pourquoi il faut intégrer cette densité jusqu'à 0. On a donc bien une probabilité d'erreur égale à l'intégrale 3.2.

Mais intégrale est très difficile à calculer directement (même en s'aidant de logiciels de calcul). Cependant, elle a le bon goût d'être très concentrée et positive et l'utilisation de la méthode des trapèzes donne de bons résultats.

Grâce à cela, on peut maintenant calculer la probabilité que les erreurs aient telle ou telle répartition. Connaissant ce comportement, il parait intéressant d'en tirer profit dans l'algorithme (ce qui est loin d'être le cas). C'est le sujet de la section suivante.

3.3 Modifications

L'idée est d'exploiter le fait de connaître le comportement des erreurs pour un bruit donné. En essayant d'augmenter le nombre d'erreurs t à 4, je me suis aperçu de la lenteur de l'algorithme pour le peu d'amélioration que cela apportait. C'est pourquoi j'ai voulu essayer $t = 3$. Les bits étant triés par ordre de fiabilité, le nombre d'erreurs dans la seconde partie est souvent supérieur au nombre d'erreurs dans la première. Pour $t = 3$, l'algorithme va donc chercher les motifs ayant une erreur dans la première moitié et deux dans la seconde. Quelques tests montrent que pour un temps donné ces deux algorithmes ont à peu près le même taux de réussite. Pour un taux de réussite fixé, $t = 2$ est plus rapide si le rendement est faible et c'est $t = 3$ qui prend la tête pour un taux élevé.
Ces simulations sont longues et le nombre de paramètres à prendre en compte est grand. C'est pourquoi ces tests ne sont pas suffisants. De plus, l'implémentation réalisée en début de stage peut certainement être encore un peu optimisée. Il sera donc intéressant d'effectuer une nouvelle batterie de tests afin de savoir si

Rendement	Espérance	Variance	Bruit (max)
1/2	4.470	4.244	1.000
1/3	5.966	5.590	1.681
1/4	7.286	6.748	2.401
1/5	8.380	7.548	3.121
1/8	10.551	8.932	5.281
1/10	11.894	9.914	6.722
1/20	15.115	11.665	13.932
1/30	17.088	12.412	21.144
1/40	18.320	12.807	28.356
1/50	19.277	13.182	35.570
1/60	20.064	13.337	42.783

TAB. 3.1 – *Paramètres de la loi binomiale par rendement*

le fait d'utiliser $t = 3$ peut être intéressant ou non.

Voyant que les modifications apportées par l'utilisation de $t = 3$ pouvaient se révéler pertinentes, j'ai voulu aller plus loin. C'est pour cela que je me suis intéressé au comportement des erreurs. J'ai alors reconstruit l'algorithme, mais cette fois ci, en séparant l'ensemble en 4 parties non forcément égales. On a ainsi toute notre liberté pour répartir les erreurs sur les différentes parties. On peut alors choisir des paramètres qui permettent de bons taux de réussite avec un très petit nombre de motifs calculés. Cependant, comme l'on coupe le support en 4, il faut chercher des quadruplets de motifs d'erreur partiels qui donnent le bon syndrome. Voici comment cela peut se faire, à noter que le temps pris par cette recherche est plus important que le temps gagné en diminuant le nombre de motifs calculés.

3.3.1 Recherche de collisions sur 4 listes

On a 4 listes de syndromes partiels et il faut trouver les combinaisons qui donnent le syndrome du mot reçu.
Pour cela, j'ai utilisé une méthode présentée dans [2]. Je vais la présenter en pseudo-langage :
s est le syndrome reçu, l'ensemble des $(k+h)$ positions est séparé en 4 quarts Q_i sur lesquels on autorise N_i erreurs. On notera $s(m)$ le syndrome du vecteur m. $\overset{stocker}{\longrightarrow}$ signifie que l'on stocke le vecteur en question dans la case dont le numéro correspond au syndrome de ce vecteur. En cas de collision, on utilise une liste chaînée.

RECHERCHEMOTIFS(s,Q_1,Q_2,Q_3,Q_4,N_1,N_2,N_3,N_4)
1 Créer T_3 et T_4 tables de hachage de taille 2^h
2 Génerer les motifs de N_3 erreurs sur $Q_3 \overset{stocker}{\longrightarrow} T_3$
3 Génerer les motifs de N_4 erreurs sur $Q_4 \overset{stocker}{\longrightarrow} T_4$

```
4   for i = 0..2^h − 1
5       do
6           for m parcourant les motifs de N_1 erreurs sur Q_1
7               do
8                   Chercher dans T_3 à l'adresse s(m) ⊕ i les vecteurs stockés u
9                   Stocker dans une liste chainée L_1 les vecteurs m ⊕ u
10          for m parcourant les motifs de N_2 erreurs sur Q_2
11              do
12                  Chercher dans T_4 à l'adresse s(m) ⊕ i ⊕ s les vecteurs stockés v
13                  Stocker dans une liste chainée L_2 les vecteurs m ⊕ v
14          for a ∈ L_1,b ∈ L_2
15              do
16                  a ⊕ b est un vecteur de syndrome s
```

Pour un i donné,

- la liste L_1 contient des éléments de type $m \oplus u$ tels que $s(m \oplus u) = i$.
- la liste L_2 contient des éléments de type $m \oplus v$ tels que $s(m \oplus v) = i \oplus s$.

C'est pourquoi $\forall a \in L_1, \forall b \in L_2$ on a $s(a \oplus b) = s$ avec $a \oplus b$ contenant N_j erreurs sur Q_j.

3.3.2 Discussion

L'algorithme de Valembois est plus performant que la version dans laquelle on sépare l'ensemble d'information en 4, du moins quand on est dans le cas $R < C$. On verra dans la dernière partie de ce rapport pourquoi j'ai quand même utilisé la version modifiée pour effectuer la cryptanalyse du DES.
Pour ce qui est des modifications sur t, il faudra mieux comparer $t = 2$ et $t = 3$, et pourquoi pas essayer de couper l'ensemble d'information en deux parties de façon un peu dissymétrique. Une autre idée serait de générer tous les motifs d'erreur de poids inférieur à un certain t en utilisant un poids généralisé prenant en compte la fiabilité des bits, on n'aurait plus à se poser la question du découpage. Reste à savoir si on gagnerait suffisament pour compenser le temps passé à générer ces motifs.

Troisième partie

Résultats

Chapitre 1

Equations

1.1 Equations trouvées

Rappelons que pour utiliser l'algorithme de décodage des codes de Reed-Muller, il faut fixer un masque de sortie. La difficulté est de donc de trouver des masques de sortie donnant de bonnes approximations (sur 8 tours on veut un biais d'ordre 10^{-4} au moins). On a donc commencé par regarder l'équation utilisée par Matsui dans sa cryptanalyse linéaire [5]. Nous avons donc obtenu un masque de sortie qui, après décodage, nous a fourni 10 équations. Nous avons ensuite utilisé le masque de l'équation de Kalisky et Robshaw et avons obetenu 14 équations.
Au final nous avons donc obtenu 24 équations mais il s'avère que cela ne suffit pas pour avoir de bons résultats pour la cryptanalyse.
C'est alors que nous avons utilisé la symétrie du DES. Comme il est dit dans la section 1.1, un schéma de Feistel est facilement inversible. Dans le cas du DES, la permutation finale fait que la seule modification à faire pour déchiffrer est de prendre les sous-clés dans l'ordre inverse. Un masque d'entrée peut donc être utilisé comme masque de sortie. on trouvera alors au moins une équation de biais équivalent plus d'autres équations parmi lesquelles peuvent se trouver des équations ayant un biais intéressant. Nous avons réussi à ce jour à trouver 135 équations de biais compris entre 0.000145 et 0.000615 (la somme des carrés de ces biais est de 10^{-5}). Mais on ne peut pas utiliser directement ces 135 équations, c'est le sujet de la section suivante.

1.2 Indépendance des équations

Nous modélisons le canal correspondant à la cryptanalyse par un canal sans mémoire. Cela suppose que les différentes équations utilisées soient indépendantes deux à deux. Ceci n'est pas une évidence car certaines combinaisons de bits se retrouvent d'une équation à une autre.

Calcul des biais Nous allons expliquer ici comment calculer le biais des équations trouvées.
Il s'agit de prendre un échantillon suffisamment grand de couples de messages

chiffrés avec des clés différentes et de regarder combien de fois les équations se réalisent. Notons N le nombre de messages de l'échantillon et
$D_j = \sum_{i=1}^{N} \langle P^{(i)},\Pi^{(j)} \rangle \oplus \langle C^{(i)},\Gamma^{(j)} \rangle \oplus \langle K^{(i)},\kappa^{(j)} \rangle \oplus c^{(j)}$.
D_j est égal au nombre de couples pour lesquels l'équation j est vérifiée.
La probabilité de tirer un couple qui vérifie l'équation j dans l'échantillon choisit est $\frac{D_j}{N}$ et donc le biais de l'équation sur cet échantillon est $\frac{D_j}{N} - \frac{1}{2}$. Plus la taille de l'échantillon est grande, plus le biais de l'équation sur l'échantillon est proche de son biais réel. En statistique, on utilise en général un échantillon ayant une taille d'ordre de $O(1/\varepsilon^2)$. Je ne rentrerais pas dans les détails mais les estimations des biais ont été effectués dans des conditions donnant une précision à 10^{-6} près avec une probabilité de 0.95.

Calcul des covariances Dire que le canal est sans mémoire revient à faire une hypothèse d'indépendance entre les D_j. On va donc calculer les covariances $Cov(\mathbf{D}_{j_1},\mathbf{D}_{j_2})$. Pour ce faire, on prend chaque élément de l'échantillon $(P^{(i)},C^{(i)},K^{(i)})_{1\leq i\leq n}$ puis on regarde pour chaque équation si elle est vérifiée par cet élément. On obtient ainsi les $D_j = \sum_{i=1}^{N} \langle P^{(i)},\Pi^{(j)} \rangle \oplus \langle C^{(i)},\Gamma^{(j)} \rangle \oplus \langle K^{(i)},\kappa^{(j)} \rangle \oplus c^{(j)}$ comme pour le calcul des biais. On va aussi regarder, pour tout $(j_1,j_2)_{1\leq j_1<j_2\leq n}$, si cet élément vérifie à la fois l'équation j_1 et l'équation j_2. D_{j_1,j_2} est égal au nombre d'éléments de l'échantillon qui vérifient à la fois l'équation j_1 et l'équation j_2.
On peut alors exprimer la covariance par :

$$Cov(\mathbf{D}_{j_1},\mathbf{D}_{j_2}) = D_{j_1,j_2}/N - D_{j_1}D_{j_2}/N^2$$

Les calculs pour les 135 équations trouvées ont été effectués pour des tailles croissantes d'échantillon, les résultats montrent que les corrélations sont négligeables.

1.3 Traitement des équations

Tout d'abord, il faut savoir que ces 135 équations ne mettent pas en jeu chacun des 56 bits de clé. Le résultat de la cryptanalyse ne nous donnera donc qu'un certain nombre de bits de clé et non pas les 56. Cependant, une fois un certain nombre de bits de clé fixés, une recherche exhaustive sur le reste des bits est négligeable dans le calcul de la complexité de l'attaque. Notons k' le nombre de bits de clé intervenant dans les équations regardées. On a donc une matrice génératrice du code qui a n colonnes et k' lignes. Le problème est qu'il arrive que la matrice obtenue soit de rang $k'' < k'$. Le code alors généré est de dimension k''. Plutôt que d'obtenir des informations sur k' bits de clé, on obtient alors des informations sur k'' combinaisons de bits de clé. Cela se produit quand un bit de clé intervient de la même manière qu'un autre dans les équations traitées .

Exemple simple
Supposons que l'on ait les cinq équations suivantes :

$$P_1 = K_1 \oplus K_2 \oplus K3 \oplus K4$$

$$P_1 \oplus C_2 = K_1 \oplus K_2 \oplus K4$$

$$P_2 \oplus C_1 \oplus C_2 = K_1 \oplus K3$$
$$P_3 \oplus C_1 = K_2 \oplus K3 \oplus K4$$
$$P_2 \oplus C_1 \oplus C_3 = K_2 \oplus K4$$

Cela nous donne la matrice génératrice suivante :

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

On voit que les lignes 2 et 4 sont égales. Il faudra alors utiliser la matrice

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

Où la deuxième ligne correspondra à $K_2 \oplus K_4$. On a donc perdu un bit d'information par rapport à ce que l'on pouvait espérer.

Un dernier problème peut apparaître, dans l'algorithme, on utilise des codes poinçonnés. Un code poinçonné est un code qui, normalement, a la même dimension que le code de départ. Cependant, si on prend une base du corps de départ et que l'on ote certains bits des mots de la base, il se peut que l'ensemble forme une famille liée.
Dans notre cas, on obtient une matrice de dimension 20 avec 131 équations. Or, deux groupes d'équations se distinguent, le premier fait intervenir 9 bits et le second 13. Ces deux groupes n'ont donc en commun que deux bits de clé. On a donc une matrice avec une grande quantité de 0 et quand on fait tourner l'algorithme, on se retrouve souvent avec une matrice poinçonnée qui contient une ligne de 0. Il est donc encore très fréquent que l'algorithme ne renvoie rien. On a donc isolé les deux groupes d'équations mis en évidence et on a appliqué le décodage à chacun de ces sous-groupes. Pour le moment, on arrive à de meilleurs résultats qu'en utilisant les 131 équations mais cela pourrait changer avec l'arrivée de suffisamment d'équations ne rentrant dans aucun des deux groupes.
Cependant, le fait d'utiliser deux groupes d'équations pourrait être intéressant avec l'utilisation du décodage itératif. Ces deux groupes d'équations ont 3 bits de clé en commun, c'est-à-dire qu'il y a trois bits de clé qui interviennent à la fois dans les équations du groupe 1 et du groupe 2. On peut alors effectuer un décodage pour le premier groupe, récupérer une liste des clés partielles les plus vraisemblables et en déduire la probabilité que chacun des 3 bits communs soit égal à 0. On effectue ensuite le décodage sur le second groupe avec cette information sur les trois bits communs. On peut donc espérer de meilleurs résultats en utilisant cette méthode.

Chapitre 2

Cryptanalyse

2.1 Etude du nombre de messages nécessaires

Dans [1], une etude est faite sur le nombre de clés testées en moyenne par l'algorithme avant de trouver la bonne. Les auteurs s'intéressent donc à la taille de la liste des clés plus vraisemblables que la clé utilisée. Notons K la véritable clé utilisée. Par additivité de l'espérance, l'espérance de la taille de la liste est égale à la somme sur les clés K' de la probabilité que K' soit plus vraisemblable que K. Pour une clé $K' \in \mathbb{F}_2^k$, on note $\mathcal{D}_{K'}$ l'ensemble des compteurs $D \in \{0,\dots,N\}^n$ pour lesquels K' est plus vraisemblable que K. Le nombre moyen de clés plus vraisemblables que K est alors :

$$E = \sum_{K' \in \mathbb{F}_2^k - \{K\}} \sum_{D \in \mathcal{D}_{K'}} P(\mathbf{D} = D \mid \mathbf{K} = K)$$

Notre algorithme de décodage renvoie la clé la plus vraisemblable, c'est pourquoi on va s'intéresser à la probabilité d'erreur de l'algorithme c'est-à-dire la probabilité que la clé renvoyée par l'algorithme ne soit pas la clé utilisée.

On se place sur un canal à bruit blanc gaussien de variance σ^2. On note $\mathbf{X}$ la variable aléatoire correspondant au mot envoyé, $\mathbf{Y}$ celle correspondant au mot reçu. On envoie X à travers le canal bruité, on reçoit Y que l'on décode en X'. Le canal étant symétrique, la probabilité d'erreur est la même quel que soit le mot du code qui a été envoyé. On supposera donc que l'on a envoyé le mot nul de façon à simplifier les calculs. La probabilité qu'un mot de code soit plus vraisemblable que le mot émis ne dépend alors que du poids du mot en question ce qui m'amène à définir le polynôme énumérateur d'un code $\mathcal{C}$.

Définition 2.1.1 *Le polynôme énumerateur d'un code de longueur n est le polynôme :*

$$A(D) = \sum_{w=0}^{n} A_w D^w$$

A_w est le nombre de mot de code de poids de Hamming égal à w.

La probabilité qu'un mot de code soit plus vraisemblable que le mot émis (le mot nul dans notre cas) est :

$$P_E = P(\max_{X' \in \mathcal{C}-\{0\}} P(\mathbf{Y} = Y \mid \mathbf{X} = X') \geq P(\mathbf{Y} = Y \mid \mathbf{X} = 0) \mid \mathbf{X} = 0)$$

Par un simple argument ensembliste, on peut majorer ceci par la somme :

$$P_E \leq \sum_{X' \in \mathcal{C}-\{0\}} P(P(\mathbf{Y} = Y \mid \mathbf{X} = X') \geq P(\mathbf{Y} = Y \mid \mathbf{X} = 0) \mid \mathbf{X} = 0)$$

On note $hw(X)$ le poids de Hamming de X, la probabilité ne dépend que du poids du code et donc on peut écrire :

$$P_E \leq \sum_{w=1}^{n} A_w P(P(\mathbf{Y} = Y \mid hw(\mathbf{X}) = w) \geq P(\mathbf{Y} = Y \mid \mathbf{X} = 0) \mid \mathbf{X} = 0)$$

Par un calcul standard (changement de variable), l'intégrale multi-dimensionnelle correspondant à cette probabilité est égale à $\left(1 - \Phi\left(\frac{\sqrt{w}}{\sigma}\right)\right)$ où $\Phi\left(\frac{\sqrt{w}}{\sigma}\right)$ s est la fonction de répartition de la loi normale centrée réduite.
Pour x positif, on va borner l'intégrale suivante :

$$\begin{aligned}
\int_x^\infty \frac{1}{\sqrt{2\pi}}.e^{-\frac{y^2}{2}}\,dy &= e^{-\frac{x^2}{2}} \int_x^\infty \frac{1}{\sqrt{2\pi}}.e^{\frac{x^2-y^2}{2}}\,dy \\
&= e^{-\frac{x^2}{2}} \int_x^\infty \frac{1}{\sqrt{2\pi}}.e^{-\frac{(y-x)(y+x)}{2}}\,dy \\
&\leq e^{-\frac{x^2}{2}} \int_x^\infty \frac{1}{\sqrt{2\pi}}.e^{-\frac{(y-x)^2}{2}}\,dy \\
&\leq e^{-\frac{x^2}{2}} \int_0^\infty \frac{1}{\sqrt{2\pi}}.e^{-\frac{(y)^2}{2}}\,dy \\
&\leq \frac{1}{2} e^{-\frac{x^2}{2}}
\end{aligned}$$

Et donc,

$$P_E \leq \frac{1}{2} \sum_{w=1}^{n} A_w e^{-\frac{w}{2\sigma^2}} = \frac{1}{2}\left[A\left(e^{-\frac{1}{2\sigma^2}}\right) - A_0\right]$$

On va chercher à expliciter les A_w. Un code peut être défini par une matrice de parité (n-k lignes, n colonnes). Si on tire les coefficients de cette matrice de façon aléatoire dans $\mathbb{F}_2$, alors on peut définir un code associé. Ce code est alors un code aléatoire. Pour un élément non nul de $\mathbb{F}_2^n$, la probabilité d'appartenir à un tel code est 2^{k-n} car il y a 2^k mots de codes. Quant au mot nul, par linárité, il appartient ùa tous les codes linéaires. L'espérance des A_w est donc $\binom{n}{w} 2^{k-n}$ et,

$$A(D) = \sum_{w=0}^{n} A_w D^w = 1 + \sum_{w=1}^{n} A_w D^w = 1 + \sum_{w=1}^{n} \binom{n}{w} 2^{k-n} D^w$$

$$A(D) = 1 - 2^{k-n} + 2^{k-n} \sum_{w=0}^{n} \binom{n}{w} D^w = 1 + 2^{k-n}((1+D)^n - 1)$$

Ce qui donne finalement,

$$P_E \leq \frac{1}{2}\left(2^{k-n}(1+e^{-\frac{1}{2\sigma^2}})^n - 1\right) = \frac{\left(1+e^{-\frac{1}{2\sigma^2}}\right)^n - 1}{2^{n-k+1}} \leq \frac{1}{2}\left(\frac{1+e^{-\frac{1}{2\sigma^2}}}{2^{1-R}}\right)^n$$

On peut par ailleurs montrer que le terme trouvé est en fait égal à l'espérance E à un coefficient polynomial prêt. Or, il s'avère que pour certains paramètres, la probabilité que la liste soit vide est très grande et, avec une probabilité exponentiellement basse, la taille de cette liste est exponentiellement grande. Dans ce cas, il peut arriver que la quantité exponentielle de clés plus vraisemblables l'emporte sur la probabilité faible que l'événement se réalise. On aura alors une espérance élevée qui nous fera croire que le décodage ne peut être correctement effectué alors que la clé renvoyée sera la bonne dans une grande majorité des cas. C'est pourquoi on va regarder une autre façon d'évaluer le nombre de messages nécessaires.

Cette autre méthode est d'utiliser les notions d'entropie et de capacité. On note $\mathbf{K}$ la variable aléatoire associée au vecteur $(\langle K,\kappa^{(j)}\rangle)_{1\leq j\leq n}$ et $\mathbf{D} = (\mathbf{D}_j)_{1\leq j\leq n}$ la variable de comptage définie au premier chapitre. On a alors :

$$I(\mathbf{K};\mathbf{D}) = \mathcal{H}(\mathbf{D}) - \mathcal{H}(\mathbf{D} \mid \mathbf{K})$$

Comme le canal est sans mémoire, $\mathcal{H}(\mathbf{D} \mid \mathbf{K}) = \sum_{j=1}^{n} \mathcal{H}(\mathbf{D}_j \mid \mathbf{K}_j)$. L'inégalité suivante est standard en théorie de l'information :

$$\mathcal{H}(\mathbf{D}) \leq \sum_{j=1}^{n} \mathcal{H}(\mathbf{D}_j)$$

Et donc :

$$I(\mathbf{K};\mathbf{D}) \leq \sum_{j=1}^{n} \mathcal{H}(\mathbf{D}_j) - \sum_{j=1}^{n} \mathcal{H}(\mathbf{D}_j \mid \mathbf{K}_j) \tag{2.1}$$

$$\leq \sum_{j=1}^{n} I(\mathbf{K}_j;\mathbf{D}_j) \tag{2.2}$$

$$\leq \sum_{j=1}^{n} C(\sigma_j^2) \tag{2.3}$$

$$\tag{2.4}$$

La capacité $C(\sigma_j^2)$ est la capacité du canal gaussien pour un bruit de variance $\sigma_j^2 = \dfrac{1}{4N\mathcal{E}^{(j)^2}}$ (cf. 1.2).

On peut donc écrire l'inégalité suivante :

$$\mathcal{H}(\mathbf{K} \mid \mathbf{D}) = \mathcal{H}(\mathbf{K}) - I(\mathbf{K}; \mathbf{D}) \tag{2.5}$$

$$\geq \mathcal{H}(\mathbf{K}) - \sum_{j=1}^{n} C(\sigma^{(j)}) \tag{2.6}$$

$$\geq k - \sum_{j=1}^{n} C(\sigma^{(j)}) \tag{2.7}$$

$$\tag{2.8}$$

Avoir $\mathcal{H}(\mathbf{K} \mid \mathbf{D})$ petit signifie que l'on a beaucoup d'informations sur $\mathbf{K}$ avec $\mathbf{D}$. Pour trouver la clé, on veut donc avoir $\mathcal{H}(\mathbf{K} \mid \mathbf{D})$ le plus près de 0 possible. On va donc devoir prendre $\sum_{j=1}^{n} C(\sigma^{(j)})$ ègal à $\mathcal{H}(\mathbf{K}) = k$.

Exemple Prenons un code binaire linéaire aléatoire de rendement 1/2. On veut regarder pour quelles variances du bruit le décodage peut s'effectuer correctement. Le théorème de capacité d'un canal dit que c'est le cas si $R < C$. On peut être plus précis, si $R \geq C$ alors la probabilité d'erreur se rapproche de 1 à une vitesse exponentielle en la longueur du code n. Si $R < C$, alors cette probabilité décroît exponentiellement quand n augmente.
On va donc obtenir deux valeurs limites de σ^2. On notera σ_E^2 la valeur obtenue avec le calcul de l'espérance et σ_C^2 celle obtenue grâce à l'étude de la capacité du canal.

σ^2	0	$\sigma_E^2 = 0.567$	$\sigma_C^2 = 0.958$
Espérance (E)	$O(\alpha^n)$ $\alpha < 1$	|	$O(\beta^n)$ $\beta > 1$
Probabilité d'erreur (P_E)	$O(\gamma^n)$ $\gamma < 1$		

Dans le cadre de la cryptanalyse, σ^2 est inversement proportionnel au nombre de messages N. Avoir une borne supérieure sur σ^2 nous donne donc une borne inférieure sur N. Ce tableau nous montre clairement que la borne obtenue avec le calcul de l'espérance est plus pessimiste que celle obtenue par le raisonnement sur la capacité du canal.

2.2 Algorithme de décodage utilisé

Le but de l'utilisation de plusieurs équations est principalement de faire diminuer le nombre de couples nécessaires. On va donc essayer de décoder dans une zone où $R > C$.
L'algorithme de Valembois est alors peu performant, même en augmentant le nombre de tours, on n'arrive pas à atteindre le taux de réussite de la version modifiée. Pour la cryptanalyse, il sera donc intéressant de prendre cet algorithme

modifié.
Une autre question est : en quoi l'utilisation d'un algorithme de décodage est-elle plus intéressante que la méthode exposée dans [1] ?
On peut facilement modifier l'algorithme de décodage pour lui demander de renvoyer une liste des l clés partielles les plus vraisemblables et ensuite effectuer une recherche exhaustive dans cet ordre. On obtient alors un algorithme qui effectue la cryptanalyse de la même manière que l'algorithme [1]. La différence est que ce dernier va calculer les vraisemblances des 2^k clés alors que l'algorithme de décodage ne va calculer la vraisemblance que d'une fraction des clés. En contrepartie, peut être que certaines clés étant parmi les plus vraisemblables ne seront pas regardées. On a donc une probabilité que la clé utilisée ne soit pas renvoyée par l'algorithme car l'algorithme n'aura pas calculé sa vraisemblance. Cependant, il est possible de faire un choix de paramètres tel que cette probabilité soit très faible, le gain de temps restant conséquent.

2.3 Résultats obtenus

On a donc obtenu deux groupes d'équations. Le premier contient 55 équations qui font intervenir 9 bits de clé. Le second en contient 76 qui font intervenir 13 bits de clé. Il y a aussi la possibilité d'utiliser presque toutes les équations trouvées. On a alors un ensemble de 131 équations qui font intervenir 20 bits de clé. Le problème est que cette matrice contient beaucoup de zéros et que donc la matrice poinçonnée n'est quasiment jamais une application injective. Avec les équations dont nous disposons, il est plus intéressant d'effectuer deux fois la cryptanalyse avec les deux ensembles que de le faire une fois avec les 131 équations.

Complexité Nous allons présenter les résultats obtenus sur le DES à 8 tours. Ces résultats sont au dessus de ce que la borne de la capacité laissait espérer.
Pour ce qui est du temps d'exécution, il est très faible car le nombre de bits de clé considérés est faible. Il est cependant intéressant de regarder ce que nous fait gagner l'utilisation d'un algorithme de décodage.
Lors d'une cryptanalyse, on a vu que ce qui prend le plus de temps est de générer un mot de code et de calculer sa vraisemblance. On a pu observer, grâce à un 'profiler', que c'est aussi le cas pour l'algorithme de décodage. Dans l'algorithme de [1], on effectue l'opération pour les 2^k mots de code. L'utilisation d'un algorithme de décodage permet de diminuer cette quantité.
Notons v le nombre de fois que l'algorithme calcule un mot de code puis sa vraisemblance. Pour l'algorithme [1] $v = 2^k$ et pour notre algorithme, les différentes valeurs de v sont données dans les tableaux ci-dessous.
On voit que l'économie de temps est très intéressante. De plus, comme les taux d'erreur obtenus sont en dessous des taux prévus, cela signifie que le gain de temps n'altère presque pas la qualité du résultat.
Ceci est très important car lorsque l'on aura des équations sur le DES à 16 tours, on aura beaucoup plus de bits intervenant dans les équations et donc ce gain de temps sera loin d'être négligeable.

2.3.1 Premier groupe

Ce groupe génère un code de longueur 55 et dimension 9 la capacité donne la borne suivante sur le nombre de messages nécessaires :

$$N \geq 2^{19.49}$$

Voici les résultats que l'on obtient :

Nombre de couples	Taux de réussite (%)	v
2^{16}	6	171
2^{17}	18	169
2^{18}	26	157
2^{19}	48	132
2^{20}	70	93

2.3.2 Second groupe

Ce groupe génère un code de longueur 76 et dimension 13 la capacité donne la borne suivante sur le nombre de messages nécessaires :

$$N \geq 2^{19.84}$$

Voici les résultats que l'on obtient :

Nombre de couples	Taux de réussite (%)	v
2^{16}	0	430
2^{17}	0	448
2^{18}	2	415
2^{19}	20	283
2^{20}	48	230

Précisions Il y a deux raisons aux fluctuations de v.
La première est que le nombre de tests effectués n'est pas très grand car générer un grand nombre de couples clair/chiffré demande du temps. Le nombre de tests est suffisant pour avoir une estimation du taux d'erreur au pourcent près mais pas suffisant pour avoir une valeur de v aussi précise.
La seconde est que j'ai utilisé des paramètres différents pour chaque cas. Par exemple, pour 2^{20} messages le bruit est plus faible que pour 2^{16}. Il est donc inutile de regarder autant de motifs d'erreur dans ce cas.

Chapitre 3

Conclusion

On a vu dans ce rapport que les techniques de décodage sont un outils puissant pour la cryptanalyse linéaire. Durant ce stage, une attaque sur le DES à 8 tours a été montée. Une attaque Rapide nécessitant seulement 2^{18} couples de messages.
Cette cryptanalyse s'effectue en deux étapes. La première est la recherche d'approximations linéaires ayant un biais intéressant (d'ordre 10^{-4}). Une des techniques connues pour trouver ces équations est le décodage des codes de Reed-Muller. Pour pouvoir utiliser cette méthode, il faut avoir à sa disposition de bons masques de sortie , c'est-à-dire des masques de sorties qui donneront des approximations ayant un bon biais. On a donc utilisé les sorties déjà connues (Matsui) puis il a fallu trouver d'autres équations. C'est le fait d'utiliser les masques d'entrée en temps que masques de sortie qui a fait décoller la situation nous permettant d'obtenir un nombre d'équations suffisant pour effectuer la cryptanalyse du DES à 8 tours.
La seconde étape est de traiter les données fournies par les messages et les équations. On a vu que l'utilisation d'un algorithme de décodage souple permettait de réduire significativement la complexité de la cryptanalyse. De plus, l'utilisation de l'algorithme de Valembois généralisé permet de s'approcher du taux d'erreur minimal de notre cryptanalyse car il est plus adapté au décodage dans les conditions $R > C$.

Le prolongement de ce travail serait une attaque sur le DES complet. La difficulté est de trouver des équations sur le DES à 16 tours. En effet, vu que le nombre de tour double, on s'attend à obtenir des équations de biais d'ordre $10^{-4^2} = 10^{-8}$. Or, l'algorithme probabiliste de décodage des codes de Reed-Muller a une complexité en $1/\varepsilon^2$. L'utilisation de cet algorithme semble donc compromise, d'autant plus que l'on aura aussi affaire à des problèmes de mémoire. Une première solution est de travailler sur cet algorithme en espérant règler ces problémes et donc l'utiliser pour les 16 tours du DES. Une seconde est de chaîner les équations obtenues sur 8 tours. Dans tous les cas, on s'attend à ce qu'il existe un nombre important d'équations sur le DES à 16 tours ayant des biais d'ordre 10^{-8} et donc, on s'attend à pouvoir faire passer le nombre de messages nécessaires sous les 2^{36}.

Bibliographie

[1] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In *CRYPTO '*04, pages 1–22, 2004.

[2] Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks : An algorithmic point of view. In *EUROCRYPT '*02, pages 209–221, 2002.

[3] T. M. Cover and J. A. Thomas. *Elements of information theory.* John Wiley & Sons Inc., New York, 1991. A Wiley-Interscience Publication.

[4] Burton S. Kaliski Jr. and M. J. B. Robshaw. Linear Cryptanalysis Using Multiple Approximations. In *CRYPTO '94*, pages 26–39, London, UK, 1994. Springer-Verlag.

[5] Matsui M. Linear Cryptanalysis Method for the DES Cipher. In *EUROCRYPT '93*, pages 386–397, London, UK, 1993. Springer-Verlag.

[6] Hill Raymond. *A first course in coding theory.* Oxford University Press, Oxford, 1986.

[7] Cédric Tavernier. *Testeurs, problèmes de reconstruction univariés et multivariés, et application à la cryptanalyse du DES.* PhD thesis, Ecole Polytechnique, 2004.

[8] Antoine Valembois. *Détection, Reconnaissance et Décodage des Codes Linéaires Binaires.* PhD thesis, Université de Limoges, 2000.