Cryptographie à clé secrète

Thierry P. Berger [1]

22 novembre 2014

1. UFR des Sciences de Limoges, 123 av. A. Thomas, 87060 Limoges CEDEX, FRANCE
e-mail: thierry.berger@unilim.fr tel: (33) 5 55 45 73 38

Sources: François Arnault, "Théorie des Nombres et Cryptographie"
Anne Canteaut, "La Cryptographie symétrique" et http://www.picsi.org/
Claude Carlet, "Cours de cryptographie"
Marine Minier, http://www.picsi.org/

Table des matières

Chapitre 1

Introduction

1.1 Chiffrement par bloc et chiffrement à flot

Objectif : garantir la confidentialité de messages.

Le chiffrement symétrique se caractérise par le fait qu'on utilise la même clé pour chiffrer ET déchiffrer. Cette méthode de chiffrement est aussi appelée chiffrement à clé secrète ou privée.

Le terme symétrique illustre le fait que la même clé est utilisée pour chiffrer et déchiffrer un message à chaque extrémité de la liaison (c'est à dire par les deux utilisateurs du système qui désirent communiquer de façon confidentielle). Le terme secret rappelle quand à lui que dans ce mode de chiffrement la clé est unique et doit être gardée secrète par ses utilisateurs. Cette méthode se distingue des méthodes de chiffrement à clés publiques, appelée aussi chiffrement asymétrique, qui utilisent des paires de clés publiques, clés privées.

Ce type de chiffrement se divise en deux catégories :

- le chiffrement par bloc qui consiste à traiter des blocs de données de taille fixe.
- le chiffrement à flot qui consiste à traiter les données bit à bit. Le chiffrement à flot utilise souvent un générateur pseudo-aléatoire, dont la sortie est xorée avec le message à chiffrer (one-times pad).

1.2 Un exemple de chiffrement par bloc : le DES

Voir Annexe A

1.3 Un exemple de chiffrement à flot : RC4

Voir Annexe B

Sécurité. La plupart des attaques connues sur RC4 exploitent des faiblesses de la phase d'initialisation. Ainsi, plusieurs attaques par distingueur ont été proposées, notamment par Mantin et Shamir [3]. De façon générale, les premiers mots de la suite générée sont particulièrement vulnérables, et il est généralement conseillé de ne pas rendre en compte les 512 premiers octets produits [2].

Il n'existe à ce jour aucune attaque par recouvrement de clef sur RC4 sauf dans des contextes particuliers. Par exemple, Fluhrer, Mantin et Shamir [1] ont montré qu'en l'absence de valeur initiale, le mécanisme de re-synchronisation utilisé dans le norme WEP IEEE 802.11 conduisait à l'utilisation de la même suite chiffrante pour chiffrer différents messages. Le niveau de sécurité offert par RC4 avec un protocole de re-synchronisation de ce type est donc extrêmement faible. Plus généralement, on considère que RC4 est particulièrement sensible aux attaques liées au protocole de réinitialisation.

Propriété intellectuelle. RC4 est un algorithme propriétaire de la société RSA Data Security, Inc.

1.4 Fonction de hachage

On nomme fonction de hachage une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une empreinte servant à identifier rapidement, bien qu'incomplètement, la donnée initiale. Les fonctions de hachage sont utilisées en informatique et en cryptographie.

Les fonctions de hachage servent à rendre plus rapide l'identification des données : calculer l'empreinte d'une donnée ne doit coûter qu'un temps négligeable. Une fonction de hachage doit par ailleurs éviter autant que possible les collisions (états dans lesquels des données différentes ont une empreinte identique) : dans le cas des tables de hachage, ou de traitements automatisés, les collisions empêchent la différenciation des données ou, au mieux, ralentissent le processus.

En cryptographie les contraintes sont plus exigeantes et la taille des empreintes est généralement bien plus longue que celle des données initiales ; un mot de passe dépasse rarement une longueur de 8 caractères, mais son empreinte peut atteindre une longueur de plus de 100 caractères. La priorité principale est de protéger l'empreinte contre une attaque par force brute, le temps de calcul de l'empreinte passant au second plan.

Fonctions de hachage cryptographiques Une fonction de hachage cryptographique est utilisée entre autres pour la signature électronique, et rend également possibles des mécanismes d'authentification par mot de passe sans stockage de ce dernier. Elle doit être résistante aux collisions, cest-à-dire que deux messages distincts doivent avoir très peu de chances de produire la même signature. De par sa nature, tout algorithme de hachage possède des collisions mais on considère le hachage comme cryptographique si les conditions suivantes sont remplies :

- il est très difficile de trouver le contenu du message à partir de la signature (attaque sur la première préimage) à partir d'un message donné et de sa signature,
- il est très difficile de générer un autre message qui donne la même signature (attaque sur la seconde préimage)
- il est très difficile de trouver deux messages aléatoires qui donnent la même signature (résistance aux collisions) Par très difficile, on entend techniquement impossible en pratique , par toutes techniques algorithmiques et matérielles, en un temps raisonnable.

SHA-1 : un exemple de fonction de hachage à partir d'une fonction de compression

Principe du chaînage : on utilise une fonction de compression dans un schémas de chaînage :

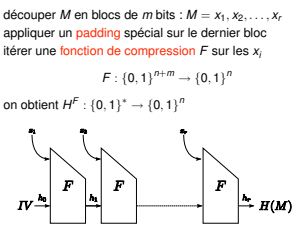


FIGURE 1.1 – Chaînage de Merckle-damgard

Le message m est d'abord "paddé" comme suit :
$M = m||10\ldots0||\ell$ où ℓ est la longueur en binaire de m sur 64 bits, et le nombre de 0 ajoutés est tel que la longueur de M est un multiple de 512 bits.

On a alors $M = M_1||M_2||\dots||M_n$ où les M_i sont des blocs de 512 bits
Algorithme SHA-1 :

Entrée : Entier n et blocs M_1 à M_n de longueur 512.
Sortie : Empreinte de 160 bits.

$K_0 := 5A827999$, $K_1 := 6ED9EBA1$, $K_2 := 8F1BBCDC$, $K_3 := CA62C1D6$
$A := 67452301$, $B := EFCDAB89$, $C := 98BADCFE$, $D := 10325476$, $E := C3D2E1F0$
Pour i de 1 à n répéter
  Pour j de 0 à 15 définir W_j par $M_i = W_0 \,\|\, \cdots \,\|\, W_{15}$
  Pour j de 16 à 79 répéter
    $W_j := (W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16}) \ll 1$
  $A' := A$, $B' := B$, $C' := C$, $D' := D$, $E' := E$
  Pour j de 0 à 79 répéter
    $t := \lfloor i/20 \rfloor$
    $E' := A' \ll 5 + f_t(B', C', D') + E' + W_j + K_t$
    $(A', B', C', D', E') := (E', A', B' \ll 30, C', D')$
  $A := A + A'$, $B := B + B'$, $C := C + C'$, $D := D + D'$, $E := E + E'$
Retourner $A \,\|\, B \,\|\, C \,\|\, D \,\|\, E$

où les additions notées $+$ s'entendent modulo 2^{32} et où le symbole $\ll$ désigne une rotation à gauche. Quand aux f_t, ce sont les fonctions booléennes suivantes, définies sur des mots de 32 bits :

$$f_0(B, C, D) = (B \wedge C) \vee (\overline{B} \wedge D)$$
$$f_1(B, C, D) = f_3(B, C, D) = B \oplus C \oplus D$$
$$f_2(B, C, D) = (B \wedge C) \vee (C \wedge D) \vee (D \wedge B)$$

Bibliographie

[1] S. Fluhrer, I. Mantin, and A. Shamir. " Weaknesses in the Key Scheduling Algorithm of RC4 ". In *Selected Areas in Cryptography - SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2001.

[2] C. Gehrmann and M. Naslund. " ECRYPT Yearly Report on Algorithms and Keysizes ". `www.ecrypt.eu.org/documents/D.SPA.10-1.1.pdf`, 2005.

[3] I. Mantin and A. Shamir. " A practical attack on broadcast RC4 ". In *Fast Software Encryption - FSE 2001*, volume 2335 of *Lecture Notes in Computer Science*, pages 152–164. Springer-Verlag, 2001.

[4] R. Rivest. " The RC4 encryption algorithm ". RSA Data Security, 1992.

Chapitre 2

Fonctions de hachage

2.1 Définition et objectifs de sécurité

Définitions)

Définition 1 *Une fonction de hachage h est une application de l'espace des messages $\mathcal{M} = \{0,1\}^*$ (dans la pratique, il y a souvent une limite N sur la taille des messages) à valeur dans l'espace des empreintes $\mathcal{E} = \{0,1\}^n$ tel qu'il existe un algorithme "rapide" pour calculer h.*
En cryptographie, on demande en plus

- *h à sens unique : étant donné un haché $y \in \mathcal{E}$, il est difficile de trouver une message $x \in \mathcal{M}$ tel que $h(x) = y$.*
- *résistance aux collisions : il est difficile de trouver 2 message x et x' ayant le même haché ($h(x) = h(x')$.*
- *résistance à la seconde pré-image : étant donné un message x et son haché $y = h(x)$, il est difficile de trouver un second message x' tel que $h(x') = y$.*

La résistance à la seconde pré-image entraîne la résistance aux collisions.

Attaque naïve sur la fonction à sens unique :

Entrée : $e \in E$
Sortie $m \in \mathcal{M}$ tel que $h(m) = e$.

Répéter
Tirer m au hasard dans $\mathcal{M}$ jusqu'à $h(m) = e$
Retourner m.

Sous l'hypothèse d'équi-répartition du choix de e et de m, la probabilité de succès est $1/2^n$. Le coût moyen de l'attaque est donc 2^n.

En posant $e = h(m_1)$, on obtient un algorithme de recherche de seconde préimage en 2^n.

Pour les collisions, on obtient un coût moyen de $2^{n/2}$ en utilisant le paradoxe des anniversaires.

Objectifs de sécurité L'objectif de sécurité d'une fonction de hachage est donc que le coût d'une attaque de la propriété sens-unique doit être au moins 2^n et de la collision au moins $2^{n/2}$.

2.2 Fonctions de hachage itératives

Principe du hachage itératif La plupart des fonctions de hachage connues (par exemple la famille MD-SHA) utilisent une fonction de compression f de $\{0,1\}^m$ dans $\{0,1\}^n$, avec $n < m$.

Le message est "paddé", de manière à obtenir une longueur totale divisible par $m-n$, puis divisé en blocs de longueur $m-n$: M_1, M_2, ..., M_ℓ. L'empreinte $h(x)$ est alors calculée de la manière suivante :

- $H_0 := IV$, (IV (Initial Value) est une constante dans $\{0,1\}^n$.
- $H_i := f(x_i||H_{i-1})$ pour $i := 1$ à ℓ
- $h(x) := H_\ell$.

Fonctions de compressions à partir d'algorithmes de chiffrement par blocs : on utilise une fonction de chiffrement par bloc où la taille des clés est égale à la longueur des blocs et assez grande (typiquement 160 bits). voici des exemples de constructions :

- $f(x||k) = e_k(x) \oplus x$
- $f(x||k) = e_k(x) \oplus x \oplus k$
- $f(x||k) = e_k(x \oplus k) \oplus x$
- $f(x||k) = e_k(x \oplus k) \oplus x \oplus k$

Précaution de Damgård et Merkle : on ajoute à la fin du message à compresser des informations sur sa longueur :

$$X = x||1000\ldots000||\ell$$

où ℓ est la longueur de x exprimé sur 64 bits, et où le nombre de 0 avant le 1 est choisi de manière à ce que la longueur totale de X soit divisible par $m-n$.

Théorème 1 (Damgård - Merkle) *Soit f une fonction de compression de $\{0,1\}^m$ dans $\{0,1\}^n$, avec $n+1 < m$ résistante aux collisions. Si les messages sont complétés en un nombre entier de blocs de longueur $m-n-1$ par un procédé réversible, alors on obtient une fonction de hachage résistante aux collisions en posant*
$H_1 := f(0^{n+1}||x_1), \quad H_i := f(H_{i-1}||1||x_{i-1})$, *pour $i \geq 2$.*

Attaque du long message Il s'agit de trouver une seconde pré-image d'un message très long $M \in \mathcal{M}$:

Entrée $M = M_1||..||M_\ell$, formé de ℓ blocs de taille $n' = m-n$.
Sortie : M' tel que $h(M') = h(M)$.
Structure de données : une table (initialement vide) d'empreintes

---

$h_0 := IV$
Pour $i := 1$ à ℓ faire
$h_i := f(M_i, h_{i-1})$
Si h_i est déjà dans la table, alors
Soit $j < i$ tel que $h_j = h_i$
$M' := M_1||\ldots||M_j$
$M_{i+1}||\ldots||M_\ell$
Retourner M' et Terminer
Sinon mémoriser h_i Fin pour
Répéter
Choisir un bloc M'_1 au hasard
$h'_1 := f(M'_1, IV)$
Rechercher si h'_1 est dans la table Jusqu'à trouver h'_1 dans la table
Soit j tel que $h'_1 = h_j$ $M' := (M'_1$
$M_{j+1}||\ldots||M_\ell$
Retourner M'.

Le coût de la première boucle est de ℓ itérations de la fonction f. Le nombre moyen d'itérations de la seconde boucle est $2^n/\ell$. Le coût total est donc $\ell + 2^n/\ell$. Il est optimisé pour $\ell = 2^{n/2}$ pour un coût total de l'attaque(en temps et mémoire) de $2^{(n+1)/2}$ itérations de h.

2.3 Modèle de l'éponge

Modèle de l'oracle aléatoire

Un oracle aléatoire est une "boite noire", qui à une question, répond de manière aléatoire, mais réponds toujours pareille à la même question.

On peut fabriquer un oracle aléatoire de la manière suivante :

On considère l'ensemble R des réponses possibles (considérée ici comme un ensemble fini). Lorsqu'on soumet une nouvelle question à l'oracle

- S'il s'agit d'une nouvelle question, il tire au hasard la réponse avec une loi uniforme sur l'ensemble des réponses. Il stocke alors la question et sa réponse.
- Si la question lui a déjà été posée, il recherche la réponse en mémoire.

Oracle aléatoire en tant que fonction de hachage

Une fonction de hachage sur un alphabet $\mathcal{A}$ prend en entrée un mot fini sur $\mathcal{A}$ et retourne un haché dans $\mathbb{F}_2^n$. On peut donc supposer qu'une bonne fonction de hachage est indistinguable d'un oracle aléatoire.

La plupart des fonctions de hachage utilisée (et utilisable) travaille sur un chaînage des données qui permet de travailler sur une chaîne de valeurs modifiée itérativement par une fonction qui prend en argument le message. On peut alors hacher un message sans le stocker préalablement en mémoire.

Un processus itératif conduit à des collisions de l'état interne i.e. de la valeur de chaînage. En particulier, si on a une collision M_1 et M_2, alors par concaténation $M_1|N$ et $M_2|N$ est une autre collision, alors que dans le modèle de l'oracle aléatoire, $M_1|N$ et $M_2|N$ n'ont aucune raison particulière d'être une collision.

Les "Sponge Functions", ou fonctions éponge cherchent à modéliser le processus de chaînage.

Fonctions éponge

Une éponge fonctionne comme un oracle aléatoire : il prend en entrée un message de longueur variable et en sortie une séquence de taille donnée n (potentiellement infinie).

Pour définir une éponge, on a besoin

- d'un alphabet fini $\mathcal{A}$ muni d'une structure de groupe additif et d'un élément neutre 0. Typiquement $\mathcal{A}$ est $\mathbb{F}_2$, ou bien $\mathbb{F}_2^k$ muni de l'addition coordonnées par coordonnées. k est souvent la taille des mots machine (8, 1, 32, 64 bits...). On pose $r = log_2(\mathcal{A})$ le taux de l'éponge.
- d'un ensemble $\mathcal{C}$ fini qui représente l'état interne de l'éponge. On pose $c = log_2(\mathcal{C})$ la capacité de l'éponge. En général, c est un entier (automate binaire)
- D'un "élément neutre" $0 \in \mathcal{C}$ qui est simplement un état initial de l'automate (pas forcément nul, prends en compte les IV).
- d'une fonction $p(m)$ qui est une fonction injective de l'ensemble des messages dans l'ensemble des chaînes de caractères de $\mathcal{A}$ de telle manière que $|p(m)$
  $geq1$ et que le dernier caractère de $p(m)$ n'est jamais 0. On peut insérer du padding dans $p(m)$ si besoin est.

Définition 2 *Une fonction éponge prend en entrée une chaîne de caractère $p = (p_i)$ de longueur variable d'éléments de $\mathcal{A}$ et produit une séquence infinie z de caractères de $\mathcal{A}$ qui sera dans la pratique tronquée à n. Elle est déterminée par une fonction de $\mathcal{A}\times\mathcal{C}$ dans lui-même. p doit vérifier les 2 conditions $\ell = |p| \geq 1$ et, si $p = p_0...p_{\ell-1}$, alors $p_{\ell-1} \neq 0$. La fonction éponge a un état interne $S = (S_\mathcal{A}, S_\mathcal{C}) \in \mathcal{A}\times\mathcal{C}$, de valeur initiale $(0,0) \in \mathcal{A}\times\mathcal{C}$. La fonction éponge est évaluée en 2 temps :*

- *Phase d'absorption (absorbing) : Pour chaque caractère d'entrée p_i, $S := f(S_{\mathcal{A}} + p_i, S_{\mathcal{C}})$*
- *Phase d'extraction (squeezing) : la chaîne infinie de caractères z est obtenue en évaluant*

$$z_j := S_{\mathcal{A}}$$

et en mettant à jour l'état interne

$$S := f(S)$$

Notation :
$S = S_f[p]$ désigne l'état interne de l'éponge à la fin de la phase d'absorption. On a $S_f[] = (0,0)$ et $S_f[x|a] = f(S_f[x] + a)$, où x est une chaîne caractères et a un caractère et $S + a = (S_{\mathcal{A}} + a, S_{\mathcal{C}}$.

On a $z_j = S_{\mathcal{A},f}[p|0^j]$ pour $j \geq 0$.

Pour une entrée p, les états traversés sont les $S_f[p']$ pour tout p' préfixe de $p|0^\infty$.

Propriété (injection de p et $p(m)$ ne finit jamais par 0) :

$$(m_1, j) \neq (m_2, k) \Rightarrow p(m_1)|0^j \neq p(m_2)|0^k$$

De plus $|p(m)| \geq 1$ assure que la fonction f est évaluée au moins une fois.

Définition 3 *Une collision d'état est une paire de chemins distincts $p \neq q$ tels que $S_f[p] = S_f[q]$.*

Une collision d'état dans la phase d'absorption peut conduire à une collision sur la fonction de hachage, car $S_f[p|0^j] = S_f[q|0^j]$ pour tout j.

Une collision d'état peut aussi modéliser des cycles de la fonction de sortie lorsque $S_f[p] = S_f[q|0^d]$ pour un d donné.

Définition 4 *Une collision d'état interne est une paire de chemins distincts $p \neq q$ conduisant au même état de la partie interne $S_{\mathcal{C},f}[p] = S_{\mathcal{C},f}[q]$.*

collision d'état $\Rightarrow$ collision interne.

On peut facilement fabriquer l'inverse : il suffit de trouver a et b dans $\mathcal{A}$ tels que $S_{\mathcal{A},f}[p] + a = S_{\mathcal{A},f}[q] + b$.

Éponges aléatoires

Définition 5 *T-sponge. Une éponge à fonction de transition aléatoire est une fonction éponge telle que f est choisie aléatoirement par tirage uniforme dans l'ensemble des $\left(2^{k+c}\right)^{2^{k+c}}$ fonctions de transitions possibles entre $\mathcal{A} \times \mathcal{C}$ dans lui-même.*

Définition 6 *P-sponge. Une éponge à fonction de transition permutation aléatoire est une fonction éponge telle que f est choisie aléatoirement par tirage uniforme dans l'ensemble des $2^{k+c}!$ permutations possibles de $\mathcal{A} \times \mathcal{C}$.*

Pour distinguer une éponge aléatoire, on utilise une "boite noire" qui retourne avec proba 1/2 une sortie de RS (Random Sponge) ou de RO (Random Oracle). Un distingueur d'éponge aléatoire est un algorithme qui répond à la question "la sortie est une RS" avec une probabilité de succès supérieure à 1/2.

Théorème 2 *La sortie retournée par une éponge aléatoire à une série de requêtes sont indépendantes et uniformément distribuées s'il n'y a aucune collision interne durant les requêtes.*

Ce théorème signifie que, tant qu'il n'y a pas eu de collision interne durant le jeu de question-réponse, une RS est indifférenciable d'un RO.

Résistance intrinsèque d'une éponge

En vu d'utiliser une éponge comme fonction de hachage, on veut calculer la résistance intrinsèque d'une Random Sponge à 4 opérations :

- Collision interne : trouver 2 chemins distincts qui conduisent au même état interne. $p \neq q$, $S_{\mathcal{C},f}[p] = S_{\mathcal{C},f}[q]$.
- Chemin conduisant à un état interne : étant donné $S_{\mathcal{C}}$, trouver p tel que $S_{\mathcal{C},f}[p] = S_{\mathcal{C}}$.
- Cycles en sortie : trouver un chemin p et un entier d tel que $S_f[p] = S_f[p|0^d]$.
- Relier une chaîne de caractère en sortie à un état : étant donné une chaîne de caractère $t = (t_0, t_1, ..., t_m)$, trouver un état S tel que $S_{\mathcal{A}} = t_0$, $f_{\mathcal{A}}(S) = t_1$, $f_{\mathcal{A}}(f(S)) = t_2$ ..., $f_{\mathcal{A}}(f^{m-1}(S)) = t_m$.
  Il faut distinguer 2 cas :
  - Chaîne courte : $mk < c$. Le nombre de sorties possibles de longueur $m+1$ est plus petit que le nombre d'états internes. L'espérance du nombre de solutions est 2^{c-km}.
  - Chaîne longue : $mk > c$. Le nombre de sorties possibles de longueur $m+1$ est plus grand que le nombre d'états internes. Pour une chaîne t choisie aléatoirement, l'espérance du nombre de solutions est 2^{c-km}. Si on a une solution, elle est très probablement unique.

Le coût d'une attaque est compté en fonction du nombre d'appel$N$ à f pour les T-sponge et f ou f^{-1} pour les p-sponges.

Résultats du calcul de probabilité de succès (après approximations) :

	Collision interne	Chemin vers état	cycles en sortie	chaîne $km > c$	chaîne $km < c$
Taux élevé T-sponge	$2^{2y-(c+1)}$	2^{y-c}	$2^{2y-(c+k+1)}$	2^{y-c}	2^{y-km}
$k = 1$ T-sponge	$2^{2y-(c+1)}$	2^{y-c}	$2^{y-(c+2)}$	$2^{2y-(c+1)}$	2^{y-m}
Taux élevé P-sponge	$2^{2y-(c+1)}$	$2^{2y-(c+2)}$	$2^{y-(c+k)}$	2^{y-c}	2^{y-km}
$k = 1$ P-sponge	$2^{2y-(c+2)}$	$2^{2y-(c+3)}$	$2^{y-(c+1)}$	$2^{y-(c+1)}$	2^{y-m}

Avec $N = 2^y$.

Application des éponges aux fonctions de hachage

Pour une fonction de hachage, la sortie est tronquée à n bits.

- Collision en sortie, coût N attendu :
  - due à l'état interne : $2^{c+3)/2}$ - due à la sortie : $2^{n+3)/2}$
  Sous l'hypothèse $n < c$, la résistance aux collisions en sortie pour une random sponge est celle d'un oracle aléatoire.
  Si $n > c$, l'attaque par état interne devient prépondérante. Il faut donc choisir $n < c$.
  Il existe des attaques par multi-collisions qui conduisent à conseiller $2n < c$.
- 2-ème pré-image : calcul technique. La borne $2n < c$ est suffisante puisqu'une 2-ème préimage donne une collision.
- Pré-image.
  P-sponge : espérance $2^{n-k} + 2^{c/2}$. Pour tout k, le critère $n < c/2$ convient.
  T-sponge : espérance 2^n pour $n < c$. Le résultat correspond au cas de l'oracle aléatoire.

Chapitre 3

Chiffrement par bloc

3.1 Les modes de chiffrements

Avant de s'intéresser à la construction de l'algorithme de chiffrement par blocs lui-même, il est utile de préciser qu'il existe plusieurs modes qui permettent d'enchaîner le chiffrement des différents blocs de taille n, m_i pour i variant de 0 à $t-1$, la fonction de chiffrement E_K s'appliquant alors à chacun des blocs. Il s'agit donc de chaîner les $c_i = E_K(m_i)$ avec en général m_{i+1} pour i variant de 0 à $t-1$.

3.1.1 Le mode ECB

Le mode le plus simple est le mode ECB de l'anglais *Electronic Code Book* illustré à la figure 3.1 : le message à chiffrer est subdivisé en plusieurs blocs qui sont chiffrés séparément les uns après les autres. Le gros défaut de cette méthode est que deux blocs avec le même contenu seront chiffrés de la même manière, on peut donc tirer des informations à partir du texte chiffré en cherchant les séquences identiques. On obtient dès lors un dictionnaire de codes avec les correspondances entre le clair et le chiffré d'où le terme codebook.

Ce mode est pour ces raisons fortement déconseillé dans toute application cryptographique. Le seul avantage qu'il peut procurer est un accès rapide à une zone quelconque du texte chiffré et la possibilité de déchiffrer une partie seulement des données. Mais un mode bien plus sûr fondé sur un compteur permet également ces accès aléatoires et des déchiffrements partiels.

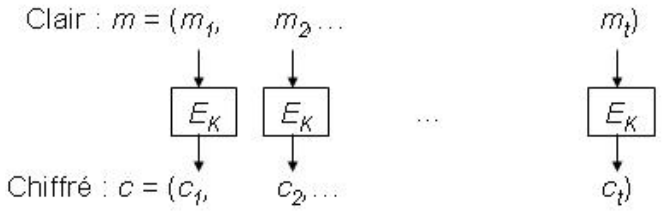


FIGURE 3.1 – Le mode ECB.

3.1.2 Le mode CBC

Un autre mode de chiffrement très employé est le mode CBC (présenté à la figure 3.2) de l'anglais Cipher Block Chaining. Il consiste à chiffrer le bloc i préalablement combiné par ou exclusif avec le chiffré du bloc précédent ainsi, $c_i = E_K(m_i \oplus c_{i-1})$ pour tout i de 1 à t, avec $c_0 = E_K(m_0 \oplus IV)$ où IV désigne un vecteur d'initialisation. C'est un bloc de données aléatoires qui permet de commencer le chiffrement

du premier bloc et qui fournit ainsi une forme de hasard indépendant du document à chiffrer. Il n ?a pas besoin d ?être lui-même chiffré lors de la transmission, mais il ne doit jamais être réemployé avec la même clé ce qui n'est pas le cas dans le WEP par exemple [13].

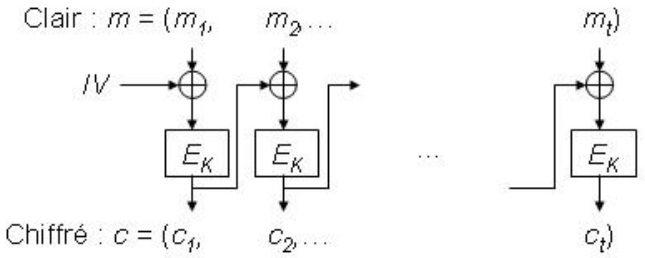


FIGURE 3.2 – Le mode CBC.

Il existe plusieurs autres modes que nous ne présenterons pas ici qui sont essentiellement des variantes du mode CBC. On peut citer : le mode CFB de l'anglais Cipher Feedback Block qui utilise un chiffrement à rétroaction, le mode OFB de l'anglais Output Feedback Block qui utilise un chiffrement à rétroaction de sortie,...

3.2 Schéma de Feistel

Le schéma élémentaire

Un chiffrement de Feistel est un chiffrement itéré qui utilise à chaque étage le même schéma, défini de la manière suivante :

Définition 7 *Soit f_1 une fonction de $I_n = \{0,1\}^n$ dans lui-même et x^0, x^1, x^2, x^3 quatre éléments de I_n. On définit le schéma de Feistel Ψ lié à f_1 sur $I_{2n} = \{0,1\}^{2n}$ de la manière suivante :*

$$\forall (x^0, x^1) \in (I_n)^2, \Psi(f_1)[(x^0, x^1)] = (x^2, x^3) \Leftrightarrow \begin{cases} x^2 = x^1 \\ x^3 = f_1(x^1) \oplus x^0 \end{cases}$$

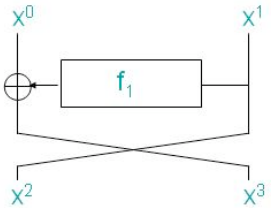


FIGURE 3.3 – Schéma de Feistel

Précisons quelques propriétés élémentaires de ce schéma [11] :
– Dans tous les cas et pour toute fonction f_1, $\Psi(f_1)$ est une permutation de I_{2n} car on peut retrouver x^0 et x^1 à partir de x^2 et x^3 de façon unique :

$$x^2 = x^1 \text{ et } x^3 = x^0 \oplus f_1(x^1) \tag{3.1}$$

donc on retrouve tout d'abord la valeur de x^1 puis celle de x^3 grâce à la connaissance de x^1. Cette définition se généralise de la façon suivante : la fonction réciproque de $\Psi(f_1)$ est très facile à calculer :

$$\Psi(f_1)^{-1} = \sigma \circ \Psi(f_1) \circ \sigma$$

où σ désigne la permutation de I^{2n} qui inverse les parties droite et gauche d'un mot de I^{2n}.

Ψ représente un schéma de Feistel sur un étage (voir figure 3.3). On peut alors généraliser la définition de Ψ à plusieurs étages comme cela est fait pour le DES : soit f_1, f_2,..., f_r, r fonctions de $\{0,1\}^n$ vers $\{0,1\}^n$, on définit $\Psi^r(f_1, \cdots, f_r)$ de $\{0,1\}^{2n}$ vers $\{0,1\}^{2n}$ par :

$$\Psi^r(f_1, ..., f_r) = \Psi(f_r) \circ \cdots \circ \Psi(f_2) \circ \Psi(f_1).$$

On définit ainsi un schéma de Feistel sur r étages. Précisons également que pour le DES, on a n=32 bits et que les blocs d'entrées sont donc de taille 64 bits.

Des schémas modifiés

Nous venons de voir le schéma général de la fonction d'étage qui est utilisé dans le DES. Ce schéma possède plusieurs variantes tant dans la façon de placer la fonction f_1 que dans le nombre de blocs d'entrées. Dans le cas du DES, les blocs d'entrée sont découpés en deux afin de fournir x^0 et x^1. On peut imaginer découper cette entrée en plus de blocs comme c'est le cas dans l'algorithme MARS [7] (figure 3.4) ou l'algorithme CAST-256 [6] (figure 3.5), deux candidats malheureux de la compétition AES qui s'est tenu entre 1997 et 2001 afin de choisir un nouvel algorithme de chiffrement symétrique américain pour le XXIème siècle.

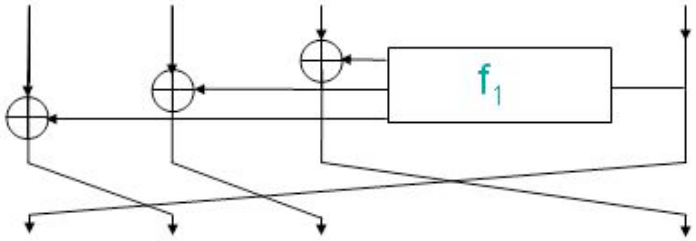


FIGURE 3.4 – Schéma de Feistel modifié pour l'algorithme MARS

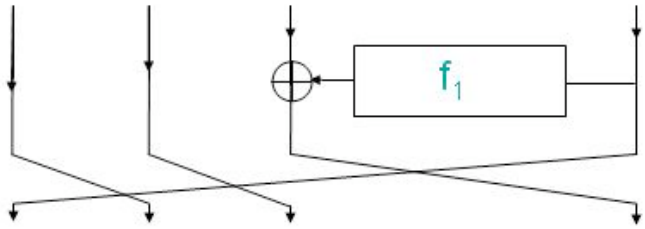


FIGURE 3.5 – Schéma de Feistel modifié pour l'algorithme CAST-256

On peut également citer ici le cas de MISTY [12] qui utilise une variante du schéma de Feistel, non pas en ce qui concerne le nombre de blocs d'entrée, mais pour la manière dont il fait agir la fonction f_1.

Le schéma de MISTY est décrit à la figure 3.6. Cet algorithme a cependant révélé plusieurs faiblesses, plusieurs autres versions ont été proposé, notamment MISTY1, une version optimisée en hardware de ce chiffrement et nommée KASUMI qui est utilisé dans des modes particuliers (f8 pour la confidentialité et f9 pour l'intégrité [3]) pour les téléphones mobiles de troisième génération notamment dans la norme européenne UMTS [4].

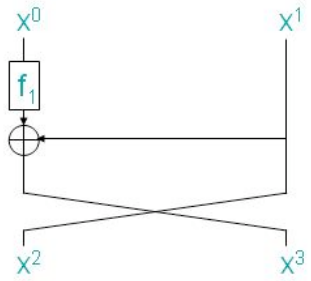


FIGURE 3.6 – Schéma de Feistel modifié pour les algorithmes MISTY et KASUMI

Nous venons donc de voir des schémas dérivés de celui de Feistel qui permettent de construire une fonction d'étage d'un algorithme de chiffrement par blocs itératifs. La question à se poser à présent est : de quoi doit être composée la fonction f_1 pour optimiser la confusion et la diffusion ?

En général, cette fonction peut se décomposer en deux parties : une partie non linéaire qui va permettre une bonne confusion et une partie linéaire qui va permettre d'optimiser la diffusion et le mélange des bits d'entrée. La partie non linéaire est en général composée de "boîtes S", c'est à dire de permutations non linéaires qui agissent sur chaque sous-bloc de taille 4, 8 ou 16 bits et qui permettent de casser la structure linéaire du chiffrement.

La deuxième fonction qui compose f_1 est une application linéaire chargée de maximiser la diffusion. Nous verrons des exemples pratiques de ces applications dans les sections suivantes.

3.3 Réseau de substitutions-permutations

Le principe diffusion - confusion

Un réseau de substitutions-permutations est également un système de chiffrement itératif. A chaque étage, on applique au bloc d'entrée une substitution non linéaire puis une fonction généralement linéaire appelée abusivement permutation. Ces réseaux prennent au mot les deux concepts définis par Shannon. La substitution, en général représentée par une boîte S, garantit, si elle est bien choisie, une bonne confusion (faire disparaître les structures tant linéaires qu'algébriques du chiffrement) et la permutation garantit une bonne diffusion de l'information (faire en sorte que chaque bit de sortie soit influencé par le plus grand nombre possible de bits d'entrée).

Exemple de l'AES : voir Annexe C.

3.4 Cryptanalyse des systèmes de chiffrement par bloc

3.4.1 Qu'est ce qu'une attaque ou les règles du jeu en cryptanalyse

La cryptanalyse d'un système peut être alors soit partielle (l'attaquant découvre le texte clair correspondant à un ou plusieurs messages chiffrés interceptés), soit totale (l'attaquant peut déchiffrer tous les messages, par exemple en trouvant la clé). Il existe plusieurs types d'attaques selon les moyens dont dispose l'attaquant :

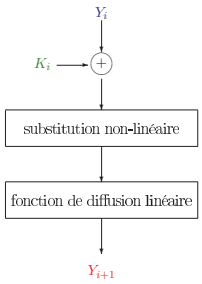


FIGURE 3.7 – Fonction itérée d'un réseau de substitutions-permutations

- *Attaques à chiffré connu* : l'attaquant a seulement accès à des messages chiffrés.
- *Attaques à clair connu* : l'attaquant dispose d'un ou plusieurs messages clairs et des chiffrés correspondants.
- *Attaques à clair choisi* : l'attaquant choisit des clairs et peut obtenir les chiffrés correspondants.
- *Attaques à chiffré choisi* : l'attaquant peut déchiffrer les messages de son choix.

L'attaque la plus simple et la plus brutale est la recherche exhaustive. L'attaquant teste l'ensemble des clés possibles sur un cryptogramme donné dont il est supposé connaître au moins partiellement le clair ; il a découvert la bonne clé lorsque le déchiffrement redonne le clair attendu. La complexité d'une recherche exhaustive sur un algorithme de chiffrement par blocs dont la taille des clés est n bits est de l'ordre de 2^n chiffrements. Il est donc nécessaire lorsque l'on souhaite créer un algorithme de chiffrement de prendre des clés suffisamment longues pour se prémunir contre ce type d'attaque. Malheureusement, ce n'est pas la seule condition à vérifier.

On suppose donc ici que l'on a le droit de monter des attaques à partir du moment où elles coûtent moins chères que la recherche exhaustive. En pratique, la plupart du temps, on réduit le nombre d'étages à attaquer afin de mettre en lumière une faiblesse particulière de l'algorithme.

3.4.2 Attaque par distingueur d'un schéma de chiffrement par bloc itératif

On considère un chiffrement par bloc itératif avec une fonction de tour F_{K_i} paramétrée par la clé de tours K_i.

On cherche à attaquer le dernier tour. On regarde alors l'ensemble des applications de $\{0,1\}^n$ dans lui-même qui correspondent aux résultats des $r-1$ premiers tours du chiffrement :

Soit $G_K = F_{K_{r-1}} \circ \ldots \circ F_{K_1}$. On cherche un distingueur, c'est-à-dire une fonction $\mathcal{T}$ qui à d éléments $(x_1, \ldots, x_d)$ de taille n et leurs images par une permutation $(\pi(x_1), \ldots, \pi(x_d))$, associe une valeur binaire telle que la probabilité qu'elle soit égale à 1 quand π est dans l'ensemble des chiffrements réduits (i.e. $\pi = G_K$ pour une certaine clé, soit significativement plus élevée que lorsque π est une permutation aléatoire.

La différence entre ces 2 probabilités est l'avantage du distingueur.

Dans le cadre d'une attaque statistique sur le dernier tour, on fait une recherche exhaustive sur les différentes valeurs de la sous-clé K_r du dernier tour.

A partir de la donnée de d couples clairs/chiffrés $(x_1, \ldots, x_d)/(y_1, \ldots, y_d)$, on applique $F_{K_r}^{-1}$ à $(y_1, \ldots, y_d)$, on obtient $(\pi(x_1), \ldots, \pi(x_d))$, et on applique le distingueur pour valider si la clé K_r est la bonne.

Dans la pratique, il n'est pas possible de faire une recherche exhaustive sur la clé, mais on cherche un distingueur "localisé" qui ne fasse apparaître un biais que sur une partie du chiffré, ou bien ne soit sensible qu'à une partie de la sous-clé. On fait alors une recherche exhaustive sur cette partie de sous-clé, puis une recherche exhaustive sur les bits manquants, ou bien on utilise un autre distingueur.

3.4.3 Cryptanalyse différentielle

La cryptanalyse différentielle a été introduite par E. Biham et A. Shamir en 1991 [1], c'est une attaque *à message clair choisi* applicable aux algorithmes de chiffrement par blocs itératifs.

Le principe général de cette attaque consiste à considérer des couples de clairs X et X' présentant une différence ΔX fixée et à étudier la propagation de cette différence initiale à travers le chiffrement. Les différences sont définies par une loi de groupe, en général le x-or bit à bit. Cette attaque utilise la faiblesse potentielle de la fonction itérée f dans une dérivation à l'ordre 1.

Plus précisément, la différentielle en a de la fonction G est $D_aG : x \mapsto G(x+a) - G(x)$, où G est la fonction du chiffrement réduit à $r-1$ tours. On cherche à voir si la distribution de cette fonction est éloignée de celle de la différentielle pour une fonction aléatoire.

Dans la pratique, on ne peux pas déterminer la distribution complète, par contre, on rechercher un couple (a, b) tel que

$$|\{x \in F_2^n \mid G(x+a) - G(x) = b\}| >> 1$$

3.4.4 Cryptanalyse linéaire

La cryptanalyse linéaire a été introduite par Gilbert, Chassé et Tardy-Corfdir [9, 10].

Elle exploite l'existence d'une relation linéaire biaisée entre les entrées et les sorties du chiffrement réduit

On utilise un couple (a, b) non nuls tels que la distribution de la fonction

$$x \mapsto a.G(x) + b.x$$

n'est pas uniforme pour tout chiffrement réduit $G \in \mathcal{G}$.

Si

$$Pr[a.G(x) + b.x = 1] = 1/2 + \varepsilon$$

il est possible de distinguer cette distribution à l'aide d'un nombre de couples clairs-chiffrés de l'ordre de ε^2.

Exercice : exemple de l'approximation linéaire sur 3 tours du DES (Voir [5] 1,7,1 p.35.

Bibliographie

[1] E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, 4, no. 1, 1991.

[2] Adi Shamir. "stream ciphers : Dead or alive ?". invited talk of *SASC - The State of the Art of Stream Ciphers*, 2004.

[3] 3rd Generation PartnerShip Project. "3gpp ts 35.201 - specification of the 3gpp confidentiality and integrity algorithms - document 1 : f8 and f9 specification".
`http://www.3gpp.org/ftp/Specs/html-info/35201.htm`, 2001.

[4] 3rd Generation PartnerShip Project. "3gpp ts 35.202 - specification of the 3gpp confidentiality and integrity algorithms - document 2 : Kasumi specification".
`http://www.3gpp.org/ftp/Specs/html-info/35202.htm`, 2001.

[5] A Canteaut. "La cryptographie symétrique". Notes de cours,2008.

[6] C. Adams and J. Gilchrist. "the cast-256 encryption algorithm". RFC 2612 - IETF, 1999.

[7] D. Coppersmith, C. Burwick, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. Matyas Jr., L. O'Connor, M. Peyravian, D.Safford, and N. Zunic. Mars - a candidate ciphe for aes. In *The First Advanced Encryption Standard Candidate Conference*. N.I.S.T., 1998.

[8] FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T.

[9] H. Gilbert and G. Chassé. A statistical attack of the FEAL-8 cryptosystem. In *Advances in Cryptology - CRYPTO'90*,p. 22-33, LNCS v.537, Springer-Verlag 1991.

[10] A. Tardy-Corfdir and H. Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In *Advances in Cryptology - CRYPTO'91*,p. 172-182, LNCS v.576, Springer-Verlag 1991.

[11] J. Patarin. *Etudes des Générateurs de Permutations Pseudo-aléatoires Basés sur le Schéma du DES*. PhD thesis, Université Paris VI, 1991.

[12] M. Matsui. New block encryption algorithm misty. In *Fast Software Encryption'97*, Haifa, Isral, pages 54–68. Lectures Notes in Computer Science 1267, Springer-Verlag, 1997.

[13] Wikipédia. "article wired equivalent privacy (WEP)".
`http://fr.wikipedia.org/wiki/WEP`, 2005.

Chapitre 4

Chiffrement à flot

4.1 Introduction au chiffrement à flot

Définition. Le chiffrement à flot, appelé également *chiffrement par flux* ou *chiffrement à la volée* (stream cipher en anglais), est une des deux grandes familles de chiffrements à clef secrète. Dans un chiffrement à flot, le texte chiffré est obtenu en combinant par ou exclusif bit-à-bit le message clair avec une suite binaire secrète, de même longueur que le message. Cette suite binaire, appelée suite chiffrante, peut être

- soit une suite aléatoire entièrement secrète partagée par les deux utilisateurs : cette situation correspond à la technique du masque jetable ;
- soit une suite pseudo-aléatoire, c'est-à-dire produite à partir d'une clef secrète par un générateur pseudo-aléatoire.

Les algorithmes par flot s'opposent donc aux algorithmes de chiffrement par blocs, comme l'AES ou le DES, qui consistent à découper le message à transmettre en blocs de taille fixe (généralement en blocs de 128 bits), puis à transformer par le même procédé chacun de ces blocs en un bloc de chiffré.

Notons que certains algorithmes de chiffrement à flot, dits auto-synchronisants, d'utilisation peu fréquente ne sont pas couverts par cette définition.

Propriétés générales et avantages. Avec un algorithme de chiffrement par blocs, on ne peut commencer à chiffrer et à déchiffrer un message que si l'on connaît la totalité d'un bloc. Ceci occasionne naturellement un délai dans la transmission et nécessite également le stockage successif des blocs dans une mémoire-tampon. Au contraire, dans les procédés de chiffrement à flot, chaque nouveau bit transmis peut être chiffré ou déchiffré indépendamment des autres, en particulier sans qu'il soit nécessaire d'attendre les bits suivants.

D'autre part, les chiffrements à flot ne requièrent évidemment pas de padding, c'est-à-dire l'ajout de certains bits au message clair dont le seul objectif est d'atteindre une longueur multiple de la taille de bloc. Ceci peut s'avérer particulièrement souhaitable dans les applications où la bande passante est très limitée ou quand le protocole employé impose la transmission de paquets relativement courts (auquel cas, le padding représente une proportion non négligeable des données échangées).

Un autre avantage de ces techniques est que, contrairement aux algorithmes par blocs, le processus de déchiffrement ne propage pas les erreurs de transmission. Supposons qu'une erreur survenue au cours de la communication ait affecté un bit du message chiffré. Dans le cas d'un chiffrement à flot, cette erreur affecte uniquement le bit correspondant du texte clair, et ne le rend donc généralement pas complètement incompréhensible. Par contre, dans le cas d'un chiffrement par blocs, c'est tout le bloc contenant la position erronée qui devient incorrect après déchiffrement. Ainsi, une erreur sur un seul bit lors de la transmission affecte en réalité 128 bits du message clair. C'est pour cette raison que le chiffrement à la volée est également utilisé pour protéger la confidentialité dans les transmissions bruitées.

Contextes d'utilisation. Outre le fait qu'ils sont bien adaptés pour les transmissions bruitées ou à faible passante, les procédés de chiffrement à flot sont généralement privilégiés dans des contextes où il est primordial de pouvoir chiffrer et déchiffrer très rapidement ou au moyen de ressources très limitées. Leur utilisation est par exemple systématique dans les applications qui imposent de fortes contraintes sur la taille et la consommation électrique du circuit électronique dédié au chiffrement. C'est le cas de la plupart des systèmes embarqués, tels les téléphones mobiles.

C'est également parmi les algorithmes à flot que l'on trouve les systèmes de chiffrement les plus rapides. Ils sont donc utilisés dès que l'on souhaite une vitesse de chiffrement extrêmement élevée, que l'on ne peut atteindre avec des algorithmes par blocs.

Chiffrement à flot et générateurs pseudo-aléatoires. L'algorithme de chiffrement à flot le plus simple est le célèbre chiffre du masque jetable, qui offre une sécurité parfaite mais qui nécessite l'échange au préalable d'une clef secrète aussi longue que le message à chiffrer. Il ne peut donc pas être utilisé en pratique sauf dans des cas extrêmement particuliers où l'on dispose de moyens physiques pour échanger des clefs de grande taille (typiquement, les communications diplomatiques). Les algorithmes de chiffrement à flot employés en pratique sont donc des versions affaiblies du chiffre du masque jetable dans lesquelles la suite aléatoire secrète est remplacée par une suite produite par un générateur pseudo-aléatoire.

4.2 Les générateurs pseudo-aléatoires dérivés d'un algorithme par blocs

Il est toujours possible de réaliser un générateur pseudo-aléatoire au moyen d'un algorithme de chiffrement par bloc utilisé dans un mode opératoire particulier. La suite chiffrante est alors produite par blocs dont la taille correspond à la taille de bloc de l'algorithme par bloc sous-jacent.

Ainsi, les modes *Output FeedBack (OFB)* et *Compteur (CTR)* sont des modes opératoires bien connus et standardisés depuis de nombreuses années qui permettent de produire une suite pseudo-aléatoire à partir d'une clef secrète et d'une valeur initiale. Le mode OFB a été initialement défini dans la norme FIPS 81 [1] ; le mode CTR a été ajouté aux modes opératoires usuels dans la publication spéciale du NIST, NIST SP 800-38A [4]. Leurs spécifications sont notamment reprises dans la norme récente ISO/IEC 10116 [3].

Mode OFB. Dans le mode OFB, l'algorithme par bloc paramétré par la clef secrète K est appliqué à la valeur initiale (IV), assimilée au texte clair. Le chiffré correspondant fournit le premier bloc de suite chiffrante. Chacun des blocs de suite pseudo-aléatoire suivant est ensuite égal à l'image par l'algorithme par bloc du chiffré précédent (*cf.* Figure 4.1).

Mode CTR. Dans le mode CTR, le $i$ème bloc de suite chiffrante correspond à l'image par l'algorithme par bloc paramétré avec la clef secrète K de la valeur initiale additionnée (par ou exclusif) à un compteur c_i (qui prend généralement la valeur i) (*cf.* Figure 4.2).

Sécurité des modes OFB et CTR. Les générateurs pseudo-aléatoires obtenus au moyen d'un algorithme par bloc résistant à la cryptanalyse sont souvent considérés comme offrant une sécurité raisonnable. Toutefois, d'un point de vue théorique, ils ne sont pas sûrs puisqu'ils sont tous deux vulnérables à une attaque par distingueur à valeur initiale choisie [2].

Ainsi, pour le mode CTR, le $i$ème bloc de la suite générée à partir de la valeur initiale IV est toujours égal au $j$ème bloc de celle générée à partir de la valeur initiale $IV \oplus c_i \oplus c_j$. De même, pour le mode OFB, le $i$ème bloc de la suite générée à partir de la valeur initiale IV est toujours égal au $(i-1)$ème bloc de celle obtenue à partir d'une valeur initiale égale au premier bloc de la suite précédente.

Ces propriétés permettent évidemment de distinguer aisément la suite produite par chacun de ces modes d'une suite aléatoire. Cette faiblesse a récemment conduit à des modifications de chacun de ces

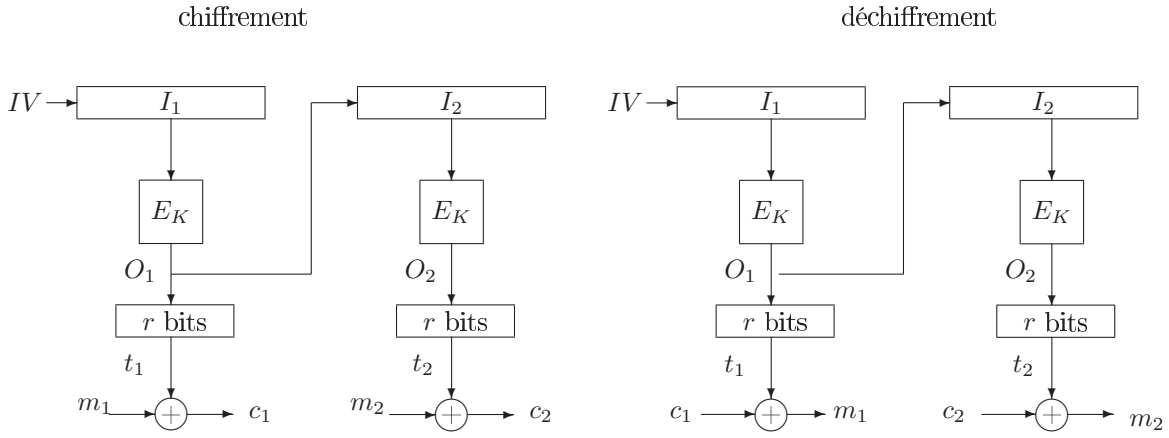


FIGURE 4.1 – Le mode opératoire OFB

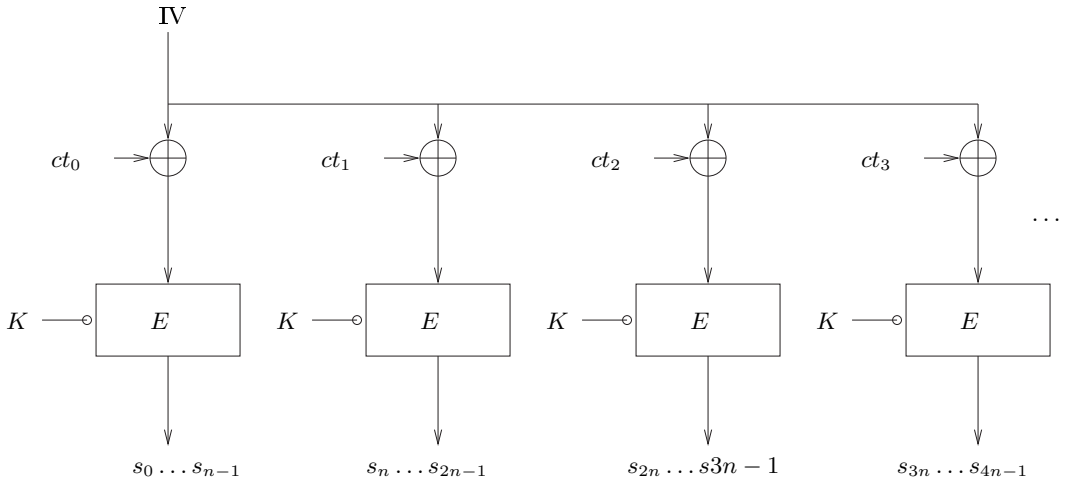


FIGURE 4.2 – Le mode opératoire CTR (Compteur)

modes, pour lesquelles il est au contraire possible de démontrer que les suites produites ne peuvent être distinguées d'une suite aléatoire si le chiffrement par bloc sous-jacent possède une propriété similaire.

4.3 Les générateurs pseudo-aléatoires dédiés

Les générateurs pseudo-aléatoires dédiés sont les seules constructions qui permettent d'atteindre un débit de chiffrement supérieur à celui d'un algorithme par blocs (de l'ordre de quelques cycles du processeur par octet en logiciel), ou qui puissent être implémentés par un circuit électronique de petite taille et à faible consommation.

État de l'art. Au contraire des générateurs utilisant un algorithme par blocs, les contraintes liées à leur mise en œuvre, dans un environnement logiciel ou matériel, interviennent ici directement dans la conception. Ces contraintes, combinées aux exigences de sécurité, rendent la tâche relativement difficile. Ainsi, à l'heure actuelle, très peu de générateurs dédiés sont considérés comme sûrs, au sens où ils ne sont vulnérables à aucune attaque plus rapide que la recherche exhaustive de la clef secrète. Quelques propositions récentes, SNOW 2.0 et MUGI, ont été prises en compte dans la dernière version de travail de la norme internationale de chiffrement ISO/IEC 18033-4, mais leur conception reste trop récente pour pouvoir se prononcer sur leur sécurité à long terme. De façon générale, la conception de nouveaux générateurs pseudo-aléatoires dédiés est actuellement un sujet de recherche extrêmement actif ; on peut ainsi mentionner le projet eSTREAM lancé par le réseau européen ECRYPT [5] suite auquel une trentaine de nouveaux générateurs dédiés ont été proposés en avril 2005 et dont la sécurité et les performances doivent encore faire l'objet d'une évaluation approfondie.

Modèle général d'un générateur pseudo-aléatoire dédié. Un générateur pseudo-aléatoire dédié est un automate à états finis qui engendre à chaque instant un ou plusieurs bits déterminés par la valeur de son état interne. Son fonctionnement, décrit à la figure 4.3, est généralement modélisé par trois fonctions différentes.

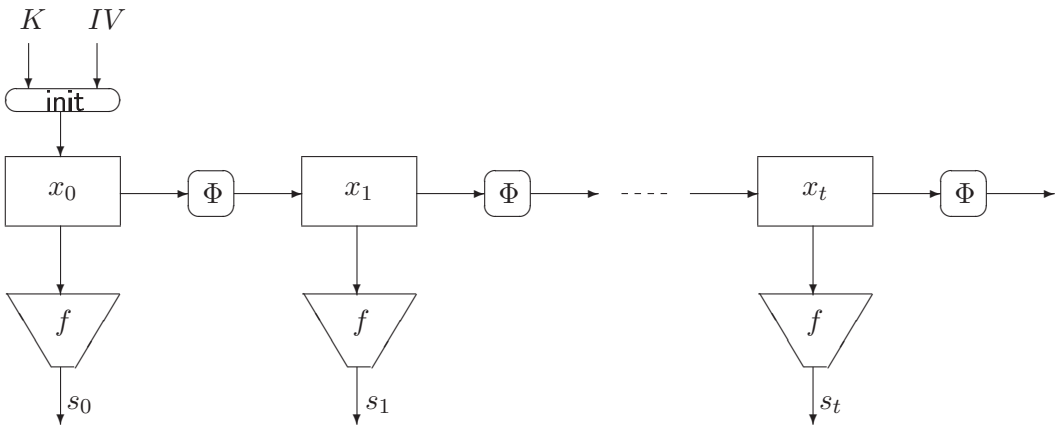


FIGURE 4.3 – Modèle d'un générateur pseudo-aléatoire dédié

- *une procédure d'initialisation* qui détermine l'état initial du générateur pseudo-aléatoire à partir de la clef secrète et de la valeur initiale. Notons que l'initialisation est parfois divisée en deux phases : l'une, dite de chargement de clef, calcule une certaine quantité qui ne dépend que de la clef (et non de la valeur initiale), l'autre, dite d'injection d'IV ou de re-synchronisation, détermine l'état initial du générateur à partir de la valeur calculée précédemment et de la valeur initiale. Le fait de découper de la sorte la phase d'initialisation permet de réduire le coût de la procédure qui consiste à changer la valeur initiale sans modifier la clef. Il s'agit en effet d'une opération beaucoup plus fréquente en pratique que le changement de clef, notamment pour les protocoles de communication pour lesquels la longueur des paquets échangés est relativement petite. Par exemple, dans le communications GSM, on change l'IV tous les 228 bits alors que la clef reste inchangée tout au long de la conversation.
- *une fonction de transition* (notée Φ sur la figure 4.3) qui fait évoluer l'état interne du générateur entre l'instant t et $(t+1)$. Cette fonction peut dépendre de la clef, de la valeur initiale et du temps, mais elle est fixe dans l'immense majorité des générateurs destinés à une mise en œuvre matérielle pour des raisons évidentes de simplicité et d'encombrement.

– *une fonction de filtrage* (notée f sur la figure 4.3) qui, à chaque instant, produit un ou plusieurs bits de suite à partir de l'état interne courant. Tout comme la fonction de transition, la fonction de filtrage peut varier avec la clef, la valeur initiale et le temps, mais elle est généralement fixe pour les raisons évoquées précédemment.

Il est important de noter que, si l'état interne est composé de n bits, on reviendra nécessairement à un état interne déjà rencontré après au plus 2^n tops d'horloge, ce qui implique que la suite produite a une période inférieure ou égale à 2^n. Toute utilisation doit donc imposer un changement de clef ou de valeur initiale dès que 2^n bits ont été produits. Dans de nombreuses applications, le nombre maximal de bits produits sans changement de clef ou de valeur initiale est généralement limité à une valeur bien inférieure, mais d'un point de vue théorique, on considère des attaques qui peuvent nécessiter jusqu'à 2^k bits connus de suite chiffrante, où k est le nombre de bits de la clef secrète.

Les contraintes imposées par les attaques génériques. Chacun des composants d'un générateur pseudo-aléatoire dédié doit être choisi avec précaution. En particulier, les critères fondamentaux suivants sont dictés par la nécessité de résister à des attaques classiques qui s'appliquent à tous les générateurs de ce type.

– La taille de l'état interne doit être suffisamment grande pour parer la recherche exhaustive, mais aussi les attaques dites par compromis temps-mémoire (voir la fiche consacrée à ces attaques pour plus détails). Celles-ci imposent en particulier que la taille de l'état interne doit être au minimum le double de la taille de la clef secrète.

– La fonction de filtrage doit assurer que la sortie du générateur est uniformément distribuée. Par exemple, dans le cas où un seul bit est produit à chaque unité de temps, il faut que ce bit prenne les valeurs 0 et 1 avec la même probabilité. Pour cela, la fonction de filtrage doit être équilibrée, ce qui signifie que le nombre de vecteurs x dont l'image par f vaut y est le même quelle que soit la valeur de y. Dans le cas contraire, la suite produite par le générateur est biaisée, au sens qu'elle contient plus de 0 que de 1, ce qui permet de la distinguer d'une suite aléatoire dès lors que l'on connaît de l'ordre de

$$\frac{1}{|Pr_X[f(X)=1]-\frac{1}{2}|^2}$$

bits de la suite engendrée, le nombre d'opérations à effectuer pour mener à bien cette attaque par distingueur étant du même ordre.

– La fonction de transition Φ doit garantir que la suite chiffrante possède une période élevée quel que soit l'état initial. Sinon, il devient facile de distinguer la suite produite d'une suite aléatoire.

– L'une des deux fonctions au moins, Φ ou f, doit être non linéaire. Sinon, la suite produite dépend linéairement des bits de l'état initial. Dans ce cas, la connaissance de n bits de suite chiffrante, où n est la taille de l'état interne, fournit un système de n équations à n inconnues (les bits de l'état initial), que l'on peut résoudre simplement au moyen d'un pivot de Gauss. Cette attaque permettrait de retrouver l'état initial du générateur en n^3 opérations, ce qui est très inférieur à la complexité de la recherche exhaustive de la clef.

Les grandes familles de générateurs pseudo-aléatoires dédiés. La classification des différents types de générateurs pseudo-aléatoires dédiés est une tâche délicate dans la mesure où leur conception est largement liée à l'environnement applicatif auquel le générateur est destiné. On peut toutefois distinguer trois grandes familles suivant le type de fonction de transition employé. Mais ces classes peuvent elles-mêmes parfois être subdivisées suivant que le générateur vise une mise en œuvre logicielle ou matérielle.

– *Les chiffrements à transition linéaire.* L'utilisation d'une fonction de transition linéaire est en effet un choix naturel en termes de simplicité d'implémentation, dans la mesure où la fonction de filtrage garantit que la suite produite ne dépend pas linéairement de l'état initial du générateur). Parmi les fonctions de transition linéaires, celles qui sont mises en œuvre au moyen de registres à décalage à rétroaction linéaire (LFSR) sont privilégiées à la fois pour le faible coût de leur implémentation matérielle et parce que l'on dispose de nombreux résultats théoriques sur les propriétés statistiques

des suites produites. Les générateurs utilisant des registres à décalage à rétroaction linéaire sont sans aucun doute ceux qui ont fait l'objet des études les plus nombreuses. Ces systèmes peuvent être destinés soit à un environnement matériel, soit à un environnement logiciel. Mais, on utilise généralement dans ce dernier cas des registres à décalage à rétroaction linéaire non plus binaires, mais opérant sur un alphabet plus grand (typiquement sur des octets ou des mots de 32 bits). Parmi les générateurs à base de LFSR d'utilisation courante, on peut mentionner E0, déployé dans la norme Bluetooth, A5/1 utilisé pour chiffrer les communications des téléphones mobiles dans la norme GSM ou SNOW 2.0 qui, lui, vise des applications logicielles, est qui est inclus dans la dernière version de la norme ISO/IEC 18033. Les progrès récents dans le domaine de la cryptanalyse, notamment les attaques dites algébriques proposées dernièrement, mettent toutefois en évidence des faiblesses inhérentes à de nombreux générateurs de ce type. De multiples précautions liées à l'émergence de ces attaques doivent donc être prises lors de la conception de générateurs utilisant une fonction de transition linéaire.

– *Les chiffrements à transition non-linéaire.* Afin d'éviter les faiblesses pouvant résulter du caractère linéaire de la fonction transition, certaines conceptions récentes privilégient une évolution non-linéaire. Toutefois, la fonction de transition choisie doit garantir que les états internes du générateur ne forment pas une suite de faible période, et ce quel que soit la valeur de l'état initial. Contrairement aux fonctions linéaires, il est relativement difficile d'obtenir de tels résultats théoriques pour des fonctions non-linéaires. Cette difficulté peut être contournée si la taille de l'état interne du générateur n'est pas limitée drastiquement par des contraintes d'implémentation (c'est le cas des applications logicielles destinées aux ordinateurs usuels). Dans ce cas, même en l'absence de résultats théoriques, on peut estimer qu'il est très peu probable qu'un état initial engendre une suite de faible période si l'état interne est suffisamment grand. L'exemple type est celui de RC4, générateur proposé par R. Rivest et utilisé dans de nombreuses applications (SSL/TLS,...), dont l'état interne correspond à un tableau de 512 octets dont les valeurs sont modifiées de façon non-linéaire à chaque itération de l'algorithme (voir la fiche concernant RC4 pour une description précise).
  Toutefois, dans les applications matérielles, les contraintes sur la taille du circuit correspondant imposent que l'état interne du générateur ne soit pas trop grand, autrement dit que sa taille n'excède pas sensiblement le double de la longueur de la clef (qui est la taille minimale pour résister aux attaques par compromis temps-mémoire). Dans ce cas, il est indispensable de disposer de résultats théoriques sur la période de la fonction de transition. A l'heure actuelle, très peu de fonctions offrent ces garanties et les générateurs pseudo-aléatoires les utilisant sont encore au stade de développement. On peut mentionner certains registres à décalage à rétroaction non-linéaire, les registres à décalage à rétroaction avec retenues [7, 6] et les T-fonctions [8, 9], cette toute dernière proposition récente s'avérant peu souhaitable tant à cause de certaines faiblesses liées à son emploi que de sa lenteur même en logiciel [10].
– *Les conceptions hybrides.* Dans certains générateurs pseudo-aléatoires, l'état interne est divisé en deux parties, l'une étant mise à jour par une fonction linéaire, l'autre par une fonction non-linéaire. Lorsque la partie qui évolue de manière non-linéaire est beaucoup plus petite que l'autre, elle est généralement assimilée à une mémoire interne. Autrement dit, le générateur est souvent classé comme un générateur à transition linéaire avec mémoire. C'est le cas par exemple des générateurs SNOW 2.0 et E0, qui sont usuellement qualifiés de systèmes à transition linéaire. Toutefois, il existe des générateurs dans lesquels les parties à évolution linéaire et non-linéaire de l'état interne sont de tailles similaires. On trouve dans cette catégorie le générateur MUGI conçu par la société Hitachi et qui figure dans la dernière version de travail de la norme ISO/IEC 18033-4 au même titre que SNOW 2.0.

4.4 Les propriétés statistiques des générateurs pseudo-aléatoires

Pour qu'un chiffrement à flot résiste aux attaques par distingueur, il faut que le générateur pseudo-aléatoire utilisé possède de bonnes propriétés statistiques. Même si l'existence d'une attaque par distin-

gueur ne remet pas toujours en cause complètement la sécurité du chiffrement (car elle ne permet pas nécessairement de retrouver le texte clair, ou la clef secrète), un critère classique de sécurité est que 2^k bits de sortie du générateur pseudo-aléatoire ne puissent pas être distingués d'une suite aléatoire au moyen d'un algorithme dont le coût soit inférieur à 2^k, où k correspond ici au nombre de bits de la clef secrète.

Pour les générateurs pseudo-aléatoires utilisés en pratique pour le chiffrement à flot, il n'existe pas de preuve théorique de l'absence de distingueur de complexité polynomiale. Toutefois, on dispose d'un certain nombre de tests statistiques classiques que tout générateur pseudo-aléatoire doit au minimum vérifier pour prétendre à une sécurité raisonnable, mais il ne s'agit évidemment pas d'une condition de sécurité suffisante.

Les premières propriétés statistiques requises pour une suite pseudo-aléatoire ont été décrites par Knuth [12] et Golomb [11]. Elles ont donné lieu à des familles de tests, qui ont été enrichies depuis. On distingue usuellement les tests probabilistes dits *de normalité*, qui déterminent la probabilité pour qu'une suite tirée aléatoirement vérifie une propriété identique à celle de la suite à tester, et les tests dits *de compression* qui déterminent si la suite à tester peut être compressée de façon significative.

Les tests de normalité. Les tests de normalité reposent sur le principe suivant : à toute suite de n bits on associe une certaine quantité dont on peut déterminer la distribution de probabilité pour une suite aléatoire. Par exemple, dans le test de fréquence qui vérifie si l'écart entre le nombre de zéros et le nombre de uns dans une suite binaire de longueur n est raisonnablement faible, on utilise la quantité

$$X = \frac{(n_0 - n_1)^2}{n} \tag{4.1}$$

où n_0 (resp. n_1) est le nombre de 0 (resp. de 1) dans la suite. Cette quantité suit une distribution du χ^2 de degré 1 quand $n \geq 10$ (en pratique, il est préférable d'effectuer ce test sur une longueur de l'ordre de quelques milliers de bits).

On calcule ensuite la valeur de cette quantité pour la suite pseudo-aléatoire à tester (ou pour l'ensemble de suites produites par le générateur) et on la compare à la valeur moyenne attendue pour une suite aléatoire. Plus exactement, on détermine, à partir de la distribution de référence, la probabilité d'obtenir un tel écart par rapport à la moyenne. Par exemple, si la suite pseudo-aléatoire à tester est de longueur 100, et possède 38 zéros et 62 uns, on obtient pour la formule (4.1) la valeur $x = 5,76$. On déduit de la distribution de probabilité du χ^2 que la probabilité pour qu'une suite aléatoire possède une valeur de X supérieure à 5,02 est $0,025$. Ainsi il y a moins de $2,5$ % de chance d'obtenir cette valeur pour une suite aléatoire. On décide donc que la suite testée passe ou ne passe le test en fonction d'un niveau de confiance, c'est-à-dire en fonction de la probabilité que ce test échoue pour une suite aléatoire.

De façon similaire, on peut tester la proportion de zéros et de uns dans des blocs de k bits. Parmi les autres tests de normalité classiques, on peut aussi mentionner le test de répétition, le test d'oscillation, le test du poker...

Les tests de compression. Ces tests statistiques déterminent si une suite peut être compressée de façon significative sans perte d'information, ce qui la distinguerait d'une suite aléatoire. Parmi ces tests, les plus connus sont le test universel de Maurer [13], le test de compression Lempel-Ziv, le test de l'entropie de Pincus, Singer et Kalman et le test de la complexité linéaire.

Les librairies de tests statistiques.

Il existe plusieurs librairies classiques qui fournissent une implémentation de séries de tests statistiques, dont la plus célèbre et la plus utilisée est celle du NIST.

La série de tests du NIST (National Institute of Standards and Technology), détaillée dans la publication spéciale du NIST 800-22 [14], et dont une description précise et une implémentation est disponible sur `http://csrc.nist.gov/rng/`. Cette librairie implémente 16 tests différents dont :

- les tests de normalité classiques comme le test de fréquence global et par blocs, le test de répétition...
- le test de rang de la matrice binaire,
- le test spectral (calcul de la transformée de Fourier discrète),
- le test de la complexité linéaire
- le test universel de Maurer
- le test de l'entropie
- le test de la complexité de Lempel-Ziv.

La série des tests DIEHARD développée par G. Marsaglia de l'Université de Floride, et disponible sur `http://stat.fsu.edu/ geo/diehard.html`

La série de tests Crypt-XS développée par le centre de recherche en sécurité de l'Université du Queensland en Australie, et disponible sur `http://www.isrc.qut.edu.au/cryptx/`

Bibliographie

[1] FIPS 81. " DES Modes of Operation ". Federal Information Processing Standards Publication 81, 1980. U.S. Department of Commerce/National Bureau of Standards.

[2] H. Gilbert. " The security of "One-Block-to-Many" Modes of Operation ". In *Fast Software Encryption - FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 376–395. Springer-Verlag, 2003.

[3] ISO/IEC 10116. " Information technology – Security techniques – Modes of operation for an n-bit block cipher ". International Organization for Standardization, 1997.

[4] NIST SP 800-38A. " Recommendation for Block Cipher Modes of Operation ". NIST Special Publication 800-38A, 2001. National Institute of Standards and Technology.

[5] ECRYPT - European Network of Excellence in Cryptology. " The eSTREAM Stream Cipher Project ". `http://www.ecrypt.eu.org/stream/`, 2005.

[6] F. Arnault and T.P. Berger. " F-FCSR : design of a new class of stream ciphers ". In *Fast Software Ecryption - FSE 2003*, volume 3557 of *Lecture Notes in Computer Science*, pages 83–97. Springer-Verlag, 2005.

[7] A. Klapper and M. Goresky. " Feedback shift registers, 2-adic span and combiners with memory ". *Journal of Cryptology*, 10(2), 1997.

[8] A. Klimov and A. Shamir. " A new class of invertible mappings ". In *CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 470–483. Springer-Verlag, 2002.

[9] A. Klimov and A. Shamir. " Cryptographic applications of T-functions ". In *Selected Areas in Cryptography - SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[10] H. Molland and T. Helleseth. " A Linear Weakness in T-functions ". In *Proceedings of the 2005 IEEE International Symposium on Information Theory - ISIT 2005*. IEEE, 2005.

[11] S.W. Golomb. *Shift register sequences*. Aegean Park Press, 1982.

[12] D. E. Knuth. *The Art of Computer Programming*, volume 2 - Seminumerical Algorithms. Addison Wesley, 1969.

[13] U. Maurer. " A universal statistical test for random bit generators ". In *Advances in Cryptology - CRYPTO'90*, volume 473 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[14] NIST SP 800-22. " A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications ". NIST Special Publication 800-22, 2000. National Institute of Standards and Technology.

Chapitre 5

Les LFSRs pour le chiffrement par blocs

5.1 Les LFSR

5.1.1 Introduction

Définition. Un registre à décalage à rétroaction linéaire, usuellement désigné par l'abréviation anglo-saxonne LFSR (pour Linear Feedback Shift Register), est un dispositif permettant d'engendrer une suite infinie qui satisfait une relation de récurrence linéaire. Plus précisément, un LFSR binaire de longueur L est composé d'un registre à L cellules contenant chacune un bit. Les L bits contenus dans le registre forment l'état interne du LFSR. Ces L cellules sont initialisées par L bits, $s_0, \ldots, s_{L-1}$.

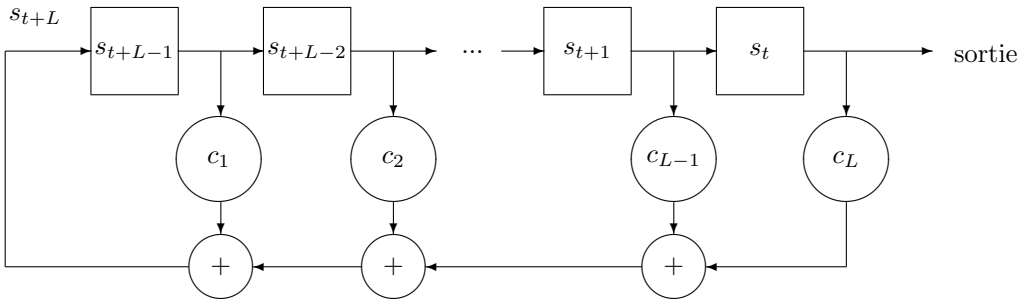


FIGURE 5.1 – Registre à décalage à rétroaction linéaire (LFSR)

Ce registre à décalage est contrôlé par une horloge externe. Au cours de chaque unité de temps, chaque bit est décalé d'une cellule vers la droite. Le contenu de la cellule la plus à droite, s_t, sort du registre, alors que la cellule la plus à gauche reçoit le bit de rétroaction, s_{t+L}. La valeur de ce dernier est obtenue par une combinaison linéaire des valeurs des autres cellules, dont les coefficients sont des éléments fixés qui valent 0 ou 1 et appelés *coefficients de rétroaction du LFSR* :

$$s_{t+L} = \sum_{i=1}^{L} c_i s_{t+L-i} \ ,$$

où la somme est une somme modulo 2 dans le cas d'un LFSR binaire.

Ainsi, le LFSR de longueur L et de coefficients de rétroaction $c_1, \dots c_L$ engendre, à partir de son état initial $s_0, \dots, s_{L-1}$, la suite obtenue par la relation de récurrence linéaire d'ordre L

$$s_{t+L} = \sum_{i=1}^{L} c_i s_{t+L-i} \text{ pour tout } t \geq 0 \ .$$

Exemple. Le tableau 5.1 donne les états successifs du LFSR binaire de longueur 4 dont les coefficients de rétroaction sont $c_1 = c_2 = 0$, $c_3 = c_4 = 1$ à partir de l'état initial $(s_0, s_1, s_2, s_3) = (1, 0, 1, 1)$. Ce LFSR est représenté à la figure 5.1.1. Il correspond à la relation de récurrence

$$s_{t+4} = s_{t+1} + s_t \bmod 2 \ .$$

La suite $s_0 s_1 \dots$ engendrée par ce LFSR est $1011100 \dots$.

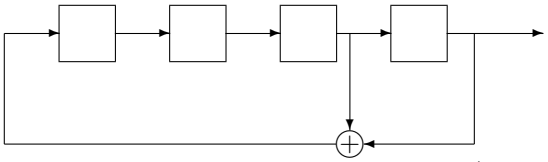


FIGURE 5.2 – LFSR binaire de coefficients de rétroaction $(c_1, c_2, c_3, c_4) = (0, 0, 1, 1)$

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_t	1	0	1	1	1	1	0	0	0	1	0	0	1	1	0	1
s_{t+1}	0	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0
s_{t+2}	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1
s_{t+3}	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1

TABLE 5.1 – tats successifs du LFSR de coefficients de rétroaction $(c_1, c_2, c_3, c_4) = (0, 0, 1, 1)$, initialisé par $(s_0, s_1, s_2, s_3) = (1, 0, 1, 1)$

Propriétés statistiques classiques. On peut démontrer que la suite engendrée par un LFSR possède de bonnes propriétés statistiques lorsque les coefficients de rétroaction du LFSR sont bien choisis. Ces propriétés dépendent essentiellement de la forme du polynôme utilisé pour représenter les coefficients de rétroaction. Ce dernier, appelé *polynôme de rétroaction du LFSR*, est défini par

$$P(X) = 1 + \sum_{i=1}^{L} c_i X^i \ .$$

Par exemple, le polynôme de rétroaction du LFSR de la figure 5.1.1 est $P(X) = 1 + X^3 + X^4$.

Les principales propriétés utiles dans le cadre du chiffrement à flot sont les suivantes.

– Si le polynôme de rétroaction est irréductible, alors la suite engendrée par le LFSR quelle que soit son initialisation (à part l'état nul) ne peut pas être engendrée par un LFSR plus court. La taille du plus petit LFSR permettant de produire une suite donnée est un paramètre fondamental en cryptographie, appelé *la complexité linéaire de la suite*. En effet, une suite de complexité linéaire Λ peut être entièrement reconstituée dès lors qu'un attaquant en connaît 2Λ bits consécutifs au moyen de l'algorithme de Berlekamp-Massey (pour plus de détails sur la complexité linéaire et une description complète de cet algorithme, voir la fiche *Complexité linéaire et algorithme de Berlekamp-Massey*). Autrement dit, le choix d'un polynôme de rétroaction irréductible garantit que la complexité linéaire de toute suite produite par le LFSR est maximale.

- Si le coefficient c_L est non nul (on dit dans ce cas que le LFSR est non-singulier), alors la suite engendrée par le LFSR est une suite périodique de période au plus $2^L - 1$. En effet, un registre de longueur L peut avoir au plus 2^L états différents, et l'état entièrement nul doit être exclu car il est toujours suivi de lui-même.
- Plus précisément, si le polynôme de rétroaction est un polynôme primitif, alors la plus petite période de la suite engendrée à partir de n'importe quelle initialisation (sauf l'état nul) est égale à $2^L - 1$.

Pour plus de détails sur ces propriétés ainsi que sur celles des LFSRs non binaires, voir la fiche *approfondissement sur les LFSRs.*

Utilisation d'un LFSR comme générateur pseudo-aléatoire. Il est évidemment impossible d'utiliser la suite produite par un LFSR comme suite chiffrante dans un chiffrement à flot. En effet, si les coefficients du LFSR sont publics (ce qui est généralement le cas quand le LFSR est implémenté sous forme d'un circuit électronique), il suffit à un attaquant qui connaît L bits consécutifs de la suite d'appliquer la relation de récurrence pour retrouver tous les bits suivants. Dans le cas où les coefficients de rétroaction sont secrets, l'algorithme de Berlekamp-Massey permet de les reconstituer, ainsi que l'état initial, à partir de $2L$ bits de suite chiffrante.

Toutefois, les LFSR sont des dispositifs extrêmement rapides et d'implémentation peu coûteuse pour engendrer des suites ayant de bonnes qualités statistiques, notamment une période élevée. C'est pour cette raison qu'ils sont souvent utilisés comme module de base dans les générateurs pseudo-aléatoires dédiés, mais au sein d'un dispositif plus complexe.

5.1.2 Approfondissement

LFSR q-aires. On peut définir un LFSR sur tout corps fini à q éléments, $\mathbf{F}_q$ de façon similaire au cas binaire. Le LFSR de longueur L sur $\mathbf{F}_q$ et de coefficients de rétroaction $c_1, \ldots, c_L$ est le dispositif permettant d'engendrer la suite semi-infinie $s_0, s_1, \ldots$, d'éléments de $\mathbf{F}_q$ qui satisfait la relation de récurrence linéaire

$$s_{t+L} = \sum_{i=1}^{L} c_i s_{t+L-i}, \quad \forall t \geq 0 \ .$$

Le polynôme de rétroaction du LFSR est alors défini par

$$P(X) = 1 - \sum_{i=1}^{L} c_i X^i \ .$$

On peut également représenter les coefficients de rétroaction du registre au moyen de son *polynôme minimal*, qui est le polynôme réciproque du polynôme de rétroaction :

$$P^\star(X) = X^L P(1/X) = X^L - \sum_{i=1}^{L} c_i X^{L-i} \ .$$

Les LFSR sur une extension du corps $\mathbf{F}_2$ sont classiquement utilisés dans les applications logicielles, où l'unité de base du processeur est l'octet ou le mot de 32 bits. On emploie alors des LFSR sur $\mathbf{F}_{2^8}$ ou $\mathbf{F}_{2^{32}}$.

LFSR non-singuliers. Un LFSR est dit non-singulier si le degré de son polynôme de rétroaction correspond à la longueur du registre (autrement dit si le coefficient de rétroaction c_L est non nul.) Toute suite engendrée par un LFSR q-aire non-singulier de longueur L est périodique de période inférieure ou égale à $q^L - 1$. Si le LFSR est singulier, alors la suite produite est ultimement périodique, ce qui signifie qu'on obtient une suite périodique si on enlève les premiers termes jusqu'à un certain rang.

Caractérisation des suites produites par un LFSR. Un LFSR q-aire donné de longueur L peut produire q^L suites différentes, qui correspondent aux q^L initialisations, et ces suites forment un espace vectoriel sur $\mathbf{F}_q$.

L'ensemble des suites produites par le LFSR de polynôme de rétroaction P est caractérisé par la propriété suivante. Une suite $(s_t)_{t\geq 0}$ est produite par un LFSR q-aire de longueur L et de polynôme de rétroaction P si et seulement s'il existe un polynôme $Q \in \mathbf{F}_q[X]$ de degré strictement inférieur à L tel que le développement en série formelle de $(s_t)_{t\geq 0}$ vérifie

$$\sum_{t\geq 0} s_t X^t = \frac{Q(X)}{P(X)} \ .$$

De plus, le polynôme Q est entièrement déterminé par les coefficients de P et l'état initial du registre :

$$Q(X) = -\sum_{i=0}^{L-1} X^i \left(\sum_{j=0}^{i} c_{i-j} s_j \right) \ ,$$

où $P(X) = \sum_{i=0}^{L} c_i X^i$. Ceci signifie qu'il y a une bijection entre les suites engendrées par un LFSR de longueur L et de polynôme de rétroaction P et les fractions rationnelles $Q(X)/P(X)$ avec $\deg(Q) < L$. Ce résultat fondamental a deux conséquences importantes du point de vue du chiffrement à flot.

Tout d'abord, toute suite produite par le LFSR de polynôme de rétroaction P est également engendrée par tout LFSR dont le polynôme de rétroaction est multiple de P. Cette propriété est utilisée dans plusieurs attaques sur les LFSRs [3, 5, 6].

De même, une suite produite par le LFSR de polynôme de rétroaction P est également engendrée par un LFSR plus court, de polynôme de rétroaction P' si les polynômes P et Q intervenant dans la fraction rationnelle $Q(X)/P(X)$ ne sont pas premiers entre eux. Par conséquent, parmi toutes les suites produites par le LFSR de polynôme de rétroaction P, il y en a au moins une qui peut être engendrée par un LFSR plus court dès que P n'est pas irréductible.

Polynôme minimal. On déduit de la propriété précédente que, pour toute suite $(s_t)_{t\geq 0}$ vérifiant une relation de récurrence linéaire, il existe un unique polynôme unitaire P_0 tel que le développement en série formelle associé est donné par $Q_0(X)/P_0(X)$ où P_0 et Q_0 sont premiers entre eux. Ainsi, le plus petit LFSR permettant d'engendrer $(s_t)_{t\geq 0}$ a pour longueur $L = \max(\deg(P_0), \deg(Q_0) + 1)$, et son polynôme de rétroaction est égal à P_0. Le polynôme réciproque de P_0, $X^L P_0(1/X)$, est donc le polynôme caractéristique du plus petit LFSR produisant $(s_t)_{t\geq 0}$. Il est appelé *polynôme minimal* de la suite. C'est lui qui détermine la relation de récurrence linéaire d'ordre minimal vérifiée par la suite.

Le degré du polynôme minimal d'une suite récurrente linéaire correspond à sa *complexité linéaire*. Il s'agit de la longueur du plus petit LFSR qui permette d'engendrer cette suite. Le polynôme minimal d'une suite $\mathbf{s} = (s_t)_{t\geq 0}$ de complexité linéaire $\Lambda(\mathbf{s})$ peut être déterminé à partir de la connaissance de $2\Lambda(\mathbf{s})$ bits consécutifs de $\mathbf{s}$ au moyen de l'algorithme de Berlekamp-Massey. Pour plus de précisions sur la notion de complexité linéaire et sur l'algorithme de Berlekamp-Massey, voir la fiche correspondant.

Exemple. Considérons le LFSR binaire de longueur 10 décrit à la figure 5.3. Ce LFSR a pour polynôme de rétroaction

$$P(X) = 1 + X + X^3 + X^4 + X^7 + X^{10} \ ,$$

et son état initial $s_0 \dots s_9$ est 1001001001.

La série génératrice de la suite produite par ce LFSR est donnée par

$$\sum_{t\geq 0} s_t X^t = \frac{Q(X)}{P(X)}$$

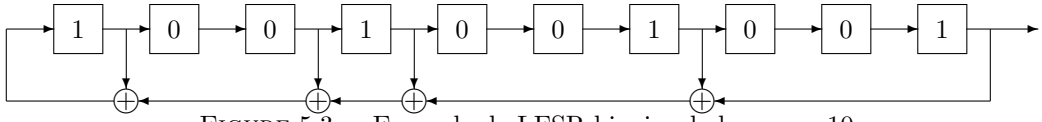


FIGURE 5.3 – Exemple de LFSR binaire de longueur 10

où Q est obtenu à partir des coefficients de P et de l'état initial :

$$Q(X) = 1 + X + X^7 \ .$$

On a donc

$$\sum_{t\geq 0} s_t X^t = \frac{1 + X + X^7}{1 + X + X^3 + X^4 + X^7 + X^{10}} = \frac{1}{1 + X^3} \ ,$$

car $1 + X + X^3 + X^4 + X^7 + X^{10} = (1 + X + X^7)(1 + X^3)$ dans $\mathbf{F}_2[X]$. Ceci implique que $(s_t)_{t\geq 0}$ est également engendrée par le LFSR de polynôme de rétroaction $P_0(X) = 1 + X^3$ décrit à la figure 5.4, et que sa complexité linéaire est égale à 3.

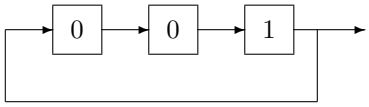


FIGURE 5.4 – LFSR de longueur 3 engendrant la même suite que le LFSR de la figure 5.3

Période de la suite engendrée par un LFSR. Le polynôme minimal d'une suite récurrente linéaire joue un rôle majeur puisqu'il détermine à la fois la complexité linéaire et la plus petite période de la suite. En effet, la plus petite période d'une suite récurrente linéaire est égale à l'ordre de son polynôme minimal. L'ordre d'un polynôme P de $\mathbf{F}_q[X]$ tel que $P(0) \neq 0$ est le plus petit entier positif e tel que $P(X)$ divise $X^e - 1$. En conséquence, $\mathbf{s}$ est de période maximale $q^{\Lambda(\mathbf{s})} - 1$ si et seulement si son polynôme minimal est *primitif* (c'est-à-dire d'ordre maximal).

Par exemple, la suite produite par le LFSR de la figure 5.4 est de période 3 car son polynôme minimal $X^3 + 1$ est d'ordre 3. On peut vérifier que ce LFSR produit en effet la suite $100100100\ldots$.

Au contraire, toutes les suites engendrées par le LFSR de la figure 5.5, sauf la suite entièrement nulle, sont de période 15. En effet, le polynôme minimal d'une telle suite correspond au polynôme caractéristique

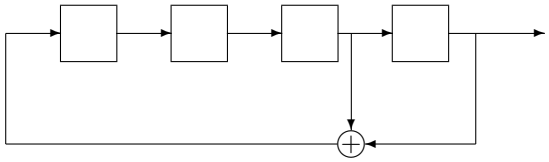


FIGURE 5.5 – Exemple de LFSR de longueur 4 de période maximale

du LFSR $P^\star(X) = 1 + X + X^4$, puisque celui-ci est irréductible. De plus, $P^\star$ est un polynôme primitif.

On voit ici que toute suite $\mathbf{s} = (s_t)_{t\geq 0}$ produite par un LFSR q-aire de longueur L dont le polynôme de rétroaction est primitif est à la fois de complexité linéaire maximale, $\Lambda(\mathbf{s}) = L$ et de période maximale $q^L - 1$. Ces suites sont appelées *suites récurrentes linéaires de longueur maximale* (m-sequences en anglais).

Du fait de leur optimalité, ce sont elles qui sont utilisées dans les générateurs pseudo-aléatoires. En d'autres termes, tout LFSR entrant dans la construction d'un générateur pseudo-aléatoire doit avoir un polynôme de rétroaction primitif.

Propriétés statistiques des suites récurrentes linéaires de longueur maximale. Les suites récurrentes linéaires de longueur maximale, c'est-à-dire les suites engendrées par un LFSR ayant un polynôme de rétroaction primitif, possèdent un certain nombre de propriétés statistiques qui font qu'elles sont souvent utilisées comme briques de base de générateurs pseudo-aléatoires.

Ainsi, toute suite produite par un LFSR binaire de longueur L ayant un polynôme de rétroaction primitif présente notamment les caractéristiques suivantes :

- *équilibre :* l'écart entre le nombre de zéros et le nombre de uns dans tout ensemble de $2^L - 1$ bits consécutifs de la suite est de 1.
- *propriétés des plages :* on appelle *plage* (*run* en anglais) tout ensemble de 0 consécutifs encadrés par deux 1, ou tout ensemble de 1 consécutifs encadrés par deux 0. Par exemple, dans la suite 0100011 figure une plage de 0 de longueur 4. Si l'on considère l'ensemble des plages à l'intérieur de $(2^L - 1)$ bits consécutifs d'une suite récurrente linéaire de longueur maximale, alors la moitié d'entre elles sont des plages de longueur 1, un quart sont des plages de longueur 2, un huitième sont des plages de longueur 3,..., et finalement une seule plage est de longueur $(L-1)$ et une seule est de longueur L. De plus, parmi toutes les plages de longueur $k \leq L-2$, il y a autant de plages de 0 que de plages de 1.
- *auto-corrélation à deux niveaux :* la fonction d'auto-corrélation pour une suite binaire de période N est définie par

$$C(\tau) = \sum_{t=0}^{N-1} (-1)^{s_t + s_{t+\tau}} \ .$$

  La valeur de $C(\tau)$ mesure donc la distance entre la suite $s_0 s_1 \dots s_{N-1}$ et la suite obtenue en la décalant de τ positions, c'est-à-dire $s_\tau s_{\tau+1} \dots s_{N-1} s_0 s_1 \dots s_{\tau-1}$. En effet, $C(\tau)$ correspond à la différence entre le nombre de positions en lesquelles ces deux suites coïncident et le nombre de positions en lesquelles elles diffèrent. Autrement dit

$$C(\tau) = N - 2|\{t,\ 0 \leq t \leq N-1,\ \ s_t \neq s_{t+\tau \bmod N}\}| \ .$$

  On a donc trivialement que $C(\tau) = N$ si τ est un multiple de la période N puisque les deux suites considérées sont identiques. De plus, toute suite produite par un LFSR binaire de polynôme de rétroaction primitif vérifie

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

  pour tout τ qui n'est pas un multiple de $2^L - 1$. Cette propriété est notamment utilisée en télécommunications dans les techniques de synchronisation.
- *propriétés du multigramme :* le L-uplet $(s_t, s_{t+1}, \dots, s_{t+L-1})$ parcourt l'ensemble des $2^L - 1$ L-uplets non nuls quand t varie entre 0 et $2^L - 2$.

Les trois premières propriétés détaillées précédemment sont connues sous le nom de postulats de Golomb [11].

Exemple : les 31 premiers bits de la suite engendrée à partir de l'état initial 10000 par le LFSR de longueur 5 de polynôme de rétroaction primitif $x^5 + x^3 + 1$ sont

$$1000010101110110001111100110100$$

On peut vérifier que cette suite comporte 16 uns et 15 zéros. Elle est donc équilibrée.

Cette suite est constituée de 16 plages, parmi lesquelles on trouve 8 plages de longueur 1 (4 de zéros et 4 de uns), 4 plages de longueur 2 (2 de zéros et 2 de uns), 2 plages de longueur 3 (une de zéros et une de uns), une plage de zéros de longueur 4 et une plage de uns de longueur 5.

5.1.3 Complexité linéaire et algorithme de Berlekamp-Massey

Définition.

- La *complexité linéaire d'une suite infinie* $\mathbf{s} = (s_t)_{t\geq 0}$, notée $\Lambda(\mathbf{s})$, est le plus entier Λ tel que $\mathbf{s}$ peut être engendrée par un LFSR de longueur Λ ; elle est infinie si aucun LFSR ne permet de l'engendrer. Par convention, la complexité linéaire de la suite nulle est égale à 0. La complexité linéaire d'une suite vérifiant une relation de récurrence linéaire correspond au degré de son polynôme minimal.
- La *complexité linéaire d'une suite finie* $\mathbf{s^n} = s_0 s_1 \dots s_{n-1}$ de n éléments, notée $\Lambda(\mathbf{s^n})$, est la longueur du plus petit LFSR qui permette de produire une suite dont les n premiers éléments correspondent à $\mathbf{s^n}$. Pour toute suite finie $\mathbf{s^n}$ de longueur n, le LFSR de longueur $\Lambda(\mathbf{s^n})$ qui engendre $\mathbf{s^n}$ est unique si et seulement si $n \geq 2\Lambda(\mathbf{s^n})$ [8].

La complexité linéaire d'une suite infinie $\mathbf{s}$ et celle de la suite finie composée de ses n premiers éléments sont liées par la propriété suivante. Si la suite infinie $\mathbf{s}$ est de complexité linéaire Λ, alors $\mathbf{s^n}$ est également de complexité linéaire Λ si $n \geq 2\Lambda$. Dans ce cas, $\mathbf{s^n}$ correspond aux n premiers éléments produit par l'unique LFSR de longueur Λ qui engendre $\mathbf{s}$ [8].

Une notion plus fine est celle de *profil de complexité linéaire* d'une suite infinie $\mathbf{s} = s_0 s_1 \dots$. On désigne par ce terme la suite $(\Lambda(\mathbf{s^n}))_{n\geq 1}$ composée des complexités linéaires de chacune des sous-suites $\mathbf{s^n} = s_0 \dots s_{n-1}$ formées par les n premiers éléments de $\mathbf{s}$.

Complexité linéaire d'une suite aléatoire binaire. L'espérance de la complexité linéaire d'une suite binaire $\mathbf{s^n} = s_0 \dots s_{n-1}$ de n variables aléatoires binaires indépendantes et uniformément distribuées est donnée par

$$E[\Lambda(\mathbf{s}^n)] = \frac{n}{2} + \frac{4 + \varepsilon(n)}{18} + 2^{-n}\left(\frac{n}{3} + \frac{2}{9}\right) ,$$

où $\varepsilon(n) = n \bmod 2$.

Dans le cas d'une suite infinie binaire de période 2^n obtenue en répétant une suite $s_0 \dots s_{2^n-1}$ de 2^n variables aléatoires binaires indépendantes et uniformément distribuées, l'espérance de la complexité linéaire est

$$E[\Lambda(\mathbf{s})] = 2^n - 1 + 2^{-2^n} .$$

De plus amples résultats, notamment sur le profil de complexité linéaire d'une suite aléatoire, peuvent être trouvés dans [2].

Algorithme de Berlekamp-Massey. L'algorithme de Berlekamp-Massey est un algorithme qui permet de déterminer la complexité linéaire d'une suite finie, ainsi que le polynôme de rétroaction d'un LFSR de longueur minimale qui permet de l'engendrer. Cet algorithme est dû à Massey [8] qui a montré que l'algorithme itératif de décodage des codes BCH proposé par Berlekamp [7] pouvait également être utilisé pour trouver le plus petit LFSR engendrant une suite donnée.

Pour une suite $\mathbf{s}^n$ de longueur n, l'algorithme de Berlekamp-Massey effectue n itérations. La t-ième itération détermine en fait un LFSR de longueur minimale qui engendre les t premiers éléments de $\mathbf{s}^n$. L'algorithme se déroule de la manière suivante.

Dans le cas particulier d'une suite binaire, il n'est pas nécessaire de stocker la valeur de d' puisque celle-ci est toujours égale à 1. Dans ce cas, le polynôme de rétroaction est remis à jour simplement par la formule

$$P(X) \leftarrow P(X) + P'(X)X^{t-m} .$$

Le nombre d'opérations effectuées pour calculer la complexité linéaire d'une suite de longueur n est proportionnel à n^2.

Il est important de noter que le LFSR de longueur minimale qui génère une suite $\mathbf{s^n}$ de longueur n est unique si et seulement si $n \geq 2\Lambda(\mathbf{s^n})$.

Le tableau suivant décrit par exemple le déroulement de l'algorithme de Berlekamp-Massey appliqué à la suite binaire de longueur 7, $s_0 \dots s_6 = 0111010$. Les valeurs de Λ et P obtenues à la fin de l'itération t

Entrée. $\mathbf{s^n} = s_0 s_1 \dots s_{n-1}$, suite q-aire de n éléments de $\mathbf{F}_q$.

Sortie. Λ, la complexité linéaire $\mathbf{s^n}$ et P, le polynôme de rétroaction d'un LFSR de longueur Λ qui engendre $\mathbf{s^n}$.

Initialisation.

$P(X) \leftarrow 1$, $P'(X) \leftarrow 1$, $\Lambda \leftarrow 0$, $m \leftarrow -1$, $d' \leftarrow 1$.

Pour t **de** 0 **à** $n-1$ **faire**

$d \leftarrow s_t + \sum_{i=1}^{\Lambda} p_i s_{t-i}$.

Si $d \neq 0$ alors

$T(X) \leftarrow P(X)$.

$P(X) \leftarrow P(X) - d(d')^{-1} P'(X) X^{t-m}$.

si $2\Lambda \leq t$ alors

$\Lambda \leftarrow t + 1 - \Lambda$.

$m \leftarrow t$.

$P'(X) \leftarrow T(X)$.

$d' \leftarrow d$.

Retourner Λ et P.

correspondent à la complexité linéaire de la suite $s_0 \dots s_t$ et au polynôme de rétroaction d'un LFSR de taille minimale permettant de l'engendrer.

t	s_t	d	Λ	$P(X)$	m	$P'(X)$
			0	1	-1	1
0	0	0	0	1	-1	1
1	1	1	2	$1+X^2$	1	1
2	1	1	2	$1+X+X^2$	1	1
3	1	1	2	$1+X$	1	1
4	1	0	2	$1+X$	1	1
5	0	1	4	$1+X+X^4$	5	$1+X$
6	0	0	4	$1+X+X^4$	5	$1+X$

TABLE 5.2 – Déroulement de l'algorithme de Berlekamp-Massey appliqué à $s_0 \dots s_6 = 0111010$

5.2 Les générateurs pseudo-aléatoires à base de LFSRs

Il est évident qu'un registre à décalage à rétroaction linéaire seul produit une suite aux piètres propriétés cryptographiques. En effet, la connaissance du polynôme de rétroaction permet de deviner, à partir de L bits consécutifs de la suite produite, tous les bits suivants, où L est la longueur du registre utilisé (ou plus précisément sa complexité linéaire). Même lorsque le polynôme de rétroaction est secret (ce qui est relativement rare car il est généralement câblé dans les implémentations matérielles), l'algorithme de Berlekamp-Massey permet de déterminer la suite à partir de la connaissance de $2L$ bits consécutifs. Pour engendrer une suite pseudo-aléatoire au moyen de LFSRs de taille raisonnable et bénéficier de leur simplicité d'implémentation, il est donc indispensable d'introduire une composante non-linéaire dans le

système. Ceci peut être fait au moyen de deux constructions classiques : la construction par combinaison de LFSRs et la construction dite du LFSR filtré. On suppose bien entendu dans toute la suite que les LFSRs utilisés dans ces constructions sont définis par des polynômes de rétroaction primitifs.

5.2.1 Combinaison de LFSRs

Principe. Un générateur par combinaison de LFSRs est composé de n LFSRs qui fonctionnent en parallèle, et dont les sorties sont combinées au moyen d'une fonction booléenne à n variables. C'est la sortie de cette fonction f à l'instant t qui fournit le bit correspondant de suite chiffrante :

$$s_t = f(x_t^1, x_t^2, \ldots, x_t^n) \ .$$

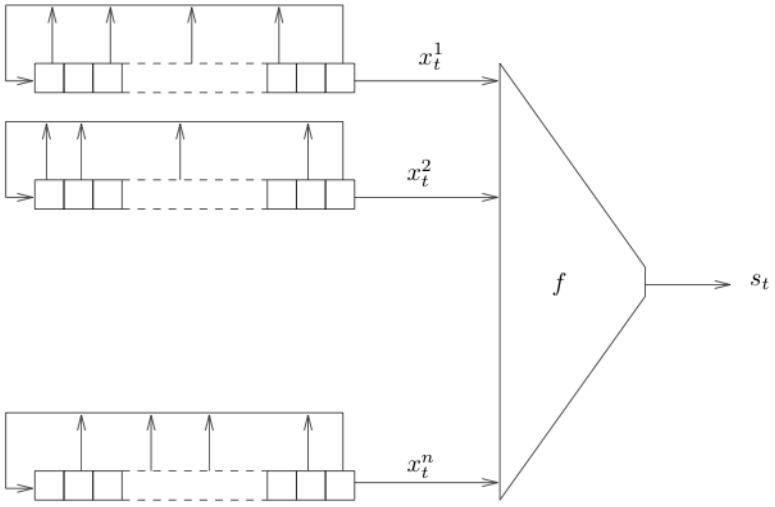


FIGURE 5.6 – Générateur pseudo-aléatoire par combinaison de LFSR

Par exemple, le générateur proposé par Geffe en 1973 [9] (*cf.* Figure 5.7) est composé de trois LFSRs de longueurs distinctes combinés par la fonction

$$f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3 \ .$$

Les systèmes par combinaison de registres visent généralement les implémentations matérielles dans lesquelles les différents LFSRs peuvent évoluer en parallèle. Ils sont évidemment peu adaptés aux implémentations logicielles dans la mesure où les registres y sont mis à jour l'un après l'autre. Dans un contexte logiciel, on leur préférera donc souvent un LFSR filtré d'état interne de grande taille. En effet, le fait de diviser l'état interne en plusieurs parties mises à jour indépendamment ne présente aucun avantage en terme d'implémentation, et est souvent à l'origine de failles de sécurité, notamment d'attaques de type diviser pour mieux régner comme les attaques par corrélation.

Propriétés statistiques. La suite chiffrante $\mathbf{s}$ obtenue en sortie d'un générateur par combinaison de LFSRs est également une suite récurrente linéaire. Sa période et sa complexité linéaire dépendent à la fois des LFSRs utilisés et de la forme algébrique normale, c'est-à-dire de l'écriture polynomiale, de la fonction de combinaison. En effet, si l'on considère deux suites $\mathbf{u}$ et $\mathbf{v}$ de complexité linéaire $\Lambda(\mathbf{u})$ et $\Lambda(\mathbf{v})$, c'est-à-dire produites par des LFSRs de longueur $\Lambda(\mathbf{u})$ et $\Lambda(\mathbf{v})$, on a :

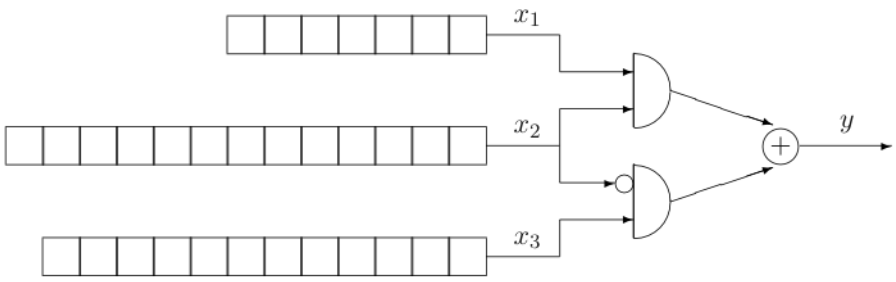


FIGURE 5.7 – Générateur de Geffe

– la suite $\mathbf{u} + \mathbf{v} = (u_t + v_t)_{t \geq 0}$ est une suite récurrente linéaire, dont la complexité linéaire satisfait

$$\Lambda(\mathbf{u} + \mathbf{v}) \leq \Lambda(\mathbf{u}) + \Lambda(\mathbf{v}) \ ,$$

avec égalité si et seulement si les polynômes minimaux de $\mathbf{u}$ et de $\mathbf{v}$ sont premiers entre eux. De plus, en cas d'égalité, la période de $\mathbf{u} + \mathbf{v}$ est le PPCM des périodes de $\mathbf{u}$ et de $\mathbf{v}$.

– la suite $\mathbf{uv} = (u_t v_t)_{t \geq 0}$ est une suite récurrente linéaire, dont la complexité linéaire satisfait

$$\Lambda(\mathbf{uv}) \leq \Lambda(\mathbf{u})\Lambda(\mathbf{v}) \ ,$$

avec égalité dès que les polynômes minimaux de $\mathbf{u}$ et de $\mathbf{v}$ sont primitifs et que $\Lambda(\mathbf{u}) \neq \Lambda(\mathbf{v})$ et sont tous deux supérieurs ou égaux à 2. Il existe d'autres conditions suffisantes d'égalité, moins restrictives que la précédente (*cf.* par exemple [11, 14, 10].

Ainsi, la suite chiffrante $\mathbf{s}$ qui résulte de la combinaison par f de n LFSRs binaires dont les polynômes de rétroaction sont primitifs vérifie la propriété suivante [14] : si les longueurs des LFSRs $L_1, \ldots, L_n$ sont toutes distinctes et supérieures à 2 (et que l'état initial de chacun des LFSRs est non nul), alors la complexité linéaire de $\mathbf{s}$ est égale à

$$\Lambda(\mathbf{s}) = f(L_1, L_2, \ldots, L_n)$$

où le polynôme représentant f est ici évalué sur les entiers. Par exemple, le générateur de Geffe décrit à la Figure 5.7, composé de trois LFSRs de longueurs L_1, L_2 et L_3, engendre une suite de complexité linéaire

$$\Lambda(\mathbf{s}) = L_1 L_2 + L_2 L_3 + L_3 \ .$$

Sécurité. La sécurité d'un générateur par combinaison de LFSRs dépend fortement de la forme de la fonction f utilisée pour assembler les différents registres. Cette fonction doit en particulier posséder un certain nombre de propriétés cryptographiques, notamment être équilibrée, et avoir un degré, un ordre de non-corrélation et une non-linéarité élevés (voir la fiche *La sécurité des générateurs par combinaison de LFSRs* pour plus de détails). Toutefois, une analyse précise de la complexité de différentes attaques publiées récemment montre qu'il semble difficile à l'heure actuelle de concevoir un générateur par combinaison de LFSRs qui offre une sécurité suffisante. Il paraît notamment nécessaire d'ajouter à cette construction d'autres composantes, par exemple, de la mémoire dans la fonction de combinaison, comme dans E0 ou un mécanisme de contrôle irrégulier de l'horloge comme dans A5/1.

5.2.2 LFSR filtré

Principe. Un LFSR filtré est composé d'un unique LFSR dont l'état interne est filtré au moyen d'une fonction non-linéaire f. Ainsi, si on note $(u_t)_{t \geq 0}$ la suite engendrée par le LFSR, la suite chiffrante $\mathbf{s}$ produite par le LFSR filtré à l'instant t est définie par

$$s_t = f(u_{t+\gamma_1}, u_{t+\gamma_2}, \ldots, u_{t+\gamma_n})$$

où le nombre de variables d'entrées de f, n, est inférieur ou égal à la longueur du LFSR et $(\gamma_i)_{1\leq i\leq n}$ est une suite décroissante d'entiers positifs qui définit les positions des bits de l'état interne pris en entrée de la fonction de filtrage.

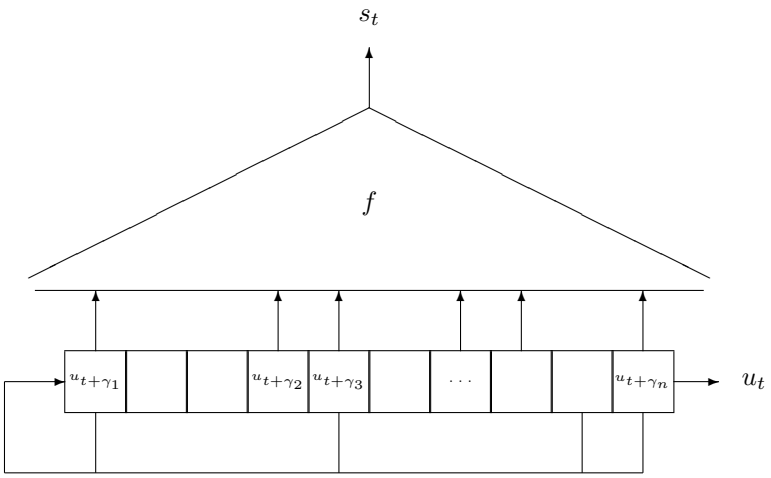


FIGURE 5.8 – Générateur pseudo-aléatoire par LFSR filtré

Propriétés statistiques. La suite chiffrante $\mathbf{s}$ obtenue en sortie d'un LFSR filtré est également une suite récurrente linéaire. Sa période et sa complexité linéaire sont liées à la longueur du LFSR utilisé et au degré de la fonction de filtrage f. Pour un LFSR binaire de longueur L et dont le polynôme de rétroaction est primitif, la complexité linéaire de $\mathbf{s}$ satisfait [12, 13]

$$\Lambda(\mathbf{s}) \leq \sum_{i=0}^{\deg(f)} \binom{L}{i},$$

et la période de $\mathbf{s}$ est un diviseur de $2^L - 1$. De plus, si L est un nombre premier de grande taille, on a

$$\Lambda(\mathbf{s}) \geq \binom{L}{\deg(f)}$$

pour la plupart des fonctions de filtrage f [2].

Tout LFSR filtré est équivalent à une combinaison de LFSRs. En effet, si l'on considère la combinaison par la fonction f de n copies du LFSR utilisé dans le générateur filtré, mais dont les états initiaux sont décalés de γ_i positions vers la droite, on voit que la sortie de ce générateur par combinaison est identique à la sortie du registre filtré.

Sécurité. De manière générale, la sécurité d'un générateur par LFSR filtré dépend fortement du polynôme de rétroaction du registre et du choix de la fonction de filtrage f. Ils devront nécessairement respecter un certain nombre de critères, sinon le générateur sera vulnérable à certaines attaques bien connues. En particulier, la fonction f doit être équilibrée, avoir un degré, une non-linéarité et une immunité algébrique élevés et un ordre d'immunité aux corrélation au moins égal à 1. Par ailleurs, le polynôme de rétroaction du LFSR doit être relativement dense et ne doit pas posséder de multiples creux de degré faible. Enfin, les positions d'entrées de la fonction de filtrage, $(\gamma_i)_{1\leq i\leq n}$, doivent être choisies de telle sorte que tous les écarts $|\gamma_i - \gamma_j|$ soient premiers entre eux, ou si cela n'est pas possible, il est indispensable de minimiser le nombre d'écarts $|\gamma_i - \gamma_j|$ ayant un diviseur commun (voir la fiche *La sécurité des LFSRs filtrés* pour plus de détails).

5.2.3 Les attaques par corrélation

Principe. Les attaques par corrélation entrent dans la catégorie plus générale des attaques de type *diviser pour régner* qui s'appliquent à chaque fois que l'on peut décomposer un système en composantes plus petites, cryptographiquement faibles.

Dans le cas du chiffrement à flot, ces attaques ont été introduites en 1985 par Siegenthaler [17] contre les générateurs par combinaison de LFSRs. Mais, cette cryptanalyse est en fait valide sur tous les générateurs pseudo-aléatoires dont l'état interne est décomposable en plusieurs parties mises à jour indépendamment les unes des autres. On peut alors chercher séparément la valeur initiale de chaque partie de l'état interne.

L'attaque repose sur l'existence d'éventuelles corrélations entre la sortie de la fonction de filtrage et un sous-ensemble de ses entrées qui correspond à la partie incriminée de l'état interne.

Description. Supposons comme à la figure 5.9 que l'on peut séparer l'état interne du générateur à l'instant t en deux parties x_t et y_t de tailles respectives ℓ et $(n-\ell)$, mises à jour indépendamment par Φ_0 et Φ_1.

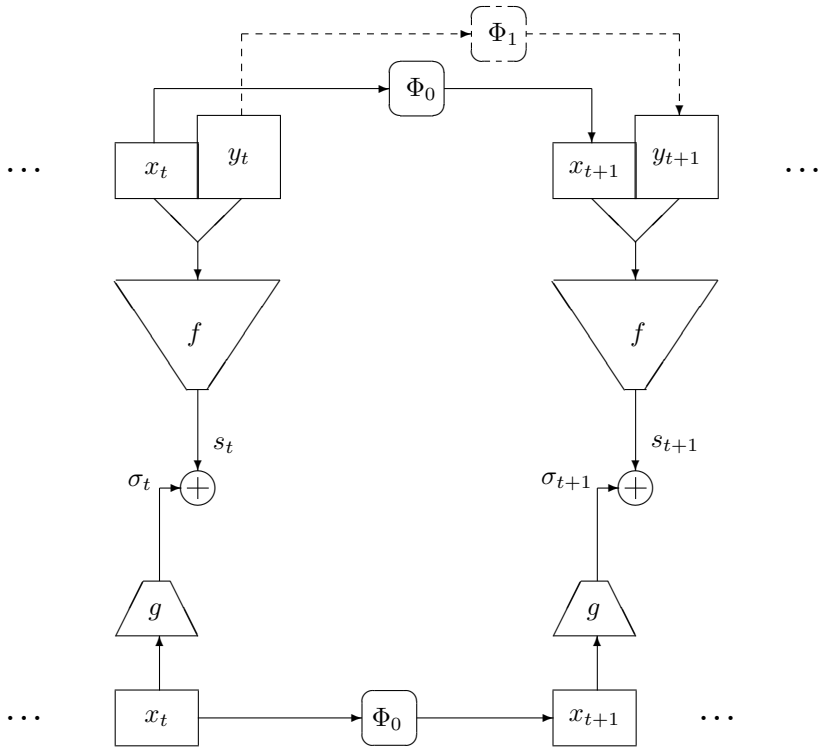


FIGURE 5.9 – Modèle de l'attaque par corrélation

Plaçons-nous dans le cas où l'attaquant cherche à retrouver la valeur de la première partie de l'état initial, x_0. Le vecteur d'entrée de la fonction de filtrage f se décompose de la même manière en deux parties, x et y. On peut alors appliquer l'attaque s'il existe une fonction g à ℓ variables (c'est-à-dire ne dépendant que de x) qui coïncide avec la sortie de f dans plus de la moitié des cas, autrement dit si la probabilité

$$p_g = Pr_{X,Y}[f(X,Y) = g(X)] > \frac{1}{2} \ .$$

La suite $\boldsymbol{\sigma}(x_0)$ produite par le générateur réduit d'état initial x_0 et de fonction de filtrage g est alors

corrélée à la suite chiffrante **s** car, pour tout $t \geq 0$,

$$Pr[s_t = \sigma_t] = p_g > \frac{1}{2} \ .$$

Dans ce cas, l'attaque par corrélation consiste à effectuer une recherche exhaustive sur la partie incriminée de l'état initial, x_0, au moyen de l'algorithme suivant.

Entrée. les N premiers bits de la sortie du générateur pseudo-aléatoire, $(s_t)_{t<N}$.

Sortie. x_0, vecteur de ℓ bits correspondant à une partie de l'état initial.

Algorithme

Pour chaque vecteur x_0 de ℓ bits

– Calculer les N bits de la suite $\boldsymbol{\sigma}(x_0)$ définie par

$$\sigma_t = g \circ \Phi_0^t(x_0) \ .$$

– Calculer la corrélation sur N bits entre les suites $\boldsymbol{s}$ et $\boldsymbol{\sigma}(x_0)$:

$$c(\boldsymbol{s}, \boldsymbol{\sigma}(x_0)) = \sum_{t=0}^{N-1} (-1)^{\sigma_t + s_t} \ .$$

Retourner la valeur de x_0 qui maximise $c(\boldsymbol{s}, \boldsymbol{\sigma}(x_0))$.

TABLE 5.3 – Attaque par corrélation

Au lieu de retourner la valeur la plus probable pour x_0, on peut également choisir de retourner l'ensemble des valeurs pour lesquelles $c(\boldsymbol{s}, \boldsymbol{\sigma}(x_0))$ dépasse un certain seuil.

L'attaque repose sur le fait que, lorsque la valeur de x_0 essayée ne correspond pas à la valeur effectivement employée dans l'initialisation, alors les suites $\boldsymbol{s}$ et $\boldsymbol{\sigma}(x_0)$ ne sont pas corrélées et la valeur moyenne de la corrélation est nulle. En revanche, lorsque la valeur essayée x_0 est correcte, on a

$$E[c(\boldsymbol{s}, \boldsymbol{\sigma}(x_0))] = 2N \left(p_g - \frac{1}{2} \right) \ .$$

Complexité. Le nombre de bits N de suite chiffrante nécessaires pour retrouver la valeur correcte de x_0 est de l'ordre de

$$\left(\frac{1}{p_g - \frac{1}{2}} \right)^2 \ .$$

La complexité en temps pour retrouver les ℓ bits x_0 de l'état initial est de l'ordre de

$$\frac{2^\ell}{(p_g - \frac{1}{2})^2} \ ,$$

mais peut être réduite à

$$\ell \, 2^\ell$$

quand les fonctions Φ_0 et g sont des fonctions linéaires, grâce à un algorithme de transformée de Fourier rapide [16]. Dans ce dernier cas, la complexité en temps de l'attaque peut être considérablement améliorée en employant les attaques dites *par corrélation rapide* (voir la fiche d'approfondissement sur ce sujet).

Exemple : attaque du générateur de Geffe Considérons par exemple le générateur par combinaison de LFSR, qui suit le modèle proposé par Geffe en 1973 [9]. Il est composé de trois LFSR de longueurs respectives 7, 12 et 13 combinés par la fonction

$$f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3 \ .$$

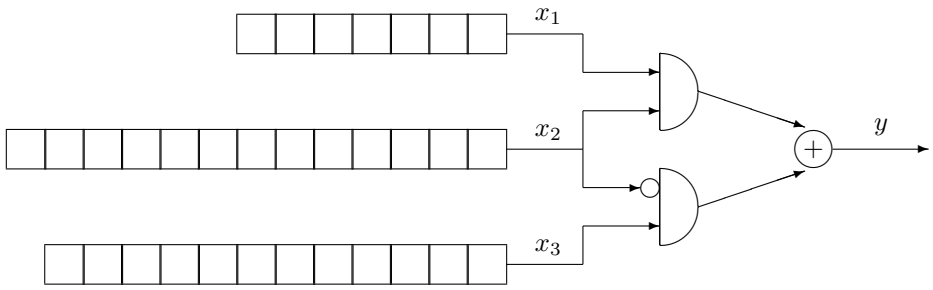


FIGURE 5.10 – Générateur de Geffe

Une recherche exhaustive sur la clef secrète, qui comporte 7+12+13 = 32 bits, nécessiterait d'examiner ses $2^{32} = 4.294.967.296$ valeurs possibles. En fait, on peut déterminer de manière indépendante les 7 premiers bits de la clef, qui correspondent à l'initialisation du premier registre. On considère les $2^7 = 128$ initialisations possibles pour ce registre. Pour chacune d'entre elles, on génère les 100 premiers bits produits par le premier registre initialisé de la sorte et on compare ces 100 bits avec les 100 premiers bits produits par le générateur. Si l'initialisation du premier registre que l'on essaie est correcte, ces deux suites de 100 bits vont coïncider sur environ 75 bits puisque la sortie du générateur est égale à la sortie du premier registre dans 75 % des cas. Par contre, si l'initialisation essayée n'est pas la bonne, les deux suites que l'on compare ne sont pas corrélées. Autrement dit, elles vont coïncider sur environ la moitié des positions. Ainsi, la connaissance des 100 premiers bits produits par le générateur suffit à déterminer l'initialisation du premier registre en 128 essais. De la même façon, on peut ensuite retrouver l'initialisation du troisième registre (de longueur 12) en $2^{12} = 4096$ essais en exploitant le fait que la sortie du générateur coïncide également dans 75 % des cas avec la suite engendrée par le troisième registre. Enfin, il ne reste plus qu'à retrouver les 13 bits de clef restant à déterminer (l'initialisation du deuxième registre) par une recherche exhaustive. L'attaque complète du générateur a donc nécessité $2^7 + 2^{12} + 2^{13} = 12.416$ essais au lieu des $2^{32} = 4.294.967.296$ demandés par une recherche exhaustive de la clef.

Résistance aux attaques par corrélation. La fonction g étant choisie par l'attaquant, celui-ci doit naturellement utiliser celle qui conduit à la plus grande corrélation, c'est-à-dire celle pour laquelle la valeur de p_g est la plus éloignée possible de 1/2. On peut démontrer que la fonction g est optimale pour l'attaque si et seulement si elle vérifie

$$\begin{cases} g(x) = 1 & \text{si } Pr_Y[f(x,Y) = 1] > \dfrac{1}{2} \\ g(x) = 0 & \text{si } Pr_Y[f(x,Y) = 1] < \dfrac{1}{2} \end{cases}$$

Il s'ensuit que la probabilité p_g optimale mise en jeu dans la complexité de l'attaque est donnée par

$$\max_g p_g = \frac{1}{2} + \frac{1}{2^\ell} \sum_{x \in \mathbf{F}_2^\ell} \left| \frac{1}{2} - Pr_Y[f(x,Y) = 1] \right| \ .$$

On en déduit également le critère de sécurité garantissant qu'un générateur résiste à l'attaque par corrélation. En effet, il est impossible de mener l'attaque si la valeur correcte de x_0 ne peut pas être

distinguée des autres, c'est-à-dire si la probabilité p_g est égale à $1/2$ pour tous les choix possibles de la fonction g. Cela signifie exactement que la probabilité $Pr_Y[f(x,Y)=1]$ vaut $1/2$ pour toutes les valeurs de x, autrement dit que la fonction f reste équilibrée (i.e., uniformément distribuée) quand ses ℓ premières entrées sont fixées. Les fonctions ℓ-résilientes (dites aussi dans ce cas *sans corrélation d'ordre* ℓ) sont un exemple de fonctions vérifiant ce critère. Par exemple, pour se prémunir de l'attaque par corrélation décrite précédemment contre le générateur de Geffe, il aurait fallu choisir comme fonction de combinaison une fonction qui soit au moins 1-résiliente. Dans la fiche consacrée aux *générateurs pseudo-aléatoires à base de LFSR*, on trouvera plus de détails sur les critères de résistance aux attaques par corrélation dans ce cas particulier.

5.2.4 Les attaques algébriques

Principe. Les attaques algébriques sont des attaques à clair connu qui exploitent des relations algébriques entre les bits du clair, ceux du chiffré et ceux de la clef secrète. La connaissance de plusieurs couples clairs-chiffrés fournit donc un système d'équations dont les inconnues sont les bits de la clef secrète. Ces derniers peuvent alors être retrouvés en résolvant le système, ce qui est possible s'il est de degré faible, de petite taille ou qu'il possède une structure particulière.

Attaque algébrique élémentaire. Une attaque algébrique immédiate, décrite par Shannon [32, p.711], consiste simplement à écrire les équations de chiffrement, c'est-à-dire à exprimer les bits du chiffré en fonction des bits de clair et des bits de clef.

On peut alors résoudre le système obtenu par une méthode simple, appelée linéarisation ; il s'agit d'identifier chacun monôme dans lequel interviennent des bits de clef à une nouvelle variable. Pour un système de degré maximal d, on obtient ainsi un système linéaire en au plus

$$\sum_{i=1}^{d} \binom{n}{i} \text{ variables}$$

où n est le nombre de bits de la clef secrète. Ce gros système linéaire peut alors être inversé par une simple élimination de Gauss ou des techniques d'inversion un peu plus sophistiquées comme la méthode de Strassen, dont la complexité est donnée par le nombre d'équations du système, élevé à un exposant ω avec $\omega = 3$ dans le cas du pivot de Gauss, ou $\omega = 2,37$ [23]. Il existe d'autres techniques beaucoup plus efficaces pour résoudre de tels systèmes algébriques : il s'agit des algorithmes de calcul de bases de Grbner, largement étudiés en calcul formel. Les algorithmes de base de Grbner les plus utiles dans le contexte de la cryptanalyse sont décrits et étudiés dans [29, 20], notamment les algorithmes F4 et F5 de Jean-Charles Faugère.

Cette attaque algébrique simple peut s'avérer extrêmement performante par exemple contre les générateurs pseudo-aléatoires constitués d'un unique LFSR filtré par une fonction booléenne f de degré faible. Ainsi, si l'on note $z_0, \ldots, z_{n-1}$ les n bits de l'état initial du LFSR et L la fonction linéaire qui exprime l'état interne du registre en fonction de l'état précédent, la connaissance de N bits de la suite chiffrante $\boldsymbol{s}$ conduit au système d'équations

$$\begin{cases} s_0 &= f(z_0, \ldots, z_{n-1}) \\ s_1 &= f(L(z_0, \ldots, z_{n-1})) \\ s_2 &= f(L^2(z_0, \ldots, z_{n-1})) \\ &\vdots \end{cases}$$

en n inconnues (les bits de l'état initial), de degré égal au degré de la fonction de filtrage f. L'état initial peut donc être retrouvé en linéarisant ce système. On obtient alors de l'ordre de

$$\sum_{i=1}^{d} \binom{n}{i} \simeq n^d$$

variables. L'attaque nécessite donc la connaissance de n^d bits de suite chiffrante et de l'ordre de

$$n^{\omega n} \text{ opérations}$$

avec $\omega \simeq 2,37$. Si la taille de l'état initial correspond au double de la taille de la clef k (taille minimale pour résister aux attaques par compromis temps-mémoire), cette attaque algébrique est plus performante que la recherche exhaustive dès que le degré de f vérifie

$$\deg f \leq 0.42 \left[\frac{k}{1 + \log_2 k} \right] .$$

Une condition nécessaire de sécurité pour les tous les générateurs pseudo-aléatoires dont la fonction de transition est linéaire est donc que le degré de la fonction de filtrage doit être élevé.

Attaques algébriques évoluées sur les chiffrements à flot. En 2003, Courtois et Meier [26] ont proposé une amélioration de l'attaque précédente, qui peut parfois aboutir même lorsque le degré de la fonction de filtrage f est élevé. L'attaque fonctionne dès lors qu'il existe des relations de petit degré entre la sortie de la fonction et ses entrées [31]. Plus précisément, l'attaquant recherche des fonctions g et h de petit degré qui vérifient

- pour tout $(x_1, \ldots, x_n)$, $g(x_1, \ldots, x_n) f(x_1, \ldots, x_n) = 0$,
- ou pour tout $(x_1, \ldots, x_n)$, $h(x_1, \ldots, x_n)\,[1 + f(x_1, \ldots, x_n)] = 0$.

Si de telles fonctions g ou h de degré d existent, on peut engendrer un système d'équations de degré d de la manière suivante :

- si $s_t = 1$, on a $g(z_t, \ldots, z_{t+n-1}) = 0$ où $(z_t, \ldots, z_{t+n-1})$ est l'état du registre à l'instant t ;
- si $s_t = 0$, on a $h(z_t, \ldots, z_{t+n-1}) = 0$.

En exprimant l'état du registre à l'instant t comme une fonction linéaire de l'état initial, on obtient comme précédemment un système d'équations de degré d en n variables (les bits de l'état initial), que l'on peut résoudre par les techniques évoquées plus haut.

Pour se mettre à l'abri de ces attaques, il est donc essentiel que toutes les fonctions g et h qui possèdent la propriété décrite ci-dessus soient de degré élevé. Les fonctions g à n variables qui vérifient $g(x_1, \ldots, x_n) f(x_1, \ldots, x_n) = 0$ forment un idéal de l'anneau des fonctions booléennes à n variables, appelé *annulateur de la fonction* f, $\mathcal{AN}(f)$. Le paramètre essentiel, qui influence la complexité des attaques algébriques, est donc le degré minimal des fonctions (non nulles) de $\mathcal{AN}(f)$ et $\mathcal{AN}(1+f)$. Ce paramètre est appelé *immunité algébrique de* f, noté $AI(f)$:

$$AI(f) = \min \deg \{ g \in \mathcal{AN}(f) \cup \mathcal{AN}(1+f),\ g \neq 0 \} .$$

Ainsi, pour un générateur pseudo-aléatoire dont l'état interne est de taille $n = 2k$ où k est le nombre de bits de la clef secrète et dont la fonction de transition est linéaire (par exemple, un LFSR filtré), l'attaque précédente sera plus performante que la recherche exhaustive de la clef dès que l'immunité algébrique de la fonction de filtrage vérifie

$$AI(f) \geq 0.42 \left[\frac{k}{1 + \log_2 k} \right] .$$

Immunité algébrique des fonctions de filtrage. L'immunité algébrique de la fonction de filtrage, au même titre que son degré, est donc un paramètre important qui doit être pris en compte lors de la conception d'un générateur pseudo-aléatoire dont la fonction de transition est linéaire. On peut aisément démontrer [26] que l'immunité algébrique d'une fonction booléenne à n variables est toujours inférieure ou égale à

$$\left\lfloor \frac{n+1}{2} \right\rfloor .$$

A l'heure actuelle, les rares familles générales de fonctions connues dont l'immunité algébrique est maximale présentent d'autres inconvénients qui rendent leur utilisation directe peu souhaitable dans un chiffrement à flot.

En revanche, le fait que les fonctions d'immunité algébrique élevée sont nécessairement éloignées des fonctions de degré 1 (au sens où elles ont une non-linéarité élevée) est une propriété intéressante. Ainsi, les générateurs à base de LFSRs qui résistent aux attaques algébriques ne sont généralement pas vulnérables aux attaques par corrélation rapides.

Une caractérisation plus poussée des fonctions d'immunité algébrique élevée, l'établissement d'éventuelles relations entre ce paramètre et d'autres quantités intervenant dans la résistance à d'autres types d'attaques, comme l'ordre de non-corrélation, font actuellement l'objet de recherches plus poussées.

Attaques algébriques rapides. Il existe une variante plus générale et souvent plus efficace des attaques algébriques contre les chiffrements à flot, introduite par Courtois [24] sous le nom d'attaque algébrique rapide. Une première amélioration de l'attaque précédente, permettant de diminuer le degré du système d'équations à résoudre, consiste à rechercher des équations de petit degré en les bits de l'état initial, qui font intervenir plusieurs bits consécutifs de suite chiffrante. La recherche de ces équations est souvent un problème complexe, dont la complexité augmente considérablement quand le nombre de bits de suite chiffrante intervenant simultanément dans les équations augmente. On peut alors être amené à ne considérer que des relations ayant une forme particulière, obtenues en additionnant plusieurs relations connues de manière à faire baisser le degré [30]. Un exemple d'attaque algébrique rapide est celle qui permet de cryptanalyser le générateur par LFSR filtré Sfinks, candidat à l'appel eSTREAM [25].

Des variantes plus sophistiquées des attaques algébriques s'appliquent également lorsque la fonction de filtrage fait intervenir quelques bits de mémoire mis à jour de manière non linéaire, comme dans le système E0 [19, 18], ou lorsqu'il s'agit non pas d'une fonction booléenne, mais d'une fonction qui produit plusieurs bits de sortie à chaque instant [27].

Attaques algébriques sur les chiffrements par blocs. Le principe de base des attaques algébriques décrit par Shannon s'applique évidemment aussi aux algorithmes de chiffrement par blocs dès lors qu'il est possible d'exprimer le chiffrement par des équations de degré faible en les bits de la clef (ou des sous-clefs). Ainsi, Courtois et Pieprzyk [28] ont suggéré d'appliquer cette technique pour cryptanalyser l'AES, en exploitant le fait que la fonction d'inversion dans le corps fini à 256 éléments employée dans l'AES conduit à des relations de degré 2 entre les bits de l'entrée et ceux de la sortie de chaque tour de l'algorithme. Toutefois, le système résultant de ces équations, même s'il n'est que de degré 2, est de très grande taille car il fait intervenir les bits de toutes les sous-clefs de l'AES. A l'heure actuelle, les méthodes connues de résolution de systèmes algébriques ne semblent donc pas permettre de le résoudre plus efficacement que la recherche exhaustive de la clef. Les attaques algébriques contre les chiffrement par blocs semblent donc encore hors de portée, même si elles ont été mises en œuvre récemment sur des systèmes de petite taille [22, 20].

Bibliographie

[1] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, 1983.

[2] R.A. Rueppel. *Analysis and Design of stream ciphers*. Springer-Verlag, 1986.

[3] A. Canteaut and M. Trabbia. " Improved fast correlation attacks using parity-check equations of weight 4 and 5 ". In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.

[4] G. Gong. " Sequence analysis ". Lecture Notes for CO739x, `http://www.comsec.uwaterloo.ca/ ggong/CO739x/seq-main.ps`, 1999.

[5] H. Molland. " Improved Linear Consistency Attack on Irregular Clocked Keystream Generators ". In *Fast Software Encryption - FSE'04*, volume 3017 of *Lecture Notes in Computer Science*, pages 109–126. Springer-Verlag, 2004.

[6] H. Molland and T. Helleseth. " An improved correlation attack against irregular clocked and filtered generator ". In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 373–389. Springer-Verlag, 2004.

[7] E.R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968.

[8] J.L. Massey. " Shift-register synthesis and BCH decoding ". *IEEE Transactions on Information Theory*, 15 :122–127, janvier 1969.

[9] P.R. Geffe. " How to protect data with ciphers that are really hard to break ". *Electronics*, pages 99–101, 1973.

[10] R. Göttfert and H. Niederreiter. " On the minimal polynomial of the product of linear recurring sequences ". *Finite Fields and Their Applications*, 1(2) :204–218, 1995.

[11] T. Herlestam. " On functions of linear shift register sequences ". In F. Pichler, editor, *Advances in Cryptology - EUROCRYPT '85*, volume 219 of *Lecture Notes in Computer Science*, pages 119–129. Springer-Verlag, 1986.

[12] E.L. Key. " An analysis of the structure and complexity of nonlinear binary sequence generators ". *IEEE Transactions on Information Theory*, 22 :732–736, 1976.

[13] J.L. Massey. " The ubiquity of Reed-Muller Codes ". In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes - AAECC-14*, volume 2227 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2001.

[14] R.A. Rueppel and O.J. Staffelbach. " Products of Linear Recurring Sequences with Maximum Complexity ". *IEEE Transactions on Information Theory*, 33(1) :124–131, 1987.

[15] A. Canteaut. " Fast Correlation Attacks Against Stream Ciphers and Related Open Problems ". In *Proceedings of the 2005 IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security*, pages 49–54. IEEE, 2005.

[16] P. Chose, A. Joux, and M. Mitton. " Fast correlation attacks : an algorithmic point of view ". In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.

[17] T. Siegenthaler. " Decrypting a class of stream ciphers using ciphertext only ". *IEEE Trans. Computers*, C-34(1) :81–84, 1985.

[18] F. Armknecht. " A linearization attack on the Bluetooth keystream generator ". `http://th.informatik.uni-mannheim.de/people/armknecht/E0.ps`, 2002.

[19] F. Armknecht and M. Krause. " Algebraic attacks on combiners with memory ". In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 162–176. Springer-Verlag, 2003.

[20] G. Ars. " *Applications des bases de Grbner à la cryptographie* ". PhD thesis, Université de Rennes 1, 2005.
`http://name.math.univ-rennes1.fr/gwenole.ars/`.

[21] A. Canteaut. " Open problems related to algebraic attacks on stream ciphers ". In *Workshop on Coding and Cryptography - WCC 2005*, Lecture Notes in Computer Science. Springer-Verlag, 2006.

[22] C. Cid, S. Murphy, and M. Robshaw. " Small Scale Variants of the AES ". In *Fast Software Encryption - FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Springer Verlag, 2005.

[23] D. Coppersmith and S. Winograd. " Matrix multiplication via arithmetic programming ". *Journal of Symbolic Computation*, (9) :251–280, 1990.

[24] N. Courtois. " Fast algebraic attacks on stream ciphers with linear feedback ". In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 177–194. Springer-Verlag, 2003.

[25] N. Courtois. " Cryptanalysis of Sfinks ". IACR Eprint report 2005/243, 2005.
`http://eprint.iacr.org/2005/243`.

[26] N. Courtois and W. Meier. " Algebraic attacks on stream ciphers with linear feedback ". In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.

[27] N.T. Courtois. " Algebraic Attacks on Combiners with Memory and Several Outputs ". In *ICISC 2004*, Lecture Notes in Computer Science. Springer-Verlag, 2005.
`http://eprint.iacr.org/2003/125/`.

[28] N.T. Courtois and J. Pieprzyk. " Cryptanalysis of Block Ciphers with Overdefined Systems of Equations ". In *Advances in Cryptology - ASIACRYPT 2002*, volume 2502 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.

[29] A. Canteaut et al.. " Open Research Areas in Symmetric Cryptography and Technical Trends in Lightweight Cryptography ". ECRYPT report, 2005.
`www.ecrypt.eu.org/documents/D.STVL.3-2.5.pdf`.

[30] P. Hawkes and G. G. Rose. " Rewriting variables : the complexity of fast algebraic attacks on stream ciphers ". In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 390–406. Springer-Verlag, 2004.

[31] W. Meier, E. Pasalic, and C. Carlet. " Algebraic attacks and decomposition of Boolean functions ". In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer-Verlag, 2004.

[32] C.E. Shannon. " Communication theory of secrecy systems ". *Bell system technical journal*, 28 :656–715, 1949.

Chapitre 6

Fonctions booléennes et fonctions vectorielles

6.1 Fonctions booléennes

6.1.1 Algèbre de Boole

On considère un ensemble $\mathcal{A}$ munis de deux lois binaires + et . et d'une loi unaire .

Définition 8 *$\mathcal{A}(+,.,^-)$ est une algèbre de Boole si elle vérifie les axiomes suivants :*

1. *Il existe un élément neutre 0 pour l'addition et un élément neutre 1 pour la multiplication. De plus, $x.0 = 0$, $x + 1 = 1$.*
2. *Idempotence : $x^2 = x$, $x + x = x$.*
3. *Commutativité : $xy = yx$, $x + y = y + x$.*
4. *Associativité : $x(uz) = (xy)z$, $x + (y + z) = (x + y) + z$.*
5. *Absorption : $x(x + y) = x$, $x + xy = x$.*
6. *Distributivité : $x(y + z) = xy + xz$. En particulier, $(x + y)(x + z) = x + yz$.*
7. *$x\overline{x} = 0$, $\overline{\overline{x}} = x$*
8. *Lois de Morgan (déductibles des autres) : $\overline{xy} = \overline{x} + \overline{y}$, $\overline{x + y} = \overline{x}\,\overline{y}$*

Exemple 1 • *L'ensemble des parties d'un ensemble, munis de $\cap$ (addition), $\cup$ (multiplication) et du complémentaire.*
• *L'ensemble $\{V, F\}$ muni des relations "ou" $\vee$, "et" $\wedge$, "non" $\neg$.*

6.1.2 Structure d'anneau associé à une algèbre de Boole

Une algèbre de Boole n'est pas une algèbre, puisque certains éléments n'ont pas d'inverses pour l'addition.

On peut cependant contourner le problème : dans le cas du calcul booléen, on peut remplacer le "ou" par le "ou exclusif". Dans le cas des parties d'un ensemble, on peut remplacer l'union par la différence symétrique Δ.

Plus généralement,

Proposition 1 *Soit $\mathcal{A}(+,.,^-)$ une algèbre de Boole. On définit sur $\mathcal{A}$ la loi binaire suivante : $x \oplus y = x\overline{y} + \overline{x}y$. Alors $\mathcal{A}(\oplus,.)$ est un anneau idempotent (c'est-à-dire $x^2 = x$ pour tout x).*

Proposition 2 *Soit $\mathcal{A}(\oplus, .)$ un anneau idempotent. On peut munir $\mathcal{A}$ d'une structure d'algèbre de Boole de la manière suivante :*
$x + y = x \oplus y \oplus xy$ et $\overline{x} = 1 \oplus x$.

Remarque 1 *A partir d'une algèbre de Boole, si on construit l'anneau associé en utilisant la proposition 1, puis l'algèbre de Boole associée en utilisant la proposition 2, on retrouve bien la structure de départ.*

6.1.3 Relation d'ordre dans une algèbre de Boole

Définition 9 *Soit $\mathcal{A}$ une algèbre de Boole. On définit sur $\mathcal{A}$ une relation d'ordre partielle par une des trois propriétés suivantes :*

$$y \leq x \quad \Leftrightarrow \quad xy = y \tag{6.1}$$

$$y \leq x \quad \Leftrightarrow \quad \overline{x}y = y \tag{6.2}$$

$$y \leq x \quad \Leftrightarrow \quad x + y = x \tag{6.3}$$

Démonstration : Si $xy = y$, alors $\overline{x}(xy) = \overline{x}y$, c'est-à-dire $0 = \overline{x}y$.

Réciproquement, si $y\overline{x} = 0$. On a $y.1 = y(x + \overline{x} = y$, donc $yx = y$.

De même, si $x + y = x$, alors $\overline{x}(x + y) = \overline{x}x$, c'est-à-dire $\overline{x}y = 0$. Réciproquement, supposons $\overline{x}y = 0$. On a $x + y = 1(x + y) = (\overline{x} + x)(x + y) = x + xy = x$.

On vérifie qu'il s'agit bien d'une relation d'ordre.

□

Remarque 2 *Dans le cas des parties d'un ensemble, il s'agit de l'inclusion $\subseteq$.*

Propriétés

- Pour tout $x \in \mathcal{A}$, on a $0 \leq x \leq 1$.
- Si $y < x$, alors $\overline{y}x \neq 0$.
  Si $y < x$, alors $y \not\geq x$, donc $\overline{y}x \neq 0$.
- Si $x \leq z$ et $y \leq z$, alors $x + y \leq z$.
- Si $x < y$, alors $\overline{y} < \overline{x}$.

Définition 10 *Un atome x de $\mathcal{A}$ est un élément minimal non nul pour la relation d'ordre, i.e. $x \neq 0$ et il n'existe pas de y tel que $0 < y < x$.*
Un élément x est maximal si $x \neq 1$ et s'il n'existe pas de y tel que $x < y < 1$.

Remarque 3 *Dire que x est un atome est équivalent à dire que $\overline{x}$ est maximal.*

Exemple 2 *Dans $\mathcal{P}(E)$, les atomes sont les singletons.*

Proposition 3 *Le produit de 2 atomes distincts est nul. De plus, si z est un atome, alors pour tout $x \in \mathcal{A}$, $z \leq x$ ou $z \leq \overline{x}$.*

Démonstration : Soient z et z' deux atomes. On a $z + zz' = z$, donc $zz' \leq z$. De même, $zz' \leq z'$. Si $zz' \neq 0$, on a $zz' = z = z'$ puisque z et z' sont des atomes.

Soit z un atome et $x \in \mathcal{A}$. On a $xz \leq z$. Si $xz = 0$, cela signifie $z \leq \overline{x}$. Si $xz = z$, on a bien $z \leq x$. □

Proposition 4 *Soit $\mathcal{A}$ une algèbre de Boole finie. Soit x un élément non nul de $\mathcal{A}$. Il existe un atome z tel que $z \leq x$.*

Démonstration :

Si x est un atome, c'est terminé. Si x n'est pas un atome, il existe x_1 tel que $0 < x_1 < x$. On réitère le procédé sur x_1. On construit ainsi une suite x, x_1, x_2 d'éléments distincts non nuls de $\mathcal{A}$. Puisque $\mathcal{A}$ est fini, le procédé s'arrête, c'est-à-dire que l'on obtient un atome x_n qui vérifie bien $x_n \leq x$.

□

Théorème 3 *Tout élément x d'une algèbre de Boole finie est la somme des atomes qu'il contient.*

Démonstration : Soit $y = \sum_{i=1}^{n} x_i$, la somme des atomes contenus dans x. Puisque $x_i \leq x$ pour tout x, on a $y \leq x$.

Supposons $y \leq x$, alors $\overline{y}x \neq 0$. Soit z un atome contenu dans $\overline{y}x$. On a alors $z \leq x$, il s'agit donc d'un des x_i. Ceci entraîne $z \leq y$, mais $z \leq \overline{y}$, donc $z = 0$, ce qui est impossible, puisqu'il s'agit d'un atome. Donc $y = x$. □

Théorème 4 *Tout élément x de $\mathcal{A}$ est le produit des éléments maximaux qui le contiennent.*

Démonstration : C'est la conséquence du théorème précédent et de la loi de Morgan. □

Théorème 5 *Deux sommes d'atomes ne sont égales que si l'on somme sur les mêmes atomes.*

Démonstration : Si $\sum_{i=1}^{l} x_i = \sum_{j=1}^{k} y_j$, alors $x_1 \sum_{i=1}^{l} x_i = x_1 = x_1 \sum_{j=1}^{k} y_j$. Il existe alors un indice j tel que $y_j = x_1$. On réitère le procédé pour les autres atomes. □

Théorème 6 *Toute algèbre de Boole finie $\mathcal{A}$ est isomorphe à une algèbre de Boole des parties d'un ensemble.*

Démonstration : Soient $a_1, \ldots, a_n$ les atomes de $\mathcal{A}$. Soit $E = \{a_1, \ldots, a_n\}$. On définit l'application f de $\mathcal{A}$ dans $\mathcal{P}(E)$ par :
Si $x = \sum_{i \in J} a_i$ la décomposition de x en atomes, alors $f(x) = \bigcup_{i \in J} \{a_i\}$.
f est une bijection : surjection de manière évidente, injection : il s'agit du théorème 5.
f est un morphisme d'algèbre de Boole : addition facile, complémentaire aussi.
Pour la multiplication, il faut utiliser les 2 précédentes et la relation

$$xy = \overline{\overline{x} + \overline{y}}$$

. □

Corollaire 1 *Si n est le nombre d'atomes d'une algèbre de Boole finie $\mathcal{A}$, son cardinal est alors 2^n.*

6.1.4 Fonctions Booléennes

Définition 11 *L'ensemble des fonctions booléennes à n variables est l'ensemble des fonctions de $\{0,1\}^n$ à valeurs dans $\{0,1\}$:*

$$\mathcal{B}_n = \{f : \to \{0,1\}\}$$

- Si l'on identifie $\{0,1\}$ au corps à 2 éléments $\mathbb{F}_2$, on muni $\mathcal{B}_n$ d'une structure d'anneau idempotent.
- Si l'on muni $\{0,1\}$ de la structure d'algèbre de Boole, on muni $\mathcal{B}_n$ d'une structure d'algèbre de Boole
  On considère d'abord la deuxième structure.

$X_i : (x_1, \ldots, x_n) \to x_i$
$\overline{X_i} : (x_1, \ldots, x_n) \to \overline{x_i}$
Soit $x \in \{0,1\}^n$, alors $\delta_x = \tilde{X}_1 \ldots \tilde{X}_n$ avec $\tilde{x}_i = X_i$ si $x_i = 1$, $\tilde{x}_i = \overline{X_i}$ sinon.
L'ensemble des δ_x s'appellent des monômes booléens.
Faire des exemples pour $n = 3$.

Proposition 5 *Soit $f \in \mathcal{B}_n$, alors $f = \sum_{f(x)=1} \delta_x$.*

On obtient alors une représentation sous forme de polynômes des fonctions booléennes.
Dans $\mathcal{B}_n$ la relation d'ordre des algèbres de Boole devient :

$$f \leq g \Leftrightarrow (f(x) = 1 \Rightarrow g(x) = 1)$$

En exercice : faire la représentation des fonctions booléennes sous la forme

$$\mathbb{F}_2[X_1, \ldots, X_n]/(X_1^2 + X_1) \ldots (X_n^2 + X_n)$$

6.1.5 Forme algébrique normale et transformée de Möbius

Dans la dfinition d'une fonction booléenne, il apparaît naturellement une deuxième fonction booléenne (resp. booléenne vectorielle) sur F_2^m : la fonction $u \to a_u$. Cette fonction s'appelle la *transformée de Möbius* de f.
On a, pour toute fonction booléenne sur F_2^m :

$$f(x) = \sum_{u \in F_2^m} g(u)\, x^u$$

où g est la transformée de Möbius de f.

Proposition 6 *On définit sur F_2^m l'ordre partiel suivant :*

$$(v_1, \ldots, v_m) \preceq (u_1, \ldots, u_m) \Longleftrightarrow \forall i = 1, \ldots, m,\ v_i \leq u_i.$$

Alors, la transformée de Möbius d'une fonction booléenne (resp. booléenne vectorielle) quelconque f est la fonction g définie par :

$$g(u) = \sum_{v \in F_2^m \,|\, v \preceq u} f(v),$$

la somme étant calculée dans F_2 (resp. $GF(2^r)$).

Démonstration : Calculons d'abord l'expression de f en fonction de g : pour tout mot u et tout mot v, le produit $v^u = \prod_{i=1}^{m} {v_i}^{u_i}$ est égal à 1 si et seulement si chacun de ses facteurs vaut 1, c'est à dire si $u \preceq v$.
On a donc :

$$f(v) = \sum_{u \preceq v} g(u).$$

Il nous reste à montrer que l'application qui à f fait correspondre la fonction $v \to \sum_{u \in F_2^m \,|\, u \preceq v} f(u)$ est involutive.
Or, si on l'applique deux fois à f, on obtient la fonction :

$$v \to \sum_{w \preceq u} \sum_{u \preceq v} f(w) = \sum_{w \preceq v} f(w)\, |\{u \in F_2^m |\, w \preceq u \preceq v\}|$$

qui est bien la fonction f puisque l'ensemble $\{u \in F_2^m |\, w \preceq u \preceq v\}$ est de cardinal impair si et seulement si $w = v$. □

6.2 Critères cryptographiques sur les fonctions booléennes

Il y avait jusqu'à maintenant quatre critères importants, et un cinquième critère est apparu très récemment.

Équilibre

Une fonction booléenne utilisée comme fonction de combinaison doit être *équilibrée* :

Définition 12 *Une fonction Booléenne f à m variables est équilibrée si sa sortie est uniformément distribuée, i.e.*

$$\#\{x \in \mathbb{F}_2^m, f(x) = 0\} = \#\{x \in \mathbb{F}_2^m, f(x) = 1\} = 2^{m-1}.$$

Proposition 7 *Soit f une fonction booléenne équilibrée, alors $deg(f) \geq m - 1$.*

Démonstration : Le coefficient de degré m est $g(1, ..., 1)$, où g est la transformée de Möbius de f. On $g(1, ..., 1) = \sum_{v \in \mathbb{F}_2^m} f(v) \pmod 2 = \#\{x \in \mathbb{F}_2^m, f(x) = 1\} \pmod 2 = 2^{m-1} \pmod 2 = 0.$ □

Degré algébrique

Puisque la suite générée par le système est périodique, elle est attaquable par l'algorithme de Berlekamp-Massey. Il faut donc qu'elle soit de complexité linéaire élevée. On démontre que la suite générée par une fonction $f(x_1, \cdots, x_m) = \sum_{u \in F_2^m} a_u \left(\prod_{i=1}^{m} {x_i}^{u_i} \right) = \sum_{u \in F_2^m} a_u \, x^u$ prenant en entrées des suites ML générées par des LFSR de longueurs $L_1, \cdots, L_m$ est de complexité linéaire $f(L_1, \cdots, L_m)$, c'est à dire $\sum_{u \in F_2^m} a_u \left(\prod_{i=1}^{m} {L_i}^{u_i} \right)$, où le calcul de la somme est effectué dans $\mathbf{R}$ et non modulo 2. On voit qu'il faut utiliser une fonction de *haut degré algébrique* pour atteindre de hautes complexités linéaires. Dans le cas du registre filtré, si L est la longueur du LFSR et si son polynôme de rétroaction est primitif, alors la complexité linéaire de la suite vérifie :

$$\mathcal{L} \leq \sum_{i=0}^{d^\circ f} \binom{L}{i}.$$

Si L est premier alors :

$$\mathcal{L} \geq \binom{L}{d^\circ f}.$$

Non-corrélation et résilience

Le modèle avec fonction de combinaison peut être attaqué par l'attaque de Siegenthaler ou par l'une de ses améliorations qui ont suivi. Rappelons le principe de l'attaque par corrélation de Siegenthaler : supposons qu'il y ait une corrélation entre la suite pseudo-aléatoire $(s_i)_{i \geq 0}$ générée par le générateur et la sortie $(x_j^i)_{i \geq 0}$ de l'un des LFSR (le $j$ème) ; la probabilité p_j que $s_i = x_j^i$ est alors différente de $1/2$. En d'autres termes, la probabilité que $s_i = 0$ sachant que $x_j^i = 0$ est différente de $1/2$. De façon équivalente, la probabilité que $f(x) = 0$ sachant que $x_j = 0$ est différente de $1/2$. On peut réaliser une attaque exhaustive sur ce LFSR : on essaie toutes les initialisations possibles jusqu'à ce qu'on observe la corrélation attendue ; si l'initialisation est incorrecte, alors la corrélation entre la suite de sortie du LFSR et la suite s_i est celle qu'on observe entre deux suites indépendantes : la probabilité que $s_i = x_j^i$ est égale à $1/2$; sinon, elle est égale à p_j dans le cas d'une attaque à clair connu (dans laquelle on suppose connue une sous-suite du chiffré et la sous-suite correspondante du clair) et elle vaut une probabilité calculable et différente de $1/2$, dans le cas d'une attaque à chiffré seul, en supposant que la probabilité d'apparition du 0 dans le clair est elle-même différente de $1/2$. Une fois l'initialisation du $j$ème LFSR obtenue, on attaque le reste du schéma par une attaque exhaustive, ce qui remplace la complexité de l'attaque exhaustive globale du système, égale à $O(2^{\sum_{i=1}^{m} L_i})$, par $O(2^{L_j} + 2^{\sum_{1 \leq i \leq m / \, i \neq j} L_i})$ (d'où un gain important). Une fonction permet une résistance optimale à cette attaque par corrélation si, pour tout j,

la fonction $f(x)$ étant équilibrée, elle reste équilibrée si l'on fixe la valeur de x_j. Nous verrons que c'est équivalent au fait que l'on a $\sum_{x \in F_2^m} (-1)^{f(x)+x_j} = 0$ pour tout j.

Le même principe d'attaque peut se faire en testant plusieurs LFSR (disons au plus k) au lieu d'un seul. Une fonction permet une résistance optimale à l'attaque consistant à exploiter une corrélation entre la sortie de f et au plus k sorties des LFSR si la fonction $f(x)$ reste équilibrée si l'on fixe les valeurs d'au plus k variables $x_{j_1}, \cdots, x_{j_k}$.

Définition 13 *Une fonction est sans corrélation d'ordre k si, quel que soit le sous ensemble I de cardinal k de $\{1, \ldots, n\}$ et quel que soit le vecteur $a \in F_2^I$, la restriction de f obtenue en fixant les valeurs x_i, $i \in I$ à a_i a la même distribution de valeurs que f elle-même (i.e. la probabilité qu'elle sorte un 0 est la même). La fonction est dite k-résiliente si elle est de plus équilibrée.*

Une caractérisation fait intervenir une transformation qui joue un rôle très important dans l'étude des fonctions booléennes :

La transformée de Walsh est l'application qui à tout vecteur $a \in F_2^m$ fait correspondre $\widehat{\mathrm{f}}(a) = \sum_{x \in F_2^m} (-1)^{f(x)+a \cdot x}$. La transformée de Walsh est un cas particulier d'une transformation générale appelée la transformée de Fourier.

Définition 14 *Soit φ une fonction numérique (i.e. à valeurs dans $\mathbf{Z}$ ou $\mathbf{R}$ ou $\mathbf{C}$) sur F_2^m. La transformée de Fourier de φ est la fonction*

$$\widehat{\varphi}(s) = \sum_{x \in F_2^m} \varphi(x) \, (-1)^{s \cdot x}$$

où "$\cdot$" désigne le produit scalaire *usuel sur F_2^m, défini par :*

$$s \cdot x = \sum_{i=1}^{m} s_i x_i \, mod 2.$$

Algorithme :
- Écrire la table de valeurs de la fonction en mettant les mots binaires de longueur m dans l'ordre naturel
- Soit ψ la fonction sur F_2^{m-1} correspondant a la partie supérieure de la table et ψ' celle correspondant a la partie inférieure. Remplacer ψ par $\psi + \psi'$ et ψ' par $\psi - \psi'$ (sommes calculées dans $\mathbf{Z}$).
- Recommencer récursivement sur chacune des moitiés ainsi obtenues.

Lorsque l'algorithme termine (c'est a dire lorsque on en est arrivé à des fonctions sur F_2^1), les nombres obtenus en face de chaque mot sont les valeurs de la transformée de Fourier.
La complexité est en $\mathrm{O}(m2^m)$.

Remarque 4 *Il existe une définition légèrement différente, dans laquelle on divise la somme $\sum_{x \in F_2^m} \varphi(x) \, (-1)^{s \cdot x}$ par $2^{m/2}$. Nous l'appellerons (pour des raisons qui apparaîtront plus loin) la transformation de Fourier normalisée.*

Rappelons une propriété que nous avons déjà rencontrée plusieurs fois sous des formes différentes :

Proposition 8 *Soit E un sous-espace vectoriel de F_2^m. On note 1_E son indicatrice. Alors, on a :*

$$\widehat{1_E} = |E| \, 1_{E^\perp}.$$

La preuve est laissée au lecteur.
On utilisera souvent cette propriété avec $E = F_2^m$:

$$\widehat{1} = 2^m \, \delta_0$$

où δ_0 est le symbole de Dirac ($\delta_0(x) = 1$ si $x = 0$ et 0 sinon).
On retrouve le résultat utilisé à l'occasion de l'identité de Mac Williams :

Corollaire 2 *[Formule de sommation de Poisson] Pour tout sous-espace vectoriel E de F_2^m, on a :*

$$\sum_{s\in E}\widehat{\varphi}(s) = |E| \sum_{x\in E^\perp}\varphi(x).$$

Démonstration :

$$\sum_{s\in E}\widehat{\varphi}(s) = \sum_{x\in F_2^m}\varphi(x)\,\widehat{1_E}(x) = |E| \sum_{x\in E^\perp}\varphi(x).$$

□

La transformée de Fourier a en outre deux propriétés très utiles qui en font un outil beaucoup utilisé en informatique et en mathématiques discrètes :

Proposition 9 *Pour toute fonction φ :*

$$\widehat{\widehat{\varphi}} = 2^m\,\varphi.$$

La transformation de Fourier normalisée est donc involutive.

Démonstration :

$$\widehat{\widehat{\varphi}}(x) = \sum_{s\in F_2^m}\widehat{\varphi}(s)\,(-1)^{x\cdot s} = \sum_{s\in F_2^m}\sum_{y\in F_2^m}\varphi(y)\,(-1)^{x\cdot s+s\cdot y} =$$

$$\sum_{y\in F_2^m}\left(\sum_{s\in F_2^m}(-1)^{(x+y)\cdot s}\right)\varphi(y) = \sum_{y\in F_2^m}\widehat{1}(x+y)\,\varphi(y) = 2^m\,\varphi(x).$$

□

On en déduit que la transformée de Fourier est une permutation de l'ensemble des fonctions sur F_2^m à valeurs dans $\mathbf{R}$ ou dans $\mathbf{C}$ (mais pas dans $\mathbf{Z}$, et on ne sait pas caractériser efficacement les transformées de Fourier des fonctions sur F_2^m à valeurs dans $\mathbf{Z}$).

Exercice 1 *Déduire de la proposition précédente qu'une fonction numérique φ est constante si et seulement si sa transformée de Fourier est nulle en tout mot non nul, et qu'elle est constante sauf en 0 si et seulement si sa transformée de Fourier est elle-même constante sauf en 0.*

Avant d'énoncer la deuxième propriété, rappelons ce qu'est le produit de convolution de deux fonctions numériques sur F_2^m :

$$(\varphi\otimes\psi)(x) = \sum_{y\in F_2^m}\varphi(y)\psi(x+y).$$

Cette définition peut se donner plus généralement dans le cadre des fonctions numériques définies sur un groupe abélien (elle s'écrit avec $x-y$ au lieu de $x+y$, mais ici, nous sommes en caractéristique 2).

Proposition 10 *Pour toute fonction φ et toute fonction ψ, on a :*

$$\widehat{\varphi\otimes\psi} = \widehat{\varphi}\,\widehat{\psi}$$

et

$$\widehat{\varphi}\otimes\widehat{\psi} = 2^m\,\widehat{\varphi\,\psi}.$$

Démonstration :

$$\widehat{\varphi \otimes \psi}(s) = \sum_{x \in F_2^m} (\varphi \otimes \psi)(x)\,(-1)^{s\cdot x} = \sum_{x \in F_2^m} \sum_{y \in F_2^m} \varphi(y)\psi(x+y)\,(-1)^{s\cdot x} =$$

$$\sum_{x \in F_2^m} \sum_{y \in F_2^m} \varphi(y)\psi(x+y)\,(-1)^{s\cdot y + s\cdot(x+y)} = \sum_{y \in F_2^m} \varphi(y)(-1)^{s\cdot y} \left(\sum_{x \in F_2^m} \psi(x+y)\,(-1)^{s\cdot(x+y)} \right) =$$

$$\left(\sum_{y \in F_2^m} \varphi(y)(-1)^{s\cdot y} \right) \left(\sum_{x \in F_2^m} \psi(x)\,(-1)^{s\cdot x} \right) = \widehat{\varphi}(s)\,\widehat{\psi}(s).$$

La première égalité est donc vérifiée. La deuxième s'obtient en appliquant la première aux fonctions $\widehat{\varphi}$ et $\widehat{\psi}$ et en utilisant la proposition 9. □

La deuxième égalité appliquée en le mot 0 et pour $\psi = \varphi$ donne la relation dite de *Parseval* :

Corollaire 3 *Pour toute fonction numérique φ :*

$$\sum_{s \in F_2^m} (\widehat{\varphi}(s))^2 = 2^m \sum_{s \in F_2^m} (\varphi(s))^2.$$

Si de plus, φ est à valeurs ± 1, alors :

$$\sum_{s \in F_2^m} (\widehat{\varphi}(s))^2 = 2^{2m}.$$

Propriétés des fonctions sans corrélation et résilientes

Proposition 11 *Une fonction est sans corrélation d'ordre k si et seulement si on a $\widehat{\mathrm{f}}(s) = 0$ pour tout vecteur non nul de poids compris entre 1 et k. Elle est k-résiliente si et seulement si on a $\widehat{\mathrm{f}}(s) = 0$ pour tout vecteur de poids au plus k.*

C'est une conséquence directe de la formule de sommation de Poisson appliquée à la fonction $f(x+u)$ où le vecteur $u \in F_2^m$ coïncide avec a pour les indices appartenant à I (voir la définition 13), et à l'espace vectoriel $E = \{s \in F_2^m / \, s_i = 0, \forall i \notin I\}$. On notera que la fonction $f(x+u)$ a le même ordre de non corrélation que f, car, pour tout vecteur a, on a $\sum_{x \in F_2^m} (-1)^{f(x+u)+a\cdot x} = (-1)^{a\cdot u} \sum_{x \in F_2^m} (-1)^{f(x)+a\cdot x}$. ◇

Exercice 2 *Soit f une fonction k-résiliente qui n'est pas $(k+1)$-résiliente. On suppose sans perte de généralité (quitte à permuter les coordonnées) que $\widehat{\mathrm{f}}(1, \cdots, 1, 0, \cdots, 0) \neq 0$ où $(1, \cdots, 1, 0, \cdots, 0)$ est de poids $k+1$. Montrer que l'une des fonctions g ne dépendant que des variables $x_1, \cdots, x_{k+1}$ qui sont les plus proches de f est alors la fonction $x_1 + \cdots + x_{k+1}$ ou la fonction $x_1 + \cdots + x_{k+1} + 1$. On pourra exprimer la valeur de $g(a_1, \cdots, a_{k+1})$ en fonction des poids des restrictions de f obtenues en fixant les $k+1$ coordonnées de x et en déduire que pour au moins l'une des fonctions g, pour tout $j \leq k+1$, la valeur de g change nécessairement quand on change la valeur de n'importe lequel de ses bits d'entrée.*

On a la borne suivante sur le degré de f :

Proposition 12 *Si f est sans corrélation d'ordre k, alors elle est de degré au plus $m-k$.*
Si f est résiliente d'ordre k, alors :
- si $k \leq m-2$, son degré est au plus $m-k-1$,
- sinon $k = m-1$ et son degré est 1.

Démonstration : La preuve est une conséquence directe des propositions 6 et 11. □

Remarque 5 *Si f est sans corrélation d'ordre k, alors son degré est majoré par $m-k-1$ sous la condition plus faible que le poids de f soit divisible par 2^{k+1}.*

La notion de fonction sans corrélation est liée à celle de tableau orthogonal et il existe plusieurs constructions, directes ou récursives, de fonctions sans corrélations.
Citons une construction directe [1], dite de *Maiorana-McFarland* :

Proposition 13 *Soit r un entier compris entre 1 et $m-1$, g une fonction booléenne sur F_2^{m-r} et ϕ une application de F_2^{m-r} dans F_2^r. Soit f la fonction définie sur F_2^m par :*

$$\forall x \in F_2^r, \forall y \in F_2^{m-r}, f(x \,|\, y) = x \cdot \phi(y) + g(y)$$

(où "$\cdot$" désigne ici le produit scalaire dans F_2^r et $|$ la concaténation).
Alors f est résiliente d'ordre $k \geq inf\{\omega(\phi(y))|\, y \in F_2^{m-r}\} - 1$, où $\omega(\phi(y))$ désigne le poids de Hamming du mot $\phi(y)$.

Démonstration : On a, pour tout mot s de F_2^r et tout mot t de F_2^{m-r} :

$$\widehat{\mathrm{f}}(s \,|\, t) = \sum_{x \in F_2^r} \sum_{y \in F_2^{m-r}} (-1)^{x\cdot\phi(y)+g(y)+s\cdot x+t\cdot y} = \sum_{y \in F_2^{m-r}} (-1)^{g(y)+t\cdot y} \left(\sum_{x \in F_2^r} (-1)^{x\cdot(\phi(y)+s)} \right)$$

(on note de la même façon les produits scalaires dans F_2^r et dans F_2^{m-r}).
D'après la proposition 8, la somme $\sum_{x \in F_2^r} (-1)^{x\cdot(\phi(y)+s)}$ est nulle si et seulement si $s \neq \phi(y)$.
Si s est de poids strictement inférieur au plus petit poids des éléments $\phi(y), y \in F_2^{m-r}$, alors $\widehat{\mathrm{f}}(s \,|\, t)$ est donc nul.
f est donc résiliente d'ordre $k \geq inf\{\omega(\phi(y))|\, y \in F_2^{m-r}\} - 1$. □

Cette proposition permet d'obtenir des fonctions dont les degrés atteignent la borne rappelée ci-dessus :

Exercice 3 *Pour tout r compris entre 1 et $m-2$, montrer qu'il existe des fonctions définies par la proposition précédente qui sont de degré $m-r$ et résilientes d'ordre $r-1$.*

Nonlinéarité

A cause de la formule de Parseval, il n'existe pas de fonction de m variables qui soit m-résiliente. Il y a donc forcément une corrélation entre la sortie de la fonction et un sous-ensemble strict des coordonnées de x. Il est alors préférable que les corrélations non nulles soient aussi faibles que possible, c'est à dire que $\max_{a \in F_2^m} |\widehat{\mathrm{f}}(a)|$ soit aussi petit que possible. On rappelle que la distance de Hamming de f à l'ensemble des fonction affines (i.e. le code de Reed-Muller d'ordre 1) est égale à $2^{m-1} - \frac{1}{2} \max_{a \in F_2^m} |\widehat{\mathrm{f}}(a)|$. Le critère évoqué ci-dessus [2] est donc que cette distance soit aussi grande que possible.

Définition 15 *On appelle nonlinéarité d'une fonction booléenne f et on note N_f sa distance à l'ensemble des fonctions affines.*

On a donc

$$N_f = 2^{m-1} - \frac{1}{2} \max_{a \in F_2^m} |\widehat{\mathrm{f}}(a)|. \tag{6.4}$$

1. P. Camion, C. Carlet, P. Charpin et N. Sendrier, *On correlation-immune functions*, Advances in Cryptology, CRYPTO'91, Lecture Notes in Computer Sciences 576, Springer Verlag, pp 86-100 (1992)
2. De plus, toute approximation affine fournit un distingueur et don à une faiblesse du système

D'après la formule de Parseval, on a $N_f \leq 2^{m-1} - 2^{m/2-1}$ (borne dite universelle). Cette borne est valable pour toutes les fonctions, et elle est atteinte, si m est pair, par les fonctions dites *courbes* (qui sont telles que $\widehat{\mathrm{f}}(a) = \pm 2^{m/2}$ pour tout a). En particulier (prendre $a = 0$) elles ne sont pas équilibrées. Par contre, leurs dérivées sont équilibrées :

Exercice 4 *Montrer qu'une fonction booléenne est courbe si et seulement si toutes ses dérivées $D_a f(x) = f(x) + f(x+a)$, $a \neq 0$ sont équilibrées.*

Un exemple de fonction courbe : $f(x,y) = x \cdot \pi(y)) + h(y)$, où $x, y \in F_2^{n/2}$, où π est une permutation quelconque de $F_2^{n/2}$ et où h est une fonction booléenne quelconque (f ainsi définie est une fonction dite de Maiorana-McFarland).

Dans le cas des fonctions résilientes, on peut prouver une meilleure borne que la borne universelle.

Proposition 14 (Borne de Sarkar-Maitra) *Soit f une fonction k-résiliente ($k \leq m-2$).*
1. Alors, pour tout $a \in F_2^m$, le coefficient de Walsh $\widehat{\mathrm{f}}(a)$ est divisible par 2^{k+2}.
2. On en déduit que $N_f \leq 2^{m-1} - 2^{k+1}$. De plus, si $k \leq m/2 - 2$ avec m pair, alors on a $N_f \leq 2^{m-1} - 2^{m/2-1} - 2^{k+1}$.

Démonstration : 1. On démontre la divisibilité de $\widehat{\mathrm{f}}(a)$ par 2^{k+2} par récurrence sur le poids de Hamming de $a \in F_2^m$. Pour $w_H(a) \leq k$, elle est évidente. Pour $w_H(a) = k+1$, puis pour $w_H(a) > k+1$, on la déduit de la formule de sommation de Poisson avec $E = \{x \in F_2^m / \, x \preceq a\}$.
2. On en déduit que, d'après la relation (6.4), N_f est divisible par 2^{k+1}. Comme on sait que $N_f < 2^{m-1}$, on en déduit $N_f \leq 2^{m-1} - 2^{k+1}$. De plus, comme on sait que toute fonction courbe est non-équilibrée, on a $N_f < 2^{m-1} - 2^{m/2-1}$. Donc N_f est inférieur ou égal au plus grand multiple de 2^{k+1} qui est strictement inférieur à $2^{m-1} - 2^{m/2-1}$. Si $k \leq m/2 - 2$, alors on a $N_f \leq 2^{m-1} - 2^{m/2-1} - 2^{k+1}$, si m est pair. □

Remarque : d'après la relation de Parseval et la relation (6.4), N_f est majoré par $2^{m-1} - \dfrac{2^{m-1}}{\sqrt{\sum_{i=k+1}^{m} \binom{m}{i}}}$. Donc N_f est majoré par le plus grand entier divisible par 2^{k+1} qui est inférieur ou égal à $2^{m-1} - \dfrac{2^{m-1}}{\sqrt{\sum_{i=k+1}^{m} \binom{m}{i}}}$.

Un critère lié aux attaques algébriques

Si une fonction de filtrage (par exemple) est telle qu'il existe deux fonctions g et h de bas degrés (disons de degrés au plus d) telles que $f * g = h$ (où $*$ désigne le produit) et $g \neq 0$, alors on en déduit l'attaque suivante : notons n la longueur du LFSR filtré par f ; on considère f comme une fonction de n variables, même si elle dépend en fait de moins de variables ; on sait qu'il existe une application linéaire L de F_2^n dans F_2^n telle que la sortie du générateur soit $s_i = f(L^i(u_1, \ldots, u_n))$, où $u_1, \ldots, u_n$ est l'initialisation du LFSR (ceci est vrai plus généralement pour tous les automates linéaires combinés par une fonction booléenne). On a alors, pour tout i : $s_i * g(L^i(u_1, \ldots, u_n)) = h(L^i(u_1, \ldots, u_n))$. Cette équation en $u_1, \ldots, u_n$ est de degré au plus d, puisque L est linéaire. Une fonction offre une résistance optimale à cette attaque s'il n'existe pas g et h de bas degrés, $g \neq 0$, telles que $f * g = h$.

Exercice 5 *Montrer qu'il existe $g \neq 0$ et h telles que $f * g = h$ si et seulement si il existe $g \neq 0$, telle que $f * g = 0$ ou $(f+1) * g = 0$.*

On appelle immunité algébrique de f et on note $AI(f)$ le degré minimal des fonctions g non nulles telles que $f * g = 0$ ou $(f+1) * g = 0$.

Exercice 6 *Montrer que $AI(f) \leq \lceil n/2 \rceil$.*

6.3 Fonctions booléennes vectorielles pour les schémas par blocs

6.3.1 Fonctions booléennes vectorielles

Soient n et m deux entiers positifs. Les applications de F_2^n dans F_2^m, sont appelées des (n, m)-fonctions.

Représentation en polynôme multivarié $\mathbb{F}_{2^m}[X_1, ..., X_m]/(X_i^2 - X_i)$.

Représentation par fonction coordonnées Une (n, m)-fonction F étant donnée, les fonctions Booléennes $f_1, \ldots, f_m$ définies, pour tout $x \in F_2^n$, par

$$F(x) = (f_1(x), \ldots, f_m(x))$$

sont appelées les fonctions coordonnées de F. Quand les nombres n et m ne sont pas spécifiés, on appelle les (n, m)-fonctions des fonctions Booléennes vectorielles ou des boites-S.

Pour une résistance aux attaques différentielles, on veut que le nombre de solutions des équations $F(x) + F(x + a) = b$ soit aussi faible que possible pour tout a non nul dans F_2^n et tout b dans F_2^m. Le minimum est 2.

Définition 16 *Une (n, n)-fonction F est dite presque parfaitement nonlinéaire (notation : APN, pour "almost perfect nonlinear") si, pour tout a non nul dans F_2^n et tout b dans F_2^m, l'équation $F(x)+F(x+a) = b$ admet au plus 2 solutions (c'est à dire, soit deux solutions soit aucune).*

Pour une résistance aux attaques linéaires, on veut que toutes les fonctions coordonnées aient une haute non-linéarité. On veut aussi que toutes les combinaisons linéaires des fonctions coordonnées aient une haute non-linéarité.

Les combinaisons linéaires des fonctions coordonnées sont obtenus par $x \mapsto b.F(x) = \sum_{i=1}^{m} b_i f_i(x)$, pour b parcourant $\mathbb{F}_2^m$.

On cherche donc une fonction vectorielle $f(x)$ telle que le poids de la fonction $b.F(x) + a.x$ soit aussi proche que possible de 2^{n-1} pour tout $a \in \mathbb{F}_2^n$ et tout $b \in \mathbb{F}_2^m$.

Définition 17 *La nonlinéarité d'une (n, m)-fonction F est égale au poids minimal des fonctions $b \cdot F(x) + a \cdot x + \epsilon$, où $b \in F_2^{m*}$, $a \in F_2^n$ et $\epsilon \in F_2$.*

La nonlinéarité $\mathcal{NL}(F)$ d'une (n, m)-fonction F est donc égale à la nonlinéarité minimale des fonctions $b \cdot F$, où $b \neq 0$. D'après ce qui a été vu au chapitre précédent, on a donc :

$$\mathcal{NL}(F) = 2^{n-1} - \frac{1}{2} \max_{b \in \mathcal{B}^{m*};\ a \in \mathcal{B}^n} \left| \sum_{x \in \mathcal{B}^n} (-1)^{b \cdot F(x) \oplus a \cdot x} \right|$$

et la nonlinéarité est bornée supérieurement par $2^{n-1} - 2^{n/2-1}$. Une (n, m)-fonction est dite courbe si elle atteint cette borne, c'est à dire si toutes les fonctions booléennes $b \cdot F$, où $\beta \neq 0$, sont courbes.
Un exemple de $(n, n/2)$-fonction courbe : $F(x, y) = L(x\,\pi(y)) + H(y)$, où le produit $x\,\pi(y)$ est calculé dans le corps $F_{2^{n/2}}$, où L est une application linéaire ou affine surjective (et donc équilibrée) de $F_{2^{n/2}}$ dans F_2^m, où π est une permutation quelconque de $F_{2^{n/2}}$ et où H est une $(\frac{n}{2}, m)$-fonction quelconque (F ainsi définie est une fonction dite de Maiorana-McFarland).
D'après le résultat prouvé à l'exercice 1 du TD 2, une fonction courbe ne peut donc pas être équilibrée (i.e. avoir une sortie uniformément distribuée), ce qui la rend impropre à une utilisation cryptographique directe. D'après l'exercice 4 et le résultat prouvé à l'exercice 1 du TD 2, F est courbe si et

seulement si toutes ses dérivées $D_aF(x) = F(x) + F(x+a)$ sont équilibrées. On dit que F est alors également parfaitement nonlinéaire : cela traduit le fait que le nombre maximal de solutions de l'équation $F(x) + F(x+a) = b$, $a \in F_2^{n*}$, $b \in F_2^m$, est minimal (égal à 2^{n-m}). Mais on démontre que les fonctions courbes n'existent que pour n pair (ce qui est évident) et $m \leq n/2$.

Pour $m = n$ (qui est, de nos jours, par exemple pour l'AES, le cas pratique), on a donc $\mathcal{NL}(F) \leq 2^{n-1} - 2^{\frac{n-1}{2}}$.

Définition 18 *Une (n,n)-fonction F est dit presque courbe (notation : AB, pour "almost bent") si $\mathcal{NL}(F) = 2^{n-1} - 2^{\frac{n-1}{2}}$.*

Exercice 7 *Montrer que si une fonction est AB alors elle est APN.*

6.3.2 Représentation polynomiale des (m,m) fonctions vectorielles

Toute $(m,m$ fonction booléenne vectorielle peut être considérée comme une application de $\mathbb{F}_{2^m}$ dans lui-même.

Proposition 15 *L'ensemble des applications de $\mathbb{F}_{2^m}$ dans lui-même est isomorphe à $\mathcal{A} = \mathbb{F}_{2^m}[X]/(X^{2^m} - X$.*

Polynômes linéaires.

A un polynôme multivarié en m variables, on fait correspondre un polynôme en une variable de $\mathcal{A}$.

Définition 19 *Le degré algébrique d'un élément de $\mathcal{A}$ est le maximum des poids binaires des exposants i du polynôme.*

Proposition 16 *Le degré algébrique de $f(X) \in \mathcal{A}$ est le degré algébrique de la fonction booléenne vectorielle associée.*

Propriété : le degré algébrique est stable par composition à droite ou à gauche par une fonction affine.

Exercice 8 *Montrer que les deux notions de APN et AB sont stables par composition à droite et à gauche par un automorphisme affine, par ajout d'un endomorphisme affine, et par passage à l'inverse (dans le cas, bien sûr, où F est bijective).*

Exemples connus de fonctions APN et AB : les seuls exemples connus (aux transformations ci-dessus près) sont des fonctions définies dans le corps F_{2^n} (qu'on peut identifier à F_2^n) et de la forme $F(x) = x^s$.

Fonctions APN :

- $s = 2^n - 2$, pour n impair. $F(x)$ égale $\dfrac{1}{x}$ si $x \neq 0$ et 0 sinon. L'équation $x^{2^n-2} + (x+1)^{2^n-2} = b$ $(b \neq 0)$ admet 0 et 1 pour solutions si et seulement si $b = 1$; et elle admet des solutions différentes de 0 et 1 si et seulement si il existe $x \neq 0, 1$ tel que $\dfrac{1}{x} + \dfrac{1}{x+1} = b$, c'est à dire $x^2 + x = \dfrac{1}{b}$.

  Exercice 9 *Montrer qu'un tel x existe si et seulement si $tr\left(\dfrac{1}{b}\right) = 0$.*

  Donc F est APN si et seulement si $tr(1) = 1$, c'est à dire si n est impair.
- $s = 2^h + 1$ avec $pgcd(h,n) = 1$ et $1 \leq h \leq \dfrac{n-1}{2}$ (exercice !) ;
- $s = 2^{2h} - 2^h + 1$ avec $pgcd(h,n) = 1$ et $2 \leq h \leq \dfrac{n-1}{2}$;
- $s = 2^{\frac{4n}{5}} + 2^{\frac{3n}{5}} + 2^{\frac{2n}{5}} + 2^{\frac{n}{5}} - 1$, avec n divisible par 5.
- $s = 2^{(n-1)/2} + 3$ (n impair) ;
- $s = 2^{(n-1)/2} + 2^{(n-1)/4} - 1$, où $n \equiv 1 \pmod 4$;
- $s = 2^{(n-1)/2} + 2^{(3n-1)/4} - 1$, où $n \equiv 3 \pmod 4$.

Fonctions AB :

- $s = 2^h + 1$ avec $pgcd(h, n) = 1$ et $1 \leq h \leq \dfrac{n-1}{2}$.
- $s = 2^{2h} - 2^h + 1$ avec $pgcd(h, n) = 1$ et $2 \leq h \leq \dfrac{n-1}{2}$.
- $s = 2^{(n-1)/2} + 3$.
- $s = 2^{(n-1)/2} + 2^{(n-1)/4} - 1$, où $n \equiv 1 \pmod 4$.
- $s = 2^{(n-1)/2} + 2^{(3n-1)/4} - 1$, where $n \equiv 3 \pmod 4$.

Annexe A

DES : Data Encryption Standard

A.1 Description générale

Le DES (Data Encryption System) chiffre des blocs de 64 bits en utilisant une clé de longueur 56 bits. Les blocs chiffrés ont aussi une taille de 64 bits.

L'algorithme en lui-même se compose en

- une permutation initiale IP,
- 16 itérations d'une fonction étage f de $\mathbb{F}_2^{32}$ dans lui-même dans un schéma de Feistel,
- une transformation finale composée de la permutation qui échange les moitiés droite et gauche du mot de 64 bits et de la permutation initiale inversée IP^{-1}.

Voir figure A.1

La permutation initiale IP est une simple permutation bit à bit qui opère de la manière suivante : le bit numéro i du premier tableau est transformé en le bit correspondant.

bloc d'entrée									bloc de sortie							
1	2	3	4	5	6	7	8		58	50	42	34	26	18	10	2
9	10	11	12	13	14	15	16		60	52	44	36	28	20	12	4
17	18	19	20	21	22	23	24		62	54	46	38	30	22	14	6
25	26	27	28	29	30	31	32	$\xrightarrow{IP}$	64	56	48	40	32	24	16	8
33	34	35	36	37	38	39	40		57	49	41	33	25	17	9	1
41	42	43	44	45	46	47	48		59	51	43	35	27	19	11	3
49	50	51	52	53	54	55	56		61	53	45	37	29	21	13	5
57	58	59	60	61	62	63	64		63	55	47	39	31	23	15	7

A.2 Fonction d'étage f

La fonction d'étage f prends 32 bits en entrée et retourne 32 bits. Elle est composée de :

- une fonction d'expansion affine E qui transforme un bloc de 32 bits en bloc de 48 bits
- d'un x-or avec la sous-clé K_i (longue de 48 bits) de l'étage i
- d'une fonction S constituée de 8 S-boîtes, chacune transforme un mot de 6 bits en mot de 4 bits
- d'une permutation P finale.

On a donc $L_{i+1} = L_i \oplus f(R_i) = L_i \oplus P(S(E(R_i) \oplus K_i))$.

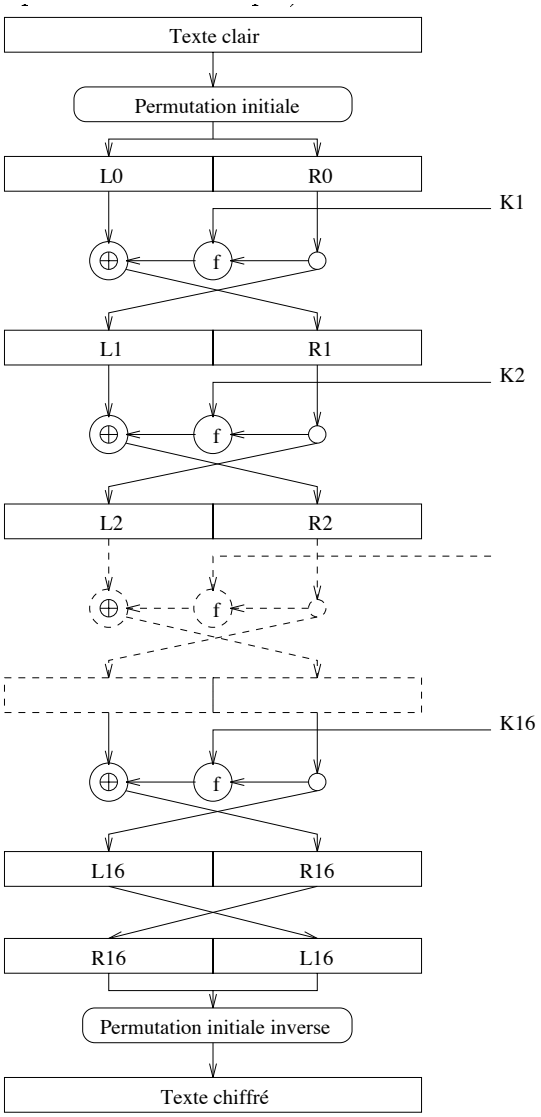
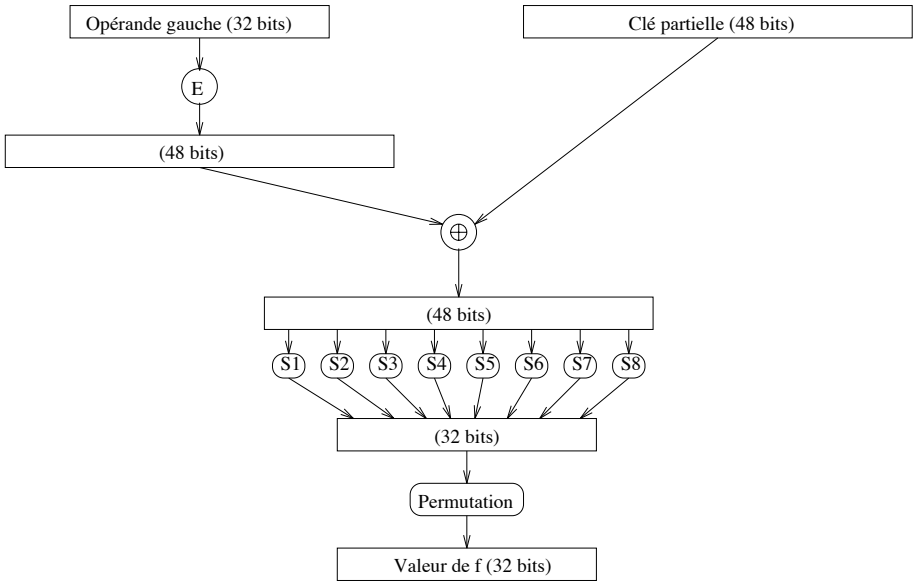


FIGURE A.1 – D.E.S. : Data Encryption System

La fonction f



Fonction E d'expansion

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Permutation P finale

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Les S-boîtes

Il y a huit S-boîtes différentes. On les représente par des tableaux à 2 lignes et 16 colonnes. Les premiers et derniers bits de l'entrée déterminent une ligne du tableau, les autres bits déterminent une colonne. La valeur numérique trouvée à cet endroit indique la valeur des quatre bits de sortie.

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

A.3 Génération des sous-clés

La clé maître K de 56 bits est utilisée pour créer 16 sous-clés de tour K_i de taille 48.
Les étapes sont les suivantes :

- on applique une permutation $PC1$ à K.
- A chaque itération, chaque moitié de la chaîne de 56 bits subit une permutation circulaire à gauche , d'un cran aux étapes 1, 2, 9 et 16, de 2 crans aux autres.
- La clé de tours K_i de 48 bits est extraite par la règle d'extraction $PC2$.

Voir Figure A.2

Permutation PC_1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Règle d'extraction PC_2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

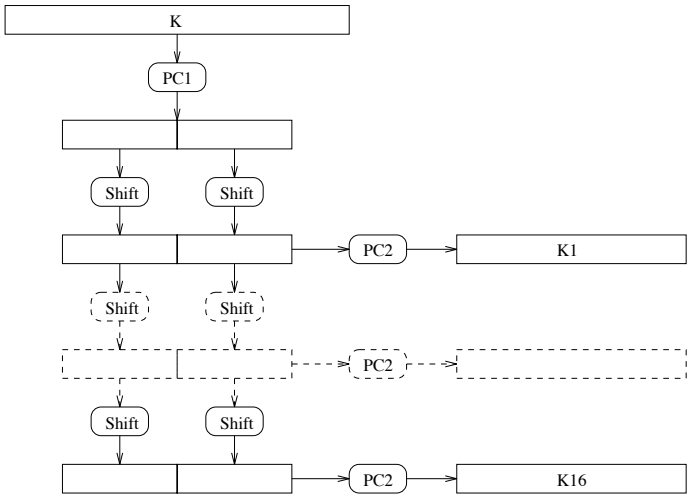


FIGURE A.2 – Génération des sous-clés

Annexe B

RC4

RC4 est un algorithme de chiffrement à flot conçu en 1987 par Rivest et dédié aux applications logicielles. Il est largement déployé notamment dans le protocole SSL/TLS et la norme WEP pour les réseaux sans fil, IEEE 802.11. RC4 présente certaines faiblesses dues à son initialisation, qui est relativement faible, et qui ne prévoit pas de valeur initiale pour la re-synchronisation. Employé par exemple avec le protocole de re-synchronisation choisi dans la norme IEEE 802.11, RC4 s'avère extrêmement faible.

RC4 est un algorithme de chiffrement à flot destiné aux applications logicielles. Il a été conçu par R. Rivest en 1987 pour les laboratoires RSA. Malgré un certain nombre de faiblesses, il est encore très utilisé aujourd'hui, notamment du fait de sa vitesse élevée (7 cycles par octet sur un Pentium III par exemple). Il est employé par exemple dans SSL/TLS, protocole permettant d'assurer la confidentialité des transactions Web, dans la norme de chiffrement WEP (Wired Equivalent Privacy) pour les réseaux sans fil, IEEE 802.11b...

Paramètres.

- Longueur de clef : variable, entre 40 et 1024 bits.
- Pas de valeur initiale.

L'état interne de RC4 est formé d'un tableau de 2^n éléments de n bits. A chaque instant, ce tableau correspond à une permutation des mots de n bits. Généralement, on prendra $n = 8$, qui correspond à un tableau de 256 octets.

Description. RC4 est composé d'une phase d'initialisation, qui consiste à engendrer une permutation des mots de n bits à partir de la clef secrète.

Puis, à chaque instant, on modifie le tableau initial en permutant deux de ses éléments, et on produit un mot de n bits de suite chiffrante, qui correspond à un élément du tableau.

Toutes les additions effectuées dans RC4 sont bien sûr des additions modulo 2^n.

Clef secrète. K, composé de k mots de n bits, $K[0], \ldots, K[k-1]$.

Initialisation.

- Pour i de 0 à $2^n - 1$, $S[i] \leftarrow i$.
- $j \leftarrow 0$
- Pour i de 0 à $2^n - 1$ faire
  - $j \leftarrow j + S[j] + K[i \bmod k]$
  - échanger $S[i]$ et $S[j]$

Génération de la suite chiffrante.

- $i \leftarrow 0, j \leftarrow 0$
- Répéter
  - $i \leftarrow i + 1$
  - $j \leftarrow j + S[i]$
  - échanger $S[i]$ et $S[j]$.
  - Retourner $S[S[i] + S[j]$

TABLE B.1 – RC4

Annexe C

AES : Advanced Encryption Standard

l'AES est un algorithme itératif de chiffrement par blocs qui chiffre des blocs de longueur B=128 bits (dans la version initiale de Rijndael trois longueurs étaient proposées : 128, 192 ou 256 bits) sous des clés de taille k=128, 192 ou 256 bits. Il est composé d'une addition de clé initial (avec la sous-clé K_0), d'un nombre variable d'étages : 10, 12 ou 14, fonction de la longueur de la clé et de celle des blocs à chiffrer :

Nb tours	B=128	B=192	B=256
k=128	10	12	14
k=192	12	12	14
k=256	14	14	14

Notons que la norme FIPS-197 n'autorise que le cas d'un bloc de 128 bits.

Il utilise une structure parallèle mettant en œuvre quatre transformations où les blocs à chiffrer sont représentés par des matrices d'octets :

- SubBytes : substitution non linéaire faisant appel à une boîte S pour garantir de bonnes propriétés de confusion.
- ShiftRows : application linéaire qui décale les lignes de la matrice.
- MixColumns : application linéaire (multiplication matricielle dans le corps $GF(256)$, fondée sur les codes MDS) garantissant une bonne diffusion.
- AddRoundKey : addition de clé classique (x-or entre le bloc courant et la sous-clé de l'étage.

Notons que le dernier étage ne comporte pas la transformation MixColumns. On peut donc représenter la fonction de chiffrement par le schéma de la figure C.1.

La fonction de déchiffrement de l'AES se déduit très facilement de la fonction de chiffrement. Il suffit, avec la même structure générale, d'utiliser les fonctions inverses des précédentes et d'inverser leur position dans le tour.

Dans la suite, nous nous représenterons un bloc de 128 bits à chiffrer par la matrice 4×4 d'octets suivante :

$$\mathrm{A} = \begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array}$$

La Fonction d'étage

Nous allons à présent décrire la fonction d'étage de l'AES composée de 4 transformations : SubBytes, ShiftRows, MixColumns et AddRoundKey.

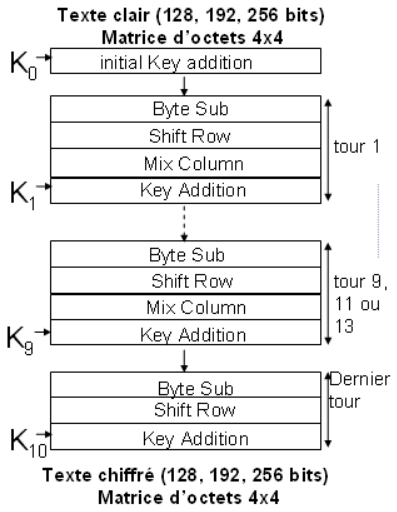


FIGURE C.1 – Schéma général de l'AES.

SubBytes. L'opération SubBytes fait intervenir une boîte S, permutation non linéaire de $GF(256) \approx GF(2)[x]/(x^8+x^4+x^3+x+1)$ dans lui-même, de la façon suivante :

$$\begin{array}{|c|c|c|c|} \hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline S(a_{0,0}) & S(a_{0,1}) & S(a_{0,2}) & S(a_{0,3}) \\ \hline S(a_{1,0}) & S(a_{1,1}) & S(a_{1,2}) & S(a_{1,3}) \\ \hline S(a_{2,0}) & S(a_{2,1}) & S(a_{2,2}) & S(a_{2,3}) \\ \hline S(a_{3,0}) & S(a_{3,1}) & S(a_{3,2}) & S(a_{3,3}) \\ \hline \end{array}$$

Si on écrit un octet d'entrée $a_{i,j} = (xy)_h$ comme la concaténation de x et y deux mots de 4 bits, la sortie correspondante $b_{i,j}$ sera l'élément de la ligne x et de la colonne y de la table donnée à la figure C.2 représentant S.

Par exemple, si l'entrée est $(53)_h$, la sortie correspondante est, toujours en hexadécimal, $(ed)_h$.

La boîte S utilisée dans la transformation SubBytes est la composée de deux transformations :

- L'inversion (avec $0^{-1} = 0$) dans le corps $GF(256)$ représenté sous la forme $GF(256) \approx GF(2)[x]/(x^8+x^4+x^3+x+1)$. Cette opération permet de garantir une bonne confusion la plus forte non linéarité possible, c'est à dire de rendre minimaux les coefficients utilisés en cryptanalyse différentielle et en cryptanalyse linéaire. Ici, on a $DP_S = 2^{-6}$ et $LP_S = 2^{-6}$
- Une transformation affine de $GF(2)^8$ dans $GF(2)^8$ de la forme $y = A \cdot x + B$ où A est une matrice 8×8 à coefficients dans $GF(2)$ et où B est un vecteur colonne de taille 8 également à coefficients dans $GF(2)$. Les octets d'entrée et de sortie sont représentés sous forme d'un vecteur colonne de taille 8 à coefficients dans $GF(2)$:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

L'utilisation d'une transformation affine est destinée à cacher les propriétés algébriques remarquables propres à l'inversion. Cette dernière a été également choisie pour éviter la présence de

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

FIGURE C.2 – La boîte S de l'AES de $GF(256)$ dans $GF(256)$.

points fixes (i.e. tels que $S(x) = x$).

ShiftRows. L'opération ShiftRows consiste à faire subir une permutation circulaire vers la gauche aux lignes de la matrice à chiffrer :

m	n	o	p
i	j	k	l
d	e	f	g
w	x	y	z

pas de décalage ←

décalage de 1 ←

décalage de 2 ←

décalage de 3 ←

m	n	o	p
j	k	l	i
f	g	d	e
z	w	x	y

MixColumns. MixColumns peut être vu comme une multiplication matricielle portant sur chaque colonne de la matrice représentant le bloc d'entrée :

$$\begin{bmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \\ b_{3,i} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{bmatrix} \text{ pour } i \text{ allant de 0 à 3}$$

Pour le calcul, on représente le corps $GF(256)$ par $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$ et chaque octet par un polynôme de degré au plus 7. La multiplication est donc définie comme une multiplication de polynômes binaires modulo $x^8 + x^4 + x^3 + x + 1$. Par exemple, la multiplication par $'02'$ correspond à la multiplication par x, les chiffres entre côtes étant exprimés en hexadécimal. On note $\bullet$ cette multiplication.

AddRoundKey. L'addition de clé est un simple Xor bit à bit sur tous les octets de la matrice :

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

$=$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$\oplus$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

La génération des sous-clés

Cette opération permet de générer les sous-clés du schéma de chiffrement à partir de la clé maître. Le cadencement de clé se déroule en deux étapes : d'abord l'expansion de clé, où la clé de chiffrement est transformée en une clé étendue dont la longueur est égale au nombre de bits nécessaires à la formation de toutes les sous-clés, ensuite la sélection des sous-clés générées à partir de la clé étendue.

L'Expansion de Clé. On note :

- Nr le nombre de tours.
- Nb le nombre de colonnes des blocs à chiffrer dans la représentation matricielle, c'est à dire la longueur des blocs divisée par 32. Ici, $Nb = 4$.
- Nk le nombre de colonnes de la clé de chiffrement ($= \frac{\text{longueur de clé}}{32}$).

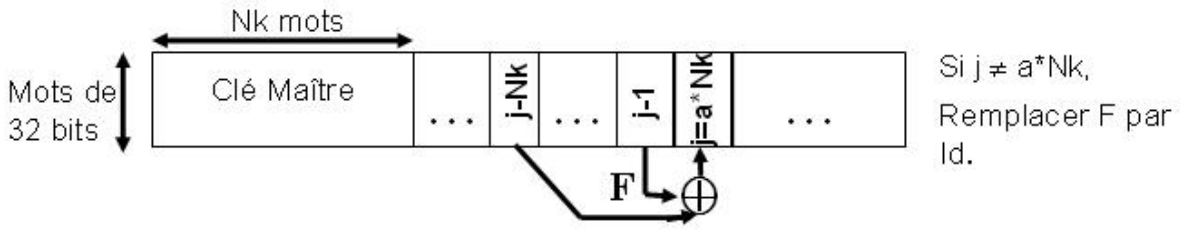
La clé étendue est représentée par un tableau de mots de 4 octets qui sont notés $W[0]$ à $W[Nb*(Nr+1)-1]$. On note CC la clé de chiffrement, représentée de la même manière. On peut alors représenter l'expansion de clé par la procédure récursive suivante :

- Les Nk premières valeurs de la clé étendue (c'est à dire les Nk premiers mots de 4 octets) reçoivent les Nk premières valeurs de la clé de chiffrement.
- Pour les valeurs de j multiples de Nk, on applique la transformation suivante : $W[j] \leftarrow W[j-Nk] \oplus Sub4Byte(W[j-1]^{<<1}) \oplus Rcon[\frac{j}{Nk}]$ où $Sub4Byte$ est la boîte S de Rijndael du paragraphe 1.2 appliquée à chacun des quatre octets, $^{<<1}$ désigne la rotation circulaire d'un octet vers la gauche et $Rcon$ une table de constantes.
- Pour les valeurs i telles que $i \bmod Nk \neq 0$ et $i \bmod Nk \neq 4$ pour $Nk = 8$, on utilise la relation : $W[i+j] \leftarrow W[i+j-Nk] \oplus W[i+j-1]$.
- Si dans le cas $Nk = 8$ on a $i \bmod Nk = 4$, alors on a $W[i] \leftarrow W[i-Nk] \oplus Sub4Byte(W[i-1])$

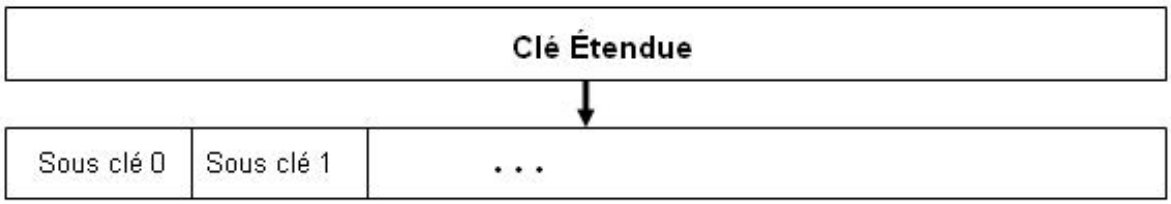
L'expression algorithmique en est :

Pour i allant de 0 à $(Nk-1)$ faire
  $W[i] \leftarrow CC[i]$
Fin Pour
Pour i allant de Nk à $Nb*(Nr+1)-1$ faire
  Si (i mod Nk=0) alors
    $W[i] \leftarrow W[i-Nk] \oplus Sub4Byte(W[i-1]^{<<1}) \oplus Rcon[\frac{i}{Nk}]$
  Sinon Si ($Nk > 6$) et (i mod Nk=4) alors
    $W[i] \leftarrow W[i-Nk] \oplus Sub4Byte(W[i-1])$
  Sinon $W[i] \leftarrow W[i-Nk] \oplus W[i-1]$
  Fin Si
  Fin Si
Fin Pour

On peut résumer l'expansion de clé par le schéma suivant :



Sélection des Sous-Clés. La sous-clé i, de taille Nb est simplement donnée par les mots $W[Nb*i]$ à $W[Nb*(i+1)-1]$. On obtient par exemple le schéma suivant :


Clé Étendue
Sous clé 0
Sous clé 1
. . .