Cours de Cryptanalyse

Guilhem Castagnos

Septembre – Décembre 2018

version du 11 septembre 2018

Table des matières

I Introduction **1**
- 1 Sources . . . 1
- 2 Cryptographie Symétrique . . . 1
  - 2.1 Définition . . . 1
  - 2.2 Cryptanalyse et Sécurité . . . 2
  - 2.3 Deux grandes familles : chiffrement par flot et par bloc . . . 3
- 3 Cryptographie asymétrique . . . 5

II Chiffrement par flot, introduction **7**
- 1 Chiffrement par flot additif et chiffrement de Vernam . . . 7
- 2 Chiffrements synchrones et auto synchronisant . . . 7
- 3 Un exemple : RC4 . . . 8
- 4 Vecteur d'initialisation . . . 9
- 5 Premières attaques . . . 10
  - 5.1 Tailles clef, IV, état interne . . . 10
  - 5.2 Critères statistiques . . . 11

III Registres à décalage à rétroaction linéaire **13**
- 1 LFSR et suites à récurrence linéaire . . . 13
- 2 LFSR et polynômes de $\mathbf{F}_2[X]$ . . . 15
- 3 L'algorithme de Berlekamp-Massey . . . 17

IV LFSR combinés et filtrés **19**
- 1 Constructions . . . 19
- 2 Fonctions booléennes . . . 20
- 3 LFSR combinés et attaques par corrélation . . . 21
- 4 LFSR filtrés et attaques algébriques . . . 23

V Quelques chiffrements par flot actuels **27**
- 1 E0 . . . 27
- 2 A5/1 . . . 27
- 3 Snow 2.0 . . . 28
- 4 Grain . . . 29
- 5 ChaCha20 . . . 29

VI Fonction de hachage **31**
- 1 Définitions . . . 31
- 2 Applications . . . 32
- 3 Exemples . . . 32
- 4 Construction de Merkle-Damgård . . . 33
- 5 Quelques cryptanalyses . . . 34

VII Chiffrement par bloc, introduction **37**
1 Introduction 37
2 Schéma de Feistel, le DES 38
3 Schéma substitution permutation (SPN), l'AES 41

VIII Cryptanalyses linéaire et différentielle **43**
1 Cryptanalyse linéaire 43
2 Cryptanalyse différentielle 45

IX Réseaux euclidiens et cryptanalyse **49**
1 Réseaux euclidiens 49
2 Réduction de réseau 51
2.1 Cas de la dimension 2 (Lagrange - Gauss) 51
2.2 En dimension supérieure 52
3 Application à la cryptanalyse 54
3.1 Cryptanalyse de Merkle-Hellman 54
3.2 Attaques de Coppersmith 56

Chapitre I

Introduction

1. Sources

- G. Zémor, *Cours de cryptographie*, Cassini, 2000.
- A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *The Handbook of Applied Cryptography*, Fifth Printing, CRC Press, 2001, en ligne sur `http://www.cacr.math.uwaterloo.ca/hac/`
- C. Bachoc, *Cours de cryptographie symétrique* et *Réseaux et cryptographie*, voir en ligne sur `http://www.math.u-bordeaux1.fr/~bachoc/enseignement.html`
- D. Vergnaud, *Exercices et problèmes de cryptographie*, Dunod, 2012
- Le *Référentiel général de sécurité* (RGS) de l'Agence nationale de la sécurité des systèmes d'information, (ANSSI), en ligne sur `http://www.ssi.gouv.fr/administration/reglementation/confiance-numerique/le-referentiel-general-de-securite-rgs/`

2. Cryptographie Symétrique

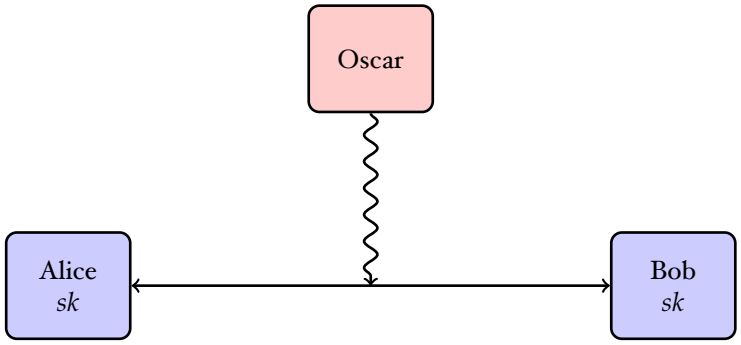
2.1. Définition

La communication se fait de manière symétrique entre Alice et Bob.

Définition I – 1 (Schéma de chiffrement symétrique). Constitué de trois algorithmes :

- KeyGen : un algorithme probabiliste avec en entrée un paramètre de sécurité et en sortie la clef secrète s_k ;
- Encrypt, on notera $\mathsf{Encrypt}_{sk}(m) = c$ algorithme déterministe avec en entrée le message en clair et la clef secrète et en sortie le message chiffré ;
- Decrypt, on notera $\mathsf{Decrypt}_{sk}(c) = m$ algorithme déterministe avec en entrée le message chiffré et la clef secrète et en sortie le message en clair.

Le système est correct si : $\mathsf{Decrypt}_{sk}(\mathsf{Encrypt}_{sk}(m)) = m$



$$\mathsf{Encrypt}_{sk}(m_A) = c_A \longrightarrow \mathsf{Decrypt}_{sk}(c_A) = m_A$$

$$\mathsf{Encrypt}_{sk}(m_B) = c_B \longleftarrow \mathsf{Decrypt}_{sk}(c_B) = m_B$$

2.2. Cryptanalyse et Sécurité

On définit la sécurité en définissant un attaquant : ses moyens et ses buts.

Moyens de l'attaquant

On a, par ordre décroissant de difficulté pour l'attaquant :

- *l'attaque à chiffré seul* : l'attaquant ne connaît que le chiffré (Oscar dans dessin classique) moyen le plus faible ;
- *l'attaque à clair(s) connu(s)* : l'attaquant connaît un ou plusieurs (éventuellement un grand nombre) de couples clairs chiffrés, réaliste : entête de fichier connu : mail, image jpeg... ;
- *l'attaque à clair(s) choisi(s)* : semblable à la précédente, mais pour laquelle l'attaquant peut choisir les clairs de façon que son attaque réussisse avec une meilleure probabilité. Elle est dite *adaptative* si les choix sont faits en fonction des résultats des attaques sur les couples précédents ; algorithme en boite noire : exécuté sur carte à puce par exemple. Moyen le plus fort ;
- *l'attaque à chiffré(s) choisi(s)* : qui peut être elle aussi adaptative, où l'attaquant obtient les déchiffrements correspondants, (idem attaque précédente).

But du cryptanalyste

Principe de Kerkhoffs (1883) : les algorithmes de chiffrement Encrypt et de déchiffrement Decrypt sont connus. Seule la clef sk est secrète (et nécessite un échange préalable).

- retrouver la clef sk (but le plus fort) ;
- Déchiffrer un chiffré, c'est à dire casser la notion de sens unique (OW) : à partir de c retrouver m tel que $\mathsf{Decrypt}_{sk}(c) = m$;
- Retrouver une information (par exemple un bit, le poids de Hamming,...) sur le clair m à partir de son chiffré c : casser la sécurité sémantique (but le plus faible).

Recherche exhaustive

En pratique tous les systèmes sont cassables si l'espace des clefs utilisés est fini. En effet, on peut retrouver la clef par l'attaque exhaustive. Elle consiste à essayer toutes les clés l'une après l'autre jusqu'à trouver la bonne. Cette méthode a l'avantage d'être générique et parallélisable (on peut distribuer le calcul sur de nombreuses machines). L'attaquant essaye toutes des clés sk jusqu'à ce que $\mathsf{Decrypt}_{sk}(c)$ ressemble à un texte clair (attaque à chiffré seul visant à retrouver la clef). Ou alors attaque à clair connu : l'attaquant possède (m, c) il teste toutes les s_k possibles c'est à dire si $\mathsf{Decrypt}_{sk}(c) = m$. Le temps moyen de cette attaque est égal au temps d'un déchiffrement multiplié par la moitié de la taille de l'espace des clés (on suppose que les clefs sont tirées avec probabilité uniforme). En effet, il faudra, en moyenne, essayer la moitié des clés avant de trouver la bonne.

Remarque. En exercice : temps de recherche exhaustive pour 56 bits et 128 bits. On suppose que l'on peut faire 10^6 test par seconde et qu'il y a 10^5 secondes dans un jour. Combien de temps dure la recherche exhaustive d'une clef de 56 bits ? et de 128 bits ?

Un exemple réel : un groupe de pression américain, l'EFF, a investi $250000 dans une machine spécialisée, nommée *Deep crack*, en 1998, qui essaye toutes les clés de l'algorithme DES (56 bits) en environ 3 jours. Ceci utilise des processeurs spécialisés, construits pour l'occasion et absolument inutile pour quoi que ce soit d'autre que casser du DES. À la fin des années 2000, plusieurs machines à base de FPGA ont été développées (COPACOBANA...). Elles cassent le DES en moins d'une journée pour un coût bien moindre (moins de $10000).

De nos jours, l'ordre de grandeur des grands calculs distribués est supérieur, par exemple, le record de factorisation (RSA–768) en 2009 est estimé à un calcul de 2^{66} opérations élémentaires. Le calcul de collisions pour SHA1 en 2017 a nécessité l'équivalent de $2^{63.1}$ hachages (ce qui est estimé être légèrement plus coûteux). Dans `https://eprint.iacr.org/2013/635.pdf` (Lenstra, Kleinjung, Thomé), l'énergie utilisée pour faire ce calcul est estimée équivalent à celle nécessaire pour faire bouillir l'eau de deux piscines olympiques, initialement à 20 degrés. Faire bouillir toute l'eau de la planète est estimé équivalent à faire 2^{114} opérations...

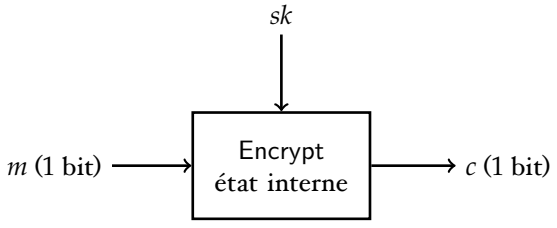
Buts du cryptanalyste et cryptographe

Le but du cryptanalyste est de trouver une attaque plus rapide que la recherche exhaustive. Le but du cryptographe est de concevoir un système dont la seule attaque connue est la recherche exhaustive. La sécurité est donc donnée par la taille de la clef. Aujourd'hui, on veut au minimum 80 bits voire 128 bits *de sécurité*. Cela signifie que les attaques prennent au minimum 2^{80} (resp. 2^{128}) « opérations élémentaires » (exécution de l'algorithme de déchiffrement par exemple).

2.3. Deux grandes familles : chiffrement par flot et par bloc

Chiffrement par flot (*stream cipher*)

L'état de Encrypt est initialisé par sk puis évolue à chaque bit chiffré (sinon par exemple, 0 est chiffré par 1 et 1 par 0 tout le temps). L'état au temps t est fonction de l'état précédent au temps $t-1$ et potentiellement de la clef et du message clair. Notons que globalement, le chiffrement d'une chaîne de bit est déterministe.



Ce chiffrement est plus rapide que le chiffrement par bloc, surtout en implantation matérielle car la complexité matérielle est plus faible. Il est basé sur des primitives simples (LFSR cf. suite du cours).

Les chiffrements par flots peuvent être traités avec une mémoire limitée, ils traitent le message clair bit par bit et il n'est donc pas nécessaire de stocker tout le texte clair comme c'est le cas pour les systèmes de chiffrement par blocs. Ils sont donc utilisés dans les communications pour garantir la confidentialité : GSM (A5/1), Bluetooth (EO), WiFi (WEP, RC4).

Beaucoup d'attaques sur ces systèmes par exemple :

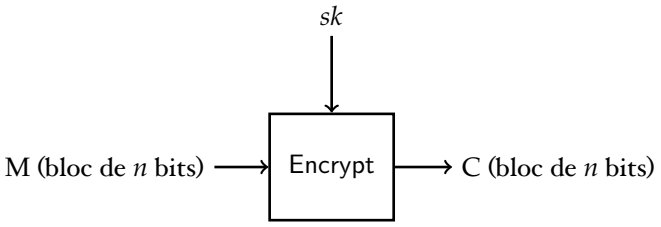
- RC4 (1987) clef de 40 à 256 bits, utilisé dans le WEP et WPA, et dans TLS (interdit en 2015), très nombreuses attaques lorsqu'il est utilisé dans ces protocoles

- A5/1 (1987), 54 bits, utilisé dans le GSM, attaque à texte clair connue nécessite une minute de calcul et quelques secondes de conversation en clair

D'autres résistent : par exemple : snow-2.0 (02) , 128 ou 256 bits standardisé, snow 3G, variante utilisée pour la 3G/4G ou ChaCha20 (2008) utilisé dans TLS à partir de la version 1.2.

Concours eSTREAM initié en 2004 : 35 candidats. En 2008 : 7 candidats sélectionnés. Un concours public permet d'apporter la sécurité par expertise de la communauté cryptographique. cf. `http://www.ecrypt.eu.org/stream/` (Chacha20 est une variante de Salsa20 adopté par eSTREAM).

Chiffrement par Bloc (*block cipher*)



Un même message est toujours chiffré de la même manière pour une clef donnée.

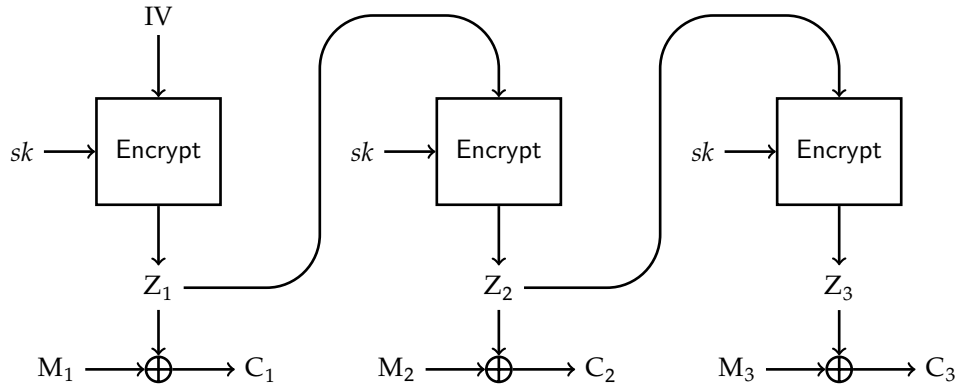
Dans un système par blocs, chaque clair est découpé en blocs de même longueur et chiffré bloc par bloc. La longueur n des blocs doit également être suffisante pour éviter les attaques dites par dictionnaire : on établit la correspondance entre les 2^n blocs chiffrés possibles et les blocs clairs lors d'une attaque à clairs connus.

Les schémas par blocs sont plus lents et ont une implantation matérielle ou logicielle plus coûteuse que les schémas par flots. Il sont bien adaptés au stockage informatique.

La construction d'un système de chiffrement par bloc utilise un schéma itératif, une même fonction est itérée successivement sur le bloc à chiffrer (plusieurs tours ou rondes) avec des clef de tours dérivées de la clef secrète.

Exemples : DES 1974, 56 bits de clef, blocs de 64 bits, puis AES, 2000, 128, 192 ou 256 bits de clef, blocs de 128 bits. La sécurité de ces schémas par blocs retenus comme standards repose sur le fait qu'avant d'être retenus (et après !), ils ont été massivement attaqués par la communauté cryptographique. Ces schémas qui ont résisté sont alors considérés comme sûrs.

Plusieurs mode d'opérations : *Electronic CodeBook* (ECB) décrit au dessus, un permet d'être utilisé comme chiffrement par flot : *Output FeedBack* (OFB). Vu la faiblesse relative de nombreux chiffrements à flot on utilise souvent l'AES en mode OFB (un des but du concours eSTREAM était de proposer des chiffrements à flots sûrs et plus rapides et moins coûteux que l'AES).



3. Cryptographie asymétrique

On a une définition formelle similaire à celle de la cryptographie symétrique. De même pour la définition de l'attaquant.

Principales différences entre chiffrement asymétrique et symétrique :

- Cryptographie asymétrique ou à clef publique ;
- Cryptographie asymétrique permet autre application que le chiffrement : signature, authentification, échange de clef symétrique ;
- Beaucoup plus lent que le symétrique ;
- Sécurité basée sur un problème algorithmique difficile, dont la difficulté est paramétrée par la taille des clefs utilisées (ne repose plus sur la recherche exhaustive : aucun sens de comparer pour la sécurité la taille des clef symétrique et asymétrique) ;
- Clefs beaucoup plus longues qu'en symétrique (1024 bits).

Pour la définition précise de la sécurité et la réduction à des problèmes algorithmiques voir le cours de cryptographie avancée. Pour l'étude et l'attaque de ces problèmes algorithmiques (logarithme discret, factorisation), voir le cours d'algorithmique arithmétique.

Dans ce cours, on verra des cryptanalyses en clef publique à base de réduction des réseaux euclidiens : par exemple des attaques sur RSA dans certains cas particuliers.

Chapitre II

Chiffrement par flot, introduction

1. Chiffrement par flot additif et chiffrement de Vernam

Les algorithmes de chiffrement à flot sont inspirés de l'algorithme du chiffrement de Vernam (1917, ou masque jetable, *One-Time Pad*). Ce système consiste à chiffrer un message m de n bits $m = (m_1, \dots, m_n)$ avec une clef aléatoire secrète de même longueur que le message clair, $sk = (z_1, \dots, z_n)$, à l'aide d'un « ou exclusif » (XOR). Le texte chiffré c est calculé de la manière suivante :

$$\forall 1 \leqslant i \leqslant n, \quad c_i = m_i \oplus z_i.$$

Si la clef s_k est choisie avec probabilité uniforme parmi les chaînes de n bits et utilisée une seule fois, le chiffrement de Vernam assure une sécurité inconditionnelle (au sens de la théorie de l'information). C'est à dire que la connaissance du chiffré n'apporte aucune information sur le message : la probabilité que le message clair soit un certain message m ($1/2^n$ en supposant que le message est uniformément distribué) est la même connaissant c ou non. (théorème de Shannon, 1949).

Le principal inconvénient de ce système est que la clef doit être de la même longueur que le message. Comme la clef ne doit être utilisée qu'une seule fois, il faut donc établir un canal sûr permettant de transmettre cette clef pour chaque message que l'on souhaite transmettre. Exceptés des cas très particuliers (« téléphone rouge »), ce système est trop contraignant pour être utilisé en pratique.

Les systèmes de chiffrement à flot s'inspirent du système de Vernam en conservant l'opération de XOR pour le chiffrement et le déchiffrement et en résolvant le problème de transmission de clef très longue. En effet, le principe d'un système de chiffrement à flot consiste à générer une **suite chiffrante**, $z_1, z_2, \dots, z_n$, correspondant à la clef de Vernam, à partir d'une clef secrète courte (par exemple 80 ou 128 bits).

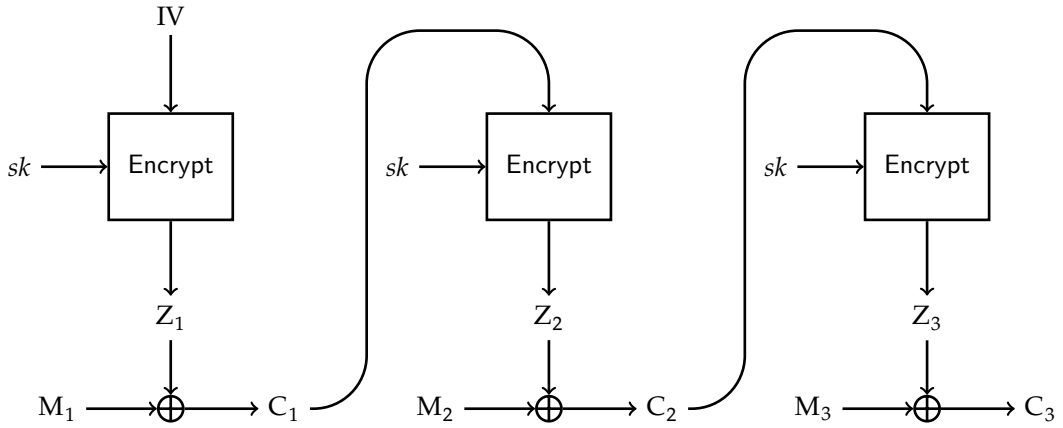
En revanche, contrairement au système de Vernam, les systèmes de chiffrement à flot ne peuvent pas garantir une sécurité inconditionnelle puisque les suites chiffrantes générées ne sont plus aléatoires mais pseudo-aléatoires (seule la clef secrète est choisie aléatoirement au départ. La suite chiffrante produite ensuite est déterministe, elle doit pouvoir être reproduite par Bob pour déchiffrer).

La quasi totalité des chiffrements par flot sont dits additifs car reprenant cette construction d'addition bit à bit d'une suite chiffrante au message clair. Celle-ci peut-être produite de deux manières, correspondant aux systèmes dits **synchrones** et **auto synchronisant**.

2. Chiffrements synchrones et auto synchronisant

La suite chiffrante produite par un système de chiffrement **auto-synchronisant** dépend de la clef secrète s_k et du texte chiffré précédemment généré. Plus précisément, après une phase d'initialisation, au temps i, le bit de suite chiffrante z_i est fonction uniquement de la clef secrète sk et des ℓ bits de chiffrés précédents, $c_{i-1-\ell}, \dots, c_{i-1}$ (il n'y a pas d'état interne).

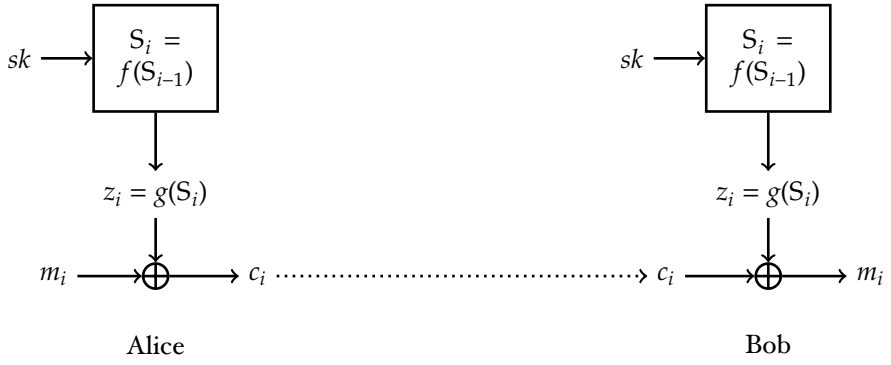
C'est un mode de fonctionnement proche du mode CFB (*Cipher FeedBack*) des chiffrements par blocs.



Quelques caractéristiques d'un chiffrement par flot auto-synchronisant :

- auto-synchronisant ! Si des caractères sont perdus ou ajoutés dans le texte chiffré, le procédé de déchiffrement se re-synchronise dès que l'on a ℓ bits de chiffrés consécutifs valides.
- Diffusion : un bit du clair influe sur toute la suite du texte chiffré. (rend plus difficiles les attaques basées sur une analyse statistique utilisant les redondances du texte clair).
- Propagation d'erreurs : si un bit de chiffré est modifié ou supprimé, le déchiffrement de seulement ℓ bits (les suivants) est corrompu.

En pratique, peu de systèmes auto-synchronisant sûrs ont été proposés. La majorité des systèmes de chiffrements par flot sont **synchrones additifs** (dans la suite on se concentrera là dessus). Pour de tels systèmes, la suite chiffrante est générée indépendamment du texte clair et du texte chiffré, elle ne dépend que de la clef secrète. Ces systèmes peuvent être vus comme des automates, dont l'état interne est initialisé par une clef secrète à l'instant $t = 0$, $S_0 = h(sk)$. Au temps i, l'état interne S_i est fonction de l'état au temps précédent, $f(S_i) = f(S_{i-1})$ et un bit de suite chiffrante fonction de l'état est produit, $z_i = g(S_i)$, est produit, générant terme à terme une suite pseudo aléatoire.



Alice et Bob doivent être synchronisés. Ainsi, si des bits de chiffrés sont perdus ou ajoutés lors de la transmission, le déchiffrement échoue. Pour pallier cela, on peut réinitialiser, ou placer des marqueurs à intervalles réguliers dans le texte chiffré. Par contre, si un bit est modifié lors de la transmission, il ne pertube pas le déchiffrement des bits suivants. Il est donc important d'utiliser de plus un mécanisme garantissant l'intégrité des données.

3. Un exemple : RC4

L'algorithme RC4 a été conçu par Rivest en 1987 et utilisé dans de nombreux protocoles pour sa rapidité et sa simplicité, par exemple dans le WEP ou SSL. Ce système de chiffrement par flot synchrone additif n'a

jamais été décrit officiellement mais une description de son fonctionnement a été donné sur une *mailing list* en 1994.

L'état interne de RC4 représente une permutation S des entiers de 0 à 255. On note les 256 images de cette permutation sous forme d'un tableau de 256 d'octets : S[0], S[1], ... S[255]. La clef secrète sk est constituée de k octets avec $5 \leqslant k \leqslant 32$ (40 à 256 bits). On note, pour $0 \leqslant i < k$, $sk[i]$ chaque octet de la clef, identifié avec un entier entre 0 et 255.

L'état est initialisé comme suit :

- Initialiser la permutation S à l'identité, soit $S[i] = i$ pour $0 \leqslant i < 256$.
- Initialiser un indice j à 0
- Pour i de 0 à 255 :
  - $j \leftarrow (j + S[i] + sk[i \mod k]) \mod 256$
  - Échanger $S[i]$ et $S[j]$

La production de suite chiffrante (une suite d'octets) et la mise à jour de l'état se font ainsi

- Initialiser deux indices i et j à 0
- Boucle :
  - $i \leftarrow (i + 1) \mod 256$
  - $j \leftarrow (j + S[i]) \mod 256$
  - Échanger $S[i]$ et $S[j]$
  - Sortir $z = S[(S[i] + S[j]) \mod 256]$

Les principales attaques sur RC4 (notamment celle Fluhrer, Mantin and Shamir en 2001) reposent sur des faiblesses de l'algorithme d'initialisation : les premiers octets de suite chiffrante donnent des informations sur la clef secrète et ont des biais statistiques important. Ceci a servi de base aux attaques sur le WEP (où l'étape d'initialisation contient de plus un mécanisme d'IV) ainsi qu'à des attaques en 2013 sur TLS. RC4 est maintenant considéré comme obsolète.

4. Vecteur d'initialisation

L'utilisation typique d'un chiffrement à flot (et des chiffrements symétriques en général) pour sécuriser une session de communication entre Alice et Bob commence par un échange de clef asymétrique (Diffie-Hellman par exemple) pour établir la clef secrète sk . Si Alice et Bob réutilisent lors de deux sessions la même clef secrète s_k, la même suite chiffrante $(z_i)_{i\geqslant 0}$ sera produite. Ainsi deux chaînes de bits m et m' de même longueur seront chiffrées par $c = m \oplus z$ et $c' = m' \oplus z$. On retrouve donc des informations sur le clair : $c \oplus c' = m \oplus m'$. D'autre part une attaque à clair connue à partir de (m, c) permet de retrouver z et donc m' à partir de c' en cas de réutilisation de la même clef secrète.

Dans les chiffrements à flot modernes, on utilise une entrée auxiliaire, un vecteur d'initialisation public, IV, comme dans le mode OFB, afin de pouvoir produire plusieurs suites chiffrantes à partir de la même clef secrète. Au bout de n bits produits avec une même clef secrète, un nouvel IV (public) est choisi par Alice et Bob et l'état interne du chiffrement à flot est réinitialisé avec la clef secrète et ce nouvel IV. En effet la suite chiffrante produite est nécessairement périodique, il faut donc limiter le nombre de bits utilisé : l'état interne de l'automate produisant la suite chiffrante est fini de longueur ℓ : au plus 2^ℓ états différents sont donc possibles. Au bout de 2^ℓ itérations, on retombe donc un état déjà rencontré, et la suite chiffrante est donc périodique de période au plus 2^ℓ.

Pour résumer, on a

- Initialisation : choix d'un IV et $S_0 = h(sk, IV)$
- Pour $i = 1, \dots, n$
  - Mise à jour de l'état : $S_i = f(S_{i-1})$
  - Extraction du terme de suite chiffrante : $z_i = g(S_i)$
- Retour à l'initialisation.

5. Premières attaques

5.1. Tailles clef, IV, état interne

Pour éviter la recherche exhaustive, la taille de la clef secrète doit être au moins de 80 voire 128 bits. La taille de l'IV doit permettre d'éviter les collisions. Si on utilise deux fois le même IV la même suite chiffrante est générée. Par exemple le WEP utilisait RC4 avec un IV de 24 bits, soit 2^{24} IV possibles. Par le paradoxe des anniversaires, au bout de $2^{12} = 4096$ choix d'IV, on a une bonne probabilité que la même suite chiffrante soit utilisée.

L'état interne est également soumis à des attaques par recherche exhaustive, la suite chiffrante aux temps $t \geqslant t_0$ ne dépend que de l'état interne au temps t_0. On suppose que dans le cadre d'une attaque à clair connu, Oscar récupère ℓ bits de message clair $m_{t_0}, m_{t_0+1}, \dots, m_{t_0+\ell-1}$ ainsi que les chiffrés correspondants $c_{t_0}, c_{t_0+1}, \dots, c_{t_0+\ell-1}$. Il connaît donc ℓ bits de suite chiffrante $z_{t_0}, z_{t_0+1}, \dots, z_{t_0+\ell-1}$. Il peut donc faire une recherche exhaustive sur l'état S_{t_0} en vérifiant si les bits de suite chiffrante correspondent. Une fois le bon état interne trouvé, Oscar peut générer tous les bits $(z_i)_{i>t_0+\ell}$ et donc décrypter les chiffrés $(c_i)_{i>t_0+\ell}$.

L'état interne doit donc être de longueur au moins 80 bits. Cependant des **compromis temps mémoire** (attaque assez générale, introduite par Babbage et Golić en 1995 pour les chiffrements à flots) permettent d'améliorer cette attaque. On note P le nombre d'opérations de précalculs, M le nombre de bits de mémoire utilisés et T le nombre d'opérations de l'attaque active et ℓ la taille de l'état interne (c'est à dire que l'on suppose, pour simplifier, que le nombre de bits de suite chiffrante récupérés est le même que la taille de l'état interne). La recherche exhaustive précédente consiste à $P = 0, M = \mathcal{O}(\ell)$ et $T = \mathcal{O}(2^\ell)$. Le compromis temps mémoire consiste à faire des précalculs pour diminuer la complexité de l'attaque active.

On note $\phi : \{0,1\}^\ell \to \{0,1\}^\ell$ la fonction qui à un état interne au temps t_0 associe les ℓ bits de suite chiffrante produits à partir du temps t_0, $(z_{t_0}, z_{t_0+1}, \dots, z_{t_0+\ell-1})$. L'**attaque par dictionnaire** consiste à précalculer les 2^ℓ images $y_i = \phi(x_i)$ et à stocker les couples (x_i, y_i), triés par rapport aux y_i. La phase active consiste à chercher dans la table les ℓ bits de suite chiffrante récupérés par l'attaque à clair connu, pour retrouver l'état S_{t_0} (noter que ϕ n'est pas nécessairement bijective et que plusieurs états équivalents peuvent donner la même suite chiffrante). L'attaque par dictionnaire donne donc $P = \mathcal{O}(2^\ell), M = \mathcal{O}(\ell 2^\ell)$ et $T = \mathcal{O}(\ell)$ (recherche dans liste triée).

Le compromis temps mémoire consiste à équilibrer le coût des précalculs et de la phase active. On suppose ℓ pair. Les précalculs consistent maintenant à calculer les images de $2^{\ell/2}$ états internes pris au hasard. On suppose maintenant connus $2^{\ell/2} + \ell - 1$ bits consécutifs de suite chiffrante à partir de t_0, $(z_{t_0}, z_{t_0+1}, \dots, z_{t_0+2^{\ell/2}+\ell-2})$, soit $2^{\ell/2}$ images par ϕ. On recherche chacune de ces $2^{\ell/2}$ images dans la table. On utilise alors une variante du paradoxe des anniversaires.

Proposition II – 1. Soit L un ensemble à m éléments. On construit un ensemble L_1 par tirages uniformes indépendants sans remise de k_1 éléments de L. On construit également un ensemble L_2 par tirages uniformes indépendants avec remise de k_2 éléments de L. Soit p la probabilité que l'on ait une collision, c'est à dire que $L_1 \cap L_2$ soit non vide. On a

$$p \geqslant 1 - e^{-\frac{k_1 k_2}{m}}.$$

Dans notre cas, L_1 représente les images des états internes pris au hasard (on peut les supposer distinctes) et L_2 les suites de ℓ bits obtenues dans la suite chiffrante (on peut trouver plusieurs fois la même). On a $m = 2^\ell$ et $k_1 = k_2 = \sqrt{m}$. Donc la probabilité d'avoir une collision est supérieure à $1 - e^{-1} \approx 0,63$. En doublant la taille des listes : $k_1 = k_2 = 2\sqrt{m}$, la probabilité devient supérieure à $1 - e^{-4} \approx 0,98$, ainsi cette

probabilité tend extrêmement rapidement vers 1. Une collision donne alors un état S_t avec $t \geqslant t_0$ ce qui permet donc de retrouver toute la suite chiffrante à partir du temps t.

Au final, on a $P = \mathcal{O}(2^{\ell/2}), M = \mathcal{O}(\ell 2^{\ell/2}), T = \mathcal{O}(\ell 2^{\ell/2})$. Si on considère que l'on veut une sécurité de k bits (c'est à dire que les attaques doivent faire plus de 2^k opérations élémentaires) il faut donc prendre un état interne de ℓ bits avec $\ell \geqslant 2k$, donc en pratique $\ell \geqslant 160$.

5.2. Critères statistiques

Comme on l'a vu précédemment, la sécurité d'un système de chiffrement par flot synchrone additif contre une attaque à clair connu est équivalente à la sécurité de la suite chiffrante. Le système de chiffrement peut être vu comme un générateur pseudo aléatoire initialisé par la clef secrète (et l'IV éventuel). Pour qu'un système de chiffrement par flot soit considéré sûr on demande donc que la suite chiffrante ait de bonnes propriétés statistiques.

On veut par exemple que la connaissance de bits consécutifs ne permette pas de prévoir facilement la valeur du bit suivant avec probabilité sensiblement distincte de 1/2. Plus fort, on veut que la suite chiffrante ne puisse pas être distingué d'une suite aléatoire (notion de distingueur). Noter qu'un biais dans la suite chiffrante fait « fuir » de l'information sur le clair. Par exemple si $Pr(z_i = 0) = 1/2 + \epsilon$ alors si $c_i = m_i \oplus z_i$, $Pr(c_i = m_i) = 1/2 + \epsilon$.

Parmi les **critères statistiques**, les plus classiques sont les **trois critères de Golomb** (1982), donnés dans le cadre général d'une suite z périodique de période T :

1. Dans chaque période, les nombres de 0 et de 1 diffèrent au plus de 1 : $|\sum_{i=0}^{T-1}(-1)^{z_i}| \leqslant 1$.
2. Une série (de 0 ou de 1) est une succession de bits identiques, maximale (c'est à dire encadrée par des bits différents) Dans chaque période, soit S l'ensemble des séries, si $2^k \leqslant |S| < 2^{k+1}$, on trouve au moins $|S|/2$ séries de longueur 1, $|S|/4$ séries de longueur 2, ... , $|S|/2^k$ séries de longueur k, et pour chaque longueur, autant (à 1 près) de séries de 0 que de séries de 1. (implique le premier item)
3. La fonction d'auto corrélation $C(\tau)$ vaut T en 0 et une unique autre valeur pour $\tau \neq 0$, ce qui signifie que z est indépendante de ses translatées (même nombres d'égalités et de différences entre z et ses translatées) avec
$$C(\tau) := \sum_{i=0}^{T-1}(-1)^{z_i + z_{i+\tau}}.$$

Exercice : exemple du *Handbook of Applied Cryptography*, 5.30 page 181. Soit $T = 15$ et la suite de période,

$$0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1.$$

Vérifier que cette suite satisfait les critères de Golomb.

On va voir de la suite comment construire efficacement des suites satisfaisant ces critères avec des registres à décalage à rétroaction linéaire ou *Linear Feedback Shift Register* (LFSR). En particulier, cette suite est la suite périodique produite par un LFSR de longueur 4 et de polynôme de rétroaction, $X^4 + X + 1$.

Depuis ces critères bien d'autres ont été proposés, notamment les tests du *National Institute of Standards and Technology* (NIST), voir `http://csrc.nist.gov/groups/ST/toolkit/rng/index.html`.

Chapitre III

Registres à décalage à rétroaction linéaire

1. LFSR et suites à récurrence linéaire

Dans ce chapitre on va étudier un moyen pour produire des suites chiffrantes ayant de bonnes propriétés statistiques, les registres à décalage à rétroaction linéaire ou *Linear Feedback Shift Register*, (LFSR). Comme on le verra, pour construire un chiffrement par flot, on ne peut se contenter de LFSR, mais ceux ci forment un élément de base de nombreuses constructions de chiffrement à flot synchrone additif.

Définition III – 1. Un registre à décalage à rétroaction linéaire (LFSR) binaire de longueur ℓ est un automate composé d'un registre S à décalage de ℓ bits. On note $S^{(t)} = (S_0^{(t)}, \dots, S_{\ell-1}^{(t)})$ l'état du registre au temps $t \geqslant 0$. Soit $(z_0, z_1, \dots, z_{\ell-1})$ et $(c_1, c_2, \dots, c_\ell)$ deux chaînes de ℓ bits. L'état initial est $S^{(0)} = (z_0, \dots, z_{\ell-1})$.

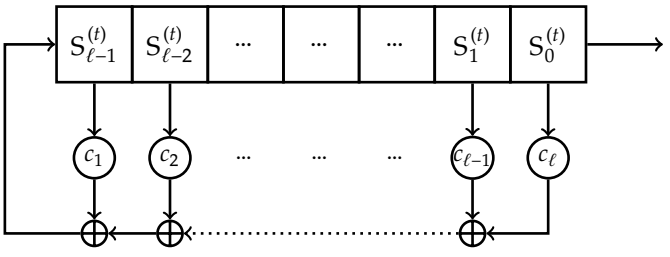
À l'instant t, on sort le bit d'indice 0, $S_0^{(t)}$, et on met à jour pour donner le registre $S^{(t+1)}$ de la façon suivante : les bits d'indices 1 à ℓ sont décalées :

$$S_i^{(t+1)} = S_{i+1}^{(t)}, \text{ pour } 0 \leqslant i \leqslant \ell - 2,$$

et le bit d'indice $\ell - 1$ est mis à jour par une fonction linéaire :

$$S_{\ell-1}^{(t+1)} = c_1 S_{\ell-1}^{(t)} \oplus c_2 S_{\ell-2}^{(t)} \oplus \cdots \oplus c_{\ell-1} S_1^{(t)} \oplus c_\ell S_0^{(t)},$$

où les calculs sont dans $\mathbf{F}_2$. On représente un LFSR ainsi :



Proposition III – 2. Si $z_0, \dots, z_{\ell-1}$ sont les bits de l'état initial d'un LFSR de longueur ℓ, on note pour tout $t \geqslant \ell$, $z_t = c_1 z_{t-1} \oplus c_2 z_{t-2} \oplus \cdots \oplus c_{\ell-1} z_{t-\ell+1} \oplus c_\ell z_{t-\ell}$. Alors la suite à récurrence linéaire d'ordre ℓ, $(z_t)_{t \geqslant 0}$ est la suite des bits de sortie du LFSR. De plus, pour tour $t \geqslant 0$, $S^{(t)} = (z_t, z_{t+1}, \dots, z_{t+\ell-1})$.

Inversement, toute suite binaire à récurrence linéaire d'ordre ℓ, peut être produite par un LFSR de longueur ℓ.

Exemple. On considère le LFSR de longueur $\ell = 3$ avec $(c_1, c_2, c_3) = (1, 0, 1)$, initialisé par $(z_0, z_1, z_2) = (1, 0, 0)$. Exercice : représenter l'état du LFSR pour $t = 0, \dots, 7$. Donner la suite de sortie, et sa période.

Proposition III – 3. La suite z produite par un LFSR de longueur ℓ est ultimement périodique, c'est-à-dire qu'il existe une pré période t_0 telle que la suite $(z_t)_{t \geqslant t_0}$ est périodique : $z_{t+\mathrm{T}} = z_t$ pour tout $t \geqslant t_0$. La période est $\mathrm{T} \leqslant 2^\ell - 1$. De plus si $c_\ell = 1$, il n'y a pas de pré période, la suite est périodique : $z_{t+\mathrm{T}} = z_t$ pour tout $t \geqslant 0$.

Démonstration. Un registre peut prendre au plus 2^ℓ états. S'il vaut $(0, \dots, 0)$ alors les registres successifs sont tous nuls et la suite de sortie est elle-même nulle à partir de ce rang, elle est donc ultimement périodique de période 1.

Si les registres ne sont jamais nuls alors parmi les 2^ℓ registres $\mathrm{S}^{(0)}, \mathrm{S}^{(1)}, \dots, \mathrm{S}^{(2^\ell-1)}$, au moins deux registres sont identiques. Supposons $\mathrm{S}^{(t_0)} = \mathrm{S}^{(t_0+\mathrm{T})}$, alors la suite des registres $\mathrm{S}^{(t_0)}, \mathrm{S}^{(t_0+1)}, \dots, \mathrm{S}^{(t_0+\mathrm{T}-1)}$ se répète indéfiniment. On a donc $z_{t+\mathrm{T}} = z_t$ pour tout $t \geq t_0$ avec $\mathrm{T} \leq 2^\ell - 1$.

On note

$$\mathrm{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ & & & & 0 \\ 0 & \cdots & & 0 & 1 \\ c_\ell & c_{\ell-1} & \cdots & c_2 & c_1 \end{pmatrix}$$

En considérant les registres comme des vecteurs colonnes, cette matrice permet d'exprimer la rétroaction linéaire :

$$\mathrm{S}^{(t+1)} = \mathrm{AS}^{(t)}.$$

Ainsi, on a $\mathrm{S}^{(t)} = \mathrm{A}^t\mathrm{S}^{(0)}$. En développant par rapport à la première colonne, on remarque que le déterminant de A est égal à c_ℓ. Si $c_\ell = 1$, la matrice A est inversible et le LFSR ne passe jamais par le registre nul si l'état initial $\mathrm{S}^{(0)}$ est non nul. La condition $\mathrm{S}^{(t_0)} = \mathrm{S}^{(t_0+\mathrm{T})}$ devient $\mathrm{A}^{t_0}\mathrm{S}^{(0)} = \mathrm{A}^{t_0+\mathrm{T}}\mathrm{S}^{(0)}$ mais comme A est inversible, on en déduit $\mathrm{S}^{(0)} = \mathrm{A}^{\mathrm{T}}\mathrm{S}^{(0)} = \mathrm{S}^{(\mathrm{T})}$ et la suite s est périodique. □

Si $c_\ell = 0$, en notant i_0 le plus grand indice inférieur à ℓ tel que $c_{i_0} \neq 0$, la suite z peut se voir comme une suite avec un préfixe $z_0, z_1, \dots, z_{\ell-i_0-1}$, suivie d'une suite engendrée par un LFSR de longueur $\ell' = i_0$, qui vérifie $c'_{\ell'} \neq 0$ en notant $c'_i = c_i$ pour $i \in \{1, \dots, i_0\}$.

On **supposera désormais que** $c_\ell = 1$ **et que la suite** z **est strictement périodique et non nulle**. Clairement, pour des applications cryptographiques, il est souhaitable que la suite z ait une période aussi longue que possible, c'est-à-dire, d'après ce qui précède, une période $\mathrm{T} = 2^\ell - 1$. Remarquons que, dans ce cas, comme dans l'exemple avec $\ell = 3$, les registres prennent tous les états possibles de $\mathbf{F}_2^\ell$ sauf le registre nul. En particulier, deux telles suites associées aux mêmes coefficients de récurrence mais pas au même état initial sont en fait décalées l'une de l'autre.

Définition III – 4. Si la suite z est produite par un LFSR de longueur ℓ (avec $c_\ell = 1$) a pour période $2^\ell - 1$, on dit que z est une m-**suite** ou m-**séquence**, ou encore qu'elle est **de longueur maximale**.

Une telle suite de longueur maximale sera souhaitable pour des applications cryptographique. De plus elle a de bons critères statistiques.

Proposition III – 5. Une m-suite vérifie les 3 critères de Golomb.

Démonstration. On a vu que pendant une période, le registre prend toutes les valeurs possibles de $\mathbf{F}_2^\ell$ sauf le registre nul. Le premier élément de chaque registre prend donc $2^{\ell-1}$ fois la valeur 1 et $2^{\ell-1}-1$ la valeur 0. Ce premier élément étant à chaque tour le bit de sortie, on en déduit que dans une période, dans la m-suite produite, le nombre de 1 et de 0 diffèrent de 1, on satisfait donc le premier critère de Golomb. Le deuxième critère compte le nombre de séries de 0 et de 1. On montre de manière similaire qu'il est aussi vérifié. Le troisième critère concerne l'auto corrélation de la suite. Soit $\tau \neq 0$, la suite $(z_t \oplus z_{t+\tau})_{t\in\mathbf{N}}$ peut être produite par le même LFSR que la suite z. On peut donc appliquer le premier critère ce qui signifie que $C(\tau) := \sum_{t=0}^{2^\ell-2}(-1)^{z_t\oplus z_{t+\tau}} = -1$. □

2. LFSR et polynômes de $\mathbf{F}_2[X]$

Définition III – 6. Soit un LFSR de longueur ℓ de coefficients de rétroaction $(c_1, c_2, \dots, c_\ell)$. Son polynôme de rétroaction (parfois appelé polynôme de connexion) est le polynôme de $\mathbf{F}_2[X]$

$$f(X) = 1 \oplus c_1 X \oplus c_2 X_2 \oplus \cdots \oplus c_\ell X^\ell.$$

Dans la suite on notera donc $c_0 = 1$.

On associe à la suite $z = (z_t)_{t\geqslant 0}$ la série génératrice

$$Z(X) = \sum_{t\geqslant 0} z_t X^t.$$

Proposition III – 7. Une suite z strictement périodique est produite par un LFSR de longueur ℓ dont le polynôme de rétroaction est $f(X) = 1 \oplus c_1 X \oplus c_2 X_2 \oplus \cdots \oplus c_\ell X^\ell$ si et seulement si son développement en série formelle vérifie

$$Z(X) = g(X)/f(X),$$

où g est un polynôme de $\mathbf{F}_2[X]$ tel que $deg(g) < deg(f)$. En outre, le polynôme g est entièrement déterminé par l'état initial du registre :

$$g(X) = \sum_{i=0}^{\ell-1} X^i \sum_{j=0}^{i} c_j z_{i-j}.$$

Démonstration. Supposons z produite par un LFSR de polynôme de rétroaction $c_0 \oplus c_1 X \oplus \cdots \oplus c_\ell X^\ell$ avec $c_0 = c_\ell = 1$. On pose

$$g(X) = Z(X)f(X) = (z_0 \oplus z_1 X \oplus \cdots)(c_0 \oplus c_1 X \oplus \cdots \oplus c_\ell X^\ell).$$

On vérifie que g est bien un polynôme. On note g_i le coefficient de g de degré i. On a $g_0 = z_0 c_0$, $g_1 = z_0 c_1 \oplus z_1 c_0$ et pour tout i avec $0 \leqslant i < \ell$, $g_i = \sum_{j=0}^{i} c_j z_{i-j}$. Ensuite, pour tout $i \geqslant 0$,

$$g_{\ell+i} = \sum_{j=0}^{\ell} c_j z_{\ell+i-j} = c_0 z_{\ell+i} \oplus c_1 z_{\ell+i-1} \oplus \cdots \oplus c_\ell z_i = 0,$$

car on retrouve l'équation de rétroaction. Donc g est bien un polynôme de degré inférieur à ℓ. Réciproquement, si $Z(X) = g(X)/f(X)$, alors la suite z satisfait une récurrence linéaire d'ordre ℓ, donnée par la formule précédente. □

On peut voit un LFSR comme un automate calculant la division suivant les puissances croissantes de deux polynômes à coefficients dans $\mathbf{F}_2$.

Exemple. On revient sur l'exemple avec $\ell = 3$, $c_1 = c_3 = 1$, et l'initialisation $S^{(0)} = (1, 0, 0)$. Exercice : retrouver la suite chiffrante par calcul de $Z(X)$.

Afin d'obtenir une forme canonique de la série génératrice de Z, on définit le polynôme de rétroaction minimal : c'est un diviseur de $f(X)$, qui de plus est le polynôme de plus bas degré parmi les polynômes de rétroaction de tous les LFSR possibles qui génèrent la suite z.

Définition III – 8. Soit un LFSR de longueur ℓ d'initialisation non nulle et z sa suite de sortie supposée strictement périodique. Son **polynôme de rétroaction minimal** est l'unique polynôme unitaire f de $\mathbf{F}_2[X]$ tel qu'il existe $g \in \mathbf{F}_2[X]$, avec $deg(g) < deg(f)$ et $pgcd(f, g) = 1$, vérifiant $Z(X) = g(X)/f(X)$. La complexité linéaire du LFSR produisant la suite z, notée $\Lambda(z)$, est alors égale au degré de f : c'est la longueur du plus petit LFSR permettant d'engendrer z.

Sur l'exemple, on a $pgcd(1 \oplus X, 1 \oplus X \oplus X^3) = 1$ donc $\Lambda(z) = 3$. Plus généralement, si le polynôme de rétroaction est irréductible de degré ℓ, on a bien $\Lambda(z) = \ell$.

Ces liens entre suites produites par un LFSR et polynôme de $\mathbf{F}_2[X]$ vont nous permettre de caractériser les LFSR produisant des m-suites.

Proposition III – 9. Soit un LFSR de longueur ℓ d'initialisation non nulle et de polynôme de rétroaction f primitif de degré ℓ. Alors la suite produite z est de période maximale $2^\ell - 1$.

Démonstration. Soit A la matrice de rétroaction

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \\ c_\ell & c_{\ell-1} & \cdots & c_2 & c_1 \end{pmatrix}$$

Soit $T \leqslant 2^\ell - 1$ la période de z (z n'a pas de pré période comme $c_\ell = 1$). On a $S^{(T)} = S^{(0)}$ et on a vu que $S^{(T)} = A^T S^{(0)}$. Le vecteur non nul $S^{(0)}$ est donc un vecteur propre de A^T associé à la valeur propre 1. Le polynôme caractéristique de A est $\tilde{f}(X) = det(X \times I_\ell - A) = X^\ell \oplus c_1 X^{\ell-1} \oplus \cdots \oplus c_{\ell-1} X \oplus c_\ell$ (on le voit facilement par récurrence, en développant par rapport à la première colonne). Ce polynôme est le polynôme réciproque de f : $\tilde{f}(X) = X^L f(1/X)$, et A est en fait la (transposée de la) matrice compagnon de $\tilde{f}$. Comme f est primitif, il a ses ℓ racines distinctes dans $\mathbf{F}_{2^\ell}$: $\alpha, \alpha^2, \ldots, \alpha^{2^{\ell-1}}$ et elles sont toutes du même ordre $2^\ell - 1$. Les racines de $\tilde{f}$ sont les inverses de celles de f, elles sont donc aussi toutes distinctes et d'ordre $2^\ell - 1$. On les note $\beta_0, \beta_1, \ldots, \beta_{\ell-1}$. La matrice A est donc diagonalisable dans $\mathbf{F}_{2^\ell}$: il existe une matrice inversible P telle que $P^{-1}AP = Diag(\beta_0, \ldots, \beta_{\ell-1})$. On a alors $(P^{-1}AP)^T = P^{-1}A^TP = Diag(\beta_0^T, \ldots, \beta_{\ell-1}^T)$. Comme A^T a la valeur propre 1 il existe j avec $0 \leqslant j \leqslant \ell - 1$ tel que $\beta_j^T = 1$. Donc $(2^\ell - 1) \mid T$ car tous les β_i sont d'ordre $2^\ell - 1$. Finalement, $(2^\ell - 1) \leqslant T \leqslant (2^\ell - 1)$. On conclut donc que $T = 2^\ell - 1$. □

On peut en fait montrer la réciproque : z est une m-suite implique que le polynôme de rétroaction minimal est primitif. Sur l'exemple vu précédemment, le polynôme $X^3 \oplus X \oplus 1$ est primitif et on a une m-suite de période $2^3 - 1 = 7$.

Un LFSR de polynôme de rétroaction primitif est donc un peu candidat pour construire un chiffrement à flot : on a une implantation logicielle et matérielle très rapide, de bonnes propriétés statistiques et une grande période possible (un LFSR de ℓ bits peut produire une suite de $2^\ell - 1$ bits). Cependant, on ne peut pas les utiliser directement : ℓ bits consécutifs de la suite de sortie fournissent directement l'état interne. Une solution pourrait être de garder comme clef secrète la fonction de rétroaction, donc la valeur de $(c_1, \ldots, c_\ell)$: même si on récupère l'état interne, on ne pourrait « dérouler » ou « rembobiner » le LFSR. Cependant on va voir qu'on peut retrouver très efficacement ces coefficients.

3. L'algorithme de Berlekamp-Massey

Proposition III – 10. Soit z une suite produite par un LFSR de longueur ℓ et de polynôme de rétroaction f irréductible de degré ℓ. Si on connaît 2ℓ bits consécutifs de z alors on peut retrouver les coefficients de rétroaction en inversant un système linéaire $\ell \times \ell$.

Démonstration. Comme z est une suite produite par un LFSR de longueur ℓ et de polynôme de rétroaction f irréductible de degré ℓ, on a $\Lambda(z) = \ell$. On suppose que l'on connaît 2ℓ bits à partir du temps t : $z_t, z_{t+1}, z_{t+2}, \dots, z_{t+2\ell-1}$. On construit le système linéaire suivant :

$$\begin{pmatrix} z_t & z_{t+1} & z_{t+2} & \cdots & z_{t+\ell-1} \\ z_{t+1} & z_{t+2} & \cdots & \cdots & z_{t+\ell} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ z_{t+\ell-2} & \cdots & \cdots & \cdots & z_{t+2\ell-3} \\ z_{t+\ell-1} & z_{t+\ell} & \cdots & \cdots & z_{t+2\ell-2} \end{pmatrix} \begin{pmatrix} c_\ell \\ c_{\ell-1} \\ \vdots \\ c_2 \\ c_1 \end{pmatrix} = \begin{pmatrix} z_{t+\ell} \\ z_{t+\ell+1} \\ \vdots \\ z_{t+2\ell-2} \\ z_{t+2\ell-1} \end{pmatrix}.$$

Les lignes de la matrice sont les registres $S^{(t)}, S^{(t+1)}, \dots, S^{(t+\ell-1)}$. Si le système a une solution, les coefficients de rétroaction seront donc bien solutions. On montre dans la suite que les lignes de la matrice sont indépendantes, ce qui prouve que le système a une unique solution.

Si elles ne l'étaient pas, il existerait une combinaison linéaire, $a_0 S^{(t)} \oplus a_1 S^{(t+1)} \oplus \cdots \oplus a_{\ell-1} S^{(t+\ell-1)} = 0$. Soit $k \in \mathbf{N}$, avec $0 < k < \ell$, le plus grand indice avec $a_k \neq 0$. On peut alors exprimer $S^{(t+k)}$ en fonction des k registres précédents : ceci s'étend en une récurrence linéaire sur la suite z avec $k < \ell$, ce qui contredit le fait que $\Lambda(z) = \ell$.

□

Cette proposition montre que si l'on connaît la complexité linéaire ℓ d'une suite z produite par un LFSR, alors on peut retrouver ce LFSR avec 2ℓ bits consécutifs.

Soit n un entier avec $n \geqslant 2\ell$. On suppose connaître n bits consécutifs de z, que l'on suppose strictement périodique. Sans perte de généralité, on note ces bits $z_0, z_1, \dots, z_{n-1}$. Dans la suite, on étudie un algorithme qui va permettre de trouver le plus petit LFSR qui a permis de produire ces bits sans connaître *a priori* la valeur de la complexité linéaire ℓ.

Trouver le plus petit LFSR revient à chercher un polynôme de rétroaction f de degré ℓ et un polynôme g avec $deg\, g < deg\, f$ avec $pgcd(g, f) = 1$, tel que $z_0 \oplus z_1 X \oplus \cdots \oplus z_{n-1} X^{n-1} \equiv Z(X) \pmod{X^n} \equiv g(X)/f(X)$. C'est à dire que l'on veut trouver à partir de $\tilde{Z} := Z(X) \mod X^n$ et X^n des polynômes f, g et h tel que

$$f(X)\tilde{Z}(X) \oplus h(X)X^n = g(X),$$

avec les contraintes précédentes sur les degrés. Ceci peut se faire par l'algorithme d'Euclide étendu appliqué à X^n et $\tilde{Z}(X)$. On désigne par $u_i(X)$, $v_i(X)$ et $r_i(X)$ les résultats intermédiaires tels qu'à chaque tour, $u_i(X)X^n \oplus v_i(X)\tilde{Z}(X) = r_i(X)$. La suite des degrés des $r_i(X)$ est strictement décroissante. On peut montrer que pour l'indice j tel que $deg\, r_j < \lfloor n/2 \rfloor$ alors le polynôme recherché, $f(X)$ est égal à $v_j(X)$.

L'algorithme proposé par Massey en 1969 (initialement introduit dans le contexte de décodage des codes BCH par Berlekamp), effectue plus ou moins les mêmes opérations que cette variante de l'algorithme d'Euclide étendu. Contrairement à Euclide étendu, les résultats intermédiaires de l'algorithme de Berlekamp-Massey vont donner directement des informations sur la suite étudiée. Étant donné une suite $z_0, \dots, z_{n-1}$, cet algorithme retourne pour k allant de 1 à n le LFSR de longueur minimale ℓ_k et le polynôme de rétroaction associé f_k avec $deg\, f_k \leqslant \ell_k$ produisant les suites tronquées $z_0, \dots, z_{k-1}$. La suite des (ℓ_k, f_k) est appelée profil de complexité de z. Pour une suite aléatoire, on a $\ell_k \approx \frac{k}{2}$.

Supposons avoir trouvé au rang k, des polynômes f_k et g_k tels que $f_k(X)Z(X) \equiv g_k(X) \pmod{X^k}$ et ℓ_k la longueur du LFSR minimal. Au rang $k+1$, soit ces polynômes conviennent toujours, c'est à dire $f_k(X)Z(X) \equiv g_k(X) \pmod{X^{k+1}}$ et on peut poser $f_{k+1} = f_k$ et $\ell_{k+1} = \ell_k$. Soit,

$$f_k(X)Z(X) \equiv g_k(X) \oplus X^k \pmod{X^{k+1}}.$$

Dans ce cas, étant donné un autre rang $m < k$ tel que

$$f_m(X)Z(X) \equiv g_m(X) \oplus X^m \pmod{X^{m+1}},$$

en multipliant cette équation par X^{k-m}, on obtient :

$$f_m(X)Z(X)X^{k-m} \equiv g_m(X)X^{k-m} \oplus X^k \quad (mod\ X^{k+1}),$$

donc en combinant avec l'équation au rang $k+1$, on a

$$[f_k(X) \oplus X^{k-m} f_m(x)]Z(X) \equiv g_k(X) \oplus X^{k-m} g_m(X) \quad (mod\ X^{k+1}).$$

On pose donc $f_{k+1}(X) = f_k(X) \oplus X^{k-m} f_m(X)$ et $g_{k+1}(X) = g_k(X) \oplus X^{k-m} g_m(X)$.

On montre de plus que si m est le plus grand entier tel que $m < k$ et tel que $\ell_m < \ell_k$, f_{k+1} est le polynôme de rétroaction d'un LFSR de longueur minimale avec $\ell_{k+1} = max(\ell_k, k+1-\ell_k)$.

Au final, on obtient l'algorithme suivant : f joue le rôle de f_k et $\tilde{f}$ celui de f_m. On initialise ces deux polynômes aux indices respectifs 0 et −1 par $f = \tilde{f} = 1$. En effet, ce polynôme constant 1 engendre la suite nulle : $Z(X) = 0 = g(X)/f(X)$ avec $g(X) = 0$ et $f(X) = 1$. Avec ces choix, pour le premier bit de suite non nul, disons z_k, f deviendra bien le polynôme $1 \oplus X^{k+1}$ comme attendu.

Algorithme de Berlekamp-Massey

Entrée : $(z_0, z_1, \dots, z_{n-1})$
Sortie : la suite des (ℓ_k, f_k) pour $k = 1, \dots, n$

$f \leftarrow 1, \ell \leftarrow 0, m \leftarrow -1, \tilde{f} \leftarrow 1$
Pour k **de** 0 **à** $n-1$ **faire**
  Noter $c_1, c_2, \dots, c_u$ tels que $f = 1 \oplus c_1 X \oplus \cdots \oplus c_u X^u$ (on peut avoir $\ell > u$)
  $d \leftarrow z_k \oplus \sum_{i=1}^{u} c_i z_{k-i}$
  Si $d = 1$ **alors**
    $h \leftarrow f, f \leftarrow f \oplus \tilde{f} X^{k-m}$
    Si $\ell \leqslant \frac{k}{2}$ **alors**
      $\ell \leftarrow k+1-\ell, m \leftarrow k, \tilde{f} \leftarrow h$

Théorème III − 11. Soit $(z_t)_{t \geqslant 0}$ une suite produite par un LFSR de complexité linéaire $\Lambda(z) = \ell$. À partir de 2ℓ bits consécutifs de z, l'algorithme de Berlekamp-Massey détermine en temps quadratique l'unique LFSR de longueur ℓ qui produit z.

Exemple. Exercice : appliquer l'algorithme de Berlekamp-Massey à la suite de 8 bits $z = 1, 1, 1, 0, 1, 0, 1, 1$.

En **conclusion**, si on utilise un LFSR dont la rétroaction (primitive) est secrète pour faire un chiffrement à flot additif avec une taille de registre ℓ, seulement 2ℓ bits de suite chiffrante permettent de récupérer la rétroaction lors d'une attaque à clair connu. C'est pratiquement inutilisable. De manière générale, la complexité linaire Λ de la suite chiffrante d'une chiffrement à flot additif doit être élevé pour se mettre à l'abri de l'algorithme de Berlekamp-Massey. On va voir dans le chapitre suivant comment augmenter cette complexité linéaire tout en ayant des tailles de registres utilisables en combinant ou en filtrant des LFSR.

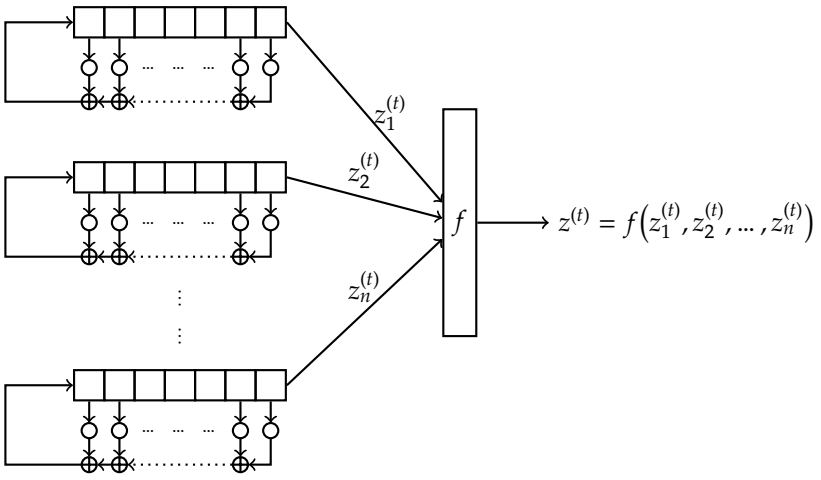
Chapitre IV

LFSR combinés et filtrés

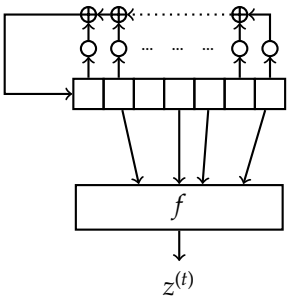
1. Constructions

On va étudier deux types de constructions à partir de LFSR visant à obtenir une suite chiffrante z de complexité linaire élevée.

Tout d'abord la combinaison de n LFSR distincts par une fonction booléenne :



Puis le filtrage de l'état interne d'un unique LFSR par une fonction booléenne :



Avant cela on fait quelques rappels rapides sur les fonctions booléennes.

2. Fonctions booléennes

Définition IV – 1. On appelle **fonction booléenne** à n variables une application $f : \mathbf{F}_2^n \to \mathbf{F}_2$. Une fonction booléenne **vectorielle** à n variables et m composantes et une application de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. C'est la juxtaposition de m fonctions booléennes : $(f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n))$. Ces fonctions vectorielles correspondent aux *S-box* utilisées en cryptographie symétrique.

Définition IV – 2. On appelle **support** d'une fonction booléenne f, notée, Supp(f), l'ensemble des valeurs $x \in \mathbf{F}_2^n$ telle que $f(x)$ est non nul. Le **poids** de f, noté $w(f)$ est le cardinal de son support. Si v_f est le vecteur de $\mathbf{F}_2^{2^n}$ contenant toutes les images de f :

$$v_f = \Big(f(0,0,\dots,0,0), f(0,0,\dots,0,1), \dots, f(1,1,\dots,1,1)\Big),$$

alors $w(f)$ est le poids de Hamming de v_f. On définit également la **distance** entre deux fonctions booléennes f et g, $d(f,g) = w(f+g) = Card\{u \in \mathbf{F}_2^n, f(u) \neq g(u)\}$. On dit que f est **équilibrée** si $w(f) = 2^{n-1}$, dans ce cas

$$Card\{x \in \mathbf{F}_2^n, f(x) = 0\} = Card\{x \in \mathbf{F}_2^n, f(x) = 1\}.$$

Remarque.

- Si f est utilisée pour filtrer ou combiner des LFSR, cela assurera que la suite chiffrante générée est équilibrée.
- Il y a 2^{2^n} fonctions booléennes à n variables. Si $n = 6$, cela fait déjà 2^{64} fonctions. Il n'est donc pas envisageable de trouver par recherche exhaustive une fonction booléenne ayant de « bonnes propriétés ».

Exemple. On considère la fonction de $\mathbf{F}_2^3 \to \mathbf{F}_2$ donnée par sa table de vérité :

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

On a $w(f) = 4$, Supp$(f) = \{(0,0,1),(1,0,1),(1,1,0),(1,1,1)\}$, et f est équilibrée. À partir de l'expression de ce support, on peut reconstituer une expression algébrique pour f, on a

$$\begin{aligned} f(x_1,x_2,x_3) &= (1+x_1)(1+x_2)x_3 \oplus x_1(1\oplus x_2)x_3 \oplus x_1x_2(1\oplus x_3) \oplus x_1x_2x_3 \\ &= (1\oplus x_2)x_3(1\oplus x_1\oplus x_1) \oplus x_1x_2(1\oplus x_3\oplus x_3) \\ &= x_3 \oplus x_2x_3 \oplus x_1x_2 \end{aligned}$$

En fait, toute fonction booléenne peut s'écrire comme précédemment sous forme d'un polynôme. De plus, on remarque qu'évaluer x^2 ou x^k avec $k \geq 1$ est équivalent à évaluer x dans $\mathbf{F}_2$. Ainsi dans l'exemple précédent, $f(x_1,x_2,x_3) = x_3 \oplus x_2x_3 \oplus x_1x_2 = x_3^2 \oplus x_2^5x_3 \oplus x_1^7x_2^2$.

Définition IV – 3. Soit une fonction booléenne f à n variables. Il existe une unique classe de polynômes

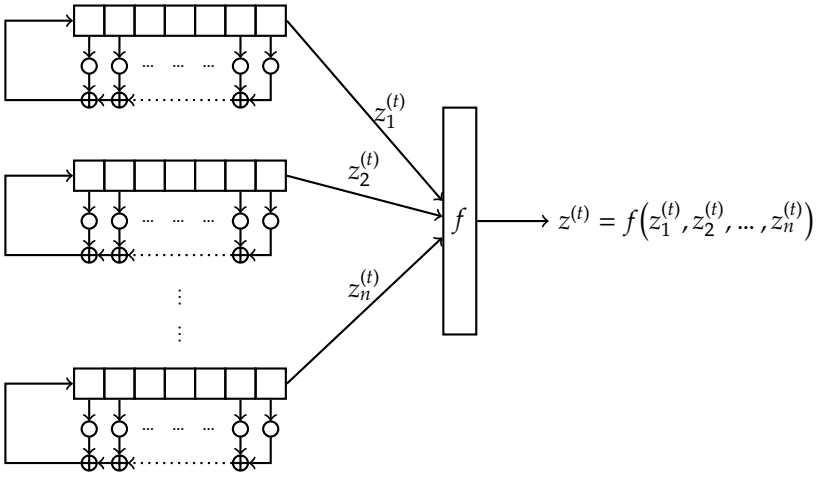
$$\Pi \in \frac{\mathbf{F}_2[x_1, \dots, x_n]}{(x_1^2 - x_1, \dots, x_n^2 - x_n)},$$

tel que $f(u_1, \dots, u_n) = \mathrm{P}(u_1, \dots, u_n)$, pour tout $(u_1, \dots, u_n) \in \mathbf{F}_2^n$ et tout $\mathrm{P} \in \Pi$. La **forme algébrique normale** de f est le représentant de $\mathrm{P} \in \Pi$ avec $deg_{x_i}(\mathrm{P}) \leq 1$ pour tout $i \in \{1, \dots, n\}$. Le **degré** de f, noté $deg(f)$, est le degré de ce polynôme, soit le nombre maximal de termes dans les monômes de P.

Dans l'exemple précédent, $deg(f) = 2$, f est dite **quadratique**. Les fonctions de degrés 1 sont dites **affines**, par exemple $1 \oplus x_1 \oplus x_3$. S'il n'y a pas de terme constant on a une fonction **linéaire**, soit une forme linéaire de $\mathbf{F}_2^n \to \mathbf{F}_2$. Dans ce cas, il existe $a = (a_1, \dots, a_n) \in \mathbf{F}_2^n$ tel que f s'écrive $f(x) = \langle a, x \rangle = a_1 x_1 \oplus \cdots \oplus a_n x_n$.

3. LFSR combinés et attaques par corrélation

On prend une fonction booléenne f à n variables équilibrée pour garantir que la suite chiffrante z satisfait le premier critère de Golomb. La clef secrète est l'initialisation des n LFSR. À chaque tour, les LFSR sont mis à jour et le bit de sortie $z^{(t)}$ est calculé suivant la fonction de combinaison f.



On va voir que la suite chiffrante $(z^{(t)})_{t \in \mathbf{N}}$ est toujours une suite récurrente linéaire. On veut assurer que sa complexité linéaire est élevée pour résister à l'algorithme de Berlekamp-Massey. Pour cela, on évalue la complexité linéaire de sommes et produits de suites à récurrence linéaire.

Proposition IV – 4. Si $z = (z_t)_{t \in \mathbf{N}}$ et $z' = (z'_t)_{t \in \mathbf{N}}$ sont deux suites produites par des LFSR de polynômes de rétroactions minimaux f et f' tels que $pgcd(f, f') = 1$ alors la somme $z \oplus z' = (z_t \oplus z'_t)_{t \in \mathbf{N}}$ est une suite à récurrence linéaire et

$$\Lambda(z \oplus z') = \Lambda(z) + \Lambda(z').$$

Démonstration. Il existe deux polynômes g et g' avec $deg(g) < deg(f)$, $deg(g') < deg(f')$ et $pgcd(g, f) = (g', f') = 1$. Soit Z et Z′ les séries associées aux suites z et z', on a

$$(\mathrm{Z} \oplus \mathrm{Z}')(\mathrm{X}) = \frac{g(\mathrm{X})}{f(\mathrm{X})} \oplus \frac{g'(\mathrm{X})}{f'(\mathrm{X})} = \frac{g(\mathrm{X})f'(\mathrm{X}) \oplus g'(\mathrm{X})f(\mathrm{X})}{f(\mathrm{X})f'(\mathrm{X})}.$$

On peut montrer que cette dernière fraction est réduite, d'où le résultat. □

On a une proposition similaire pour le produit. Au final, on a le théorème (admis) suivant.

Théorème IV – 5. On considère la suite $(z^{(t)})_{t\in\mathbf{N}}$ produite par la combinaison par une fonction booléenne f à n variables de n LFSR de polynômes de rétroactions primitifs de longueurs $\ell_1, \ell_2, \dots, \ell_n$ deux à deux distinctes et supérieures ou égales à 2, alors

$$\Lambda(z) = f(\ell_1, \ell_2, \dots, \ell_n),$$

où f est évaluée dans $\mathbf{Z}$.

Une fonction f de degré élevé permettra d'obtenir une suite de complexité linéaire élevée.

Exemple. Le **générateur de Geffe** (1973) est une construction qui utilise 3 LFSR de polynômes de rétroaction primitifs et de longueurs ℓ_1, ℓ_2, ℓ_3 avec $\ell_1 \neq \ell_2$, $\ell_1 \neq \ell_3$, $\ell_2 \neq \ell_3$. La fonction de combinaison est la fonction étudiée dans les exemples précédents, soit $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$. La complexité linéaire de la suite chiffrante z est donc $\Lambda(z) = \ell_1\ell_2 + \ell_2\ell_3 + \ell_3$ et cette suite est équilibrée.

La recherche exhaustive de la clef secrète d'une combinaison de LFSR, consiste à tester toutes les initialisations possibles. Dans le cas général d'une combinaison de n LFSR, cela a pour complexité $\prod_{i=1}^{n} 2^{L_i}$. L'**attaque par corrélation**, proposée par Siegenthaler en 1985, permet de diminuer cette complexité en trouvant les initialisations des LFSR de manière indépendante, à condition qu'il y ait une corrélation entre la suite chiffrante z et les suites produites par les LFSR, z_i.

Exemple. On détaille l'attaque par corrélation dans le cas du générateur de Geffe. À l'instant t, on considère $Z_1 = z_1^{(t)}$, $Z_2 = z_2^{(t)}$ et $Z_3 = z_3^{(t)}$ comme des variables aléatoires binaires indépendantes, prenant les valeurs 0 et 1 avec équiprobabilité. On note de même $Z = f(Z_1, Z_2, Z_3)$. On calcule les probabilités que $Z_i = Z$ pour $i = 1, 2, 3$. On a $Z = Z_1Z_2 \oplus Z_2Z_3 \oplus Z_3 = Z_1Z_2 \oplus Z_3(Z_2 \oplus 1)$. On trouve par la formule des probabilités totales :

$$\begin{aligned} Pr[Z_1 = Z] &= Pr[Z_1 = Z|Z_2 = 0] \times Pr[Z_2 = 0] + Pr[Z_1 = Z|Z_2 = 1] \times Pr[Z_2 = 1] \\ &= Pr[Z_1 = Z_3] \times \frac{1}{2} + 1 \times \frac{1}{2} = \frac{1}{4} + \frac{1}{2} = \frac{3}{4}. \end{aligned}$$

On a bien $Pr[Z_1 = Z_3] = \frac{1}{2}$ car $Pr[Z_1 = Z_3] = Pr[Z_1 = 1 \wedge Z_3 = 1] + Pr[Z_1 = 0 \wedge Z_3 = 0] = Pr[Z_1 = 1]\,Pr[Z_3 = 1] + Pr[Z_1 = 0]\,Pr[Z_3 = 0] = \frac{1}{2}$.

On trouve de même que $Pr[Z_2 = Z] = \frac{1}{2}$ et que $Pr[Z_3 = Z] = \frac{3}{4}$. Ces probabilités se retrouvent en regardant la table de vérité de f.

On a donc l'égalité $Z_2 = Z$ avec probabilité $\frac{1}{2}$ ce qui serait le cas si Z_2 et Z étaient des variables aléatoires indépendantes. Il n'y a pas de corrélation entre ces deux variables. Par contre, pour Z_1 et Z_3 ces probabilités sont $\frac{3}{4}$ il y a une corrélation que l'on peut exploiter pour retrouver indépendamment l'initialisation des LFSR 1 et 2.

On suppose récupéré dans une attaque à clair connu ℓ bits consécutifs de suite chiffrante, de $z^{(t_0)}$ à $z^{(t_0+\ell-1)}$. Pour retrouver l'initialisation du premier LFSR, on teste les 2^{ℓ_1} initialisations possibles. Pour chaque initialisation,

- On calcule $z_1^{(t_0)}, z_1^{(t_0+1)}, \dots, z_1^{(t_0+\ell-1)}$;
- On compare bit à bit avec $z^{(t_0)}, z^{(t_0+1)}, \dots, z^{(t_0+\ell-1)}$;
- Si on a près de $\frac{3}{4}$ d'égalité, l'initialisation est sûrement la bonne, sinon ($\approx \frac{1}{2}$ d'égalité) on continue.

On retrouve de même le 3e LFSR, puis le deuxième par recherche exhaustive. La complexité totale de l'attaque est donc de $2^{\ell_1} + 2^{\ell_2} + 2^{\ell_3}$ opérations au lieu de $2^{\ell_1+\ell_2+\ell_3}$ par recherche exhaustive.

Dans le cas de n LFSR, on calcule $p_i = Pr[Z_i = Z]$ pour $i = 1, \dots, n$. Dès qu'un p_i s'écarte assez de $\frac{1}{2}$, on peut faire une attaque par corrélation, la complexité devient

$$\sum_{i, p_i \neq \frac{1}{2}} 2^{\ell_i} + \prod_{i, p_i = \frac{1}{2}} 2^{\ell_i}.$$

Cette attaque peut-être généralisée en regardant la corrélation de la sortie avec une somme de k sortie de LFSR. Pour s'en protéger, il faut prendre des fonctions de combinaisons garantissant qu'il n'y ait pas de corrélation.

Définition IV – 6. On dit qu'une fonction booléenne f à n variables est **non corrélée** à l'ordre k si, pour des variables aléatoires binaires équilibrées et indépendantes $X_1, \dots, X_n$, la variable aléatoire $f(X_1, \dots, X_n)$ est indépendante des variables $\sum_{i \in I} X_i$, où I est un sous-ensemble de $\{1, \dots, n\}$ de cardinal au plus k. Cela revient à dire que $d(f, \sum_{i \in I} x_i) = 2^{n-1}$. On dit que f est k-**résiliente** si f est non corrélée à l'ordre k, et équilibrée.

On peut montrer que si f est k-résiliente alors son degré est tel que $deg(f) < n - k$ si $k < n - 1$. Pour choisir f il faut donc faire un compromis entre augmenter k pour éviter les attaques par corrélation et avoir un degré suffisamment élevé pour augmenter la complexité linéaire et éviter les attaques par l'algorithme de Berlekamp-Massey.

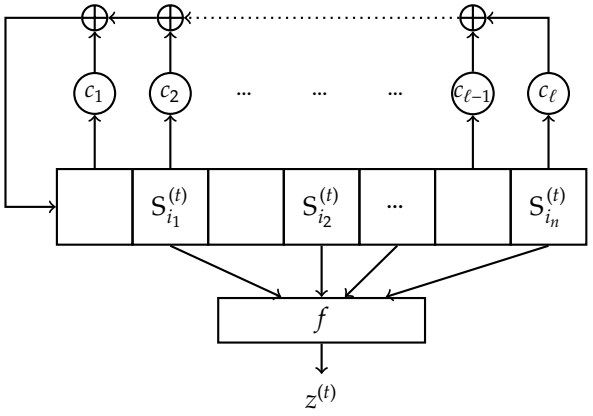
On peut voir la suite chiffrante comme une version bruitée de la suite produite par un LFSR. En utilisant un algorithme de décodage adapté, Meier et Staffelbach en 1988 ont amélioré l'attaque par corrélation donnant lieu à une **attaque par corrélation rapide**.

4. LFSR filtrés et attaques algébriques

C'est une autre construction simple pour obtenir une complexité linéaire élevée en conservant les bonnes propriétés statistiques des LFSR.

On utilise un seul LFSR de longueur ℓ. La clef secrète est l'initialisation du LFSR. Le registre du LFSR est filtré par une fonction booléenne f à n variables avec $1 < n \leqslant \ell$. On choisit n cases du registres, notées $\ell - 1 \geqslant i_1 \geqslant i_2 \geqslant \cdots \geqslant i_n \geqslant 0$, sur lesquelles on applique la fonction booléenne. À chaque tour, le LFSR est mis à jour. À l'instant t, le bit de sortie $z^{(t)}$ est calculé en filtrant l'état interne $S^{(t)}$:

$$z^{(t)} = f(S_{i_1}^{(t)}, S_{i_2}^{(t)}, \dots, S_{i_n}^{(t)}).$$



Cette construction est équivalente à une combinaison de n LFSR utilisant tous le même polynôme de rétroaction et étant initialisés respectivement par $S^{(i_1)}, S^{(i_2)}, \dots, S^{(i_n)}$. Comme on utilise plusieurs fois le même LFSR, on ne peut pas appliquer le résultat sur la complexité linéaire des LFSR combinés. On a le résultat suivant.

Proposition IV – 7. La complexité linéaire d'une suite chiffrante $z^{(t)}$ produite par un LFSR de longueur ℓ filtré par une fonction booléenne de degré d satisfait

$$\Lambda(z) \leqslant \sum_{r=1}^{d} \binom{\ell}{r}.$$

Si ℓ est premier assez grand alors $\Lambda(z) \approx \binom{\ell}{d}$ pour la plupart des fonctions booléennes de degré d.

Il existe des attaques par corrélation adaptées aux LFSR filtrés. D'autres attaques, dites par inversion, exploitent les distances $|i_j - i_{j'}|$ pour $j \neq j'$, c'est à dire les distances entre les entrées de la fonction de filtrage. En particulier, $i_1 - i_n$ doit être le plus grand possible et $pgcd(i_j - i_{j+1})$ doit être le plus petit possible. Une autre attaque possible que l'on va détailler ensuite est l'**attaque algébrique**.

Cette attaque n'est pas spécifique aux LFSR filtrés, c'est une attaque générique en cryptographie symétrique. Lors d'une attaque à clair connu, on peut toujours écrire un système d'équations reliant les chiffrés et les clairs, dont les inconnues sont les bits de clef (déjà mis en avant par Shannon en 1949). Si ce système n'est pas trop complexe, on peut tenter de le résoudre, par exemple par linéarisation (en exprimant un monôme faisant intervenir plusieurs bits de clefs comme une nouvelle variable pour résoudre un système linéaire), ou par des outils plus efficaces comme les bases de Gröbner.

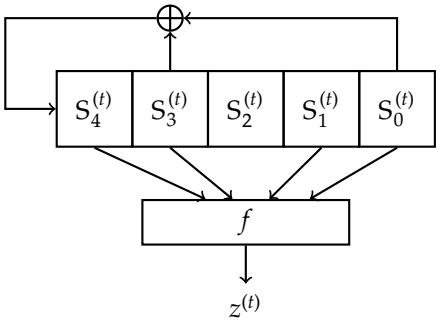
Dans le cas d'un LFSR filtré, ce système est simple à décrire. Soit un LFSR de longueur ℓ, de polynôme de rétroaction $1 \oplus c_1 X \oplus \cdots \oplus c_\ell X^\ell$ filtré par une fonction booléenne f de degré d. On note $S^{(0)} = (sk_0, sk_1, \dots, sk_{\ell-1})$ l'initialisation, où les sk_i sont les bits inconnus de clef secrète. Si on note la matrice

$$A := \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 \\ c_\ell & c_{\ell-1} & \dots & c_3 & c_2 & c_1 \end{pmatrix}$$

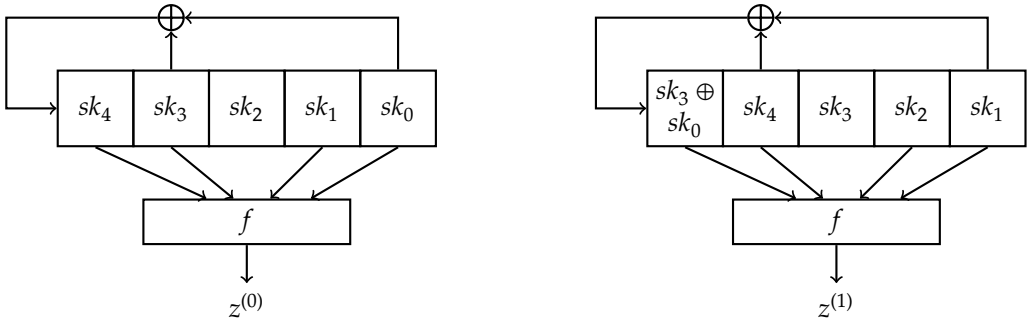
on a déjà vu que le registre au temps t, $S^{(t)} = A^t S^{(0)}$. Chaque bit de $S^{(t)}$ s'exprime donc comme une équation linéaire en les bits de clefs, $sk_0, sk_1, \dots, sk_{\ell-1}$. Le bit de sortie $z^{(t)}$ est obtenu en appliquant f à certains bits de $S^{(t)}$. On obtient ainsi une équation exprimant le bit de suite chiffrante $z^{(t)}$ comme une équation de degré d en $sk_0, sk_1, \dots, sk_{\ell-1}$.

Avec plusieurs bits de suite chiffrante, on obtient un système de degré d en les bits de clef. On peut ensuite **linéariser** ce système. Chaque monôme de degré strictement supérieur à 1, du type $sk_{j_1} sk_{j_2} \dots sk_{j_r}$ avec $r \leqslant d$ est exprimé comme une nouvelle variable. On obtient au plus $\binom{\ell}{r}$ nouvelles variables pour les monômes de degré r. Après linéarisation on a ainsi un système linéaire avec au plus $\sum_{r=1}^{d} \binom{\ell}{r}$ variables : on retrouve la majoration de la complexité linéaire.

Exemple. On considère le LFSR de longueur 5 de polynôme de rétroaction primitif $1 \oplus X^2 \oplus X^5$, filtré par la fonction booléenne quadratique $f(x_1, x_2, x_3, x_4) = x_1 x_2 \oplus x_3 x_4 \oplus x_1$, appliquée en les cases $(i_1, i_2, i_3, i_4) = (4, 3, 1, 0)$:



On note $sk_0, sk_1, \dots, sk_4$ les 5 bits inconnus de clef secrète. À $t = 0$, on a $z^{(0)} = sk_4sk_3 \oplus sk_1sk_0 \oplus sk_4$, équation quadratique en les bits de clefs.



À $t = 1$, on obtient $z^{(1)} = sk_3sk_4 \oplus sk_0sk_4 \oplus sk_2sk_1 \oplus sk_3 \oplus sk_0$, toujours une équation quadratique en les bits de clefs. On trouve de même que $z^{(2)} = (sk_4 \oplus sk_1)(sk_3 \oplus sk_0) \oplus sk_3sk_2 \oplus sk_4 \oplus sk_1 = sk_4sk_3 \oplus sk_4sk_0 \oplus sk_1sk_3 \oplus sk_1sk_0 \oplus sk_3sk_2 \oplus sk_4 \oplus sk_1$.

De manière générale, n bits de suite chiffrante donneront n équations quadratiques en $sk_0, \dots, sk_4$. Dans la phase de linéarisation on introduit $\binom{5}{2} = 10$ nouvelles variables pour chacun des monômes de degré deux : $sk_0sk_1, sk_0sk_2, \dots sk_3sk_4$. On obtient ainsi $\binom{5}{2} + \binom{5}{1} = 10 + 5 = 15$ variables. Ainsi, si on obtient 15 bits de suite chiffrante, et si les équations sont linéairement indépendantes, on pourra résoudre le système pour retrouver la clef secrète.

Le nombre d'inconnues après linéarisation peut vite être important : ainsi si on a un registre de $\ell = 256$ éléments et si f est de degré $d = 7$, on a $\sum_{r=1}^{7} \binom{256}{r} > 2^{43}$. Une manière de diminuer cette complexité est de considérer une autre fonction booléenne g de degré inférieur à $deg(f) = d$ permettant d'écrire des équations avec moins d'inconnues. C'est le principe des **attaques algébriques rapides** de Courtois et Meyer (2003) : on suppose qu'il existe une fonction booléenne $g \neq 0$ (resp. $h \neq 0$) de degré strictement inférieur à d avec $fg = 0$ (resp. $(1 \oplus f)h = 0$). La relation $z^{(t)} = f(S_{i_1}^{(t)}, S_{i_2}^{(t)}, \dots, S_{i_n}^{(t)})$ donne une équation de degré d. Si $z^{(t)} = 1$, alors $g(S_{i_1}^{(t)}, S_{i_2}^{(t)}, \dots, S_{i_n}^{(t)}) = 0$ et on obtient une équation de degré inférieur. De même si $z^{(t)} = 0$, alors $h(S_{i_1}^{(t)}, S_{i_2}^{(t)}, \dots, S_{i_n}^{(t)})$ donne une équation de degré inférieur. Plus que le degré de f, c'est donc celui de telles fonctions g et h qui donne la complexité de l'attaque algébrique.

Définition IV – 8. Si f est une fonction booléenne, on appelle annulateur de f :

$$\mathrm{AN}(f) = \{g, g \neq 0, fg = 0\}.$$

Et on appelle **immunité algébrique** de f :

$$\mathrm{AI}(f) = min\{deg(g), g \in \mathrm{AN}(f) \cup \mathrm{AN}(1 \oplus f)\}.$$

Exemple. On considère la fonction booléenne à 3 variables $f(x_1, x_2, x_3) = x_1x_2x_3 \oplus x_1$ de degré 3. On note g la fonction affine $g(x_1, x_2, x_3) = x_1 \oplus 1$. On a $fg = 0$, ce qui donne $\mathrm{AI}(f) = 1$.

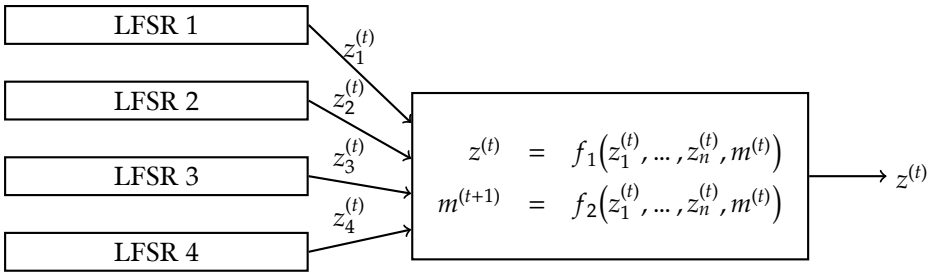
On voit facilement que $\mathrm{AI}(f) \leqslant deg(f)$ car $f \in \mathrm{AN}(1 \oplus f)$. De plus, si f est une fonction booléenne à n variables, on peut montrer que $\mathrm{AI}(f) \leqslant \lceil \frac{n}{2} \rceil$. Un critère de choix pour une fonction booléenne utilisée dans un LFSR filtré et d'avoir une immunité algébrique élevée.

Chapitre V

Quelques chiffrements par flot actuels

1. E0

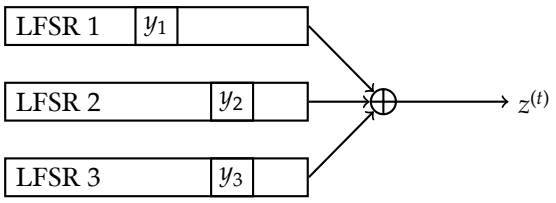
Ce chiffrement par flot est utilisé dans le protocole Bluetooth. Il utilise des **LFSR combinés avec mémoire**. On rajoute à la fonction de combinaison une mémoire (idée initialement due à Rueppel en 1985, avec le *summation generator*, utilisant une addition avec retenue) afin de rendre plus difficile les attaques par corrélations. Cette fonction avec mémoire est parfois appelée machine à état fini (FSM). De manière générale, on a le schéma suivant :



Pour E0, on utilise 4 LFSR et une mémoire m de 4 bits et une clef secrète de 128 bits. Plusieurs attaques on été proposées sur E0. La plus efficace est due à Lu, Meier et Vaudenay en 2005. C'est une attaque par corrélation spécialisée pour E0. Elle requiert environ 2^{28} bits de suite chiffrante et 2^{38} opérations.

2. A5/1

Ce chiffrement par flot est utilisé dans le GSM. Développé en 1987, les détails du système étaient gardés secrets avant d'être retrouvés par ingénierie inverse en 1999. C'est un générateur à **contrôle d'horloge**. Il utilise trois LFSR qui ne sont pas mis à jour régulièrement, de longueurs respectives 19, 22 et 23. La sortie est la somme des bits de sortie des trois LFSR. On note y_1, y_2, y_3 trois cases fixes des registres des trois LFSR. À chaque tour, on calcule le bit majoritaire : $b := \mathrm{Maj}(y_1, y_2, y_3)$ (par exemple $\mathrm{Maj}(1, 0, 0) = 0$), puis on met à jour les LFSR tels que $y_i = b$. La sortie est la somme des bits de sortie des trois LFSR. Ce contrôle d'horloge permet de casser la linéarité des LFSR.



On dénombre de nombreuses attaques sur ce système, notamment un compromis temps mémoire nécessitant deux minutes de conversation en clair, avec une phase active rapide mais de nombreuses données pré calculées (300 gigabits). Une attaque plus dévastatrice a été proposée en 2005 par Barkan et Biham (une attaque par corrélation adaptée). Elle nécessite quelques secondes de clair, et une phase active rapide (quelques minutes). De plus, elle peut être transformée en une attaque à chiffré seul nécessitant quelques minutes de conversation chiffrée. D'autres attaques exploitent l'utilisation de A5/1 dans le cadre du GSM.

Une version plus faible A5/1 a également été proposée dans certaines régions du monde. La meilleure attaque sur ce chiffrement est due à Barkan, Biham et Keller en 2003 (attaque algébrique).

3. Snow 2.0

Ce chiffrement par flot a été proposé par Johansson et Ekdahl en 2002. C'est un standard ISO, une variante, Snow 3G, est utilisée pour la téléphonie (3G et 4G/LTE). La clef secrète est de 128 ou 256 bits, un vecteur d'initialisation (IV) de 128 bits est requis. Snow 2.0 utilise un LFSR de longueur 16 défini sur $\mathbf{F}_{2^{32}}$ de registre S et de polynôme de rétroaction $1 \oplus \alpha^{-1} X^5 \oplus X^{14} \oplus \alpha X^{16}$ avec α un certain élément de $\mathbf{F}_{2^{32}}$ et où $\oplus$ désigne l'addition dans $\mathbf{F}_{2^{32}}$. Ce LFSR est filtré par une machine à état fini (FSM) incluant deux mémoires de 32 bits notées R_1, R_2 non linéairement mises à jour :

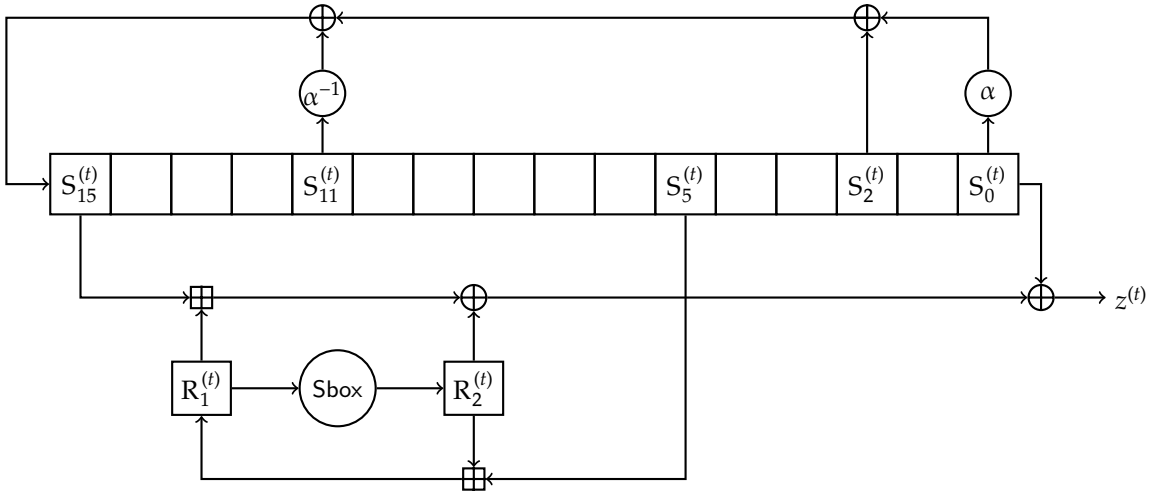
$$\begin{aligned} R_1^{(t+1)} &= S_5^{(t)} \boxplus R_2^{(t)} \\ R_2^{(t+1)} &= \mathsf{Sbox}(R_1^{(t)}) \end{aligned}$$

où $\boxplus$ désigne l'addition modulo 2^{32} et la transformation non linéaire Sbox provient de la fonction de tour de l'AES.

La sortie de la FSM au temps t est $(S_{15}^{(t)} \boxplus R_1^{(t)}) \oplus R_2^{(t)}$.

À chaque temps t le générateur sort un mot $z^{(t)}$ de 32 bits provenant de la somme du mot de sortie de la FSM et du mot de sortie $S_0^{(t)}$ du LFSR :

$$z^{(t)} = (S_{15}^{(t)} \boxplus R_1^{(t)}) \oplus R_2^{(t)} \oplus S_0^{(t)}$$



Pour la **phase d'initialisation**, la clef, ainsi que le vecteur public d'initialisation sont injectés dans le LFSR de la manière décrite ci-dessous. On ne la détaille que pour le cas 128 bits. On divise la clef sk en 4 mots de 32 bits (sk_3, sk_2, sk_1, sk_0), où sk_0 est le mot de poids faible. L'IV est de la même façon considéré comme 4 mots de 32 bits (IV_3, IV_2, IV_1, IV_0). On a alors :

$$\begin{array}{llll} S_{15}^{(0)} = sk_3 \oplus IV_0, & S_{14}^{(0)} = sk_2, & S_{13}^{(0)} = sk_1, & S_{12}^{(0)} = sk_0 \oplus IV_1, \\ S_{11}^{(0)} = sk_3 \oplus 1, & S_{10}^{(0)} = sk_2 \oplus 1 \oplus IV_2, & S_9^{(0)} = sk_1 \oplus 1 \oplus IV_3, & S_8^{(0)} = sk_0 \oplus 1, \\ S_7^{(0)} = sk_3, & S_6^{(0)} = sk_2, & S_5^{(0)} = sk_1 & S_4^{(0)} = sk_0, \\ S_3^{(0)} = sk_3 \oplus 1, & S_2^{(0)} = sk_2 \oplus 1, & S_1^{(0)} = sk_1 \oplus 1, & S_0^{(0)} = sk_0 \oplus 1, \end{array}$$

Après que le LFSR ait été initialisé, les registres R_1 et R_2 sont remis à zéro. Le système est alors mis à jour 32 fois dans un mode spécial sans produire de suite chiffrante. La sortie de la FSM est dans ce mode réinjectée dans le LFSR en la sommant avec la rétroaction.

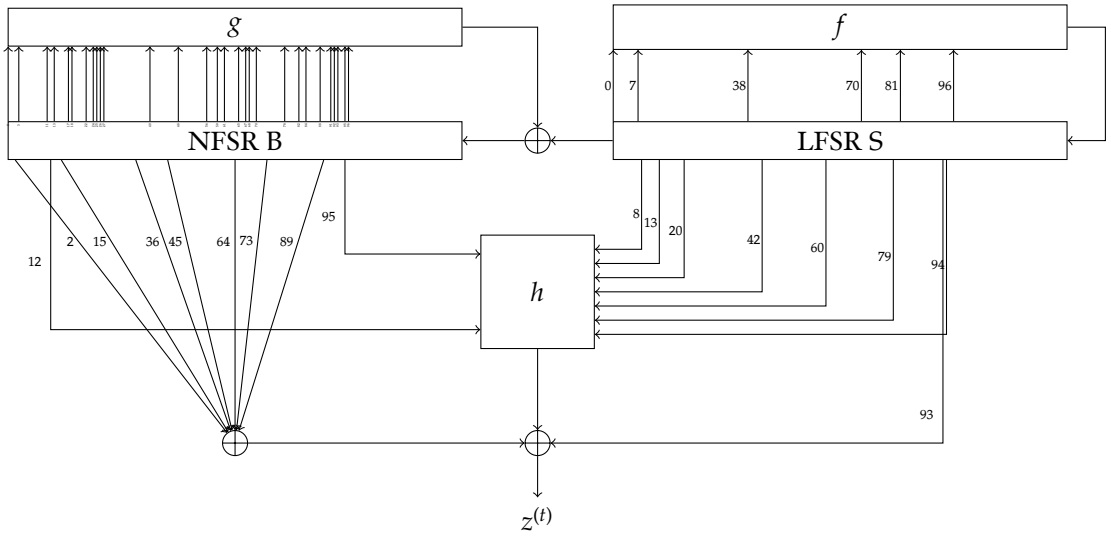
4. Grain

Le chiffrement par flot Grain a été proposé par Hell, Johansson et Meier dans le cadre de la compétition eSTREAM en 2004 et retenu parmi les 7 « vainqueurs ». Il utilise une clef secrète de 80 ou 128 bits, un IV de 64 ou 96 bits. Une nouvelle version, Grain-128a, proposée en 2011 (Ågren, Hell, Johansson, Meier) met à jour la version originale 128 bits, et ajoute un mécanisme d'authentification via un MAC.

On détaille ici cette version Grain-128a sans la partie authentification. On utilise un LFSR binaire de longueur 128 de registre S de polynôme de rétroaction primitif $f(X) = 1 \oplus X^{32} \oplus X^{47} \oplus X^{58} \oplus X^{90} \oplus X^{121} \oplus X^{128}$ et un registre à décalage non linéaire (NFSR) de longueur 128 de registre B, de polynôme de rétroaction $g(X) = 1 \oplus X^{32} \oplus X^{37} \oplus X^{72} \oplus X^{102} \oplus X^{128} \oplus X^{44}X^{60} \oplus X^{61}X^{125} \oplus X^{63}X^{67} \oplus X^{69}X^{101} \oplus X^{80}X^{88} \oplus X^{110}X^{111} \oplus X^{115}X^{117} \oplus X^{46}X^{50}X^{58} \oplus X^{103}X^{104}X^{106} \oplus X^{33}X^{35}X^{36}X^{40}$. D'autre par le bit de sortie du LFSR est sommée à la rétroaction du NFSR. Ces deux registres à décalage sont filtrés par une fonction booléenne de degré 3, h, à 9 variables, sortant à l'instant t :

$$B_{12}^{(t)}S_{8}^{(t)} \oplus S_{13}^{(t)}S_{20}^{(t)} \oplus B_{95}^{(t)}S_{42}^{(t)} \oplus S_{60}^{(t)}S_{79}^{(t)} \oplus B_{12}^{(t)}B_{95}^{(t)}S_{94}^{(t)},$$

qui est ensuite additionné avec $B_{2}^{(t)} \oplus B_{15}^{(t)} \oplus B_{36}^{(t)} \oplus B_{45}^{(t)} \oplus B_{64}^{(t)} \oplus B_{73}^{(t)} \oplus B_{89}^{(t)} \oplus S_{93}^{(t)}$ pour donner le bit de suite chiffrante $z^{(t)}$.



Pour la **phase d'initialisation**, la clef est chargée dans le registre B du NFSR et l'IV dans les 96 premières cellules du registre S du LFSR, le reste étant rempli avec des 1 et un 0 en dernière position. Ensuite, Grain est mis à jour 256 fois sans produire de suite chiffrante, la sortie étant sommée dans la rétroaction du LFSR et du NFSR.

Plusieurs **attaques** de type corrélations rapides ont été proposé sur la famille des chiffrements Grain. Ainsi en 2018, Todo, Isobe, Meier, Aoki, Zhang obtiennent des attaques de complexités estimées meilleures que la recherche exhaustive (marginalement pour la version Grain-128a : 2^{115} en temps et 2^{114} en mémoire, mais de l'ordre de 2^{76} en temps et en mémoire pour la version issue de la compétition eSTREAM).

5. ChaCha20

Ce chiffrement par flot a été proposé par Bernstein en 2008. C'est une variante de Salsa20, sélectionné par eSTREAM, visant à améliorer la diffusion de chaque tour. Suite à la disgrâce de RC4, il a été largement

adopté : il est utilisé dans TLS en remplacement de RC4 ainsi que dans OpenSSH et comme générateur pseudo aléatoire dans Linux. Il est estimé être environ trois fois plus rapide que l'AES en implantation logicielle.

On décrit ici la variante de ChaCha20 définie dans la RFC 7539 qui diffère de la version originale par l'utilisation d'un IV de 96 bits et d'un compteur de 32 bits au lieu d'un IV et d'un compteur chacun de 64 bits dans la version originale. On utilise une clef de 256 bits.

Basée sur une construction de fonction de hachage, la famille des Salsa et Chacha utilise plusieurs tours faisant intervenir des opérations de base très simples. L'état interne est représenté par une matrice 4×4 d'éléments de 32 bits,

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}$$

La fonction de base transforme un vecteur (x_0, x_1, x_2, x_3) en (y_0, y_1, y_2, y_3) en calculant

$$\begin{aligned} b_0 &= x_0 \boxplus x_1, & b_3 &= (x_3 \oplus b_0) \lll 16 \\ b_2 &= x_2 \boxplus b_3, & b_1 &= (x_1 \oplus b_2) \lll 12 \\ y_0 &= b_0 \boxplus b_1, & y_3 &= (b_3 \oplus y_0) \lll 8 \\ y_2 &= b_2 \boxplus y_3, & y_1 &= (b_1 \oplus y_2) \lll 7, \end{aligned}$$

où $\boxplus$ désigne l'addition modulo 2^{32} et $\lll n$ désigne une rotation de n bits vers les bits de poids forts.

Cette version utilise 20 tours numérotés de 1 à 20. Sur les tours impairs la fonction de base est appliquée sur les quatre colonnes de la matrice, et sur les tours pairs sur les quatre « diagonales » : $(x_0, x_5, x_{10}, x_{15})$, $(x_1, x_6, x_{11}, x_{12})$, (x_2, x_7, x_8, x_{13}), (x_3, x_4, x_9, x_{14}). On note Round la fonction correspondant à appliquer ces 20 tours sur la matrice.

La suite chiffrante est construite en s'inspirant du mode opératoire compteur des chiffrements par bloc. À chaque valeur du compteur, on associe la matrice X_i :

$$\mathrm{X}_i = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ b_i & \mathrm{IV}_0 & \mathrm{IV}_1 & \mathrm{IV}_2 \end{pmatrix}$$

où $c_0, \ldots, c_3$ sont des constantes de 32 bits, $k_0, \ldots, k_7$ constituent la clef, b_i la valeur du compteur écrite sur 32 bits et $\mathrm{IV}_0, \ldots, \mathrm{IV}_2$ le vecteur d'initialisation. Les 512 bits de suite chiffrante obtenus au temps i sont

$$\mathrm{Z}_i = \mathrm{X}_i \boxplus \mathsf{Round}(\mathrm{X}_i),$$

où $\boxplus$ désigne toujours l'addition modulo 2^{32}.

Chapitre VI

Fonction de hachage

1. Définitions

Une fonction de hachage h est une application prenant en entrée des messages, des suites de bits de longueurs quelconques et retournant un **haché** ou une empreinte de longueur fixé, par exemple une suite binaire de longueur n :

$$h : \{0,1\}^* \longrightarrow \{0,1\}^n.$$

On veut que h soit rapide à évaluer (algorithme déterministe polynomial en la taille de l'entrée).

Originellement, ces fonctions ont été introduits dans le contexte des bases de données afin d'y « ranger » des objets de natures diverses. On cherche donc en général à éviter les collisions (le fait que $h(x) = h(y)$ pour $x \neq y$) pour éviter que deux objets ne se retrouvent au même endroit, ce qui allongent la recherche dans la base de données. En cryptographie, on veut des propriétés plus restrictives (on parle parfois de **fonctions de hachage cryptographiques**).

Propriétés de sécurité attendues

- À **sens-unique** : étant donné un haché $y \in \{0,1\}^n$ il est « difficile » de trouver $m \in \{0,1\}^*$ tel que $h(m) = y$.
- Résistance à la **seconde pré-image** : étant donné un message $m \in \{0,1\}^*$, il est difficile de trouver $m' \neq m$ avec $m' \in \{0,1\}^*$ tel que $h(m) = h(m')$.
- Résistance aux **collisions** : il est difficile de trouver $m \neq m'$ avec $m, m' \in \{0,1\}^*$ tels que $h(m) = h(m')$.

Si h résiste aux collisions elle résiste à la seconde pré-image, sinon on choisit m aléatoirement, on trouve une seconde pré-image m' à $h(m)$ et on obtient une collision. Il n'y a a pas de relation directe avec la notion de sens-unique : par exemple, $h = Id : \{0,1\}^n \to \{0,1\}^n : m \mapsto m$ est injective donc résistante aux collisions mais n'est pas à sens-unique.

Si h n'est pas injective, il existe des collisions. Par exemple, si h désigne l'application qui a une personne associe le jour de son anniversaire (à valeurs dans $\{1, \dots, 365\}$), alors le paradoxe des anniversaires nous dit qu'avec 23 évaluations différentes de h on a plus de 50% de chances d'avoir une collision. De manière générale, si h est à valeurs dans $\{0,1\}^n$, en $\mathcal{O}(2^{n/2})$ évaluations on a une bonne probabilité d'obtenir une collision. En pratique, on prend donc $n \geqslant 160$ pour avoir une sécurité de 80 bits.

Plus généralement, on aimerait qu'une fonction de hachage se comporte comme un oracle aléatoire : pour obtenir le haché de m, on soumet m à un oracle modélisant h qui répond par une valeur aléatoire de $\{0,1\}^n$, $h(m)$, obtenue avec équiprobabilité. Par contre, si on a déjà soumis m à l'oracle, alors on obtient la même valeur $h(m)$.

2. Applications

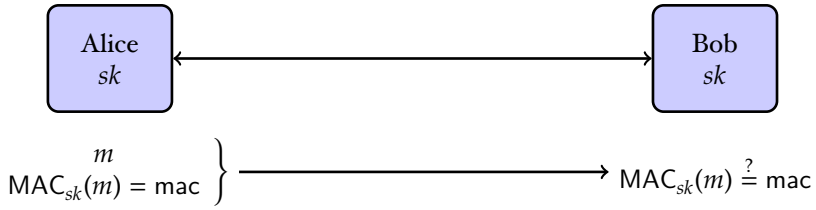
Une application directe est d'assurer l'**intégrité** de messages ou de fichiers. Modifier un fichier donné afin qu'il ait toujours un haché donné correspond à trouver une seconde pré-image (mais cette application pose le problème de l'intégrité du haché). Cette propriété est aussi utilisé pour les **signatures numériques**. On ne signe pas directement un message mais un haché du message. La résistance à la seconde pré-image empêche alors un adversaire à partir d'un message et de sa signature de produire un autre message ayant la même signature. Cela permet également d'éviter par exemple les attaques homomorphes sur la signature RSA, ou de construire d'abord une signature σ puis un message ayant σ pour signature si la fonction de hachage est à sens-unique.

Le **chiffrement asymétrique** utilise aussi souvent des fonctions de hachage (par exemple le standard OAEP) pour résister aux attaques à chiffrés choisis. Pour toutes ces applications en cryptographie asymétrique, on idéalise souvent les fonctions de hachage utilisée, c'est le modèle de l'oracle aléatoire.

Une autre application est la **vérification de mot de passe**. Plutôt que de stocker des mot de passe sur un serveur, on stocke leur haché. Lors d'une authentification, le serveur hache le mot de passe reçu et compare les hachés. Cela évite la divulgation des mots de passe en cas de compromission du serveur. Retrouver les mots de passe à partir du haché revient à attaquer la notion de sens-unique (en général on fait une attaque par dictionnaire, le mot de passe appartenant à un ensemble de cardinal petit).

D'autres applications comme extracteur d'aléa, pour la génération de mot de passe à usage unique (*One-Time Password*), ou la dérivation de clefs de chiffrement symétrique (à partir d'un mot de passe par exemple).

Une dernière application des fonctions de hachage est de fournir une construction de *Message authentication code* (**MAC**). Lors d'une communication, un MAC permet de garantir l'intégrité du message et authentifie l'expéditeur par l'utilisation d'une clef secrète. Par contre un MAC ne garantit pas la non-répudiation comme les signatures numériques : toutes les personnes qui peuvent vérifier un MAC (c'est à dire qui possèdent la clef secrète) peuvent aussi créer un MAC. La sécurité d'un MAC consiste à ce qu'un attaquant ne puisse créer un nouveau MAC valide étant connus des couples $(m_i, \mathsf{MAC}_{sk}(m_i))$.



La construction la plus populaire est HMAC, proposée par Bellare, Canetti, et Krawczyk en 1996, qui est standardisée et utilisée dans de nombreux protocoles. On a $\mathsf{HMAC}_{sk}(m) = h(sk_2 \| h(sk_1 \| m))$ avec $sk_2 = s_k \oplus \mathsf{opad}$ et $sk_1 = s_k \oplus \mathsf{ipad}$, avec opad et ipad des constantes. La notation $\|$ désigne la concaténation des chaînes binaires. Seules des attaques de HMAC utilisant MD4 ont été découvertes à ce jour. On verra en section 5 pourquoi on évite des constructions plus simples du type $\mathsf{MAC}_{sk}(m) = h(sk \| m)$.

3. Exemples

- MD4, (*Message Digest*), 128 bits, Rivest 1990, collision en quelques millisecondes, 2 hachages, obsolète.
- MD5 128 bits, Rivest 1992, collision en quelques secondes, 2^{21} hachages, obsolète.
- SHA0, (*secure hash algorithm*), 160 bits, NIST 1993, collision trouvée en 2^{51} opérations par Joux *et. al.* en 2004, obsolète.
- SHA1, 160 bits, NIST 1993, collision trouvée en $2^{63.1}$ opérations par Stevens, Bursztein, Karpman, Albertini, Markov en 2017
- SHA2, plusieurs variantes de 224 à 512 bits, NIST 2001, pas d'attaques sur la fonction complète

- SHA3, vainqueur de la compétition du NIST (2007-2012), Keccak, 224 à 512 bits, Bertoni, Daemen, Peeters, Van Assche, 2008

Les résultats sur les attaques par collisions sur MD4, MD5 et les premières sur SHA1 sont surtout dues à Wang *et. al.* en 2004. Ce sont des attaques dites différentielles : on considère une paire de messages avec une petite différence et on cherche à contrôler la propagation des différences.

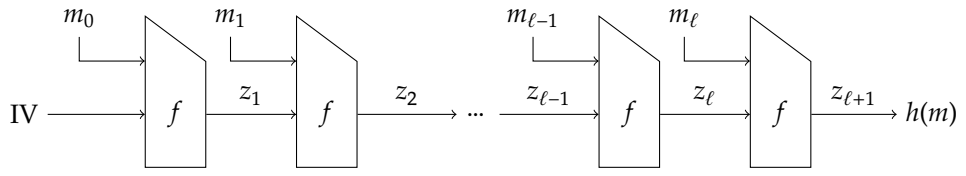
Les collisions sur MD5 mènent à des attaques « pratiques ». En effet, en 2008, Sotirov *et. al.* ont utilisé une attaque par collision avec préfixe fixé sur MD5 pour créer un certificat X.509 valide : la partie à signer par l'autorité avait le même haché MD5 qu'un certificat légitimement établi. La recherche de collision a été faite à l'aide d'un *cluster* de 200 PlayStation 3 en 64 heures. Une technique similaire a été utilisé pour le virus Flame découvert en 2012 et déployé à des fins d'espionnage : un faux certificat de Microsoft a été utilisé pour signer le virus afin de le faire passer pour un composant légitime, toujours à l'aide d'une collision sur MD5.

L'attaque de Stevens, Bursztein, Karpman, Albertini, Markov sur SHA1 a pris l'équivalent de 6500 années CPU et 100 ans GPU ce qui est estimé être légèrement plus que les records actuels de factorisations et calculs de logarithme discret de 768 bits. SHA1 est non recommandé par le NIST depuis 2015 et les principaux navigateurs web ne le supportent plus en 2017. En résumé, il ne faut plus utiliser SHA1 !

4. Construction de Merkle-Damgård

Cette construction est suivie par les fonctions MD5, SHA1 et SHA2 avec des variations sur les itérations initiale et finale. Elle permet de transformer une fonction de hachage admettant une entrée de longueur fixée (dite fonction de compression) en une fonction de hachage admettant une entrée de longueur (presque) quelconque. On note f une fonction de compression de $\{0,1\}^{n+k}$ dans $\{0,1\}^n$, avec $k > 0$. Soit IV un élément fixé de $\{0,1\}^n$. Soit m message à hacher. On commence par découper m en ℓ blocs de k bits, $m_0, \dots, m_{\ell-1}$ en « paddant » $m_{\ell-1}$ par $10000\dots$ pour obtenir un bloc de k bits. Dans le bloc m_ℓ, on code sur exactement k bits le nombre de bits du message m. Il faut donc que m ait strictement moins de 2^k bits. Ce rajout de la longueur de m est parfois appelé *Merkle–Damgård strengthening*. Il permet d'éviter que le haché de m soit le même que celui d'une sous-chaîne de m en ajustant l'IV, ou alors de construire des collisions à l'aide d'une pré-image de l'IV.

On note $z_0 = \text{IV}$ et pour $i = 0, \dots, \ell$, $z_{i+1} = f(m_i \| z_i)$. Le haché $h(m)$ de m est alors $h(m) = z_{\ell+1}$.



Théorème VI – 1 (Merkle, Damgård 1989)**.** Soit f une fonction de compression résistante aux collisions. La construction précédente appliquée à f donne une fonction de hachage h résistante aux collisions.

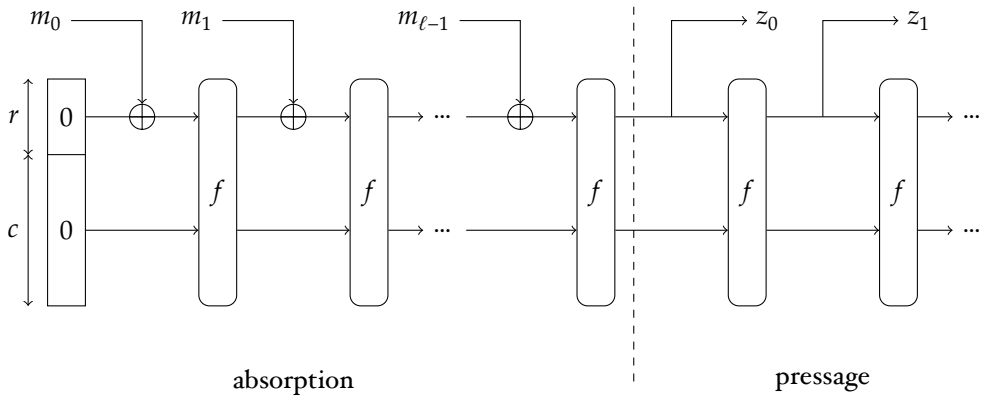
Démonstration. Soit $m \neq m'$ tel que $h(m) = h(m')$. Supposons que m et m' aient des longueurs différentes. La dernière itération donne $h(m) = f(m_\ell \| z_\ell) = h(m') = f(m'_{\ell'} \| z'_{\ell'})$. On a alors $m_\ell \neq m'_\ell$ puisque ce bloc code la longueur. Donc on a trouvé une collision sur f.

Supposons maintenant m et m' de même longueur, ce qui implique que le nombre d'itérations est le même pour m et m' : $\ell = \ell'$. On a $f(m_\ell \| z_\ell) = f(m'_\ell \| z'_\ell)$, avec $m_\ell = m'_\ell$. Soit $z_\ell \neq z'_\ell$ et on a trouvé une collision sur f, soit $z_\ell = z'_\ell$. Dans ce cas, on a, à l'itération précédente, $f(m_{\ell-1} \| z_{\ell-1}) = f(m'_{\ell-1} \| z'_{\ell-1})$, soit $m_{\ell-1} \| z_{\ell-1} \neq m'_{\ell-1} \| z'_{\ell-1}$ et on a trouvé une collision, soit $m_{\ell-1} \| z_{\ell-1} = m'_{\ell-1} \| z'_{\ell-1}$. On remonte ainsi jusqu'à trouver une collision sur f. S'il n'y a pas de collision, $m_i \| z_i = m'_i \| z'_i$ pour $i = 0$ à ℓ, ce qui implique que $m = m'$ et on a une contradiction. □

Pour construire une fonction de compression, on peut par exemple utiliser la construction de Davies et Meyer qui utilise un chiffrement par bloc : on a $f(m_i||z_i) = \mathsf{Encrypt}_{m_i}(z_i) \oplus z_i$ (le rajout de z_i permet de complexifier les collisions). En pratique, on utilise des constructions plus rapide.

Pour MD5, on utilise Merkle Damgård avec une fonction f de compression $\{0,1\}^{n+k}$ dans $\{0,1\}^n$ avec $n = 128$ et $k = 512$. Cette fonction f consiste à appliquer 64 fois une opération de base (64 tours), opérant sur 4×32 bits et composant des permutations, des additions modulo 2^{32}, des rotations, et l'application de fonctions booléennes. Pour SHA1, $n = 160$ et $k = 512$. La fonction f consiste en 80 tours et l'état interne fait maintenant 5×32 bits. Pour SHA2, SHA256 (et SHA224 qui est une version tronquée) utilise $n = 256$ et $k = 512$ et une fonction f de 64 tours opérant sur un état interne de 8×32 bits et SHA512 (et SHA384) $n = 512$ et $k = 1024$ et une fonction f de 80 tour opérant sur un état interne de 8×64 bits.

Une autre construction de fonction de hachage a été proposée et utilisée par les auteurs de Keccak (Bertoni, Daemen, Peeters, Van Assche), le vainqueur du concours SHA3 : les fonctions éponges. Un état interne de $r + c$ bits est utilisé. On note $m_0, \ldots, m_{\ell-1}$ le message avec *padding*, découpé en blocs de r bits. À chacun des ℓ tours, le bloc m_i est ajouté bit à bit aux r premiers bit de l'état, puis une permutation f est appliqué à l'état (c'est la partie d'absorption). Une fois que tout le message a été traité, le haché est produit sur plusieurs tours : à chaque tour les r premiers bits de l'état donnent une partie du haché ($z_0, z_1, \ldots$ sur le dessin ci-dessous), et la fonction f est appliquée sur l'état (c'est la partie où l'on presse l'éponge). Quand f est une permutation aléatoire, les sorties de cette construction sont indistinguables d'un oracle aléatoire.



SHA3 a été standardisé par le NIST en Août 2015. (cf. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`). Comme pour SHA2, plusieurs versions de SHA3 sont spécifiées : SHA3-224, SHA3-256, SHA3-384, SHA3-512, Ces versions correspondent à fixer $r + c = 1600$ bits dans la construction de Keccak. Les tailles de hachés n sont respectivement $224, 256, 384$ et 512 bits, et $c = 2n$, on a donc r égal respectivement à $1152, 1088, 832$ ou 576 suivant la taille du haché. L'état interne, de 1600 bits est donc beaucoup plus gros que pour les fonctions basées sur Merkle Damgård. Un seul tour de pressage d'éponge est effectuée comme r est plus grand que n (les n bits de poids fort sont utilisés). La permutation f est construite en appliquant 24 fois une fonction de tour. L'état interne est vu comme un pavé de $5 \times 5 \times 64$ bits. Cette fonction de tour est la composition de 5 fonctions. La première, θ, consiste à rajouter à chaque bit de l'état la parité de deux colonnes. La deuxième ρ, est une rotation circulaire sur chaque ligne (dans le sens de l'axe des z). La fonction π, est une permutation des bits de chaque tranche du plan (x, y). La fonction χ est la seule fonction non linéaire, elle consiste à appliquer la fonction booléenne $x_1 \oplus x_2 x_3 \oplus x_3$ sur 3 bits d'une même ligne (dans le sens de l'axe des x). Enfin la fonction ι, change uniquement une ligne (dans le sens de l'axe des z) en lui ajoutant une constante ne dépendant que du numéro de tour, calculée à l'aide d'un LFSR.

5. Quelques cryptanalyses

Recherche exhaustive de collisions

Pour la recherche de collisions sur une fonction de hachage, l'attaque naïve consiste à tirer des messages aléatoires, et à stocker les hachés, dans une liste triée jusqu'à avoir une collision. Si la fonction de hachage

fournie des hachés de n bits, le paradoxe de anniversaires implique de stocker $2^{n/2}$ hachés pour avoir une bonne chance de trouver une collision. Cette quantité de mémoire va être un facteur limitant comparé au temps de calcul.

Une recherche en $2^{n/2}$ utilisant peu de mémoire peut-être faite en s'inspirant de l'algorithme ρ de Pollard de factorisation ou de calcul de logarithme discret. On prend un message m_0 aléatoire puis pour $i > 0$, on pose $m_i = h(m_{i-1})$. On itère la fonction h, on doit nécessairement arriver à une collision, puis à un cycle, car l'ensemble des hachés est fini. Pour une fonction aléatoire à valeurs dans $\{0,1\}^n$, la première collision arrive en $\mathcal{O}(2^{n/2})$. On espère que h se comporte comme une fonction de aléatoire.

La détection de la collision peut se faire avec l'algorithme de recherche de cycle de Floyd comme pour Pollard. À chaque tour seuls $m_i = h(m_{i-1})$ et $m_{2i} = h(h(m_{2(i-1)}))$ sont calculés et comparés, ce qui utilise très peu de mémoire.

Attaques par extension sur Merkle-Damgård et applications aux MAC

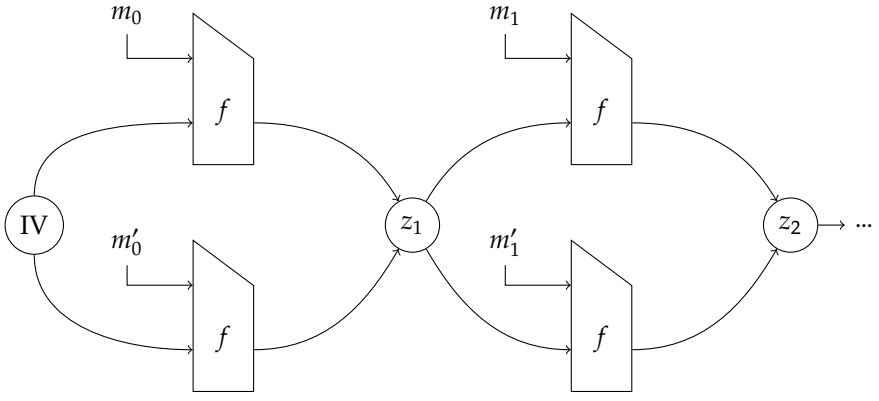
Soit h une fonction de hachage suivant la construction de Merkle-Damgård. Supposons connaître un haché de $h(m)$ et la longueur du message m. Avec les notations de la construction, on a $z_{\ell+1} = h(m)$. En rajoutant des itérations à partir de $z_{\ell+1}$, on peut obtenir le haché de $m' = m\|\mathsf{pad}(m)\|s$ pour tout message s, où $\mathsf{pad}(m)$ désigne le ***padding*** utilisé : $10\ldots0$ au bloc $m_{\ell-1}$ et m_ℓ, le bloc codant la longueur de m. À la dernière itération, on code la longueur de m' connaissant celle de m.

Cette attaque permet d'attaquer la construction simple d'un MAC : $\mathsf{MAC}_{sk}(m) = h(sk\|m)$. Connaissant un MAC existant, c'est à dire le haché $h(sk\|m)$, on peut construire $h(sk\|m\|\mathsf{pad}(sk\|m)\|s)$, c'est à dire $\mathsf{MAC}_{sk}(m\|\mathsf{pad}(sk\|m)\|s)$, sans connaître ni sk ni m, mais leur longueur.

La construction $\mathsf{MAC}_{sk}(m) = h(m\|sk)$, permet de traduire une attaque par collision sur h en une attaque ***offline*** sur le MAC. On trouve $h(m) = h(m')$. Par l'attaque par extension, on obtient $h(m\|\mathsf{pad}(m)\|sk) = h(m'\|\mathsf{pad}(m')\|sk)$ quelque soit la clef sk. On a donc $\mathsf{MAC}_{sk}(m\|\mathsf{pad}(m)) = \mathsf{MAC}_{sk}(m'\|\mathsf{pad}(m'))$.

Attaques par multicollisions sur Merkle-Damgård

Soit $r > 1$ un entier. Trouver une r–collision sur une fonction de hachage consiste à trouver $m_1, \ldots, m_r$ tous distincts tel que $h(m_1) = \cdots = h(m_r)$. Avec $r = 2$ on retrouve les collisions classiques. Une généralisation du paradoxe des anniversaires donne que si h est à valeurs dans $\{0,1\}^n$, en $\mathcal{O}(2^{(r-1)n/r})$ évaluations on a une bonne probabilité d'obtenir une collision. En 2004, Joux a observé que trouver des r-collisions avec $r = 2^k$ sur une construction de type Merkle-Damgård est plus facile que cela, ce qui la distingue d'un oracle aléatoire, tout comme l'attaque par extension.



On commence par chercher deux blocs $m_0 \neq m'_0$ tel que $f(m_0\|\mathrm{IV}) = f(m'_0\|\mathrm{IV}) = z_1$. Une telle valeur peut se trouver en $\mathcal{O}(2^{n/2})$ par le paradoxe des anniversaires. Puis on cherche, avec la même complexité, deux blocs $m_1 \neq m'_1$ tel que $f(m_1\|z_1) = f(m'_1\|z_1) = z_2$. On continue ainsi, jusqu'à obtenir deux blocs $m_{k-1} \neq m'_{k-1}$ tel que $f(m_{k-1}\|z_{k-1}) = f(m'_{k-1}\|z_{k-1}) = z_k$. En complétant les messages par le ***padding*** adéquat, on obtient une 2^k-collision en considérant des messages commençant par $m_0\|m_1\|\ldots\|m_{k-1}$, $m'_0\|m_1\|\ldots\|m_{k-1}, \ldots, m'_0\|m'_1\|\ldots\|m'_{k-1}$. Ceci s'obtient en $\mathcal{O}(k2^{n/2})$.

Cette attaque permet d'évaluer la sécurité de la concaténation de deux fonctions de hachage à valeurs dans $\{0,1\}^n$: soit h définie par $h(m) = h_1(m)||h_2(m)$. Il est facile de voir que si h_1 ou h_2 est résistante aux collisions alors h est résistante aux collisions.

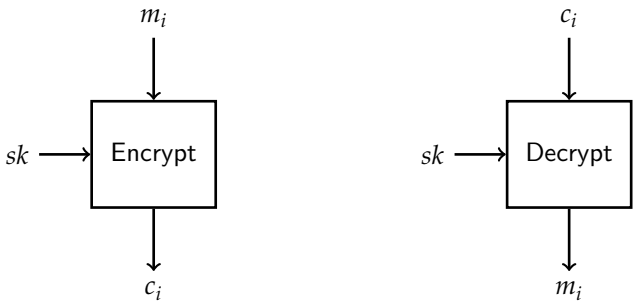
Supposons maintenant que h_1, par exemple, suit la construction de Merkle-Damgård. On peut trouver une $2^{n/2}$-collision sur h_1 en $\mathcal{O}(n2^{n/2})$. Parmi ces $2^{n/2}$ messages, on a une bonne chance d'avoir une collision sur h_2. On construit donc une collision sur h en $\mathcal{O}(n2^{n/2})$ au lieu de $\mathcal{O}(2^n)$.

Chapitre VII

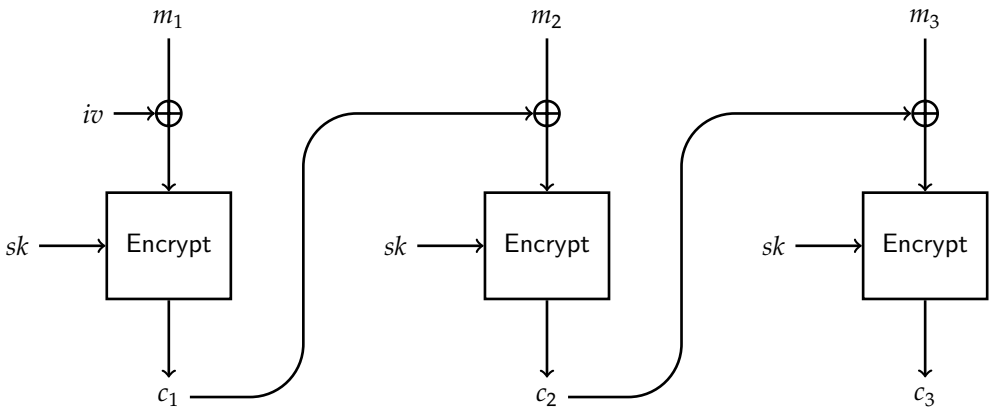
Chiffrement par bloc, introduction

1. Introduction

Un algorithme de **chiffrement par bloc** prend en entrée un message clair de n bits (que l'on peut voir comme un élément de $\mathbf{F}_2^n$) et donne en sortie un chiffré, qui est un autre bloc de bits (en général également n). Si le message clair est de taille plus grande que n on le découpe en des blocs, $m_0, m_1, \ldots$ de taille n. Puis pour chaque $i = 0, 1, 2, \ldots$, on applique le chiffrement. Le déchiffrement se fait de manière similaire. On peut voir la fonction de chiffrement comme une permutation de $\mathbf{F}_2^n$ sélectionnée par la clef (2^k choix pour une clef de k bits) parmi les $2^n!$ possibles.



C'est le mode ECB, *Electronic CodeBook*. Ce mode ne cache pas les redondances éventuelles du texte clair, (par exemple si $m_i = m_j$ avec $i \neq j$ alors $c_i =_j$). D'autres modes remédient à ce problème. On a déjà vu les modes OFB (*Output FeedBack*) et CFB (*Cipher FeedBack*) mimant respectivement des chiffrements à flot synchrone, et auto-synchronisant. Le mode CBC (*Cipher Block Chaining*) utilise le schéma suivant :



L'inconvénient de ce mode est que l'on peut faire une attaque à message clair choisi adaptative permettant de distinguer les chiffrés d'un message précis (observation due à Rogaway en 1995). Plus précisément, un attaquant obtient les blocs $c_1, \dots, c_k$ chiffrés en mode CBC. Il souhaite vérifier si le bloc de clair i, avec $1 \leqslant i \leqslant k$, vaut une certaine valeur m, c'est à dire si $m_i = m$ ou non. L'attaquant construit un clair $m_{k+1} = c_k \oplus m \oplus c_{i-1}$ et demande son chiffrement. Ce message sera chiffré en $c_{k+1} = \mathsf{Encrypt}_{sk}(m_{k+1} \oplus c_k) = \mathsf{Encrypt}_{sk}(m \oplus c_{i-1})$. Si $m_i = m$, on aura $c_{k+1} = c_i$.

Cette attaque était possible dans TLS v1.0. Dans les versions suivantes de TLS (à partir de 2006), on impose l'utilisation à chaque bloc de message m_i d'un iv aléatoire, transmis avec c_i. Ainsi l'attaquant ne pas prédire l'iv utilisé pour chiffré le message m_{k+1}, contrairement au mode CBC classique. Cependant, en 2011, cette contremesure étant peu déployée, une attaque dévastatrice à pu être menée : BEAST due à Duong et Rizzo. En utilisant les idées ci-dessus et en exploitant d'autres failles de TLS, elle permet de retrouver entièrement le message clair m_i.

La *construction* des chiffrements par bloc utilise la plupart du temps un schéma itératif. Les itérations, appelées tours ou rondes, sont en général identiques (à part la première et la dernière), seule la clef de tour, créée à partir de la clef secrète sk au moyen d'un algorithme dit de cadencement de clef, change.

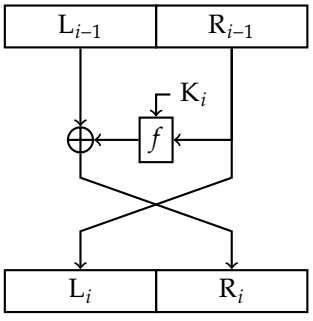
$$m = x_0 \xrightarrow{F_{K_0}} x_1 \xrightarrow{F_{K_1}} x_2 \longrightarrow \dots \xrightarrow{F_{K_{r-1}}} x_r = c$$

Pour obtenir au final une permutation, deux classes générales de constructions ont été proposées : les schémas de Feistel et les schémas substitution permutation (SPN).

2. Schéma de Feistel, le DES

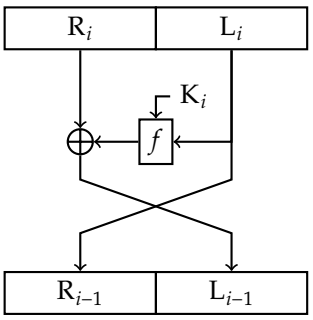
Cette construction a été introduite par Feistel dans les années 70. On suppose que le bloc de message clair est de longueur paire n et on le découpe en deux blocs de longueur $\frac{n}{2}$, notés L_0 et R_0. On note $m = \mathrm{L}_0 \| \mathrm{R}_0$. À chaque tour $i = 1, 2, \dots, r$, on prend en entrée un bloc $(\mathrm{L}_{i-1}, \mathrm{R}_{i-1})$ et on le transforme en un bloc $(\mathrm{L}_i, \mathrm{R}_i)$ en faisant intervenir la clef de tour K_i. On note f une fonction prenant en entrée et en sortie des blocs de $n/2$ bits. La transformation se fait par les formules :

$$\mathrm{L}_i = \mathrm{R}_{i-1}, \qquad \mathrm{R}_i = \mathrm{L}_{i-1} \oplus f(\mathrm{R}_{i-1}, \mathrm{K}_i)$$



Au bout de r tours, le chiffré est $c = \mathrm{R}_r \| \mathrm{L}_r$ (on ne « croise » pas les flèches au dernier tour). Ce schéma est inversible, (si l'on connaît les clefs de tours), que f soit une bijection ou non. En effet, on a

$$\mathrm{R}_{i-1} = \mathrm{L}_i, \qquad \mathrm{L}_{i-1} = \mathrm{R}_i \oplus f(\mathrm{R}_{i-1}, \mathrm{K}_i).$$



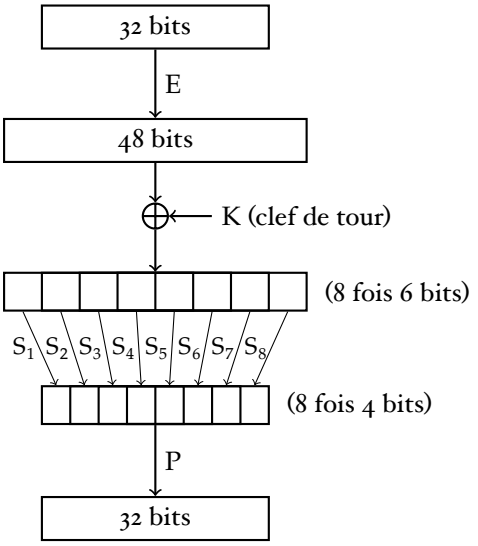
Le déchiffrement de $c = R_r||L_r$ se fait donc avec exactement le même procédé que le chiffrement en appliquant les clefs de tours dans l'ordre inverse. On retrouve bien, à la dernière étape, (toujours sans croiser) $L_0||R_0$.

Des résultats théoriques ont été obtenus sur la sécurité des schémas de Feistel. Si la fonction de tour f est une fonction pseudo-aléatoire (c'est-à-dire dont les sorties sont indistinguables d'une fonction aléatoire) alors 3 tours de schémas de Feistel permette d'obtenir une permutation pseudo-aléatoire (Luby et Rackoff, 1988). Par conséquent, ils sont très utilisés en cryptographie symétrique pour construire des chiffrements par bloc. Il est facile de voir que deux tours ne suffisent pas : en effet $L_0||R_0$ est transformé en $L_2||R_2$ et $(L_0 \oplus L_0^\star)||R_0$ est transformé en $(L_2 \oplus L_0^\star||R_2')$. Un tel comportement semble peu probable pour une permutation aléatoire.

On les retrouve également en cryptographie asymétrique : le procédé OAEP (*Optimal Asymetric Encryption*) est utilisé en entrée d'une permutation à trappe comme la fonction RSA pour garantir la sécurité sémantique pour des attaques à chiffrés choisis. Cela consiste en un schéma de Feistel à 2 tours, en rajoutant de l'aléa au message clair, les fonctions de tour étant des fonctions de hachage (vu comme des oracles aléatoires).

Le **DES**, ***Data Encryption Standard***, standard de chiffrement par bloc de 1977 à 2000 utilise un schéma de Feistel. Les blocs de clair et chiffré sont de 64 bits, la clef secrète de 56 bits et les clefs de tour de 48 bits. On effectue 16 tours de schéma de Feistel avec des permutations initiale et finale.

La fonction de tour opère sur des mots de 32 bits. Elle est la composition de plusieurs fonctions, suivant le schéma suivant.



La fonction d'expansion E est linéaire de $\mathbf{F}_2^{32} \to \mathbf{F}_2^{48}$ (on répète certains bits). La fonction P est une permutation des 32 bits, donc aussi une fonction linéaire de $\mathbf{F}_2^{32} \to \mathbf{F}_2^{32}$. Ces deux fonctions, E et P apportent de la **diffusion** c'est-à-dire que si l'on change un bit en entrée, il y aura plusieurs bits changés en

sortie. Cela permet d'empêcher les attaques statistiques (par exemple si un bit du clair est souvent égal à 1, cela ne doit pas se voir sur la distribution des chiffrés correspondant).

Les fonctions $S_1, \dots, S_8$ sont appelées **boîtes S** ou *S-box*. Ce sont des fonctions booléennes vectorielles, ici de $\mathbf{F}_2^6 \to \mathbf{F}_2^4$, non linéaires. Elles permettent d'apporter de la **confusion** : le but est de rendre complexe les relations entre bits de chiffré et bits de clef (dans une attaque à clair connu, on peut faire une attaque algébrique en écrivant les bits de chiffrés en fonction des bits de clef. On veut que ces équations soient complexes. En particulier changer un bit de clef, change complètement le chiffré. Les concepts de diffusion et confusion ont été introduits par Shannon en 1949). Les boites S sont décrites par la table des 2^6 sorties possibles. Elles ont été construites suivant des critères de conception non publics. Cependant on s'est aperçu *a posteriori* qu'elles avaient été choisies pour résister à la cryptanalyse différentielle, redécouverte par Biham et Shamir en 1988 (nécessite 2^{47} clairs choisis). De plus, elles sont à grande distance de Hamming des fonctions affines, ce qui rend le système assez résistant à la cryptanalyse linéaire (nécessite 2^{43} clairs connus) découverte par Matsui en 1993. En pratique la meilleure attaque reste la recherche exhaustive qui est maintenant largement faisable.

Pour palier à la faiblesse du DES due à sa clef de 56 bits trop courte , on utilise encore couramment aujourd'hui (dans le monde bancaire) une variante utilisant 3 clefs DES sk_1, sk_2, sk_3, appelée **Triple DES**. Cela consiste à composer trois fois le DES (donc trois fois plus lent que le DES) de la manière suivante :

$$c = \mathsf{Encrypt}_{sk_3}\Big(\mathsf{Decrypt}_{sk_2}\big(\mathsf{Encrypt}_{sk_1}(m)\big)\Big),$$

avec trois options sur le choix des clefs :

Option 1 : trois clefs distinctes, ce qui revient à 168 bits de clef ;

Option 2 : $sk_1 = sk_3$ et $sk_2 \neq sk_1$, ce qui revient à 112 bits de clef ;

Option 3 : $sk_1 = sk_2 = sk_3$, une seule clef de 56 bits, et $c = \mathsf{Encrypt}_{sk_1}(m)$ on a un chiffrement classique du DES pour garantir la compatibilité (ce qui explique pourquoi on alterne chiffrement et déchiffrement).

Ces compositions de plusieurs chiffrements sont vulnérables à l'attaque ***meet in the middle*** que l'on doit à Diffie et Hellman (1977). On étudie le cas où l'on fait seulement une composition en définissant

$$c = \mathsf{Encrypt}_{sk_2}\big(\mathsf{Encrypt}_{sk_1}(m)\big).$$

On suppose un couple (m, c) connu. On construit une table de couples $(sk_1, \mathsf{Encrypt}_{sk1}(m))$ pour toutes les clefs sk_1 possibles. Pour toute clef sk_2 on calcule $\mathsf{Decrypt}_{sk_2}(c)$ et on cherche ce résultat dans la table précédente. Ceci donnera un couple (sk_1, sk_2) pour lequel $\mathsf{Decrypt}_{sk_2}(c) = \mathsf{Encrypt}_{sk1}(m)$, c'est à dire $c = \mathsf{Encrypt}_{sk_2}\big(\mathsf{Encrypt}_{sk_1}(m)\big)$. Si les clefs font k bits, ceci donne une attaque en $\mathscr{O}(2^{k+1})$ opérations et $\mathscr{O}(2^k)$ mémoire. Ainsi, en composant seulement une fois le DES (au lieu d'utiliser l'option 2), on obtient une sécurité équivalente à un seul DES.

Au final, la sécurité du Triple DES est la suivante :

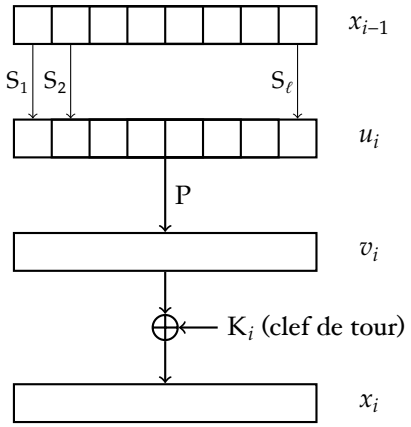
Option 1 : En « coupant » la composition, l'attaque *meet in the middle* fait tomber la sécurité à $2 \times 56 = 112$ bits.

Option 2 : En prenant $sk_1 = sk_3$ et $sk_2 \neq sk_1$, on ne peut plus faire cette attaque. Cependant d'autres attaques sont possibles (notamment une de Merkle et Hellman en 1981), et on estime que la sécurité est de 80 bits.

Option 3 : On a un DES classique donc 56 bits de sécurité.

3. Schéma substitution permutation (SPN), l'AES

C'est une autre construction itérative de chiffrement par bloc. On utilise maintenant une fonction de tour bijective en alternant les opérations de diffusion et de confusion. On effectue une étape initiale d'ajout bit à bit de la première clef de tour : $x_0 = m + \mathrm{K}_0$. Puis pour $i = 1, \dots, r$, on calcule $x_i = \mathrm{F}_{\mathrm{K}_i}(x_{i-1})$ pour obtenir $c = x_r$. La fonction f commence par l'application de ℓ boîtes S bijectives (étape de confusion) sur x_{i-1} découpé en ℓ sous blocs, pour donner un nouveau bloc u_i. Puis on applique une permutation P (étape de diffusion) sur les bits de u_i, on note v_i le résultat. Enfin, on ajoute la clef de tour : $x_i = v_i + \mathrm{K}_i$.



L'étape initiale évite qu'un attaquant puisse calculer le début du chiffrement jusqu'à l'ajout de clef K_1. Le déchiffrement se fait en « remontant » tout le chiffrement, toutes les opérations étant inversibles.

Un exemple de tel schéma est l'**AES**, ***Advanced Encryption Standard***. Ce standard pour remplacé le DES est issu d'un concours qui s'est déroulé de 1997 à 2001. Le vainqueur a été l'algorithme Rijndael conçu par Joan Daemen et Vincent Rijmen. L'AES utilise une clef du 128, 192 ou 256 bits avec des blocs de 128 bits. Suivant la taille de la clef, le nombre de tours est respectivement 10, 12 et 14. On détaille le fonctionnement de l'AES dans le cas 128 bits. L'état interne est vu comme un tableau de 4×4 octets. Chaque octet étant identifié avec un élément de $\mathbf{F}_{2^8}$ par le choix d'un polynôme irréductible (non primitif) de degré 8. L'état est ainsi une matrice de $\mathscr{M}_4(\mathrm{F}_{2^8})$, l'ensemble des matrices 4×4 à coefficients dans $\mathbf{F}_{2^8}$.

La fonction de diffusion est la composée de deux fonctions linéaires, *ShiftRows* et *MixColumns*. La fonction *ShiftRows* consiste en une permutation circulaire des lignes de la matrice tandis que *MixColumns* est une multiplication par une matrice fixe inversible de $\mathscr{M}_4(\mathbf{F}_{2^8})$. Ces fonctions linéaires n'agissent que sur les octets, et pas sur les bits de l'état.

La fonction de substitution consiste en une seule boîte S appliquée 16 fois, sur chaque octet de l'état. Cette fonction consiste à composer l'inversion dans $\mathbf{F}_{2^8}$ (complétée par $0 \mapsto 0$) avec une transformation affine dans $\mathbf{F}_2^8$, pour casser le caractère algébrique de l'inversion (qui peut se ré écrire $x \mapsto x^{254}$), du type $Ax \oplus b$ avec A une matrice inversible fixe de $\mathscr{M}_8(\mathbf{F}_2)$ et b un vecteur fixe de $\mathbf{F}_2^8$.

Tout le déroulement de l'AES suit le schéma général d'un SPN, hormis le tour final qui n'inclut pas la fonction *MixColumns*.

De nombreuses attaques ont été proposées sur des versions réduites de l'AES (en réduisant le nombre de tours). À ce jour la meilleure attaque connue sur l'AES complet (Bogdanov, Khovratovich, Rechberger, 2011) ne gagne qu'un facteur 4 sur la recherche exhaustive (donc 2^{126} opérations pour une clef de 128 bits).

Chapitre VIII

Cryptanalyses linéaire et différentielle

1. Cryptanalyse linéaire

On doit la cryptanalyse à Matsui en 1992. C'est une attaque à clairs connus sur les chiffrements par bloc itératifs. Le **principe général** consiste à approcher le chiffrement par des équations linéaires. Les équations qui relient bits de message clair, de message chiffré et bits de clef (celles que l'on exhibe lors d'une attaque algébrique) sont de haut degré. Cependant, on peut écrire des équations linéaires qui vont être parfois vérifiées suivant les messages clairs considérées.

Soit $x = (x_1, \dots, x_n) \in \mathbf{F}_2^n$ un bloc de n bits. Une équation linéaire faisant intervenir ces bits peut s'écrire comme le produit scalaire $\langle \alpha, x \rangle = \alpha_1 x_1 \oplus \cdots \oplus \alpha_n x_n$ avec $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbf{F}_2^n$. Si x est choisi avec équiprobabilité dans $\mathbf{F}_2^n$, l'équation $\langle \alpha, x \rangle = 0$ est vérifiée avec probabilité $\frac{1}{2}$ quelque soit $\alpha \neq 0$. En effet, la forme linéaire $f : \mathbf{F}_2^n \to \mathbf{F}_2, x \mapsto \langle \alpha, x \rangle$ étant non nulle, son noyau est de dimension $n-1$: le théorème du rang donne $n = dim(\mathrm{Im} f) + dim(ker\, f)$. La moitié des x donne donc $\langle \alpha, x \rangle = 0$.

On note $c = \mathsf{Encrypt}_{sk}(m)$. On cherche donc des équations du type $\langle \alpha, m \rangle \oplus \langle \beta, c \rangle \oplus \langle \gamma, sk \rangle = 0$ vraies avec probabilité $\frac{1}{2} + \epsilon$, avec $\epsilon > 0$, si m est choisi avec probabilité uniforme. La connaissance d'un telle équation et de sa probabilité permet déjà d'obtenir une équation linéaire sur les bits de sk. On suppose connaître des couples (m, c). Pour chacun de ses couples, on calcule $\langle \alpha, m \rangle \oplus \langle \beta, c \rangle$. Si le résultat vaut plus souvent 0, alors on en déduit que $\langle \gamma, sk \rangle = 0$ car l'équation $\langle \alpha, m \rangle \oplus \langle \beta, c \rangle \oplus \langle \gamma, sk \rangle = 0$ est vraie avec probabilité $\frac{1}{2} + \epsilon$, $\epsilon > 0$.

Une attaque plus pertinente consiste à utiliser de telles équations comme distingueur pour deviner la clef de **dernier tour** d'un chiffrement par bloc itératif. On note toujours

$$m = x_0 \xrightarrow{\mathrm{F}_{\mathrm{K}_0}} x_1 \xrightarrow{\mathrm{F}_{\mathrm{K}_1}} x_2 \longrightarrow \cdots \longrightarrow x_{r-1} \xrightarrow{\mathrm{F}_{\mathrm{K}_{r-1}}} x_r = c$$

et on suppose connaître une équation $\langle \alpha, m \rangle \oplus \langle \beta, x_{r-1} \rangle = 0$ vraie avec probabilité $\mathrm{P}_{\alpha,\beta} = \frac{1}{2} + \epsilon$, $\epsilon \neq 0$ (on verra dans la suite comment trouver une telle équation, et notamment comment faire pour ne plus dépendre de la clef sk). Lors d'une attaque à clairs connus, on récupère un grand nombre de couples clairs chiffrés, (m, c). Pour ces couples, on doit avoir $\langle \alpha, m \rangle \oplus \langle \beta, x_{r-1} \rangle = 0$ avec probabilité $\mathrm{P}_{\alpha,\beta}$. Pour chaque couple (m, c), on remonte le dernier tour, en calculant pour toutes les valeurs possibles K de la clef de dernier tour K_{r-1},

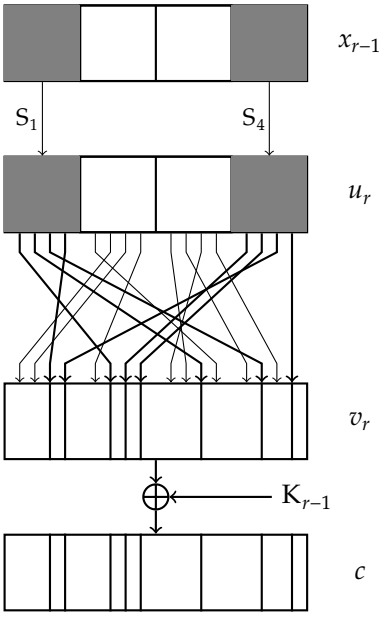
$$z_{\mathrm{K}} = \mathrm{F}_{\mathrm{K}}^{-1}(c)$$

Si $\langle \alpha, m \rangle \oplus \langle \beta, z \rangle = 0$, on incrémente un compteur pour la clef K. Pour la « bonne » clef, $\mathrm{K} = \mathrm{K}_{r-1}$, le compteur doit être incrémenté avec probabilité $\mathrm{P}_{\alpha,\beta} = \frac{1}{2} + \epsilon$. Pour les mauvaises, les valeurs z_{K} sont indépendantes

de m et on s'attend à avoir $\langle \alpha, m\rangle \oplus \langle \beta, z\rangle = 0$ avec probabilité $\frac{1}{2}$. Le compteur le plus éloigné de N/2 si N est le nombre de couples utilisés doit nous permettre de retrouver K_{r-1}.

Cette approche se heurte à un premier problème : la clef du dernier tour, K_{r-1} est en général de taille similaire à la clef secrète. Le coût de cette attaque est donc similaire à la recherche exhaustive. Cependant si β ne s'étend pas sur tout le bloc, on peut limiter la recherche sur seulement quelques bits de K_{r-1}. On voit cela dans le **cadre d'un schéma SPN**. On appelle **boîtes actives** les boîtes S du dernier tour dont au moins un bit d'entrée intervient dans l'équation. Ainsi, pour vérifier si $\langle \alpha, m\rangle \oplus \langle \beta, x_{r-1}\rangle = 0$, il suffit de le vérifier au niveau de l'entrée des boîtes actives. La sortie de ces boîtes est diffusée par la permutation P sur plusieurs bit du bloc. On note I le sous ensemble de $\{1, \dots, n\}$ correspondant à ces bits. L'attaque devient donc : pour chaque couple (m, c), on remonte partiellement le dernier tour, en calculant l'entrée des boîtes actives pour toutes les valeurs possibles des bits d'indices I de la clef de dernier tour K_{r-1}. Puis on teste l'équation grâce au résultat de ce calcul.

Sur l'exemple suivant, on a $n = 4\times 4$, et l'élément β a son support inclus dans $\{1, 2, 3, 4, 13, 14, 15, 16\}$ et seules les boîtes S_1 et S_4 sont actives. Après la permutation P, la sortie de S_1 et S_4 est diffusée sur les bits d'indices $I = \{3, 4, 6, 7, 8, 11, 14, 16\}$. Il suffira donc de faire une recherche exhaustive sur ces indices de la clef de dernier tour K_{r-1}.



Il nous reste à voir comment établir une telle équation $\langle \alpha, m\rangle \oplus \langle \beta, x_{r-1}\rangle = 0$ et sa probabilité. On le voit toujours dans le cadre d'un SPN, en observant l'évolution d'une équation pour chaque étape élémentaire. L'étape de **permutation** des bits du bloc, P, va permuter les bits utilisés dans l'équation. On suppose avoir établi une équation $\langle \alpha, m\rangle \oplus \langle \beta, u\rangle = 0$ vérifiée avec probabilité $\frac{1}{2} + \epsilon$. On note $v = P(u)$. On aura $\langle \alpha, m\rangle \oplus \langle P(\beta), v\rangle = 0$ vérifiée toujours avec probabilité $\frac{1}{2} + \epsilon$. En effet, si non note toujours P la permutation des indices $\{1, \dots, n\}$, on a pour tout $i \in \{1, \dots, n\}$, $u_i = v_{P(i)}$. On a $\langle \beta, u\rangle = \sum_i \beta_i u_i = \sum_i \beta_i v_{P(i)} = \sum_i \beta_{P^{-1}(i)} v_i = \sum_i P(\beta)_i v_i = \langle P(\beta), v\rangle$.

Pour l'**ajout de clef**, supposons que l'on ait $\langle \alpha, x\rangle \oplus \langle \beta, v\rangle = 0$ vérifiée avec probabilité $\frac{1}{2} + \epsilon$. Soit $z = v \oplus K$. Alors l'équation devient $\langle \alpha, x\rangle \oplus \langle \beta, z\rangle \oplus \langle \beta, K\rangle = 0$, toujours vraie avec même probabilité. La quantité $\langle \beta, K\rangle$ est fixe et ne dépend pas du choix de l'entrée. Si $\langle \beta, K\rangle = 0$, l'équation $\langle \alpha, x\rangle \oplus \langle \beta, z\rangle = 0$ est vraie avec probabilité $\frac{1}{2} + \epsilon$ sinon, si $\langle \beta, K\rangle = 1$,cette équation est vraie avec probabilité $1 - (\frac{1}{2} + \epsilon) = \frac{1}{2} - \epsilon$. Au final, quelque soit la clef, l'équation $\langle \alpha, x\rangle \oplus \langle \beta, z\rangle = 0$ est vérifiée avec probabilité $\frac{1}{2} \pm \epsilon$.

L'étape de **substitution** va être le cœur du calcul de l'approximation linéaire. Soit S une boîte S de $\mathbf{F}_2^s \to \mathbf{F}_2^s$. On établit la matrice L à coefficients entiers de taille $2^s \times 2^s$:

$$L(\alpha, \beta) = Card\left\{x \in \mathbf{F}_2^s \mid \langle \alpha, x\rangle \oplus \langle \beta, S(x)\rangle = 0\right\},$$

où $\alpha, \beta \in \mathbf{F}_2^s$ sont identifiés avec les entiers de 0 à $2^s - 1$ pour les indices de positions dans la matrice. De cette matrice, on déduira les probabilités

$$p_{\alpha,\beta} = Pr\Big[\langle \alpha, x\rangle \oplus \langle \beta, S(x)\rangle = 0\Big] = \frac{L(\alpha,\beta)}{2^s}.$$

Une fois calculées les matrices L pour chaque boîte S intervenant dans un SPN, on peut calculer la propagation au travers de l'étape de confusion à chaque tour. Ainsi, si $x = x^{(1)}|x^{(2)}|\dots|x^{(\ell)}$ est transformé en $u = S_1(x^{(1)})|S_2(x^{(2)})|\dots|S_\ell(x^{(\ell)})$ et si $\alpha = \alpha^{(1)}|\dots|\alpha^{(\ell)}$ et $\beta = \beta^{(1)}|\dots|\beta^{(\ell)}$ alors l'équation $\langle \alpha, x\rangle \oplus \langle \beta, u\rangle = 0$ est la somme des équations $\langle \alpha^{(j)}, x^{(j)}\rangle \oplus \langle \beta^{(j)}, u^{(j)}\rangle = 0$ pour $j = 1, \dots, \ell$. Chaque sous bloc évoluant de manière indépendante, on calcule la probabilité de cette somme d'équations en utilisant le lemme d'empilement (*Piling-up Lemma*) suivant.

Lemme VIII – 1 (*Piling-up Lemma*)**.** Soit $Z_1, Z_2, \dots, Z_r$ des variables aléatoires binaires indépendantes telles que $Pr[Z_i = 0] = \frac{1}{2} \pm \epsilon_i$, pour tout $i \in \{1, \dots, r\}$. Alors,

$$Pr[Z_1 \oplus \cdots \oplus Z_r = 0] = \frac{1}{2} \pm 2^{r-1} \prod_{i=1}^{r} \epsilon_i.$$

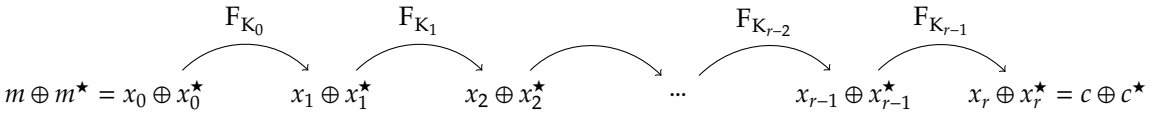
Démonstration. Ce lemme se démontre par récurrence. La démonstration repose sur le cas de deux variables. Pour avoir $Z_1 \oplus Z_2 = 0$, on a deux cas : $Z_1 = Z_2 = 0$ et $Z_1 = Z_2 = 1$. Comme Z_1 et Z_2 sont indépendantes, ceci arrive avec probabilité $(\frac{1}{2} + \epsilon_1)(\frac{1}{2} + \epsilon_2) + (\frac{1}{2} - \epsilon_1)(\frac{1}{2} - \epsilon_2) = \frac{1}{2} + 2\epsilon_1\epsilon_2$. □

Au final, on est capable de calculer des approximations linéaires pour chaque tour, du type $\langle \alpha_{i-1}, x_{i-1}\rangle \oplus \langle \alpha_i, x_i\rangle = 0$ vérifiée avec probabilité $\frac{1}{2} \pm \epsilon_i$. Pour en déduire la probabilité de l'équation $\langle \alpha, m\rangle \oplus \langle \beta, x_{r-1}\rangle = 0 = \langle \alpha_0, x_0\rangle \oplus \langle \alpha_{r-1}, x_{r-1}\rangle$, on réutilise le lemme d'empilement, sous l'hypothèse d'indépendance des approximations de tours. En effet, en sommant chaque approximation, on trouve $\langle \alpha, m\rangle \oplus \langle \beta, x_{r-1}\rangle = 0 = \langle \alpha_0, x_0\rangle \oplus \langle \alpha_{r-1}, x_{r-1}\rangle$ avec probabilité $\frac{1}{2} \pm 2^{r-1} \prod_{i=1}^{r} \epsilon_i$.

2. Cryptanalyse différentielle

On doit la cryptanalyse différentielle à Biham et Shamir en 1990 mais elle était connue des concepteurs du DES dans les années 70. C'est une attaque à messages clairs choisis, sur les constructions itératives de chiffrement par bloc.

Le **principe général** est le suivant. On considère deux messages clairs m et $m^\star$, et on appelle différence la somme $\alpha := m \oplus m^\star$. On s'intéresse à l'évolution des différences à chaque tour (appelée chemin différentiel).



Si la fonction de tour était linéaire, on aurait à chaque tour $x_{i+1} \oplus x_{i+1}^\star = F_{K_i}(x_i) \oplus F_{K_i}(x_i^\star) = F_{K_i}(x_i \oplus x_i^\star)$. Ainsi, quelque soit $m, m^\star$ satisfaisant une différence α en entrée, on aurait toujours le même chemin différentiel, $\alpha, F_{K_0}(\alpha), F_{K_1}(F_{K_0}(\alpha)), \dots$. Comme cette fonction de tour n'est pas linéaire, un tel chemin différentiel ne va pas être suivi avec probabilité 1 (sur les choix de $m, m^\star$ tel que $m \oplus m^\star = \alpha$) mais va être suivi avec probabilité plus ou moins grande.

On s'intéresse en particulier à la probabilité

$$P_{\alpha,\beta} = Pr\Big[x_{r-1} \oplus x_{r-1}^\star = \beta \mid m \oplus m^\star = \alpha\Big]$$

et on suppose connaître (α, β) tel que $P_{\alpha,\beta}$ est élevée (éloignée de l'uniforme, $\frac{1}{2^n}$ où n est la taille des blocs). On se sert de cette probabilité pour tester des candidats pour la clef K_{r-1} du dernier tour comme pour la cryptanalyse linéaire.

Lors d'une attaque à clairs choisis, on récupère un grand nombre de couples clairs chiffrés, (m, c) et $(m^\star, c^\star)$ tel que $m \oplus m^\star = \alpha$. Pour ces couples, on doit avoir la différence à l'entrée du dernier tour, $x_{r-1} \oplus x^\star_{r-1}$ égale à β avec probabilité $P_{\alpha,\beta}$. Pour chaque couple (m, c) et $(m^\star, c^\star)$, on remonte le dernier tour, en calculant pour toutes les valeurs possibles K de la clef de dernier tour K_{r-1},

$$z_K = F_K^{-1}(c) \quad \text{et} \quad z^\star_K = F_K^{-1}(c^\star).$$

Si $z_K \oplus z^\star_K = \beta$, on incrémente un compteur pour la clef K. Pour la « bonne » clef, $K = K_{r-1}$, le compteur doit être incrémenté avec probabilité $P_{\alpha,\beta}$. Pour les mauvaises, les valeurs z_K et $z^\star_K$ sont indépendantes de m et $m^\star$ et on s'attend à avoir $z_K \oplus z^\star_K = \beta$ avec probabilité $\frac{1}{2^n}$.

Comme pour la cryptanalyse linéaire, on ne fait pas une recherche exhaustive sur toute la clef du dernier tour, K_{r-1}. On choisit un β donnant peu de boîtes actives et on se limite à vérifier si $x_{r-1} \oplus x^\star_{r-1} = \beta$ au niveau de l'entrée de celles ci.

Pour calculer la probabilité $P_{\alpha,\beta}$, toujours dans le cas d'un SPN, on la calcule en observant la propagation des différences sur les étapes élémentaires comme pour la cryptanalyse linéaire. Pour les étapes linéaires, de type $v = P(u)$, on a déjà vu qu'une différence α devient $\beta = P(\alpha)$ avec probabilité 1. Pour l'ajout de la clef de tour, $x = v \oplus K$, une différence α reste inchangée, donc $\beta = \alpha$ avec probabilité 1. En particulier les probabilités de propagation des différences ne dépendent pas de la clef secrète utilisée, elles ne dépendent que de l'algorithme de chiffrement et peuvent donc être pré calculées.

Le calcul des propagations des différences au travers des boîtes S va être le cœur du calcul d'un chemin différentiel. Soit S une boîte S de $\mathbf{F}_2^s \to \mathbf{F}_2^s$. On calcule la matrice D à coefficients entiers de taille $2^s \times 2^s$:

$$D(\alpha, \beta) = Card\Big\{(x, x^\star) \in \mathbf{F}_2^s \times \mathbf{F}_2^s \mid x \oplus x^\star = \alpha \text{ et } S(x) \oplus S(x^\star) = \beta\Big\},$$

où $\alpha, \beta \in \mathbf{F}_2^s$ sont identifiés avec les entiers de 0 à $2^s - 1$ pour les indices de positions dans la matrice. De cette matrice, on déduira les probabilités

$$p_{\alpha,\beta} = Pr\Big[S(x) \oplus S(x^\star) = \beta \mid x \oplus x^\star = \alpha\Big] = \frac{D(\alpha, \beta)}{2^s}.$$

Une fois calculées les matrices D pour chaque boîte S intervenant dans un SPN, on peut calculer la propagation au travers de l'étape de confusion à chaque tour. Ainsi, si $x = x^{(1)}|x^{(2)}|\ldots|x^{(\ell)}$ est transformé en $u = S_1(x^{(1)})|S_2(x^{(2)})|\ldots|S_\ell(x^{(\ell)})$ et si $\alpha = \alpha^{(1)}|\ldots|\alpha^{(\ell)}$ et $\beta = \beta^{(1)}|\ldots|\beta^{(\ell)}$ alors on vérifie facilement, chaque sous bloc évoluant de manière indépendante que

$$Pr[u \oplus u^\star = \beta \mid x \oplus x^\star = \alpha] = \prod_{j=1}^{\ell} Pr[u^{(j)} \oplus u^{\star(j)} = \beta^{(j)} \mid x^{(j)} \oplus x^{\star(j)} = \alpha^{(j)}].$$

Au final, on est capable de calculer des probabilités de propagation pour chaque tour, du type $P_{\alpha_{i-1},\alpha_i} = Pr[x_i \oplus x^\star_i = \alpha_i \mid x_{i-1} \oplus x^\star_{i-1} = \alpha_{i-1}]$. Pour calculer $Pr\Big[x_{r-1} \oplus x^\star_{r-1} = \beta \mid m \oplus m^\star = \alpha\Big]$, Biham et Shamir font l'hypothèse que les tours sont indépendants (les clefs de tours rajoutant de l'aléa). Cette hypothèse est assez bien vérifiée en pratique. De plus, plutôt que de considérer toutes les possibilités de chemins différentiels donnant $m \oplus m^\star = \alpha$ puis $x_{r-1} \oplus x^\star_{r-1} = \beta$, on s'intéresse à un chemin particulier, du type

$$\alpha = \alpha_0 \xrightarrow{F_{K_0}} \alpha_1 \xrightarrow{F_{K_1}} \alpha_2 \longrightarrow \ldots \xrightarrow{F_{K_{r-2}}} \alpha_{r-1} = \beta$$

Sous l'hypothèse d'indépendance des tours, un tel chemin se produit avec probabilité

$$\prod_{i=1}^{r-1} \mathrm{P}_{\alpha_{i-1},\alpha_i} = \prod_{i=1}^{r-1} \Pr[x_i \oplus x_i^\star = \alpha_i \mid x_{i-1} \oplus x_{i-1}^\star = \alpha_{i-1}],$$

ce qui minore $\Pr\left[x_{r-1} \oplus x_{r-1}^\star = \beta \mid m \oplus m^\star = \alpha\right]$.

Chapitre IX

Réseaux euclidiens et cryptanalyse

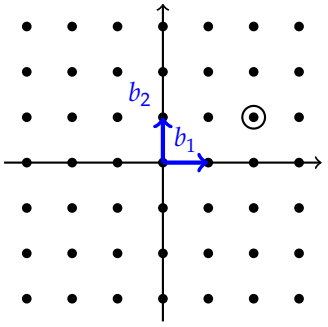
1. Réseaux euclidiens

Un **réseau** (*lattice*) est un arrangement régulier de points de $\mathbf{R}^n$, muni du produit scalaire défini par : pour $x, y \in \mathbf{R}^n$, $\langle x, y\rangle = \sum_{i=1}^n x_i y_i$. La norme euclidienne est $\|x\| = \sqrt{\langle x, x\rangle} = \sqrt{\sum_{i=1}^n x_i^2}$.

Plus précisément, un réseau L est un sous groupe discret de $(\mathbf{R}^n, +)$, c'est à dire que $\mathrm{L} \neq \emptyset$, si $v \in \mathrm{L}$ alors $-v \in \mathrm{L}$, si $v_1, v_2 \in \mathrm{L}$ alors $v_1 + v_2 \in \mathrm{L}$. En particulier, le vecteur nul appartient à L et L est stable par combinaisons linéaires à coefficients entiers : si $b_1, b_2, \dots, b_k \in \mathrm{L}$ alors $\sum \lambda_i b_i \in \mathrm{L}$ avec $\lambda_i \in \mathbf{Z}$. Discret signifie que L n'a pas de point d'accumulation : pour tout $v \in \mathrm{L}$, il existe une boule B de $\mathbf{R}^n$ assez petite telle $\mathrm{B} \cap \mathrm{L} = \{v\}$. Par exemple $0 \cup \{1/n, n \in \mathbf{N}^*\}$ n'est pas discret.

Exemple. $\mathbf{Z}$ est un réseau de $\mathbf{R}$:

C'est aussi un réseau de $\mathbf{R}^2$, de même $\mathbf{Z}^2$ est un réseau de $\mathbf{R}^2$:



De manière générale, pour tout $d \leqslant n$, $\mathbf{Z}^d$ est un réseau de $\mathbf{R}^n$.

Tout réseau peut être représenté par une **base**. Si L est un réseau de $\mathbf{R}^n$, il existe d vecteurs linéairement indépendants de R^n avec $d \leqslant n$, $b_1, b_2, \dots, b_d$, tels que

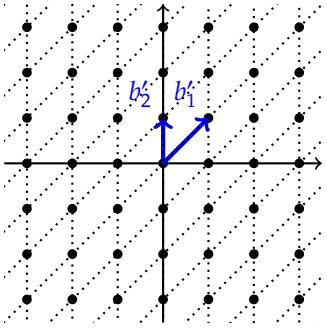
$$\mathrm{L} = \left\{ v \in \mathbf{R}^n, v = \sum_{i=1}^{d} \lambda_i b_i, \lambda_i \in \mathbf{Z} \right\}$$

On note $\mathscr{B} = (b_1, \dots, b_d)$ une base du réseau (non unique). Toutes les bases de L ont le même nombre d'éléments appelé **dimension du réseau**. On note $dim\,\mathrm{L} = d$. Si $d = n$ on dit que L est de rang plein. Pour tout $v \in \mathrm{L}$, il existe une unique combinaison linéaire $\lambda_1, \dots, \lambda_d \in \mathbf{Z}$, $v = \sum_{i=1}^d \lambda_i b_i$.

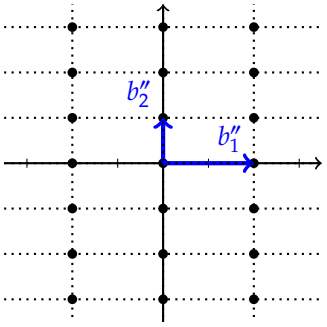
Pour les applications cryptographiques, en général, les b_i sont à coordonnées entières. Un réseau sera représenté par sa base, sous forme d'une matrice $d \times n$ à coefficients entiers dont les lignes expriment les coordonnées des b_i dans la base canonique.

Si B et B′ sont deux matrices représentant deux bases d'un même réseau de dimension d alors il existe une matrice $d \times d$ à coefficients entiers P avec $det\,\mathrm{P} = \pm 1$ telle que $\mathrm{B}' = \mathrm{PB}$. Si $\mathscr{B} = (b_1, \dots, b_d)$ est une base, les opérations élémentaires qui permettent de conserver une base sont $b_i \leftarrow -b_i$, $b_i \leftarrow b_i + kb_j$ avec $k \in \mathbf{Z}$ et $j \neq i$ et enfin l'échange de deux vecteurs $b_i \leftrightarrow b_j$.

Exemple. Sur l'exemple du réseau $\mathbf{Z}^2$ inclus dans $\mathbf{R}^2$, $\mathscr{B} = (b_1, b_2)$ avec $b_1 = \begin{pmatrix}1\\0\end{pmatrix}$, $b_2 = \begin{pmatrix}0\\1\end{pmatrix}$ est une base représentée par la matrice $\mathrm{B} = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$. On note $b'_1 = b_1 + b_2$, $b'_2 = b_2$. La matrice $\mathrm{B}' = \begin{pmatrix}1 & 1\\0 & 1\end{pmatrix}$ représente le même réseau :



On a $\mathrm{B}' = \mathrm{PB}$ avec $\mathrm{P} = \begin{pmatrix}1 & 1\\0 & 1\end{pmatrix}$. Par contre la base $\mathrm{B}'' = \begin{pmatrix}2 & 0\\0 & 1\end{pmatrix}$ ne donne pas le même réseau :



Soit B une matrice représentant une base $(b_1, \dots, b_d)$ d'un réseau L. La **matrice de Gram** de L est

$$\mathrm{G} = \mathrm{B}\,\mathrm{B}^{\mathrm{T}} = \left(\langle b_i, b_j\rangle\right)_{1 \leqslant i,j \leqslant d}$$

Le **déterminant de Gram** est $det\ \mathrm{G}$, il est positif et non nul. On appelle $det\,\mathrm{L} = \sqrt{det\,\mathrm{G}}$ le **déterminant du réseau** L. C'est le volume du parallélotope formé par les vecteurs de base : $\{\sum_{i=1}^{d} t_i b_i,\ \forall i, 0 \leq t_i \leq 1, t_i \in \mathbf{R}\}$.

On remarque que ce déterminant ne dépend pas de la base choisie : si $\mathrm{B}' = \mathrm{PB}$, $det\ \mathrm{G}' = det(\mathrm{PBB}^{\mathrm{T}}\mathrm{P}^{\mathrm{T}}) = (det\,\mathrm{P})^2\ det\,\mathrm{G} = det\,\mathrm{G}$. D'autre part, si L est de rang plein B est une matrice carrée et $det\ \mathrm{G} = (det\,\mathrm{B})^2$. Dans ce cas, on a donc $det\,\mathrm{L} = \sqrt{(det\,\mathrm{B})^2} = |det\,\mathrm{B}|$.

Sur le dernier exemple, le premier réseau était de déterminant 1 alors que le second était de déterminant 2.

Exemple. Soit

$$\mathrm{B} = \begin{pmatrix}1 & 2\\0 & 3\end{pmatrix}, \mathrm{B}' = \begin{pmatrix}-1 & 1\\1 & 2\end{pmatrix},$$

deux bases respectives de réseaux L et L′ de $\mathbf{R}^2$. Exercice : montrer que $\mathrm{L} = \mathrm{L}'$.

Soit L un réseau et soit M un réel strictement positif. Comme L est discret, l'ensemble $\{x \in L \setminus \{0\}, \|x\| \leqslant M\}$ est fini. Il existe donc un plus petit vecteur non nul. Sa norme est appelée **minimum du réseau**. On note $\lambda_1(L) = min_{x \in L \setminus \{0\}} \|x\|$. Étant donné une base d'un réseau, trouver un vecteur (il peut y en avoir plusieurs) qui atteint le minimum est appelé problème SVP (*Shortest Vector Problem*). Ce problème a été prouvé NP-difficile par Ajtai en 1996.

Exemple. On a $\lambda_1(\mathbf{Z}^2) = 1$ atteint par $(1,0), (0,1), (-1,0), (0,-1)$. Dans l'exemple précédent, le minimum était $\sqrt{2}$ atteint par $(-1,1)$ et $(1,-1)$.

La deuxième base B′ obtenue dans cet exemple avait de meilleures propriétés que la première base B : ces vecteurs étaient plus orthogonaux et de plus elle contient un vecteur atteignant le minimum du réseau. Dans la suite, on va s'intéresser à un procédé algorithmique permettant d'obtenir une « bonne base » telle que la base B′ à partir d'une base quelconque.

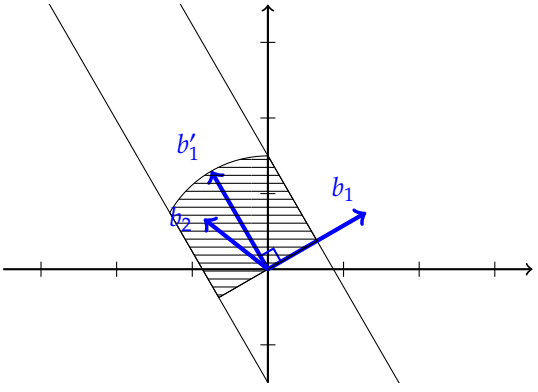
D'autres problèmes interviennent en cryptographie basée sur les réseaux, notamment le problème CVP (*Closest Vector Problem*) : étant donnés une base d'un réseau, et un vecteur t (non nécessairement dans le réseau), trouver un vecteur du réseau le plus proche de t. Ce problème est aussi NP-difficile (van Emde Boas 1981). On peut réduire SVP à CVP.

2. Réduction de réseau

La réduction de réseau consiste à partir d'une base donnée d'un réseau à obtenir une base constituée de vecteurs le plus court possible et aussi orthogonaux que possible. Elle vise donc en particulier à résoudre ou du moins à approcher le problème SVP. En dimension 2, on va voir qu'il existe un procédé polynomial. En dimension quelconque, il va falloir relâcher les propriétés d'une « bonne base » pour garder un algorithme polynomial.

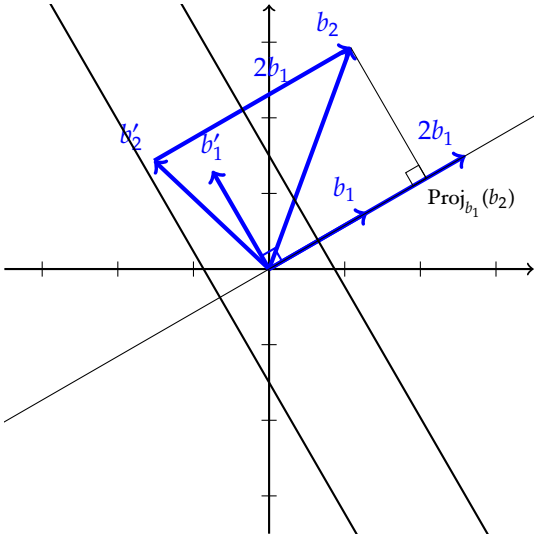
2.1. Cas de la dimension 2 (Lagrange - Gauss)

On veut obtenir une base le plus proche possible d'une base orthogonale avec des vecteurs le plus court possible. Plus précisément, soit L un réseau de dimension 2 engendré par des vecteurs $b_1, b_2 \in \mathbf{R}^2$. On dira que la base (b_1, b_2) est **réduite au sens de Lagrange - Gauss** (1773-1801) si $\|b_1\| \leqslant \|b_2\|$ et si (b_1, b_1') est une base orthogonale directe de $\mathbf{R}^2$ alors $b_2 \in \{t_1 b_1 + t_2 b_1', t_1, t_2 \in \mathbf{R}, |t_1| \leqslant \frac{1}{2}, t_2 > 0\}$.



On peut montrer que si (b_1, b_2) est réduite au sens de Lagrange - Gauss, alors b_1 est un vecteur minimal de L, autrement dit, $\lambda_1(L) = \|b_1\|$.

Si (b_1, b_2) est une base de L, on peut « ramener » b_2 dans $\{t_1 b_1 + t_2 b_1', t_1, t_2 \in \mathbf{R}, |t_1| \leqslant \frac{1}{2}, t_2 > 0\}$ en lui retranchant un multiple de b_1. Comme (b_1, b_1') est une base de $\mathbf{R}^2$, il existe deux réels, t_1, t_2 tel que $b_2 = t_1 b_1 + t_2 b_1'$. On a $\langle b_1, b_2 \rangle = t_1 \langle b_1, b_1 \rangle$. On a donc $t_1 = \langle b_1, b_2 \rangle / \langle b_1, b_1 \rangle$. On pose $b_2' = b_2 - \lfloor t_1 \rceil b_1$ (entier le plus proche) qui est bien un point de L. Ainsi, $b_2' = (t_1 - \lfloor t_1 \rceil) b_1 + t_2 b_1'$ et $|t_1 - \lceil t_1 \rceil| \leqslant \frac{1}{2}$. Notons que $t_1 b_1$ est en fait la projection orthogonale de b_2 sur la droite engendrée par b_1.



L'algorithme de réduction de Lagrange-Gauss consiste à répéter cette transformation jusqu'à avoir $\|b_1\| \leqslant \|b_2\|$. On peut voir cette algorithme comme une généralisation vectorielle de l'algorithme d'Euclide.

Algorithme de Lagrange-Gauss

Entrée : (b_1, b_2) une base de L avec $\|b_1\| \leqslant \|b_2\|$
Sortie : une base de L réduite au sens de Lagrange - Gauss

Répéter

$b_2 \leftarrow b_2 - \left\lfloor \frac{\langle b_1, b_2 \rangle}{\langle b_1, b_1 \rangle} \right\rceil b_1$

Si $\|b_1\| > \|b_2\|$, échanger $b_1 \leftrightarrow b_2$

Sinon **Retourner** (b_1, b_2) ou éventuellement $(b_1, -b_2)$

Exemple. On part d'une base $b_1 = (6, 1), b_2 = (10, 3)$. Appliquer l'algorithme de Lagrange-Gauss.

Il est clair que l'algorithme de Lagrange-Gauss retourne bien une base réduite. On peut montrer que sa complexité est quadratique en la taille des coefficients de l'entrée.

2.2. En dimension supérieure

Comme en dimension deux, on va définir une notion de base réduite en comparant avec une base orthogonale. Pour cela on va donner des conditions sur les coefficients d'orthogonalisation de Gram-Schmidt. On rappelle d'abord le procédé de Gram-Schmidt.

Orthogonalisation de Gram-Schmidt

Soit $(b_1, \dots, b_d)$ une famille libre de d vecteurs de $\mathbf{R}^n$. On pose

$$\begin{cases} b_1^\star &= b_1 \\ b_i^\star &= b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^\star \text{ avec } \mu_{i,j} = \frac{\langle b_i, b_j^\star \rangle}{\langle b_j^\star, b_j^\star \rangle} \text{ pour } 1 \leqslant j < i, 2 \leqslant i \leqslant d \\ &= b_i - \sum_{j=1}^{i-1} \mathrm{Proj}_{b_j^\star}(b_i) \end{cases}$$

On pose de plus $\mu_{i,i} = 1$ pour $i = 1, \dots, d$.

Les $\mu_{i,j}$ sont les **coefficients de l'orthogonalisation**. La famille $(b_1^\star, \dots, b_d^\star)$ est **l'orthogonalisé de Gram-Schmidt** de $(b_1, \dots, b_d)$. Elle est caractérisée par

- si $i \neq j$, $\langle b_i^\star, b_j^\star \rangle = 0$.
- les espaces vectoriels engendrés par $(b_1^\star, \dots, b_i^\star)$ et $(b_1, \dots, b_i)$ sont égaux pour $1 \leqslant i \leqslant d$;
- le vecteur $b_i^\star - b_i$ est dans l'espace vectoriel engendré par $(b_1, \dots, b_{i-1})$, pour $2 \leqslant i \leqslant d$.

Attention, $(b_1^\star, \dots, b_d^\star)$ et $(b_1, \dots, b_d)$ engendrent le même $\mathbf{R}$–espace vectoriel mais pas forcément le même réseau.

Soit $(b_1, \dots, b_d)$ une base d'un réseau L. On dit que $(b_1, \dots, b_d)$ satisfait la **condition de taille** si les coefficients de son orthogonalisation de Gram-Schmidt vérifient :

$$|\mu_{i,j}| \leqslant \frac{1}{2}, \forall 1 \leqslant j < i \leqslant d.$$

Comme en dimension 2, on va pouvoir en temps polynomial trouver une base satisfaisant la condition de taille en temps polynomial (par des opérations du type, $b_i \leftarrow b_i - \lfloor \mu_{i,j} \rceil b_j$).

La condition de réduction de Lagrange-Gauss se traduit en terme de l'orthogonalisation de Gram-Schmidt. Soit (b_1, b_2) réduite au sens de Lagrange - Gauss : $\|b_1\| \leqslant \|b_2\|$ et si (b_1, b_1') est une base orthogonale directe de $\mathbf{R}^2$ alors $b_2 = t_1 b_1 + t_2 b_1'$, $t_1, t_2 \in \mathbf{R}$, $|t_1| \leqslant \frac{1}{2}$, $t_2 > 0$.

On pose $b_1^\star = b_1$ et $b_2^\star = t_2 b_1'$. Ainsi $b_2^\star = b_2 - t_1 b_1^\star$ et $(b_1^\star, b_2^\star)$ est l'orthogonalisé de (b_1, b_2). En posant $\mu_{2,1} = t_1$ on obtient donc comme condition $|\mu_{2,1}| \leqslant \frac{1}{2}$: on retrouve la condition de taille.

La deuxième condition $\|b_1\| \leqslant \|b_2\|$ se traduit par $\|b_1^\star\|^2 \leqslant \mu_{2,1}^2 \|b_1^\star\|^2 + \|b_2^\star\|$ soit $\|b_1^\star\|^2(1 - \mu_{2,1}^2) \leqslant \|b_2^\star\|^2$. Comme $|\mu_{2,1}| \leqslant \frac{1}{2}$, on obtient $(1 - \mu_{2,1}^2) \geqslant \frac{3}{4}$, ce qui donne

$$\frac{3}{4}\|b_1^\star\|^2 \leqslant \|b_2^\star\|^2.$$

Pour réduire la taille des vecteurs en dimension supérieure comme en dimension 2, l'idée est de faire des échanges tant que la base n'est pas assez réduite. Mais pour établir un algorithme polynomial, il ne fait pas faire trop d'échanges, et donc établir une condition assez lâche. Au rang i avec $1 < i \leqslant d$, on a $b_i^\star = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^\star$.

Supposons que l'on échange les vecteurs b_{i-1} et b_i. Les vecteurs de la base orthogonalisée vont changer à partir du rang $i-1$. Le nouveau vecteur au rang $i-1$ va être égal à

$$\mathrm{B}_{i-1}^* := b_i - \sum_{j=1}^{i-2} \mu_{i,j} b_j^\star = b_i^\star + \mu_{i,i-1} b_{i-1}^\star.$$

La condition utilisée dans l'algorithme LLL est de ne pas échanger les vecteurs si

$$\frac{3}{4}\|b_{i-1}^*\|^2 \leqslant \|b_i^\star + \mu_{i,i-1} b_{i-1}^\star\|^2.$$

On voit facilement que cette condition est équivalente à

$$\left(\frac{3}{4} - \mu_{i,i-1}^2\right)\|b_{i-1}^\star\|^2 \leqslant \|b_i^\star\|^2.$$

L'algorithme LLL (Lenstra, Lenstra, Lovász (1982)

Une base $(b_1, \dots, b_d)$ d'un réseau est **LLL réduite** si elle satisfait la **condition de taille** et si son orthogonalisé vérifie $(\frac{3}{4} - \mu_{i,i-1}^2)\|b_{i-1}^\star\|^2 \leqslant \|b_i^\star\|^2$, pour $2 \leqslant i \leqslant d$ (**condition de Lovász**).

Si $(b_1, \dots, b_d)$ est une base LLL réduite d'un réseau L, alors on peut montrer que $\|b_1\| \leqslant 2^{\frac{d-1}{4}} (det\, \mathrm{L})^{\frac{1}{d}}$. De plus, $\|b_1\| \leqslant 2^{\frac{d-1}{2}} \lambda_1(\mathrm{L})$. On obtient donc une approximation exponentielle du minimum de L. En pratique, LLL fait mieux (surtout en petite dimension).

L'algorithme LLL retourne une base LLL réduite à partir d'une base $(b_1, \dots, b_d)$ d'un réseau $\mathrm{L} \subset \mathbf{R}^n$ en temps $\mathscr{O}(d^5 n \log^3 \mathrm{B})$ où $\mathrm{B} = max(\|b_i\|)$, sans arithmétique rapide.

Algorithme LLL (simplifié)

Entrée : $(b_1, \dots, b_d)$ une base de L
Sortie : une base de L, LLL réduite

$k \leftarrow 2$
Tant que $k \leqslant d$
  Modifier b_k pour que $(b_1, \dots, b_k)$ satisfasse la condition de taille
  Si la condition de Lovász est satisfaite au rang k
    alors $k \leftarrow k + 1$
    sinon échanger $b_k \leftrightarrow b_{k-1}$ et $k \leftarrow max(2, k-1)$
Retourner $(b_1, \dots, b_d)$

3. Application à la cryptanalyse

3.1. Cryptanalyse de Merkle-Hellman

Cette cryptanalyse consiste à résoudre un problème de type sac à dos grâce à l'algorithme LLL. Le ***subset sum problem*** consiste étant donné $a_1, \dots, a_n$ et s des entiers naturels à déterminer s'il existe un sous-ensemble $\mathrm{I} \subset \{1, \dots, n\}$ tel que

$$s = \sum_{i \in \mathrm{I}} a_i.$$

Ce problème est NP-complet.

Pour utiliser ce problème pour construire un système de chiffrement à clef publique, il faut une instance faible, c'est à dire une instance que l'on sera résoudre connaissant une trappe, la clef secrète. On dit que $a_1, \dots, a_n$ est à **super-croissance** si $a_j > \sum_{i=1}^{j-1} a_i$, pour $2 \leqslant j \leqslant n$, Dans ce cas, on peut résoudre efficacement le *subset sum problem* avec l'algorithme glouton.

Algorithme glouton pour le *subset sum problem*

Entrée : $(a_1, \dots, a_n)$ et s
Déterminer le plus grand des a_i tel que $a_i \leqslant s$
$s \leftarrow s - a_i$
Recommencer tant que $s \geqslant min(a_i)$

En exercice : Si $(a_1, \dots, a_n)$ est à super-croissance, à la fin de l'algorithme, on a $s \neq 0$ si et seulement si s ne se décompose pas. Si $s = 0$, les a_i utilisés donnent la décomposition.

Le système de chiffrement de Merkle-Hellman (1978) consiste à masquer une suite super-croissante.

Cryptosystème de Merkle-Hellman

Génération des clefs :

n un paramètre de sécurité

Générer aléatoirement $a_1, \dots, a_n$ à super-croissance, avec $2^{n-1} < a_1 < 2^n$ puis $\sum_{i=1}^{j-1} a_i < a_j < 2^{n+j-1}$ pour $j = 2, \dots, n$

Générer aléatoirement N tel que $\sum_{i=1}^{n} a_i < \mathrm{N} < 2^{2n}$

Générer aléatoirement $\mu \in (\mathbf{Z}/\mathrm{N}\mathbf{Z})^\times$

Poser $b_i \equiv a_i\mu \pmod{\mathrm{N}}$ pour $i = 1, \dots, n$

Retourner $(b_1, \dots, b_n)$ comme clef publique et $\mathrm{N}, \mu, (a_1, \dots, a_n)$ comme clef privée

Chiffrement de $m = (m_1, \dots, m_n) \in \{0,1\}^n$

Retourner $c = \sum_{i=1}^{n} m_i b_i$.

Déchiffrement de c

Poser $s = c\mu^{-1} \pmod{\mathrm{N}}$.

Résoudre le ***subset sum problem*** $(a_1, \dots, a_n), s$ avec l'algorithme glouton et retourner la solution

Il est facile de voir que ce chiffrement est consistant. En effet, si c est un chiffré de $(m_1, \dots, m_n)$, on a $c = \sum_{i=1}^{n} m_i b_i \equiv \sum_{i=1}^{n} m_i a_i \mu \pmod{\mathrm{N}}$. Donc $s \equiv c\mu^{-1} \equiv \sum_{i=1}^{n} m_i a_i \pmod{\mathrm{N}}$. Comme $\sum_{i=1}^{n} m_i a_i < \mathrm{N}$ on a en fait $s \mod n = \sum_{i=1}^{n} m_i a_i$. Comme $a_1, \dots, a_n$ est à super-croissance, l'algorithme glouton retourne bien $(m_1, \dots, m_n)$.

On considère une attaque consistant, étant donné un chiffré c et la clef publique $(b_1, \dots, b_n)$ à retrouver le message clair m, c'est à dire retrouver $(m_1, \dots, m_n)$ tel que $c = \sum_{i=1}^{n} m_i b_i$. On doit donc résoudre une instance du ***subset sum problem a priori*** difficile.

On peut montrer que trouver des solutions à de telles instances revient à résoudre un problème de type CVP dans un réseau adéquat. Pour résoudre un problème CVP, une technique heuristique générale due à Kannan consiste à résoudre un problème SVP dans un réseau de dimension plus grande. Ceci mène à la méthode de Lagarias et Odlyzko que l'on détaille dans la suite.

La densité d'une instance du ***subset sum problem*** est défini par

$$d = \frac{n}{\log \max(b_i)}.$$

On sait résoudre presque tous les ***subset sum problem*** en temps polynomial en utilisant LLL s'ils sont à faible densité ($d < 1/n$, Lagarias-Odlyzko 1985). De plus Lagarias-Odlyzko ont montré que pour $d < 0.645$ alors presque tous les ***subset sum problem*** se réduisent à un problème SVP.

Pour le système de Merkle-Hellman on a $d \approx \frac{n}{2n} \approx \frac{1}{2}$. On va voir comment faire notre attaque en cherchant un vecteur court dans un réseau en suivant les idées de Lagarias-Odlyzko.

À c, $(b_1, \dots, b_n)$ on associe le réseau $\mathrm{L} \subset \mathbf{R}^{n+2}$ de dimension $n+1$ engendré par

$$\mathrm{A} := \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & \mathrm{K}b_1 \\ 0 & 1 & 0 & 0 & \dots & 0 & \mathrm{K}b_2 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 & \mathrm{K}b_n \\ 0 & 0 & \dots & 0 & 0 & 1 & -\mathrm{K}c \end{pmatrix}$$

où K est un entier.

Soit $e_1, \dots, e_{n+1}$ les lignes de A. Calculons les coordonnées d'un vecteur $v \in \mathrm{L}$. On a $v = \sum_{i=1}^{n+1} v_i e_i = (v_1, v_2, \dots, v_{n+1}, \mathrm{K}(\sum_{i=1}^{n} v_i b_i - v_{n+1} c))$. On a donc $\|v\|^2 = v_1^2 + v_2^2 + \dots v_{n+1}^2 + \mathrm{K}^2(\sum_{i=1}^{n} v_i b_i - v_{n+1} c)^2$. À une solution du ***subset sum problem***, $c = \sum_{i=1}^{n} m_i b_i$, on fait correspondre le vecteur $m = m_1 e_1 + \dots m_n e_n + e_{n+1} \in \mathrm{L}$, soit $m = (m_1, \dots m_n, 1, 0)$ car $(\sum_{i=1}^{n} m_i b_i - c) = 0$. Ce sont donc des vecteurs courts, $\|m\|^2 \leqslant n+1$.

On pose K tel que $K^2 > n+1$. Soit $v \in L$ et $\|v^2\| \leqslant n+1$ alors on obtient que

$$v_1^2 + v_2^2 + \ldots v_{n+1}^2 + K^2\left(\sum_{i=1}^{n} v_i b_i - v_{n+1}c\right)^2 < n+1.$$

Or $v_1^2 + v_2^2 + \ldots v_{n+1}^2 + K^2(\sum_{i=1}^n v_i b_i - v_{n+1}c)^2 > K^2(\sum_{i=1}^n v_i b_i - v_{n+1}c)^2 > (n+1)(\sum_{i=1}^n v_i b_i - v_{n+1}c)^2$. En combinant les deux inégalités, on obtient que

$$(n+1)\left(\sum_{i=1}^{n} v_i b_i - v_{n+1}c\right)^2 < n+1,$$

soit $(\sum_{i=1}^n v_i b_i - v_{n+1}c)^2 < 1$, donc $\sum_{i=1}^n v_i b_i = v_{n+1}c$ qui sera une solution du ***subset sum problem*** si $v_{n+1} = 1$.

Les solutions du ***subset sum problem*** donnent donc des vecteurs courts et réciproquement des vecteurs courts peuvent donner des solutions. En pratique, on réduit la base A avec LLL et on cherche dans la base obtenue un vecteur qui correspondent à une solution ($v_i \in \{0,1\}$, $v_{n+1} = 1$ et $v_{n+2} = 0$). Cette méthode donne de bons résultats.

3.2. Attaques de Coppersmith

En 1997, Coppersmith a proposé plusieurs attaques, notamment sur RSA, en utilisant des réductions de réseaux. Depuis de nombreux développements de ces attaques ont été proposés en cryptanalyse.

Considérons le problème RSA suivant. On pose $N = pq$ un entier RSA, produit de deux grands nombres premiers distincts. Soit $C \equiv M^e \pmod N$ avec e premier avec $\varphi(N)$. Retrouver M à partir de C revient à trouver une racine au polynôme $x^e - C$ dans $\mathbf{Z}/N\mathbf{Z}$ ce qui est un problème difficile si N n'est pas premier. Si $M < N^{\frac{1}{e}}$, alors $M^e \pmod N = M^e$ dans $\mathbf{Z}$. Il suffit donc de trouver une racine de $x^e - C$ dans $\mathbf{Z}$, ce qu'on sait faire efficacement.

Coppersmith propose d'étendre cette attaque. Supposons que $0 \leqslant M < N$ et que l'on connaît une proportion de $1 - \frac{1}{e}$ bits de poids fort de M (par exemple $e = 3$, et on connaît 2/3 des bits de M). On a alors $M = M_0 + M'$ avec M' connu et M_0 inconnu et $M_0 < N^{\frac{1}{e}}$. L'équation $C \equiv M^e \pmod N$ devient $C \equiv (M_0 + M')^e \pmod N$ et M_0 est une petite racine de $(x + M')^e - C$ dans $\mathbf{Z}/N\mathbf{Z}$. L'algorithme de Coppersmith va permettre de retrouver une telle racine.

Théorème IX – 1 (Coppermith 96)**.** Soit N un entier à la factorisation inconnue, et f un polynôme de degré k unitaire de $\mathbf{Z}[x]$. Il existe un algorithme retrouvant les entiers x_0 avec $|x_0| < N^{\frac{1}{k}}$ et $f(x_0) \equiv 0 \pmod N$ en temps $\mathcal{O}(k^5 \log^9 N)$.

Pour l'application sur RSA, on a $k = e$ qui est petit et l'algorithme devient polynomial. La démonstration de ce théorème est constructive, on explicite l'algorithme. L'idée est de construire à partir de f un polynôme g avec des coefficients relativement petit (par rapport à N) afin que $g(x_0) = 0$ dans $\mathbf{Z}$. Le relativement petit est précisé par le lemme suivant.

Lemme IX – 2 (Howgrave-Graham 97)**.** Soit $g(x)$ un polynôme de degré k de $\mathbf{Z}[x]$, $m \in \mathbf{N}$ un paramètre. Supposons que $g(x_0) \equiv 0 \pmod{N^m}$ avec $|x_0| < X$. On note $g(xX)$ le vecteur $(g_0, g_1X, g_2X^2, \ldots, g_kX^k)$ si g est le polynôme $g(x) = \sum_{i=0}^k g_i x^i$. Supposons de plus que $\|g(xX)\| < \frac{N^m}{\sqrt{k+1}}$ alors $g(x_0) = 0$ dans $\mathbf{Z}$.

Démonstration. On a

$$\begin{aligned} |g(x_0)| = \Big|\sum_{i=0}^{k} g_i x_o^i\Big| \leqslant \sum_{i=0}^{k} |g_i x_o^i| \leqslant \sum_{i=0}^{k} |g_i| X^i = & \quad < (|g_0|, |g_1|X, \ldots, |g_k|X^k), (1, \ldots, 1) > \\ \leqslant & \quad \|\|g(xX)\| \times \|(1, \ldots, 1)\| \\ = & \quad \|g(xX)\|\sqrt{k+1} < N^m. \end{aligned}$$

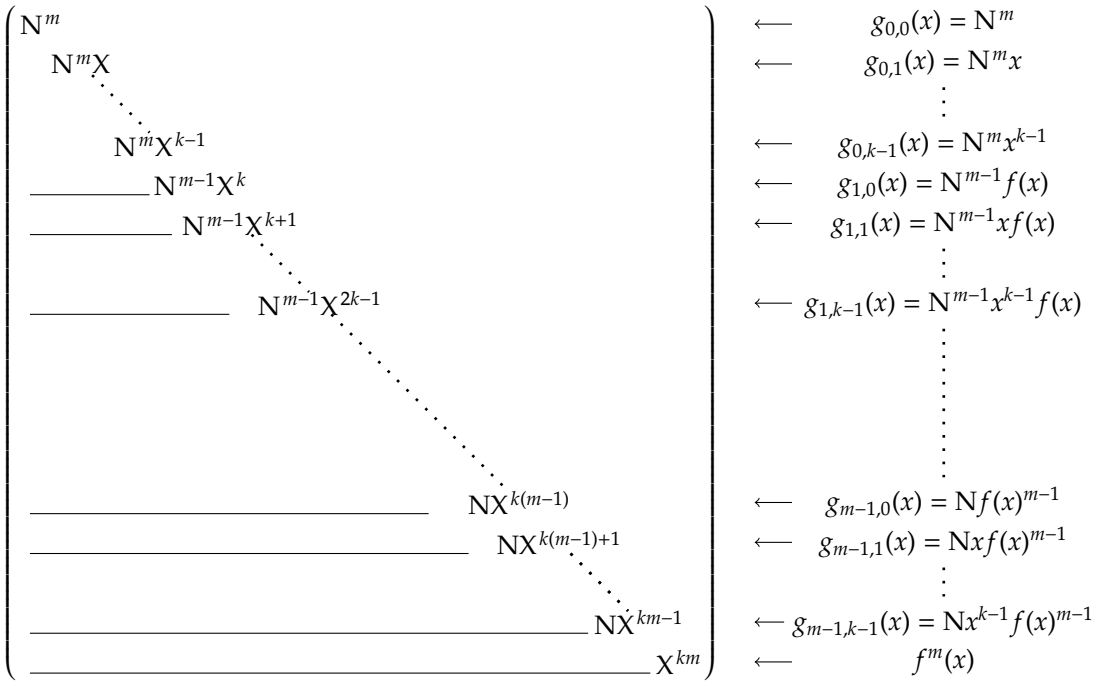


Fig. IX.1 : Matrice G

La majoration du produit scalaire par le produit des normes correspond à l'inégalité de Cauchy-Schwarz. Comme $|g(x_0)| \equiv 0 \ (mod\ N^m)$, on a $|g(x_0)| = AN^m$ dans $\mathbf{Z}$ pour un certain $A \in Z$ mais comme $|g(x_0)| < N^m$ on a forcément $A = 0$, c'est à dire $g(x_0) = 0$ dans $\mathbf{Z}$. □

Soit f un polynôme de degré k unitaire de $\mathbf{Z}[x]$, x_0 avec $|x_0| < N^{\frac{1}{k}}$ et $f(x_0) \equiv 0 \ (mod\ N)$. La réduction de réseau va intervenir pour construire à partir de f un polynôme avec de petits coefficients satisfaisant les hypothèses du lemme précédent. Soit m un paramètre. On considère une famille de polynômes de la forme

$$g_{i,j}(x) = N^{m-i} x^j f(x)^i \ \text{ pour } \ 0 \leqslant i \leqslant m-1 \ \text{ et } \ 0 \leqslant j \leqslant k-1.$$

On rajoute à cette famille le polynôme f^m, on obtient ainsi $d := mk + 1$ polynômes.

Il existe un entier $A \in \mathbf{Z}$ tel que $f(x_0) = AN$. On a donc pour tout i, j, $g_{i,j}(x_0) = N^{m-i} x_0^j A^i N^i \equiv 0 \ (mod\ N^m)$. De même, on obtient que $f^m(x_0) \equiv 0 \ (mod\ N^m)$.

On construit un réseau L de dimension d engendré par la matrice carrée G dont les lignes sont formées par les vecteurs $g_{i,j}(xX)$ pour $0 \leqslant i \leqslant m-1$ et $0 \leqslant j \leqslant k-1$ et le vecteur $f^m(xX)$ où X est une borne à déterminer plus tard. Comme ces polynômes sont de degrés croissants, on obtient la matrice triangulaire inférieure, G, donnée en figure IX.1.

Chaque vecteur de ce réseau peut être identifié à un polynôme ayant x_0 comme racine modulo N^m.

Comme L est de rang plein, on a $det\, L = |det\, G|$. Comme la matrice est triangulaire, son déterminant est le produit des éléments diagonaux. Sur la diagonale, les X^ℓ pour ℓ de 1 à km interviennent, et chaque N^ℓ pour ℓ de 1 à m apparaît k fois. On a donc

$$det\, L = N^{k\frac{m(m+1)}{2}} X^{\frac{km(km+1)}{2}}.$$

On pose $X = N^\alpha$, comme $d = km + 1$, on a

$$log_N(det\, L) = k\frac{m(m+1)}{2} + \alpha\frac{d(d-1)}{2}.$$

En appliquant LLL sur la matrice G, le premier vecteur de la base LLL-réduite satisfait

$$\|b_1\| \leqslant 2^{\frac{d-1}{4}} (det\, \mathrm{L})^{\frac{1}{d}}.$$

Si ce vecteur est de norme inférieure à $\frac{\mathrm{N}^m}{\sqrt{k+1}}$ et que $|x_0| \leqslant \mathrm{X}$ alors le lemme s'applique, ce vecteur b_1 donne un polynôme ayant x_0 comme racine dans $\mathbf{Z}$. On pourra donc le trouver.

On cherche donc la valeur maximale de α permettant d'avoir cette condition satisfaite, c'est à dire

$$2^{\frac{d-1}{4}} (det\, \mathrm{L})^{\frac{1}{d}} \leqslant \frac{\mathrm{N}^m}{\sqrt{k+1}}.$$

On néglige les quantités ne dépendant pas de N cette inégalité devient

$$det\, \mathrm{L} \leqslant \mathrm{N}^{md}.$$

En prenant le log_{N}, on obtient,

$$km(m+1) + \alpha d(d-1) \leqslant 2md.$$

Comme $d = km + 1$, en simplifiant par m , on a

$$k(m+1) + \alpha(km+1)k \leqslant 2(km+1).$$

La condition sera toujours vraie si

$$k(m+1) + \alpha(km+1)k \leqslant 2km.$$

en simplifiant par k, cela donne,

$$(m+1) + \alpha(km+1) \leqslant 2m.$$

Soit

$$\alpha \leqslant \frac{m-1}{km+1}.$$

La fraction de droite tendant vers $\frac{1}{k}$ quand m tend vers plus l'infini, on trouvera bien toute racine $|x_0| \leqslant \mathrm{N}^{\frac{1}{k}}$ pour m suffisament grand.

De nombreuses applications de généralisation de cette technique sont possibles. Par exemple (Coppersmith 97), étant donné $\mathrm{N} = pq$ et une approximation $\tilde{p}$ de p avec $|p - \tilde{p}| < \mathrm{N}^{1/4}$, on peut retrouver p en temps polynomial en cherchant une petite racine du polynôme $\tilde{p} + x \pmod{p}$, connaissant seulement N.

Une autre application due à Boneh et Durfee en 1999 utilise des petites racines d'un polynôme à deux variables. Étant donné une clef publique RSA, (N, e), si l'exposant de déchiffrement satisfait $d \leqslant 0,292$ alors il est possible de factoriser N en temps polynomial.