Etude de la cryptographie

MICHELUTTI Laura

Stage: L1 MPCI 2023-2024

3 juin - 30 juin

- Encadré par François BRUCKER -

Remerciements

Je tiens tout d'abord à remercier M.François BRUCKER, enseignant à la MPCI et à l'Ecole Centrale de Marseille (ECM) et chercheur au sein du laboratoire d'informatique et système (LIS), pour avoir accepter de me prendre en stage ainsi que pour son accompagnement de tous les jours, instructif et bienveillant.

Je souhaite aussi remercier les autres stagiaires de M.Brucker pour leurs idées et conseils ainsi que pour avoir rendu mon stage très agréable.

Je tiens aussi à remercier Pascal PREA, chercheur au LIS pour ses précieux conseils tout au long de mon stage. Mais également Thomas DURT pour sa présentation de la cryptographie quantique.

Enfin, je remercie M.Julien DESCHAMPS, responsable des stages en L1, pour son accompagnement.

Contents

Présentation: LIS

Le LIS, le Laboratoire d'Informatique et des Systèmes réunit 375 membres dont 190 membres permanents chercheurs et enseignants chercheurs.
Le laboratoire est une Unité Mixte de Recherche (UMR) sous tutelle du Centre National de la Recherche Scientifique (CNRS) qui est rattachée à l'Institut des sciences informatiques et de leurs intéractions (CNRS Science informatiques) de l'Université d'Aix Marseille (AMU) et de l'Université de Toulon (UTLN).
Les locaux du LIS sont répartis sur les campus de Saint-Jérôme et de Luminy à Marseille et sur le campus de l'Université de Toulon. Les recherches fondamentales et appliquées du LIS dans les domaines de l'informatique, de l'automatique, du signal et de l'image sont structurées en 4 pôles:

- Pôle Calcul
- Pôle Science des données
- Pôle Analyse et contrôle des systèmes
- Pôle Signal et Image

De plus, l'Ecole Centrale de Marseille (ECM) est partenaire du LIS. En effet, certains chercheurs enseignent à l'école les spécialités: informatique, génie électrique et automatique et ont donc leurs locaux au sein de l'ECM.

Introduction

La cryptologie est considérée comme "la science du secret", elle se sépare en deux branches, avec d'une part la cryptographie qui sert à chiffrer et donc protéger des données et d'autre part la cryptanalyse qui est la technique consistant à vouloir déduire d'un texte chiffré, les informations cachées.

Historiquement, la cryptologie a eu son importance notamment dans le domaine militaire afin de protéger les informations sensibles. Le code de César chez les Romains ou "Enigma" chez les Allemands durant la seconde guerre mondiale en sont les parfaits exemples. La sécurité de ces codes devient un enjeu majeur pour ceux les utilisant en temps de guerre comme en temps de paix. De nos jours, toutes les informations personnelles sont cryptées afin de garantir leur sécurité et leur confidentialité. Cette science est par conséquent omniprésente dans nos vies et indispensable dans notre société où les échanges d'informations sont toujours plus nombreux.

L'objectif de ce stage était donc de réaliser une initiation à la cryptologie et d'analyser le fonctionnement des registres à décalage (LFSR), c'est-à-dire de comprendre leur usage dans le cryptage des données.

Dans un premier temps, nous verrons le principe du One Time Pad puis, le fonctionnement des registres à décalage à rétroaction linéaire (LFSR) ainsi que la recherche du meilleur LFSR selon ces propriétés. Enfin, nous présenterons le décryptage et ce qu'on peut faire pour lutter contre.
Tout le code réalisé durant ce stage est accessible sur github grâce au lien dans la bibliographie (cf: [4]). Vous pouvez y trouver, toutes les fonctions, tests et mains écrits.

1 Présentation: One Time Pad

En cryptographie, le chiffrement le plus sûr est celui de Vernam appelé *One Time Pad*. Cette méthode consiste à appliquer un OU exclusif (XOR) équivalent de l'addition sur le corps $Z/2Z$ entre le message binaire et une suite de nombres binaires appelée clé. Le tableau suivant récapitule tous les résultats possibles pour l'addition ($\oplus$), la multiplication ($\otimes$) et la soustraction ($\ominus$).

A	B	A$\oplus$B	A$\otimes$ B	A $\ominus$ B
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	0

Ce qui donne, par exemple: $100100110 \oplus 011010010 = 111110100$. De plus, l'opération XOR possède 2 propriétés intéressantes tel que: $A \oplus A = 0$ et $A \oplus 0 = A$ qui nous seront utiles par la suite.

Seulement, ce chiffrement se doit de respecter les 4 conditions suivantes:

- la clé doit être au moins aussi longue que le message à chiffrer
- la clé doit absolument être secrète
- la clé ne doit jamais être utilisée plus d'une fois
- la clé doit être aléatoire

Ce chiffrement est considéré comme étant le plus sûr car il a été prouvé mathématiquement incassable (cf: preuve de Shannon). En effet, même en utilisant la force brute et en testant toutes les possibilités, on tombe sur pleins de résultats différents mais on est incapable de déterminer le bon messages.

Preuve de l'incassabilité/Shannon:
Soit X et Y deux variables aléatoires indépendantes sur D = {0, 1} avec X uniforme. Soit i un index fixé alors, on a:

- $P(X_i = 0) = P(X_i = 1) = 0.5$ car X est uniforme sur D.
- $P(Y_i = 0) = p_i$ et $P(Y_i = 1) = 1 - p_i$

Comme $(X \bigoplus Y)_i = 0$ seulement quand X_i et Y_i valent en même temps 0 ou 1, on en déduit que:

$$P[(X \bigoplus Y)_i = 0] = P(X_i = 0, Y_i = 0) + P(X_i = 1, Y_i = 1)$$

Or X et Y sont indépendants donc X_i et Y_i le sont aussi, par conséquent:

$$P(X_i = a, Y_i = b) = P(X_i = a) * P(Y_i = b)$$

On en déduit donc:

- $P(X_i = 0, Y_i = 0) = P(X_i = 0) * P(Y_i = 0) = 0.5p_i$
- $P(X_i = 1, Y_i = 1) = P(X_i = 1) * P(Y_i = 1) = 0.5 * (1 - p_i)$

Or, $P[(X \bigoplus Y)_i = 0] = P(X_i = 0, Y_i = 0) + P(X_i = 1, Y_i = 1)$ Ainsi, on en déduit:

$$P[(X \bigoplus Y)_i = 0] = 0.5p_i + 0.5(1 - p_i) = 0.5$$

Par conséquent, $(X \bigoplus Y)$ est une variable aléatoire uniforme.
Si on associe X comme étant la clé (c) et Y comme étant le message (m), d'après le résultat précédent, le message crypté aussi dit public (p) correspondant à $(X \bigoplus Y)$ ne donne aucune information sur Y ce qui prouve l'inviolabilité. En effet, un message crypté (p) peut-être issue de n'importe quel message original (m), il suffit pour cela de choisir un clé tel que $c = m \bigoplus p$, par conséquent avec ce chiffrement, on ne peut pas retrouver le message original.

1.1 Première condition: une clé assez longue

La première condition est nécessaire car si la clé est plus courte que le message confidentiel alors les derniers bits du message ne seront pas cryptés et donc accessibles à tous.

Exemple: Considérons une clé (c) et un message (m)
Si on a une clé de longueur 4 tel que $c = 1010$ et un message de longueur 9 tel que $m = 011010011$, alors le message public (p) correspond à:

$$p = c \oplus m = 101000000 \oplus 011010011 = 1100\mathbf{10011}$$

où les 5 derniers bits (en gras) ne sont pas protégés et donc accessibles pour tous car on a $A \oplus 0 = A$.

1.2 Deuxième condition: une clé secrète

Le *One Time Pad* est dit "symétrique" car la clé sert à la fois à chiffrer et à déchiffrer le message comme le montre l'égalité suivante:

$$M \oplus C = P \text{ et } P \oplus C = M \oplus C \oplus C = M$$

car $C \oplus C = 0$ et $M \oplus 0 = M$, où M est le message, C la clé et P le message crypté dit aussi message public.
Cette égalité nous montre qu'en appliquant un XOR entre le message crypté et la clé, on retombe sur le message non-codé car le message codé correspond au XOR du message non-codé et de la clé.
Partant du principe que le message crypté est généralement connu, si on connaît la clé, en appliquant un simple XOR, on retrouve le message confidentiel. Avoir une clé secrète est donc indispensable.

1.3 Troisième condition: Utilisation unique de la clé

Si 2 messages ont été chiffrés avec la même clé, en appliquant un XOR entre les messages cryptés on obtient un XOR entre les messages confidentiels:

$$P_1 \oplus P_2 = M_1 \oplus C \oplus C \oplus M_2 = M_1 \oplus M_2$$

où P_1 = message1 crypté/public, C = clé et M_2 = message2 non-codé. Si on connaît ensuite une partie d'un des 2 messages confidentiels alors on trouve le second, c'est-à-dire qu'on retrouve n'importe quel message chiffré par cette même clé (cf: partie décryptage).
On appelle parfois le *One Time Pad*: "masque jetable" ce qui fait référence à l'usage unique de la clé.

1.4 Quatrième condition: une clé aléatoire

Enfin, la clé doit être aléatoire afin que si on retrouve une partie de la clé, on ne puisse pas en déduire la suite. C'est pourquoi, on cherche à générer des nombres pseudo-alétoires construits à partir d'une clé plus courte aussi appelée: *seed* qui est plus facilement protégeable. On s'intéresse donc aux LFSR.

2 Registre à décalage: LFSR

Un registre à décalage à rétroaction linéaire ou LFSR (de l'anglais Linear Feedback Shift Register) est un circuit électronique ou un logiciel qui permet de générer une suite de bits qui peut être vue comme une suite récurrente linéaire sur le corps F_2, c'est-à-dire le corps des entiers modulo 2 : $Z/2Z$ composé de $0, 1$.

2.1 Définitions

Définition 1: Un LFSR de longueur l est composé de l celulles, $\forall l > 0$ et est défini par l coefficients binaires ou coefficients de connexion $(c_1, c_2, ..., c_l) \in (Z/2Z)^l$. A partir d'un état initial $S_0 = (s_0, s_1, ..., s_{l-1}) \in Z/2Z$, cet LFSR va générer $(s_i)_{i\geq 0}$, la suite récurrente linéaire d'ordre l définie sur $Z/2Z$ par la relation de rétroaction:

$$s_{i+l} = (c_1 s_{i+l-1} \oplus c_2 s_{i+l-2} \oplus ... \oplus c_l s_i) \ , \forall i \geq 0$$

La *seed* correspond ici à l'état initial S_0.

Définition 2: Soit un LFSR de longueur l et de coefficients $(c_1, c_2, ..., c_l)$. On définit son polynôme de rétroaction dit *feedback* (en anglais) comme étant le polynôme $P \in (Z/2Z)[X]$suivant:

$$P(X) = 1 + c_1 X^1 + c_2 X^2 + ... + c_l X^l = 1 + \sum_{i=1}^{l} c_i X^i$$

Définition 3: Soit un LFSR de longueur l qui génère la suite $(s_i)_{i\geq 0}$ à partir de l'état initial: $S_0 = (s_0, s_1, ..., s_{l-1})$. Pour tout entier $i \in N$, l'état interne du registre à l'itération i est défini comme la suite de l bits $S_i = (s_i, s_{i+1}, ..., s_{i+l-1})$

2.1.1 Exemple

On considère un LFSR de longueur: $l = 5$ et de coefficients de connexion: $(c_1, c_2, c_3, c_4, c_5) = (1, 0, 0, 1, 1)$ et dont l'état initial: $S_0 = (1, 1, 0, 1, 0)$. Ainsi, on définit son *feedback* polynôme comme étant: $P(X) = 1 + X + X^4 + X^5$ car $c_2 = c_3 = 0$.

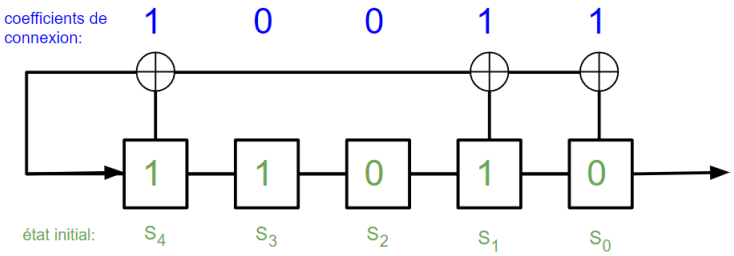


Figure 1: LFSR de longueur $l = 5$

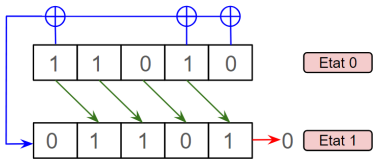
On représente généralement un polynôme dont les coefficients appartiennent au corps $Z/2Z$ comme une suite de 0 et de 1.
Ainsi, le polynôme $P(X) = 1 + X + X^4 + X^5$ peut être représenté comme "110011" où le "1" le plus à gauche correspond au "$X^0 = 1$" dans le polynôme, le "1" d'après au "$X^1 = X$", les "0" à l'absence de "X^2" et de "X^3" dans le polynôme et les deux "1" qui suivent au "X^4" et "X^5".

2.2 Fonctionnement

A chaque itération du registre, tous les bits sont décalés d'un cran sur la droite, le bit le plus à droite sort donc du registre et rejoint les termes de la suite. Le bit manquant appelé "feedback" bit provient d'une combinaison linaire de l'opération XOR. En effet, il vaut $s_{i+l} = (c_1 s_{i+l-1} \oplus c_2 s_{i+l-2} \oplus ... \oplus c_l s_i)$, $\forall i \geq 0$, où i est le nombre d'itérations.

2.2.1 Exemples

Exemple 1: On considère comme précédemment, le LFSR de longueur: $l = 5$ et de coefficients de connexion: $(c_1, c_2, c_3, c_4, c_5) = (1, 0, 0, 1, 1)$ et d'état initial: $S_0 = (1, 1, 0, 1, 0)$. Dans ces conditions, $c_2 = c_3 = 0$ et $l = 5$, donc le *feedback* bit prend pour valeur: $s_{i+5} = c_1 s_{i+4} \oplus c_4 s_{i+1} \oplus c_5 s_i$, $\forall i \geq 0$
De ces informations, on en déduit:



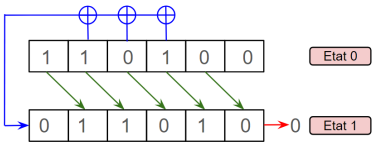
Fonctionnement du LFSR considéré

itérations	état interne du LFSR	bit sortant
0	11010	-
1	01101	0
2	10110	1
3	01011	0
4	00101	1
5	10010	1
6	01001	0
7	10100	1
8	11010	0
9	01101	0
10	10110	1

Tableau: donnant les différents états du LFSR considéré

Ce LFSR génère ainsi la suite pseudo-aléatoire suivante (cf: programme *python* exemple LFSR [4]): 0101101001011010010110100101101001 0...

Exemple 2: On considère un autre LFSR de longueur $l = 6$ cette fois et de coefficients de connexion: $(c_1, c_2, c_3, c_4, c_5, c_6) = (0, 1, 1, 1, 0, 0)$ et d'état initial: $S_0 = (1, 1, 0, 1, 0, 0)$. Dans ces conditions, le *feedback* bit prend pour valeur: $s_{i+6} = c_2 s_{i+4} \oplus c_3 s_{i+3} \oplus c_4 s_{i+2}, \forall i \geq 0$ De ces informations, on en déduit:



Fonctionnement du LFSR considéré

itérations	état interne du LFSR	bit sortant
0	110100	-
1	011010	0
2	001101	0
3	000110	1
4	100011	0
5	010001	1
6	101000	1
7	110100	0
8	011010	0
9	001101	0
10	000110	1

Tableau: donnant les différents états du LFSR considéré

Ce LFSR génère ainsi la suite pseudo-aléatoire suivante (cf: programme *python* exemple2 LFSR [4]): 0010110001011000101100010110001011 00...

Si on regarde de plus près, on remarque que les suites générées par les 2 LFSR sont différentes entre elles.

- Premier LFSR: 0101101001011010010110100101101001 0
- Deuxième LFSR: 0010110001011000101100010110001011 00

En effet, notons que sur la suite 2, il y a parfois "000" qui apparaît alors que cette combinaison n'apparaît jamais dans la suite 1. C'est pourquoi, une étude statistique est utile afin de trouver quelles sont les meilleures suites et par conséquent les LFSR les plus efficaces.

2.3 Vérification des propriétés statistiques des LFSR

On cherche à vérifier si les suites binaires générées par les LFSR respectent bien les mêmes propriétés statistiques qu'une suite réellement aléatoire c'est-à-dire qu'il y ait autant de "0" que de "1" ou encore autant de "00", "01", "10" que de "11" et ainsi de suite.

2.3.1 La loi du χ^2

Lors de notre étude statistique sur les suites binaires générées par les LFSR, on utilise le test du χ^2 d'adéquation (détaillé sur [2]). Celui-ci étant un test d'hypothèse, il permet de vérifier si une série de données suit ou pas une loi de probabilité: ici l'hypothèse nulle correspondant à de l'aléatoire.

- Tout d'abord, on calcule algèbriquement la distance entre les données observées et les données théoriques attendues. Pour cela, on utilise la formule suivante.

$$\chi^2_{calculé} = \sum_i \frac{(n_i - n_i^*)^2}{n_i^*}$$

où n_i correspond aux effectifs observés et n_i^* désigne les effectifs en adéquation avec la loi de probabilité testée, ici l'aléatoire.

- On détermine ensuite une distance critique grâce à la table du χ^2 (cf: [2]) pour un risque d'erreur choisi (par défaut on prend un risque de 5%) et un degré de liberté propre à chaque situation. Cette distance critique aussi appelée $\chi^2_{critique}$ correspond au χ^2 ayant une probabilité de dépassement égale au risque fixé lorsqu'on considère l'hypothèse nulle comme vraie.

- Enfin, on compare le $\chi^2_{calculé}$ au $\chi^2_{critique}$. Avec un risque d'erreur fixé à 5%, il y a seulement 5% de chance que le $\chi^2_{critique}$ soit supérieur à celui calculé.
  Si c'est le cas, on considère que les valeurs testées ne sont pas en adéquation avec la loi testée avec une marge d'erreur de 5%.
  D'autre part, si la valeur de $\chi^2_{calculé}$ est inférieure à celle de $\chi^2_{critique}$, les chances de se tromper sont inférieures à 0.05 si on considère les variables comme indépendantes (c'est-à-dire que les données testées sont aléatoires)

2.3.2 Résultat du χ^2 pour différents LFSR

On relève dans le tableau ci-dessous les valeurs du χ^2 pour différents degrés de liberté (k) avec un risque d'erreur toujours de 5% et des LFSR dont le *feedback* polynôme est de degré 3 avec un état initial de $S_0 = (1, 1, 0)$. Un degré de liberté de:

- k = 1, correspond à l'étude de l'apparition des "0" et "1"
- k = 2, correspond à l'étude de l'apparition des "00", "01", "10" et "11"
- k = 3, correspond à l'étude de l'apparition des "001", "010", "011", "100", "101", "110" et "111"

Dans la dernière colonne, on étudie la fréquence selon laquelle $s_i = s_{i+1}$

feedback polynôme	χ^2 pour k = 1	χ^2 pour k = 2	χ^2 pour k = 3	χ^2 pour $s_i = s_{i+1}$
1001	3.93	14.0	68.46	4.58
1010	39.09	118.19	271.39	37.23
1011	0.58	2.57	6.02	0.86
1100	39.09	118.19	271.39	37.23
1101	0.58	2.57	6.02	0.86
1110	3.93	14.0	68.46	4.58
1111	0.02	0.09	41.14	0.02
$\chi^2_{\ critique}$	7.88	10.60	12.84	7.88

On remarque que seuls 2 des 7 *feedback* polynômes (1011 et 1101) ont toutes leurs valeurs de χ^2 calculées inférieures à celle du $\chi^2_{critique}$ donc seuls ces 2 polynômes génèrent des suites de binaires respectant les lois statistiques des suites aléatoires.
Du résultat précédent, on en déduit que tous les *feedback* polynômes n'ont pas les mêmes propriétés et qu'il faut donc les étudier.

3 Recherche du meilleur LFSR

Les LFSR comportent un nombre fini l de cellules, avec seulement 2 possibilités pour les remplirent: 0 ou 1. Ainsi, il existe au maximun 2^l états pour l cellules. Or l'état composé uniquement de 0 n'est pas possible car il engendre une suite infini de 0. On en déduit donc qu'il y a $2^l - 1$ états internes possibles pour un LFSR. Ainsi, pour un LFSR dont le *feedback* polynôme est de degré l (c'est-à-dire que $c_l = 1$), la suite qu'il génère est par conséquent périodique de période maximun $2^l - 1$. On appelle ces polynômes des *non-singulars* polynômes.

3.1 Etude des périodes des différents LFSR

Grâce à l'algorithme du lièvre et de la tortue (détaillé sur [3]) on peut facilement trouver la période d'une suite de nombres. On relève les valeurs des périodes pour toutes les suites de nombres avec un *feedback* polynôme de degré 3 et avec tous les états initiaux possibles pour un LFSR de longueur 3 (cf: programme *python* Etude des périodes [4]).
On peut réaliser la même étude pour des polynômes de degré autre que 3.

feedback polynôme avec $S_0 =$	001	010	011	100	101	110	111
1001	3	3	3	3	3	3	1
1010	1	2	2	2	2	1	1
1011	7	7	7	7	7	7	7
1100	1	1	1	1	1	1	1
1101	7	7	7	7	7	7	7
1110	1	3	3	3	3	3	3
1111	4	2	4	4	2	4	1

De ce tableau, on remarque que 2 *feedback* polynômes (1011 et 1101) se démarquent des autres car ils ont tous les 2 une période de $2^3 - 1 = 7$ pour n'importe quel état initial. De plus, ce sont les 2 mêmes polynômes qui avaient les meilleurs résultats lors de l'étude statistique.
Ainsi, on en déduit que les *feedback* polynômes les plus intéressants sont ceux ayant les périodes les plus longues. Cela correspond à une période de $2^l - 1$ avec l la longueur du LFSR.
Pour les trouver, il faut chercher les polynômes respectant ces 2 conditions:

Condition 1: Les polynômes doivent être *irréductibles*.
En effet, on remarque que certains LFSR de grande longueur génèrent en réalité la même suite de nombres que certains LFSR de longueur moindre. C'est le cas, par exemple des polynômes 11011001001 et 1001 qui correspondent à $P(X) = 1 + X + X^3 + X^4 + X^7 + X^{10}$ et $P'(X) = 1 + X^3$ qui génèrent pour une seed donnée la même suite de valeurs (cf: programme *python* "Vérification même valeurs" [4]).
Or le *feedback* polynôme de degré 10 signifie qu'il faut une *seed* de longueur 10 pour une période valant au maximun $2^3 - 1 = 7$ qui est la période maximale pour le second polynôme. Dans ces conditions, c'est-à-dire pour un même résultat statisque il est préférable d'utiliser le LFSR de degré 3 car la seed à transmettre de manière sécurisée sera plus courte.
Ainsi, en cryptographie on cherche toujours des polynômes *irréductibles* sur le corps $Z/2Z$ qu'on appelle *minimal polynome* en anglais.

Condition 2: Les polynômes doivent être *primitifs* sur le corps considéré.
La plus petite période d'un LFSR est égale au degré du polynôme *minimal* P_0, c'est-à-dire au plus petit e tel que P_0 divise $1 + X^e$, grâce à un programme *python* (cf: [4]) on peut le retrouver. On en déduit d'après le travail d'ANNE CANTEAUT (cf: [1]) qu'une séquence a pour période maximale $2^{degP_0} - 1$ si et seulement si P_0 est un polynôme *primitif*.
Or comme c'est cette période qui nous intéresse, les polynômes utilisés se doivent d'être *primitifs*.
La séquence produite par un LFSR avec un *feedback* polynôme primitif est appelée *maximal-length sequence* (m-séquence) ce qui correspond à la suite binaire ayant la plus grande période.

Ces polynômes qui remplissent ces deux conditions et qui produisent donc des m-sequences sont répertoriés dans des tables telles que celles présentes dans *Introduction to finite fields and their applications* de RUDOLF LIDL et HARALD NIEDERREITER (cf: [1])

4 Décryptage

4.1 Utiliser les informations connues

Parfois savoir certaines informations sur la manière dont les messages sont chiffrés peut permettre de décrypter ces derniers facilement, par exemple quand deux messages sont chiffrés par la même clé. Effectivement, comme expliqué précédement (cf: One Time Pad), lorsqu'on opère un XOR entre deux messages cryptés par une seule et même clé, celle-ci s'annule laissant entre les mains du pirate un XOR entre les deux messages privés: $M_1 \oplus M_2$.
Ensuite, selon les informations connues, on peut en déduire plus ou moins d'informations:

- Si on connaît un des deux messages alors il suffit d'appliquer un XOR entre celui-ci et $M_1 \oplus M_2$ pour trouver le second message privé.
- Si on connaît uniquement le thème ou une partie d'un des deux messages, alors on fait un XOR entre cette information et $M_1 \oplus M_2$ afin d'en déduire une information sur le message privé inconnu (cf: programme de décryptage par déduction [4]). Seulement comme la position de cette information n'est pas connue, il est nécessaire de tester ce XOR à différents endroits du $M_1 \oplus M_2$ puis de répéter cette même opération avec les élèments obtenus afin de réussir à trouver le plus d'informations par tatônnements.

Exemple: Prenons 2 messages encodés avec la même clé tels que:
- Clé: $C = 11010110010001111010110010001111010110010001111010110 0...$
- Message 1 privé : $M_1 =$"Elle va sur la plage" $= 01000101011011000110110001 1...$
Ce qui donne comme message encodé $P_1 = 10010011001010111100000011101 0...$
- Message 2 privé : $M_2 =$"Il mange une glace" $= 01001001011011000010000 00...$
Ce qui donne comme message encodé $P_2 = 10011111001010111000110011100 0...$

On réalise un XOR entre P_1 et P_2 ce qui donne $M_1 \oplus M_2$ car les deux messages sont chiffrés avec la même clé.

Cas 1: On connaît le message 1 (M_1)
Alors, on applique un XOR entre $M_1 \oplus M_2$ et M_1 ce qui donne M_2. Ainsi, sachant que les 2 phrases sont encodées avec la même clé et connaissant le message: "Elle va sur la plage" je peux retrouver "Il mange une glace" soit le deuxième message (cf: programme *python* Décryptage connaissance d'un message [4]).

Cas 2: On connaît juste le thème de la conversation: *vacances d'été* Alors, on réfléchit aux mots qui pourraient être présents dans un des deux messages tel que: plage, soleil, mer, glace, repos, fête, 14 juillet,...
Choisissons par exemple le mot: "plage", après l'avoir mis sous forme binaire, on applique un XOR entre celui-ci et le début du message, puis le milieu et ainsi de suite jusqu'à avoir testé toutes les positions. Dans notre cas, le mot "plage" étant à la fin, le résultat de cette opération en plaçant "plage"

à la fin donne "glace". Par déduction, on peut penser qu'il y ait le mot "mange" devant "glace". Ainsi, en réalisant la même opération avec "mange", on trouve différents résultats mais seulement 1 qui pourrait venir de la langue française: "XXXe va XXXXXXXXXXXX" au vu du placement des espaces. Par conséquent, on peut appliquer un XOR avec cette fois les déterminants et espaces pour essayer de trouver plus d'indices en éliminant à chaque fois les réponses qui semblent improbables.
$M_1 \oplus M_2 \oplus$" la plage" = "XXXXXXXX une glace"
Puis on peut faire des essais avec des bouts de phrases tels que:
$M_1 \oplus M_2 \oplus$"mange une glace" = "XXXe va sur la plaXX"
$M_1 \oplus M_2 \oplus$"elle va sur la plage" = "il mange une glace "
Ainsi, en faisant par déduction et en ne connaissant que le thème de la conversation, on est capable de retrouver les 2 phrases cachées.

4.1.1 Modifier un message

Dans certains cas, on peut modifier le message crypté sans le décoder entièrement si l'on connaît sa structure et un autre message ayant la même structure et codé avec la même clé.

Exemple: Prenons deux messages chiffrés par la même clé et dont la structure est de la forme: "Virement de XXX euros pour XXXXXXX" et dont un des deux messages est connu: "Virement de 500 euros pour Andréa".
Alors, connaissant la structure et sachant coder 500 en binaire, en appliquant un XOR entre le 500 codé en binaire et 500 encodé avec la clé, on obtient la partie de la clé utilisée pour chiffrer la valeur 500 initialement. C'est-à-dire qu'on fait, $500_{binaire} \oplus 500_{codé} = clé_{partielle}$.
Par la suite, ayant la clé, on peut coder n'importe quel nombre ayant le même nombre de caractères que "500" avec la clé.
De plus, connaissant la structure du message à modifier, on peut remplacer le nombre inconnu initialement codé par celui choisi et codé avec la même clé et rendant donc un message différent lors du déchiffrage.

Dans ce cas, il est indispensable de vérifier que le message reçu correspond bien au message crypté envoyé pour être sûr de recevoir le message correct et pas un message détourné modifié.
Il est également nécessaire de ne pas utiliser une clé plusieurs fois et de choisir un LFSR ayant une période la plus longue possible afin de ne pas réutiliser une partie de suite binaire.

4.2 L'algorithme de Berlekamp Massey

Les informaticiens Berlekamp et Massey ont établi un algorithme capable de donner le *feedback* polynôme du LFSR à condition que le LFSR génère une m-sequence et de connaître une partie de longueur $2l$ de la suite binaire qu'il

génère où l désigne la longueur du LFSR (cf: implémentation sur python [4]).

Algorithme de Berlekamp-Massey: pour le corps $Z/2Z$

Entrée: $s^n = s_0 s_1 ... s_n - 1$, une séquence de n élèments de $Z/2Z$

Sortie: P, le *feedback* polynôme d'un LFSR de longueur l et génèrant s^n

Initialisation

$P(X) \leftarrow 1$, $P'(X) \leftarrow 1$, $\Delta \leftarrow 0$, $m \leftarrow -1$

Algorithme

for t from 0 to $n-1$ **do**:
$d \leftarrow s_t + \sum_{i=1}^{\Delta} p_i s_{t-i}$
if $d \neq 0$ **then**
$T(X) \leftarrow P(X)$
$P(X) \leftarrow P(X) - P'(X)X^{t-m}$
if $2\Delta \leq t$ **then**
$\Delta \leftarrow t + 1 - \Delta$
$m \leftarrow t$
$P'(X) \leftarrow T(X)$
return P

La connaissance du *feedback* polynôme permet ensuite de décrypter un message. En effet, grâce à celui-ci, on peut obtenir la suite que le LFSR correspondant produit, c'est-à-dire la clé permettant le chiffrage et par conséquent celle permettant de décrypter puisque c'est un codage symétrique.
Même si la position du début de la suite utilisée n'est pas connue, en essayant à l différents endroits correspondant à la longueur de la *seed* utilisée, on retrouve le message privé.

4.2.1 Les moyens de lutte

L'algorithme de Berlekamp Massey fonctionne car le *feedback* bit est une combinaison linéaire de l'opération XOR. Afin d'empêcher son bon fonctionnement, on peut réaliser une composition de LFSR supprimant le caractère linéaire et rendant son action compliquée voire impossible.

Pour ce faire, on utilise, le contrôle d'horloge: le registre B contrôle l'horloge de A, c'est-à-dire que le registre A génère un nouveau nombre uniquement si B sort un 1 sinon, le registre renvoie la même valeur que précedemment.

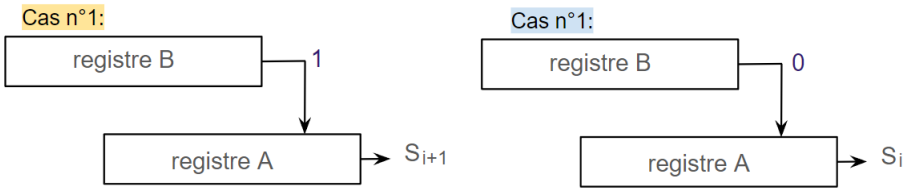


Figure 2: Composition de 2 registres par contrôle d'horloge

Lorsqu'on applique l'algorithme de Berlekamp Massey à ce type de composition, il nous renvoie généralement un *feedback* polynôme de degré équivalent à la quantité de nombres donnés et dont la valeur change selon la quantité de nombres renseignés.

Cependant, la suite générée est problèmatique. En effet, elle ne respecte pas les propriétés statistiques d'une suite aléatoire (cf: partie 2.3). Un bit est égal 3 fois sur 4 au bit précédent au lieu de 1 fois sur 2 pour le cas des suites aléatoires. C'est pourquoi, pour revenir dans des conditions statistiquement correctes, on ajoute un nouvel LFSR à la composition tel qu'on applique un XOR entre la valeur donnée par l'ensemble précédent et le nouveau registre à décalage (Figure 3).

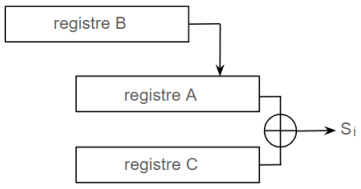


Figure 3: Composition de 3 registres par contrôle d'horloge avec XOR

L'ajout de ce nouveau LFSR permet de retrouver des résultats statistiques cohérents pour du pseudo-aléatoire (voir programme sur [4]).

5 Conclusion

A l'issue de ces quatres semaines, j'ai découvert les principes de la cryptologie et ses bases à travers le One Time Pad. J'ai compris comment utiliser un LFSR de manière efficace grâce à l'étude des *feedback* polynômes et l'analyse statistique des suites "pseudos-aléatoires" pour qu'elles correspondent aux critères du One Time Pad.
De plus, durant ce stage, j'ai commencé à réfléchir au décryptage grâce à l'algorithme de Berlekamp Massey ou encore en étudiant toutes les informations qu'on pouvait tirer de petites fuites d'informations ou de manque de précautions avec l'utilisation d'une même clé plusieurs fois. Ainsi, j'ai pu remarquer que n'importe quelle petite donnée laissée à découvert peut être utilisée tel que le thème d'une conversation et qu'il est donc absolument nécessaire d'avoir une très bonne protection. Par la suite, ça serait intéressant de voir comment on peut se transmettre la *seed* informatiquement mais de manière sécurisée en étudiant des cryptages asymétriques avec des clés dites publiques par exemple. On pourrait également s'intéresser aux moyens permettant de vérifier si on reçoit bien le message voulu et non pas modifié par un intrus.

Ce stage m'a permis de découvrir un domaine: "la cryptologie" qui m'était jusqu'alors inconnu et qui m'a donc demandé un travail de documentation. J'ai pu par ailleurs améliorer mes compétences de programmation et les mettre en pratique dans un contexte nouveau. Ce moment a également été l'occasion de découvrir le milieu de la recherche et de mieux en comprendre le fonctionnement.

References

[1] Recherche sur les registres à décalage:
- RUDOLF LIDL et HARALD NIEDERREITER, *Introduction to finite fields and their applications*, Cambridge University Press

- ANNE CANTEAUT, *Lecture Notes on Error-Correcting Codes and their Applications to Symmetric Cryptography* , partie: LFSR-based Stream Ciphers, Inria https://www.paris.inria.fr/secret/Anne.Canteaut/MPRI/chapters-10-13.pdf

[2] Recherche Loi du Khi 2:
- Loi du Khi 2 : Test du X^2. Wikipédia https://fr.wikipedia.org/wiki/Test_du_

- Table du Khi 2 https://archimede.mat.ulaval.ca/stt1920/STT-1920-Loi-du-khi-deux.pdf

[3] Recherche sur l'algorithme du lièvre et de la tortue
- Algorithme permettant de trouver la période: https://fr.wikipedia.org/wiki/Algorithme_du_li%C3%A8vre_et_de_la_tortue

[4] Lien du github où se trouvent tous les programmes écrits https://github.com/LauraMichelutti/Cryptographie.git
Tous les "main.py" créés:

- Exemple LFSR
  https://github.com/LauraMichelutti/Cryptographie/blob/67546a81ba7eb72dda94af695ee378d351161fd5/main/main_ex_lfsr.py
- Exemple 2 LFSR
  https://github.com/LauraMichelutti/Cryptographie/blob/67546a81ba7eb72dda94af695ee378d351161fd5/main/main_ex2_lfsr.py
- Vérification même valeurs:
  https://github.com/LauraMichelutti/Cryptographie/blob/eda5b77faa04def0979774aa0a2d6cdcdd318242/main/main_verification_meme_valeurs.py
- Etude des périodes
  https://github.com/LauraMichelutti/Cryptographie/blob/eda5b77faa04def0979774aa0a2d6cdcdd318242/main/main_etudes_des_periodes.py
- Modifier un message
  https://github.com/LauraMichelutti/Cryptographie/blob/

eda5b77faa04def0979774aa0a2d6cdcdd318242/main/main_modifier_un_message.py

- Résultat χ^2
  https://github.com/LauraMichelutti/Cryptographie/blob/eda5b77faa04def0979774aa0a2d6cdcdd318242/main/main_resultat_chi2.py
- Décryptage avec thème par déduction
  https://github.com/LauraMichelutti/Cryptographie/blob/347da0c001869032504eaece08db2c9f776d33b4/main/main_decryptage_avec_th%C3%A8me
- Décryptage connaissance d'un message
  https://github.com/LauraMichelutti/Cryptographie/blob/425c0365968e7c91fda84e118d8d94f130960397/main/main_decryptage_connaissance_un_message.py