# Chiffrement à flot

christina.boura@uvsq.fr

## 1 Cryptographie symétrique

La cryptographie *symétrique* ou cryptographie *à clé secrète* recense tous les algorithmes cryptographiques pour lesquels l'émetteur et le destinataire (Alice et Bob) partagent le même secret, ou autrement dit la même clé.

Dans un scenario classique, Alice et Bob souhaitent communiquer mais ne possèdent pour cela qu'un canal de communication non sécurisé qui peut être par exemple une ligne téléphonique ou un réseau internet. Tous ces canaux sont susceptibles d'être mis sur écoute par une personne malveillante que nous appellerons Oscar.

Si la communication est de nature confidentielle, Alice et Bob aimeraient qu'Oscar n'ait pas accès à son contenu et pour cela ils auront recours au chiffrement. Le chiffrement symétrique est alors une solution qui s'offre à eux. Pour cela, Alice et Bob doivent échanger préalablement une clé secrète, $k$, à travers un canal sûr. Alice chiffre ensuite le message $m$ avec un algorithme de chiffrement symétrique en utilisant la clé $k$ et produit un message chiffré $c = e_k(m)$ où par $e_k$ on note la fonction de chiffrement parametrée par $k$. De l'autre côte du canal, Bob reçoit $c$ et le déchiffre à l'aide d'un algorithme de déchiffrement $d_k$ en utilisant la même clé $k$ pour lire le message clair $m = d_k(c)$.

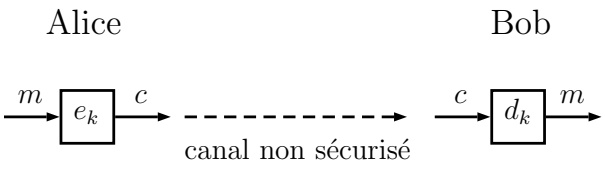



Figure 1 – Chiffrement symétrique

Les algorithmes de chiffrement symétrique se divisent en deux grandes familles : les chiffrements *à flot* qui chiffrent les bits un à un et les chiffrements *par bloc* qui découpent le message en blocs de taille fixe et qui le chiffrent bloc par bloc.

## 2 Chiffrement à flot

Le chiffrement à flot consiste à chiffrer un à un les bits du message clair au fur et à mesure. Ceci est fait en combinant chaque bit $m_i$ du message clair avec un bit $s_i$ d'une suite binaire, appelée la *suite chiffrante*, afin de produire un bit $c_i$ du message chiffré.

Les algorithmes de chiffrement à flot peuvent être implémentés de façon très compacte et sont généralement très rapides. Par conséquent, ils sont un choix excellent pour chiffrer des communications ou des données sur des dispositifs de petite taille et ne possédant pas beaucoup de ressources, comme par exemple les téléphones portables ou d'autres petits objets connectés. La norme GSM pour les communications téléphoniques intègre notamment dans ses specifications des chiffrements à flot pour protéger les communications vocales.

### 2.1 Chiffrement et déchiffrement

Les fonctions de chiffrement et de déchiffrement du chiffrement à flot sont extrêmement simples. En effet, chaque bit $m_i$ du message clair est transformé à un bit $c_i$ en faisant une simple addition modulo 2 avec un bit $s_i$ d'une suite chiffrante $s$.

**Chiffrement** $c_i = e_{s_i}(m_i) \equiv m_i + s_i \mod 2$.

**Déchiffrement** $m_i = d_{s_i}(c_i) \equiv c_i + s_i \mod 2$.

Tout d'abord nous allons montrer que l'opération de déchiffrement produit le résultat attendu, c'est-à-dire qu'en déchiffrant le bit $c_i$ on retrouve bien le bon $m_i$. Le contraire aurait été embêtant pour une fonction de chiffrement !

$$\begin{aligned} d_{s_i}(c_i) &\equiv c_i + s_i \mod 2 \\ &\equiv m_i + s_i + s_i \mod 2 \\ &\equiv m_i + 2s_i \mod 2 \\ &\equiv m_i. \end{aligned}$$

Le résultat vient du fait que $2 \equiv 0 \mod 2$ donc $2s_i \equiv 0 \mod 2$ .

Nous pouvons à ce stade remarquer que la fonction de chiffrement est identique à la fonction de déchiffrement et qu'en plus celle-ci consiste en une simple addition modulo 2. Nous allons maintenant rapidement discuter pourquoi l'addition modulo 2 donne une bonne fonction de chiffrement. Faisons pour cela sa table de vérité.

| $m_i$ | $s_i$ | $c_i = m_i + s_i \mod 2$ |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 1 | 1 | **0** |

Nous pouvons d'abord remarquer que cette table de vérité correspond à la table de vérité d'un opérateur logique bien-connu qui est le OU exclusif. Le OU exclusif est aussi appelé *XOR* et est noté comme $\oplus$. C'est cette notation que nous adopterons par la suite.

Observons maintenant cette table de vérité un peu mieux et supposons qu'on veut chiffrer le bit 0. Selon la valeur du bit de la suite chiffrante, le chiffré sera soit 0 (si $s_i = 0$) soit 1 (si $s_i = 1$). Si $s_i$ est une valeur complètement aléatoire, c.-à-d. que $s_i = 0$ avec probabilité 0.5 et $s_i = 1$ avec probabilité 0.5, alors chaque valeur du chiffré a la même probabilité d'apparaitre. La même observation tient si on veut chiffrer le bit 1. On peut alors voir que l'opération XOR est parfaitement équilibrée, c'est-à-dire que si on observe le bit de la sortie, il y a 50% de chance pour que chacune des valeurs du bit clair se cache derrière. C'est à cela que se distingue l'opérateur XOR des autres opérateurs logiques comme le "OU INCLUSIF" ou le "ET".

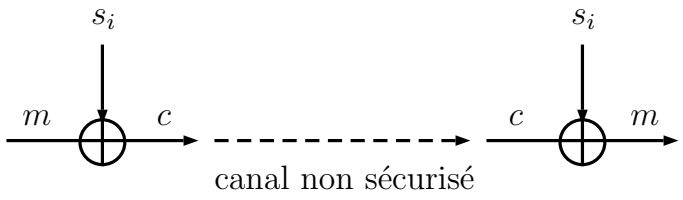



FIGURE 2 – Chiffrement de déchiffrement dans le cadre du chiffrement à flot

# 3 Générateurs de nombres aléatoires

Toute la sécurité d'un chiffrement à flot depend de la qualité de la suite chiffrante $(s_i)_i$. Cette suite doit évidemment être le plus *aléatoire* possible. Pour la générer, on utilise des méthodes appelées générateurs de nombres aléatoires dont on peut distinguer trois familles :

**Générateurs de nombres vraiment aléatoires** Les générateurs de nombres vraiment aléatoires (true random number generators (TRNG) en anglais) sont des méthodes de génération de l'aléa ayant la particularité que la suite générée ne peut pas être reproduite. On connait de telles méthodes depuis toujours : lancement de dés, roulette, loto, le pile ou face etc .. D'autres générateurs de cette catégorie plus sophistiqués produisent de l'aléa à partir de certains phénomènes physiques comme les bruits thermiques ou électromagnétiques ou encore des phénomènes quantiques. De plus, pour le cas spécifique du chiffrement à flot, des générateurs de cette catégorie ne sont pas pratiques pour la raison suivante : Le résultat de ces générateurs ne peut généralement pas être reproduit donc il serait difficile pour Bob de régénérer la même suite chiffrante qu'Alice pour déchiffrer le message.

Ce type de générateurs sont nécessaires pour certains processus cryptographiques, notamment pour la génération de clés. Pour cela, ce sont les ordinateurs qui sont chargés de produire du "vrai" aléa. Ils font cela en exploitant typiquement les déplacements ou les clics de la souris, le temps écoulé entre les frappes clavier, l'activité réseau et d'autres phénomènes de ce type.

La plupart de ces générateurs, même s'ils produisent de l'aléa de bonne qualité sont des méthodes qui restent assez lentes et nécessitant beaucoup d'énergie.

**Générateurs de nombres pseudo-aléatoires** Les générateurs de nombres pseudo-aléatoires (pseudorandom number generators (PRNG) en anglais) génèrent des suites qui sont calculées à partir d'une graine initiale. La construction la plus courante produit cette suite de façon récursive à l'aide d'une fonction $f$ :

$$\begin{aligned} s_0 &= \text{graine} \\ s_{i+1} &= f(s_i), \quad i = 0, 1, \ldots \end{aligned}$$

Une forme de générateur très populaire de ce type sont les générateurs congruentiels linéaires pour lequels la fonction $f$ est une fonction affine modulo un entier $m$.

$$\begin{aligned} s_0 &= \text{graine} \\ s_{i+1} &= as_i + b_i \mod m \quad i = 0, 1, \ldots \end{aligned}$$

avec $a$ et $b$ certaines constantes. Un exemple bien connu de ce type de générateur est la fonction `rand()` du langage `C`, utilisée pour produire de l'aléa. Ses paramètres sont :

$$\begin{aligned} s_0 &= 12345 \\ s_{i+1} &= 1103515245 s_i + 12345 \mod 31 \quad i = 0, 1, \ldots, \end{aligned}$$

Les générateurs de nombres pseudo-aléatoires ne sont pas "réellement" aléatoires dans le sens où la suite produite est calculé de façon déterministe et peut être facilement reproduite. Cependant, la suite produite a des bonnes qualités dans le sens que leur sortie approxime assez bien une suite des vrais nombres aléatoires. Ceci peut être vérifié à l'aide des tests statistiques. Plusieurs tests statistiques existent dans ce but et un générateur de nombres pseudo-aléatoire est dit de bonne qualité si sa sortie passe avec succès tous ces différents tests.

**Générateurs de nombres aléatoires cryptographiquement sûrs** Les générateurs de nombres aléatoires cryptographiquement sûrs sont des générateurs de nombres pseudo-aléatoires mais ayant une contrainte supplémentaire : Leur sortie doit être *imprévisible*. En pratique, ceci veut dire qu'étant donnés $n$ bits consécutifs générés $s_i, s_{i+1}, \ldots, s_{i+n-1}$ il est calculatoirement impossible de prédire la valeur de $s_i$.

Ce dernier type de générateur est justement celui qui est utilisé dans le cadre du chiffrement à flot. En effet, ce serait catastrophique si une attaque pouvaît prédire la suite d'une suite chiffrante après avoir observé un bout de celle-ci.

# 4 Le chiffre de Vernam ou One-Time-Pad (OTP)

Le *chiffre de Vernam*, autrement appelé *masque jetable* ou encore *One-Time-Pad (OTP)* est un système de chiffrement symétrique inventé par Gilbert Vernam en 1917. En pratique, il s'agit d'un chiffrement à flot pour lequel la suite chiffrante $(s_i)_i$ vérifie les conditions suivantes :

1. Elle doit être générée avec un vrai générateur aléatoire.
2. Chaque clé ne doit être utilisée qu'une seule fois. En pratique donc, une clé différente doit être générée pour chiffrer chaque nouveau message. Dans le contexte du chiffrement à flot, une clé peut être vue comme un masque, d'où le nom "masque jetable".
3. La taille de la clé (en bits) doit être identique à la taille du message à chiffrer.

L'intérêt et la popularité du chiffre de Vernam viennent du fait que Claude Shannon a prouvé en 1949 qui si les trois conditions ci-dessus sont respectées le chiffrement offre une "sécurité parfaite".

**Définition 4.1** (Shannon 1949)**.** Un chiffrement est dit *inconditionnellement sûr* ou *parfait* s'il ne peut pas être cassé même si l'attaquant possède des ressources de calcul infinis.

Claude Shannon a donc démontré à l'aide de la théorie des probabilités que le chiffre de Vernam est inconditionnellement sûr. L'argument théorique derrière cette preuve est le suivant : Dans le cadre d'une attaque à chiffré seul (on suppose que la seule information détenue par un attaquant est la connaissance du chiffré), puisque toutes les clés sont équiprobables et que l'opération de chiffrement est le XOR, alors tous les textes clairs ont la même probabilité d'être cachés derrière le chiffré. Par exemple, supposons que le message ne fait que 2 bits et que le texte chiffré est `c = 10`. Si toutes les suites chiffrantes de deux bits `00`, `10`, `01`, `11` sont équiprobables, alors tous les textes clairs à 2 bits sont équiprobables également puisque en XORant `c` avec toutes les clés possibles on voit bien qu'on obtient tous les messages possibles :

$$\texttt{10} \oplus \texttt{00} = \texttt{10}, \quad \texttt{10} \oplus \texttt{10} = \texttt{00}, \quad \texttt{10} \oplus \texttt{01} = \texttt{11}, \quad \texttt{10} \oplus \texttt{11} = \texttt{01}.$$

Cependant, malgré l'atout théorique indiscutable de ce chiffrement, sa mise en pratique présente d'important inconvénients :

— Tout d'abord, la taille de la clé doit être la même que la taille du message à chiffrer. Donc, si on doit chiffrer un document ou une conversation de 1Go, la clé doit faire la même taille, ce qui correspond à une très grande quantité d'information à sauvegarder et transmettre.
— Générer une clé vraiment aléatoire est un processus complexe et coûteux.
— Chaque clé doit être unique. Générer donc une clé réellement aléatoire pour chaque nouveau message est assez compliqué. En plus, un canal sécurisé doit être trouvé pour partager la clé entre les deux destinataires. Si un tel canal existe, pourquoi alors ne pas l'exploiter pour y envoyer le message directement ?

Malgré les difficultés liées à sa mise en pratique, le chiffre de Vernam a été utilisé à quelques reprises comme méthode de chiffrement. Par exemple, la hotline entre Moscou et Washington, établie en 1963 après la crise des missile cubains, était protégée par ce chiffrement. Les clés, sous forme de bande de papier, tout comme le message, ont été transportées par la valise diplomatique d'une ambassade à l'autre.

# 5 Le chiffrement à flot en pratique

Nous venons de voir que le chiffrement parfait existe mais il n'est pas pratique à mettre en œuvre. Pour cette raison, on utilise en pratique des générateurs de nombres pseudo-aléatoires (PRNG en anglais) pour générer la suite $s_i$. Cette suite est initialisée avec une valeur secrète, la clé $k$, connue par Alice et Bob. Il doit être impossible pour un attaquant de pouvoir reproduire la même suite chiffrante sans la connaissance de cette clé.

Tous les chiffrements à flot modernes utilisent la configuration de la Figure **??**. Cependant il faut faire très attention au choix du générateur. Un mauvais choix peut compromettre toute la sécurité du chiffrement à flot. Nous montrons maintenant que si un générateur congruentiel linéaire est utilisé par exemple comme PRNG, malgré ses très bonnes propriétés statistiques, la sécurité du chiffrement est réduite à zéro.

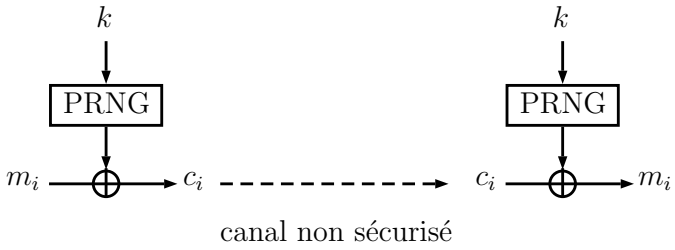



FIGURE 3 – Le chiffrement à flot en pratique

# 6 Utiliser un générateur congruentiel linéaire pour le chiffrement à flot

Rappelons qu'un générateur congruentiel linéaire fonctionne de la manière suivante :

$$\begin{aligned} s_0 &= \text{graine} \\ s_{i+1} &= as_i + b_i \mod m \quad i = 0, 1, \ldots, \end{aligned}$$

On suppose que la valeur $m$ est publiquement connue et la clé secrète $k$ est formée du couple $(a, b)$. Nous montrons maintenant comment casser un chiffrement à flot basé sur ce générateur, même si les valeurs de $a$ et $b$ sont choisies de façon à donner des bonnes propriétés statistiques à celui-ci.

L'attaquant connaît évidemment les bits $c_i$ du chiffré mais on peut également supposer qu'il a connaissance des bits $m_0, m_1$ et $m_2$. Cette supposition peut être totalement légitime puisque en pratique pour les communications internet et pour de nombreux protocoles le message doit commencer par une certaine en-tête fixe, comme par exemple un numéro de protocole. L'attaquant peut alors calculer

$$s_i = c_i + m_i \mod m, \quad i = 0, 1, 2.$$

Une fois les $s_i$ récupérés, l'attaquant peut créer le système suivant :

$$\begin{aligned} s_1 &\equiv as_0 + b \mod m \\ s_2 &\equiv as_1 + b \mod m \end{aligned}$$

Il s'agit d'un système linéaire de 2 équations avec 2 inconnues. L'attaquant peut résoudre ce système et retrouver les deux inconnues $a$ et $b$ :

$$\begin{aligned} a &= (s_1 - s_2)(s_0 - s_1)^{-1} \mod m \\ b &= s_1 - s_0(s_1 - s_2)(s_0 - s_1)^{-1} \mod m \end{aligned}$$

Dans le cas où $\text{pgcd}((s_0 - s_1), m) \neq 1$ ce système a plusieurs solutions (puisque nous sommes dans $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$), mais on peut toujours trouver l'unique solution avec la connaissance d'un clair complémentaire. Une fois les valeurs de $a$ et de $b$ retrouvées l'attaquant est en mesure de calculer toute la suite chiffrante $s$.

# 7 Registres à décalage à retroaction linéaire (LFSR)

Le *registre à décalage à rétroaction linéaire* constitue l'élément de base des générateurs pseudo-aléatoires utilisés pour la génération de la suite chiffrante dans le chiffrement à flot. On utilise souvent son acronyme anglais : LFSR (Linear Feedback Shift Register).

Un LFSR est composé d'un registre à décalage contenant une suite binaire de longueur $m$ ($s_i, s_{i+1}, \ldots, s_{i+m-1}$), et d'une fonction de retroaction linéaire. La taille du registre $m$ est appelé le *degré* du LFSR. A chaque coup d'horloge, le bit de poids faible $s_i$ constitue la sortie du registre tandis que les autres sont

décalés vers la droite. Le nouveau bit $s_{i+m}$ est calculé à l'aide de la fonction de retroaction linéaire et il est placé dans la cellule de poids fort du registre. La fonction de retroaction linéaire est typiquement le XOR du contenu de certains cellules du registre. Comme le XOR est une opération linéaire, c'est la raison pour laquelle ce générateur est appelé registre à rétroaction linéaire.

**Exemple**. On considère un LFSR de degré 3 avec une fonction de retroaction comme sur la figure **??**. Les bits du registre sont notés $s_i$ et sont décalés vers la droite à chaque coup d'horloge. Le bit le plus à droite constitue la sortie du LFSR. Le bit le plus à gauche est quant à lui calculé à l'aide de la fonction de retroaction. Plus précisément,

$$s_{i+3} = s_i \oplus s_{i+1}, \quad i = 0, 1, \ldots$$

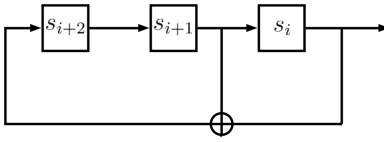



FIGURE 4 – Exemple d'un générateur de degré $m = 3$

Supposons que l'état initial de ce LFSR est $(s_2, s_1, s_0) = (1, 0, 0)$. Nous pouvons calculer le contenu de son état pour huit coups d'horloge consécutifs.

| $i$ | $s_{i+2}$ | $s_{i+1}$ | $s_i$ |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 |
| 6 | 0 | 0 | 1 |
| 7 | 1 | 0 | 0 |
| 8 | 0 | 1 | 0 |

La dernière colonne constitue la sortie du LFSR, c'est donc le début de la suite chiffrante :

$$00101110001011100101110\ldots$$

On remarque que la suite commence à se répéter à partir de $i = 7$. On peut dire alors que la suite est périodique de *période* 7. Même si comme nous allons le voir cette suite est de période maximale pour un LFSR de cette taille, nous aimerons construire des suites avec une période plus longue. Pour cette raison nous utilisons des LFSR de degré plus élevé.

# 8 Description mathématique d'un LFSR

La forme générique d'un LFSR de degré $m$ est dessinée sur la figure **??**. Cette forme présente $m$ positions possibles de rétroaction. Les valeurs $c_i$ sont appelées *coefficients de retroaction* et sont des valeurs binaires.

— Si $c_i = 0$ la valeur correspondante n'est pas prise en compte pour calculer la valeur suivante.
— Si $c_i = 1$ la valeur est alors prise en compte.

Supposons que le contenu initial du registre sont les valeurs $s_0, s_1, \ldots, s_{m-1}$. Les deux bits de sortie suivants du LFSR $s_m$ et $s_{m+1}$, peuvent être calculés à partir des équations suivantes :

$$\begin{aligned} s_m &= c_{m-1}s_{m-1} + \cdots + c_1 s_1 + c_0 s_0 \mod 2, \\ s_{m+1} &= c_{m-1}s_m + \cdots + c_1 s_2 + c_0 s_1 \mod 2. \end{aligned}$$

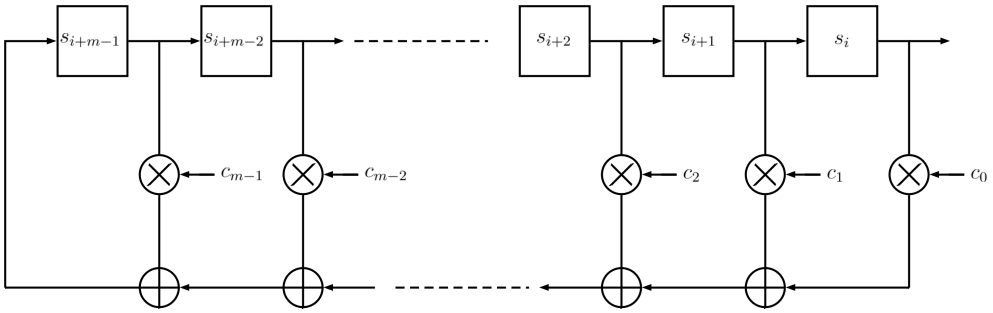



FIGURE 5 – Forme générique d'un LFSR avec des coefficients de rétroaction $c_i$

Nous pouvons donc donner la description mathématique suivante pour le calcul de la suite $s$ :

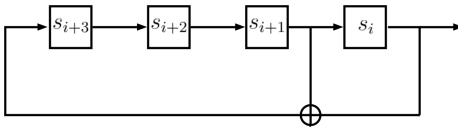
$$s_{i+m} = \bigoplus_{k=0}^{m-1} c_k s_{i+k}, \quad i = 0, 1, 2 \ldots$$

Puisque il n'y a qu'un nombre fini d'états de $m$ bits la suite produite est forcement périodique. Le choix des coefficients $c_i$ infuence la période.

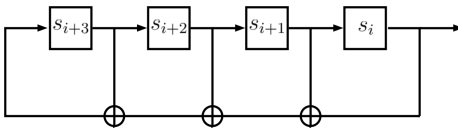
**Théorème 8.1.** *La période maximale générée par un LFSR de degré $m$ est $2^m - 1$.*

La preuve de ce théorème est simple. Le nombre de suites différentes de $m$ bits est $2^m$. Cependant, l'état $(0, 0, \ldots, 0)$ n'est jamais atteint, puisque un tel état aurait produit une suite nulle. Par conséquent le nombre d'états non-nuls possibles est $2^m - 1$. Donc après avoir généré les $2^m - 1$ suites possibles nous rentrons forcement dans une boucle.

**Exemple** Le LFSR suivant de degré 4 et avec des coefficients de retroaction $c_3 = c_2 = 0$, $c_1 = c_0 = 1$ a une période maximale $2^4 - 1 = 15$.




**Exemple** Le LFSR suivant de degré 4 et avec des coefficients de retroaction $c_3 = c_2 = c_1 = c_0 = 1$ a une période égale à 5.




Les LFSR permettent de produire des suites très longues ayant de très bonnes propriétés statistiques et qui nécessitent en plus très peu de ressources pour être construites. Cependant, nous ne pouvons pas utiliser des LFSR tels quels comme générateurs pseudo-aléatoires, en considérant les $m$ bits de l'initialisation comme la clé secrète, car ils ne sont pas imprévisibles. En effet, la connaissance de la description mathématique du LFSR et des premiers $m$ bits nous permettent d'entièrement calculer la suite chiffrante en entier.

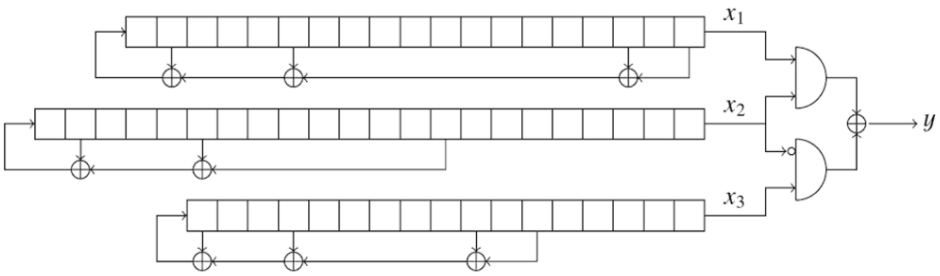
En pratique on combine donc plusieurs LFSR ensemble en y ajoutant en plus quelques composantes non-linéaires.

# 9 Générateur de Geffe

Le générateur de Geffe est un générateur de nombres pseudo-aléatoires, conçu par P. Geffe en 1973 pour le chiffrement à flot. Il emploie trois LFSR de tailles différentes. La clé secrète est constituée par l'ensemble des initialisations des trois registres et sa taille est égale à la somme des tailles (degrés) de ceux-ci. À chaque coup d'horloge, les bits de sortie des trois registres sont combinés de façon non-linéaire afin de produire un bit de la suite chiffrante. Plus précisément, un bit $y$ de la suite chiffrante est calculé de la manière suivante :

$$y = x_1 x_2 \oplus (1 + x_2) x_3,$$

où $x_1$ est le bit de sortie du premier registre, $x_2$ est le bit de sortie du deuxième registre et $x_3$ est le bit de sortie du troisième registre.




Malheureusement, le générateur de Geffe n'est pas cryptographiquement sûr (voir TD).