

HAI709I - Fondements cryptographiques pour la sécurité

Cours 3 - Chiffrements par flot

Rocco Mora

29 Septembre, 2025

Université de Montpellier – Faculté des Sciences
M1 informatique, parcours Algo, IASD, Imagine

Vers des schémas de chiffrement pratiques

La dernière fois, on a vu quelques constructions de schémas de chiffrement à clé privée :

- un schéma EAV-sûr à partir d'un PRG
- un schéma CPA-sûr à partir d'une PRF

PRGs et PRFs sont les “briques” des schémas de chiffrement

Cependant, il y avait quelques inconvénients :

- L'EAV-sécurité est une notion de sécurité faible.
- Dans les deux cas, seuls les messages de longueur fixe pouvaient être chiffrés.
- Le schéma CPA-sûr pourrait être adapté à des messages de longueur arbitraire, mais devenait alors très inefficace (texte chiffré long).

Étudions comment les PRG et les PRF sont instanciés dans le monde réel

Chiffrement de flux

Le chiffrement de flux/par flot est utilisé dans la pratique pour instancier des PRGs.

Chiffrement par flot [🇬🇧 Stream cipher]

Un chiffrement par flux est une paire d'algorithmes déterministes (Init, Next) t.q.

- Init prend en entrée une graine s et un vecteur d'initialisation [🇬🇧 initialization vector] IV facultatif et produit un état initial [🇬🇧 initial state] st .
- Next prend en entrée un état st et produit un bit y et un état mis à jour st' .

GetBits

Entrée : longueur de sortie 1^ℓ , état initial st_0

for $i = 1, \dots, \ell$ **do**

 Calculer $(y_i, st_i) := \text{Next}(st_{i-1})$

end for

return la chaîne de ℓ bits $y = y_1, \dots, y_\ell$ et st_ℓ

GetBits_1 identique, mais ne renvoie que y

Les générateurs pseudo-aléatoires ont une **longueur de sortie fixe**.

- Un chiffrement de flux sûr (Init, Next) **sans** IV n'est qu'un **générateur pseudo-aléatoire plus flexible** : soit $\ell = \ell(n) > n$ et définissons

$$G^\ell(s) \stackrel{\text{def}}{=} \text{GetBits}_1(\text{Init}(s), 1^\ell).$$

Alors le chiffrement de flux est **sûr** si G^ℓ est un générateur pseudo-aléatoire pour tout polynôme ℓ .

- Un chiffrement de flux sûr (Init, Next) **avec un** IV : soit $\ell = \ell(n) > n$ et définissons

$$F_s^\ell(IV) \stackrel{\text{def}}{=} \text{GetBits}_1(\text{Init}(s, IV), 1^\ell).$$

Alors le chiffrement de flux est **sûr** si F^ℓ est une fonction pseudo-aléatoire pour tout polynôme ℓ .

Chiffrement de flux à partir de fonctions pseudo-aléatoires

Construction

Soit F une fonction pseudo-aléatoire. Définissons un chiffrement de flux (Init, Next) comme suit, où Init accepte un IV de $3n/4$ bits et Next produit n bits à chaque appel :

- Init : en entrée $s \in \{0, 1\}^n$ et $IV \in \{0, 1\}^{3n/4}$, sortie $\text{st} = (s, IV, 0)$.
- Next : en entrée $\text{st} = (s, IV, i)$, sortie $y = F_s(IV || \langle i \rangle)$ et état mis à jour $\text{st}' = (s, IV, i + 1)$, où $\langle i \rangle$ est encodé comme un entier de $(n/4)$ bits.

La sortie du chiffrement par flux est alors

$$F_s(IV || \langle 0 \rangle), F_s(IV || \langle 1 \rangle), \dots$$

- + Construction très générale et flexible
- Il existe des solutions plus pratiques

Chiffrement de flux (récap)

PRG avec facteur d'expansion $\ell(n)$:

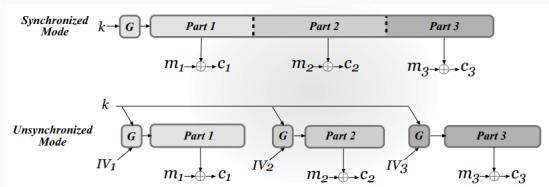
- pas facile de chiffrer des messages de longueur $\ell' > \ell$ avec une seule clé de n bits
- on peut tronquer la sortie pour les messages de longueur $\ell' < \ell$, mais cela représente un gaspillage

Chiffrement par flot : les bits de sortie sont produits progressivement et à la demande.

- + pas de limite supérieure sur les bits pouvant être générés
- + aucun bit inutile n'est généré

Comment chiffrer des messages **de longueur arbitraire** avec un chiffrement de flux ?

Différentes **générations de flux de clés**



Mode synchronisé [🇬🇧 Synchronized mode]

Il suppose que tous les **messages arrivent dans l'ordre** et qu'aucun message n'est perdu (pour les sessions de communication en ligne, p. ex. TCP) :

1. Les deux parties (l'expéditeur A et le destinataire B) appellent $\text{Init}(k)$ pour obtenir le même état initial st_0 .
 2. Soit st_A l'état actuel de A . Si A souhaite chiffrer m , il calcule $(y, st'_A := \text{GetBits}(st_A, 1^{|m|}))$, envoie $c := m \oplus y$ au destinataire et met à jour son état local à st'_A .
 3. Soit st_B l'état actuel de B . Lorsque B reçoit un texte chiffré c de A , il calcule $(y, st'_B := \text{GetBits}(st_B, 1^{|c|}))$, produit le message $m := c \oplus y$ et met à jour son état local à st'_B .
- similaire au générateur pseudo-aléatoire, mais ℓ n'a pas besoin d'être fixé à l'avance et le message n'a pas besoin d'être chiffré en une seule fois.
 - Avec une deuxième clé, B peut également envoyer des messages à A .
 - **Pas besoin d'utiliser un IV** et pas d'expansion du texte chiffré.

Mode non synchronisé [🇬🇧 Unsynchronized mode]

Construction

Soit $(\text{Init}, \text{Next})$ un chiffrement de flux qui prend un IV de n bits. Définissons un schéma de chiffrement à clé privée pour des messages **de longueur arbitraire** :

- Gen : pour une entrée 1^n , produit une sortie uniforme $k \in \{0, 1\}^n$.
- Enc : en entrée $k \in \{0, 1\}^n$ et un message $m \in \{0, 1\}^*$, **choisit un $IV \in \{0, 1\}^n$ uniforme** et produit le texte chiffré

$$(IV, \text{GetBits}_1(\text{Init}(k, IV), 1^{|m|}) \oplus m).$$

- Dec : en entrée $k \in \{0, 1\}^n$ et un texte chiffré (IV, c) , affiche le message

$$m := \text{GetBits}_1(\text{Init}(k, IV), 1^{|c|}) \oplus c.$$

CPA-sûr !

Registres à décalage à rétroaction linéaire

Comment **instancier** un chiffrement de flux dans la pratique ?

Une première solution consiste à utiliser :

Un **registre à décalage à rétroaction linéaire** [🇬🇧 Linear-Feedback Shift Register] (**LFSR**) se compose de

- un tableau de n **registres** $s_{n-1}, \dots, s_0$ et
- une **boucle de rétroaction** [🇬🇧 feedback loop] spécifiée par n **coefficients de rétroaction** [🇬🇧 feedback coefficients] booléens $c_{n-1}, \dots, c_0$.
- La taille du tableau est appelée **degré du LFSR**.
- Chaque registre stocke un seul bit
- L'**état** st d'un LFRS est constitué des bits contenus dans ses registres.
- À chaque cycle, le LFSR produit la valeur du registre le plus à droite s_0 .

Mise à jour d'un LFSR

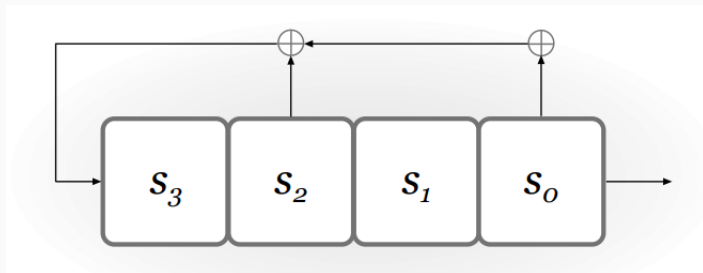
- st est mis à jour par les cycles en **décalant** les valeurs de tous les registres **vers la droite**
- et le registre le plus à gauche est égal au **XOR d'un sous-ensemble du registre actuel**, déterminé par les coefficients de rétroaction. Si, à l'instant t , l'état est $s_{n-1}^{(t)}, \dots, s_0^{(t)}$, alors après le cycle, il est $s_{n-1}^{(t+1)}, \dots, s_0^{(t+1)}$, avec

$$s_i^{(t+1)} := s_{i+1}^{(t)}, \quad i = 0, \dots, n-2, \quad s_{n-1}^{(t+1)} := \bigoplus_{i=0}^{n-1} c_i s_i^{(t)}.$$

- Par conséquent, si $y_0, y_1, \dots$ sont les **bits de sortie**, alors

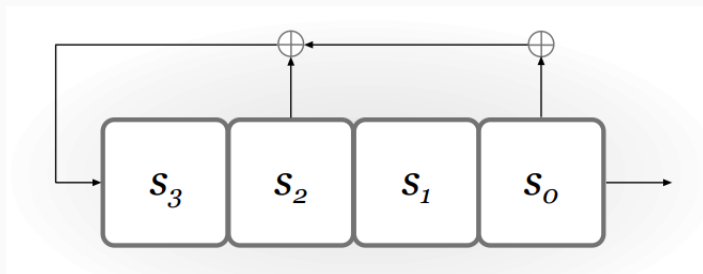
$$y_i = s_i^{(0)}, \quad i = 0, \dots, n-1$$
$$y_i = \bigoplus_{j=0}^{n-1} c_j y_{i-n+j}, \quad i > n-1.$$

Exemple de LFSR



Le LFSR dans l'image a degré 4 avec $c_0 = c_2 = 1$ et $c_1 = c_3 = 0$.

Exemple de LFSR

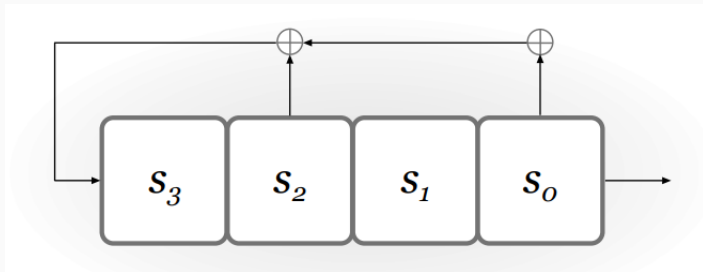


Le LFSR dans l'image a degré 4 avec $c_0 = c_2 = 1$ et $c_1 = c_3 = 0$.

Supposons que l'état initial soit $(s_3, s_2, s_1, s_0) = (0, 0, 1, 1)$. Alors les premiers états sont :

$$(0, 0, 1, 1) \rightarrow (1, 0, 0, 1) \rightarrow (1, 1, 0, 0) \rightarrow (1, 1, 1, 0) \rightarrow (1, 1, 1, 1) \rightarrow \dots$$

Exemple de LFSR



Le LFSR dans l'image a degré 4 avec $c_0 = c_2 = 1$ et $c_1 = c_3 = 0$.

Supposons que l'état initial soit $(s_3, s_2, s_1, s_0) = (0, 0, 1, 1)$. Alors les premiers états sont :

$$(0, 0, 1, 1) \rightarrow (1, 0, 0, 1) \rightarrow (1, 1, 0, 0) \rightarrow (1, 1, 1, 0) \rightarrow (1, 1, 1, 1) \rightarrow \dots$$

et la sortie est le flux de bits $1, 1, 0, 0, 1, \dots$

Les LFSR en tant que chiffrements de flux

Étant donné un LFSR, définissons

- `Init` : prend en entrée une clé k de n bits et définit l'état initial du LFSR à k ;
- `Next` : correspond à un cycle, produisant un seul bit et mettant à jour l'état du LFSR.

Un LFSR de degré n a 2^n états possibles $\Rightarrow$ un état finira par se répéter :

- l'état tout à 0 a une boucle sur lui-même,
- un LFSR est de **longueur maximale** s'il passe par tous les $2^n - 1$ états non nuls avant de se répéter (cela dépend des coefficients).

+ approximativement le même nombre de 1 et de 0 dans la sortie

— pas sûr en tant que chiffrement de flux

Attaque par récupération de clé

- Selon le principe de Kerchoffs, les coefficients de rétroaction doivent être publics. Les premiers n bits de sortie révèlent l'état initial
→ tous les bits de sortie suivants peuvent être calculés
- Que se passe-t-il si c_i font également partie de la clé? L'attaquant observe $2n$ bits $y_0, \dots, y_{2n-1}$. Alors

$$\begin{cases} y_n &= c_{n-1}y_{n-1} \oplus \dots \oplus c_0y_0 \\ y_{n+1} &= c_{n-1}y_n \oplus \dots \oplus c_0y_1 \\ &\vdots \\ y_{2n-1} &= c_{n-1}y_{2n-2} \oplus \dots \oplus c_0y_{n-1} \end{cases}$$

est un système de n équations linéaires à n coefficients de rétroaction inconnus
→ déterminé

Les systèmes linéaires peuvent être résolus en temps polynomial !

Ajout de non-linéarité

Plusieurs approches possibles :

1. Introduire des opérateurs non linéaires,

- XOR est un opérateur linéaire
- AND/OR sont des opérateurs non linéaires

→ obtenir un FSR : étant donné une fonction non linéaire g ,

$$s_i^{(t+1)} := s_{i+1}^{(t)}, \quad i = 0, \dots, n-2$$

$$s_{n-1}^{(t+1)} := g(s_{n-1}^{(t)}, \dots, s_0^{(t)}).$$

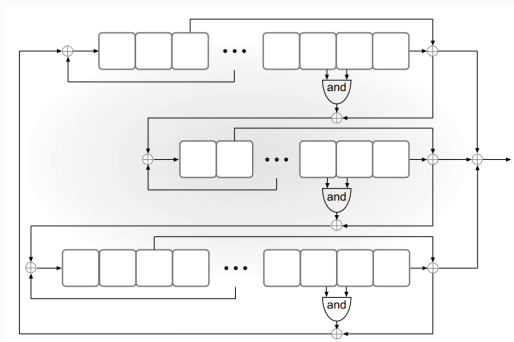
g doit être équilibré : $\Pr(g(s_{n-1}, \dots, s_0) = 1) \approx 1/2$.

- ## 2. Sortie non linéaire
- la sortie à chaque cycle est une fonction non linéaire g (appelée **filtre**) de l'état actuel.
- ## 3. Générateur à combinaison non linéaire
- on utilise plusieurs LFSR et on combine leur sortie de manière non linéaire (plusieurs éléments à prendre en compte pour garantir la sécurité).

Trivium [De Cannière, Preneel, 2005]

Trivium : partie du projet européen eSTREAM

- Trois FSR non linéaires (AND) de degrés 93, 84, 111
→ l'état st est donné par
 $93 + 84 + 111 = 288$ bits
- la sortie est le XOR des trois sorties
- Les 3 FSR sont **couplés** : à chaque cycle, les registres les plus à gauche sont calculés comme des fonctions non linéaires de deux FSR.
- Init prend une clé et un *IV* de 80 bits.



On ne connaît pas de meilleure attaque que la recherche exhaustive !

RC4 [Rivest, 1987]

Init algorithm for RC4

Input: 16-byte key k

Output: Initial state (S, i, j)

(Note: All addition is done modulo 256)

for $i = 0$ to 255:

$S[i] := i$

$k[i] := k[i \bmod 16]$

$j := 0$

for $i = 0$ to 255:

$j := j + S[i] + k[i]$

 Swap $S[i]$ and $S[j]$

$i := 0, j := 0$

return (S, i, j)

GetBits algorithm for RC4

Input: Current state (S, i, j)

Output: Output byte y ; updated state (S, i, j)

(Note: All addition is done modulo 256)

$i := i + 1$

$j := j + S[i]$

Swap $S[i]$ and $S[j]$

$t := S[i] + S[j]$

$y := S[t]$

return $(S, i, j), y$

- Beaucoup plus rapide que les LFSR pour les implémentations logicielles
- Utilisé dans le passé dans la norme de chiffrement Wired Equivalent Privacy (WEP)
- Diverses attaques ont été découvertes → RC4 ne doit pas être utilisé

- Introduit en 2008, il remplace RC4
- Basé sur une **permutation fixe** P sur des chaînes de 512 bits qui utilise uniquement des **instructions au niveau assembleur** sur des mots de 32 bits : addition $\boxplus$ (mod 2^{32}), rotation bit à bit (cyclique), XOR $\rightarrow$ conception très efficace basée sur ARX
- P utilisé pour construire une **fonction pseudo-aléatoire** F prenant une clé de 256 bits, une constante de 128 bits et mappant des entrées de 128 bits vers des sorties de 512 bits :

$$F_k(x) = P(\text{const} || k || x) \boxplus \text{const} || k || x.$$

- Ainsi, la sortie du chiffrement par flux est

$$F_k(IV || \langle 0 \rangle), F_k(IV || \langle 1 \rangle), \dots$$

où $\langle i \rangle$ est encodé sous forme d'entier 64 bits.

- En combinaison avec Poly1305 MAC (nous verrons cela plus tard) pour construire un schéma de chiffrement authentifié utilisé dans **le protocole TLS**