COURS ID12 « INTRODUCTION À LA CRYPTOLOGIE »
COURS 3
CHIFFREMENT SYMÉTRIQUE : CHIFFREMENT PAR BLOC, PAR FLOT

PAUL ZIMMERMANN (CM), LAURENT FOUSSE (TD)

Référence : chapitres 6 et 7 du *Handbook of Applied Cryptography.*

1. Chiffrement par flot

Le principe du *chiffrement par flot* est de chiffrer une suite de caractères (ou octets ou mots-machine) un à la fois, à l'aide d'une transformation qui varie au fur et à mesure du texte. Au contraire, le *chiffrement par bloc* utilise une transformation fixe, sur des blocs plus gros, typiquement 64 ou 128 bits.

L'avantage du chiffrement par flot est qu'il s'implante mieux en hardware, et qu'il ne nécessite pas de zone tampon (*buffer*).

1.0.1. *One-time pad.* C'est le procédé le plus simple. Soit m_i, $i = 1, 2, 3, \ldots$ les caractères (ou octets) du texte clair à chiffrer. On utilise un chiffrement de Vernam :

$$c_i = m_i \oplus k_i,$$

où k_i, $i = 1, 2, 3, \ldots$ est le flot de chiffrement.

Si les k_i sont produits de façon aléatoire et indépendante, on appelle alors ce procédé *one-time pad.* C'est un des seuls procédés prouvés sûrs de façon inconditionnelle. (À titre de comparaison, la sûreté de RSA est conditionnée par la difficulté de la factorisation d'entier.)

Le problème du *one-time pad* est que la clé doit être aussi longue que le message à chiffrer (ou déchiffrer). Une solution est d'utiliser un générateur pseudo-aléatoire pour produire le flot de chiffrement (k_i), à partir d'une clé de petite taille.

On distingue deux classes de chiffrement par flot :

(1) le chiffrement par flot synchrone ;

(2) le chiffrement par flot auto-synchronisant.

1.1. **Chiffrement par flot synchrone.** Le flot de chiffrement est produit indépendamment du texte clair, et du texte chiffré : soit σ_0 l'état initial, f une fonction produisant un nouvel état, g une fonction produisant le flot de chiffrement (z_i), et h la fonction de sortie produisant le texte chiffré c_i à partir de z_i et du texte clair m_i.

$$\begin{aligned} \sigma_{i+1} &= f(\sigma_i, k) \\ z_i &= g(\sigma_i, k) \\ c_i &= h(z_i, m_i) \end{aligned}$$

Le déchiffrement est identique, en remplaçant $c_i = h(z_i, m_i)$ par $m_i = h^{-1}(z_i, c_i)$. Avec un tel procédé, Alice et Bob doivent être synchronisés, c'est-à-dire utiliser la même clé et être au même état σ_i pour un bon déchiffrement. Ainsi, si des caractères 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.

Date: zimmerma@loria.fr,laurent@komite.net.

Par contre, si un caractère est modifié lors de la transmission, il ne pertube pas le déchiffrement des caractères suivants.

Attaques actives : l'ajout ou la suppression d'un caractère par un adversaire actif fait perdre la synchronisation, et donc sera immédiatement détecté par Bob. Par contre, la substitution de quelques caractères peut passer inaperçue auprès de Bob ; il est donc important d'utiliser en plus un mécanisme garantissant l'intégrité des données et/ou l'identité de l'expéditeur.

1.1.1. *Chiffrement par flot additif.* On prend comme fonction de sortie h le « ou exclusif » :

$$\begin{aligned} \sigma_{i+1} &= f(\sigma_i, k) \\ z_i &= g(\sigma_i, k) \\ c_i &= z_i \oplus m_i \end{aligned}$$

1.2. **Chiffrement par flot auto-synchronisant.** Ici, le flot de chiffrement est produit à partir de la clé et d'un nombre fixe de caractères du flot chiffré :

$$\begin{aligned} \sigma_{i+1} &= (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}) \\ z_i &= g(\sigma_i, k) \\ c_i &= h(z_i, m_i) \end{aligned}$$

où l'état initial (public) est $\sigma_0 = (c_{-t}, c_{-t+1}, \ldots, c_{-1})$, k est la clé, g la fonction produisant le flot de chiffrement z_i, et h la fonction de sortie.

Pour le déchiffrement, on remplace $c_i = h(z_i, m_i)$ par $m_i = h^{-1}(z_i, c_i)$. Les caractéristiques du chiffrement par flot auto-synchronisant sont :

- *auto-synchronisation* (sic !) : si des caractères sont perdus ou ajoutés dans le texte chiffré, le procédé de déchiffrement se re-synchronise au bout de t caractères (exercice : le montrer) ;
- *propagation d'erreur* : si un caractère du texte chiffré est modifié, le déchiffrement des t caractères suivants est corrompu (aucun pour un procédé synchrone) ;
- *attaques actives* : la modification d'un caractère se répercutant sur les t caractères suivants, elle a moins de chances de passer inaperçue auprès de Bob. Par contre l'ajout ou la suppression de caractères seront moins bien détectés que pour les procédés synchrones, où tout le reste du texte est perdu.
- *diffusion* : un caractère m_i du texte clair influe via $c_i = h(z_i, m_i)$ et $\sigma_{i+1} = (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1})$ sur toute la suite du texte chiffré. Cela rend plus difficiles les attaques basées sur une analyse statistique utilisant les redondances du texte clair.

1.3. **Registres à décalage.** Les *registres à décalage* sont couramment utilisés pour produire le flot de chiffrement, en particulier les registres à décalage linéaires.

Definition 1. *Un* registre à décalage linéaire *de longueur l est constitué de l cellules, numérotées de 0 à $l-1$, chacune contenant un bit s_i, ayant une entrée et une sortie, et d'une horloge. À chaque top d'horloge :*

- *le bit de la cellule 0 est sorti, pour former le flot de sortie ;*
- *le contenu de la cellule i est déplacé en cellule $i-1$, pour $1 \leq i \leq l-1$;*
- *le contenu de la cellule $l-1$ est obtenu en formant le « ou exclusif » de valeurs précédentes de $s_0, \ldots, s_{l-1}$:*

$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + \cdots + c_l s_{j-l}) \bmod 2.$$

On associe à un registre à décalage linéaire le polynôme générateur $1 + c_1 x + c_2 x^2 + \cdots + c_l x^l$.

Exemple. Le registre à décalage associé au polynôme $1 + x + x^4$ est :

$$s_j = s_{j-1} + s_{j-4} \bmod 2.$$

Avec $s_{-4} = s_{-3} = s_{-2} = s_{-1} = 0$, ce registre ne produit que des 0. Par contre, avec $[s_{-4}, \ldots, s_{-1}] = [0, 1, 1, 0]$, il produit :

i	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s_i	0	1	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1	1	0

Comme $s_{-4} = s_{11}$, $s_{-3} = s_{12}$, $s_{-2} = s_{13}$, et $s_{-1} = s_{14}$, la suite (s_i) est périodique, de période 15 : $s_i = s_{i+15}$.

Si le polynôme $f(x)$ associé à un registre à décalage linéaire est *irréductible* sur $\mathbb{F}_2$ — le corps fini à deux éléments —, et de degré l, alors chacun des $2^l - 1$ états initiaux possibles où les cellules ne sont pas toutes nulles produit une suite de période k, où k est le plus petit entier tel que $f(x)$ divise $x^k + 1$ dans $\mathbb{F}_2[x]$. (Note : k est toujours un diviseur de $2^l - 1$.)

Exemple : pour $f(x) = x^4 + x + 1$, on a $l = 4$, et le plus petit k tel que $f(x)$ divise $x^k + 1$ est $k = 15$. Comme $2^l - 1 = 15$, c'est en fait la période maximale qu'on peut obtenir avec un polynôme de degré 4.

Definition 2. *Soit p premier, et $f(x)$ un polynôme de degré l sur $\mathbb{F}_p$: $f(x)$ est dit* primitif *si les $x^i \bmod f$, $0 \leq x < p^l - 1$, engendrent tous les polynômes non nuls de degré inférieur à l sur $\mathbb{F}_p$.*

Exemple. Le polynôme $x^4 + x + 1$ est primitif sur $\mathbb{F}_2$, mais pas $x^4 + x^3 + x^2 + x + 1$, qui divise $x^5 + 1$.

Si le polynôme $f(x)$ associé à un registre à décalage linéaire est *primitif* sur $\mathbb{F}_2$, alors tout état initial non nul produit une suite de période maximale $2^l - 1$, où $l = \deg(f)$.

Les registres à décalage basés sur des polynômes primitifs sont donc de bons candidats. On peut montrer que tout mot binaire de k bits apparaît avec probabilité 2^{-k} sur la suite infinie (s_i).

Definition 3. *La complexité linéaire d'une suite infinie s de bits, notée $L(s)$, est :*
- *$L(s) = 0$ si $s_i = 0$ pour tout i ;*
- *$L(s) = \infty$ si aucun registre à décalage linéaire ne produit s ;*
- *$L(s) = n$ si le plus petit registre à décalage linéaire produisant s a longueur n.*

L'algorithme de Berlekamp-Massey détermine la complexité linéaire $L(s) \leq n$ d'une suite s à partir des n premiers éléments, en $O(n^2)$ opérations.

Remarque : il existe aussi des registres à décalage non linéaire. Par exemple $s_j = 1 \oplus s_{j-2} \oplus s_{j-3} \oplus s_{j-1}s_{j-2}$, qui produit la suite de de Bruijn.

Un exemple de procédé de chiffrement par flot est SEAL (*Software-optimized Encryption Algorithm*), qui date de 1993. SEAL a été conçu spécifiquement pour une implantation efficace en software, plus particulièrement sur architecture 32 bits. Il chiffre une suite de mots de 32 bits, avec une clé de 160 bits. Chaque octet du flot de chiffrement ne nécessite que 5 instructions, soit environ 10 fois moins que DES.

2. Chiffrement par bloc

Definition 4. *Un procédé de* chiffrement par bloc *sur n bits est une fonction $E : \{0,1\}^n \times K \to \{0,1\}^n$ telle que pour toute clé $k \in K$, $E(\cdot, k)$ est une bijection de $\{0,1\}^n$ dans lui-même, notée E_k. La fonction inverse est notée D_k.*

Pour m un texte clair de n bits, on note $c = E_k(m)$ le chiffré correspondant, et donc $m = D_k(c)$.

Une différence majeure par rapport au chiffrement par flot, et qu'ici n est relativement grand (disons $n \geq 64$).

Différents types d'attaque :

– *à texte chiffré seul* : Charlie n'a accès qu'au message chiffré. C'est l'hypothèse la plus restrictive pour l'attaquant ;
– *à texte clair connu* : Charlie connaît des paires (m_i, c_i) de textes clairs et les chiffrés associés ;
– *à texte clair choisi* : Charlie peut choisir des textes clairs m_i, et obtenir (ou calculer) le chiffré c_i correspondant. C'est l'hypothèse la plus favorable pour l'attaquant.

Il est recommandé d'utiliser des procédés résistant aux attaques à texte clair choisi, même si une telle attaque n'est pas possible pour Charlie. En effet, de tels procédés résistent aux attaques à texte chiffré seul et à texte clair connu.

On caractérise la complexité des attaques par :
– la complexité en espace : c'est le nombre de données nécessaires, par exemple de paires clair-chiffré. Pour des messages sur n bits, on peut toujours monter une attaque en espace 2^n : en effet, il suffit de tabuler les 2^n paires possibles (m_i, c_i), puis pour c donné de rechercher quel m correspond.
– la complexité en temps : c'est le temps de calcul nécessaire. Pour une clé de k bits, une attaque en temps 2^k est toujours possible. En effet, pour une paire (m, c) donnée, il suffit d'essayer les 2^k clés possibles, jusqu'à trouver $E_K(m) = c$.

On dit qu'un algorithme de chiffrement par bloc est *sûr* lorsqu'on ne connaît pas d'attaque de complexité sensiblement inférieure à 2^n en espace, ou à 2^k en temps.

Les critères d'évaluation des procédés de chiffrement par bloc sont :
– le niveau de sécurité estimé (souvent heuristique, proportionnel au nombre d'années durant lequel le procédé a résisté aux attaques des cryptanalystes) ;
– la taille de clé k (cf. recherche exhaustive sur la clé) ;
– la taille de bloc n ;
– le débit (souvent inversement proportionnel à la taille de clé et à la taille de bloc) ;
– la complexité de l'implantation : celle-ci peut varier suivant que c'est une implantation hardware ou software ;
– le facteur d'expansion : certains procédés peuvent augmenter la taille du chiffré par rapport au clair ;
– la propagation d'erreurs d'un bloc à l'autre.

On distingue 4 modes opératoires : ECB (*electronic codebook*), CBC (*cipher-block chaining*), CFB (*cipher feedback*), et OFB (*output feedback*). Par exemple, le mode ECB fonctionne comme suit :
– Chiffrement : $c_j = E_k(m_j)$;
– Déchiffrement : $m_j = E_k^{-1}(c_j)$.

Deux blocs identiques (avec la même clé k) produisent le même chiffré. Les blocs sont chiffrés indépendamment les uns des autres ; une permutation des blocs chiffrés produit donc la même permutation des blocs clairs. Une erreur de transmission sur un bloc chiffré n'affecte que ce bloc.

Quant à lui, le mode CBC fait dépendre c_j du chiffré précédent :
– Chiffrement : $c_j = E_k(c_{j-1} \oplus m_j)$;
– Déchiffrement : $m_j = c_{j-1} \oplus E_k^{-1}(c_j)$.

Ici, deux blocs identiques (même avec la même clé k) produisent des chiffrés différents. Une permutation des blocs chiffrés empêche un déchiffrement correct. Une erreur sur c_j affecte le déchiffrement de c_j et de c_{j+1} (exercice : montrer que c_{j+2} n'est pas affecté).

Les exemples classiques de chiffrement par bloc sont DES (*Data Encryption Standard*, $k = 56$, $n = 64$), FEAL (*Fast Data Encipherment Algorithm*, $k = n = 64$), IDEA (*International Data Encryption Algorithm*, $k = 128$, $n = 64$), RC5 (RC5-32 pour des mots de 32 bits, clé de 16 octets soit 128 bits), AES (cf prochain cours).