Cryptographie

Chiffrement à clé secrète

Gabriel Chênevert

7 novembre 2025

Aujourd'hui

Chiffrement de flux

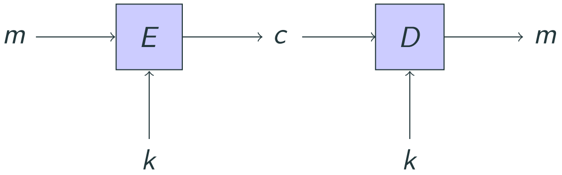
Chiffrement par bloc

Modes opératoires

Rappel : Chiffrement symétrique

Un *chiffre symétrique* est composé d'une paire de fonctions

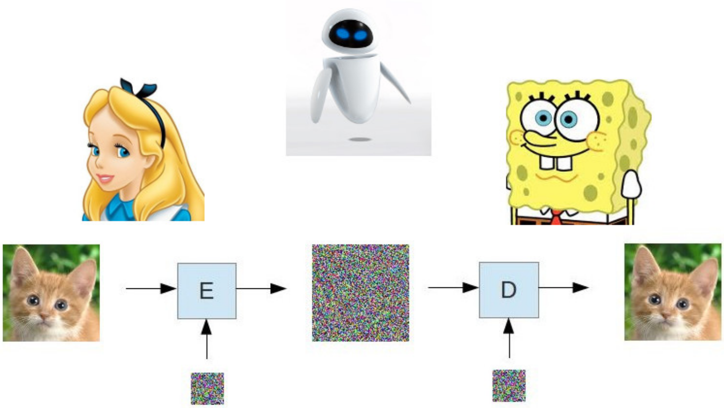
$$E : \mathcal{K} \times \mathcal{M} \longrightarrow \mathcal{C} \qquad \text{et} \qquad D : \mathcal{K} \times \mathcal{C} \longrightarrow \mathcal{M}$$



où

- $\mathcal{M}$: ensemble des messages en clair ;
- $\mathcal{C}$: ensemble des messages chiffrés ;
- $\mathcal{K}$: ensemble des clés possibles.

Rappel : chiffrement symétrique



Propriétés requises

- *Déchiffrement correct* : pour tout $k \in \mathcal{K}$ et $m \in \mathcal{M}$,

$$D(k, E(k, m)) = m.$$

- *Confidentialité* : la connaissance du message chiffré c (sans celle de k) ne doit pas permettre de retrouver en pratique (ni même aider à retrouver) m

Rappel : Chiffre de Vernam

Cas particulier du chiffrement symétrique où :

- $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$
- $E(k,m) = m \oplus k$
- $D(k,c) = c \oplus k$

$\implies$ confidentalité parfaite puisqu'étant donné c, m pourrait être n'importe quoi

(n'importe quel message m, chiffré avec la clé $k = m \oplus c$, pourrait être à l'origine de c)

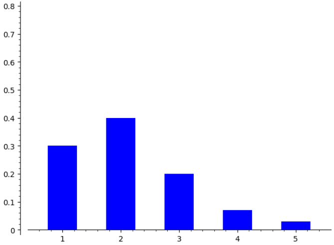
Exemple (confidentialité imparfaite)

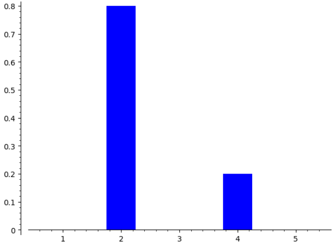
Bob : Combien de hot-dogs désires-tu ?

Alice chiffre $m \in \{1, 2, 3, 4, 5\}$ en lui ajoutant un grand entier *pair* $2k$.

Ève intercepte le message chiffré $c = 876523987428763529987687 4$

Son estimation des possibilités pour m change : elle a obtenu de l'*information*.

Avant : 

Après : 

Chiffrement de flux binaire

Un chiffrement par masque

- $E(k, m) = m \oplus \mathrm{pad}(k)$
- $D(k, c) = c \oplus \mathrm{pad}(k)$

avec $\mathcal{M} = \mathcal{C} = \{0, 1\}^n$ (arbitrairement grand), $\mathcal{K} = \{0, 1\}^\ell$ (petit)

et une fonction de génération de masque

$$\mathrm{pad} : \{0, 1\}^\ell \longrightarrow \{0, 1\}^n,$$

un *générateur de nombres pseudo-aléatoires sécurisé* (CSPRNG)

Générateur de nombres pseudo-aléatoires

Entrée [1]:

```python
import random

# uses *insecure* but efficient Mersenne Twister PRNG

random.seed(12345)

for i in range(16):

    print(hex(random.randint(0,2**128))[2:])
```

```
6facaa5090e5e945452ec40a3193ca54
6ed4e94bdfc9e3b11fcff4545f811cb9
bc428d42fa88269287f26aee175f0cd2
25ece8452aa4857e8101e89a95c5fb98
d64a3ce030a1f6d513ed748bb80e3b0d
56eaa3017576714a06057c82527122dc
94820a06c555663f29ef41d0deea959e
6a1eccdaa70ce1b51978cec0495cfa4f
df8960ad1eab5cd83b788b660a4de3e4
96af0dea41fad2962f927291ab721ab0
213f191ff56ae7eaea80db0684ab5616
f70ae8c026784184026530cdd50b612b
282fe557578b24268a04f74f5987baf1
9f3180427b1427081f1af1fac2e1dac4
265015788e7ae9af1e8fcb74b2d4f32a
f79fcaa0e47b342b2a3a46677eb14f84
```

Sécurité d'un générateur de nombres pseudo-aléatoires

On veut que la génération du masque soit *imprévisible* pour éviter que la connaissance de bits passés du masque ne permette d'en connaître des bits futurs.

- *Tous* les générateurs de nombres pseudo-aléatoires sont périodiques à partir d'un certain point

  (fonction à état déterministe avec un nombre fini d'états internes)

  $\Longrightarrow$ on veut que cette période soit longue

- La plupart des PRNG usuels sont facilement prévisibles !

Exemple : Fiasco des mots de passes prévisibles de Kaspersky (2021)

Générateur congruentiel linéaire (LCG)

Définition

À partir d'une *graine* x_0, génère une suite pseudo-aléatoire $(x_n)_{n=1}^{\infty}$ avec

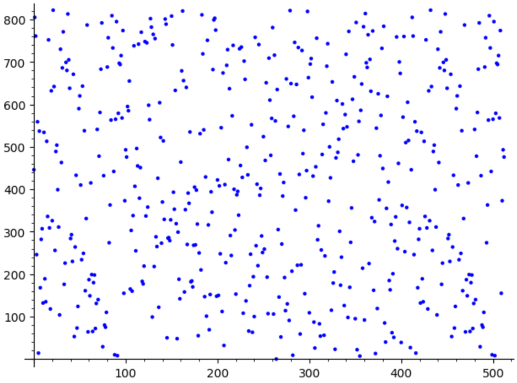
$$x_{n+1} = (ax_n + b) \,\%\, p$$

où a, b sont des constantes (entières) et p un nombre premier.

i.e. suite d'entiers mod p obtenue par itération de la fonction affine

$$f(x) \underset{p}{\equiv} ax + b.$$

Exemple : $p = 823,\ a = 816,\ b = 635,\ x_0 = 446$



Cryptanalyse d'un LCG

Problème : un LCG ne fait rien pour masquer ses paramètres internes

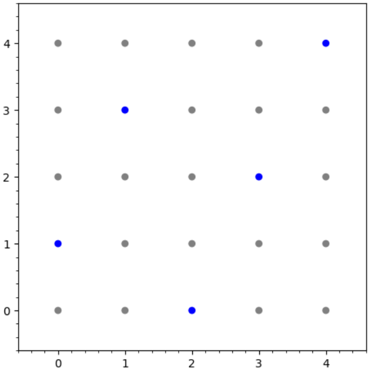
La connaissance de trois termes de sortie successifs permet de retrouver a et b !

Indice : tous les points (x_n, x_{n+1}) sont sur une même droite $y \underset{p}{\equiv} ax + b \ldots$

Exemple

Déterminer a et b pour le LCG mod 5 qui donne $x_1 = 1$, $x_2 = 3$, $x_3 = 2$.

Le LCG $y \underset{5}{\equiv} 2x + 1$



À partir de la graine (secrète ?) $x_0 = 0$, on obtient

$$
\begin{aligned}
x_1 &= 1 \\
x_2 &= 3 \\
x_3 &= 2 \\
x_4 &= 0 \\
x_5 &= 1 \\
x_6 &= 3 \\
&\vdots
\end{aligned}
$$

En pratique

On peut effectuer des constructions semblables avec des polynômes, des matrices, ...

(par exemple le célèbre *Mersenne Twister*, 1997)

mais l'état et les paramètres internes sont facilement récupérables en résolvant des équations linéaires

$\Longrightarrow$ on compose des PRNG algébriques de façon non-linéaire pour « casser » les équations

Recommendations actuelles

Le projet eSTREAM (ECRYPT 2008) propose

- HC-128, Rabbit, Salsa/Chacha20, SOSEMANUK (orientation logicielle)
- Grain, MICKEY, Trivium (orientation matérielle)

(qui forcent tous le PRNG à utiliser une valeur à usage unique (*nonce*) comme graine)

Attention : inutile d'utiliser un générateur imprévisible avec une clé prévisible !

$\Longrightarrow$ importance que la clé k soit obtenue à partir d'un *vrai* aléa

Exemple : Le mur d'entropie chez Cloudflare

Malléabilité

Chiffrement par masque :

$$E(k, x) = D(k, x) = x \oplus \mathrm{pad}(k)$$

Avantage : simplicité

Inconvenient : simplicité !

Notamment : si $c = E(k, m)$,

$$E(k, m \oplus y) = E(k, m) \oplus y$$
$$D(k, c \oplus y) = D(k, c) \oplus y$$

Différents modèles d'attaque

Ève (attaquante passive) : voit passer un message chiffré, n'apprend rien ✓

Oscar (attaquant actif) : peut manipuler le chiffré et avoir un objectif différent !

Aujourd'hui

Chiffrement de flux

Chiffrement par bloc

Modes opératoires

Changement de point de vue

Considérons (E, D) un chiffre symétrique avec $\mathcal{M} = \mathcal{C} = \{0,1\}^n$

Pour $k \in \mathcal{K}$,

$$E_k := E(k, \cdot\,) : \mathcal{M} \longrightarrow \mathcal{M}$$

admet $D_k := D(k, \cdot\,)$ comme inverse

donc E_k est une **permutation** de $\mathcal{M}$ (bijection de $\mathcal{M}$ vers $\mathcal{M}$)

E_k en tant que permutation

par exemple avec $|\mathcal{M}| = 2^8$:

$$
\begin{array}{ccc}
\mathcal{M} & & \mathcal{M} \\
00000000 & & 11100110 \\
00000001 & & 01000101 \\
00000010 & \xrightarrow{\quad E_k \quad} & 01001011 \\
00000011 & & 11111110 \\
\vdots & & \vdots \\
11111101 & & 11101110 \\
11111110 & & 00010100 \\
11111111 & & 11011110
\end{array}
$$

Idée

On pense à E_k comme une *permutation pseudo-aléatoire* de $\mathcal{M}$.

In pratique : indistinguable d'une fonction aléatoire $\mathcal{M} \to \mathcal{M}$.

Cela permettra de :

- réutiliser les clés (avec quelques précautions)
- découper un message en petits blocs qui seront tous chiffrés avec la même clé

Construction

Paradigme de Shannon : *confusion* et *diffusion*

Essentiellement, toutes les constructions modernes utilisent une conception itérative dans laquelle le message est chiffré plusieurs fois par une *fonction de tour* apportant à chaque étape un peu de confusion et de diffusion

$$\begin{cases} x_0 = m, \\ x_{i+1} = R(k_{i+1}, x_i), \quad 0 \leq i < r \\ E(k, m) = x_r \end{cases}$$

précédé d'un processus de préparation de clés $k \mapsto (k_1, \cdots, k_r)$.

Exemples célèbres

blocs de taille n, clés de taille ℓ, r tours

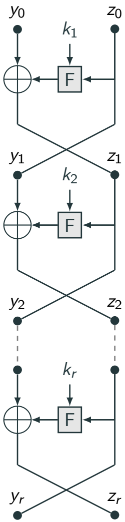
- **Lucifer** (IBM, 1971) $n = \ell = 128$, $r = 16$
- **Data Encryption Standard** (NIST, 1977) $n = 64$, $\ell = 56$, $r = 16$

  Attaque par force brute en 1997 !
- **Rijndael** (KU Leuven, 1998) *aka* **Advanced Encryption Standard** (NIST, 2001)

  $n = 128$, $\ell \in \{128, 192, 256\}$, $r \in \{10, 12, 14\}$.

Mais aussi : RC5/RC6, IDEA, Serpent, Blowfish/Twofish, ...

Schéma de conception commun à Lucifer/DES/AES



Réseau de Feistel à r tours :

On décompose chaque $x_i = y_i \parallel z_i$ en parties gauche et droite

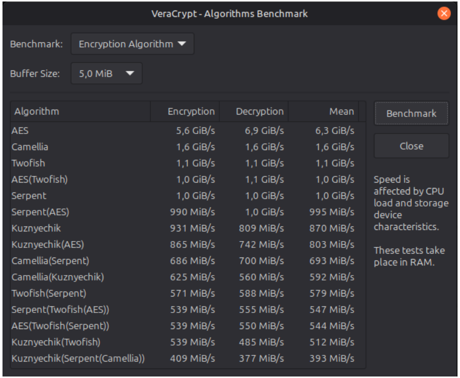
Fonction de tour :

$$\begin{cases} y_{i+1} = z_i \\ z_{i+1} = y_i \oplus F(k_{i+1}, z_i) \end{cases}$$

Facile à implémenter matériellement
(et à inverser pour réaliser un déchiffreur !)

Recommendation

En pratique : **utiliser AES** ou un autre finaliste NIST / alternative internationale



Aujourd'hui

Chiffrement de flux

Chiffrement par bloc

Modes opératoires

Modes opératoires

Pour un message composé de multiples blocs :

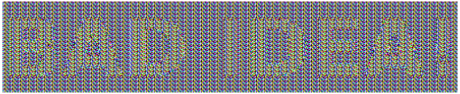
$$m = m_1 \parallel m_2 \parallel m_3 \parallel \cdots$$

Comment utiliser un chiffrement par blocs (ex. AES) pour chiffrer m ?

- mode **ECB** (*Electronic Code Book*) :

$$\begin{cases} c_i = E(k, m_i) \\ m_i = D(k, c_i) \end{cases}$$

Mode ECB ?



Problème : des blocs égaux donnent des blocs chiffrés égaux

On doit utiliser un chiffrement (pseudo-)probabiliste :

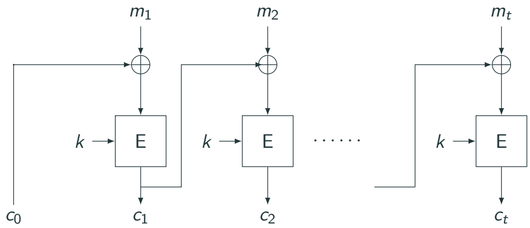
un bloc donné ne doit pas toujours être chiffré de la même façon

$\rightsquigarrow$ utilisation d'un nonce : valeur aléatoire ou compteur

Deux des modes les plus simples (parmi des dizaines) :

Mode CBC (Cipher Block Chaining)

$$\begin{cases} c_0 = \text{valeur initiale (IV) aléatoire} \\ c_i = E(k, m_i \oplus c_{i-1}) \end{cases}$$



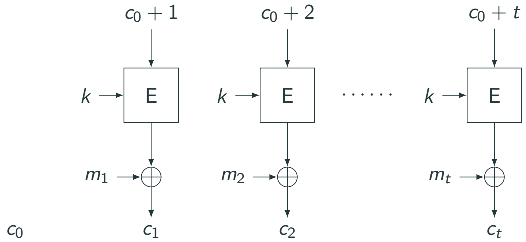
- Chiffrement séquentiel (mais le déchiffrement peut être parallélisé)

$$m_i = D(k, c_i) \oplus c_{i-1}$$

- La longueur du message doit être multiple de la taille des blocs
- Crucial que la valeur initiale c_0 soit non prévisible

Mode CTR (Randomized Counter)

$$\begin{cases} c_0 = \text{IV aléatoire} \\ c_i = m_i \oplus E(k, c_0 + i) \end{cases}$$



- Le chiffrement par blocs est utilisé comme un chiffrement de flux
- Pas de contrainte sur la taille du message
- Fortement parallélisable
- La valeur initiale aléatoire permet d'éviter la réutilisation du flux de masque

En résumé

- Deux approches pour implémenter en pratique le chiffrement à clé secrète pour garantir la confidentialité des messages : **chiffrement de flux** (comme Chacha20) et **chiffrement par bloc** (comme AES)
- Pour un même chiffrement par bloc, des **modes opératoires** (comme CBC ou CTR) auront des caractéristiques et propriétés techniques différentes
- Le chiffrement en lui-même ne garantit pas l'*authenticité* du message (manipulation par un attaquant actif)
- Restera également à s'intéresser au problème de la distribution des clés...