Introduction à la Cryptographie

2. Chiffrement symétrique (1)

Cécile Pierrot, Chargée de Recherche INRIA Nancy
cecile.pierrot@inria.fr

Supports de E. Thomé

Telecom Nancy, 2A ISS – 2021

Plan

Penser à l'attaquant

On distingue plusieurs types d'attaque :

- Selon le matériau disponible :
  - attaque à chiffré seul
  - attaque à clair connu
  - attaque à clair choisi (adaptative ou non)
    (toujours possible pour les chiffrements à clé publique)
  - attaque à chiffré choisi (adaptative ou non)
  - attaque à clés liées
- Selon l'accès au matériel chiffrant disponible :
  - cryptanalyse par canaux auxiliaires
  - cryptanalyse par injection de fautes

Penser à l'attaquant

D'une manière générale

On n'a pas le droit de se méprendre sur ce que sait l'attaquant.

- L'attaquant sait toujours quel cryptosystème on utilise.
- L'attaquant a accès illimité au canal de communication (interposition, injection).
- L'attaquant a souvent accès à une certaine forme d'oracle où il peut "utiliser" Alice ou Bob d'une façon limitée.

Les objectifs de l'attaquant sont variés. Le moins ambitieux (et le plus difficile à combattre) est celui d'un distingueur.

Qu'est-ce qu'un système sûr ?

Définition (Sécurité inconditionnelle)

- Confidentialité parfaite [Shannon, 1949]
  La connaissance du message chiffré n'apporte aucune information sur le message clair : la seule attaque possible est la recherche exhaustive.
  Pour qu'un chiffrement soit inconditionnellement sûr, il faut que la clé soit aléatoire et aussi longue que le texte clair.
- Authentification parfaite [Simmons, 1984]
  Théorie de l'authentification à clé secrète à usage unique.

C'est une sort de ≪modèle idéal≫, largement irréaliste.

Théorèmes ?

Les systèmes utilisés dans la pratique sont théoriquement cassables

Définition (Sécurité pratique, ou calculatoire)

La connaissance du message chiffré (et de certains couples clairs-chiffrés) ne permet de retrouver ni la clé ni le message clair en un temps humainement raisonnable.

- Sécurité inconditionnelle $\Rightarrow$ cryptographie à clé secrète
- La cryptographie à clé publique est rendue possible par la sécurité pratique. Elle utilise deux notions clés :
  - fonction à sens unique
  - fonction à sens unique avec trappe

Complexité d'une attaque : ordres de grandeur

$$\text{Difficulté de l'attaque} = O(\text{Temps} + \text{Mémoire} + \text{Données})$$

- On estime que requérir 2^{128} opérations représente aujourd'hui un niveau raisonnable de sécurité (≪limite de l'infaisable≫)
- Jusqu'au début des années 2000, on fixait plus couramment cette limite à 2^{80}.

Attention à la parallélisation des algorithmes.

Complexité d'une attaque : ordres de grandeur

1000 coeurs à 4GHz pendant 1 an : $\approx 2^{67}$ cycles.

n	2^n	**Exemples**
32	2^{32}	nombre d'êtres humains sur Terre
46	2^{46}	distance Terre - Soleil en millimètres nombre d'opérations effectuées par jour par un ordinateur à 1 GHz
55	2^{55}	nombre d'opérations effectuées par an par un ordinateur à 1 GHz
82	2^{82}	masse de la Terre en kilogrammes
90	2^{90}	nombre d'opérations effectuées en 15 milliards d'années (âge de l'univers) par un ordinateur à 1 GHz
155	2^{155}	nombre de molécules d'eau sur Terre
256	2^{256}	nombre d'électrons dans l'univers

Niveau de sécurité : taille des clés

- Chiffrement réputé sûr si aucune attaque n'a une complexité significativement inférieure à la recherche exhaustive.
- Clé symétrique : la taille des clés est souvent de 128 bits (AES)
- Clé asymétrique : la taille des clés est calculée de manière à offrir une sécurité supérieure à 2^{128}
  ex. 3072 bits pour un module RSA
- Problème pratique : plus la taille des clés augmente, plus les algorithmes sont lents surtout en cryptographie asymétrique

Plan

César

Le chiffrement de César :

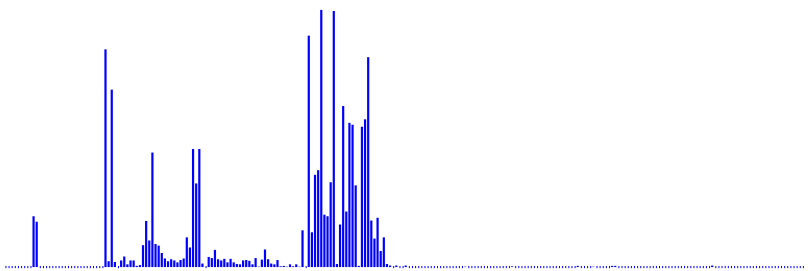
- message clair constitué de lettres (26).
- clé = un décalage constant.
- message chiffré : chaque lettre décalée.

Inconvénients :

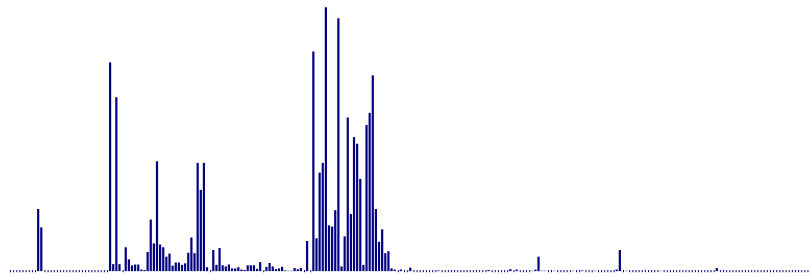
- L'espace de clés est trop faible.
- Le même décalage est utilisé pour toutes les lettres : les propriétés statistiques sont préservées.

Propriétés statistiques

Sur un texte, les fréquences d'occurence des différents symboles sont très différentes. Exemple sur 100 pages au hasard de Wikipedia (caractères ascii de 0 à 256).



Anglais



Français

Si on se restreint aux lettres de l'alphabet, ça marche aussi.

Vigénère

Un peu mieux que César.

- message clair constitué de lettres (26).
- Une clé constituée de K valeurs de décalage $\delta_1, \ldots, \delta_K$.
- message chiffré : $c_{Ki+j} = \text{'a'} + ((m_{Ki+j} - \text{'a'}) + \delta_j) \bmod 26$.

Qu'est-ce que ça change ?

- La recherche exhaustive des clés est plus difficile (si K est grand),
- mais pour j constant, les lettres m_{Ki+j} sont décalées pareil : les propriétés statistiques sont préservées si on « saucissonne ».

Vu autrement

Une autre façon de voir César.

- Message = suite d'éléments de $\mathbb{Z}/26\mathbb{Z}$.
- Clé = un décalage modulo 26.

Une autre façon de voir Vigénère, avec une clé de longueur k.

- Message = suite d'éléments de $(\mathbb{Z}/26\mathbb{Z})^k$.
- Clé = un vecteur de décalage dans $(\mathbb{Z}/26\mathbb{Z})^k$.
- Sauf que la longueur de clé fait aussi partie de la clé. Mais ça ne fait qu'ajouter un coût polynomial, pas exponentiel.

En substance, c'est donc assez semblable.

- Tous les éléments subissent le même traitement $\Rightarrow$ faiblesse.
- On retrouve d'ailleurs ce genre de faiblesse plus tard (ECB).

Abstraction

Pourquoi considérer qu'un message $=$ des éléments de $\mathbb{Z}/26\mathbb{Z}$?

- . . . aucune raison.
- Beaucoup plus génériquement, on considère qu'un message est une suite d'octets, voire de bits. Cela permet d'abstraire le contenu.

Plan

Vernam

Le chiffrement de Vernam, a.k.a. One-Time-Pad (1917).

- Phase initiale. Alice et Bob créent et se partagent une suite aléatoire ≪ infinie ≫ de bits qui constituent la clé.

$$k_0, k_1, \ldots k_N.$$

- Communication. Supposons que les bits de clé $k_0, \ldots k_{\alpha-1}$ ont déjà servi.
  - Alice chiffre son message $m_0, \ldots, m_{b-1}$ de b bits :

$$(c_0, \ldots, c_{b-1}) = (m_0 \oplus k_\alpha, \ldots, m_{b-1} \oplus k_{\alpha+b-1}).$$

  - Bob se rappelle qu'on en était à k_α, et déchiffre :

$$(m_0, \ldots, m_{b-1}) = (c_0 \oplus k_\alpha, \ldots, c_{b-1} \oplus k_{\alpha+b-1}).$$

Vernam

Le chiffrement de Vernam possède un avantage unique.

- Si la suite (k_i) est purement aléatoire, alors il n'y a aucun espoir de retrouver (m_i) à partir de (c_i).
  - Si $c_i = 0$: $\Pr(m_i = 0) = \Pr(k_i = 0) = \frac{1}{2}$,
    et $\Pr(m_i = 1) = \Pr(k_i = 1) = \frac{1}{2}$.
  - Si $c_i = 1$: $\Pr(m_i = 0) = \Pr(k_i = 1) = \frac{1}{2}$,
    et $\Pr(m_i = 1) = \Pr(k_i = 0) = \frac{1}{2}$.
- Attention : ça ne vaut que si chaque bit k_i est utilisé une unique fois. Sinon les probabilités sur k_i ne sont plus indépendantes.

Le chiffrement de Vernam possède un inconvénient unique.

- Pour chiffrer 1Mo de messages, il faut 1Mo de clé.

Quand on est à court de clé... on recycle : mauvaise idée !

Recyclage de clé dans OTP

Si on suppose qu'il y a eu recyclage, on peut chercher deux messages chiffrés avec des suites non disjointes des bits de la clé.

$$(c_0, \ldots, c_{b-1}) = (m_0 \oplus k_\alpha, \ldots, m_{b-1} \oplus k_{\alpha+b-1}),$$
$$(c'_0, \ldots, c'_{b-1}) = (m'_0 \oplus k_\beta, \ldots, m'_{b-1} \oplus k_{\beta+b-1}).$$

On suppose $[\alpha, \alpha + b[\cap [\beta, \beta + b[= [\gamma, \gamma + r[$:

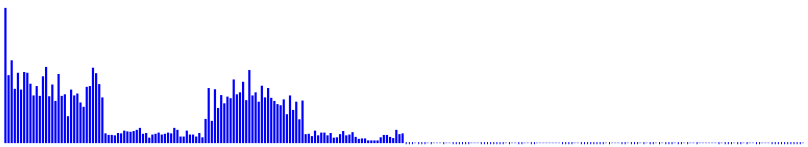
$$(c_{\gamma-\alpha}, \ldots, c_{\gamma-\alpha+r-1}) = (m_{\gamma-\alpha} \oplus k_\gamma, \ldots, m_{\gamma-\alpha+r-1} \oplus k_{\gamma+r-1}),$$
$$(c'_{\gamma-\beta}, \ldots, c'_{\gamma-\beta+r-1}) = (m'_{\gamma-\beta} \oplus k_\gamma, \ldots, m'_{\gamma-\beta+r-1} \oplus k_{\gamma+r-1}).$$

On a alors $c_{\gamma-\alpha+i} \oplus c'_{\gamma-\beta+i} = m_{\gamma-\alpha+i} \oplus m'_{\gamma-\beta+i}$, qui n'est pas aléatoire.

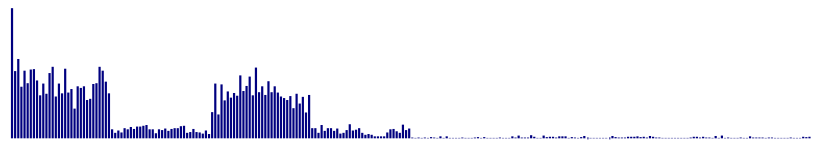
Propriétés statistiques

Quelle est la distribution statistique de $m_\alpha \oplus m_\beta$, où m_α et m_β sont deux lettres d'un texte ?

Exemple sur 100 pages au hasard de Wikipedia (ascii, 0 à 256).



Anglais



Français

- Si des bits de clé sont réutilisés, on peut chercher les alignements et retrouver des fragments de messages.
- Projet Venona (1943-1980) : étant donné un corpus de messages, tenter de deviner ces alignements : du calcul !
- Difficile quand les messages chiffrés sont courts !

Plan

Générer de l'aléa

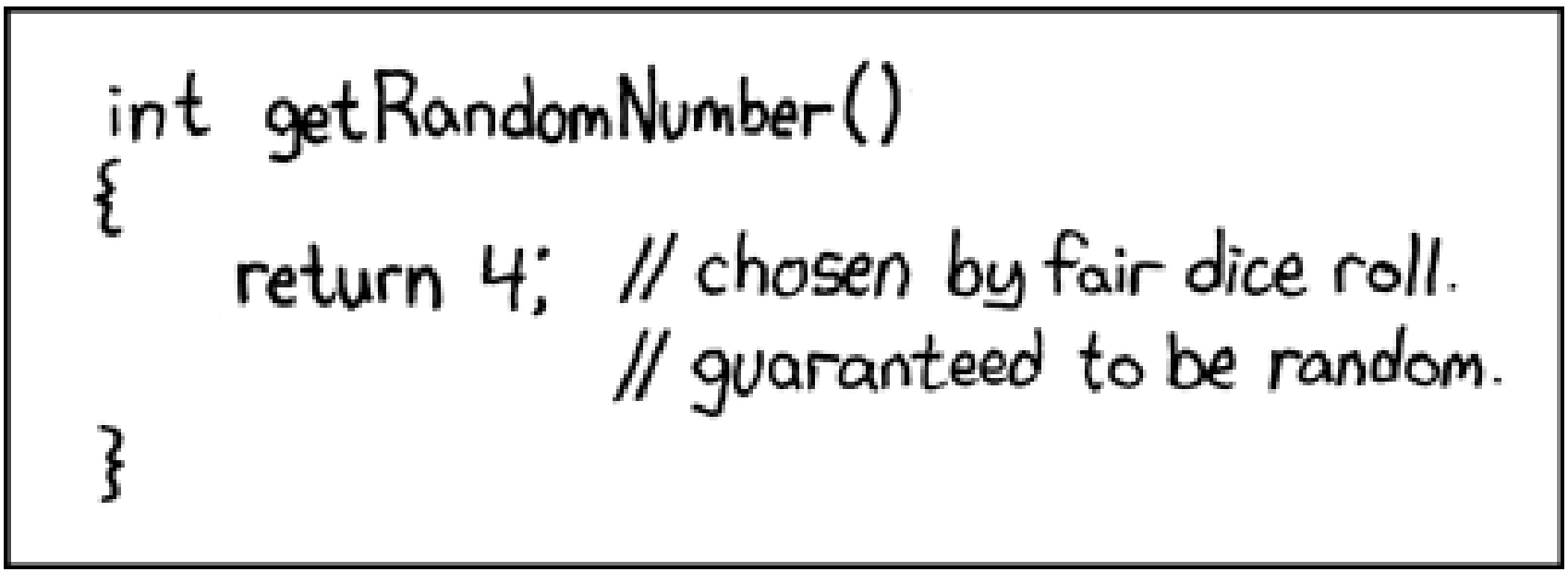
Pour One-Time-Pad, il faut fabriquer de l'aléa. Plusieurs possibilités.

- Croire que c'est facile.
- Mettre un singe devant un clavier.
- Écouter le rayonnement cosmique ou toute autre source physique.

En pratique, un ordinateur est assez déterministe.

Il existe des générateurs aléatoires ≪matériels≫, exploitant l'aléa ≪physique≫.

Comment ne pas faire ?



/dev/random

Un système linux fournit l'interface `/dev/random`, qui produit de l'aléa.

- Ressource utilisée : certains événements jugés aléatoires, associé à une information «x bits d'entropie».
  - Frappe d'une touche au clavier. Temps exact, valeur de la touche.
  - Mouvements de la souris.
  - Temps exact d'arrivée d'un paquet IP.
  - ...

  Le système a un compteur d'«entropie collectée».
- Cet input est fourni à un programme interne qui mélange.
- Résultat produit : des bits d'aléa. Chaque bit produit en sortie est soustrait du compteur d'«entropie collectée».
- Lorsque le compteur descend à zéro, plus de bit produit tant que le compteur ne remonte pas.

Retour sur `/dev/random`

- Le système de génération d'aléa collecte e bits d'entropie.
- Il est alors prêt à laisser filer e bits d'aléa produit.

Au mieux, un attaquant très fort pourra retrouver l'aléa injecté en entrée, pas davantage.

Pas de nouvelle entropie, pas d'aléa produit : on veut qu'il soit impossible d'inférer.

Quand on lit dans `/dev/random`, ça peut bloquer : quand il n'y a pas assez d'entropie collectée.

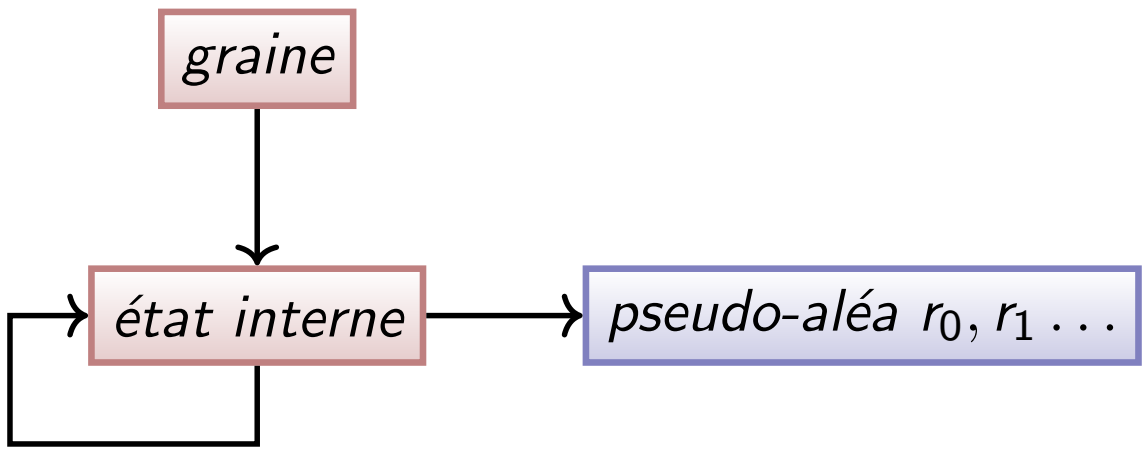
Autre version `/dev/urandom`, qui continue à produire de l'aléa même quand il n'y a plus d'entropie nouvelle. Si un attaquant réussit à inverser la fonction de mélange et deviner l'état interne, danger.

Plan

Générateurs pseudo-aléatoires

Une autre façon de fabriquer de l'aléa : les générateurs pseudo-aléatoires (PRNG).

- On part d'une graine (aléatoire).
- Cette graine alimente un état interne.
- Le générateur met à jour son état interne après chaque bit produit.



Générateurs pseudo-aléatoires

Une fois la graine spécifiée, le comportement du générateur pseudo-aléatoire est déterministe.

Usages :

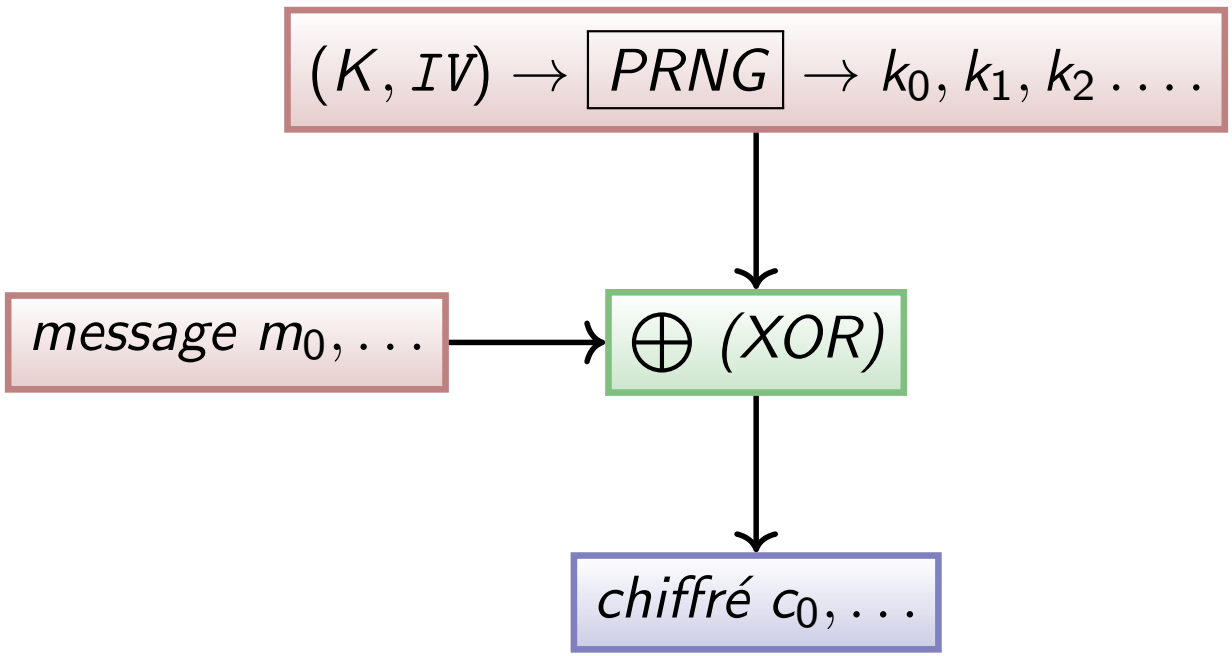
- Aléa dans les ordinateurs :
  - Pas cher à implanter.
  - Modélisation d'événements aléatoires.
  - Aussi : reproductibilité des tests.
- Crypto : utiliser une graine secrète, connue de Alice et Bob, et faire du One-Time-Pad avec.

Chiffrement à flot

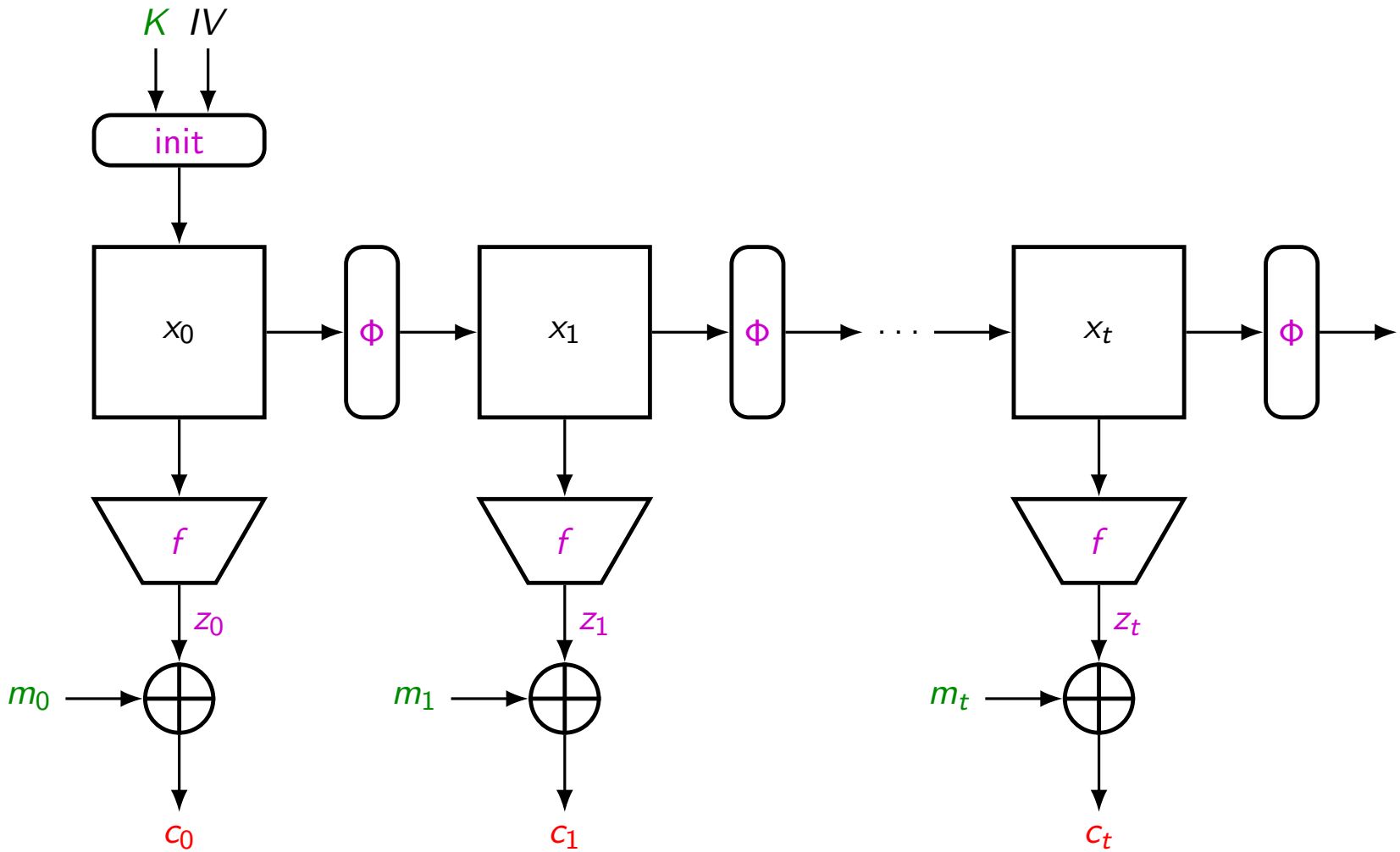
Le point-clé dans One-Time-Pad : la difficulté à générer et partager une volumineuse suite de chiffrement.

Palliatif : utiliser un (PRNG) pour fabriquer une suite chiffrante (k_i). La graine du générateur pseudo-aléatoire sert de clé.

- Chargement de l'état interne à partir de la graine.
- Un état interne $\rightsquigarrow$ bit(s) de clé, et état interne suivant.



Chiffrement par flot



Notations : x_t=état interne ; f= fonction de filtrage ; Φ= fonction de transition ; IV=valeur initiale.

Générateurs pseudo-aléatoires

On trouve plusieurs appréciations de ce qu'est un bon PRNG.

Si l'objectif est la modélisation d'événements aléatoires :

- on veut un bon comportement statistique (0 et 1 équilibrés, idem pour tous les motifs, etc, etc).

Si l'objectif est la crypto, on veut bien plus. Un adversaire doté d'une puissance de calcul polynomiale doit être incapable de :

- prédire le bit suivant avec une chance de succès $> 1/2$.
- distinguer une suite produite par le PRNG d'un vrai aléa, avec une chance de succès $> 1/2$.
- (évidemment) retrouver l'état interne et la graine du PRNG.

On rappelle le contexte des lois de Kerckhoffs : le mécanisme du PRNG est connu, seule la graine est inconnue.

PRNGs

On étudie :

- Quelques mauvais générateurs d'aléa ;
- Quelques autres moins mauvais, qui ont été proposés un jour ou l'autre comme base d'un chiffrement à flot

Un exemple de mauvais PRNG

Le rand() de la plupart des langages de programmation n'a pas de but cryptographique. Exemple proposé par POSIX.1-2001 :

```
static unsigned long x = 1;
int rand(void) {
   x = x * 1103515245 + 12345;
   return((unsigned int)(x/65536) % 32768);
}
void srand(unsigned seed) { x = seed; }
```

- C'est un générateur linéaire congruentiel (LCG).
- Chaque valeur produite donne directement 15 bits de l'état interne. À partir de quelques valeurs, on retrouve tout l'état interne.

État interne connu et feedback secret

Certains PRNG sont tellement simples qu'il n'est même pas complètement utile de connaître leurs caractéristiques pour les attaquer.

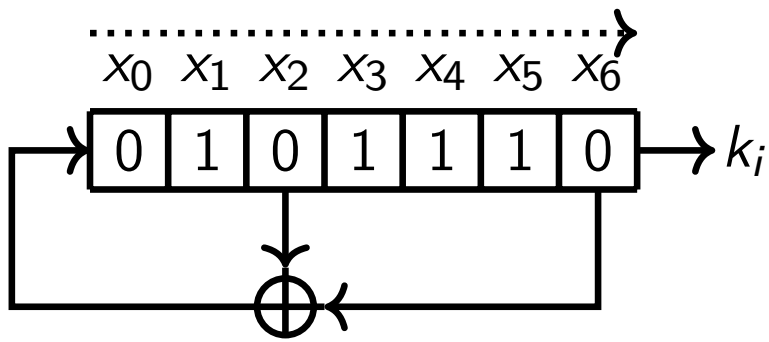
Exemple du LCG, si on connaît une suite d'états internes $x_0, x_1, \ldots$, mais pas le lien entre eux : trivial.

PRNGs : les LFSRs

LFSR=

- Linear Feedback Shift Register ;
- registre à décalage à rétroaction linéaire.

Le LFSR est un exemple basique et important de PRNG notamment pour son faible coût en matériel.



L'état interne à l'étape $i+1$ découle de l'état interne à l'étape i :

$$x_{k+1}^{(i+1)} = x_k^{(i)}, \quad x_0^{(i+1)} = x_6^{(i)} \oplus x_2^{(i)}.$$

Polynôme de rétroaction d'un LFSR

$$x_{k+1}^{(i+1)} = x_k^{(i)}, \quad x_0^{(i+1)} = x_6^{(i)} \oplus x_2^{(i)}.$$

Si on note $X = (x_0 \dots x_6)$:

$$X^{(i+1)} = X^{(i)} \times M, \quad \text{où } M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

1 ← mentalement on rajoute un 1 là

- Les coefficients sont pris dans le corps $\mathbb{Z}/2\mathbb{Z}$.
- M est une matrice compagnon. Le polynôme de rétroaction du LFSR est le polynôme caractéristique de M.
- Le polynôme de rétroaction se lit sur la première colonne.

$$z^7 + z^4 + 1.$$

Période des LFSRs

La période du LFSR de largeur n dépend principalement du polynôme de rétroaction.

- Si le polynôme est primitif c'est $2^n - 1$.
- Sinon la période s'écrit $\prod w_j$, avec $w_j \mid 2^{n_j} - 1$ et les n_j tels que $\sum n_j = n$.

Par exemple en longueur 7, on peut avoir :

- Primitif : période 127
- Sinon, par exemple : 3×7 (avec $3 \mid 2^4 - 1$ et $7 = 2^3 - 1$).

Synthèse d'un LFSR

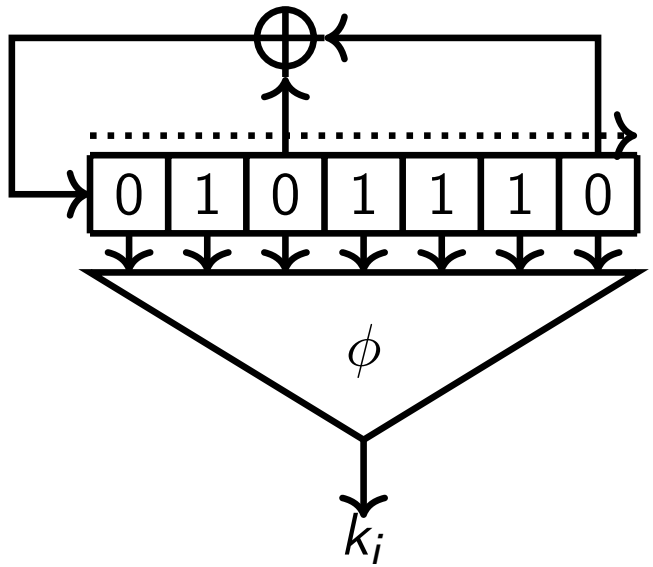
Retrouver l'état interne d'un LFSR à partir des bits en sortie est facile.

- Si le polynôme de rétroaction est connu : n bits observés permettent de retrouver l'état interne.
- Si on ne connaît pas f, il suffit de $2n$ bits pour retrouver l'état interne et f : algorithme de Berlekamp-Massey ou d'Euclide. Coût calculatoire : $O(n^2)$.

Moralité : un LFSR c'est simple, mais pas très solide tel quel.

Il faut voir les LFSR comme des briques de base.

Registres filtrés

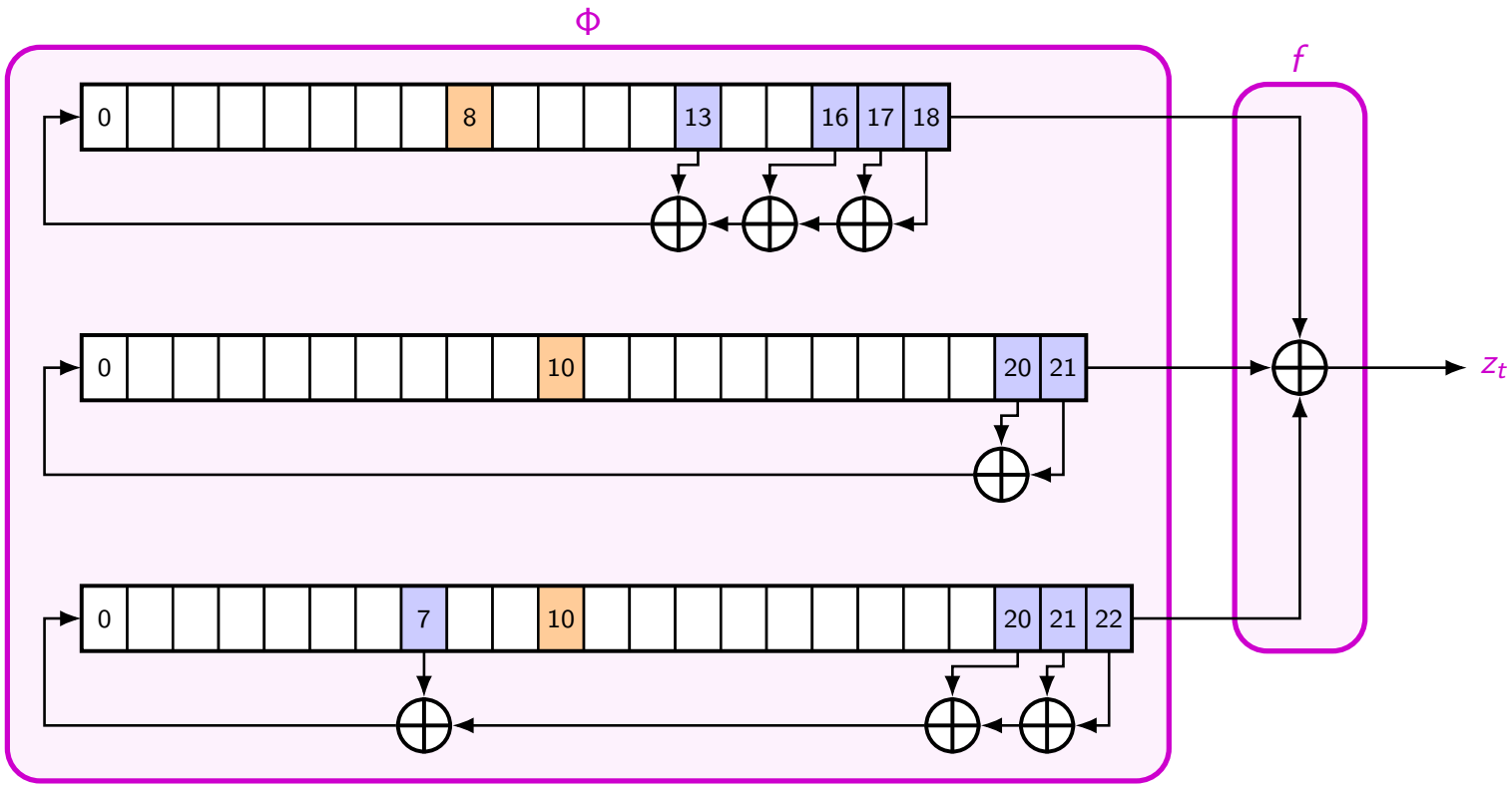


On utilise pour la fonction ϕ une fonction booléenne aux bonnes propriétés.

- Si ϕ est une fonction linéaire : peu sûr.
- On recherche les fonctions ≪les moins linéaires possibles≫.

Très utilisés, mais ϕ ne doit pas être de trop petit degré.

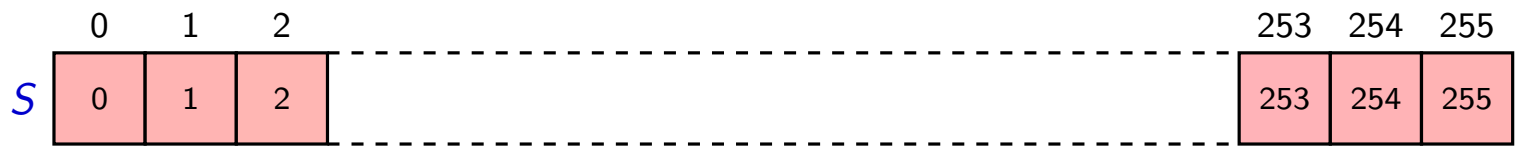
Exemple : A5/1 [1987]



- Clé secrète de 64 bits, IV de 22 bits
- Cryptanalyse en quelques minutes [Nohl & Paget, 2009]
- Snowden : *NSA «can process encrypted A5/1 in real time»*
- Toujours utilisé dans des milliards de téléphones portables (GSM)

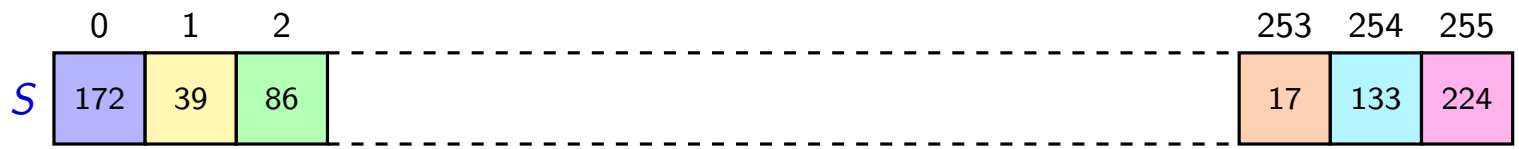
Exemple : RC4 [Rivest, 1987]

- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



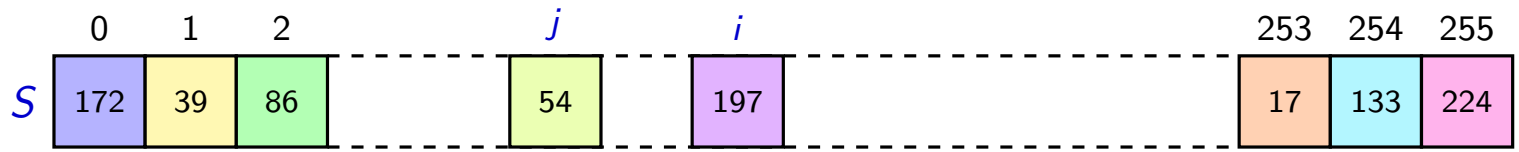
Exemple : RC4 [Rivest, 1987]

- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



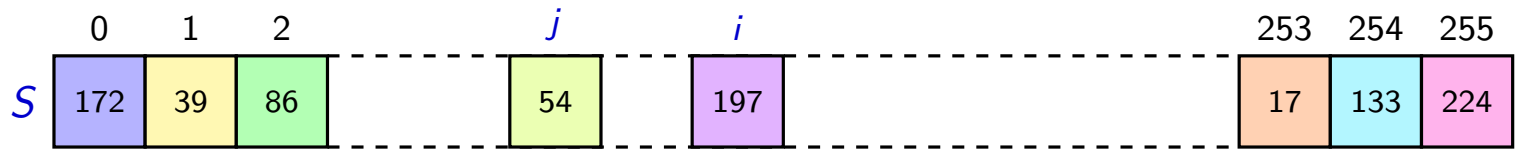
Exemple : RC4 [Rivest, 1987]

- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



Exemple : RC4 [Rivest, 1987]

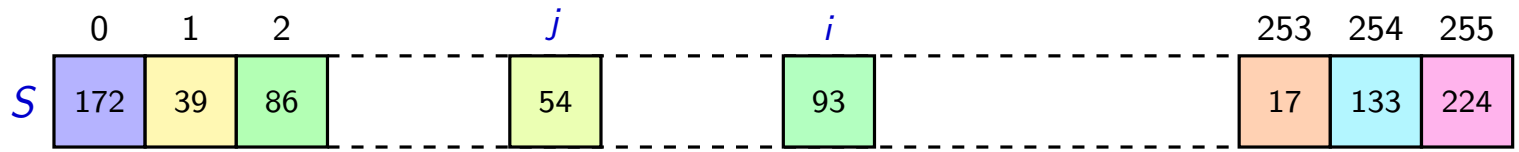
- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



- Fonction de transition Φ :
  - $i \leftarrow (i + 1) \bmod 256$
  - $j \leftarrow (j + S[i]) \bmod 256$
  - échanger les valeurs de $S[i]$ et $S[j]$

Exemple : RC4 [Rivest, 1987]

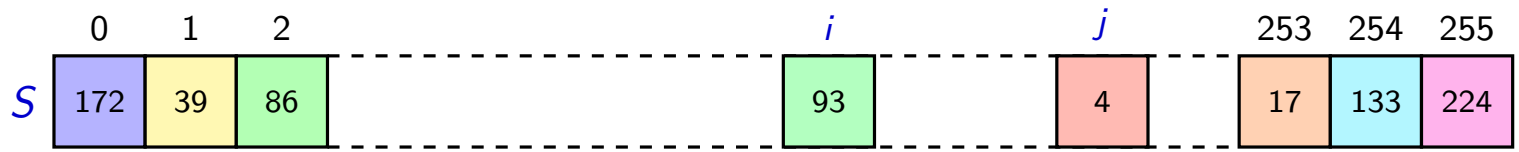
- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



- Fonction de transition Φ :
  - $i \leftarrow (i + 1) \bmod 256$
  - $j \leftarrow (j + S[i]) \bmod 256$
  - échanger les valeurs de $S[i]$ et $S[j]$

Exemple : RC4 [Rivest, 1987]

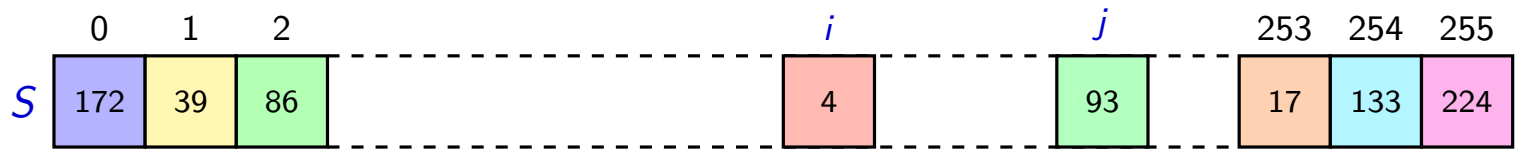
- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



- Fonction de transition Φ :
  - $i \leftarrow (i + 1) \bmod 256$
  - $j \leftarrow (j + S[i]) \bmod 256$
  - échanger les valeurs de $S[i]$ et $S[j]$

Exemple : RC4 [Rivest, 1987]

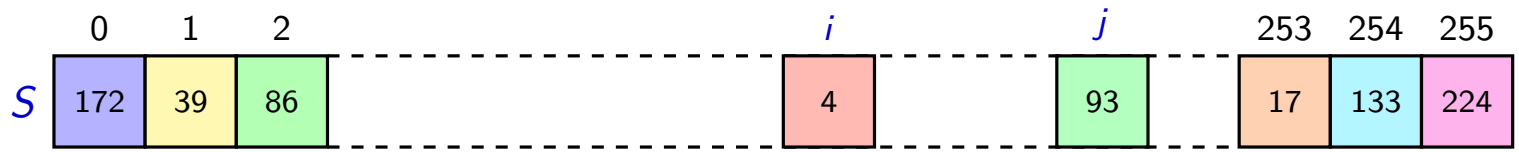
- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



- Fonction de transition Φ :
  - $i \leftarrow (i + 1) \bmod 256$
  - $j \leftarrow (j + S[i]) \bmod 256$
  - échanger les valeurs de $S[i]$ et $S[j]$

Exemple : RC4 [Rivest, 1987]

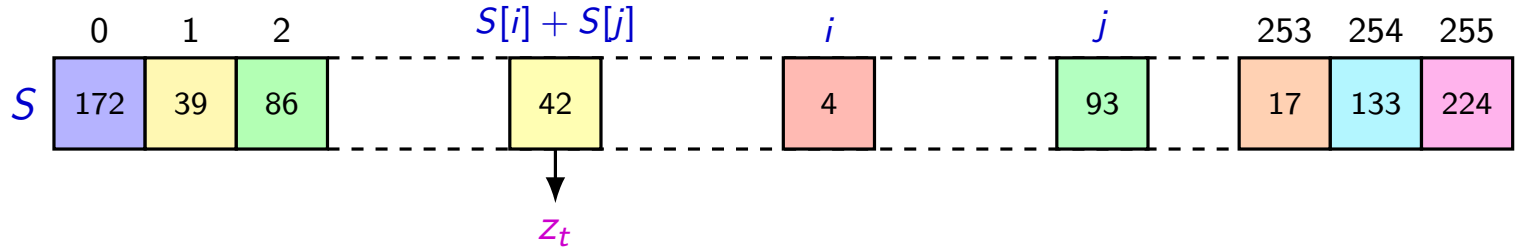
- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



- Fonction de transition Φ :
  - $i \leftarrow (i + 1) \bmod 256$
  - $j \leftarrow (j + S[i]) \bmod 256$
  - échanger les valeurs de $S[i]$ et $S[j]$
- Fonction de filtrage f : $z_t \leftarrow S[(S[i] + S[j]) \bmod 256]$

Exemple : RC4 [Rivest, 1987]

- Utilisé dans SSL/TLS, SSH, WEP, WPA, etc.
- Très efficace en logiciel, mais nombreux biais statistiques sur Z
- Clé secrète de 40 à 2048 bits, pas d'IV en tant que tel
- État interne :
  - tableau S de 256 octets : permutation de $\{0, 1, \ldots, 255\}$
  - indices i et $j \in \{0, 1, \ldots, 255\}$



- Fonction de transition Φ :
  - $i \leftarrow (i + 1) \bmod 256$
  - $j \leftarrow (j + S[i]) \bmod 256$
  - échanger les valeurs de $S[i]$ et $S[j]$
- Fonction de filtrage f : $z_t \leftarrow S[(S[i] + S[j]) \bmod 256]$