Introduction à la sécurité
Cours 2 – Cryptologie symétrique

Julien Lavauzelle

Université Paris 8

Licence 3 Informatique et Vidéoludisme

12/09/2023

Ces slides sont en partie inspirées de celles de Pablo Rauzy, concepteur initial du cours.

Voir sa page web : `pablo.rauzy.name/teaching/is`

Plan

Définitions

Dans un **cryptosystème symétrique**, les intervenants détiennent un secret commun : la **clé secrète**.
On parle donc également de **cryptosystème à clé secrète**.

Définitions

Dans un **cryptosystème symétrique**, les intervenants détiennent un secret commun : la **clé secrète**.
On parle donc également de **cryptosystème à clé secrète**.

La **cryptographie symétrique** intervient

- principalement pour du **chiffrement** (ici, on ne verra que ça)
- également dans d'**autres primitives** à clé secrète (ex : HMAC)

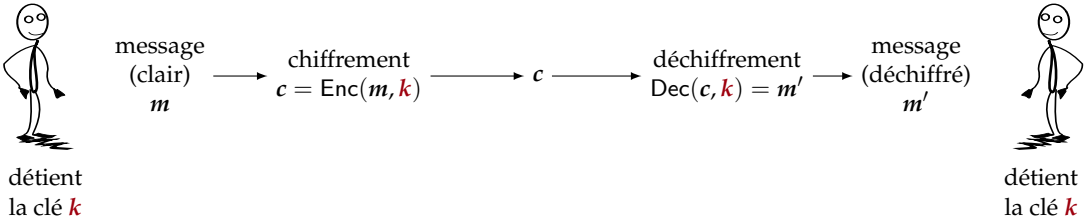
Définitions

Dans un **cryptosystème symétrique**, les intervenants détiennent un secret commun : la **clé secrète**.
On parle donc également de **cryptosystème à clé secrète**.

La **cryptographie symétrique** intervient

- principalement pour du **chiffrement** (ici, on ne verra que ça)
- également dans d'**autres primitives** à clé secrète (ex : HMAC)

Pour du chiffrement, le principe est le suivant :



Définitions

Dans un **cryptosystème symétrique**, les intervenants détiennent un secret commun : la **clé secrète**.
On parle donc également de **cryptosystème à clé secrète**.

La **cryptographie symétrique** intervient

- principalement pour du **chiffrement** (ici, on ne verra que ça)
- également dans d'**autres primitives** à clé secrète (ex : HMAC)

Pour du chiffrement, le principe est le suivant :

détient
la clé k

message (clair) m → chiffrement $c = \mathsf{Enc}(m, k)$ → c → déchiffrement $\mathsf{Dec}(c, k) = m'$ → message (déchiffré) m'

détient
la clé k

Remarque : les chiffrements par substitution du cours précédent sont des chiffrements à clé secrète.

Dorénavant, on veut **chiffrer n'importe quoi** (pas seulement des lettres de l'alphabet). Les entrées et sorties de nos algorithmes de chiffrement seront donc des chaînes de bits :

$$\boldsymbol{m} = 010001001010001...$$

Définitions

Dorénavant, on veut **chiffrer n'importe quoi** (pas seulement des lettres de l'alphabet). Les entrées et sorties de nos algorithmes de chiffrement seront donc des chaînes de bits :

$$m = 010001001010001...$$

On distingue alors **deux catégories de chiffrement symétrique** :

1. le chiffrement par **flot**,
2. le chiffrement par **bloc**.

Définitions

Dorénavant, on veut **chiffrer n'importe quoi** (pas seulement des lettres de l'alphabet). Les entrées et sorties de nos algorithmes de chiffrement seront donc des chaînes de bits :

$$m = 010001001010001...$$

On distingue alors **deux catégories de chiffrement symétrique** :

1. le chiffrement par **flot**,
2. le chiffrement par **bloc**.

→ Le chiffrement par **flot** peut traiter des données de **taille quelconque**. Ces données sont traitées « à la volée » et successivement.
Exemples : A5/1 (GSM), E0 (Bluetooth), RC4 (WEP), plus récemment ChaCha (IPsec)

Définitions

Dorénavant, on veut **chiffrer n'importe quoi** (pas seulement des lettres de l'alphabet). Les entrées et sorties de nos algorithmes de chiffrement seront donc des chaînes de bits :

$$m = 010001001010001...$$

On distingue alors **deux catégories de chiffrement symétrique** :

1. le chiffrement par **flot**,
2. le chiffrement par **bloc**.

→ Le chiffrement par **flot** peut traiter des données de **taille quelconque**. Ces données sont traitées « à la volée » et successivement.
Exemples : A5/1 (GSM), E0 (Bluetooth), RC4 (WEP), plus récemment ChaCha (IPsec)

→ Le chiffrement par **bloc** traiter des données découpées en blocs de **taille fixe**. Chaque bloc peut (pas toujours) être traité indépendamment.
Exemples : DES (ancien standard), AES (nouveau standard), Simon (chiffrement "léger")

Chiffrement de Vernam

Un type de chiffrement pourrait être classé dans les deux catégories : le **chiffrement de Vernam**, aussi appelé **chiffrement par masque jetable**.

Chiffrement de Vernam

Un type de chiffrement pourrait être classé dans les deux catégories : le **chiffrement de Vernam**, aussi appelé **chiffrement par masque jetable**.

Mise en place. Prenons un message de taille N bits

$$M = m_1 m_2 \dots m_N \, .$$

Chiffrement de Vernam

Un type de chiffrement pourrait être classé dans les deux catégories : le **chiffrement de Vernam**, aussi appelé **chiffrement par masque jetable**.

Mise en place. Prenons un message de taille N bits

$$M = m_1 m_2 \dots m_N \,.$$

Génération de clé. Pour le chiffrer avec le chiffrement de Vernam, il faut engendrer une **clé aléatoire**

$$K = k_1 k_2 \dots k_N$$

qui doit être également de taille N bits.

Chiffrement de Vernam

Un type de chiffrement pourrait être classé dans les deux catégories : le **chiffrement de Vernam**, aussi appelé **chiffrement par masque jetable**.

Mise en place. Prenons un message de taille N bits

$$M = m_1 m_2 \dots m_N \,.$$

Génération de clé. Pour le chiffrer avec le chiffrement de Vernam, il faut engendrer une **clé aléatoire**

$$K = k_1 k_2 \dots k_N$$

qui doit être également de taille N bits.

Chiffrement. Puis, le chiffré du message M par la clé K est $C = c_1 c_2 \dots c_N$, le XOR bit-à-bit du message et de la clé, qui est lui aussi de taille N bits :

$$c_i = m_i \oplus k_i, \qquad \forall i = 1 \dots N$$

Chiffrement de Vernam

Un type de chiffrement pourrait être classé dans les deux catégories : le **chiffrement de Vernam**, aussi appelé **chiffrement par masque jetable**.

Mise en place. Prenons un message de taille N bits

$$M = m_1 m_2 \dots m_N \,.$$

Génération de clé. Pour le chiffrer avec le chiffrement de Vernam, il faut engendrer une **clé aléatoire**

$$K = k_1 k_2 \dots k_N$$

qui doit être également de taille N bits.

Chiffrement. Puis, le chiffré du message M par la clé K est $C = c_1 c_2 \dots c_N$, le XOR bit-à-bit du message et de la clé, qui est lui aussi de taille N bits :

$$c_i = m_i \oplus k_i, \qquad \forall i = 1 \dots N$$

Déchiffrement. Pour déchiffrer C, on réalise la même opération :

$$c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i = m_i, \qquad \forall i = 1 \dots N$$

Chiffrement de Vernam : avantages et inconvénients

Avantage. C'est le **seul** chiffrement qui est **inconditionnellement sûr**, dans le sens où un attaquant de force quelconque ne pourra obtenir aucune information sur le message clair à partir du chiffré seul.

Inconvénients. La clé de chiffrement :

- ne peut être utilisée qu'une seule fois (d'où le nom de **masque jetable**), car l'opération

$$\begin{cases} C^1 = M^1 \oplus K \\ C^2 = M^2 \oplus K \end{cases} \quad \Longrightarrow \quad C^1 \oplus C^2 = M^1 \oplus M^2$$

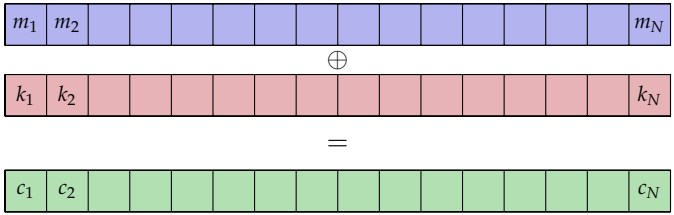
  révèle des relations entre les bits des chiffrés et ceux des messages (indésirable) ;
- est **lourde** à stocker (aussi grosse que le fichier à chiffrer) ;
- demande d'**engendrer beaucoup d'aléa**.

Conclusion. Le chiffrement de Vernam est **impossible à mettre en pratique**. En revanche, on est prêt à **perdre un petit peu de sécurité pour avoir des chiffrements plus efficaces**.

Plan

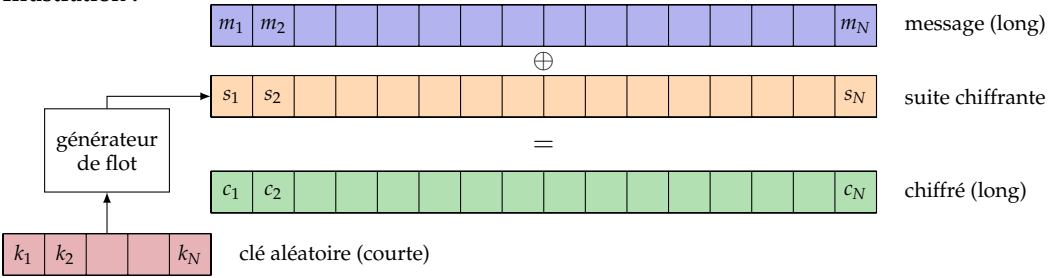
Idée : on remplace le masque jetable du chiffrement de vernam, par une **suite chiffrante pseudo-aléatoire** engendrée à partir d'une clé secrète de petite taille.

Illustration :



Idée : on remplace le masque jetable du chiffrement de vernam, par une **suite chiffrante pseudo-aléatoire** engendrée à partir d'une clé secrète de petite taille.

Illustration :



Chiffrement par flot : définitions

Un **chiffrement par flot** (synchrone et additif) est donc la donnée d'un générateur d'une suite chiffrante que l'on XOR bit-à-bit au texte clair. On parle de **générateur pseudo-aléatoire** (PRG, *pseudo-random generator*).

Chiffrement par flot : définitions

Un **chiffrement par flot** (synchrone et additif) est donc la donnée d'un générateur d'une suite chiffrante que l'on XOR bit-à-bit au texte clair. On parle de **générateur pseudo-aléatoire** (PRG, *pseudo-random generator*).

La suite chiffrante est **paramétrée** par la clé de chiffrement.

Chiffrement par flot : définitions

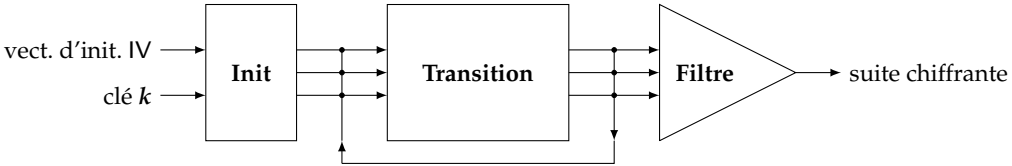
Un **chiffrement par flot** (synchrone et additif) est donc la donnée d'un générateur d'une suite chiffrante que l'on XOR bit-à-bit au texte clair. On parle de **générateur pseudo-aléatoire** (PRG, *pseudo-random generator*).

La suite chiffrante est **paramétrée** par la clé de chiffrement.

Remarque. Dans notre contexte (synchrone), la suite chiffrante est indépendante du texte clair.

Un **générateur de suite chiffrante** est modélisé par 3 fonctions :

1. Fonction d'**initialisation**, dans laquelle la **clé secrète** k est notamment intégrée.
2. Fonction de **transition d'état** (qui ne dépend pas de la clé).
3. Fonction de **filtre** (qui ne dépend pas de la clé).



Générateurs congruentiels

Un **générateur congruentiel linéaire** produit une séquence d'entiers (que l'on peut ensuite convertir en bits).

Un **générateur congruentiel linéaire** produit une séquence d'entiers (que l'on peut ensuite convertir en bits).

Il dépend de trois paramètres entiers : le module m et deux coefficients a, b inférieurs à m.

- **Initialisation :** on calcule une **graine** x_0
- **Transition :** l'état x_{n+1} à l'instant $n+1$ est calculé selon l'état x_n à l'instant n selon l'équation :

$$x_{n+1} = a \cdot x_n + b \mod m$$

- **Filtre :** aucun.

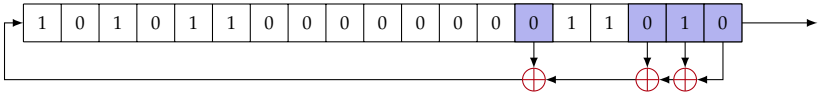
La suite d'entiers $(x_n)_{n\in\mathbb{N}}$ obtenue est **périodique** ; pour des raisons de sécurité, on cherche à prendre a, b et m de sorte que la période soit la plus grande possible (elle sera toujours plus petite que m...).

Remarque. Il existe d'autres générateurs non-linéaires. Par exemple le **générateur Blum–Blum–Shub**, très utilisé, a pour équation de transition $x_{n+1} = x_n^2 \mod m$ (avec un bon choix de m) et comme filtre le bit de poids faible.

Registre à décalage à rétroaction linéaire (LFSR)

Les **LFSR** (*linear feedback shift register*, **registre à décalage à rétroaction linéaire**) forment une autre famille de générateurs.

1. On initialise un registre de bits avec des valeurs aléatoires.
2. À chaque pas de temps :
   - le dernier bit est donné en sortie
   - les autres bits sont décalés d'une case (*shift*)
   - la case vide est remplie par une combinaison linéaire (*linear feedback*) de certains bits du registre



La chaîne de bits produite en sortie est **périodique**. La fonction de rétroaction influe sur la taille de la période (jusque $2^N - 1$, où N est la taille du registre), donc sur la sécurité.

Remarque. On peut combiner les sorties de plusieurs LFSR pour augmenter la période en parallélisant les calculs.

Un exemple : A5/1

A5/1 est un **chiffrement par flot** utilisé pour les communications GSM (standard 2G pour les mobiles).

- utilisé en Europe et aux États-Unis
- développé en 1987, gardé secret (!), puis retrouvé par rétro-ingénierie en 1999
- dont la sécurité est plus que discutable :
  - une version affaiblie A5/2 était destinée à l'export
  - on a su ultérieurement (Snowden) que la NSA savait même casser A5/1

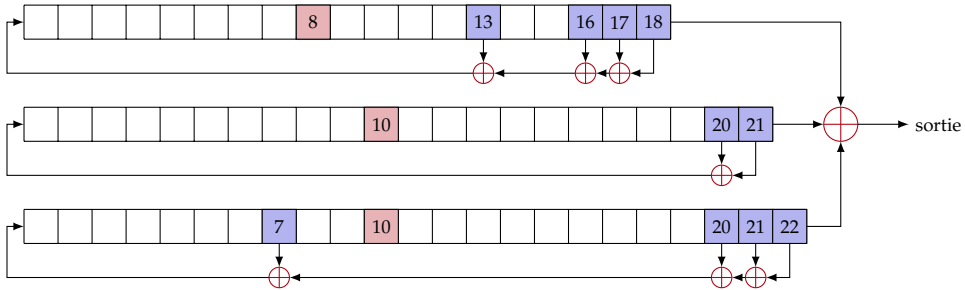
Un exemple : A5/1

Initialisation :

- une clé de 64 bits + un vecteur d'initialisation de 22 bits
- on les insère successivement sur chaque première case du registre (puis on décale), en partant des registres tout-à-zéro.

Fonction de transition :

- trois LFSR de différentes longueurs
- à chaque pas de temps, un registre est décalé si son bit « rouge » est égal à la majorité des 2 bits « rouge »



En pratique, quelques grandeurs :

- Les chiffrements par flot sont **extrêmement rapides** : on peut chiffrer 1 octet en $\simeq 5$ cycles CPU.
- La taille de clé assez variable selon les systèmes, mais des clés de taille ≥ 80 bits sont **hautement recommandées**, voire ≥ 128 bits pour du long terme.
- La **taille mémoire de l'état interne** est de plusieurs centaines de bits (rarement plus) ; ils sont donc relativement efficaces en mémoire ($\rightarrow$ utile pour du chiffrement IOT).

Quelques inconvénients en terme de sécurité, notamment dans des contextes particuliers :

- une clé doit être utilisée une seule fois : sinon, on a :

$$\mathrm{Enc}_K(M_1) \oplus \mathrm{Enc}_K(M_2) = (M_1 \oplus F_K) \oplus (M_2 \oplus F_K) = M_1 \oplus M_2$$

  où F_K est le flux de clé associé à la clé K. La connaissance des chiffrés $\mathrm{Enc}_K(M_1)$ et $\mathrm{Enc}_K(M_2)$ procure donc de l'information sur M_1 et M_2.
- le chiffrement est **malléable** : un attaquant qui intercepte un message peut flipper un bit du chiffré ; cela flippera un bit du clair pour celui qui déchiffrera le message.
- ...

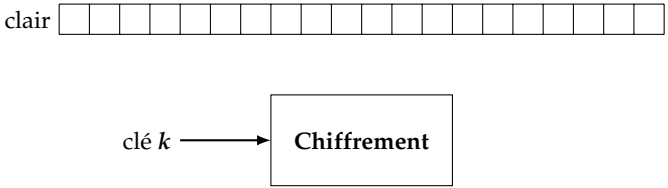
Plan

Plan

Chiffrement par bloc : intuition

Idée : on découpe le texte clair en **blocs de taille fixe**. Puis, chaque bloc est chiffré avec incorporation des certains bits de la clé dans un algorithme de chiffrement (plus complexe que le XOR).

Illustration :

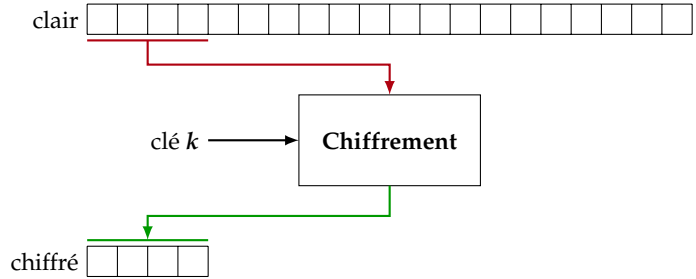


Remarque : il y a beaucoup de variété dans les manières d'incorporer les bits de clés, de les « mélanger » aux bits du texte clair, mais aussi de traiter successivement les blocs.

Chiffrement par bloc : intuition

Idée : on découpe le texte clair en **blocs de taille fixe**. Puis, chaque bloc est chiffré avec incorporation des certains bits de la clé dans un algorithme de chiffrement (plus complexe que le XOR).

Illustration :

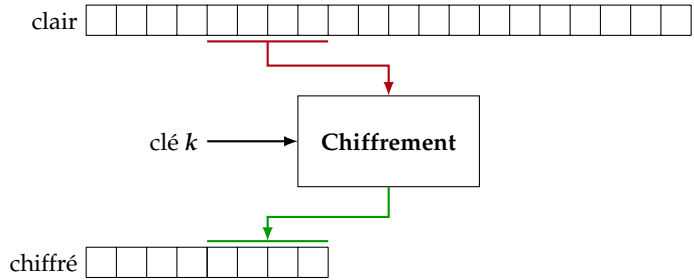


Remarque : il y a beaucoup de variété dans les manières d'incorporer les bits de clés, de les « mélanger » aux bits du texte clair, mais aussi de traiter successivement les blocs.

Chiffrement par bloc : intuition

Idée : on découpe le texte clair en **blocs de taille fixe**. Puis, chaque bloc est chiffré avec incorporation des certains bits de la clé dans un algorithme de chiffrement (plus complexe que le XOR).

Illustration :

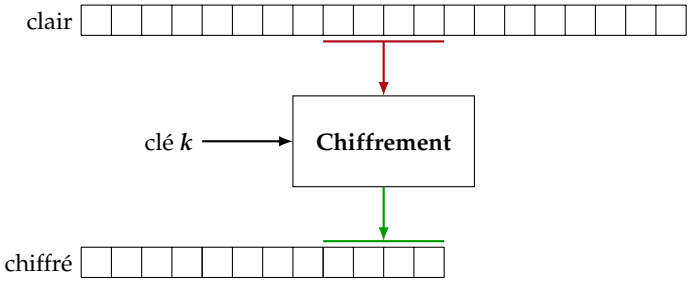


Remarque : il y a beaucoup de variété dans les manières d'incorporer les bits de clés, de les « mélanger » aux bits du texte clair, mais aussi de traiter successivement les blocs.

Chiffrement par bloc : intuition

Idée : on découpe le texte clair en **blocs de taille fixe**. Puis, chaque bloc est chiffré avec incorporation des certains bits de la clé dans un algorithme de chiffrement (plus complexe que le XOR).

Illustration :

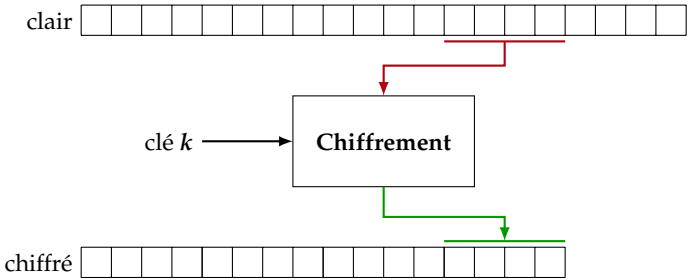


Remarque : il y a beaucoup de variété dans les manières d'incorporer les bits de clés, de les « mélanger » aux bits du texte clair, mais aussi de traiter successivement les blocs.

Chiffrement par bloc : intuition

Idée : on découpe le texte clair en **blocs de taille fixe**. Puis, chaque bloc est chiffré avec incorporation des certains bits de la clé dans un algorithme de chiffrement (plus complexe que le XOR).

Illustration :

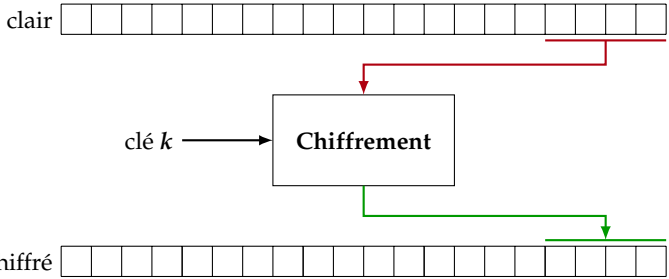


Remarque : il y a beaucoup de variété dans les manières d'incorporer les bits de clés, de les « mélanger » aux bits du texte clair, mais aussi de traiter successivement les blocs.

Chiffrement par bloc : intuition

Idée : on découpe le texte clair en **blocs de taille fixe**. Puis, chaque bloc est chiffré avec incorporation des certains bits de la clé dans un algorithme de chiffrement (plus complexe que le XOR).

Illustration :



Remarque : il y a beaucoup de variété dans les manières d'incorporer les bits de clés, de les « mélanger » aux bits du texte clair, mais aussi de traiter successivement les blocs.

Précisions sur la fonction de chiffrement

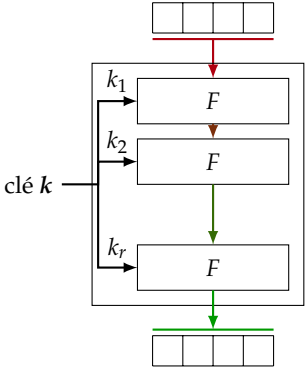
La procédure de chiffrement fonctionne très souvent de manière itérative : on applique

- un certain de nombre de fois (les **tours**),
- une **même** transformation F aux données (appelée **fonction de tour**),
- en incorporant différents morceaux de la clé (les **sous-clés**).

Précisions sur la fonction de chiffrement

La procédure de chiffrement fonctionne très souvent de manière itérative : on applique

- un certain de nombre de fois (les **tours**),
- une **même** transformation F aux données (appelée **fonction de tour**),
- en incorporant différents morceaux de la clé (les **sous-clés**).

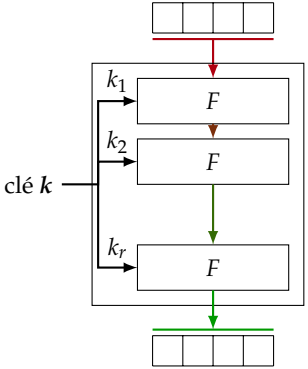


Précisions sur la fonction de chiffrement

La procédure de chiffrement fonctionne très souvent de manière itérative : on applique

- un certain de nombre de fois (les **tours**),
- une **même** transformation F aux données (appelée **fonction de tour**),
- en incorporant différents morceaux de la clé (les **sous-clés**).

Le chiffré du bloc $\boldsymbol{m}$ est alors la dernière sortie $\boldsymbol{c}_r$ de la fonction F.



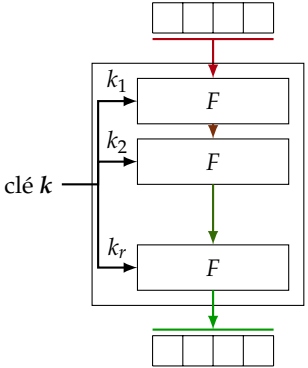
Précisions sur la fonction de chiffrement

La procédure de chiffrement fonctionne très souvent de manière itérative : on applique

- un certain de nombre de fois (les **tours**),
- une **même** transformation F aux données (appelée **fonction de tour**),
- en incorporant différents morceaux de la clé (les **sous-clés**).

Le chiffré du bloc $\boldsymbol{m}$ est alors la dernière sortie $\boldsymbol{c}_r$ de la fonction F.

Les sous-clés $k_1, k_2, \ldots, k_r$ sont déduites de la clé $\boldsymbol{k}$ par un **algorithme de préparation de clés**.



La fonction de tour

Quelques **propriétés désirées** de la fonction de tour F :

1. Il est nécessaire de pouvoir inverser le procédé de chiffrement (passer du chiffré au clair). Donc, la fonction F **doit** être **injective**.
2. Il est préférable (pour des raisons de sécurité) qu'un maximum de valeurs de sorties soient effectivement prises par la fonction F. On souhaite donc qie F soit **surjective**.
3. Les entrées et sorties sont préférablement des chaînes de bits de même taille.

Une fonction **bijective** (injective et surjective) d'un ensemble dans lui-même est appelée une **permutation**.

Ensuite, nous allons voir deux manières de créer une fonction de tour à itérer :

- les **réseaux de substitution/permutation** (SPN en anglais),
- les **réseaux de Feistel**.

Propagation de l'aléa secret

La clé secrète correspond à une suite de bits aléatoire qui doit permettre de cacher le texte clair. Il faut donc que cet **aléa se propage** efficacement tout au long du processus de chiffrement.

Préparation des clés. On introduit des morceaux de clés au fur et à mesure des tours. Plusieurs manières de créer ces **sous-clés** :

- par partition, c'est-à-dire qu'on coupe la clé en morceaux (par ex. dans TEA) ;
- en effectuant des rotations de la clé, puis en sélectionnant certaines positions (par ex. dans DES) ;
- avec des tranformations plus complexes (PRESENT, AES).

Il est important que l'utilisation des bits de la clé soit équilibrée.

Lors de chaque tour, on veut l'intégration des bits satisfassent deux propriétés.

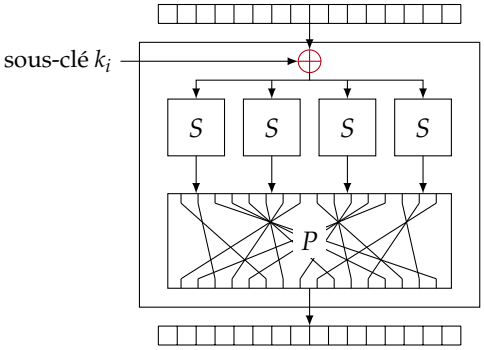
1. **Confusion :** les relations entre les bits de clé et du clair doivent être complexe (en terme de non-linéarité et de nombre de bits touchés)
2. **Diffusion :** l'aléa introduit par chaque bit de clé introduit doit se propager sur un maximum de bits en sortie.

Idéalement, **modifier un bit de clé doit affecter chacun des bits de sortie avec probabilité 1/2**.

Plan

Les **réseaux de substitution et de permutation** (SPN) forment une famille de chiffrements par bloc, dont les tours sont définis par trois opérations successives.

1. L'**intégration de la sous-clé**, souvent par un XOR bit-à-bit en entrée.
2. L'action une **boîte S** (S pour substitution), qui prend en entrée un petit nombre de bits m (souvent bien plus petit que la taille du bloc), et qui représente une opération fortement non-linéaire. **La boîte S contribue à la confusion.** Elle peut être implantée par une table de correspondance.
3. La permutation de l'ensemble des bits du bloc, par une **boîte P**. **Cela contribue à la diffusion.**



Déchiffrement et performance

Pour le **déchiffrement** d'un SPN, il faut implémenter les fonctions inverses P^{-1} et S^{-1} de la permutation P et de la substiution S, puis « remonter » le réseau.

Déchiffrement et performance

Pour le **déchiffrement** d'un SPN, il faut implémenter les fonctions inverses P^{-1} et S^{-1} de la permutation P et de la substiution S, puis « remonter » le réseau.

Remarque : en chiffrement *lightweight*, on utilise parfois des boîtes S **involutives**, c'est-à-dire telles que $S^{-1} = S$, afin d'économiser de l'espace de circuit.

Déchiffrement et performance

Pour le **déchiffrement** d'un SPN, il faut implémenter les fonctions inverses P^{-1} et S^{-1} de la permutation P et de la substiution S, puis « remonter » le réseau.

Remarque : en chiffrement *lightweight*, on utilise parfois des boîtes S **involutives**, c'est-à-dire telles que $S^{-1} = S$, afin d'économiser de l'espace de circuit.

Performances :

- La boîte S est la plus lourde à évaluer (non-linéaire $\implies$ beaucoup de portes NAND) ; c'est pour cela qu'elle est choisie de petite taille (, quitte à en monter plusieurs en parallèle.
- La boîte P est très efficace à implanter, surtout en hardware :« permutation des cases mémoire ».
- Le nombre de tours est variable selon le système et la taille de clé, mais souvent compris entre 10 et 30.
- La clé est de taille ≥ 80 bits, ≥ 128 bits recommandés à long terme.
- Temps d'exécution : quelques cycles pour un octet.

Déchiffrement et performance

Pour le **déchiffrement** d'un SPN, il faut implémenter les fonctions inverses P^{-1} et S^{-1} de la permutation P et de la substiution S, puis « remonter » le réseau.

Remarque : en chiffrement *lightweight*, on utilise parfois des boîtes S **involutives**, c'est-à-dire telles que $S^{-1} = S$, afin d'économiser de l'espace de circuit.

Performances :

- La boîte S est la plus lourde à évaluer (non-linéaire $\implies$ beaucoup de portes NAND) ; c'est pour cela qu'elle est choisie de petite taille (, quitte à en monter plusieurs en parallèle.
- La boîte P est très efficace à implanter, surtout en hardware :« permutation des cases mémoire ».
- Le nombre de tours est variable selon le système et la taille de clé, mais souvent compris entre 10 et 30.
- La clé est de taille ≥ 80 bits, ≥ 128 bits recommandés à long terme.
- Temps d'exécution : quelques cycles pour un octet.

Standard AES : bloc de taille 128 bits, boîte S de taille 8 bits, clés possibles 128–192–256 bits, tours possibles 10–12–14. Si optimisé sur machine standard, moins d'une seconde pour chiffrer un Go.

PRESENT : introduction et boîte S

PRESENT est un système de **chiffrement symétrique**, sur un modèle **itératif** de type **SPN**.

- **Auteurs :** A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, et C. Vikkelsoe. Présenté à la conférence internationale CHES en **2007**.
- Conçu spécifiquement pour des **systèmes embarqués** ("léger", rapide, optimisé hardware).

PRESENT est un système de **chiffrement symétrique**, sur un modèle **itératif** de type **SPN**.

- **Auteurs :** A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, et C. Vikkelsoe. Présenté à la conférence internationale CHES en **2007**.
- Conçu spécifiquement pour des **systèmes embarqués** ("léger", rapide, optimisé hardware).
- **Spécifications** :
  - Taille de bloc : 64 bits
  - Taille de clé : 80 bits (variante à 128 bits)
  - 31 tours (+ un ajout de sous-clé)
  - Boîte S de taille 4x4, choisie pour son efficacité hardware (14 portes logiques).
    En notation hexadécimale pour les chaînes de 4 bits :

n	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(n)$	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

PRESENT : boîte P

La **fonction de permutation** de PRESENT permute les bits du blocs de manière "presque" linéaire. Le i-ème bit est déplacé en position $P(i)$, où :

$$P(i) := \begin{cases} 16 \times i \mod 63 & \text{si } i \leq 62 \\ 63 & \text{si } i = 63 \end{cases}$$

La **fonction de permutation** de PRESENT permute les bits du blocs de manière "presque" linéaire. Le i-ème bit est déplacé en position $P(i)$, où :

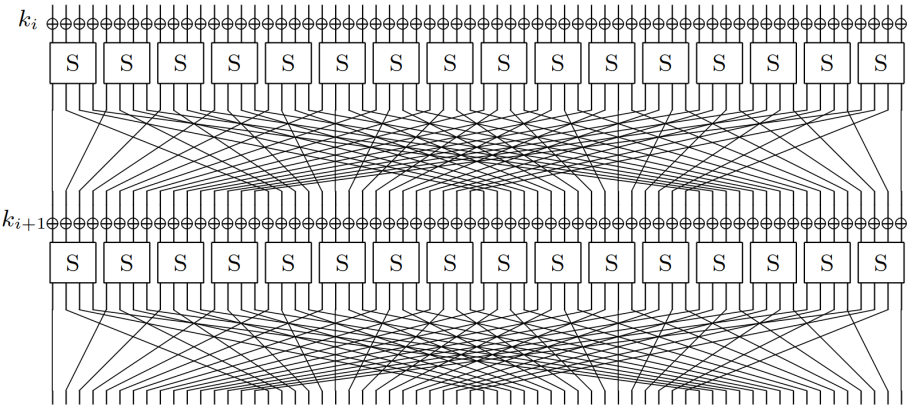
$$P(i) := \begin{cases} 16 \times i \mod 63 & \text{si } i \leq 62 \\ 63 & \text{si } i = 63 \end{cases}$$

Autre manière de voir les choses : dans le tableau ci-dessous, au lieu de lire les indices ligne par ligne, on les lit colonne par colonne

$P(0) = 0$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(1) = 16$	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(2) = 32$	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(3) = 48$	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

On observe que les bits 0, 1, 2, et 3 en sortie de la première boite S, se propagent dans les entrées de 4 boîtes S différentes. C'est la **diffusion**.

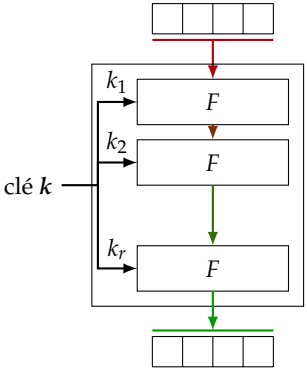
Exemple : PRESENT



PRESENT : An Ultra-Lightweight Block Cipher. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe. CHES. **2007**.

Préparation des sous-clés de PRESENT pour une clé k initiale de 80 bits :

- Les sous-clés préparées k_i auront taille 64 bits.
- On initialise $k'_0 \leftarrow k$.
- À chaque tour i, pour i entre 1 et 32, on définit la sous-clé k_i de la manière suivante :
  1. $t_i \leftarrow k'_{i-1}[19:80] \,\|\, k'_{i-1}[0:19]$ (rotation des 19 premiers bits vers la fin)
  2. $k'_i \leftarrow t_i[0:15] \,\|\, (t_i[15:20] \oplus i) \,\|\, t_i[20:75] \,\|\, S(t_i[76:80])$ où S désigne la boîte S vue précédemment.
     Remarque : cette deuxième étape ne permute que les 8 bits d'indices 16 à 19 et 76 à 80.
  3. $k_i \leftarrow k'_i[0:64]$



Remarques sur la sécurité :

- une version affaiblie de PRESENT (notamment, moins de 31 tours) a été attaquée en 2014
- la version complète de PRESENT a également été attaquée durant les années suivantes, avec une quantité de données massives
  $\rightarrow$ PRESENT **doit** donc être uniquement utilisé dans des contextes empbarqués, où peu de données transitent

Plan

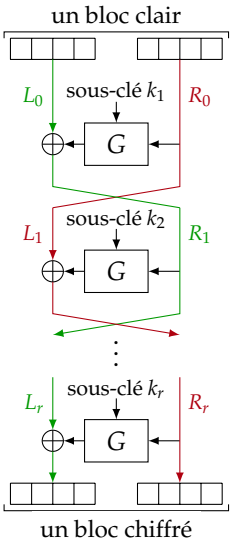
Réseaux de Feistel : présentation

Les **réseaux de Feistel** forment une autre famille de chiffrements par bloc.

Réseaux de Feistel : présentation

Les **réseaux de Feistel** forment une autre famille de chiffrements par bloc.

Pour le **chiffrement**, chaque bloc est découpé en deux morceaux de taille égale (souvent nommés L et R), auxquels on va appliquer alternativement une succession d'opérations :

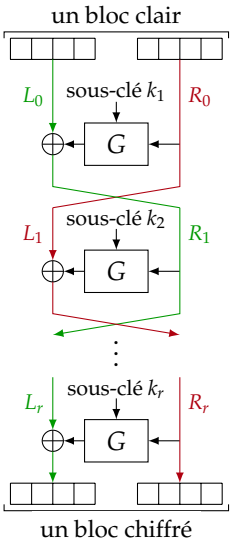


Réseaux de Feistel : présentation

Les **réseaux de Feistel** forment une autre famille de chiffrements par bloc.

Pour le **chiffrement**, chaque bloc est découpé en deux morceaux de taille égale (souvent nommés L et R), auxquels on va appliquer alternativement une succession d'opérations :

1. Le passage dans une fonction G qui prend comme paramètre la clé. La fonction G n'est **pas nécessairement** une bijection ; elle doit simplement avoir un comportement aléatoire et fortement non-linéaire.

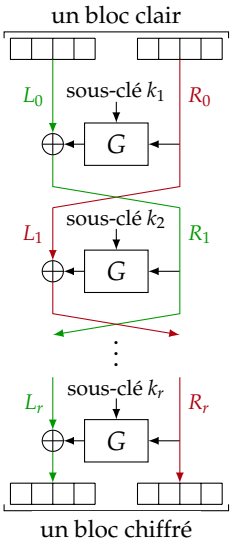


Réseaux de Feistel : présentation

Les **réseaux de Feistel** forment une autre famille de chiffrements par bloc.

Pour le **chiffrement**, chaque bloc est découpé en deux morceaux de taille égale (souvent nommés L et R), auxquels on va appliquer alternativement une succession d'opérations :

1. Le passage dans une fonction G qui prend comme paramètre la clé. La fonction G n'est **pas nécessairement** une bijection ; elle doit simplement avoir un comportement aléatoire et fortement non-linéaire.
2. Le XOR bit-à-bit avec l'autre morceau.



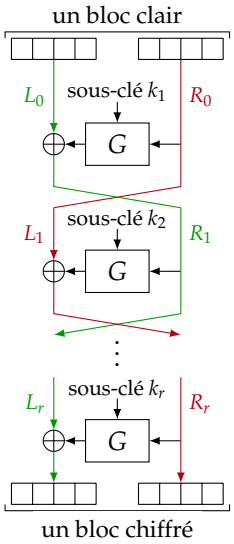
Réseaux de Feistel : présentation

Les **réseaux de Feistel** forment une autre famille de chiffrements par bloc.

Pour le **chiffrement**, chaque bloc est découpé en deux morceaux de taille égale (souvent nommés L et R), auxquels on va appliquer alternativement une succession d'opérations :

1. Le passage dans une fonction G qui prend comme paramètre la clé. La fonction G n'est **pas nécessairement** une bijection ; elle doit simplement avoir un comportement aléatoire et fortement non-linéaire.
2. Le XOR bit-à-bit avec l'autre morceau.

On peut écrire : $\begin{cases} L_{i+1} & = R_i \\ R_{i+1} & = L_i \oplus G(k_{i+1}, R_i) \end{cases}$



Réseaux de Feistel : présentation

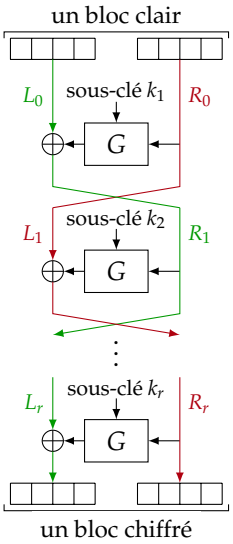
Les **réseaux de Feistel** forment une autre famille de chiffrements par bloc.

Pour le **chiffrement**, chaque bloc est découpé en deux morceaux de taille égale (souvent nommés L et R), auxquels on va appliquer alternativement une succession d'opérations :

1. Le passage dans une fonction G qui prend comme paramètre la clé. La fonction G n'est **pas nécessairement** une bijection ; elle doit simplement avoir un comportement aléatoire et fortement non-linéaire.
2. Le XOR bit-à-bit avec l'autre morceau.

On peut écrire : $\begin{cases} L_{i+1} & = R_i \\ R_{i+1} & = L_i \oplus G(k_{i+1}, R_i) \end{cases}$

Pour le **déchiffrement**, il suffit de refaire passer le chiffré dans le réseau de Feistel (inversant l'ordre des sous-clés).



Exemple : TEA

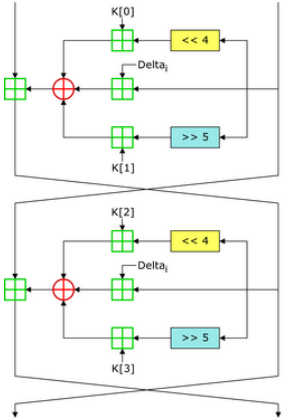
TEA (Tiny Encryption Algorithm) est un algorithme de chiffrement par bloc ayant une structure de **Feistel**, publié par D. Wheeler et R. Needham en 1994 (à la conférence FSE).

Exemple : TEA

TEA (Tiny Encryption Algorithm) est un algorithme de chiffrement par bloc ayant une structure de **Feistel**, publié par D. Wheeler et R. Needham en 1994 (à la conférence FSE).

Il est particulièrement simple à implanter :

- blocs de taille 64 bits ;
- clé de taille 128 bits ;
- 32 couples de tours identiques, comme **ci-contre**, ou le $\boxplus$ désigne l'addition avec retenue sur les entiers, et où Delta_i est une constante qui permet la diffusion.



Attaques sur TEA

TEA a été la cible d'attaques théoriques comme pratiques :

- Il a été remarqué que plusieurs clés peuvent donner le même résultat de chiffrement (on parle de **clé équivalentes**)
- Cela pose des problèmes si TEA veut être utilisé comme brique de base pour une fonction de hachage (penser intuitivement : plusieurs fichiers se hachent de la même manière). Cela a mené à une attaque sur les Microsoft XBOX :

> After reading Bruce Schneier's book on crypto, we learned that TEA was a really bad choice as a hash. The book says that TEA must never be used as a hash, because it is insecure if used this way. If you flip both bit 16 and 31 of a 32 bit word, the hash will be the same. We could easily patch a jump in the second bootloader so that it would not be recognized. This modified jump lead us directly into flash memory.
>
> But why did they make this mistake? **Obviously the designers knew nothing about crypto** - again! - and just added code without understanding it and without even reading the most basic books on the topic. A possible explanation why they chose TEA would be that they might have searched the internet for a "tiny" encryption algorithm - and got TEA.

Plan

Chiffrer plusieurs blocs

Pour chiffrement un message vu comme une succession de blocs, il y a différents **modes opératoires**, définis selon **la manière dont sont traités consécutivement les blocs** du texte clair.

Chiffrer plusieurs blocs

Pour chiffrement un message vu comme une succession de blocs, il y a différents **modes opératoires**, définis selon **la manière dont sont traités consécutivement les blocs** du texte clair.

On va en voir quatre :

1. **ECB** (Electronic Code Book)
2. **CBC** (Cipher Block Chaining)
3. **CFB** (Cipher FeedBack)
4. **OFB** (Output FeedBack)

Chiffrer plusieurs blocs

Pour chiffrement un message vu comme une succession de blocs, il y a différents **modes opératoires**, définis selon **la manière dont sont traités consécutivement les blocs** du texte clair.

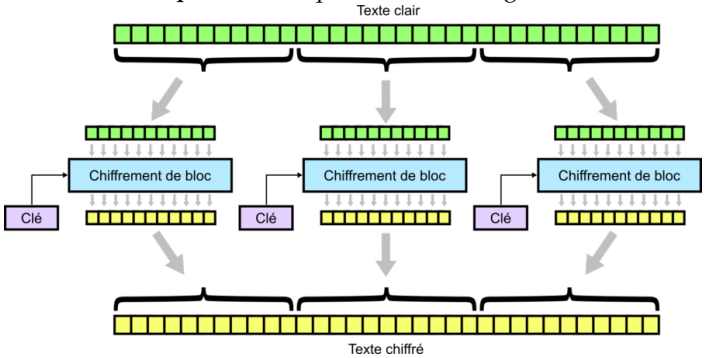
On va en voir quatre :

1. **ECB** (Electronic Code Book)
2. **CBC** (Cipher Block Chaining)
3. **CFB** (Cipher FeedBack)
4. **OFB** (Output FeedBack)

Il en existe beaucoup d'autres, selon les fonctionalités voulues pour le chiffrement.

ECB : Electronic Code Book

Le mode opératoire **ECB** (Electronic Code Book : « par dictionnaire ») est le plus simple. Il consiste à traiter **indépendemment** et **identiquement** chaque bloc du message.



source : Wikipedia

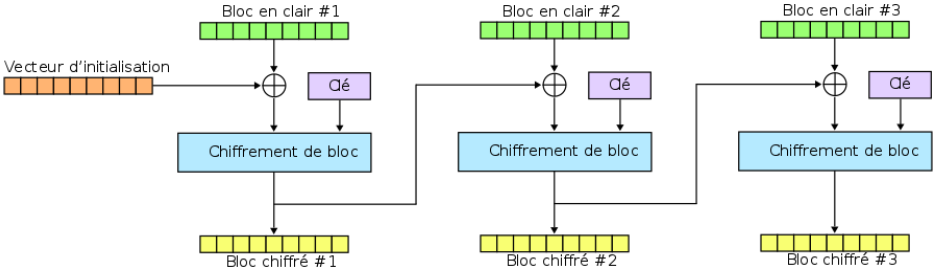
Avantage : peut se paralléliser.

(Gros) Problème : ce mode opératoire associe toujours un **même** bloc clair à un **même** bloc chiffré. On obtient donc un **chiffrement par substitution** dont l'alphabet est formé des blocs. Une attaque par analyse de fréquence des blocs devient possible...

CBC : Cipher Block Chaining

Le mode opératoire **CBC** (Cipher Block Chaining : « par chaînage ») traite le bloc courant en lui ajoutant (par un XOR) le bloc précédemment chiffré.

L'initialisation du processus se fait par un **vecteur d'initialisation** public, à générer aléatoirement pour chaque message à chiffrer.



source : Wikipedia

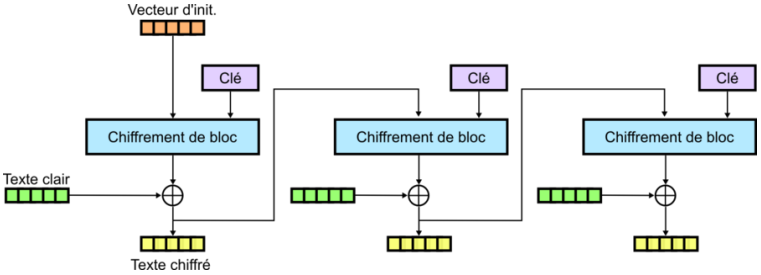
(Petit) Problème : on perd la parallélisation.

CFB : Cipher FeedBack

Le mode opératoire **CFB** (Cipher FeedBack : « par rétroaction ») permet de **transformer un chiffrement par bloc en chiffrement par flot**.

Le flux de clé est créé à partir de la sortie de la fonction de chiffrement.

Comme dans CBC, on réutilise le **chiffré précédent** pour chiffrer la partie "courante".



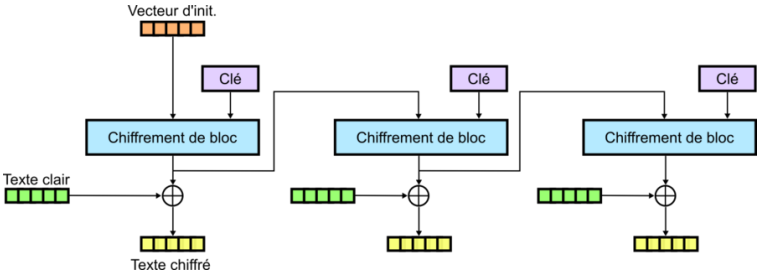
source : Wikipedia

OFB : Output FeedBack

Le mode opératoire **OFB** (Output FeedBack : « par rétroaction sur la sortie ») permet aussi de **transformer un chiffrement par bloc en chiffrement par flot**.

Le flux de clé est créé à partir de la sortie de la fonction de chiffrement.

La différence avec CFB est qu'on utilise le **flux de clé précédemment calculé** pour obtenir le flux de clé courant. Cela permet de **précalculer le flux de clé** si besoin.



source : Wikipedia

Plan

Le **principe de Kerckhoffs** exprime que la sécurité du système ne doit reposer **que sur le secret de la clé**.

Conséquence. On suppose que tout adversaire connaît les algorithmes engagés dans le protocole.

« **Adversaire** » ?

Le **principe de Kerckhoffs** exprime que la sécurité du système ne doit reposer **que sur le secret de la clé**.

Conséquence. On suppose que tout adversaire connaît les algorithmes engagés dans le protocole.

« **Adversaire** » ?
→ Différents modèles d'**adversaires**.

Le **principe de Kerckhoffs** exprime que la sécurité du système ne doit reposer **que sur le secret de la clé**.

Conséquence. On suppose que tout adversaire connaît les algorithmes engagés dans le protocole.

« **Adversaire** » ?
→ Différents modèles d'**adversaires**.

1. **Adversaires passifs.** Ces adversaires observent les communications qui transitent sur le canal public, mais ne les modifient pas.

Le **principe de Kerckhoffs** exprime que la sécurité du système ne doit reposer **que sur le secret de la clé**.

Conséquence. On suppose que tout adversaire connaît les algorithmes engagés dans le protocole.

« **Adversaire** » ?
→ Différents modèles d'**adversaires**.

1. **Adversaires passifs.** Ces adversaires observent les communications qui transitent sur le canal public, mais ne les modifient pas.
2. **Adversaires actifs.** Ces adversaires peuvent modifier les communications qui transitent sur le canal public.

Le **principe de Kerckhoffs** exprime que la sécurité du système ne doit reposer **que sur le secret de la clé**.

Conséquence. On suppose que tout adversaire connaît les algorithmes engagés dans le protocole.

« **Adversaire** » ?
→ Différents modèles d'**adversaires**.

1. **Adversaires passifs.** Ces adversaires observent les communications qui transitent sur le canal public, mais ne les modifient pas.
2. **Adversaires actifs.** Ces adversaires peuvent modifier les communications qui transitent sur le canal public.

→ Différents modèles d'**attaques**. Ces modèles varient suivant la primitive cryptographique (chiffrement, signature, etc.).

Enfin, on rappelle que l'on cherche à obtenir une sécurité **calculatoire**, c'est-à-dire que le système résiste à des capacités de calculs de l'ordre de $2^{80} - 2^{128}$ opérations.

Modes d'attaque

Avant de tenter d'opérer une attaque, l'attaquant peut avoir accès à des **informations supplémentaires**. Il existe donc différents **modes d'attaques** (scénarios d'attaque).

- **Attaque à chiffré seul** (COA, *ciphertext only attack*) : l'attaquant détient des chiffrés précédemment calculés.
  → Autrement dit, il observe ce qui passe sur le réseau.
- **Attaque à clair connu** (KPA, *known plaintext attack*) : l'attaquant a accès des couples de clairs/chiffrés précédemment calculés.
  → Autrement dit, il a vu des messages être chiffrés/déchiffrés.
- **Attaque à clair choisi** (CPA, *chosen plaintext attack*) : l'attaquant a accès à des couples de clairs/chiffrés dont il a pu choisir les clairs.
  → Autrement dit, il a eu accès à la fonction de chiffrement pendant un certain temps.
- **Attaques à chiffré choisi** (CCA, *chosen ciphertext attack* : l'attaquant a accès à des couples de clairs/chiffrés dont il a pu choisir les clairs ou les chiffrés.
  → Autrement dit, il a eu accès à la fonction de déchiffrement pendant un certain temps.

Types d'attaques (génériques)

L'objectif de l'attaque peut être de différent ordre ou gravité. On parle de **types d'attaque** différents.

Ces types d'attaques diffèrent selon le type de chiffrement considéré (symétrique ou non, flot/bloc, etc.).

Pour du **chiffrement**, on a tout de même des types **génériques** :

- **Reconstruction complète de la clé ou du secret :** en quelque sorte, ça revient à « savoir inverser » la fonction de chiffrement ;
- **Distinction de l'aléa uniforme :** on arrive à distinguer un couple clair/chiffré d'un couple clair/aléa tiré uniformément.

En cryptographie moderne, les systèmes **doivent** se prémunir des attaques de distinction de l'aléa. Le cas échéant, on parle de système **indistinguable** (IND).

Exemple de types d'attaques pour un chiffrement par flot

Pour un **chiffrement par flot**, on peut raffiner un peu les types d'attaques :

- **reconstruction de la clé secrète :** en quelque sorte, ça revient à « inverser » le générateur ;
- **reconstruction de l'état initial** du générateur ;
- **prédiction du prochain bit** de la suite chiffrante (avec bonne probabilité) ;
- **distinction de l'aléa uniforme** : on arrive à distinguer la suite chiffrante d'une suite de bits tirés uniformément.

Plan

Attaque par force brute

L'**attaque par force brute** consiste à effecteur le **calcul exhaustif** de toutes les possibilités des données secrètes (typiquement, la clé).

- Elle permet de donner un premier aperçu des tailles **nécessaires** de certains paramètres du système (notamment les clés).

  il y a 2^{128} possibilités pour une clé de 128 bits

  ordres de grandeurs :

nombre de ...	valeur
cycles par secondes sur un CPU	$4\,000\,000\,000 \simeq 2^{32}$
secondes en une année	$365 \times 24 \times 3600 = 31\,536\,000 \simeq 2^{25}$
atomes dans l'univers	$10^{79} \simeq 2^{237}$

- Elle peut se compléter d'une **attaque par dictionnaire** (liste de mots probables par exemple)

Attaque par force brute : exercice

On emploie l'attaque par dictionnaire/force brute sur des protocoles d'authentification par **mot de passe**.

Exercice : on utilise une authentification par mot de passe avec une fonction de hachage qui prend 1 μs à être évaluée.

Pour se prémunir face à une attaque d'une durée d'un mois, montée sur 16 cœurs en parallèle, quelle est la taille minimale nécessaire d'un mot de passe

1. qui utilise seulement des chiffres (0–9) ?
2. qui utilise chiffres, lettres minuscules et majuscule, et quelques caractères spéciaux ?
3. qui utilise une suite de mots de la langue française ($\simeq$ 30 000 mots), en nombre de mots utilisés ?

Analyse fréquentielle

L'**attaque par analyse fréquentielle** cherche à utiliser les **biais statistiques** des textes clairs pour en déduire des informations, notamment sur la clé de chiffrement.

L'**attaque par analyse fréquentielle** cherche à utiliser les **biais statistiques** des textes clairs pour en déduire des informations, notamment sur la clé de chiffrement.

Ce type d'attaque peut être plus ou moins évolué, suivant les outils utilisés :

- attaque sur Cesar : on sait que le E est la lettre la plus utilisée
- attaque sur Vigenère : indice de coincidence
- des analyses par test du χ^2 peuvent permettre de distinguer un chiffré d'un texte aléatoire
- ...

Plan

L'immense majorité des systèmes modernes ont des **clés secrètes de taille** ≥ 128 **bits**.

$\rightarrow$ Cela rend l'attaque par force brute sur la clé impossible.

$\rightarrow$ C'est une **recommandation forte** de l'**ANSSI** (Agence nationale de la sécurité des systèmes d'information) depuis 2020.

L'immense majorité des systèmes modernes ont des **clés secrètes de taille** ≥ 128 **bits**.

→ Cela rend l'attaque par force brute sur la clé impossible.

→ C'est une **recommandation forte** de l'**ANSSI** (Agence nationale de la sécurité des systèmes d'information) depuis 2020.

Pour attaquer un système l'idée générique est la suivante :

1. **Restreindre l'espace de recherche** des clés, en trouvant des **équations** que doivent satisfaire entre eux les bits de la clé secrète.
2. Après avoir suffisamment restreint l'espace des clés possibles, on effectue une recherche exhaustive d'une clé valide. Ce test de validité se fait grâce à des couples clairs/chiffrés précédemment calculés.

Cryptanalyse linéaire

La **cryptanalyse linéaire** est un outil de cryptanalyse qui cible principalement les **chiffrements symétriques, par bloc, et itératifs**.

Cryptanalyse linéaire

La **cryptanalyse linéaire** est un outil de cryptanalyse qui cible principalement les **chiffrements symétriques, par bloc, et itératifs**.

Une **équation linéaire** (ou affine) entre une séquence de bits $(x_1, \ldots, x_n)$ est une équation de la forme :

$$(a_1 \wedge x_1) \oplus (a_2 \wedge x_2) \oplus \cdots \oplus (a_n \wedge x_n) = b$$

où les (a_i) et b sont des bits.

La **cryptanalyse linéaire** est un outil de cryptanalyse qui cible principalement les **chiffrements symétriques, par bloc, et itératifs**.

Une **équation linéaire** (ou affine) entre une séquence de bits $(x_1, \ldots, x_n)$ est une équation de la forme :

$$(a_1 \wedge x_1) \oplus (a_2 \wedge x_2) \oplus \cdots \oplus (a_n \wedge x_n) = b$$

où les (a_i) et b sont des bits.

Exemple. On considère l'écriture en binaire des entiers compris entre 0 et 7 :

i	0	1	2	3	4	5	6	7
x_0	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1
x_2	0	0	0	0	1	1	1	1

Alors, le sous-ensemble d'entiers $\{1, 3, 6, 4\}$ satistfait l'équation $x_0 \oplus x_2 = 1$

Cryptanalyse linéaire

En **cryptanalyse linéaire**, on va chercher des équations linéaires liant (avec grande probabilité) les bits d'entrée et de sortie de la fonction de tour. On appelle cela une **approximation linéaire** de la fonction de tour.

Cryptanalyse linéaire

En **cryptanalyse linéaire**, on va chercher des équations linéaires liant (avec grande probabilité) les bits d'entrée et de sortie de la fonction de tour. On appelle cela une **approximation linéaire** de la fonction de tour.

Pourquoi cherche t-on une approximation linéaire ?
Pour une fonction de tour **linéaire**, on obtient une relation linéaire entre les bits de la clés, et les bits des couples clairs/chiffrés. Avec suffisamment de tels couples, on obtiendrait aisément la clé secrète par **résolution d'un système linéaire**.

Cryptanalyse linéaire

En **cryptanalyse linéaire**, on va chercher des équations linéaires liant (avec grande probabilité) les bits d'entrée et de sortie de la fonction de tour. On appelle cela une **approximation linéaire** de la fonction de tour.

Pourquoi cherche t-on une approximation linéaire ?
Pour une fonction de tour **linéaire**, on obtient une relation linéaire entre les bits de la clés, et les bits des couples clairs/chiffrés. Avec suffisamment de tels couples, on obtiendrait aisément la clé secrète par **résolution d'un système linéaire**.

Parmi ces approximations linéaires possibles, on va chercher celles :

1. qui sont vérifiées avec la plus grande probabilité (sur le choix de l'entrée/sortie)
2. qui lient le plus de bits (de l'entrée et de la sortie)

Exemple : supposons que la fonction de tour F soit donnée par la représentation suivante :

- La taille de bloc est 2 bits.
- La fonction est la suivante : à $x \in \{0, \ldots, 3\}$, écrit $x = (x_0 x_1)_2$ en binaire, on associe $y = F(x) = x^2 \mod 4$. Autrement dit :

x	00	01	10	11
$y = F(x)$	00	01	00	01

[cette fonction n'est pas réaliste pour une chiffrement ; elle n'est même pas bijective]

Exemple : supposons que la fonction de tour F soit donnée par la représentation suivante :

- La taille de bloc est 2 bits.
- La fonction est la suivante : à $x \in \{0, \ldots, 3\}$, écrit $x = (x_0 x_1)_2$ en binaire, on associe $y = F(x) = x^2 \mod 4$. Autrement dit :

x	00	01	10	11
$y = F(x)$	00	01	00	01

[cette fonction n'est pas réaliste pour une chiffrement ; elle n'est même pas bijective]

Cherchons des équations linéaires vérifiées par des bits d'entrée/sortie de la fonction. On a :

1. $y_0 = 0$, dans 100% des cas ;
2. $x_1 = y_1$, autrement dit $x_1 \oplus y_1 = 0$, également dans 100% des cas.

Exemple : supposons que la fonction de tour F soit donnée par la représentation suivante :

- La taille de bloc est 2 bits.
- La fonction est la suivante : à $x \in \{0, \ldots, 3\}$, écrit $x = (x_0 x_1)_2$ en binaire, on associe $y = F(x) = x^2 \mod 4$. Autrement dit :

x	00	01	10	11
$y = F(x)$	00	01	00	01

[cette fonction n'est pas réaliste pour une chiffrement ; elle n'est même pas bijective]

Cherchons des équations linéaires vérifiées par des bits d'entrée/sortie de la fonction. On a :

1. $y_0 = 0$, dans 100% des cas ;
2. $x_1 = y_1$, autrement dit $x_1 \oplus y_1 = 0$, également dans 100% des cas.

Par conséquent :

$$y_0 \oplus x_1 \oplus y_1 = 0$$

est une **bonne approximation linéaire**, car elle couvre $3/4$ des bits et elle est valide 100% des cas.

Exemple : supposons que la fonction de tour F soit donnée par la représentation suivante :

- La taille de bloc est 2 bits.
- La fonction est la suivante : à $x \in \{0, \ldots, 3\}$, écrit $x = (x_0 x_1)_2$ en binaire, on associe $y = F(x) = x^2 \mod 4$. Autrement dit :

x	00	01	10	11
$y = F(x)$	00	01	00	01

[cette fonction n'est pas réaliste pour une chiffrement ; elle n'est même pas bijective]

Cherchons des équations linéaires vérifiées par des bits d'entrée/sortie de la fonction. On a :

1. $y_0 = 0$, dans 100% des cas ;
2. $x_1 = y_1$, autrement dit $x_1 \oplus y_1 = 0$, également dans 100% des cas.

Par conséquent :

$$y_0 \oplus x_1 \oplus y_1 = 0$$

est une **bonne approximation linéaire**, car elle couvre $3/4$ des bits et elle est valide 100% des cas.

L'équation $x_0 \oplus x_1 \oplus y_0 \oplus y_1 = 0$ est **beaucoup moins bonne** : certes, elle couvre tous les bits mais n'est valable que 50% du temps (c'est l'espérance moyenne).

Et ensuite ?

Et ensuite ?

1. Si on a une approximation linéaire d'**un tour** de F, valable avec probabilité $1 - \varepsilon$ (penser 98% du temps), alors on a une aproximation linéaire **de** r **tours** de F valable avec probabilité $(1 - \varepsilon)^r$ (penser environ $\simeq (100 - 2r)\%$ du temps).

Et ensuite ?

1. Si on a une approximation linéaire d'**un tour** de F, valable avec probabilité $1 - \varepsilon$ (penser 98% du temps), alors on a une aproximation linéaire **de** r **tours** de F valable avec probabilité $(1 - \varepsilon)^r$ (penser environ $\simeq (100 - 2r)\%$ du temps).
2. Cette approximation linéaire nous donne ensuite une relation entre l'entrée $\boldsymbol{m}$ du chiffrement (le clair), la sortie $\boldsymbol{c}$ du chiffrement (le chiffré) et la sous-clé k_r.

Et ensuite ?

1. Si on a une approximation linéaire d'**un tour** de F, valable avec probabilité $1 - \varepsilon$ (penser 98% du temps), alors on a une aproximation linéaire **de** r **tours** de F valable avec probabilité $(1 - \varepsilon)^r$ (penser environ $\simeq (100 - 2r)\%$ du temps).
2. Cette approximation linéaire nous donne ensuite une relation entre l'entrée $\boldsymbol{m}$ du chiffrement (le clair), la sortie $\boldsymbol{c}$ du chiffrement (le chiffré) et la sous-clé k_r.
3. On parcourt alors tous les candidats potentiels pour k_r, et on élimine ceux pour lesquels il y a un nombre insuffisant de couples clairs/chiffrés qui satisfont la relation.

Et ensuite ?

1. Si on a une approximation linéaire d'**un tour** de F, valable avec probabilité $1 - \varepsilon$ (penser 98% du temps), alors on a une aproximation linéaire **de** r **tours** de F valable avec probabilité $(1 - \varepsilon)^r$ (penser environ $\simeq (100 - 2r)\%$ du temps).
2. Cette approximation linéaire nous donne ensuite une relation entre l'entrée $\boldsymbol{m}$ du chiffrement (le clair), la sortie $\boldsymbol{c}$ du chiffrement (le chiffré) et la sous-clé k_r.
3. On parcourt alors tous les candidats potentiels pour k_r, et on élimine ceux pour lesquels il y a un nombre insuffisant de couples clairs/chiffrés qui satisfont la relation.
4. On teste toutes les clés candidates restantes.

Et ensuite ?

1. Si on a une approximation linéaire d'**un tour** de F, valable avec probabilité $1 - \varepsilon$ (penser 98% du temps), alors on a une aproximation linéaire **de** r **tours** de F valable avec probabilité $(1 - \varepsilon)^r$ (penser environ $\simeq (100 - 2r)\%$ du temps).
2. Cette approximation linéaire nous donne ensuite une relation entre l'entrée $\boldsymbol{m}$ du chiffrement (le clair), la sortie $\boldsymbol{c}$ du chiffrement (le chiffré) et la sous-clé k_r.
3. On parcourt alors tous les candidats potentiels pour k_r, et on élimine ceux pour lesquels il y a un nombre insuffisant de couples clairs/chiffrés qui satisfont la relation.
4. On teste toutes les clés candidates restantes.

Plus de détails dans le TP...

En **cryptanalyse différentielle**, on cherche des biais provenant de **différentielles**. Une différentielle est une paire de chaînes $(\boldsymbol{a}, \boldsymbol{b})$ telle que le nombre $N(\boldsymbol{a}, \boldsymbol{b})$ d'entrées $\boldsymbol{x}$ vérifiant

$$E_K(\boldsymbol{x} \oplus \boldsymbol{a}) = E_K(\boldsymbol{x}) \oplus \boldsymbol{b}$$

est particulièrement important.

En **cryptanalyse différentielle**, on cherche des biais provenant de **différentielles**. Une différentielle est une paire de chaînes $(\boldsymbol{a}, \boldsymbol{b})$ telle que le nombre $N(\boldsymbol{a}, \boldsymbol{b})$ d'entrées $\boldsymbol{x}$ vérifiant

$$E_K(\boldsymbol{x} \oplus \boldsymbol{a}) = E_K(\boldsymbol{x}) \oplus \boldsymbol{b}$$

est particulièrement important.

C'est une nouvelle fois une **propriété distinguante** qu'une transformation aléatoire ne devrait pas avoir :

« deux entrées de différence $\boldsymbol{a}$ ont une probabilité **anormalement haute** d'avoir comme différence en sortie $\boldsymbol{b}$ »

Cryptanalyse différentielle

En **cryptanalyse différentielle**, on cherche des biais provenant de **différentielles**. Une différentielle est une paire de chaînes $(\boldsymbol{a}, \boldsymbol{b})$ telle que le nombre $N(\boldsymbol{a}, \boldsymbol{b})$ d'entrées $\boldsymbol{x}$ vérifiant

$$E_K(\boldsymbol{x} \oplus \boldsymbol{a}) = E_K(\boldsymbol{x}) \oplus \boldsymbol{b}$$

est particulièrement important.

C'est une nouvelle fois une **propriété distinguante** qu'une transformation aléatoire ne devrait pas avoir :

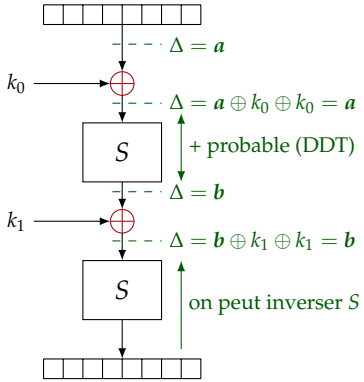
« deux entrées de différence $\boldsymbol{a}$ ont une probabilité **anormalement haute** d'avoir comme différence en sortie $\boldsymbol{b}$ »

Comme avec la cryptanalyse linéaire, l'idée est de **se ramener à un tour** de la structure itérative. On stocke donc, dans une table carrée indexée par $\boldsymbol{a}$ et $\boldsymbol{b}$ appelée DDT (*differential distribution table*), les valeurs de $N(\boldsymbol{a}, \boldsymbol{b})$ **pour la fonction de tour** F.

Poursuite de l'attaque pour un chiffrement réduit à 2 tours

Remarque : comme on sait inverser S, on peut considérer que la sortie est après le XOR de k_1 (quitte à appliquer S^{-1} à toutes les sorties).

1. On calcule la DDT pour **un** tour de la fonction de tour F.
2. On choisit $(\boldsymbol{a}, \boldsymbol{b})$ non-nul dont la valeur de $DDT(\boldsymbol{a}, \boldsymbol{b})$ est maximale. On calcule la liste $L(\boldsymbol{a}, \boldsymbol{b})$ de toutes les entrées $\boldsymbol{x}$ correspondantes.
3. On génère une liste de **couples de clairs** de différence $\boldsymbol{a}$. On calcule les chiffrés associés, puis on cherche un couple dont la différence en sortie est $\boldsymbol{b}$ (ça peut échouer). Notons $(\boldsymbol{m}, \boldsymbol{c})$ l'un des clair/chiffré correspondant à ce couple. On a :
$$\boldsymbol{c} = E_K(\boldsymbol{m}) \quad \text{et} \quad \boldsymbol{c} \oplus \boldsymbol{b} = E_K(\boldsymbol{m} \oplus \boldsymbol{a})$$
4. Le chemin différentiel $\boldsymbol{a} \longrightarrow \boldsymbol{b}$ est assez probable. On peut donc espérer qu'il a été pris, c'est-à-dire que :
$$\boldsymbol{m} \oplus k_0 = \boldsymbol{x} \quad \text{et} \quad \boldsymbol{y} \oplus k_1 = \boldsymbol{c}$$
où $(\boldsymbol{x}, \boldsymbol{y}) \in L(\boldsymbol{a}, \boldsymbol{b})$. En parcourant tous les $(\boldsymbol{x}, \boldsymbol{y})$ de $L(\boldsymbol{a}, \boldsymbol{b})$, on a une liste de clés (k_0, k_1) potentielles.
5. On teste toutes ces clés.



Démonstration en parallèle

Démo...