CRYPTOGRAPHIE GENERALE

Pierre-Louis BAYLE - Benjamin BILLET

Sommaire

I. Prologue

1. Un peu d'histoire

La cryptographie est une discipline de la cryptologie (étymologiquement "science des secrets") dont les origines remontent à l'Antiquité. Nabuchodonosor, par exemple, cachait des messages sur le crâne rasé des esclaves, le secret étant conservé par la repousse des cheveux.
De même, les Grecs, en enroulant une bande de cuir autour d'un bâton, et en déroulant cette bande après écriture du message, rendaient un message lisible uniquement pour une personne possédant un bâton de taille et de diamètre identiques.

Jules César fut l'un des premiers à imaginer une technique "mathématique" de cryptage et donna naissance à l'un des plus anciens algorithmes par substitution ; le principe consistant à déplacer chaque lettre de trois rangs dans l'alphabet (A devenant D, Y devenant B, etc…).

De même, la notion de clé fut introduite à cette époque là, avec notamment le carré de Polybe qui positionne les lettres de l'alphabet à l'intérieur d'un carré de 5x5 cases, à partir duquel les positions en hauteur et en largeur d'une lettre forment le codage de ladite lettre. La numérotation des cases devenant alors la clé d'encodage.

Toutefois, pendant des siècles, les algorithmes de chiffrement restèrent essentiellement monoalphabétiques qui présentait un défaut majeur. En effet, dans une langue donnée, la répartition des lettres n'est pas uniforme, il est alors possible de déchiffrer le message codé en analysant la fréquence de chaque symbole ou groupe de symboles.

A la fin du XVI^{eme} siècle, Blaise de Vigenère proposa le premier chiffrement polyalphabétique, qui porte son nom. Sa méthode repose entièrement sur un décalage (comme l'algorithme de César) à la différence que ce décalage va varier de lettre à lettre par rapport à une clé.
Malgré sa simplicité de mise en œuvre, il fallut attendre le milieu du XIX^{eme} pour que ce code soit cassé.

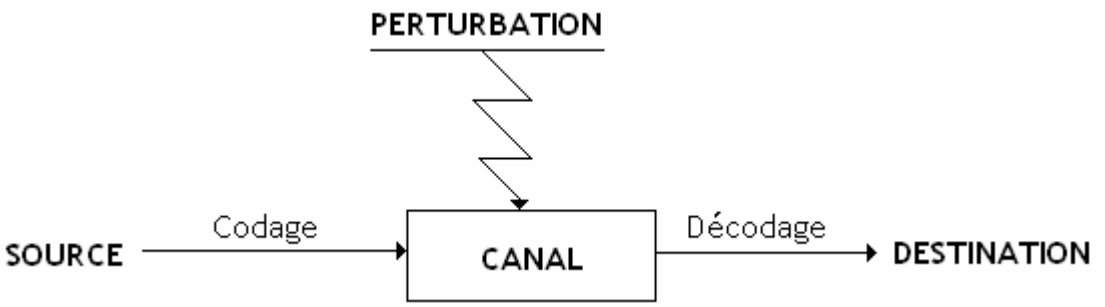
C'est cependant lors de la première et de la seconde guerre mondiale que la cryptographie prit une importance majeure. En effet, le système Enigma fut mit au point par les allemands et était capable, dans sa première version, d'avoir plus de 10^{16} possiblités d'encodage pour un caractère donné.
En réponse fut construite "The Bombe", première machine de décodage, capable de décrypter, en une vingtaine d'heure, un message issu d'Enigma. Vinrent ensuite Colossus 1 et 2, des appareils de plus en plus puissants et qui auront un rôle important dans la victoire des alliés.

Depuis, les méthodes toujours plus solides, plus complexes et plus sûres ne cessent de rivaliser d'ingéniosité. Avec l'avènement de l'informatique et l'explosion des besoins en communication, la cryptographie continue à évoluer et reste le nerf de la guerre de la sécurité informatique.

2. Enjeux et positionnements

Par abstraction, on pourrait considérer que toute communication est issue d'un cryptage. Une langue ou un dialecte, par exemple, sont des formes d'encodage de l'information, que ce soit au niveau de la syntaxe ou de la sémantique.

Ainsi, même un simple dialogue fait appel à de nombreux mécanismes de codage / décodage, ce de manière naturelle.
On peut, dès lors, définir une abstraction très simplifiée de la communication :



Une transmission de la source vers la destination va donc dépendre de trois éléments fondamentaux :

- Efficacité et concision : compression du message.
- Sécurité de l'information : cryptage du message.
- Intégrité : détection et correction des erreurs dues aux perturbations ou à des manipulations délictueuses.

Dans cette étude, nous nous concentrerons principalement sur la sécurité de l'information, et sur les méthodes garantissant cette sécurité.

II. Pré-requis

1. Vocabulaire

Source : Données initiales, le message à crypter. Noté M.

Code[1] : Le message crypté, noté C.

Alphabet : Ensemble fini, noté $V = \{v_1 , ... , v_k\}$, contenant les éléments employés pour coder une information. De manière abstraite, on peut considérer que toute information est décrite par un code (lettres, phonèmes, sons, signes, etc…).

Alphabet source et alphabet code : Respectivement l'alphabet de la source et l'alphabet du code.
Exemple : Alphabet Noir&Blanc {noir, blanc} traduit en alphabet binaire {0, 1}.

V^* : Ensemble des chaînes possibles (par concaténation) sur l'alphabet V.

V^+ : Ensemble des chaînes de longueur non nulle sur l'alphabet V.

Clé : Clé secrète, notée K, pour l'algorithme de codage / décodage d'un message.

Code monoalphabétique : La fonction d'encodage de l'alphabet source à l'alphabet cible est une bijection (un symbole source correspond toujours à un même symbole cible).

Code polyalphabétique : A l'inverse du précédent, un symbole source peut correspondre à un nombre potentiellement infini de symboles cibles.

2. Rappels généraux

a) Probabilités

Vocabulaire

Expérience aléatoire : Processus dont le résultat est incertain.

Evénement : Résultat possible d'une expérience aléatoire.

Indépendance : Deux évènements A et B sont dits *indépendants* si le fait que A se réalise ne donne aucune information sur la probabilité de B.

Loi de probabilité : Appelé aussi "distribution de probabilités", désigne l'ensemble des valeurs prises par la fonction de probabilité (probabilité de chaque évènement).

Univers : Ensemble des événements, noté Ω. Si l'univers est un ensemble discret (fini et/ou dénombrable) on parle de probabilités discrètes.

Distribution uniforme : Tous les évènements ont la même probabilité.

[1] Désigne aussi l'ensemble des transformations nécessaires pour en arriver au message codé (code).

Propriétés

- $P(X)$, probabilité de l'événement X, $0 \le P(X) \le 1$

- $P(\Omega) = 1$
  $P(\varnothing) = 0$

- Soit A et B deux évènements disjoints :
  $A \cap B = \varnothing$
  $P(A \cup B) = P(A) + P(B)$

- Soit A et B deux évènements indépendants :
  $P(A \cap B) = P(A)P(B)$

- Probabilité conditionnelle (probabilité qu'un évènement A se produise si un évènement B s'est déjà produit) :
  $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$

- Formule de Baye
  Soit un ensemble d'évènements $\{A_1, \ldots, A_n, B\}$, les probabilités $P(A_k \mid B)$ peuvent être calculées en fonction des $P(B \mid A_k)$ par :

$$P(A_k \mid B) = \frac{P(A_k \cap B)}{P(B)} = \frac{P(B \mid A_k)P(B)}{\sum_i P(B \mid A_i)P(A_i)}$$

- Lemme de Gibbs
  Soient $(p_1,\ldots,p_n)$ et $(q_1,\ldots,q_n)$ deux lois de probabilités discrètes telles que $\forall i$, $p_i > 0$ et $q_i > 0$

$$\sum_{i=1}^{n} p_i \times \log\left(\frac{p_i}{q_i}\right) \le 0$$

b) Groupes

Un groupe *(G, ∗)* est un ensemble muni d'un opérateur binaire interne vérifiant les propriétés suivantes :

- $*$ est associative : $\forall a,b,c \in G,\ a*(b*c) = (a*b)*c$
- $\exists\, e \in G$ tel que $\forall a \in G \ \Rightarrow\ a*e = e*a = e$
- $\forall a \in G,\ \exists\, a^{-1} \in G$ tel que $a*a^{-1} = a^{-1}*a = e$

Si la loi est commutative ($a*b = b*a$) le groupe est dit *abélien*.

On dit qu'un sous ensemble *H* de *G* est un sous groupe *G* lorsque *H* a une structure de groupe pour la loi $*$.

Pour un élément *a* d'un groupe *G*, on note a^n la répétition de la loi $*$ tel que
$a^n = \underbrace{a * ... * a}_{n \text{ termes}} \quad \forall n \in \mathbb{N}^*$

Si un élément $g \in G$ est tel que pour tout $a \in G$ il existe $i \in \mathbb{Z}$, tel que $a = g^i$ alors *g* est un *générateur* du groupe (*G*, $*$) appelé aussi *racine primitive*.
Un groupe est dit *cyclique* s'il possède au moins un générateur.

Soient un groupe (*G*, $*$) et $a \in G$. L'ensemble $\{a^i, i \in \mathbb{Z}\}$ est un sous groupe noté <*a*> ou G_a. Si ce sous groupe est fini, son *cardinal* est l'*ordre de a*. Si *G* est fini, le cardinal de tout sous groupe de *G* divise le cardinal de *G*.

Théorème de Lagrange : Dans un groupe fini abélien (*G*, $*$) de cardinal *n*, pour tout $x \in G : x^n = e$.

c) Anneaux

Un anneau *(A, +, ×)* est un ensemble muni de deux opérateurs binaires internes vérifiant les propriétés suivantes :

- *(A, +)* est un groupe abélien
- × est associative : $\forall a,b,c \in A,\ a \times (b \times c) = (a \times b) \times c$
- × est distributive sur + : $\forall a,b,c \in A,\ a \times (b + c) = (a \times b) + (a \times c)$ et $(b + c) \times a = (b \times a) + (c \times a)$
- Tous les éléments de *A* ont un inverse sur la loi +, mais pas forcément sur la loi ×. L'ensemble des inversibles pour la loi × est souvent noté *A**.

Si × possède un élément neutre dans *A*, *A* est dit *unitaire*.
Si × est commutative, *A* est dit *commutatif*.
Un anneau commutatif est *intègre* s'il ne possède pas de diviseur par 0, c'est-à-dire que pour deux éléments *a* et *b* vérifiant *ab = 0* alors forcément l'un d'eux au moins est nul.
Un *idéal* I est un sous-groupe d'un anneau *A* pour la loi + qui est *absorbant* pour la loi × : $\forall g \in I,\ a \in A \Rightarrow a \times g \in I$.
Pour tout $x \in A$, la partie $Ax = \{ax\},\ a \in A$ est un idéal de *A* appelé *idéal engendré* par *x*.
Un idéal I est dit *principal* s'il existe un générateur *x* tel que $I = Ax$.
Un anneau est dit *principal* si et seulement si tout ses idéaux sont principaux.

Pour un élément *a* d'un anneau *A*, on note *n.a* (ou *na*) la somme *a* + … + *a* portant sur n termes égaux à *a*, $\forall n \in \mathbb{N}^*$.

On appelle *caractéristique de l'anneau* le plus petit entier naturel *n* non nul tel que
$n.e^* = \underbrace{e^* + ... + e^*}_{n \text{ termes}} = e^+$
Si *n* n'existe pas, l'anneau est dit de *caractéristique 0*.

Deux anneaux $A(+_A,\times_A)$ et $B(+_B,\times_B)$ sont isomorphes lorsqu'il existe une bijection $f : A \to B$ vérifiant $\forall x,y \in A$:

$$f(x +_A y) = f(x) +_B f(y) \text{ et } f(x \times_A y) = f(x) \times_B f(y)$$

Si *E* est un ensemble quelconque et (*A*, +, x) un anneau tel qu'il existe une bijection f de *E* sur *A*, alors *E* peut être muni d'une structure d'anneau, évidemment isomorphe à A :

$$x +_E y = f^{-1}\left(f(x) + f(y)\right) \text{ et } x \times_E y = f^{-1}\left(f(x) \times f(y)\right)$$

Une *fonction euclidienne*, ou *stathme* υ est une fonction qui associe à tout élément non nul d'un anneau, un entier positif ou nul. On dit qu'un anneau muni d'une fonction euclidienne est *euclidien* si pour tout couples d'éléments *a* et *b* de cet anneau, il existe un *q* et *r* tels que $a = bq + r$ et $\upsilon(r) < \upsilon(b)$.
Cette opération est la *division euclidienne,* et les nombres *q* et *r* sont respectivement le *quotient* et le *reste,* notés *q = a div b* et *r = a mod b*.
Tout anneau euclidien est principal, ceci sous entend l'existence d'un *pgcd* pour tout couple d'éléments *(a, b)*. Ce pgcd est le générateur de l'idéal *Aa + Ab*.

d) Corps

Un corps (*A*, +, x) est un ensemble muni de deux opérateurs binaires internes vérifiant les propriétés suivantes :

- (*A*, +, ×) est un anneau unitaire
- (*A* \ {0}, ×) est un groupe

Si (*A*, +, ×) est commutatif, on parle de *corps commutatif*.
L'opposé de *x* par la loi + est noté $-x$. L'opposé de *x* par la loi x est noté x^{-1}.
La *caractéristique* d'un corps est sa caractéristique en tant qu'anneau.
Deux corps commutatifs sont *isomorphes* lorsqu'ils sont isomorphes en tant qu'anneaux.
On dit qu'un sous ensemble *C* d'un corps *K* est un sous corps de *K* lorsque les restrictions des opérations de *C* à *K* confèrent à *C* une structure de corps.

3. Notions préliminaires

a) Source d'information

Une source d'information est constituée d'un *alphabet source*, noté *S*, et d'une *loi de probabilité d'occurrence*, notée *P*, de chaque élément de cet alphabet (p_i est la probabilité d'apparition de l'élément s_i).
Dés lors, la source d'information est notée *(S,P)*.

Source sans mémoire : La probabilité de chaque évènement (p_i) reste stable au cours de l'émission du message et est indépendante des autres évènements.

Exemple : *3 6 4 8 9 5 1 6* est une suite de valeurs aléatoires. Chaque valeur est indépendante (le *6* n'avait pas plus de chance de sortir après le *3* qu'après le *1*).

Source markovienne : Les probabilités de chaque évènement dépendent des évènements émis précédemment.
Si l'on note p_{ij} la probabilité d'occurrence de s_i sachant que s_j vient d'être émis alors la probabilité p_i est égale à la somme des probabilités d'émission de s_i par rapport à tous les éléments de l'alphabet S.

$$p_a = \sum_j p_{aj}$$

Exemple : L'alphabet occidental. Le caractère "n" a plus de chance d'apparaître à la suite d'une voyelle qu'à la suite d'une consonne.

Exemple 2 : Soit une source d'information *(S, P)* munie de $S = \{a, b\}$ et $P = \{p_{aa}, p_{ab}, p_{bb}\}$ où p_{ab} est la probabilité d'occurrence de *a* sachant que *b* vient d'être émis.

$p_a = \sum_j p_{aj}$
$= p_{aa} + p_{ab}$

Source sans redondance : La distribution des probabilités de chaque évènement est uniforme.

b) Entropie

L'entropie est une mesure du désordre d'une source d'information *S*, notée *H(S)*, se calcule de la façon suivante.

$$H(S) = H(P) = -\sum_{i=1}^{n} p_i \times \log_2\left(p_i\right)$$

$$H(S) = H(P) = \sum_{i=1}^{n} p_i \times \log_2\left(\frac{1}{p_i}\right)$$

Propriété : $\forall S,\ 0 \le H(S) \le \log_2 n$ où *n* est le nombre d'éléments de *S*.

H(S) tend vers $\log_2 n$ lorsque le désordre est maximal (distribution uniforme).

Entropie conjointe
Soient $S_1(S_1, P_1)$ et $S_2(S_2, P_2)$ deux sources sans mémoire dont les évènements d'une source ne sont pas forcément indépendants de l'autre source.
On note $S_1 = \{s_{11}, \ldots, s_{1n}\}$, $P_1 = \{p_1, \ldots, P_n\}$ et $S_2 = \{s_{21}, \ldots, s_{1m}\}$, $P_2 = \{p_2, \ldots, P_m\}$.

On note la probabilité d'occurrence conjointe de s_{1i} et s_{2j} (la probabilité qu'un évènement s_{1i} apparaissent au même moment qu'un évènement s_{2j}) comme $p_{i,j} = P\left(S_1 = s_{1i} \cap S_2 = s_{2\,j}\right)$

On appelle l'*entropie conjointe* de S_1 et S_2 la quantité suivante :

$$H(S_1, S_2) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p_{i,j} \times \log_2\left(p_{i,j}\right)$$

Si S_1 et S_2 sont rigoureusement indépendantes alors $H(S_1, S_2) = H(S_1) + H(S_2)$

Entropie conditionnelle

Soient $S_1(S_1, P_1)$ et $S_2(S_2, P_2)$ deux sources sans mémoire dont les évènements d'une source ne sont pas forcément indépendants de l'autre source.
On note $S_1 = \{s_{11}, \ldots, s_{1n}\}$, $P_1 = \{p_1, \ldots, P_n\}$ et $S_2 = \{s_{21}, \ldots, s_{1m}\}$, $P_2 = \{p_2, \ldots, P_m\}$.
On note la probabilité d'occurrence conditionnelle de s_{1i} et s_{2j} (la probabilité d'un évènement s_{1i} par rapport à un évènement s_{2j}) comme $p_{i|j} = P\left(S_1 = s_{1i} \mid S_2 = s_{2\,j}\right)$

On appelle l'*entropie conditionnelle* de S_1 relativement à une valeur S_2 la quantité représentant le désordre d'une source par rapport à un événement d'une autre source :

$$H(S_1 \mid S_2 = s_{2j}) = -\sum_{i=1}^{n} p_{i|j} \times \log_2\left(p_{i|j}\right)$$

Par extension on appelle entropie conditionnelle de S_1 par rapport à S_2 la quantité :

$$H(S_1 \mid S_2) = \sum_{j=1}^{m} p_j \times H(S_1 \mid S_2 = s_{2j}) = \sum_{i,j} p_{i,j} \times \log_2\left(\frac{p_j}{p_{i,j}}\right)$$

Cette notion est très importante en cryptographie, **il est nécessaire que le message crypté ait une entropie forte** pour éviter que l'on puisse déduire des éléments de l'algorithme de cryptage en exploitant les éventuelles formes d'organisations dans le message.

De même, l'entropie doit rester forte si l'une des informations dépendantes (le message, l'algorithme, le code et la clé) est connue.
En effet, même la connaissance du message clair et du message crypté ne doivent donner d'indices sur la clé employée. Aussi, nous avons les relations suivantes à respecter :

$$H(M) = H(M \mid C)$$

(l'égalité n'étant vérifiée que si M et C sont indépendants, on parle alors de *chiffrement parfait*)

$$H(M, C) = H(C) + H(M \mid C)$$

c) Extension de source

L'entropie seule ne suffit pourtant pas à déterminer la quantité de désordre d'un message.
En effet *H("ABCDEFABCDEF") = H("BAFEDBCACDFE")* alors que l'on peut constater une forme d'organisation triviale pour le premier message.
On utilise alors les *extensions de source*.

Définition : Soit *S(S, P)* une source sans mémoire. La k^{eme} extension S^k de *S* est le doublet (S^k, P^k) où S^k est l'ensemble des chaînes de longueur *k* sur l'alphabet *S* et où P^k est la distribution de probabilité définie par :

Pour un mot $m = s_{i1}...s_{ik} \in S^k$
Alors $P^k(m) = P\left(s_{i1}...s_{ik}\right) = p_{i1} \times ... \times p_{ik}$

Exemple : $S = \{a, b\}$ et $P = \left\{\frac{1}{3}, \frac{1}{4}\right\}$

$S^2 = \{aa, ab, ba, bb\}$ et $P^2 = \left\{\frac{1}{16}, \frac{3}{16}, \frac{3}{16}, \frac{9}{16}\right\}$

Définition : Soit *S(S, P)* une source markovienne. La k^{eme} extension S^k de *S* est le doublet (S^k, P^k) où S^k est l'ensemble des chaînes de longueur *k* sur l'alphabet *S* et où P^k est la distribution de probabilité définie par :

Pour un mot $m = s_{i1}...s_{ik} \in S^k$
Alors $P^k(m) = p_{i_1} \times p_{i_2\ i_1} \times ... \times p_{i_k\ i_{k-1}}$ où $p_{i_k\ i_{k-1}}$ est la probabilité d'occurrence de s_{ik} par rapport à son prédécesseur dans la chaîne.

Définition : Soient *M* un message de taille *n* et S_{M^k} la source dont les probabilités correspondent aux occurrences des *k-uplets* consécutifs de *M* alors :

$$H\left(S_{M^k}\right) \leq \log_2 \left\lceil \frac{n}{k} \right\rceil$$ [1]

Ainsi, dans notre exemple cité plus haut ("ABCDEFABCDEF"), si l'on regroupe les éléments de l'alphabet en doublets, $S = \{AB, CD, EF\}$ et $P = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$, l'entropie, qui était de 2.585 pour $S = \{A, B, C, D, E, F\}$ chute à 1,585.

[1] $\lceil x \rceil$ représente la partie entière supérieure de *x*.

III. Cryptographie à clé secrète

1. Définition

Le système de cryptographie par clé secrète (on parle aussi de chiffrement symétrique) repose sur deux éléments, à savoir une clé que l'on notera K et un algorithme de chiffrement.
Ces deux éléments doivent êtres connus des personnes souhaitant s'échanger des messages crypté et uniquement par ceux-ci.
L'algorithme de chiffrement effectuera un chiffrement/déchiffrement grâce à la clé secrète, ainsi une personne extérieure devra, pour lire le message, se procurer la clé secrète.

On peut comparer ce système à un coffre fort dont la clé et les doubles de cette clé sont détenus par les seules personnes autorisées à y accéder. Une personne ne possédant pas la clé devra soit forcer le coffre soit se procurer la clé.

Ce système a l'avantage d'être peu gourmand en termes de calcul, mais impose un secret absolu au niveau de la clé secrète.

2. Chiffrement de Vernam

Le chiffrement de Vernam a été inventé par Gilbert Vernam en 1917. Il s'agit d'un chiffrement d'une simplicité extrême et exploitant une clé *K*. L'algorithme travaille dans l'ensemble des valeurs binaires $\{0,1\}$ tel que :

$C = M \oplus K$ et $M = C \oplus K$, où $\oplus$ représente le XOR logique.

Pour être efficace, la clé *K* doit obéir à certaines contraintes :

- $|M| = |K|$
- Les éléments composant la clé *K* doivent être choisis de façon totalement aléatoire.
- La clé K doit être changée à chaque nouvel échange, ce pour deux raisons :
    - Si $C_1 = M_1 \oplus K$ et $C_2 = M_2 \oplus K$ alors $C_1 \oplus C_2 = M_1 \oplus M_2$ donc des parties de M_1 et M_2 pourraient être décodés.
    - Au cas où la clé serait décodée.

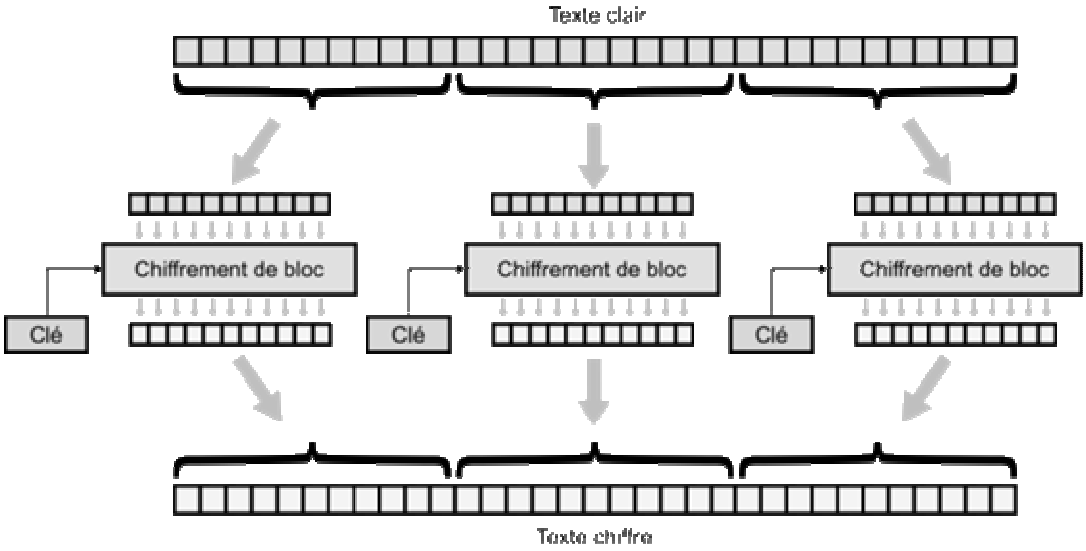
En bref, comme la clé est générée aléatoirement, le résultat de l'opération XOR n'est qu'une image de chaque élément de la clé décalée d'un rang inconnu, ceci étant encore plus flagrant lorsque les bits du message ont une signification décimale.
Ainsi, comme le code C est une suite aléatoire d'éléments, ou plus exactement une chaîne sans mémoire, son entropie est maximale et la cryptanalyse devient impossible.

De par cette propriété, le chiffrement de Vernam est un chiffrement parfait (démontré par Claude Shannon) le seul qui n'ait jamais été crée dans l'histoire de la cryptologie (démonstration de $H(M \mid C) = H(M)$ fournie en annexe).

3. Chiffrement par blocs

Le chiffrement symétrique par blocs consiste en un découpage du message de base *M* de *n* bits de long, en *s* blocs de *r* bits, ou $r = \left\lceil \frac{n}{s} \right\rceil$. La taille de *r* étant fixe, on ajuste la taille du message de base en ajoutant des bits sans signification pour que *n* soit multiple de *r* (on retrouvera ce fonctionnement dans le hachage cryptographique).

Le chiffrement symétrique par blocs agit sur un bloc de *r* bits et produit un autre bloc crypté de *r* bits, assurant la bijectivité du code.
Cette fonction de chiffrement est couplée à un mode de chiffrement (ECB, CBC, CFB, etc… détails fournis en annexe) lors du chiffrement bloc par bloc.



4. Chiffrement DES (Data Encryption System)

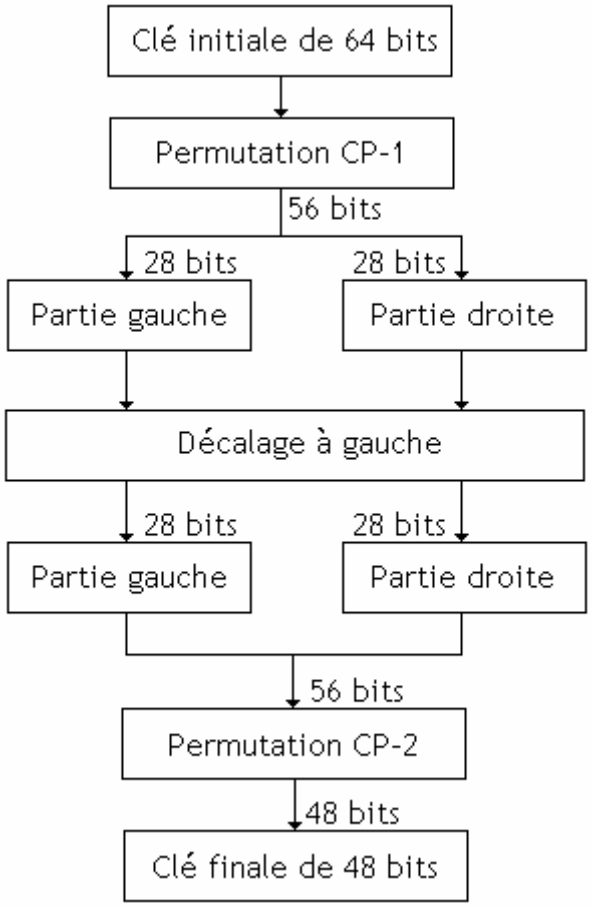
Le premier standard DES a été publié en 1977 à la demande du NBS (National Bureau of Standards) américain.

Il est basé sur un algorithme crée par IBM quelques années plus tôt : Lucifer. La sortie de DES suscita une certaine polémique, due à l'adaptation de Lucifer par la NSA (National Security Agency) qui fut soupçonnée d'avoir affaibli l'algorithme original pour pouvoir le casser plus facilement.

Etonnamment, DES se révéla plus résistant que Lucifier face à des attaques mises au point beaucoup plus tard.

a) Génération des clés

Etant donné que l'algorithme DES est public, toute la sécurité repose sur la complexité des clés de chiffrement.
Les clés du système DES sont des clés de 64 bits avec un bit de parité tout les sept bits, ce qui donne une clé effective de 56 bits, il peut exister 2^{56} (soit 7.2×1016) clés différentes. Les bits de parité servent au contrôle de parité.
Pour les nécessité de l'algorithme cette clé donne naissance a seize sous-clés de 48 bits chacune.
L'algorithme ci-dessous montre comment obtenir à partir d'une clé de 64 bits une clé diversifiée de 48 bits servant dans l'algorithme du DES :



Avant toute chose, les bits de parité de la clé sont éliminés afin d'obtenir une clé d'une longueur utile de 56 bits.

La première étape consiste en une permutation notée ***CP-1*** de la clé dépouillée des bits de parité, dont la matrice est présentée ci-dessous :

G_i						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

On note G_0 et D_0 le résultat de cette première permutation.

Ces deux blocs subissent ensuite une rotation à gauche d'un ou deux bits, de telles façons que les bits en seconde position prennent la première position, ceux en troisième position la seconde et ainsi de suite selon cet ordre(les bits en première position passent en dernière position).

Tour	Décalage	Décalage absolu	Tour	Décalage	Décalage absolu
1	1	1	9	1	15
2	1	2	10	2	17
3	2	4	11	2	19
4	2	6	12	2	21
5	2	8	13	2	23
6	2	10	14	2	25
7	2	12	15	2	27

8	2	14	16	1	28

Les 2 blocs de 28 bits sont ensuite regroupés en un bloc de 56 bits. Celui-ci passe par une permutation, notée ***CP-2***, fournissant en sortie un bloc de 48 bits, représentant la clé K_i .

CP-2											
14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

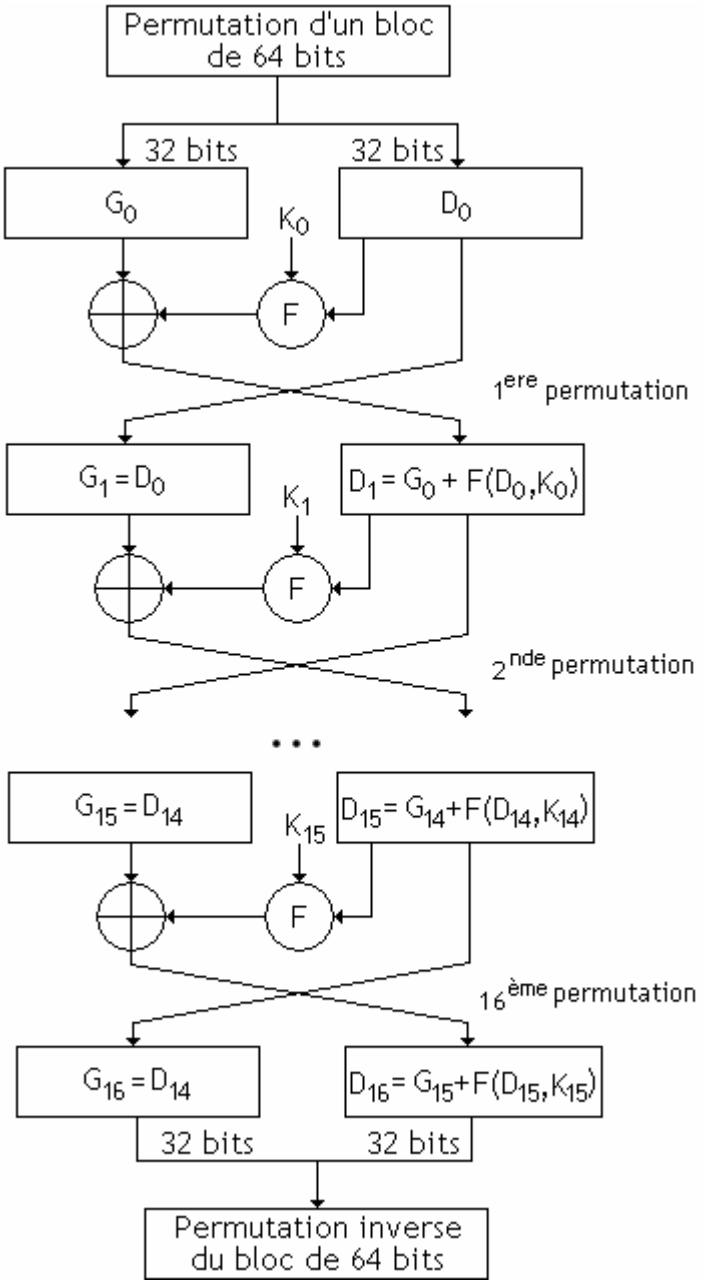
Des itérations de l'algorithme permettent de donner les 16 clés K_1 à K_{16} utilisées dans l'algorithme du DES.

b) Chiffrement

L'algorithme consiste à effectuer des combinaisons, des substitutions et des permutations entre le texte à chiffrer et la clé, en faisant en sorte que les opérations puissent se faire dans les deux sens (pour permettre un déchiffrement ultérieur). La combinaison entre substitutions et permutations est appelée code produit.

L'algorithme effectuera les opérations suivantes :

- Fractionnement du texte en blocs de 64 bits
- Permutation initiale des blocs
- Découpage des blocs en deux parties (ici nommées *G* et *D*)
- Etapes de permutation et de substitution répétées à seize reprises
- Recollement des parties *G* et *D* puis permutation initiale inverse



Le schéma ci-contre décrit le mode de chiffrement de l'algorithme DES, d'abord la permutation, puis la séparation des blocs et enfin l'échange des blocs, ainsi D_0 devient G_1 alors que D_1 est formé d'une opération XOR avec le bloc de donnée G_0 préalablement passé par la fonction *F*.

Cette fonction est a la base du chiffrement DES, et est décrite plus loin.

Permutation initiale

Dans un premier temps, chaque bit d'un bloc est soumis à la permutation initiale, pouvant être représentée par la matrice de permutation initiale (notée *PI*) suivante :

PI							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Cette matrice de permutation indique, en parcourant la matrice de gauche à droite et de haut en bas, que le 58ème bit du bloc de texte de 64 bits se retrouve en première position, le 50ème en seconde position et ainsi de suite.

Scindement en blocs *G* et *D*

Une fois la permutation initiale réalisée, le bloc de 64 bits est scindé en deux blocs de 32 bits, notés respectivement ***G*** et ***D*** (pour gauche et droite, la notation anglo-saxonne étant *L* et *R* pour *Left and Right*). On note G_0 et D_0 l'état initial de ces deux blocs :

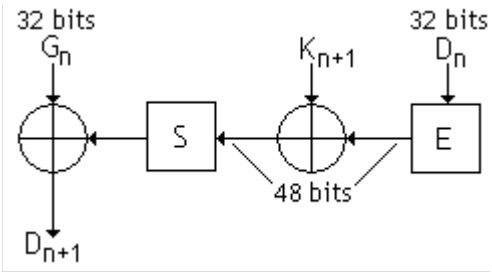
G_0							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8

D_0							
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Il est intéressant de remarquer que G_0 contient tous les bits possédant une position paire dans le message initial, tandis que D_0 contient les bits de position impaire.

c) Fonction de chiffrement

Les blocs G_n et D_n sont soumis à un ensemble de transformations itératives appelées *rondes* (ou *tours*), explicitées dans ce schéma.



Fonction d'expansion

Les 32 bits du bloc D_0 sont étendus à 48 bits grâce à une table (matrice) appelé *table d'expansion* (notée ***E***), dans laquelle les 48 bits sont mélangés et 16 d'entre eux sont dupliqués :

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Ainsi, le dernier bit de D_0 (c'est-à-dire le 7ème bit du bloc d'origine) devient le premier, le premier devient le second, ...

De plus, les bits 1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28 et 29 de D_0 (respectivement 57, 33, 25, 1, 59, 35, 27, 3, 61, 37, 29, 5, 63, 39, 31 et 7 du bloc d'origine) sont dupliqués et disséminés dans la matrice.

OU Exclusif avec la clé (XOR)

La matrice résultante de 48 bits est appelée D'_0 ou bien $E[D_0]$. L'algorithme DES procède ensuite à un *OU exclusif* entre la première clé K_1 et $E[D_0]$. Le résultat de ce *OU exclusif* est une matrice de 48 bits que nous appellerons D_0 par commodité (il ne s'agit pas du D_0 de départ!).

Fonction de substitution

D_0 est ensuite scindé en 8 blocs de 6 bits, noté D_{0i}. Chacun de ces blocs passe par des fonctions de sélection (appelées parfois *boîtes de substitution* ou *fonctions de compression*), notées généralement S_i.

Les premiers et derniers bits de chaque D_{0i} détermine (en binaire) la ligne de la

fonction de sélection, les autres bits (respectivement 2, 3, 4 et 5) déterminent la colonne. La sélection de la ligne se faisant sur deux bits, il y a 4 possibilités (0, 1, 2, 3). La sélection de la colonne se faisant sur 4 bits, il y a 16 possibilités (0 à 15). Grâce à cette information, la fonction de sélection "sélectionne" une valeur codée sur 4 bits.

Voici la première fonction de substitution :

S_1															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6

Soit D_{01} égal à *101110*. Les premiers et derniers bits donnent *10*, c'est-à-dire 2 en binaire. Les bits 2, 3, 4 et 5 donnent *0111*, soit 7 en binaire. Le résultat de la fonction de sélection est donc la valeur situé à la ligne n°2, dans la colonne n°7. Il s'agit de la valeur *11*, soit *111* en binaire.
Chacun des 8 blocs de 6 bits est passé dans la fonction de sélection correspondante, ce qui donne en sortie 8 valeurs de 4 bits chacune. Les autres fonctions de sélection sont fournies en annexes.

Chaque bloc de 6 bits est ainsi substitué en un bloc de 4 bits. Ces bits sont regroupés pour former un bloc de 32 bits.

Permutation
Le bloc de 32 bits obtenu est enfin soumis à une permutation ***P*** dont voici la table :

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

OU Exclusif avec la clé (XOR)
L'ensemble de ces résultats en sortie de ***P*** est soumis à un *OU Exclusif* avec le G_0 de départ (comme indiqué sur le premier schéma) pour donner D_1, tandis que le D_0 initial donne G_1.

Permutation inverse
A la fin des itérations, les deux blocs G_{16} et D_{16} sont recollés, puis soumis à la permutation initiale inverse :

P_{i-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

On obtient en sortie est un texte codé de 64 bits.

d) Conclusion sur DES

Utilisé pendant près de trente ans, notamment dans les systèmes bancaires, DES a eu a subir de nombreuses attaques mises au point ou adaptées pour diminuer la complexité de l'attaque en force brute.
La puissance des machines devenant de plus en plus grande, une étude fut réalisée en 1996 pour estimer les performances des attaques.

Attaquant	Budget (€)	Outil	Clef 56 bits
Hacker	300	Logiciel	38 ans
PME	3500	Circuit	18 mois
Grande entreprise	225 000	Circuit ASIC[3]	19 jours
Multinationale	7 500 000	ASIC	6 minutes
Gouvernement	225 000 000	ASIC	12 secondes

Par la suite, des machines spécifiquement conçues pour attaquer l'algorithme ont été construite. Par exemple, Deep Crack en 1998 pouvait casser un code DES en moins d'une semaine.
En 1999 un projet de calcul distribué utilisant des ordinateurs de particuliers a permis de casser un code DES en 22h30.

Désormais, DES n'est plus utilisé que dans d'anciennes applications. Il a été abandonné au profit d'algorithme plus robuste, comme son successeur AES.

5. Chiffrement AES (Advanced Encryption Standard)

AES est issu d'un appel à candidatures international organisé par le NIST (National Institute of Standards and Technology) en 2000. Concurrent avec quatre autres algorithmes (MARS, RC6, Serpent, Twofish) retenus pour une étude plus poussée, AES (originellement Rijndael, du nom de ses deux concepteurs Joan Daemen et Vincent Rijmen), fut retenu par le NIST pour devenir le nouveau standard américain du chiffrement, l'algorithme étant à la fois plus résistant et plus rapide que ces concurrents.

Il s'agit d'un algorithme de cryptage à clé secrète dont la taille des blocs et de la clé est multiple de 32 (clé et blocs compris entre 128 et 256 bits), et tout comme DES, il se base sur un ensemble d'étapes réalisées plusieurs fois successivement.
AES est un sous ensemble de l'algorithme Rijndael, qui utilise quand a lui des blocs de 128 bits de données à crypter seulement et est devenu le nouveau standard américain du chiffrement.

a) Présentation

Pour AES les blocs de données en entrée et en sortie sont des blocs de 128 bits, c'est à dire de 16 octets.
Les clés secrètes ont au choix suivant la version du système : 128 bits (16 octets), 192 bits (24 octets) ou 256 bits (32 octets).
On découpe les données et les clés en octets et on les place dans des tableaux.

[3] Application Specific Integrated Circuit : Circuit regroupant un ensemble de fonctionnalités sur mesure et optimisé pour les effectuer beaucoup plus rapidement qu'un équivalent matériel plus générique ou logiciel. Exemple : Les processeurs graphiques, destinés à réaliser des opérations 3D courantes beaucoup plus rapidement que le processeur de la machine.

Les données comportent t_d = 16 octets ($P_0, P_1, \cdots, P_{15}$) qui sont classés dans un tableau de quatre lignes et quatre colonnes. Le tableau est rempli colonnes par colonnes.
De même la clé est découpée en octets de taille t_k = 16, t_k = 24 ou t_k = 32 octets selon la version d'AES utilisée ($k_0, k_1, \cdots, k_{tk-1}$). Ces octets sont aussi classés dans un tableau de quatre lignes et N_k colonnes dépendant de la taille de la clé (N_k = 4, N_k = 6 ou N_k = 8).

P_0	P_4	P_8	P_{12}
P_1	P_5	P_9	P_{13}
P_2	P_6	P_{10}	P_{14}
P_3	P_7	P_{11}	P_{15}

K_0	K_4	K_8	K_{12}	K_{16}	K_{20}
K_1	K_5	K_9	K_{13}	K_{17}	K_{21}
K_2	K_6	K_{10}	K_{14}	K_{18}	K_{22}
K_3	K_7	K_{11}	K_{15}	K_{19}	K_{23}

Données et clés (cas N_k = 6)

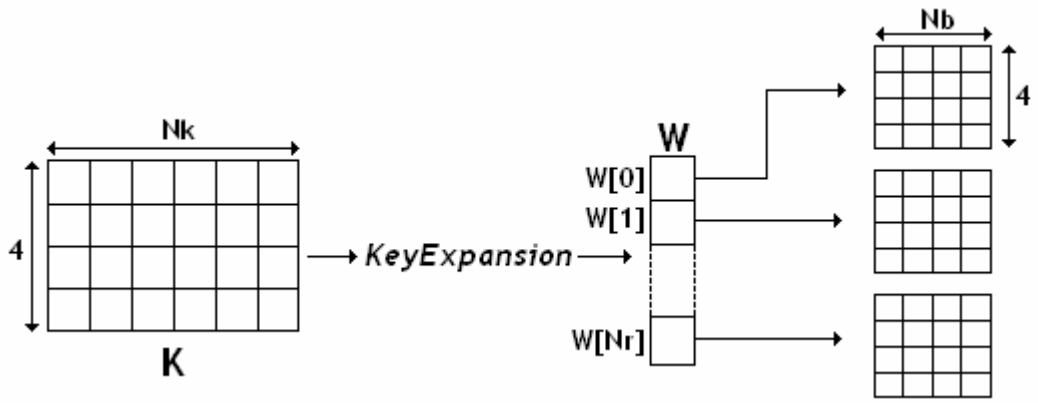
b) Génération des clés

Comme DES, AES exécute une séquence de tours qui seront détaillés par la suite. On note N_r le nombre de tours qui doivent être effectués. Ce nombre dépend des valeurs de N_b et de N_k, ou N_b et N_k représente respectivement le nombre de colonnes dans une matrice pour écrire un bloc d'octets à crypter et le nombre de colonnes pour une clé. Les différentes configurations possibles sont détaillées dans le tableau suivant :

	N_k	N_b	N_r
AES - 128	4	4	10
AES - 192	6	4	12
AES - 256	8	4	14

À partir de la clé initiale K, le système crée donc $N_r + 1$ clés de tour ayant chacune $4 * N_b$ octets.
AES n'utilisant que des blocs à chiffrer de 128 bits, N_b aura toujours la valeur 4 ($4*4$ = 16 octets soit 128 bits).

La clé de chiffrement K est stockée dans un tableau de quatre lignes et N_k colonnes et est par la suite étendue en un tableau W de quatre lignes et de $N_b * (N_r + 1)$ colonnes. On peut voir ce tableau comme un tableau de $N_r + 1$ cases contenant chacune un autre tableau de quatre lignes et N_b colonnes.



Selon la valeur de N_k ($N_k \leq 6$ ou $N_k > 6$), l'algorithme change légèrement, mais dans tous les cas, les N_k premières colonnes de K sont recopiées sans modification des les N_k

premières colonnes de *W* et les colonnes suivantes sont définies de manières récursives par rapport aux colonnes précédentes.
La fonction *KeyExpansion* utilise deux fonctions et un tableau de constantes, à savoir :

- La fonction *SubWord* qui est une fonction qui prend en paramètre un mot de quatre octet et qui applique la boîte-S *SBox*[4] sur chacun des octets du mot.

- La fonction *RotWord* qui prend en paramètre un mot de quatre octets (a = [a0, a1, a2, a3]) et effectue une permutation circulaire qui renvoie le mot [a1, a2, a3, a0].

- Le tableau de constantes de tours *Rcon[i]*, indépendant de N_k défini tel que : *Rcon[i]* = [x^{i-1}, 00, 00, 00] $\forall i \geq 1$

Paramètres en entrée : Une clé K de 4 $*$ N_k octets.
Valeur en sortie : Une clé étendue W de 4 $*$ N_b $*$ (N_r + 1) octets

Pour *i* = 0 à N_k – 1 **Faire**
    W[*i*] = K[*i*]
Fin Pour

Pour *i* = N_k à N_b $*$ (N_r + 1) – 1 **Faire**
    Tmp = W[*i* - 1]

    Si *i* modulo N_k = 0 **Alors**
        Tmp = *SubWord*(*RotWord*(Tmp)) + *Rcon[i/*N_k*]*
    Sinon, Si (N_k > 6) et (*i* modulo N_k = 4) **Alors**
        Tmp = *SubWord*(Tmp)
    Fin Si

        W[*i*] = W[*i* - N_k] + Tmp
Fin Pour

Algorithme de diversification des clés AES

c) Chiffrement

Le système AES effectue pour son chiffrement, plusieurs tours (10, 12 ou 14 selon la version d'AES) d'une même composition de transformations. L'algorithme prend en entrée un tableau *State* correspondant au texte en clair, une clé K de 128, 192 ou 256 bits selon la version d'AES et renvoie en sortie le tableau *State* chiffré.

[4] Une boîte-S (ou S-Box, pour *Substitution Box*) est une table de substitution utilisée dans un algorithme de chiffrement symétrique. Une boîte-S contribue donc à augmenter la "confusion" (terme employé par Claude Shannon) du message codé.

Paramètres en entrée : Un tableau *State* (texte clair), une clé K.
Valeur en sortie : Un tableau *State* chiffré

KeyExpansion(K, *RoundKeys*)
AddRoundKey(*State*, *RoundKeys[0]*) // Addition initiale

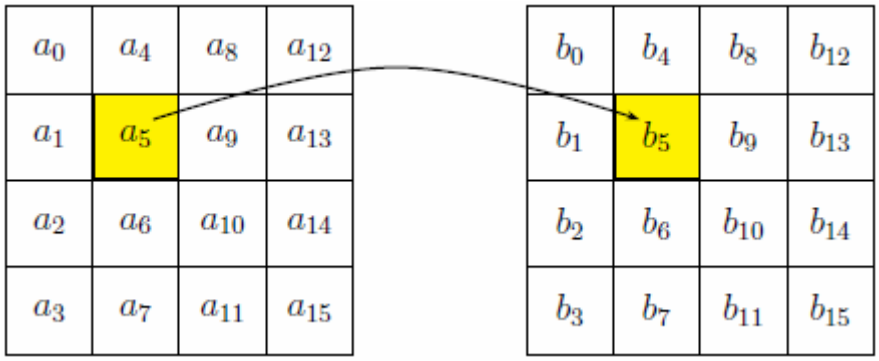
Pour r = 1 à N_r – 1 **Faire**
Subbytes(State)
ShiftRows(State)
MixColumns(State)
AddRoundKey(State, RoundKeys[r])
Fin Pour

// Tour final
Subbytes(State)
ShiftRows(State)
AddRoundKey(State, RoundKeys[N_r])

Algorithme de chiffrement AES

AES travaille sur des blocs vus comme des matrices de $4 * N_b$ éléments, défini sur un corps de 2^8 soit $\mathbb{F}_{256}$.
Le chiffrement AES consiste en une addition initiale de clef *(AddRoundKey)* suivie de N_r - *1* tours de quatre étapes chacun, à savoir *SubBytes*, *ShiftRows*, *MixColumns* et *AddRoundkey*.

Fonction SubBytes (State)
La fonction *SubBytes* est la seule transformation linéaire de l'algorithme. Elle prend chaque élement de la matrice *State* et effectue une substitution inversible (*SBox*).



La table *SBox* dérive de la fonction inverse $t : a \rightarrow a^{-1}$ sur $\mathbb{F}_{256}$.
Cette fonction est reconnue pour ses propriétés de non-linéarité.
La boite-S est construite en combinant cette fonction inverse avec une transformation affine inversible f. De plus les concepteurs de la *SBox* on fait en sorte que la boite-S n'admette ni point fixe, ni point fixe opposé.

$$SBox[a] = f(t(a)) \text{ pour tout } a \in \mathbb{F}_{256}$$
$$SBox[a] + a \neq 00 \text{ pour tout } a \in \mathbb{F}_{256}$$
$$SBox[a] + a \neq FF \text{ pour tout } a \in \mathbb{F}_{256}$$

La fonction est définie par :

$$b = f(a) \Leftrightarrow \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,0\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,1\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \\ 1\,1\,1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,1\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1\,1\,1 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

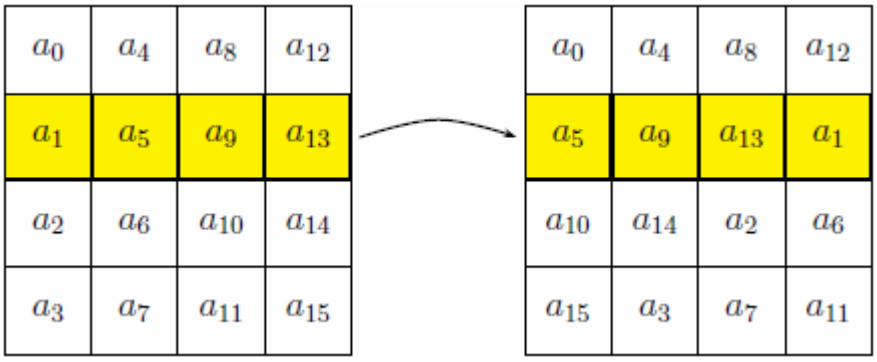
Fonction ShiftRows (State)

La procédure *ShiftRows* consiste à opérer une rotation à gauche de *n* éléments sur chaque ligne du tableau d'entrée.

Le nombre d'octet décalés dépend de la ligne considérée. La ligne *i* est décalée de C_i éléments et l'élément en position *j* de la ligne *i* est déplacé en position (*j* - C_i) Modulo N_b.

Pour AES la taille des blocs à chiffrer est de $N_b = 4$, ce qui implique que le nombre de cases dont on décale la ligne $i(0 \le i \le 3)$ est de *i*.

Ainsi, pour la première ligne il n'y aura pas de décalage (*i* = 0), à la seconde, il y aura un octet de décalage (*i* = 1) et ainsi de suite.



Fonction MixColumns (State)

La fonction *MixColumns* consiste à effectuer pour chaque colonne une multiplication par un polynôme fixe :

$$a(X) = 03.X^3 + 01.X^2 + 01.X + 02 \text{ Modulo } 1 + X^4$$

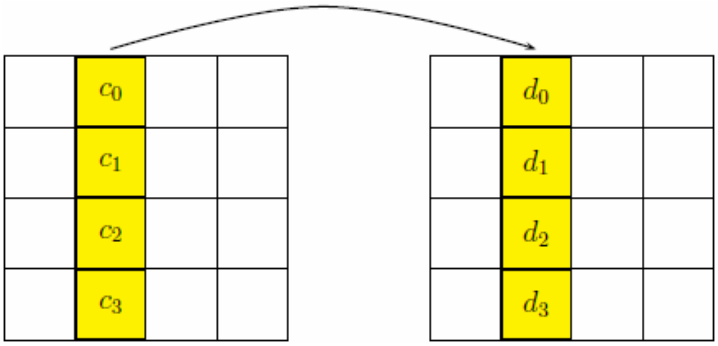
Note : Les coefficients des polynômes sont des éléments de $\mathbb{F}_{256}$ notés comme des nombres hexadécimaux.

On réalise donc matriciellement l'équation suivante :

$$(03.X^3 + 01.X^2 + 01.X + 02) \times b(x) \text{ Modulo } (1 + X^4)$$

$$c(X) = a(X) \times b(X) \mod (X^4 + 1) \Leftrightarrow \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Les colonnes qui passent par la fonction *MixColumns* sont donc transformées comme ceci :



De plus $a(X)$ et $X^4 + 1$ sont premiers entre eux $a(X)$ est inversible modulo $(1 + X^4)$ et la transformation effectuée par *MixColumns* est également inversible ce qui permet le décryptage.

Fonction AddRoundKey (State, K_i)
La procédure *AddRoundKey* est très simple. Elle consiste à faire un ou exclusif bit a bit entre les 128 bits de l'état *State* et les 128 bits de la clé de tour K_i.

$$State = State \oplus K_i$$

d) Conclusion sur AES

AES a été conçu pour résister à un grand nombre d'attaques connues.
Les différentes fonctions ont chacune un rôle important dans la sécurité et ont été imaginée dans ce sens.
Par exemple, la fonction *ShiftRows* augmente l'entropie du message en séparant les octets initialement consécutifs. De même, la fonction *MixColumn*, qui est aussi employé dans des algorithmes de détection et de correction d'erreur, permet à chaque bit de sortie de dépendre directement des bits d'entrées une fois l'ensemble des tours effectués. Ainsi, une modification minime en entrée entraîne des modifications majeures en sortie (une notion qu'on retrouvera dans le hachage cryptographique).

Actuellement, il n'existe aucune attaque efficace contre AES, mais il convient d'être prudent car l'algorithme est assez récent. Trop récent, en tout cas, pour que la cryptanalyse mondiale puisse dévoiler au grand jour ses failles éventuelles.

IV. Cryptographie à clé publique

1. Définition

Pour illustrer le concept de cryptographie à clé publique, nous allons prendre pour exemple une boîte aux lettres.

Bob possède une boîte aux lettres dans laquelle n'importe qui peut poster des messages. Toutefois, seul Bob peut ouvrir sa boîte et lire les messages.

En d'autres termes, Bob possède une clé connue de tous (clé publique) qui sert à crypter un message. Il possède aussi une clé connue de lui seul (clé privée), servant exclusivement à décrypter les messages encryptés par la clé publique.
Le principe étant qu'il est impossible, ou plutôt très difficile, de déduire la clé privée à partir de la clé publique.

Les algorithmes de création des clés se basent sur des fonctions dîtes "à sens unique". Ces fonctions sont conçues de telles sorte que :

- L'obtention de l'image de x à partir de x soit très simple.
- La recherche de x à partir de l'image de x est un problème mathématique réputé difficile, ou informatiquement trop complexe (= trop lent).

2. Chiffrement RSA

Le chiffrement RSA, mis au point en 1978 par les mathématiciens Rivest, Shamir et Adleman, nécessite un certain nombre de pré-requis en plus de ceux évoqués en début du document.

a) Nombres de Bézout

Soient *a* et *b* deux éléments d'un anneau euclidien et *d* leur pgcd. Il existe deux éléments *x* et *y*, appelés *nombres de Bézout* tels que $\upsilon(x) \leq \upsilon(b)$ et $\upsilon(y) \leq \upsilon(a)$ et vérifiant l'égalité de Bézout $ax + by = d$

Remarque : Si $d = 1$, *a* et *b* sont premiers entre eux.

b) Algorithme d'Euclide et d'Euclide étendu

L'algorithme d'Euclide permet de calculer le pgcd de deux entiers positifs. Très simple, il se présente sous la forme récursive suivante :

```
r := a div b (a et b étant les deux nombres dont le pgcd doit être calculé)
Si r = 0
        Retourner b
Sinon
        calculer pgcd (b, r) //Appel récursif
Fin Si
```

L'algorithme d'Euclide étendu introduit, en plus du calcul du pgcd, le calcul des nombres de Bézout et se présente sous la forme récursive suivante :

```
q := a div b
```

```
r := a mod b
Si r = 0
        u = 0
        v = 0
        Retourner b, u, v
Fin Si

d, u, v := calculer pgcd(b, r, u, v)
u := v
v := u - q × v

Retourner d, u, v
```

Une démonstration de l'algorithme en fonctionnement est fournie en annexe.

c) Indicatrice d'Euler

Soit $n \geq 2$ un entier. On note $\mathbb{Z}/n\mathbb{Z}^*$ l'ensemble des entiers strictement positifs, inférieurs à n et premiers avec n :

$$\mathbb{Z}/n\mathbb{Z}^* = \{x \in \mathbb{N} \ / \ 1 \leq x < n \text{ et } p\gcd(x,n) = 1\}$$

Le cardinal de $\mathbb{Z}/n\mathbb{Z}^*$ est noté $\varphi(n)$, la fonction φ étant nommée *indicatrice d'Euler*.

Propriété : Si n est un nombre premier, $\varphi(n) = n - 1$.

Propriété : Dans $\mathbb{Z}/n\mathbb{Z}^*$, tout élément à un inverse : en effet, comme tout élément est premier avec n, l'identité de Bézout assure l'existence de deux entiers de signes opposés, u et v ($|u| \leq n$ et $|v| \leq x$), tels que $u.x + v.n = 1$.
On a alors $u.x \equiv 1 \pmod{n}$ et $u \equiv x^{-1} \pmod{n}$. On appelle *u l'inverse de x modulo n* et celui-ci se calcule grâce à l'algorithme d'Euclide étendu vu plus haut.

Théorème (Euler) : Soit a un élément quelconque de $\mathbb{Z}/n\mathbb{Z}^*$. On a :

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

d) Théorème de Fermat

Note : Ce théorème découle directement du précédent.

Soit p un nombre premier.
$\forall a \in \mathbb{Z}/p\mathbb{Z} \ \Rightarrow a^p \equiv a \pmod{p}$

e) Théorème RSA

Soit $n = pq$, p et q premiers et $a \in \mathbb{Z}/n\mathbb{Z}$. On a :

$$\forall k \in \mathbb{N}, \ a^{k\varphi(n)+1} \equiv a \pmod{n}$$

f) Mise en œuvre

Création des clés

- Prendre $n = pq$, où p et q sont deux nombres premiers (en pratique, il est conseillé d'employer des nombres d'au moins 150 chiffres décimaux).
- Prendre un entier e premier avec $\varphi(n) = (p-1)(q-1)$
- Comme e est premier avec $\varphi(n)$, il existe un entier d inverse de e ($ed \equiv 1 \ [\mathrm{mod} n]$).
- La clé publique est alors le couple *(n, e)* et la clé privée le couple *(n, d)*

Chiffrement

Si M est un entier appartenant à l'ensemble $[\![0,\ldots,n-1]\!]$ et représentant le message alors $C \equiv M^e \ [\mathrm{mod} n]$.

Déchiffrement

Si C est un entier représentant le message codé alors $M \equiv C^d \ [\mathrm{mod} n]$.

Un exemple de mise en œuvre est disponible en annexe.

g) Génération de nombres premiers

Test de primalité Miller-Rabin

Soit n un nombre impair et soient s et t tels que $n-1 = t \times 2^s$ avec t impair.
$\forall a < n$, on a :

$$a^{(n-1)} - 1 = a^{t2^s}$$
$$= \left(a^t - 1\right)\left(a^t + 1\right)\left(a^{2t} + 1\right)\ldots\left(a^{(2^{s-1})t} + 1\right)$$
$$= \left(a^t - 1\right)\sum_{i=0}^{s-1}\left(a^{t2^i} + 1\right)$$

Si n est premier alors $a^{(n-1)} - 1 \equiv 0 \ [\mathrm{mod} n]$ (d'après le théorème de Fermat) ce qui sous entend que :

- soit $a^t - 1 \equiv 0 \ [\mathrm{mod} n]$
- soit $\sum_{i=0}^{s-1}\left(a^{t2^i} + 1\right) \equiv 0 \ [\mathrm{mod} n]$

On dit qu'un nombre n réussit le test de Miller-Rabin si :
$a^t - 1 \neq 0 \ [\mathrm{mod} n]$
$\sum_{i=0}^{s-1}\left(a^{t2^i} + 1\right) \neq 0 \ [\mathrm{mod} n]$

Si n est impair et non premier moins de $\frac{n-1}{4}$ nombres a échouent au test de Miller-Rabin. De même, si $a \in [\![1,\ldots,n-1]\!]$, la probabilité devient inférieure à $\frac{1}{4}$.

Ainsi, l'algorithme suivant permet de rechercher un nombre probablement premier :

- On tire au hasard un nombre impair n.
- On tire au hasard *k* nombres a_i distincts tel que $1 < a_i < n$.
- On applique le test de Miller-Rabin pour chaque a_i.
- Si aucun a_i ne réussit le test de composition, on en déduit que *n* est premier. La probabilité d'erreur est inférieure à 4^{-k}.
- Sinon on recommence avec *n* + 2 jusqu'à trouver un nombre premier.

Dans le cas de RSA, si une erreur survient au niveau du test de primalité, cela ne posera pas de problème. En effet, le destinataire se rendra vite compte que *p* et *q* ne sont pas premiers : soit la clé *d* n'est pas inversible, soit certains blocs du message crypté seront incompréhensibles. On procédera alors à un changement de système RSA, ce qui consiste à recalculer *p* et *q*.

h) Conclusion sur le chiffrement RSA

La robustesse de RSA s'appuie donc sur la difficulté de factoriser de grands entiers. En effet, si l'on peut factoriser n = pq depuis la clé publique, il devient facile de trouver $\varphi(n) = (p-1)(q-1)$ et d'en déduire la clé privée.

Après 20 ans de recherches, aucune autre méthode efficace que la factorisation de n n'a été publié pour casser RSA.
Les limites actuelles de factorisation concernent des nombres de 200 chiffres environ (record 2005).

L'utilisation de RSA appliquée à des systèmes informatiques nécessite un certain nombre de précautions d'emploi (taille des clés, etc…) décrites dans la norme *PKCS#1* (*RFC3447*) disponible sur internet.

3. Chiffrement El Gamal

Le chiffrement El Gamal a été inventé en 1985 par le mathématicien égyptien Taher El Gamal.
Tout comme son homologue RSA, cet algorithme se base sur une fonction à sens unique nommée exponentiation modulaire.

a) Exponentiation (ou puissance) modulaire

Soit *a* un élément de $\mathbb{Z}/n\mathbb{Z}^*$, l'exponentiation modulaire est le morphisme défini par :

$$\begin{array}{ccc} \mathbb{Z}/n\mathbb{Z} & \mapsto & \mathbb{Z}/n\mathbb{Z} \\ b & \rightarrow & a^b \pmod{n} \end{array}$$

Algorithme de calcul

Si b = 0
    Retourner 1
Sinon

d := recalculer puissance modulaire $\left(a^{\left\lfloor \frac{b}{2} \right\rfloor}, \left\lfloor \frac{b}{2} \right\rfloor, n\right)$

d := (d × d) mod n
Si b est impair
d = (d × *a*) mod n
Fin Si
Retourner d
Fin Si

Cet algorithme se base sur une décomposition de *b* en carrés successifs pour diminuer le nombre d'opérations au niveau informatique.

b) Logarithme discret

Rappel : Un générateur du groupe multiplicatif $\mathbb{Z}/n\mathbb{Z}^*$ est un nombre *g* tel que $\left\{g^i, i \in \mathbb{N}\right\} = \mathbb{Z}/n\mathbb{Z}^*$.

Soit *a* un élément de $\mathbb{Z}/n\mathbb{Z}^*$.
Si *a* est un générateur du groupe multiplicatif $\mathbb{Z}/n\mathbb{Z}^*$ alors la fonction d'exponentiation est associée à sa fonction de décodage.

Pour *c* dans $\mathbb{Z}/n\mathbb{Z}$, le plus petit entier positif *b* tel que $a^b = c \pmod{n}$ est appelé *logarithme discret* (ou *index*) *en base a de b modulo n*.

$$\begin{array}{ccc} \mathbb{Z}/n\mathbb{Z} & \mapsto & \mathbb{Z}/n\mathbb{Z} \\ c & \rightarrow & \log_a c \pmod{n} \end{array}$$

c) Mise en oeuvre

Création des clés

- Soit *p* un nombre premier suffisamment grand pour que le problème du logarithme discret soit difficile dans $\mathbb{Z}/p\mathbb{Z}^*$
- Soit $\mathbb{Z}/p\mathbb{Z}^*$ une racine primitive (= générateur).
- Soit *s* un nombre et $\beta = g^s \pmod{n}$
- La clé publique est alors le triplet (p, g, β) et la clé privée correspond à *s*.

Chiffrement

Soit *M* une valeur représentant le message à chiffrer et soit $k \in \mathbb{Z}/(p-1)\mathbb{Z}$ un nombre aléatoire secret.

$$C = (y_1, y_2) \text{ avec } \begin{cases} y_1 \equiv g^k \pmod{p} \\ y_2 \equiv M \times \beta^k \pmod{p} \end{cases}$$

Déchiffrement

Pour $C = (y_1, y_2) \in \mathbb{Z}/p\mathbb{Z}^*$ on définit :

$M = y_2 \times \left(y_1^{\,s}\right)^{-1}$ où $\left(y_1^{\,s}\right)^{-1} = y_1^{(p-1)-s}$

Un exemple de mise en œuvre est disponible en annexe.

d) Conclusion sur le chiffrement El Gamal

Le problème du logarithme discret (noté DLP pour *Discrete Logarithm Problem*) est le calcul inverse de la puissance modulaire.
Si la puissance modulaire est calculable en un temps raisonnable comme nous avons pu le voir pour l'algorithme présenté plus haut, ce n'est pas le cas du logarithme discret.

En effet, la résolution du problème du logarithme discret se base sur le théorème suivant :

Si g est un générateur de $\mathbb{Z}/n\mathbb{Z}^*$, alors $\forall x, y \in \mathbb{N} : g^x \equiv g^y \pmod{n}$ si et seulement si $x = y \pmod{\varphi(n)}$.

Toutefois, étant donné y, il est difficile de calculer x tel que $g^x = y$. La seule méthode simple consiste en l'énumération exhaustive de tous les x possibles. Ainsi, si $n = 10^{150}$, l'énumération va demander 10^{150} opérations, ce qui est absolument impossible en temps raisonnable à l'heure actuelle.

Ainsi, l'exponentiation modulaire et son opération inverse, le logarithme discret, sont une fonction à sens unique, pierre d'achoppement du chiffrement El Gamal.

V. Hachage cryptographique

1. Avant propos

a) Définition

Une fonction de hachage est une application $H : \{0,1\}^* \rightarrow \{0,1\}^n$ qui transforme une chaîne de taille quelconque d'un ensemble infini en une chaîne de taille n d'un ensemble fini. Une fonction de hachage ne nécessite pas de fonction H^{-1} car son but n'est pas de pouvoir retrouver le message original.
Cependant, dans le cas du hachage cryptographique, tout comme dans le chiffrement à clé publique, il est important que la fonction H^{-1} ne soit pas applicable en temps humain.

Le résultat d'une fonction de hachage est une empreinte du message original, elle peut-être utilisée pour corriger des erreurs (en recalculant l'empreinte à partir du message reçu et en vérifiant si elle n'a pas changée, on parlera alors de somme de contrôle), pour compresser des données dites discrètes (supportant les pertes : son, image, vidéo, etc…) et pour identifier un message.

Très utilisées dans les générateurs pseudo-aléatoires, dans les codes détecteurs d'erreur et dans les mécanismes d'authentification par mot de passe[5] sans stocker ce dernier, nous traiterons ici de certaines d'entre elles.

b) Vocabulaire

Empreinte : Résultat d'une fonction de hachage. Est parfois noté *condensat* ou *hash*.

Antécédent (ou préimage) : Si *y* est tel que $y = H(x)$, alors x est appelé antécédent de y (*y* étant l'empreinte de *x*).

Collision : On parle de collision entre les antécédents *x* et *x'* lorsque : $\begin{cases} x \neq x' \\ H(x) = H(x') \end{cases}$
Sachant que la taille de l'entrée d'une fonction de hachage peut avoir une taille quelconque (notamment une taille supérieure à *n*), les collisions sont inévitables mais doivent être minimisées.

Salage : Opération qui consiste à rajouter de l'information (sel) a une donnée hachée dans le but d'en compliquer la détermination.

2. Fonctionnement

a) Propriétés

Les fonctions de hachage ont une propriété naturelle de compression.

Lorsqu'elles sont employées pour de la détection d'erreur, il est important que celles-ci répondent à deux propriétés :

- Efficacité et rapidité de calcul.

[5] Liste non exhaustive

- Résultat de taille moindre que l'élément de départ.

Celles-ci peuvent répondre à d'autres propriétés souhaitables :

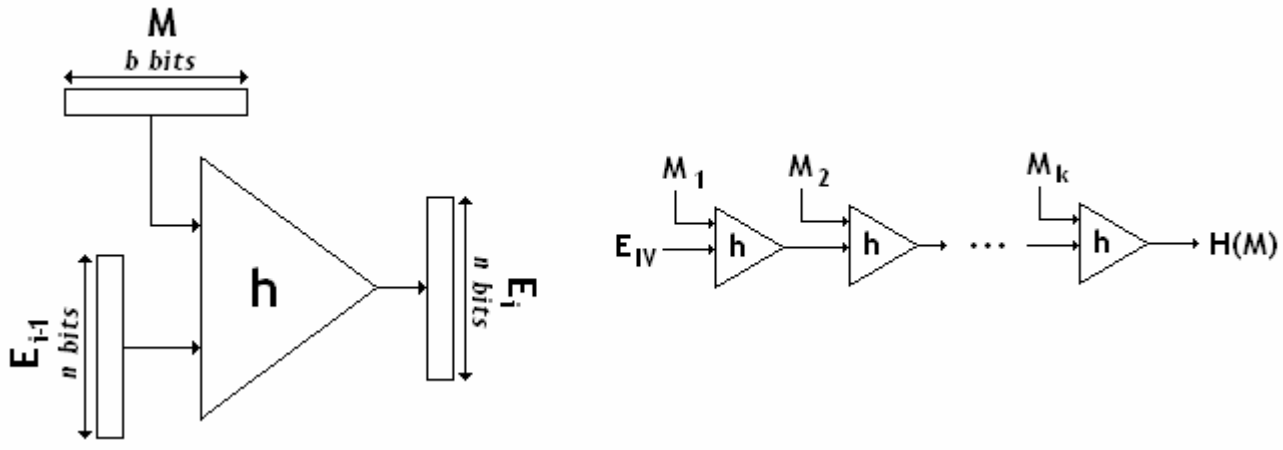
- Résistance à la préimage : A *y* donné, il est impossible de trouver en temps raisonnable un *x* tel que $y = H(x)$.
- Résistance à la seconde préimage : A *x* donné, on ne peut pas trouver en temps raisonnable un $x \neq x'$ tel que $H(x) = H(x')$.
- Résistance aux collisions : On ne peut pas trouver *x* et *x'* tel que $H(x) = H(x')$
- Effet avalanche : Un infime changement sur *x* doit entraîner une perturbation totale ou très élevée sur *H(x)*.

Une fonction de hachage à sens unique est une fonction de hachage vérifiant les propriétés de résistance à la préimage et à la seconde préimage.
Ces fonctions sont très exploitées en cryptographie (pour la sauvegarde de mot de passe ou pour créer des signatures numériques), notamment lorsqu'elles vérifient, en plus, la propriété d'effet avalanche. On parle alors de fonctions de hachage cryptographique, le temps de calcul devenant une contrainte secondaire.

Note : On considère qu'une fonction de hachage cryptographique est cassée lorsque ses propriétés de fonction de hachage à sens unique ne sont plus vérifiées.

b) Architecture générale

L'architecture la plus utilisée dans le calcul d'empreinte est celle de Merkle-Damgård, se présentant sous la forme suivante :



La fonction h génère une empreinte E_i de *n* bits à partir d'un message *M* de *b* bits et d'une empreinte d'initialisation E_{i-1} de *n* bits.

Lorsque le message *M* est plus grand que *b* bits, celui-ci est découpé en blocs de *b* bits $M_1,\ldots,M_k$ (il faut prévoir à rajouter des bits à *M* pour que celui-ci divise exactement *b*). Chaque bloc M_i passe dans la fonction *h* avec comme empreinte d'initialisation l'empreinte du bloc de message précédent (E_{i-1}). Si le bloc n'a pas de prédécesseur ($i = 1$) alors la fonction *h* prend comme empreinte d'initialisation une valeur E_{IV} définie par l'algorithme ou l'implémentation de celui-ci.

L'ensemble des itérations de *h* est nommé *fonction H*. La conclusion de Merkle et Damgård est que si *h* est résistante aux collisions alors *H* l'est aussi.

3. Hachage MD5

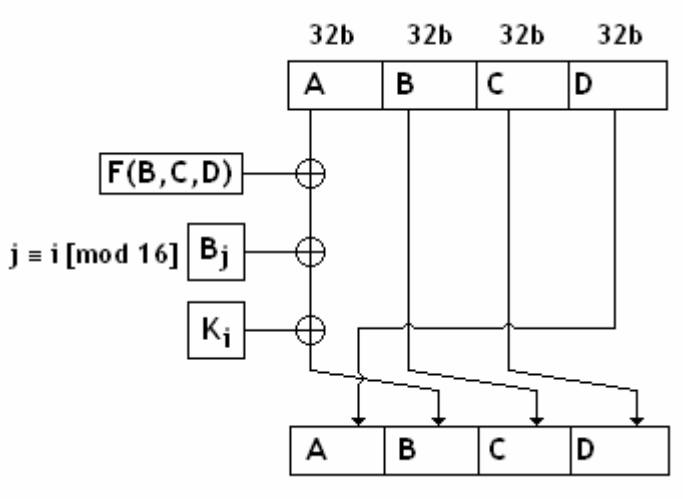
MD5 est une évolution de MD4 proposée par Rivest en 1991, celle-ci utilise un message découpés en blocs de taille *b = 512* bits et produit une empreinte de taille *n = 128* bits.

a) Mise en œuvre

Tout d'abord, le message est complété pour obtenir une taille multiple de 512 (la méthode de bourrage employée ne nous intéresse pas ici).

Ensuite, le message est découpé en blocs de 512 bits. Pour la suite de l'explication, nous considérerons que le message employé est déjà un bloc *B* de 512 bits.

Le bloc *B* est découpé en 16 sous-blocs B_k où $k \in [\![0,...,15]\!]$ de 32 bits.
64 tours sont alors effectués, chaque tour étant défini par le schéma suivant :



i-ème tour dans MD5 où $i \in [\![0,...,63]\!]$

K_i est un élément de l'ensemble *K* contenant 64 constantes fixées.

A, B, C et *D* sont quatre blocs de 32 bits constituant l'empreinte finale de 128 bits. Lors du premier tour, ces quatre blocs sont initialisés avec des constantes.
F est une fonction qui évolue tous les 16 sous-tours (4 fois au total) selon le schéma suivant :

- $F = (B \text{ AND } C) \text{ OR } (\overline{B} \text{ AND } D)$
- $F = (D \text{ AND } B) \text{ OR } (\overline{D} \text{ AND } C)$
- $F = B \oplus C \oplus D$
- $F = C \oplus (B \text{ OR } \overline{D})$

b) Conclusion

La taille d'empreinte de MD5 est de 128 bits. Détecter une collision par force brute nécessiterait 2^{64} calculs d'empreintes. Toutefois, des algorithmes ont permis de réaliser ces recherches en 2^{42}, voire en 2^{30} opérations.

En 2004, une équipe chinoise découvre des collisions complètes (deux messages différents produisant une même empreinte : résistance à la seconde préimage) en quelques heures de calculs sur un système parallélisé.

Des projets de calculs pour prouver que MD5 ne répond plus à la propriété de la résistance à la préimage ont été arrêtés dès qu'il a été prouvé que l'agorithme n'était pas cryptographiquement sur.

Ainsi, MD5 tend à être déprécié au profit d'algorithme plus sûr. Il est toutefois encore largement utilisé à des fins de détection d'erreur.

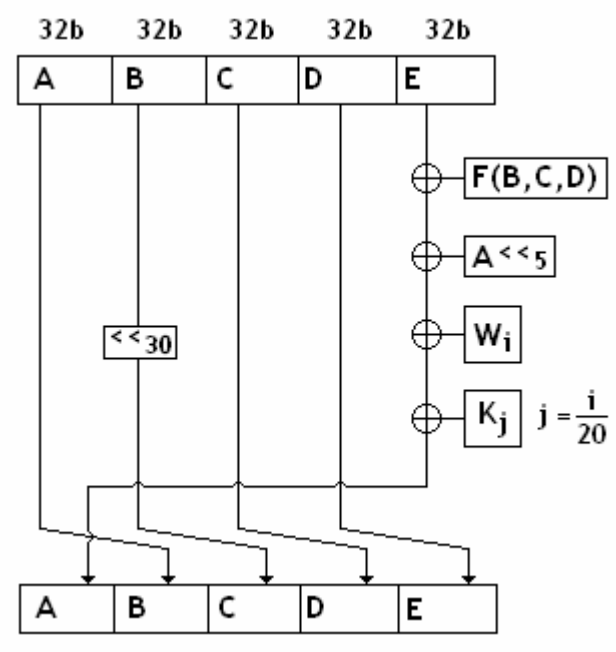
4. Hachage SHA-1 et SHA-256

SHA-1 (Secure Hash Algorithm) fut publiée sous forme d'une norme par le NIST (National Institute of Standards and Technology) en 1995.
SHA-1 exploite des blocs de taille $b = 512$ bits et produit une empreinte de taille $n = 160$ bits.

a) Mise en œuvre

On peut constater certaines similitudes avec MD5. En effet, le bloc *B* de 512 bits est lui aussi découpé en 16 sous-blocs B_k ($k \in [\![0,...,15]\!]$) de 32 bits. Ils sont ensuite étendus en 80 nouveaux blocs W_j ($j \in [\![0,...,79]\!]$). De plus, le système exploite quatre constantes K_i ($i \in [\![0,...,3]\!]$) fixées.

Comme pour MD5, 64 tours sont alors effectués, chaque tour étant défini par le schéma suivant :



i-ème tour dans SHA-1 où $i \in [\![0,...,79]\!]$

A, B, C, D et *E* sont cinq blocs de 32 bits constituant l'empreinte finale de 160 bits. Lors du premier tour, ces cinq blocs sont initialisés avec des constantes.
F est une fonction qui évolue tous les 20 sous-tours (4 fois au total) selon le schéma suivant :

- $F = (B \text{ AND } C) \text{ OR } (\bar{B} \text{ AND } D)$
- $F = B \oplus C \oplus D$
- $F = (B \text{ AND } C) \oplus (B \text{ AND } D) \oplus (C \text{ AND } D)$
- $F = B \oplus C \oplus D$

b) Conclusion

La taille d'empreinte de SHA-1 est de 160 bits. Détecter une collision par force brute nécessiterait en théorie 2^{80} calculs d'empreintes. Cependant, une technique similaire à celle employée pour casser MD5 a permis a des algorithmes de détecter des collisions en 2^{63}.

En 2004 (année apparemment fatidique pour les fonctions de hachage) des collisions complètes ont été découvertes sur SHA-0, laissant à penser que SHA-1 pourrait lui aussi subir ce genre d'attaque.
Aussi, SHA-1 est considéré comme cassé, même si l'attaque reste très difficile à l'heure actuelle (Adi Shamir estime qu'elle serait possible via un calcul distribué à l'échelle planétaire). De ce fait, SHA-1 est progressivement retiré aux profits de SHA-256 (il existe aussi des spécifications SHA-384 et SHA-512), une évolution de SHA-1 produisant des empreintes plus grandes.

Pour anecdote, la famille SHA a donné naissance, après légère adaptation, à des algorithmes de cryptage symétriques nommés SHACAL (version 1 et 2).