Table des matières

Chapitre 1

Introduction à la cryptographie

"Ce qui est secret est vulnérable."
Jacques STERN (La Recherche Juin 2008).

1.1 Définitions

La cryptographie est pluridisciplinaire par excellence, carrefour de l'algèbre linéaire, de la théorie des groupes, de la théorie de complexité, de la théorie des nombres, de la géométrie algébrique, de l'algorithmique et de l'informatique.

La cryptographie est la science qui utilise les mathématiques pour préserver la discrétion des messages. Elle permet aussi de stocker des informations sensibles ou de les transmettre à travers d'un canal non sécurisé (comme l'Internet, radio, poste etc).

Le mot cryptographie vient des mots grecs "kruptos" qui veut dire cacher, et graphein qui veut dire écrire. C'est à dire écrire en langage codé, secret, chiffré.

La cryptographie est pratiquée par des **cryptographes**.

Le **chiffrement** ou le **cryptage** est l'opération qui consiste à transformer un message clair en un message incompréhensible pour tout intrus. Le message transformé s'appelle **message chiffré**, **cryptogramme** ou message crypté.

Le **déchiffrement** ou le **décryptage** est l'opération inverse du chiffrement, elle consiste à transformer un message chiffré en un message clair.

Une **clef** est un paramètre permettant des opérations de chiffrement et/ou déchiffrement.

La **cryptanalyse** est la science qui vise à retrouver le texte en clair sans connaître la clef. Une cryptanalyse réussie peut fournir soit le texte clair soit la clef.

Une tentative de cryptanalyse s'appelle **attaque**.

La **cryptologie** est la discipline mathématique qui englobe la cryptographie et la cryptanalyse.

Un **algorithme cryptographique** est un ensemble de fonction mathématiques utilisé pour le chiffrement et le déchiffrement.

Un **crypto-système** est l'algorithme cryptographique ainsi que toutes les clés possibles et tous les protocoles qui le font fonctionner.

La **robustesse** d'un algorithme de chiffrement désigne sa force de résistance aux attaques.

La cryptographie doit assurer :

- La **confidentialité**, consiste à rendre l'information inintelligible sauf à ceux de droit. Se résout par des algorithmes cryptographiques.

- L'**authentification** : le destinataire doit pouvoir s'assurer de l'origine du message. Un intrus ne peut passer pour l'expéditeur. Se résout par la signature électronique.

- L'**intégrité** : le destinataire doit pouvoir vérifier que le message n'a pas été modifié en cours de route. Un intrus doit être incapable de faire passer un faux message pour un vrai. Se résout grâce aux fonctions de hachage (fonction qui réduit un message de taille arbitraire en une chaîne de taille fixe).

- La **non-répudiation** (ou non désaveu) : un expéditeur ne doit pas pouvoir nier à tort avoir envoyé un message.

Un système cryptographiques satisfaisant ces 4 propriétés fondamentales s'appelle **protocoles cryptographiques**.

1.2 Les principaux crypto-systèmes

1.2.1 Cryptographie à clé privée

S'appelle aussi crypto-système symétrique. Il est caractérisé par une seule clé partagée entre l'expéditeur et le destinataire et qui sert au chiffrement et au déchiffrement. Elle doit rester secrètes. Les algorithmes les plus répandus sont : DES, 3DES, IDEA, AES, ...

Ces algorithmes sont basés sur des opérations de transposition et de substitution des bits du texte clair en fonction de la clé.

La taille des clés est de l'ordre de 128 bits, 256 bits.

L'avantage principal de la cryptographie symétrique est sa rapidité. Son inconvénient principal est le partage de la clé.

1.2.2 Cryptographie à clef publique

S'appelle aussi cryptographie **asymétrique**. Elle est caractérisée par deux clés, une clé publique P_K, et une clé privée secrète S_K. La connaissance de P_K ne permet pas de déduire S_K.

Les algorithmes se basent sur des concepts mathématiques tels que l'exponentiation de grands nombres premiers (RSA), le problème des logarithmes discrets (ElGamal), ou encore le problème du sac dos (Merkle-Hellman).

Le principe de ce genre d'algorithme est qu'il s'agit d'une fonction unidirectionnelle, trappe. Une telle fonction a la particularité d'être facile à calculer dans un sens, mais difficile voire impossible dans le sens inverse. La seule manière de pouvoir réaliser le calcul inverse est de connaître une trappe.

L'algorithme de cryptographie asymétrique le plus connu est le RSA.

La taille des clés s'étend de 512 bits à 2048 bits en standard.

Le chiffrement symétrique est environ 1000 fois plus rapide que le chiffrement asymétrique.

La distribution des clés est facile car l'échange des clés secrètes n'est plus nécessaire. Chaque utilisateur conserve sa clé secrète sans jamais la divulguer. Seule la clé publique est distribuée. La connaissance de la clé publique ne permet pas de déduire la clé secrète.

1.3 La cryptanalyse

Il y a quatre niveaux d'attaques. Chacune suppose la connaissance complète de l'algorithme de chiffrement. Oscar étant l'attaquant.

Niveaux d'attaques possibles :
1 Texte chiffré connu : Seul C est connu d'Oscar
2 Texte clair connu : Oscar connaît C et M correspondant
3 Texte clair choisi : 8M, Oscar peut obtenir C
4 Texte chiffré choisi : 8C, Oscar peut obtenir M

Algorithmes d'attaques
1 Attaque brutale : Énumérer toutes les valeurs possibles de clefs.
2 Attaque par séquences connues : deviner la clef si une partie du message est connue ex : en-têtes de standard de courriels
3 Attaque par séquences forcées : faire chiffrer par la victime un bloc dont l'attaquant connaît le contenu, puis on applique l'attaque précédente ...
4 Attaque par analyse différentielle : utiliser les faibles différences entre plusieurs messages (ex : logs) pour deviner la clef

1.3.1 Attaque sur le texte chiffré connu

Le cryptanalyse dispose du texte chiffré de plusieurs messages, qui ont été chiffré avec le même algorithme. on recherche le texte clair et/ou la clé. On procède par analyse de fréquence des lettres utilisées dans le texte chiffré.

1.3.2 Attaque à texte clair connu

Étant donné un texte chiffré et un fragment de texte clair associé, on recherche le texte clair restant et/ou la clé. On procède par force brute,

1.3.3 Attaque sur un texte clair choisi

Le cryptanalyse peut choisir un texte clair M et obtenir le texte chiffré associé.

1.3.4 Attaque sur le texte chiffré choisi

Le cryptanalyse peut choisir un texte chiffré et obtenir le texte déchiffré associé M.

1.4 Algorithme publié et algorithme secret

1.4.1 Algorithme secret

De tels algorithmes sont utilisés par un plus petit nombre d ?utilisateurs. Donc il y a d'intérêts à le casser. Il est impossible de garder un Algorithme secret pour longtemps.

1.4.2 Algorithme publié

Tout le monde a le droit de l'explorer. Ainsi, les failles peuvent être plus facilement découvertes. La sécurité est donc améliorée. Ce qui permet aussi une standardisation générale. Les algorithmes publiés sont de loin les plus utilisés.

1.4.3 Principe de Kerckhoffs

En 1883 A. Kerckhoffs [**?**] a posé les principes de la cryptographie moderne : La sécurité d'un système cryptographique ne doit pas reposer sur la non divulgation de l'algorithme de chiffrement utilisé mais uniquement sur la non divulgation des clés utilisées.

Autrement dit aucun secret ne doit résider dans l'algorithme de chiffrement mais plutôt dans la clé. Ce principe est bien évidemment toujours d'actualité.

L'algorithme de confidentialité, jamais rendu public officiellement, de la norme GSM a été dévoilé et publié sur l'Internet. Il en est de même pour la RFID (Radio Frequency Identification ou identification par radio-fréquence) De nombreuses voitures intègrent un systèmes anti-vol, fondés sur la technologie RFID, relié au système d'injection de carburant.

Un système cryptographique ou un crypto-système est la donnée de :
- un ensemble fini $\mathcal{P}$ appelé l'espace des textes clairs ;
- un ensemble fini $\mathcal{C}$ appelé l'espace des textes chiffrés ou cryptogrammes ;
- un ensemble fini $\mathcal{K}$ appelé l'espace des clefs ;
- pour tout $k \in \mathcal{K}$, une fonction de chiffrement $e_k : \mathcal{P} \to \mathcal{C}$ et une fonction de déchiffrement $d_k : \mathcal{C} \to \mathcal{P}$ telles que $d_k o e_k = Id_{\mathcal{P}}$.

Pour utiliser un tel crypto-système l'émetteur et le destinataire doivent se mettre d'accord sur une clef qu'ils doivent conserver secrète. L'émetteur envoie un cryptogramme $C = \mathcal{E}(k, m)$ au destinataire qui calcule $\mathcal{D}(k, C) = M$ pour retrouver le message clair m.

Un crypto-système est mono-alphabétique si une même lettre dans le texte clair est toujours chiffré en la même lettre du cryptogramme.

1.5 Quelques crypto-systèmes historiques

Par $\mathbb{Z}_{26}$ on note l'anneau des entiers modulo 26 soit $\{0, 1, \cdots, 25\}$. On représente chaque lettre par son ordre dans l'alphabet : a, b, c ... z par les nombres 0, 1, ..., 25.

1.5.1 Chiffrement par décalage

$P = C = K = \mathbb{Z}_{26}$, pour $0 \leq k \leq 25$ $e_k(x) = x + k \ mod 26$ et $d_k(y) = x - k \ mod \ 26$ où $x, y \in \mathbb{Z}_{26}$. Le chiffrement de César est un cas particulier de ce chiffrement, il suffit de prendre $k = 3$. Il est d'une sécurité très faible, en moyenne et par recherche exhaustive on a 13 essais.

1.5.2 Chiffrement affine

Par $\mathbb{Z}_{26}^*$ on note le groupe multiplicatif des éléments inversibles dans $\mathbb{Z}_{26}$. On pose $P = C = \mathbb{Z}_{26}$, $K = \mathbb{Z}_{26}^* \times \mathbb{Z}_{26}$ pour $k = (a, b) \in K$ $e_k(x) = ax + b$ et $d_k(y) = a^{-1}(y - b)$. On montre que facilement que e_k est une fonction de chiffrement si elle est injective ce qui revient à dire que a est inversible dans l'anneau $\mathbb{Z}_{26}$.

Par exemple si $(a, b) = (3, 5)$ on a $e_k(x) = 3x + 5$ et $d_k(y) = 9(y + 21)$

Il y a $12 \times 26 = 312$ clés possibles, c'est peu !

1.5.3 Chiffrement par substitution

$P = C = \mathbb{Z}_{26}$, $K = \mathcal{S}_{26}$ l'ensemble des permutations de l'ensemble $\{0, 1, \cdots, 25\}$ on $|K| = 26!$ si $\pi \in K$ $e_\pi(x) = \pi(x)$ et $d_\pi(y) = \pi^{-1}(y)$. Le nombre de clefs est 26! soit un peu plus de 4×10^{26}. La recherche exhaustive est difficile mais il y a une autre méthode. Le chiffrement par décalage est un cas particulier du chiffrement par substitution.

Dans les chiffrements vus à présent, un caractère est toujours chiffré de la même façon. On dit que ces chiffrements sont **mono-alphabétique**. Contrairement aux chiffrements ci-dessus, dans les chiffrements ci-dessous et dans un même message, un caractère est chiffré de plusieurs façons On l'appel chiffrement **poly-alphabétique**.

1.5.4 Chiffrement de Vigenère

Mis au point en 1586 par Blaise de Vigenère, un diplomate français. Soit m un entier > 0. $P = C = K = \mathbb{Z}_{26}^m$. Si $k = (k_1, \cdots, k_m) \in K$ alors $e_k(x_1, \cdots, x_m) = (x_1 + k_1, \cdots, x_m + k_m)$ et $d_k(y_1, \cdots, y_m) = (y_1 - k_1, \cdots, y_m - k_m)$. Le nombre de clé est 26^m, par exemple pour $m = 5$, 26^m est environ $1,1 \times 10^7$. A la main c'est difficile, mais à la machine c'est très facile. Ce chiffrement est poly-alphabétique. Un caractère peut être chiffré de m façons.

1.5.5 Chiffrement de Hill (1929)

L. Hill, mathématicien cryptographe (1891-1961). Soit m un entier > 0. $P = C = \mathbb{Z}_{26}^m$, $K = GL_m(\mathbb{Z}_{26}$ le groupe des matrices $m \times m$ inversibles et à coefficients dans $\mathbb{Z}_{26}$. Soit $k = \in K$, $e_k(x_1, \cdots, x_m) = (x_1, \cdots, x_m)k$ et $d_k(y_1, \cdots, y_m) = (y_1, \cdots, y_m)k^{-1}$.

1.5.6 La machine ENIGMA

$m = 3$ et $k = (3, 1, 2)$ le texte clair `cryptographie` est chiffré en ?

Montrer que

1.5.7 Masque jetable (One time pad)

Mis au point par Vernam en 1917. Utilisé pour le téléphone rouge entre Moscou et Washington pendant la guerre froide. Il a aussi servi à chiffrer les messages télégraphiques. Le problème de distribution de clef a été résolu par la valise diplomatique. Mais pour d'autres utilisations il est peu

pratique. En fait, la clé est aussi longue que le message à chiffrer. En 1941, C. Chanon à montré que ce chiffrement est impossible à casser.

Soit K une clé et M le message à chiffrer. Ils sont de même longueur. On écrit M et K en binaire, par exemple en ASCII. On a on obtient le cryptogramme $C = M \oplus K$. Ayant la clé K, le destinataire calcule $C \oplus K$ pour obtenir M.

Pour chiffrer M = RDV A DIX H, on convertit ce message en binaire en utilisant le code ASCII,
0101001001000100010101100010000001000001001000000100010001001001010110000010000001001000
et on considère la clé K = MERCI BIEN. (y compris le point) soit en code ASCII
0100110101000101010100100100001101001001001000000100001001001001010001010100111000101110
$C = K \oplus M =$
et pour déchiffrer $C \oplus K =$
Montrer que

1.5.8 Chiffrement par permutation (transposition)

Dans ce chiffrement, les caractères ne change pas, mais ils sont réordonnés. Soit $P = C = \mathbb{Z}_{26}^m$ où m est un entier > 1 et K est l'ensemble des permutations de $\{1, 2, \cdots, m\}$. Pour $\sigma \in K$ $e_\sigma(x_1, \cdots, x_m) = (x_{\sigma(1)}, \cdots, x_{\sigma(m)})$ et $d_\sigma(y_1, \cdots, y_m) = (y_{\sigma^{-1}(1)}, \cdots, y_{\sigma^{-1}(m)})$. Ce chiffrement consiste à conserver les mêmes caractères du texte clair et on applique une permutation à chaque groupe de m caractères. Pour $m = 6$ et

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

Le chiffrement de `cryptography` donne `otpyrcyhparg` Le chiffrement par permutation est un cas particulier du chiffrement de Hill. En fait à toute permutation on peut associer une matrice (a_{ij}) de taille $m \times m$ donnée par

$$a_{ij} = \begin{cases} 1 & si\ i = \pi(j) \\ 0 & sinon \end{cases}$$

le chiffrement avec K_π est équivalent au chiffrement de Hill ? $K_\pi^{-1} = K_{\pi^{-1}}$

?

Montrer que

1.5.9 Le carré de Polybe

Polybe est un historien grec qui a vécut aux environs de -205 avant JC à -125 av. JC

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I,J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

lettre	fréquence	lettre	fréquence
a	6,16	n	6,02
b	0,40	o	5,12
c	5,35	p	2,92
d	3,86	q	0,62
e	18,61	r	5,35
f	2,24	s	6,96
g	1,79	t	7,41
h	1,48	u	5,03
i	6,35	v	1,03
j	0,04	w	0,35
k	0,13	x	0,36
l	5,26	y	1,39
m	1,97	z	0,04

TABLE 1.1 – Table des fréquences relatives des lettres en français.

Pour chiffrer un texte on remplace chaque lettre par ses coordonnées dans le tableau, en écrivant d'abord la ligne, puis la colonne. Par exemple, le A est remplacé par 11, le B par 12, le F par 21, le S par 43 etc. Si nous codons FEU nous obtenons 211545. Remarquons que nous pouvons remplir le tableau de plusieurs façons différentes.

1.5.10 Cryptanalyse des crypto-systèmes monoalphabétiques

Une méthode de cryptanalyse un crypto-systèmes est d'explorer l'espace de toutes les clés possibles. C'est ce qu'on appelle cryptanalyse exhaustive.

D'après un document rédigé et retrouvé en 1987 à Istanbul, Abu Yusuf Ya'qub ibn Is-haq ibn as-Sabbah Oòmran ibn Ismaïl al-Kindi (801-873) savant arabo-musulman du neuvième siècle, dans son traité intitulé "Manuscrit sur le déchiffrement des messages cryptographiques" a décrit la cryptanalyse par la fréquence d'apparition des lettres. C'est le premier manuscrit connu faisant mention des fréquences d'apparition des lettres.

Il explique que "la façon d'élucider un message crypté, si nous savons dans quelle langue il est écrit, est de nous procurer un autre texte en clair dans la même langue, de la longueur d'un feuillet environ, et de compter alors les apparitions de chaque lettre. Ensuite, nous nous reportons au texte chiffré que nous voulons éclaircir et relevons de même ses symboles. Nous remplaçons le symbole le plus fréquent par la lettre première (la plus fréquente du texte clair), le suivant par la deuxième, le suivant par la troisième, et ainsi de suite jusqu'à'à ce que nous soyons venus à bout de tous les symboles du cryptogramme à résoudre". Cette technique est appelée analyse des fréquences.

Il est aussi intéressant d'étudier la fréquence d'apparence de quelques diagrammes ou trigrammes en français la, le, ent, tion etc en anglais th, in, an the, ing etc.

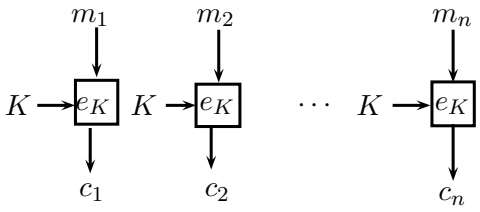


FIGURE 1.1 – Mode de chiffrement ECB

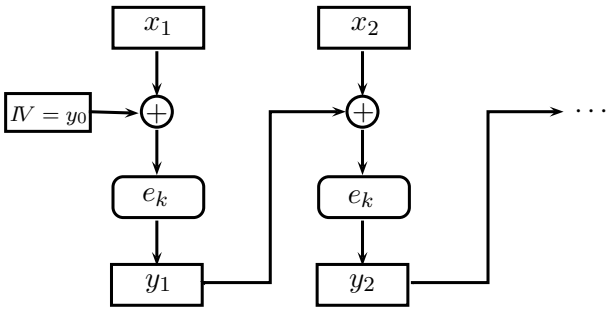


FIGURE 1.2 – Mode de chiffrement CBC

1.6 Modes de chiffrement par bloc

Il y a quatre modes de chiffrement par bloc : ,

1.6.1 Le mode ECB (Electronic CodeBook)

C'est le mode le plus simple. Voir 1.1. Le message à chiffrer est découpé en blocs. Chaque bloc est chiffré indépendamment des autres. ce mode est vulnérable aux attaques. les bloc qui se répètent sont chiffré de la même façon. Permet de paralléliser les calculs. Voir 1.1. Il n'est pas utilisé dans la pratique. Mais il peut être utilisé pour le chiffrement de mots de passe.

Un attaquant peut permuter des blocs, ou remplacer un bloc par autre sans que le destinataire ne s'en aperçoive.

1.6.2 Le mode CBC (Cipher Block Chaining)

Voir 1.2. On fixe une valeur initiale y_0 qui peut être choisie aléatoirement et partagée en clair.

$$y_i = e_k(y_{i-1} \oplus x_1) \quad \text{pour } i \geq 1$$

En plus de la clé les correspondants partagent la valeur initiale y_0. Avant de chiffrer un bloc on le xor avec le chiffré du bloc qui le précède. C'est d'ailleurs le mode le plus courant. Il rend le chiffrement plus complexe en créant une dépendance entre les blocs successifs. Mais Il est impossible de paralléliser le chiffrement.

Pour le déchiffrement en mode CBC voir 1.3.

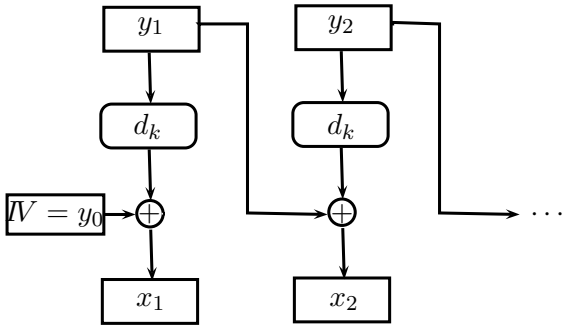


FIGURE 1.3 – Mode de déchiffrement CBC

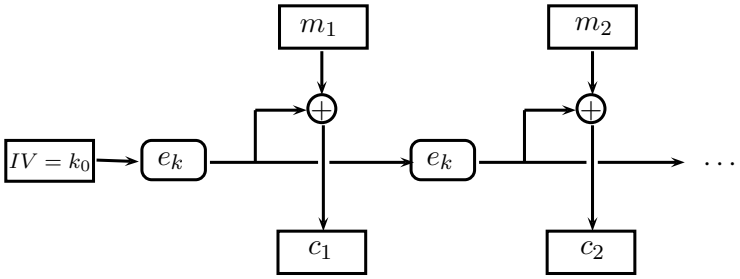


FIGURE 1.4 – Mode de chiffrement OFB

1.6.3 Le mode OFB (Output FeedBack)

C'est une variante du mode CFB qui permet d'avoir un chiffrement totalement symétrique. Peu utilisé, voir 1.4. c_0 une valeur initiale et pour $i \geq 1$ le bloc m_i est chiffré comme

$$c_i = m_i \oplus e_k(c_{i-1})$$

Ce mode est utilisé pour des transmissions sur des canaux bruités. Par exemple transmission satellitaire.

Le déchiffrement est donné par $c_i = e_k(c_{i-1})$ et $m_i = c_i \oplus e_k(c_{i-1})$

1.6.4 Le mode CFB (Cipher FeedBack)

Voir 1.5. On commence par une valeur initiale c_0, puis on chiffre le bloc clair m_i comme

$$c_i = m_i \oplus e_k(c_{i-1}) \text{ pour } i \geq 1$$

Le mode CFB est une façon de transformer une fonction E en un chiffrement par flot auto-synchronisant.

usage possible en signature et en chiffrement réseau.

Le déchiffrement, voir 1.6, ne nécessite pas d'utiliser la fonction de déchiffrement, en effet $m_i = c_i \oplus e_k(c_{i-1)}$ avec c_0 est valeur initiale.

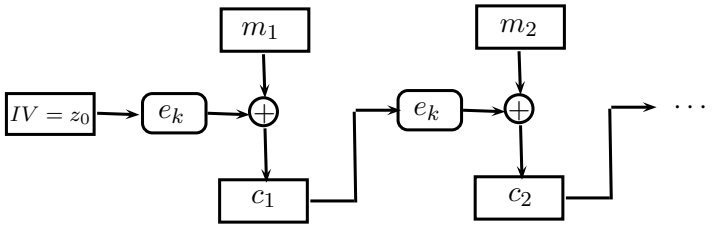


FIGURE 1.5 – Mode de chiffrement CFB

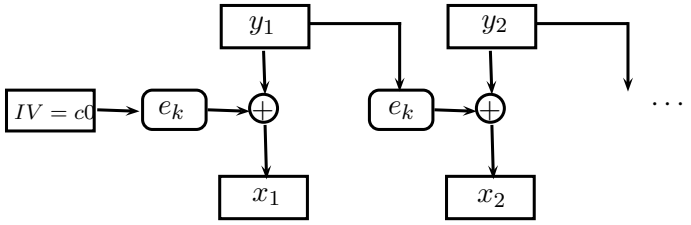


FIGURE 1.6 – Mode de déchiffrement CFB

1.6.5 Le mode CTR (Counter-mode encryption)

Ce mode utilise pour le chiffrement un compteur de valeur initiale v_0. Il est totalement symétrique. Il a de nombreux avantages : le chiffrement par flot, l'accès aléatoire aux données, parallélisable et n'utilise que la fonction de chiffrement. Le compteur utilisé peut être une suite pseudo-aléatoire qu'il sera facile de retrouver à partir de la graine (vecteur d'initialisation).

Un bloc m_i est chiffré comme

$$c_i = m_i \oplus e_k(v_0 + i)$$

Le déchiffrement se fait par

$$m_i = c_i \oplus E_k(v_0 + i)$$

Les différents calculs de chiffrement et de déchiffrement sont indépendants, comme pour le mode ECB, mais un même bloc n'est jamais chiffré de la même façon.

1.7 Cryptographie symétrique et asymétrique

Il existe deux grands types de cryptographies :

- la cryptographie symétrique, (conventionnelle ou chiffrement à clef secrète) regroupe les algorithmes pour lesquels expéditeur et destinataire partagent une seule et même clef utilisée à la fois pour le chiffrement et le déchiffrement.

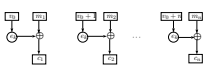


FIGURE 1.7 – Le mode de chiffrement CTR

La cryptographie à clé secrète peut se classer en deux catégories :
1) les systèmes de chiffrement par bloc.
2) les systèmes de chiffrement à flot.

Un système de chiffrement par bloc opère sur des blocs de texte clair de taille fixe, et renvoie des blocs de texte chiffré de taille fixe (en général de même taille).

Un système de chiffrement à flot opère sur les caractères individuels du texte clair par une transformation dépendant de la clé et de la position.

Le problème étant qu'on doit disposer d'un moyen sécurisé pour échanger la clef.

Ce chiffrement a des avantages. Il est très rapide. Il est particulièrement utile pour chiffrer des données à archiver. Cependant, ce type chiffrement seul est inadéquat au transmission de données sécurisées, simplement en raison de la difficulté de la distribution sécurisée de la clef.

- la cryptographie à clef publique évite le partage d'un secret entre l'expéditeur et le destinataire il suffit à l'émetteur de chiffrer le message avec la clef publique du destinataire. Ce dernier, à l'aide de la clef secrète correspondante, est seul en mesure de déchiffrer le message reçu.

Toute personne en possession d'une copie de votre clé publique peut ensuite chiffrer des informations que vous seul pourrez lire. Même des gens que vous n'avez jamais rencontrés.

Le principal avantage de la cryptographie à clé publique est qu'elle permet à des gens qui n'ont pas d'accord de sécurité préalable d'échanger des messages de manière sûre.

Les problèmes de distribution de clef sont résolus par la cryptographie à clef publique, dont le concept fut inventé par Whitfield Diffie et Martin Hellman en 1975. Mais dans **??** il est établit que les Services secrets britanniques étaient les premiers à l'inventer.

Il y a des crypto-systèmes combinent à la fois les meilleures fonctionnalités de la cryptographie symétrique et de la cryptographie asymétrique. un tel crypto-système s'appelle crypto-système hybride.

1.8 Signatures numériques

Un des avantages majeurs de la cryptographie à clé publique est qu'elle offre une méthode permettant d'utiliser des signatures numériques. La signatures numérique permet de contrôler l'authenticité, l'intégrité du message, et la non répudiation. Ces éléments sont au moins aussi importants que le chiffrement des données, sinon davantage. Une signature numérique a le même objet qu'une signature manuelle. bien qu'une signature manuelle est facile à contrefaire. Une signature numérique est pratiquement impossible à contrefaire et, de plus, elle atteste le contenu de l'information autant que l'identité du signataire.

1.9 Stéganographie

La stéganographie (du grec steganos, couvert et graphein, écriture) dissimule l'existence même de l'information secrète (encre sympathique etc...). c'est l'art de cacher un message au sein d'un autre message de caractère anodin, de sorte que l'existence même du secret en soit dissimulée. Alors qu'avec la cryptographie habituelle, la sécurité repose sur le fait que le message ne sera sans doute pas compris.

On peut cacher des messages dans des images jpeg, de fichiers mp3 ou des films. Jadis on utilisait l'encre sympathique etc...

1.10 Exercices

Exercice 1. Chiffrer une phrase à l'aide du carré de Polybe. Faites-la déchiffrer par votre voisin de classe.

Exercice 2. Soit le chiffrement $k_1 = K \in Z_{26}$ et $k_i = x_{i-1}$ pour tout message $x = (x_1, x_2, \cdots)$. On définit $E_k(x) = x + k \ mod\ 26$ et $D_k(y) = y - k \ mod\ 26$.

Décrypter le chiffré suivant : `MALVVMAFBHBUQPTSOXALTGVWWRG`.

Exercice 3. Une recherche exhaustive de la clé dans le cas du système de Vernam a-t-elle un sens ? Expliquez votre réponse.

Exercice 4. On considère une fonction de chiffrement par bloc de longueur 2 pour des clefs de longueur 2 donnée par

$$\begin{aligned} \mathcal{E}_k : \quad & \{0,1\}^2 \longrightarrow \{0,1\}^2 \\ & (m_1, m_2) \longmapsto S_1((m_1 \oplus k_1, m_2 \oplus k_2)) \end{aligned}$$

où la fonction S_1 est décrite ci-dessous

X	[0, 0]	[1, 0]	[0, 1]	[1, 1]
$S_1(X)$	[1, 1]	[1, 0]	[0, 0]	[0, 1]

1. Chiffrer le message $M = [0, 1, 1, 1, 0, 1]$ avec la clef $K = [1, 0]$

a) En utilisant le mode ECB,

b) En utilisant le mode OFB,

2. Déchiffrer le message$C = [0, 1, 1, 1, 0, 1]$ dans le cas où il a ?eté chiffré avec la clef$K = [1, 1]$ et

a) en utilisant le mode CBC,

b) en utilisant le mode CFB.

Chapitre 2

DES (Data Encryption Standard)

2.1 Introduction

HISTORIQUE :

Avant 1975 : chiffrements artisanaux (Vigenère, Hill).

1975-2000 : le DES (Data Encryption Standard), mais aussi FEAL, IDEA, RC5, ...

2000 ... ? AES (Advanced Encryption Standard), RC6, CAMELLIA, ...

Au début des années 1970, le développement des communications entre ordinateurs a nécessité la mise en place d'un système de chiffrement standard.

DES est (Data Encryption Standard), et en français Standard de Chiffrement de Données. C'est un système de chiffrement par blocs et à clé secrète. Conçu par IBM en 1975 suite à un appel d'offre de la NSB (pour National Bureau of Standards) actuellement NIST (pour National Institute of Standards and technology) des Etats Unis en 1973 pour la mise au point d'un système de cryptographie.

Le cahier des charges spécifiait que la sécurité devrait être liée à la clef, et ne devait pas dépendre de la confidentialité de l'algorithme par application du principe de Kerckhoffs en plus de la confusion et de la diffusion. Pour rappel, C. Shannon a montré que la combinaison de confusion et diffusion permettait d'obtenir une sécurité convenable. La **confusion** consiste à masquer la relation entre le clair et le chiffré. Alors que la **diffusion** consiste à chaque bit de texte clair d'avoir une influence sur une grande partie du texte chiffré. Ce qui signifie que la modification d'un bit du bloc d'entrée doit entraîner la modification de nombreux bits du bloc de sortie correspondant.

La NSA (National Security Agency) participa à l'évaluation de cet algorithme. DES est soumis à l'évaluation des chercheurs de ce domaine.

Il est adopté comme standard en 1977 après lui avoir apporté des modifications par le NSA (National Security Agency). Il est re-évalué tous les cinq ans par le NBS. Il était le système le plus utilisé dans le monde jusqu'à la fin des années 1990. Il a résisté aux différentes attaques pendant un quart de siècle. Il a été utilisé dans les transactions bancaires. Il est utilisé pour chiffrer les mots de passe des systèmes Unix.

DES chiffre par bloc de 64 bits en utilisant une clef de 56 bits, augmentée par 8 bits de parité, pour obtenir un bloc de texte chiffré de 64 bits. DES est réalisé en 16 tours ou itérations. Voir 2.1

La clef de 64 bits est utilisée pour générer 16 autres clefs de 48 bits chacune qu'on utilisera lors

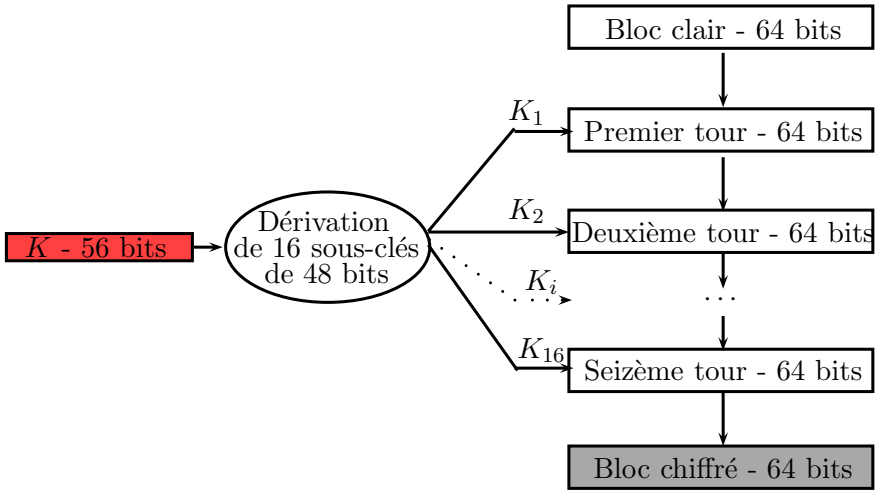


FIGURE 2.1 – Fonctionnement du DES

de chacun des 16 tours du DES. Ces clefs sont les mêmes quel que soit le bloc qu'on chiffre dans un message.

L'algorithme DES est simple, il ne combine que des permutations et des substitutions. On parle en cryptologie de techniques de confusion et de diffusion.

De plus, DES est relativement facile à réaliser matériellement et certaines puces chiffrent jusqu'à 1 Go de données par seconde.

Ainsi, DES est conçu de manière à ce qu'une légère modification dans la clé ou dans le texte clair se traduit par des changements très importants dans le texte chiffré.

Si on note par P, C et K l'ensemble des blocs clairs, l'ensemble des blocs chiffrés et l'ensemble des clés, respectivement, alors $P = C = \{0,1\}^{64}$ et $K = \{0,1\}^{56}$. À chaque clé de 56 bits, on ajoute 8 bits de parité de tel façon qu'elle devienne $b_1 b_2 \dots b_{64}$ avec $\sum_{i=1}^{8} b_{8k+i} \equiv 1 \bmod 2, 0 \leq k \leq 7$. Le nombre de clés est 2^{56}, soit environ 7.2×10^{16}.

Le DES a plusieurs avantages qui ont fait de lui un standard pendant longtemps :
il possède un haut niveau de sécurité,
il est complètement spécifié et facile à comprendre,
la sécurité est indépendante de l'algorithme lui-même, elle ne dépend que de la clé,
il est rendu disponible à tous, par le fait qu'il est publique,
il est adaptable à diverses applications (logicielles et matérielles),
il est rapide et exportable,
il repose sur une clé relativement petite, qui sert à la fois au chiffrement et au déchiffrement,
il est facile à implémenter.

2.2 Ingrédients

Dans cette section, nous décrivons, séparément, toutes les opérations qu'utilise DES.

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

TABLE 2.1 – La permutation IP

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

TABLE 2.2 – La permutation IP^{-1}

2.2.1 La permutation IP et son inverse IP^{-1}

La permutation IP (pour initiale permutation) est définie de $\{0,1\}^{64} \longrightarrow \{0,1\}^{64}$ par la Table 2.1 qui se lit de gauche à droite et de haut en bas. Un nombre dans la table indique la position du bit avant permutation et sa position dans la table indique son image après permutation.

Par exemple : le 58^e bit d'une chaîne x de 64 bits est le premier bit de $IP(x)$ et le 50^e bit de x est le deuxième bit de $IP(x)$ etc$\cdots$

Si $x = b_1 \cdots b_{64}$ alors $IP(x) = b_{58}b_{50}b_{42} \cdots b_7$

L'inverse IP^{-1} de $IP(x)$ est définie par le Table 2.2.

Si $x = b_1 \cdots b_{64}$ alors $IP^{-1}(x) = b_{40}b_8b_{48} \cdots b_{57}b_{25}$.

Les permutations $IP(x)$ et IP^{-1} n'affectent en rien la sécurité du DES. Elles sont utilisées pour rendre plus facile le chargement du texte clair ou du texte chiffré dans une puce DES, car DES est arrivé avant les microprocesseur 16 ou 32 bits.

2.2.2 L'expansion $\mathcal{E}$

La transformation $\mathcal{E} : \{0,1\}^{32} \longrightarrow \{0,1\}^{48}$ est donnée par la table 2.3. $\mathcal{E}$ permet d'étendre une chaîne x de 32 bits en un bloc de 48 bits en doublant certains bits. Par $\mathcal{E}$, $x = b_1b_2 \cdots b_{32}$ est transformé en

$$\mathcal{E}(x) = b_{32}b_1 \cdots b_4b_5b_4b_5b_6 \cdots b_{31}b_{32}b_1$$

en répétant certains bits.

Le premier bit de $\mathcal{E}(x)$ est le 32e bit de x, son 2e bit est le premier bit de x etc

2.2.3 Les S-Box

Le "S" est pour substitution. Les lignes et colonnes de tous les S_i-Box, $1 \leq i \leq 8$, sont numérotées à partir de 0 et sont en caractères gras.

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

TABLE 2.3 – L'expansion $\mathcal{E}$

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

TABLE 2.4 – La permutation P

Comment agit un S_i-box ? Les $S_i : \{0,1\}^6 \longrightarrow \{0,1\}^4$ pour $1 \leq i \leq 8$. Chaque S_i-Box associe à un bloc $B = b_1b_2b_3b_4b_5b_6$ un bloc de 4 bits :
– l'entier représenté par b_1b_6 après l'avoir transformé en décimal sélectionne une ligne de S_i-box et
– l'entier représenté par $b_2b_3b_4b_5$ après l'avoir transformé en décimal indique une colonne de S_i-box.
La valeur de $S_i(B)$ est la représentation en binaire de l'entier inscrit dans la position indiquée par la ligne et la colonne dans la $S_i - box$.

Exemple. En utilisant S_1-Box. Soit $B = b_1b_2b_3b_4b_5b_6 = 010011$ alors la ligne est $b_1b_6 = 01 = 1$ en décimal et la colonne est $b_2b_3b_4b_5 = 1001$ en décimal c'est 9, la valeur de $S_1(B) = 6$ ou 0110 en binaire et donc $S_1(010011) = 0110$.

La sécurité de DES repose sur ces tables S_i-Box de substitutions non linéaires très efficaces pour diluer les informations.

2.2.4 La permutation P

La permutation P est donnée par la Table 2.4. Pour $x = b_1b_2 \dots b32$, $P(x) = b16b_7 \dots b25$.

Par exemple sous l'action de P le bit 16 va en position 1, le bit 2 va en position 17.

Table S_1-Box

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

Table S_2-Box

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

Table S_3-Box

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

Table S_4-Box

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

Table S_5-Box

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

Table S_6-Box

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

Table S_7-Box

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

Table S_8-Box

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

2.2.5 La fonction f

La fonction $f : \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$ est définie en utilisant la fonction d'expension $\mathcal{E}$, tous les S_i-box et la permutation P. Voir la Figure 2.2 C'est une fonction de confusion.

Soit R_{i-1} et K_i deux chaînes de bits de longueurs 32 et 48 bits respectivement.

On commence par effectuer une opération de ou exclusif entre $\mathcal{E}(R_{i-1})$ et K_i. On scinde ensuite le résultat de cette opération $\mathcal{E}(R_i) \oplus K_i$ en huit blocs de 6 bits chacun, soit $B_1, \cdots, B_8$. Ainsi $\mathcal{E}(R_{i-1}) \oplus K_i = B_1.B_2 \cdots B_8$. A chaque B_j , $j = 1 \cdots 8$ on applique une fonction de substitution S_j-Box qui renvoie un bloc de 4 bits en sortie $S_j(B_j)$.

Les 8 blocs de 4 bits obtenus $C_1 = S_1(B_1),\ C_2 = S_2(B_2)\ \cdots\ C_8 = S_8(B_8)$ sont ensuite concaténés en un bloc de 32 bits auquel on applique la permutation P.

$$f(R_{i-1}, K_i) = P(S_1(B_1)S_2(B_2)\cdots S_8(B_8))$$

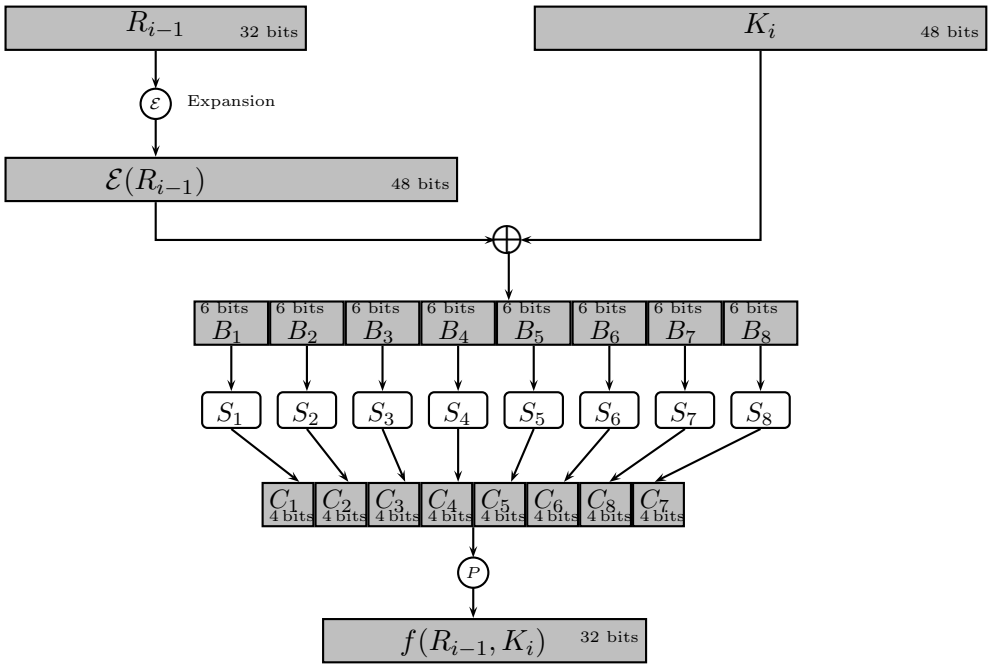


FIGURE 2.2 – Description de la fonction$f$

2.2.6 Les transformations $PC1$ et $PC2$

La permutation $PC1$ "permuted choice 1" est définie $PC1 : \{0,1\}^{64} \longrightarrow \{0,1\}^{28} \times \{0,1\}^{28}$ et donnée par la Table 2.5 où on remarque l'absence de 8, 16, ..., 64, ce qui signifie que les bits dans les positions $8, 16, 24 \cdots 64$ sont ignorés tout simplement.

La transformation $PC2$ s'appelle "permuted choice 2", elle est définie $PC2 : \{0,1\}^{28} \times \{0,1\}^{28} \longrightarrow \{0,1\}^{48}$ et donnée par la Table 2.6.

La permutation $PC2$ agit sur un bloc de 56 bits pour produire un bloc de 48 bits. Par exemple, le bit en position 33 est envoyé dans la position 35 en sortie. Les bits dans les positions 9, 18, 22, 25, 35, 38, 43, 54 sont ignorés.

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
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

TABLE 2.5 – Transformation PC1

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

TABLE 2.6 – Transformation PC2

2.2.7 Les rotations circulaires LS_i, $i = 1, \cdots, 16$

LS_i est une rotation circulaire vers la gauche d'une ou deux positions (en fonction de la ronde i) agissants sur une chaîne de 28 bits. Si $i = 1, 2, 9, \text{ou} 16$ on décale d'une position sinon on décale de deux positions.

Par exemple $LS_1(b_1 \cdots b_{28}) = b_2 b_3 \cdots b_{28} b_1$ et $LS_3(b_1 \cdots b_{28}) = b_3 b_4 \cdots b_{28} b_1 b_2$

2.3 Diversification de la clé

Une clef K est une chaîne de 56 bits, à laquelle on ajoute 8 bits de parité. Ils sont des bits de détection d'erreurs. Les bits en positions 8, $16 \cdots$ et 64 sont tels que chaque octet de la clé K contient un nombre impair de 1. Les bits de parité sont ignorés dans le procédé de diversification de la clef.

Ce procédé permet de créer 16 sous clés $K_i, i = 1, \cdots, 16$. Chaque K_i est utilisé dans le i^{eme} tour de fonctionnement de DES.

Chaque clé de tour K_i contient un sous ensemble différent des 56 bits de la clé. Soit K une clef sans bit de parité. On applique $PC1$ à K et on écrit

$$PC1(K) = C_0 D_0$$

où C_0 est formé des 28 premiers bits de $PC1(K)$ et D_0 du reste. puis pour $i = 1, \cdots, 16$ on définit

$$C_i = LS_i(C_{i-1})$$

$$D_i = LS_i(D_{i-1})$$

et les permutations circulaires LS_i vers la gauche permettent de construire les clés de tour $K_i, i = 1, \cdots, 16$.

$$K_i = PC2(C_i D_i), i = 1, \cdots, 16$$

Ainsi, pour chaque tour, on utilise une clé différente $K_i,\ i = 1, \cdots, 16$ de 48 bits obtenue a partir de la clé initiale K de 64 bits. Voir 2.3.

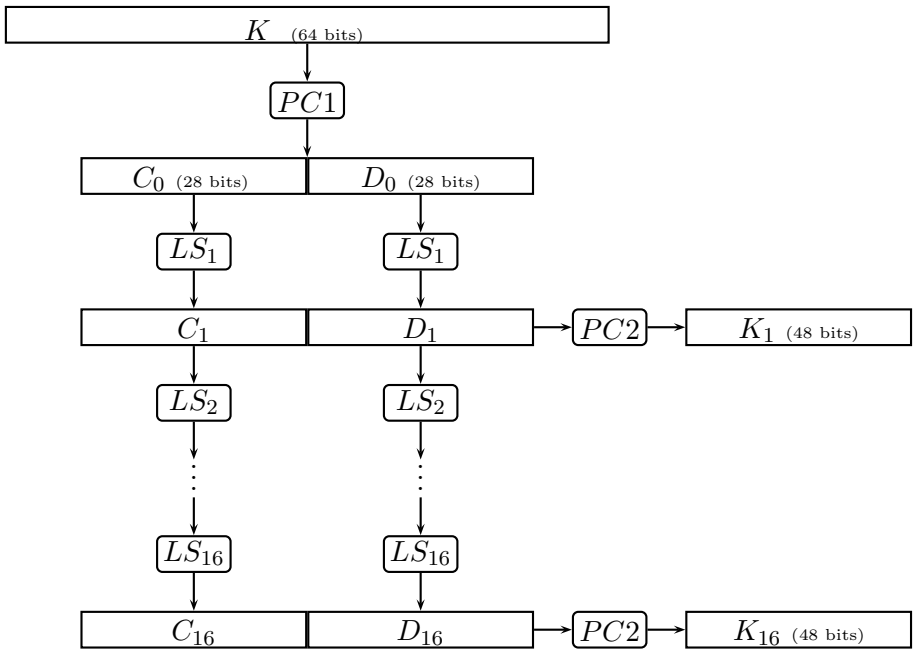


FIGURE 2.3 – Diversification de la clef dans DES

2.4 Étapes de chiffrement

Voir la Figure **??**. Soit Un message M de 64 bits. Une clé K de 56 bits. La sortie est un cryptogramme C de 64 bits. Voir Figure 2.4.

1) On applique la permutation IP à M ensuite on décompose $IP(M)$ en deux mots L_0 (gauche) et R_0 (droite) de 32 bits chacun.

$$IP(M) = L_0 R_0$$

.

2) 16 tours de la fonction f sont exécutées (combinaison de substitutions et de transpositions). Les parties gauches et droites sont modifiées comme il suit pour $i = 1 \cdots 15$

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

. R_{i-1} est de 32 bits et K_i est de 48 bits de la clé de tour la sortie de f est de 32 bits. $L_{16} = R_{15}$, $R_{16} = L_{15} \oplus f(R_{15}, K_{16})$.

3) Enfin, on applique la permutation inverse IP^{-1} à (R_{16}, L_{16}) pour obtenir le texte chiffré C.

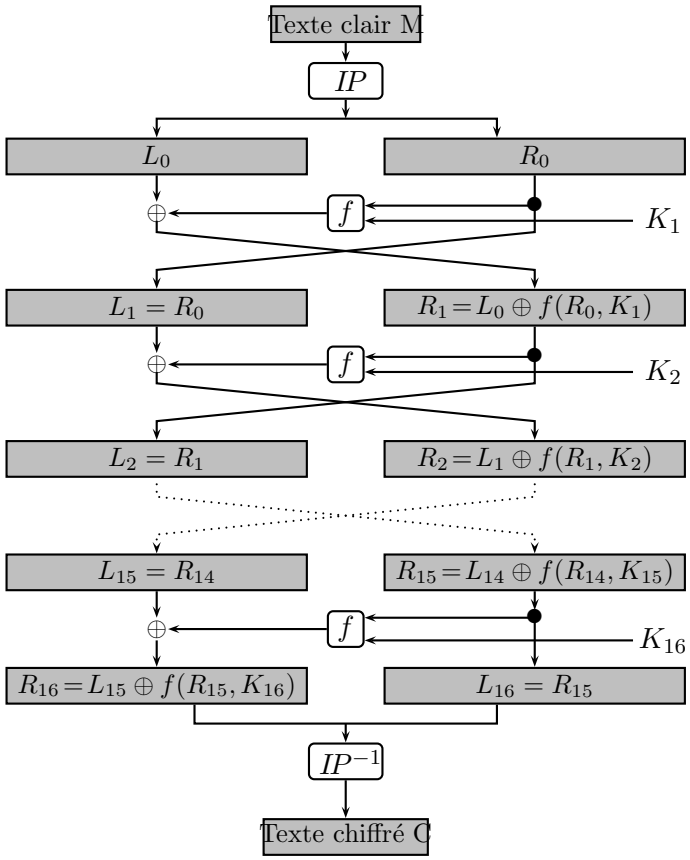


FIGURE 2.4 – L'algorithme DES

2.5 Déchiffrement

Le déchiffrement est effectué par le même algorithme de chiffrement en inversant l'ordre d'utilisation des clés de tour i.e. K_{16} en premier, puis K_{15} etc et enfin K_1. Cela est dû au fait que la permutation finale est l'inverse de la permutation initiale et

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(L_i, K_i)$$

L'algorithme qui engendre les clefs est circulaire, et le décalage se fait vers la droite : si $i = 1, 2, 9, \text{ou} 16$ on décale d'une position sinon on décale de deux positions.

2.6 Controverse

Deux faiblesses principales ont été observées dans la conception du DES :

1) Des clés de 56 bits peuvent être trop courtes pour assurer une robustesse suffisante ;

2) Les principes de choix des S-box n'ont complètement été rendus publique : aucune S_i-Box n'est une fonction linéaire ou affine des entrées. La conception des S-box autoriserait la NSA à effectuer plus rapidement une cryptanalyse. Personne n'a jamais rien trouvé concernant d'éventuelles propriétés cachées des boites de substitution Les critères de constructions des $S - Box$ ne sont pas connus. La critique sur la taille de la clé devient plus pertinente avec l'accroissement de la vitesse des ordinateurs.

DES est facilement implémentable en matériel ou logiciel. DES a été largement utilisé dans le domaine des transactions bancaires et des ministères aux USA. Cryptage de chaînes de télévision à péage. Transmission de données informatiques. Performances Excellentes - cryptage à débits très élevés (dizaine/ centaine de Mégabits/seconde).

2.7 Attaques de DES

- Clefs "faibles" : telles que $E_k(E_k(x)) = x$. Il en existe 4.

- Clefs "semi-faibles" : ce sont les paires de clefs (K_1, K_2) dont la deuxième peut décrypter un message chiffré par la première. Ce sont les clés telles que $E_{K_1}(E_{K_2}(x)) = x$. Il existe six paires de ce genre.

- Clés "pouvant être faibles" : le problème est similaire aux clés semi-faibles. Il en existe 48.

La recherche exhaustive de la clé correspondante parmi les 2^{56} soit environ $7,2.10^{16}$ possibilités.

Un processeur Intel Pentium III à 666 MHz, peut examiner environ deux millions de clés par seconde, ce qui implique un temps de recherche moyen de 600 années pour un seul PC.

EFF (Electronic Frontier Fundation) en 1998 a proposée une solution matérielle dans le seul but de prouver que DES n'est plus du tout sûr. Elle a la possibilité d'examiner 92 milliards de clés par seconde, ce qui donne un temps de recherche moyen situé entre 4 et 5 jours. En 2003, il suffisait de 4 heures et 120.000 euros.

En 1990, Biham et Shamir, ont présenté une nouvelle attaque, appelée cryptanalyse différentielle. C'est la première attaque non exhaustive et qui exige moins de temps que l'attaque exhaustive.

Attaque	paires connues / choisies	Mémoire	Temps
Pré-calcul	1	2^{56}	1
Recherche exhaustive	1	1	2^{55}
Cryptanalyse linéaire	2^{43}	textes	2^{43}
Cryptanalyse différentielle	2^{47}	textes	2^{47}

TABLE 2.7 – Les meilleures attaques connues contre DES

Une autre attaque théorique importante est la cryptanalyse linéaire. Elle a été proposée par Matsui, de Mitsubishi Electronics, en 1993. Bien qu'elle ne soit que théoriquement utile, c'est l'attaque la plus efficace connue à ce jour contre DES.

Il s'est avéré plus tard que les concepteurs de DES savaient ces attaques. [3]

Grâce à la cryptanalyse multi-linéaire, la complexité à été ramené à 2^{39} .

Suite aux failles apparues dans DES, quelques remaniements ont été apportées, mais pas toujours avec grand succès. Ce fut notamment le cas avec le 2DES. Le principe du 2DES est de chiffrer deux fois le message avec deux clefs k_1 et k_2. Il a été prouvé que 2DES était équivalent à un DES avec une clé de 57 bits, c'est tout.

Attaque par l'homme au milieu

Le 2DES est sensible à l'attaque de l'homme au milieu. Un intrus peut s'introduire dans l'échange et retrouver la clé utilisée. Alice transmet $C = f_{K_2}(f_{K_1}(M))$ à Bob. Si l'homme au milieu, Oscar, connaît M et C, il peut construire deux listes de 2^{56} messages

$$L_1 = \{f_K(M); \forall K\} \text{ et } L_2 = \{f_K^{-1}(C); \forall K\}$$

Il cherche ensuite un élément commun. Si $R = f_{K_3}(M) = f_{K_4}^{-1}(C)$, c'est que $f_{K_4}^{-1}(f_{K_4}(M)) = C$. Oscar a alors probablement trouvé $K_3 = K_1$ et $K_4 = K_2$ Ainsi, l'attaque nécessite $X = 2^n$ opérations, et $Y = 2^n$ opérations, soit $2.2^n = 2^{n+1}$.

2.8 3DES

Grâce à 2 clefs, on pratique 3 opérations :

$$E(k_1, D(k_2, E(k_1, m)))$$

C'est équivalent au fait de doubler la taille de la clé (ce qui est une longueur sûre actuellement). Il existe deux versions de 3DES : la première utilise deux clés, la seconde trois (le dernier chiffrement utilise une troisième clé).

Il est robuste contre les attaques connues. Mais, il est très lent que le DES car on triple les opérations.

Modes de chiffrement symétrique Les modes sont des méthodes pour utiliser les chiffrements par blocs. L'algorithme de chiffrement est combiné à une série d'opérations simples en vue d'améliorer sa sécurité et/ou de l'adapter à des utilisations précises.

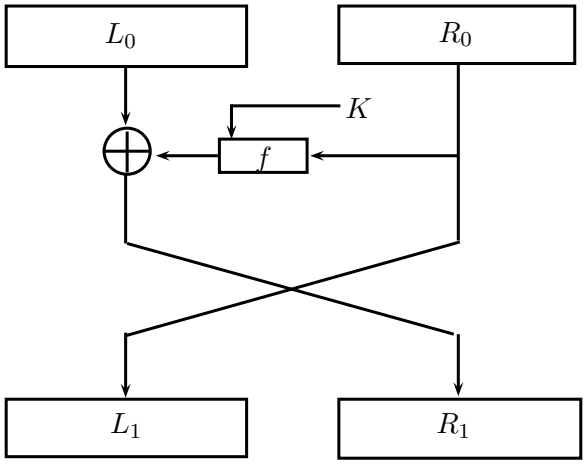


FIGURE 2.5 – Schéma Feistel

2.9 Schéma de Feistel

Feistel, ingénieur chez IBM a dirigé l'équipe qui a conçut DES.

Le schéma qui porte son nom a été décrit en 1973. Voir la Figure 2.5.

obtenir une bijection sur $2n$ bits à partir d'une fonction non-bijection sur n-bits.

Chiffrement : $L_1 = R_0$, $R_1 = L_0 \oplus f(R_0)$

Déchiffrement : $R_0 = L_1$, $L_0 = R_1 \oplus f(R_0)$

La fonction f du schéma de Feistel s'appelle fonction de confusion.

La plupart des algorithmes à clé secrète de la fin du XXè siècle sont basés sur le schémas de Feistel. Par exemple : DES, Blowfish, Twofish, Camellia, SEED, RC5, OAEP, etc

Il faut que le message chiffré soit aussi aléatoire que possible.

Le schéma de Feistel permet de construire des bijections aléatoires.

Plusieurs attaques sont possibles sur le schéma de Feistel. Les deux principales sont : la cryptanalyse linéaire et la cryptanalyse différentielle. Ces méthodes ont fait leur preuve sur DES et sur d'autres algorithmes similaires.

2.10 Exercices

Exercice 1. Calculer $S_i(110101)$ où S_i est la i^{eme} S-Box du DES et i= 5,8.

Exercice 2. Expliquer pourquoi dans le DES on a la propriété de complémentation : pour tout M et toute clé k : $DES_k(M) = \overline{DES_{\overline{k}}(\overline{M})}$

Exercice 3. On considère un diagramme de Feistel à deux rondes sur des chaînes de 8 bits avec deux fonctions f_1 et f_2.

1. On pose $f_1(a) := a \oplus 1011$ et $f_2(a) := \bar{a} \oplus 0101$ pour toute chaîne a de 4 bits.

(a) Calculer l'image de la chaîne 11010011 par ce diagramme.

(b) Déterminer une chaîne de 8 bits dont l'image par le diagramme est elle-même.

2. La propriété précédente, à savoir il existe une chaîne dont l'image par le diagramme de Feistel est elle-même, est-elle vraie pour toutes les fonctions f_1 et f_2 ? Justifier votre réponse par une démonstration ou un contre-exemple.

Exercice 4. On utilise pour chiffrer ses données privées le cryptosystème DES, paramétré par une clé secrète k de 56 bits. Comme 56 bits est bien peu de nos jours, on envisage de rendre plus sûr le stockage de ses données en chiffrant une seconde fois toutes ses données, avec la clé DES $k' = k + 1$ (pour chaque donnée en clair m, la donnée chiffrée est donc $c = DES_{k+1}(DES_k(m))$, où k désigne la clé).

1. Est-ce une bonne idée ?

2. Discuter les avantages et/ou les inconvénients.

3. On pense à une autre amélioration possible. On va chiffrer une fois avec DES, et une fois avec AES128. Comme AES128 a besoin de clés de 128 bits, on va paramétrer son chiffrement DES par sa clé secrète k, et pour son chiffrement AES128 la même clé secrète k, mais avec des zéros pour faire le remplissage. Est-ce mieux ?

4. Quelle erreur fondamentale commet-on, eu égard aux principes de Kerckhoffs ?

Bibliographie

[1] Bruce Schneier, Cryptographie appliquée, Vuibert, 2001.

[2] Douglas Stinson, Cryptographie : Théorie et pratique, 2eme Édition, Vuibert, 2003.

[3] bibitemKnud94 Knudsen, L.R., Block Ciphers-Analysis, Designs and Applications, PhD Thesis, Computer Science Department, Aarhus University, Denmark, 1994.

[4] Christof Paar and Jan Pelzl, Understanding Cryptography, Springer-Verlag 2010.

Chapitre 3

IDEA

Sommaire

3.1 Introduction

IDEA (International Data Encryption Algorithm) est un système de chiffrement symmétrique, par blocs de 64 bits avec une clé de 128 bits, qui tourne sur 8 tours et une transformation finale. IDEA et basé sur la fusion et la confusion. Mis au point par Xuejia Lai and James L. Massey [1, 2] en 1992 et proposé pour remplacer DES. La vitesse de IDEA avoisine celle de DES. IDEA a couronné deux versions : la première PES (Proposed Encryption Standard) de 1990 renforcée pour résister à la cryptanalyse différentielle (1991) pour obtenir la deuxième version IPES (Improved Proposed Encryption Standard) de 1991.

Pendant quelques années IDEA a servi de standard en remplacement à DES dans certaines applications. Il a été aussi utilisé par PGP (Pretty Goog Privacy), qui l'a rendu célèbre, pour sécuriser les courriers e-mails et par openSSL (Secure Socket Layer) pour sécuriser le trafic web.

Le même algorithme est utilisé à la fois pour chiffrer et déchiffrer. IDEA est breveté dans plusieurs pays, Il est commercialisé par la société suisse MediaCrypt. Mais libre d'utilisation à des fins non commerciales par tout dans le monde. Les droits d'exploitation sont détenus par Ascom Systec AG. www.ascom.com.

Contrairement au DES, IDEA n'utilise pas de S-Box et n'utilise pas le schéma de Feistel mais une autre méthode permettant de produire des fonctions inversibles, propriété essentielle pour le déchiffrement. Il répond aux exigences de diffusion et de confusion.

IDEA est un algorithme de chiffrement symétrique par blocs de 64 bits, se fait en 8 tours et une transformation finale et utilisant 52 sous-clefs de la clef initiale de 128 bits.

Le processus de dérivation de sous-clé en produit 52 de 96 bits chacune.

Il est facilement réalisable en matériel ou en logiciel. Les opérations utilisées dans IDEA sont des opérations simples de trois groupes algébriques :

1. le ou exclusif (xor) noté $\oplus$;
2. l'addition modulo $2^{16} = 65536$ notée $\boxplus$;
3. la multiplication modulo $2^{16} + 1 = 65537$, (qui est un nombre premier). notée $\odot$. Pour cette multiplication un bloc de 16 bits dont tous les bits sont à 0 est interprété plutôt comme 2^{16} et non comme 0.

Ces opérations manipulent des sous-blocs de 16 bits.

- IDEA était considéré par les spécialistes comme l'un des meilleurs cryptosystème à clé secrète, car la longueur de sa clé est élevée (128 bits) et la vitesse de chiffrement et de déchiffrement peut-être élevée au moyen de circuits spéciaux.

3.2 Séquencement de la clé

IDEA utilise une clef de 128 bits qui sert pour créer 52 sous-clefs, 6 pour chacun des 8 tours et 4 pour la transformation finale.

La clé $K = b_1 b_2 \cdots b_{128}$ de 128 bits est divisée en 8 blocs de 16 bits :

$$\underbrace{b_1 b_2 \cdots b_{16}}_{K_1^{(1)}} \underbrace{b_{17} \cdots b_{32}}_{K_2^{(1)}} \cdots \underbrace{b_{81} \cdots b_{96}}_{K_6^{(1)}} \underbrace{b_{97} \cdots b_{112}}_{K_1^{(2)}} \underbrace{b_{113} \cdots b_{128}}_{K_2^{(2)}}$$

Ce sont les 8 premières sous-clefs de IDEA : 6 pour le premier tour et les 2 qui restent sont les premières sous-clefs du 2e tour. La clé K est ensuite décalées circulairement vers la gauche de 25 bits, puis divisée, en 8 sous-clefs :

$$\underbrace{b_{26} b_{27} \cdots b_{41}}_{K_3^{(2)}} \cdots \underbrace{b_{42} \cdots b_{57}}_{K_6^{(2)}} \underbrace{b_{58} \cdots b_{73}}_{K_1^{(3)}} \underbrace{b_{74} \cdots b_{89}}_{K_2^{(3)}}$$

les 4 premières sont utilisées dans le 2e tour et les 4 qui restent dans le 3e tour.

La clé est décalée vers la gauche de 25 bits et ainsi de suite jusqu'à obtenir 52 clés. Ces clés forment 8 groupes de 6 sous-clés (un groupe par tour) : $K_1^{(i)}$, $K_2^{(i)}$, $K_3^{(i)}$, $K_4^{(i)}$, $K_5^{(i)}$, $K_6^{(i)}$, $i = 1, \cdots, 8$

et un groupe de 4 clés pour la transformation finale : $K_1^{(9)}$, $K_2^{(9)}$, $K_3^{(9)}$, $K_4^{(9)}$.

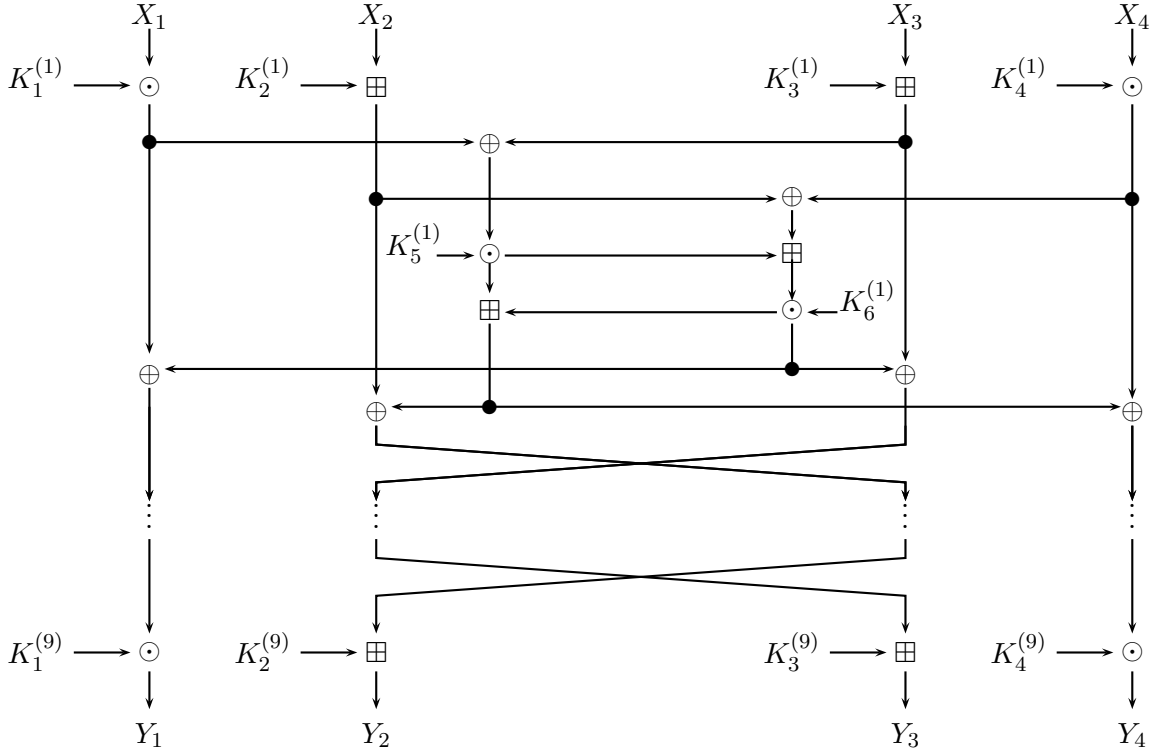
Voir la Table des sous-clés de chiffrement.

3.3 Description de IDEA

Le texte clair à chiffrer est découpé en blocs de 64 bits. Chaque bloc est divisé en quatre sous-blocs de 16 bits : X_1, X_2, X_3, X_4. Ce sont les entrées du premièr tour de l'algorithme IDEA. L'algorithme s'effectue en 8 tours. Voir l'algorithme 3.1.

tour	Sous clefs de chiffrement					
1	$K_1^{(1)}$	$K_2^{(1)}$	$K_3^{(1)}$	$K_4^{(1)}$	$K_5^{(1)}$	$K_6^{(1)}$
2	$K_1^{(2)}$	$K_2^{(2)}$	$K_3^{(2)}$	$K_4^{(2)}$	$K_5^{(2)}$	$K_6^{(2)}$
3	$K_1^{(3)}$	$K_2^{(3)}$	$K_3^{(3)}$	$K_4^{(3)}$	$K_5^{(3)}$	$K_6^{(3)}$
4	$K_1^{(4)}$	$K_2^{(4)}$	$K_3^{(4)}$	$K_4^{(4)}$	$K_5^{(4)}$	$K_6^{(4)}$
5	$K_1^{(5)}$	$K_2^{(5)}$	$K_3^{(5)}$	$K_4^{(5)}$	$K_5^{(5)}$	$K_6^{(5)}$
6	$K_1^{(6)}$	$K_2^{(6)}$	$K_3^{(6)}$	$K_4^{(6)}$	$K_5^{(6)}$	$K_6^{(6)}$
7	$K_1^{(7)}$	$K_2^{(7)}$	$K_3^{(7)}$	$K_4^{(7)}$	$K_5^{(7)}$	$K_6^{(7)}$
8	$K_1^{(8)}$	$K_2^{(8)}$	$K_3^{(8)}$	$K_4^{(8)}$	$K_5^{(8)}$	$K_6^{(8)}$
finale	$K_1^{(9)}$	$K_2^{(9)}$	$K_3^{(9)}$	$K_4^{(9)}$		

TABLE 3.1 – Sous clefs de chiffrement



X_i : sous bloc de 16 bits de texte clair
Y_i : sous bloc de 16 bits de texte chiffré
$K_i^{(j)}$: sous bloc de 16 bits de la clé
$\oplus$: ou exclusif bit à bit
$\boxplus$: addition modulo 2^{16} d'entiers de 16 bits
$\odot$: multiplication modulo $2^{16}+1$ d'entiers de 16 bits

FIGURE 3.1 – Description de l'algorithme IDEA

A chaque tour, les X_i sont combinés par les opérations $\oplus$, $\boxplus$, $\odot$ entre eux et avec les six mots de 16 bits de la clef.

À chaque tour la suite d'opérations est la suivante :

1. Etape1 = $X_1 \odot K_1^{(i)}$
2. Etape2 = $X_2 \boxplus K_2^{(i)}$
3. Etape3 = $X_3 \boxplus K_3^{(i)}$
4. Etape4 = $X_4 \odot K_4^{(i)}$
5. Etape5 = Etape1 $\oplus$ Etape3
6. Etape6 = Etape2 $\oplus$ Etape4
7. Etape7 = Etape5 $\odot K_5^{(i)}$
8. Etape8 = Etape6 $\boxplus$ Etape7
9. Etape9 = Etape8 $\odot K_6^{(i)}$
10. Etape10 = Etape7 $\boxplus$ Etape9
11. Etape11 = Etape1 $\oplus$ Etape9 $\Rightarrow X_1$ du tour suivant
12. Etape12 = Etape3 $\oplus$ Etape9 $\Rightarrow X_3$ du tour suivant
13. Etape13 = Etape2 $\oplus$ Etape10 $\Rightarrow X_2$ du tour suivant
14. Etape14 = Etape4 $\oplus$ Etape10 $\Rightarrow X_4$ du tour suivant

Les deux blocs intérieurs X_3 et X_4 obtenus sont échangés, sauf lors du dernier tour..

Pour finir, après le huitième tour, on applique une étape supplémentaire :
$Y_1 = X_1 \odot K_1^{(9)}$, $Y_2 = X_2 \boxplus K_2^{(9)}$, $Y_3 = X_3 \boxplus K_3^{(9)}$, $Y_4 = X_4 \odot K_4^{(9)}$.
Les 4 blocs Y_1, Y_2, Y_3, Y_4, forment alors le message chiffré.

3.4 Déchiffrement

Pour déchiffrer le texte, il faut d'abord inverser la dernière opération :
$Y_1 = Y_1 \odot K_1^{-1}$, $Y_2 = Y_2 - K_2$, $Y_3 = Y_3 - K_3$, , $Y_4 = Y_4 \odot K_4^{-1}$.

Les sous clefs de déchiffrement sont inverses par rapport à l'addition ou par rapport à la multiplication des sous clefs de chiffrement. (pour les besoins de IDEA, tous les sous blocs constitués uniquement de zéros représentent $2^{16} = -1$ modulo $2^{16} + 1$ pour la multiplication et l'inverse de la multiplication de zéro est donc zéro). Le calcul de ces inverses prend du temps mais on le fait qu'une fois par clef de (dé)chiffrement. Voir la Table 3.2 des sous-clefs de déchiffrement.

On applique alors les opérations suivantes selon 8 tours, en utilisant les groupes de 6 clés en partant de la dernière à la première :

1. Etape1 = $C_1 \oplus C_3$ (Etape5 lors du chiffrement)
2. Etape2 = $C_2 \oplus C_4$ (Etape6 lors du chiffrement)
3. Etape3 = Etape1 $\odot K_5$ (Etape7 lors du chiffrement)
4. Etape4 = Etape2 + Etape3 (Etape8 lors du chiffrement)

tour	Sous clefs de déchiffrement					
1	$K_1^{(9)^{-1}}$	$-K_2^{(9)}$	$-K_3^{(9)}$	$K_4^{(9)^{-1}}$	$K_5^{(8)}$	$K_6^{(8)}$
2	$K_1^{(8)^{-1}}$	$-K_3^{(8)}$	$-K_2^{(8)}$	$K_4^{(8)^{-1}}$	$K_5^{(7)}$	$K_6^{(7)}$
3	$K_1^{(7)^{-1}}$	$-K_3^{(7)}$	$-K_2^{(7)}$	$K_4^{(7)^{-1}}$	$K_5^{(6)}$	$K_6^{(6)}$
4	$K_1^{(6)^{-1}}$	$-K_3^{(6)}$	$-K_2^{(6)}$	$K_4^{(6)^{-1}}$	$K_5^{(5)}$	$K_6^{(5)}$
5	$K_1^{(5)^{-1}}$	$-K_3^{(5)}$	$-K_2^{(5)}$	$K_4^{(5)^{-1}}$	$K_5^{(4)}$	$K_6^{(4)}$
6	$K_1^{(4)^{-1}}$	$-K_3^{(4)}$	$-K_2^{(4)}$	$K_4^{(4)^{-1}}$	$K_5^{(3)}$	$K_6^{(3)}$
7	$K_1^{(3)^{-1}}$	$-K_3^{(3)}$	$-K_2^{(3)}$	$K_4^{(3)^{-1}}$	$K_5^{(2)}$	$K_6^{(2)}$
8	$K_1^{(2)^{-1}}$	$-K_3^{(2)}$	$-K_2^{(2)}$	$K_4^{(2)^{-1}}$	$K_5^{(1)}$	$K_6^{(1)}$
Finale	$K_1^{(1)^{-1}}$	$-K_2^{(1)}$	$-K_3^{(1)}$	$K_4^{(1)^{-1}}$		

TABLE 3.2 – Sous clefs de déchiffrement

5. Etape5 = Etape4 $\odot$ K_6 (Etape9 lors du chiffrement)
6. Etape6 = Etape3 + Etape5 (Etape10 lors du chiffrement)
7. Etape7 = $C_1 \oplus$ Etape5 (Etape1 lors du chiffrement)
8. Etape8 = $C_3 \oplus$ Etape5 (Etape3 lors du chiffrement)
9. Etape9 = $C_2 \oplus$ Etape6 (Etape2 lors du chiffrement)
10. Etape10 = $C_4 \oplus$ Etape6 (Etape4 lors du chiffrement)
11. Etape11 = Etape7 $\odot$ $K_1^{-1} \Rightarrow C_1$ du tour suivant
12. Etape12 = Etape8 $-$ $K_3 \Rightarrow C_3$ du tour suivant
13. Etape13 = Etape9 $-$ $K_2 \Rightarrow C_2$ du tour suivant
14. Etape14 = Etape10 $\odot$ $K_4^{-1} \Rightarrow C_4$ du tour suivant

Les 4 blocs C_1, C_2, C_3, C_4, obtenus après le dernier tour forment alors le message en clair.

3.5 Sécurité de IDEA

La sécurité de IDEA dépend de la confusion et de la diffusion. La confusion est réalisée par le mixage des trois opérations incompatibles. En effet aucune paire des trois opérations ne satisfait ni la loi de distribution

$$a \boxplus (b \odot c) \neq (a \boxplus b) \odot (a \boxplus c)$$

ni la loi d'associativité généralisée

$$a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c$$

Aussi, les trois opérations sont choisies de façon que le résultat d'une opération n'est jamais utilisé comme entrée d'une opération de même type. (i.e. un résultat de l'opération $\oplus$ ne peut être utilisé dans une autre opération $\oplus$.)

La diffusion est réalisée par le fait que chaque sous-bloc résultant dépend de tous les sous blocs d'entrée et que le nombre minimum d'opérations utilisé dans chaque structure d'addition multiple est de quatre. De plus chaque entrée et sortie d'une telle structure est une transformation inversible.

IDEA a une clef de 128 bits. L'attaque exhaustive exige 2^{128} soit 10^{38} tests. Si on a des processeurs qui testent 10^9 clefs par seconde et en utilisant 10^9 ordinateurs munis de ces processeurs en paralèlle il faudrait 10^{13} années, c'est beaucoup plus que l'age de l'univers. Il faudrait 10^{24} ordinateurs de ce type pour trouver la clef en 24 h. Mais il n'y a pas assez d'atome de silicium dans l'univers pour construire toutes ces machines.

Mais alte à d'autres algorithme de cyptanalyse ou d'autres techniques.

Il existe une classe de clefs faibles, qu'un cryptanaliste peut identifier par une attaque à clair choisi. En héxadécimal une classe de telles clefs est

```
0000  0000  0X00  0000  0000  000X  XXXX  X000
```

X représente n'importe quelle valeur héxadécimale. Mais la probabilité d'engendrer une telle clef aléatoirement est de 2^{-96}. Très faible.

Mise à part cette faiblesse de clef, jusqu'en 2002 aucune attaque plus rapide que l'attaque exhaustive n'a été publiée.

En 1996 Bruce Schneier a pensé que "IDEA est le meilleur algorithme public de chiffrement par bloc" et le plus sécurisé. Mais en 1999 il ne recommandait plus IDEA à cause de certains progrès dans sa cryptanalyse d'une par, de sa nature brevetée et de la disponibilité de nouveau algorithmes d'autre part.

Bibliographie

[1] LAI X., MASSEY J. L., A proposal for a new block encryption standard, Proc. EUROCRYPT, vol. 90, p. 389-404, Springer, 1990.

[2] LAI X., MASSEY J. L., MURPHY S., Markov ciphers and differential cryptanalysis, Advances in Cryptology - Eurocrypt, vol. 91, p. 17-38, Springer, 1992.

Chapitre 4

AES : Advanced Encryption Standard

Sommaire

Version officiele de l'AES est disponible sur [1].

En 1997 un appel d'offre du NIST (National Institute of Standards and technology) est lancé pour réaliser un cryptosystème appelé AES (Advanced Encryption Standard) destiné à remplacer le DES. En 1998, suite à une première sélection, 15 projets ont été retenus et après une deuxième sélection, ils ne sont plus que cinq :

1. MARS (IBM)
2. RC6 (Laboratoires RSA)
3. Rijndael (J. Demen et V. Rijmen)
4. Serpent (E. Biham et al.)
5. Twofish (B. Schneier et al.)

En 2000 le projet Rijndael (prononcer Raindal) est déclaré vainqueur par le NIST, il devient alors l'AES, donc le successeur du DES. Rijndaelest conçu par Joan Daemen, et Vincent Rijmen, deux chercheurs Belges, docteurs de l'université de K. U. Leuven en 1995 et 1997 respectivement.

Rijndael est un système de chiffrement symmétrique par blocs. Rijndael n'utilise pas le schéma de Fiestel. La longueur de bloc à chiffrer est variable, elle peut être de 128, 192 ou 256 bits. La clé est aussi de longueur variable, 128, 192 ou 256 bits. Rijndael est réalisé en 10, 12 ou 14 tours (respectivement) selon la longueur de la clé.

Par contre, le système de chiffrement standard AES retient uniquement la longueur de bloc qui est fixée à 128 bits et utilise des longueurs de clé variables, 128, 192 ou 256 bits.

IL est facile d'implémenter l'AES aussi bien sous forme logicielle que matérielle.

Trois critères principaux ont été respectés dans sa conception : Résistance face à toutes les attaques connues notamment les attaques différentielle et linéaire , rapidité et simplicité dans la conception. L'AES est un standard, libre d'utilisation, non breveté.

Parmi les standards commerciaux utilisant l'AES : les standards de sécurité de l'Internet IPsec, TLS, Wi-Fi IEEE 802.11i, le protocole SSH, le téléphone par Internet Skype et d'autres. Actuellement aucune attaque n'est connue hors l'attaque exhaustive.

En 2003, le NSA (National Security Agency) a autorisé l'utilisation de l'AES pour chiffrer les documents classés niveau "SECRET" avec une clé de n'importe quelle longueur, et les documents classés "TOP SECRET" avec une clé de longueur 192 ou 256 bits.

4.1 Outils mathématiques

La description de cet algorithme utilise le corps de Galois $\mathbb{F}_{2^8}$ où l'essentiel des calculs est effectué.

Rappelons que si p est un nombre premier et r un entier positif, alors il existe un corps fini d'ordre p^r donné par

$$\mathbb{F}_{p^r} = \{a_0 + a_1 t + \cdots + a_{r-1} t^{r-1} / a_0, a_1, \cdots, a_{r-1} \in \mathbb{Z}_p\} = \mathbb{Z}_p[x]/(P(x))$$

où $P(x) \in \mathbb{Z}_p[x]$ est un polynôme irréductible unitaire de degré r et t vérifie $P(t) = 0$.

L'AES utilise le corps de Galois $\mathbb{F}_{2^8}$ défini par le polynôme

$$P(x) = x^8 + x^4 + x^3 + x + 1 \in \mathbb{Z}_2[x] \tag{4.1}$$

La multiplication dans $\mathbb{F}_{p^r}$ est la multiplication usuelle de polynômes modulo $P(x)$.

4.1.1 Représentation polynomiale et hexadécimale des octets

Un octet $B = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$, est identifié au polynôme

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 \in \mathbb{F}_{2^8} = \mathbb{Z}_2[x]/(P(x)) \tag{4.2}$$

L'octet B s'écrit aussi en format hexadécimale en écrivant la valeur hexadécimale des 4 premiers bits suivies de celles des 4 derniers bits. Inversement, le passage de l'écriture hexadécimale en binaire consiste à écrire chaque chiffre hexadécimal sur 4 bits.

Les opérations de L'AES sont effectuées dans le corps $\mathbb{F}_{2^8}$ ci-dessus dont on identifie ses polynômes aux nombres hexadécimaux qui sont, à leur tour, convertis en nombres binaires.

Par exemple : `0xC3= 11000011` représente le polynôme x^7+x^6+x+1 et `0xE0= 11100000` représente le polynôme $x^8 + x^7 + x^6$ et on a `0xC3` $\oplus$ `0xE0=23=100011`.

La somme de polynômes revient à xorer les coefficients de même monôme.

Important : dans la suite les nombres hexadécimaux, les octets doivent être perçus comme des polynômes.

Par exemple le polynôme $P(x)$ ci-dessus s'écrit en hexadécimal `0x11B`.

L'ensemble des 256 octets muni de l'opération $\oplus$ et de la multiplication modulo $P(x)$ est un corps isomorphe à $\mathbb{F}_{2^8}$.

Étapes de l'algorithme : la figure montre les différentes étapes de l'algorithme AES. Il est formé d'un tour initial, puis de tours standards et d'un tour final. Quatre opérations différentes sont nécessaires pour réaliser ces rondes et la diversification de la clef.

Représentation polynomiale des mots (1 mot=4 octets) Auparavant pour représenter des octets nous avons utilisé les polynômes de $\mathbb{Z}_2[x]$. Maintenant pour représenter un mot (4 octets ou 32 bits) nous utilisons un polynôme de degré au plus 3 à coefficients dans l'anneau $\mathbb{F}_{2^8} = \mathbb{Z}_2[x]/(x^4 + 1)$. L'addition est usuelle.

La multiplication est (modulaire) effectuée modulo $x^4 + 1$.

Le polynôme $x^4 + 1$ n'est pas irréductible dans $\mathbb{F}_{2^8}[x]$, et la multiplication par $a(x)$ n'est pas nécessairement inversible. Mais le polynôme $a(x)$ est inversible.

4.2 Présentation du bloc à chiffrer et de la clé

L'AES chiffre par bloc de 128 bits. Soit $b_0b_1b_2\cdots a_{127}$ un tel bloc. On le découpe en octets et on le note $B_0B_1B_2\cdots B_{15}$ On appelle état, la présentation d'un tel bloc sous forme de matrice à 4 lignes.

$$\begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{bmatrix}$$

Soit $k_0k_1\cdots k_n$ une clé de longueur 128, 192 ou 256 bits. On la découpe en octets et on la note $K_0K_1K_2\cdots K_N$ où $N = 15, 23$ ou 31 la présentation d'un tel bloc sous forme d'une matrice toujours de 4 lignes et de $N_c = (longueurdublocenbits)/32 = 4, 6$ ou 8 colonnes (respectivement). Par exemple, une clé de 128 bits est représentée par la matrice suivante où $N_c = 4$

$$\begin{bmatrix} K_0 & K_4 & K_8 & K_{12} \\ K_1 & K_5 & K_9 & K_{13} \\ K_2 & K_6 & K_{10} & K_{14} \\ K_3 & K_7 & K_{11} & K_{15} \end{bmatrix}$$

	nombre de colonne	nombre de tour
clé de 128	4	10
clé de 192	6	12
clé de 256	8	14

TABLE 4.1 – Nombre de tour de l'AES pour chaque longueur de clé

une clé de 192 bits est représentée par la matrice suivante où $N_c = 6$

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

Si on note par N_r le nombre de tour à effectuer par L'AES pour chiffrer, on a les possibilités suivantes :

Exemple

bloc	F6	12	A8	98	05	28	20	7A	E0	5A	24	F6	88	8D	35	32
clef	21	A0	22	07	E0	08	05	F3	20	C2	01	B1	04	D3	A8	19

$$\text{bloc}=\begin{bmatrix} F6 & 05 & E0 & 88 \\ 12 & 28 & 5A & 8D \\ A8 & 20 & 24 & 35 \\ 98 & 7A & F6 & 32 \end{bmatrix} \qquad \text{clef}=\begin{bmatrix} 21 & E0 & 20 & 04 \\ A0 & 08 & C3 & D3 \\ 22 & 05 & 01 & A8 \\ 07 & F3 & B1 & 19 \end{bmatrix}$$

On a $N_k = N_b = 4$

4.3 Opération de l'algorithme AES

Le système de chiffrement AES opère sur les états (matrice à 4 lignes et $N_c = longueur(bloc)/32$ colonnes). Le chiffrement consiste en une addition initiale de clé appelé `AddRoundKey` suivie de $N_r - 1$ tours et chaque tour est formé de 4 étapes :

1. `SubBytes` : substitution d'octets par d'autres choisis dans une boite S-Box ;
2. `ShiftRows` : transposition, chaque terme de la matrice est décalé cycliquement à gauche d'un certain nombres de colonnes ;
3. `MixColomns` : produit matriciel sur chaque colonne (pris comme vecteur) de la matrice ;
4. `AddRoundKey` : combine par addition chaque octet avec l'octet correspondant dans une clé de ronde obtenue par diversification de la clé de chiffrement.

Enfin, une ronde finale nommée `FinalRound` est appliquée (c'est une ronde sans `MixColomns`)

4.3.1 L'opération `AddRoundKey`

C'est une simple opération XOR terme à terme dans $\mathbb{F}_{2^8}$ entre la matrice état (state) et la clef de la ronde.

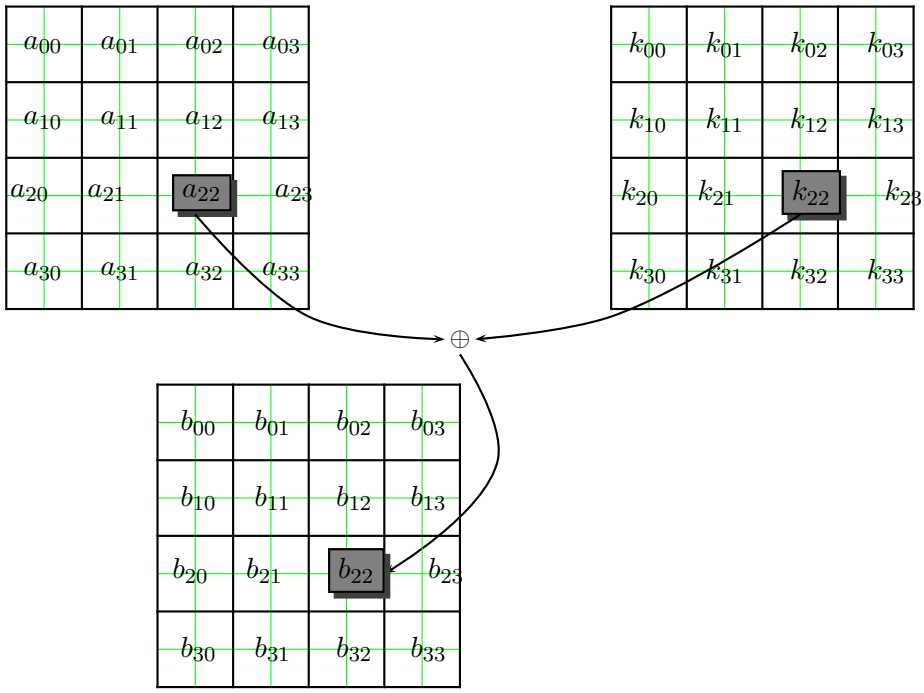


FIGURE 4.1 – L'opération `AddRoundKey`

4.3.2 L'opération SubBytes

Dans cette opération chaque octet des sous-blocs est substitué selon la table S-Box. Voir Figure 4.2 Cette opération augmente la non-linéarité des données. Elle agit sur chaque état.

Elle est similaire aux $S-box$ utilisées dans l'algorithme DES. AES a un seul $S-box$ qui est donné. Il est connu pour résister à la cryptanalyse linéaire et différentielle connues.

Contairement aux S-Box de DES, la conception de S-box de l'AES est publique : celle-ci est inversible, et construite par la composition de deux transformations :
1) l'inverse multilicatif de chaque élément est calculé dans $\mathbb{F}_{2^8}$ ('00' est son propre inverse par convention) ; voir la table des inverses 4.3. L'inverse de l'octet xy est la valeur du tableau se trouvant à l'intersection de la ligne x et de la colonne y.
2) puis on applique la transformation affine suivante sur chaque octet résultant de l'opération précédente

S-Box est la composée $S = f \circ I$ des applications $I : \mathbb{F}_{2^8} \longrightarrow \mathbb{F}_{2^8}$ $I(0) = 0$ et $I(x) = x^{-1}$ pour $x \neq 0$ et la fonction affine : $f : (\mathbb{F}_{2^8})^8 \longrightarrow (\mathbb{F}_{2^8})^8$ $f(x) = Ax + B$ donnée matricielement par

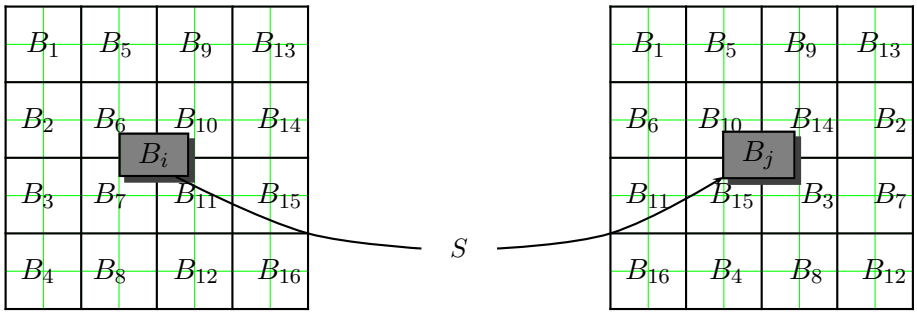


FIGURE 4.2 – Fonction Sub-Bytes

$$f(x_0, x_1, \ldots, x_7) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \mod 2$$

Si **x** et **y** représentent respèctivement le premier et le deuxième unité d'un octet hexadécimal d'un état alors la transformée par `SubBytes` du nombre **xy** est le nombre se trouvant à l'intersection de la ligne **x** et de la colonne **y** de la table S-Box.

Par exemple pour 'd9' : $\mathbf{x} = d$ et $\mathbf{y} = 9$.

Exemple d'action de *SubBytes* Soit l'état

$$s = \begin{bmatrix} F6 & 05 & E0 & 88 \\ 12 & 28 & 5A & 8D \\ A8 & 20 & 24 & 35 \\ 98 & 7A & F6 & 32 \end{bmatrix} \text{ alors } S - Box(s) = \begin{bmatrix} 42 & 6B & E1 & C4 \\ C9 & 34 & BE & 5D \\ C2 & B7 & 36 & 96 \\ 46 & DA & 42 & 23 \end{bmatrix}$$

4.3.3 Opération `ShiftRows`

Le rôle de cette opération est d'augmenter la diffusion. Sous l'action de cette opération chaque ligne l_i, $i = 0, \cdots, 3$ d'un état, est circulairement déplacée vers la gauche de c_i cases $i = 0, \cdots, 3$ dépendant de la longueur N_b du bloc d'entrée et donné par :

Décalage des lignes dans `ShiftRows` en fonction de N_b

Cette opération augmente la diffusion des données dans le ronde en séparant les octets à l'origine consécutif.

Exemple d'action de `ShiftRows`

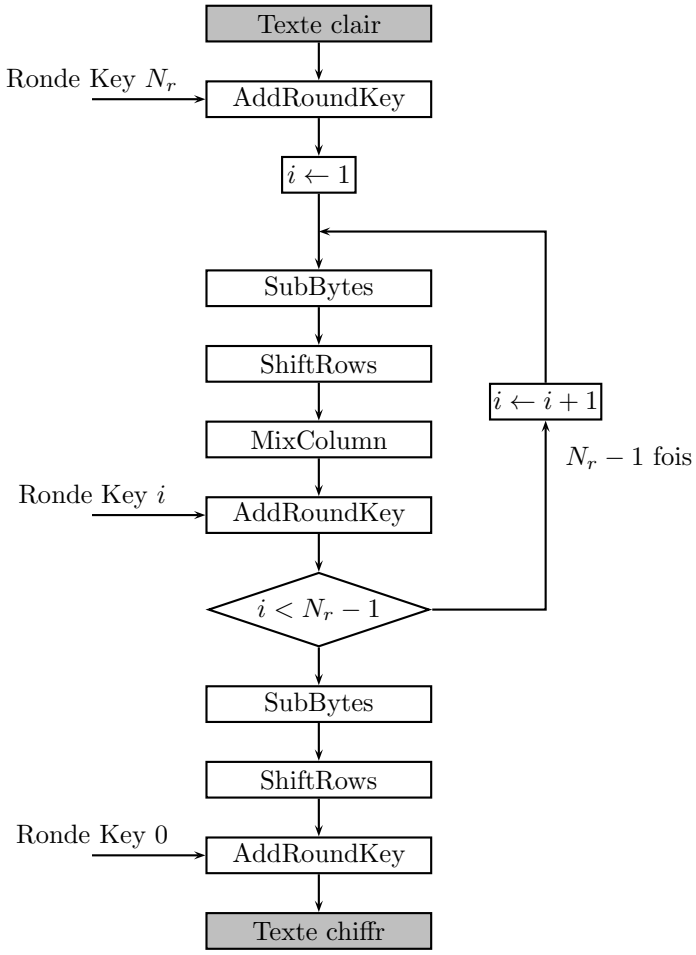


FIGURE 4.3 – L'algorithme AES

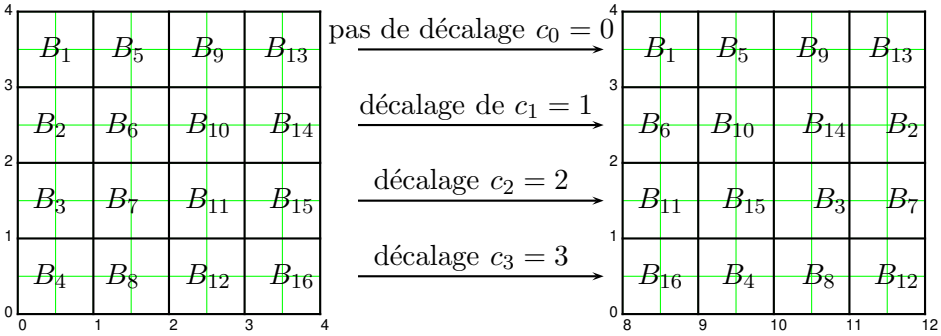


FIGURE 4.4 – `ShiftRows` dans le cas Nb=4 et $(c_0, c_1, c_2, c_3) = (0, 1, 2, 3)$

hex		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
x	**0**	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

TABLE 4.2 – S-Box de l'algorithme AES

hex		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**y**	**c**	**d**	**e**	**f**
x	**0**	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

TABLE 4.3 – `S-Box` inverse de l'algorithme AES

N_b	c_0	c_1	c_2	c_3
4	0	1	2	3
6	0	1	2	3
8	0	1	3	4

TABLE 4.4 – Nombre de décalages en fonction de N_b

$$\text{s} = \begin{bmatrix} F6 & 05 & E0 & 88 \\ 12 & 28 & 5A & 8D \\ A8 & 20 & 24 & 35 \\ 98 & 7A & F6 & 32 \end{bmatrix} \quad \text{alors} \quad \text{s'} = \begin{bmatrix} F6 & 05 & E0 & 88 \\ 28 & 5A & 8D & 12 \\ 24 & 35 & A8 & 20 \\ 32 & 98 & 7A & F6 \end{bmatrix}$$

4.3.4 Opération `MixColumns`

La combinaison de `ShiftRows` et `MixColumns` fait qu'après seulement 3 tours chaque octet dépend de tous les 16 octets du texte clair.

C'est une opération linéaire sur chaque colonne d'un état qui est considérée comme un polynômes à coefficients dans $\mathbb{F}_{2^8}$. Chaque colonne, vue comme polynôme, est multipliée (multilication modulaire) par le polynôme

$$a(x) = 03_H x^3 + 01_H x^2 + 01_H x + 02_H \mod (x^4 + 1)$$

Noter que $x^j \mod (x^4 + 1) = x^j \mod 4$.

La multiplication modulo $x^4 + 1$ est notée $\otimes$, elle n'est pas nécessairement inversible. Mais le polynôme $a(x)$ est choisi car il est inversible. En général $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ et $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$.

$d(x) = a(x) \otimes b(x)$

$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$

$d_0 = a_0b_0 \oplus a_3b_1 \oplus a_2b_2 \oplus a_1b_3$

$d_1 = a_1b_0 \oplus a_0b_1 \oplus a_3b_2 \oplus a_2b_3$

$d_2 = a_2b_0 \oplus a_1b_1 \oplus a_0b_2 \oplus a_3b_3$

$d_3 = a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus a_0b_3$

La multiplication par un polynôme fixe $a(x)$ peut s'écrire matriciellement

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Cette opération peut s'écrire matriciellement, si $b = b_3b_2b_1b_0$ est une colonne d'un état alors

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

$$\begin{bmatrix} C_4 \\ C_5 \\ C_6 \\ C_7 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix}$$

etc

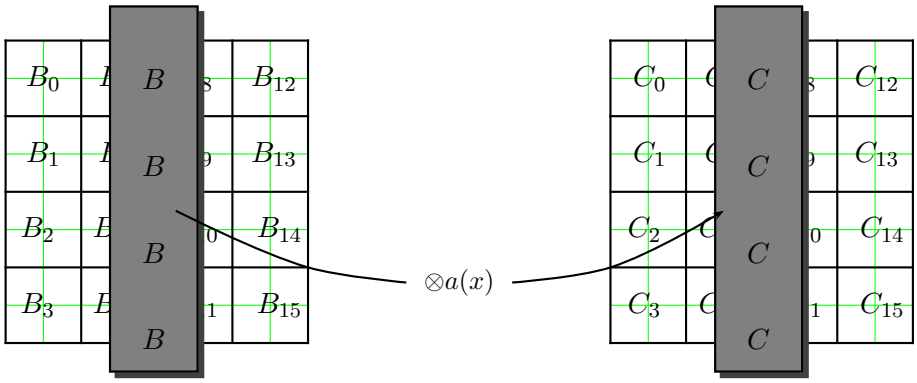


FIGURE 4.5 – Transformation MixColumn

L'inverse de $a(x)$ est le polynôme

$$a^{-1}(x) = 0B_H x^3 + 0D_H x^2 + 09_H x + 0E_H$$

On a

$$a^{-1}(x)a(x) = 1 \bmod x^4 + 1$$

Exemple : ?

4.4 Description de l'AES

4.5 Dérivation de clés de tour

Le nombre de sous clé est égale au nombre de tours plus un. Une sous clé utilisé à la fin.

Toutes les sous clé sont de 128 bits. Quelque soit la longueur de la clé.

Pour une clé de 128 bits il y a 10 tours et 11 clés de tour.

Pour une clé de 192 bits il y a 12 tours et 13 clés de tour.

Pour une clé de 256 bits il y a 14 tours et 15 clés de tour.

À partir d'une clé K (de longueur 128, 192 ou 256) on génère $N_c(N_r + 1)$ mots.

L'opération `KeyExpansion` permet de transformer la clé de chiffrement K (de longueur $4N_k$ octets) en une clé étendue W de $4N_c(N_r + 1)$ octets. Ainsi on obtient $N_r + 1$ clefs de tour et chaque tour utilise une clé de $4N_c$ octets une et une seule fois.

Voir Figure 4.5.

K et W sont des successions de colonnes, chacune de 4 octets. On note par k_i (resp. w_i) la $(i+1)$eme colonne de K (resp. W).

Les sous clés sont calculées récursivement. Elles sont rassemblé dans un tableau w formé de mots (1mot=32 bits).

Production de sous clés d'une clé 128 Soit $K_0, K_1, \ldots, K_{15}$ les octets formant une clé de 128 bits.

Les 11 sous clés sont stockées dans un tableau W de 44 octets. Chaque $W[i]$ est un mot c'est à dire de longueur 32 bits ou 4 octets.

$$K_0 = W[0]W[1]W[2]W[3] = K_0K_1 \cdots K_{15}$$

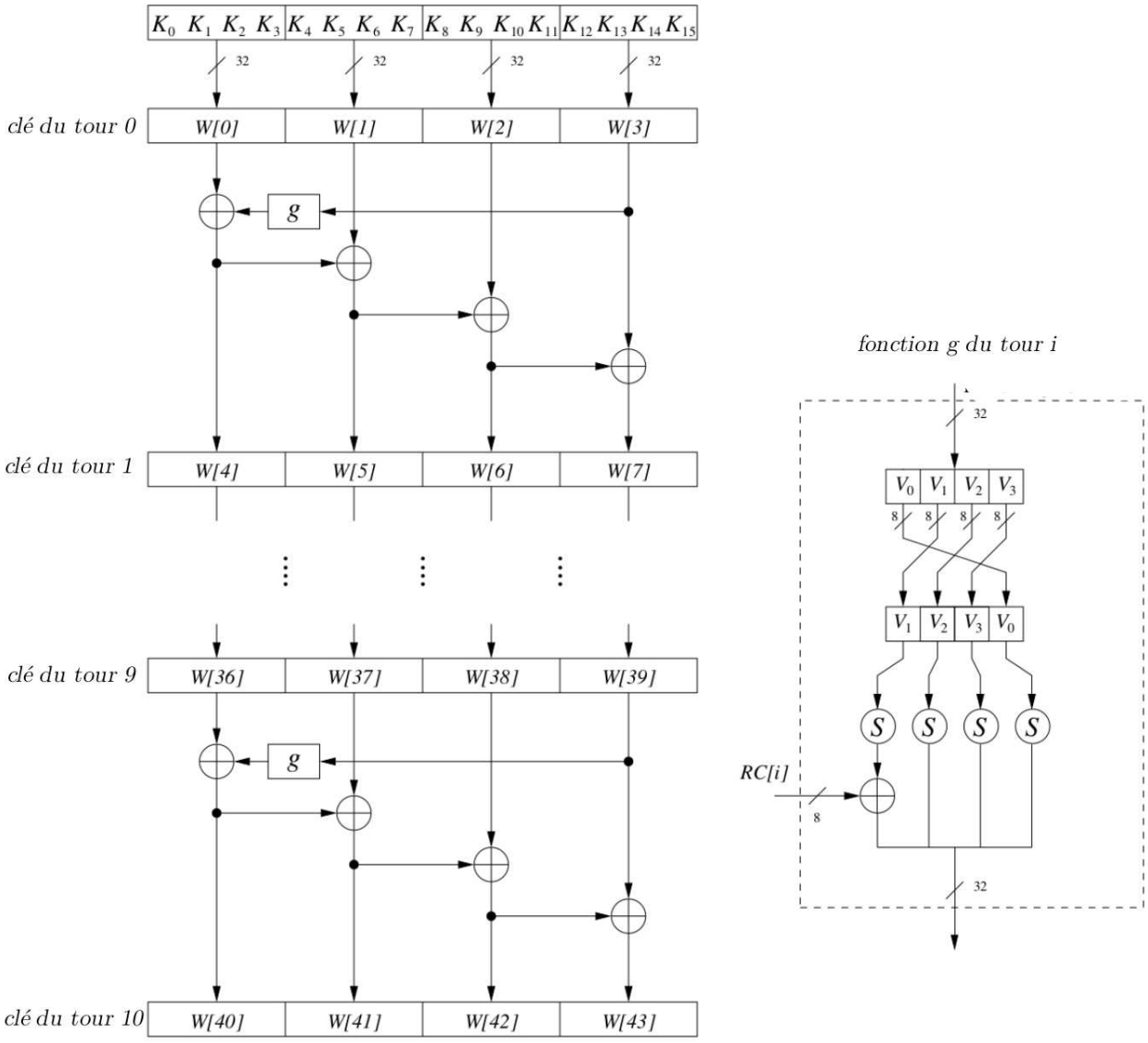


FIGURE 4.6 – Dérivation des clés de tours d'une clé de 128 bits

n'est rien d'autre que la clé initiale de 128 bits Les autres éléments du tableau W sont calculés comme il suit

$$W[4i] = W[4(i-1)] + g(W[4i-1]) \text{ pour } i = 1, \ldots, 10$$

$$W[4i+j] = W[4i+j-1] + W[4(i-1)+j], \text{ pour } i = 1, \ldots, 10 et j = 1, 2, 3.$$

La fonction g est non linéaire : $g : \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2^8 \longrightarrow \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2^8$

$$\begin{aligned} &RC[1] = x^0 = (00000001)_2, \\ &RC[2] = x^1 = (00000010)_2, \\ &RC[3] = x^2 = (00000100)_2, \\ &\ldots \\ &RC[10] = x^9 = (00110110)_2. \end{aligned}$$

La fonction $g()$ ajoute de la non-linéarité à la création des sous clés et augmente la sécurité de l'AES.

La fonction g est une composition d'une permutation des octets suivie de S-Box appliqué à chaque octet en plus de $RC[i]$ xoré avec le premier octet.

$$K_1 = W[4]W[5]W[6]W[7]$$

$$\cdots$$

$$K_2 = W[8]W[9]W[10]W[11]$$

Exemple :

$$K = \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}$$

opérations :

$S_1(w_{i-1})$ est une permutation circulaire de w_{i-1} définie par $S_1([a_0, a_1, a_2, a_3]) = [a_1, a_2, a_3, a_0]$ où $[a_0, a_1, a_2, a_3]$ est un mot de 4 octets.

`SubWord` est une opération qui agit sur des mots de 4 octets et applique S-Box à chaque octet.

La table de constante de rondes $Rcon[i]$ est indépendante de N_k. $Rcon[i]$ est définie récurssivement par $Rcon(i) = [x^{i-1} \bmod g(x), 00, 00, 00]$ où x^{i-1} est la puissance de x dans $\mathbb{F}_{2^8}$.

Les N_k premières colonnes de K sont recopiés dans les N_k premières colonnes de W sans modification. C'est à dire $w_i = k_i$ pour $i = 0, \cdots, N_k - 1$.

Pour $N_k \leq 6$ on a :

$$w_i = \begin{cases} w_{i-Nk} \oplus SubWord(S_1(w_{i-1})) \oplus rcon(\frac{i}{Nk}) & if\ i \bmod N_k = 0 \\ w_{i-Nk} \oplus w_{i-1} & if\ i \bmod N_k \neq 0 \end{cases}$$

Pour $N_k > 6$ on a :

$$w_i = \begin{cases} w_{i-N_k} \oplus SubWord(S_1(w_{i-1})) \oplus rcon(\frac{i}{N_k}) & if\ i \bmod N_k = 0 \\ w_{i-N_k} \oplus SubWord(w_{i-1}) & if\ i \bmod N_k = 4 \\ w_{i-N_k} \oplus w_{i-1} & \text{sinon} \end{cases}$$

Clé initiale $K = K_0 \quad K_1 \qquad\qquad\qquad K_{10}$

$w_0 \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6 \quad w_7 \quad \cdots \quad w_{40} \quad w_{41} \quad w_{42} \quad w_{43}$

À partir de la matrice W on peut extraire facilement la `RoundKeys`. Les N_b premières colonnes de W forment la clé pour la première ronde, Les N_b colonnes suivantes de W forment la clé pour la deuxime ronde, etc

N_k mots : longueur de la clé N_r mots : nombre des rondes $N_b(N_r + 1)$ mots : taille de la clé étendue

```
Constants: int Nb = 4;
Inputs :  int Nk = 4, 6, or 8; // the number of words in the key
           array key  of 4*Nk bytes or Nk words // input key
```

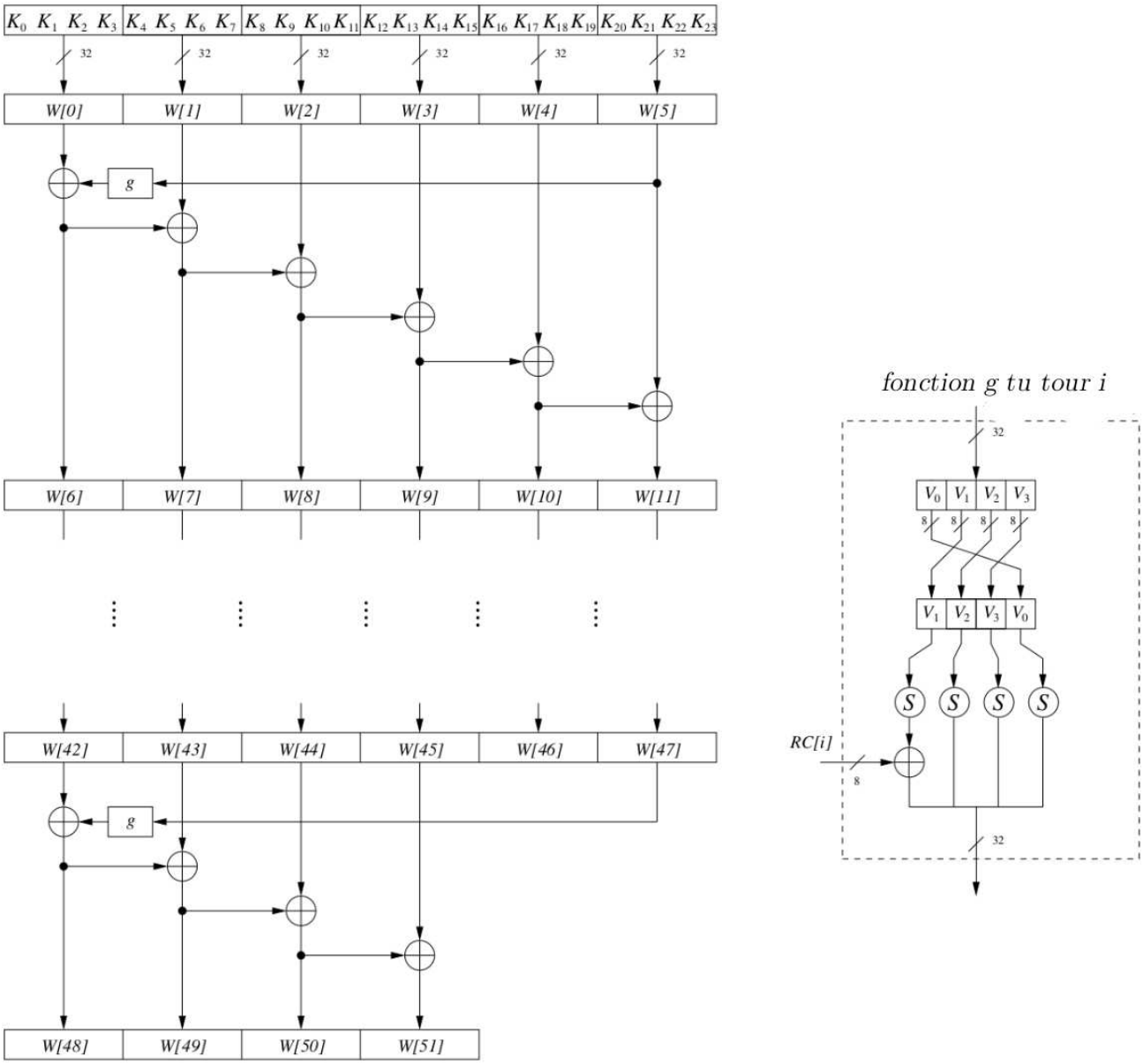


FIGURE 4.7 – Dérivation des clés de tours d'une clé de 192 bits

N_k	N_r	$N_b(N_r+1)$
4	10	44
6	12	52
8	14	60

TABLE 4.5 – Nombres de sous-clés

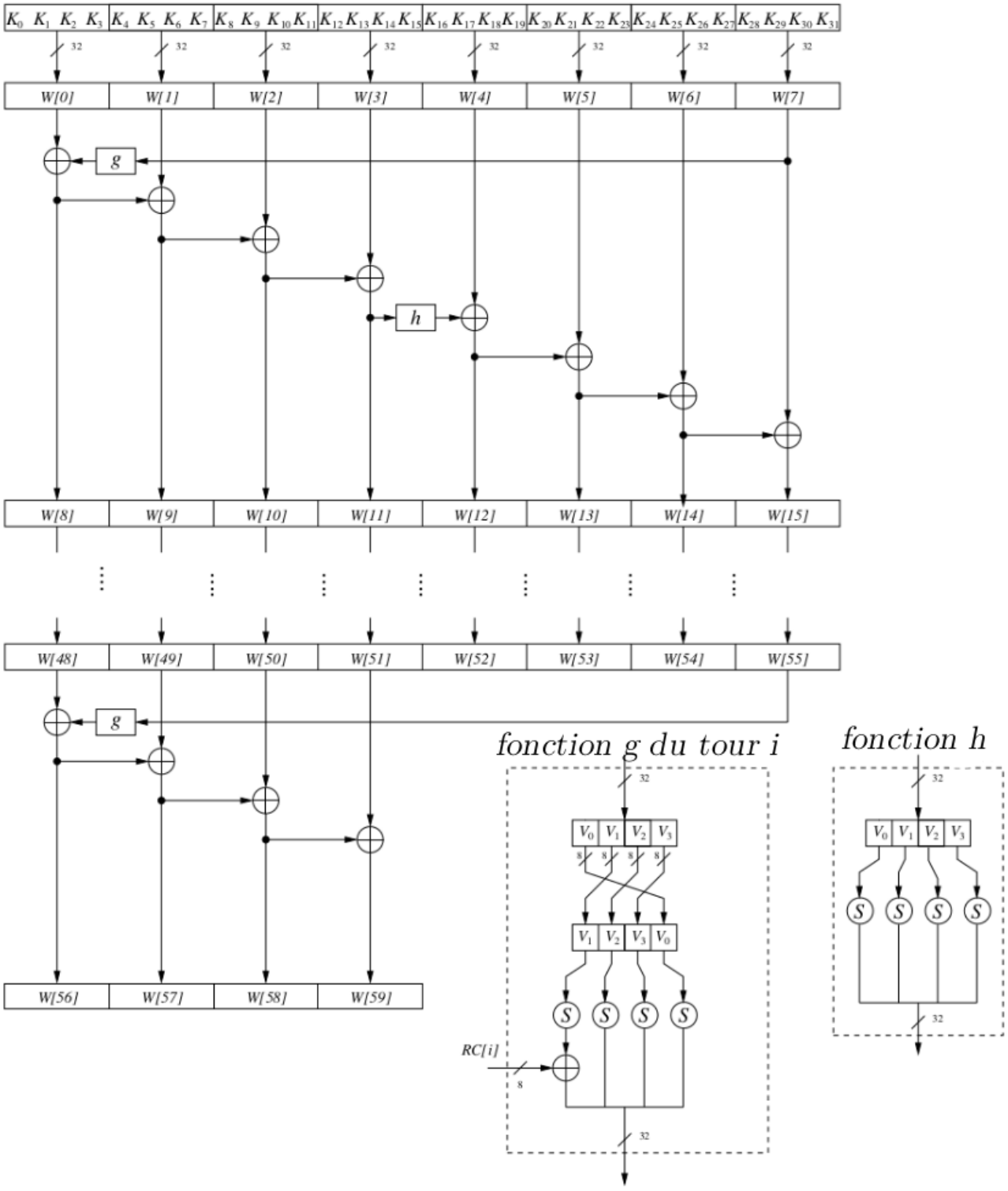
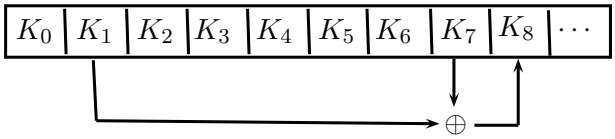
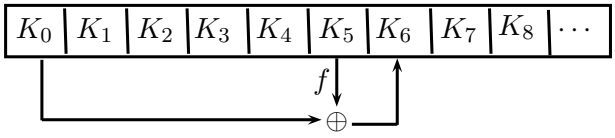


FIGURE 4.8 – Dérivation des clés de tours d'une clé de 256 bits



$K_i = K_{i-N_k} \oplus K_i$ pour $i \ mod \ N_k = 0$

FIGURE 4.9 – Génération des sous clés pour $i \ mod \ N_k \neq 0$



$K_i = K_{i-N_k} \oplus f(K_i)$ pour $i \ mod \ N_k = 0$

où $f(K_i) = SubWord \circ \sigma(K_i) \oplus Rcon(i/N_k)$

FIGURE 4.10 – Génération des sous clés pour $i \ mod \ N_k = 0$

```
Output : array w of Nb*(Nr+1) words or 4*Nb*(Nr+1) bytes // expanded key
Algorithm:
void KeyExpansion(byte[] key, word[] w, int Nw) {
      int Nr = Nk + 6;
      w = new byte[4*Nb*(Nr+1)];
      int temp;
      int i = 0;
      while ( i < Nk) {
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]);
        i++;
      }
      i = Nk;
      while(i < Nb*(Nr+1)) {
        temp = w[i-1];
        if (i % Nk == 0)
          temp = SubWord(RotWord(temp)) ^ Rcon[i/Nk];
        else if (Nk > 6 && (i%Nk) == 4)
          temp = SubWord(temp);
        w[i] = w[i-Nk] ^ temp;
         i++;
         }
}
```

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x^i	01	02	04	08	10	20	40	80	1b	36	6c	d8	ab	4d	9a

TABLE 4.6 – puissances de x = 0x02

4.6 Evaluation de l' AES

S-Box n'a ni point fixe ni point opposé ni point fixe inverse.

MixColomn combiné avec ShiftRows permet après plusieurs ronde que chaque bit de sortie dépende de tous les bits en entrée.

La fonction $x \mapsto x^{-1}$ est la meilleure connue contre les attaques linéaire et différentielle.

On a 3.4×10^{38} clés de 128-bit, 6.2×10^{57} clés de 192-bit, et 1.1×10^{77} clés de 256-bit possibles

Pour DES $7.2x10^{16}$ clés différentes possibles.

Si une machine pourrait casser une clé DES en une seconde (c-à-d calcule 2^{55} clés par seconde), alors il faudrait 149 mille milliards d'années pour cracker une clé AES. L'age de l'univers est de 20 milliards d'années au maximum.

4.7 Exercices

Exercice 1. Le corps $\mathbb{F}_{2^8}$ étant défini par le polynôme irréductible $P(x) = x^8 + x^4 + x^3 + x + 1$. Calculer :

1. 09+A0, 45+25, E1+C1
2. 03x76, 02x25, 33x12
3. $\frac{x^4+x+1}{x^7+x^6+x^3+x^2}$.
4.

Exercice 2. Soit $b = b_7x^7 + \ldots + b_1x + b_0$. Calculer dans $\mathbb{F}_{2^8}$ les produits 01xb, 02xb, 03xb.

Exercice 3. Trouver dans $\mathbb{F}_{2^8}$ les inverses de 75, 1A, C1, 10 et vérifier.

Exercice 4. Soit S l'opération S-Box de l'AES. Utiliser sa table pour calculer $S(10)$, $S(45)$, $S(1C)$.

Exercice 5. Soit S^{-1} l'opération inverse de S-Box de l'AES. Utiliser sa table pour calculer $S^{-1}(1E)$, $S^{-1}(C5)$, $S^{-1}(2C)$.

Exercice 6. Écrire en binaire les éléments de $\mathbb{F}_{2^8}$ suivants : $RC[8] = x^7$, $RC[9] = x^8$, $RC[10] = x^9$.

Exercice 7. Calculer S(88), S(54) où S est la S-Box de l'AES.

Exercice 8. Calculer le résultat du premier tour de l'AES au message M=MASTERCRYPTOSINF et la clé k=MASTERCRYPTOSINF.

Exercice 9. Montrer que l'inverse du polynôme $a(x) = 03_H X^3 + 01_H X^2 + 01_H X + 02_H \ mod\ (X^4 + 1)$ considéré comme un polynôme à coefficients dans $\mathbb{F}_{2^8}$ est $b(x) = 0B_H X^3 + 0D_H X^2 + 09_H X + 0E_H$

Exercice 10. Dans le corps $\mathbb{F}_{2^8}$ on pose $RC[1] = 1$ et pour $j \geq 2$, $RC[j] = 2RC[j-1]$. Calculer $RC[j]$ pour $j = 1, 2, \cdots, 12$.

Exercice 11. Soit la clé K=11 00 00 00 11 00 00 00 11 00 00 00 11 00 00 00 qu'on veut utiliser pour chiffrer des blocs de taille 128 bits avec l'AES. Calculer k_i pour $i = 0, 1, \cdots, 7$

Exercice 12. On considère l'AES-128 et la clé

`2c 7f 16 17 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c`

Calculer w_i, $i = 0, \cdots, 7$.

Utilisez cette clé pour donner le résultat de la première ronde , du chiffrement de

`32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34`

On donne le résultat sous forme matricielle.

Exercice 13. On rappelle le polynôme irréductible $P(x) = x^8 + x^4 + x^3 + x + 1$ sur $\mathbb{F}_2$ utilisé pour AES. Quel est l'inverse de x^7 mod $P(x)$? Calculer x^{42} mod $P(x)$.

Exercice 14. En utilisant la tables des inverse dans le corps $\mathbb{F}_{2^8}$ quel est l'inverse de 0x83 ? Vérifier.

Bibliographie

[1] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001. http ://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] Joan Daemen and Vincent Rijmen, The Design of Rijndael, AES - The Advanced Encryption Standard, Springer-Verlag 2002 (238 pp.)

Chapitre 5

Cryptographie à clef publique : RSA

Sommaire

Proof by poem :
The RSA Encryption Algorithm

Take two large prime numbers, q and p.
Find the product n, and the totient ϕ.
If e and ϕ have GCD one
and is e's inverse, then you'are done!
For sending m raised to the e
reduced mod *n gives secre-c.*

Daniel G. TREAT, *National Security Agency*
Mathematics Magazine, Vol 75 N 4, October 2002.

RSA est un crypto-système à clef publique, inventé en 1978 par R. Rivest, A. Shamir et L. Adleman (dont les initiales forment RSA) du MIT. Mais leur objectif initial était d'établir que l'idée d'un crypto-système à clef publique que W. Diffie et M. Hellman venaient d'inventer en 1976 était une impossibilité logique.

RSA est basé sur la diffuclté de factoriser un nombre qui est le produit de deux grands nombres premiers. RSA est aussi utilisé pour les signatures numériques.

5.1 Description de RSA

5.1.1 Outils mathématiques

5.1.2 Indicatrice d'Euler

L'indicatrice d'Euler est une fonction $\phi : \mathbb{N}^* \longleftrightarrow \mathbb{N}^*$, définie par $\phi(n) = card\{1 \leq m \leq n \ / \ pgcd(m,n) = 1\}$.

Exemples : $\phi(4) = card(\{1,3\}) = 2$, $\phi(8) = card(\{1,3,5,7\}) = 4$, $\phi(1) = card(\{1\}) = 1$, $\phi(9) = card(\{1,2,4,5,7,8\}) = 6$,

Si p est un nombre premier alors $\phi(p) = card(\{1,2,\ldots,p-1\}) = p-1$,

Soit $n \in \mathbb{N}^*$, alors $\phi(n)$ est égal : – au nombre d'éléments inversibles de l'anneau $\mathbb{Z}/n\mathbb{Z}$;
– au nombre de générateurs d'un groupe cyclique d'ordre n ;

La fonction $\phi(n)$ est multiplicative, c'est-à-dire que si $m, n \in \mathbb{N}^*$ premiers entre eux, alors $\phi(m.n) = \phi(m)\phi(n)$.

Le calcul de l'indicateur d'Euler est donc important. Voici quelques propriétés permettant de le calculer :

Si p est premier et $\alpha \geq 1$ alors

$$\varphi(p) = p - 1 \tag{5.1}$$
$$\varphi(p^\alpha) = p^\alpha - p^{\alpha-1} \tag{5.2}$$
$$\varphi(nm) = \varphi(n)\varphi(m) \text{ si } n, m \in \mathbb{N} \text{ et } pgcd(n, m) = 1 \tag{5.3}$$

En particulier, les propriétés précédentes permettent de calculer l'indicateur d'Euler $\varphi(n)$ connaissant la décomposition en facteurs premiers de n. Ainsi on a

$$\varphi\left(p_1^{\alpha_1}\right) \ldots \varphi\left(p_r^{\alpha_r}\right) = \left(p_1^{\alpha_1} - p_1^{\alpha_1 - 1}\right) \times \cdots \times \left(p_r^{\alpha_r} - p_r^{\alpha_r - 1}\right) = n\left(1 - \frac{1}{p_1}\right) \ldots \left(1 - \frac{1}{p_r}\right).$$

On a aussi les propriétés suivantes :

$$n = \sum_{d|n} \varphi(d)$$

$$\varphi(n) = \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

avec p premier.

5.1.3 Description de RSA

Alice, Destinataire, :

- choisit deux grands nombres premiers p et q.[1] et calcule $n = pq$.

- choisit un nombre d (grand, à l'aide de l'algorithme d'Euclide) et premier avec $(p-1)(q-1) = \varphi(n)$ (fonction d'Euler).

- calcule e inverse de de $d \mod \varphi(n)$ (à l'aide de l'algorithme d'Euclide).

- garde soigneusement secrets p, q et d

- envoie publiquement à l'émetteur (e, n), c'est la clef publique.

Bob, émetteur :

- Transforme le message M à émettre en un nombre[2] de $\mathbb{Z}_n$ (Il découpe éventuellement le texte en bloc)

- crypte par la relation $C = M^e \mod n$.

- envoie le message au destinataire.

Destinataire : déchiffre le message C par

$$C^d \bmod n = M$$

Exemple 5.1.1 *Prenons 2 nombres premiers au hasard : $p = 29$, $q = 37$. On calcul $n = pq = 29*37 = 1073$.*

*On choisit e au hasard tel $pgcd(e, (p-1)(q-1)) = 1$. (p-1)(q-1) = (29-1)(37-1) = 1008. On prend e = 71. On choisit d tel que 71*d mod 1008 = 1 On trouve d = 1079*

1. plus de 100 chiffres chacun à générer par les algorithmes probabilistes
2. par exemple espace = 00, A=01, B=02 etc

On a maintenant les clés :
- La clé publique est (e,n) = (71,1073) (=clé de chiffrement)
- La clé privée est (d,n) = (1079,1073) (=clé de déchiffrement)

On va chiffrer le message 'HELLO'. On va prendre le code ASCII de chaque caractère et on les met bout à bout : m = 7269767679.

Ensuite, on découpe le message en blocs qui comportent moins de chiffres que n. On va donc découper notre message en blocs de 3 chiffres : 726 976 767 900 (quitte à complèter par des zéros)

Ensuite on encrypte chacun de ces blocs :

$726^71 \ mod \ \ 1073 = 436$
$976^71 \ mod \ \ 1073 = 822$
$767^71 \ mod \ \ 1073 = 825$
$900^71 \ mod \ \ 1073 = 552$

Le message encrypté est 436 822 825 552. On peut le décrypter avec d :

$436^1079 \ mod \ \ 1073 = 726$
$822^1079 \ mod \ \ 1073 = 976$
$825^1079 \ mod \ \ 1073 = 767$
$552^1079 \ mod \ \ 1073 = 900$

C'est à dire la suite de chiffre 726976767900. On retrouve notre message en clair 72 69 76 76 79 : 'HELLO'.

5.2 Démonstrations mathématiques

Dans cette section, on montre **??**, comment inverser $\mod \varphi(n)$, calculer de grandes puissances modulaire, générer de grands nombres.

5.2.1 Démonstration

On démontre que $C^d = (M^e)^d = M \mod n$. On distingue deux cas :

Si M est premier avec n :

On a $(M^e \mod n)^d \mod n = M^{de} \mod n$ or $de \equiv 1 \mod \varphi(n)$ càd $de = 1 + k\varphi(n)$ où k est un entier.

Or on sait que si n est un entier et a un entier premier avec n. alors $a^{\varphi(n)} \equiv 1 \mod n$.

Puisque M est premier avec n on a $M^{\varphi(n)} \equiv 1 \mod n$ d'où

$$C^d = M^{de} = M^{k\varphi(n)+1} = 1.M \mod n = M \mod n.$$

Si M est non premier avec $n = pq$:

Dans ce cas M est multiple de p ou de q. Supposons alors que $M = p^\alpha m$ où $m \in \mathbb{N}$ et α est le plus grand entier vérifiant cette relation.

m n'est multiple ni de p ni de q d'où m est premier avec n. D'où

$$M^{de} = (p^\alpha m)^{de} = p^{\alpha de}.m^{de} \bmod n = p^{\alpha de} m \bmod n$$

Or $p^{\alpha de} \equiv p^\alpha \bmod p$ et $p^{\alpha de} \equiv p^\alpha \bmod q$ car $p^{\varphi(q)} = p^{q-1} \equiv 1 \bmod q$ or $de = 1 + k\varphi(n)$ on obtient $p^{de} = p^{1+k(p-1)(q-1)} = p.1^{k(p-1)} \equiv p \bmod q$ d'où en élevant à la puissance p : $p^{\alpha de} = p^\alpha \bmod q$,

La différence $p^{\alpha de} - p^\alpha$ est multiple de q et de p donc multiple de pq d'où d'après le lemme de Gauss $p^{\alpha de} \equiv p^\alpha \bmod n$ et donc

$$C^d = M^{de} \equiv p^{\alpha de} m \bmod n = p^\alpha m \bmod n = M \bmod n$$

5.2.2 Inversion modulo $(p-1)(q-1)$

On sait que a est inversible dans l'anneau $\mathbb{Z}/n\mathbb{Z}$ si et seulement si a est premier avec n.

Si a et n sont premiers entre eux alors il existe deux entiers u et v tels que $ua + vn = 1$. D'où en modulo n on conclut que : $ua \equiv 1 \bmod n$.

On utilise l'algorithme d'Euclide étendu pour calculer u.

Inversion modulo $(p-1)(q-1) = \varphi(n)$

Il est facile de trouver un nombre d premier avec $\varphi(n)$ en le choisissant au hasard et vérifiant à l'aide de l'algorithme d'Euclide.

Puis on doit chercher l'inverse e de $d \mod \varphi(n)$ c'est à dire $de = 1 \bmod \varphi(n)$ ou $de \equiv 1 + k\varphi(n)$ où $k \in \mathbb{N}$. C'est rien d'autre que la relation de Bezout. L'algorithme d'Euclide

$$\begin{aligned} \varphi(n) &= q_1 d + r_1 \\ d &= q_2 r_1 + r_2 \\ &\vdots \\ r_{n-2} &= q_n r_{n-1} + r_n \end{aligned}$$

donne $r_n = 1$ puisque d et $\varphi(n)$ sont premiers entre eux. En partant de la dernière équation

$$1 = r_{n-2} - q_n r_{n-1}$$

on remplace r_{n-1} par $r_{n-3} - q_{n-1} r_{n-2}$ et on remonte

Deux choses à démontrer :

- il existe au moins un couple deux entiers a et b inférieurs à n tels que $ab \equiv 1 \bmod \varphi(n)$;
- pour tout $m < n$, $(m^b)^a = m \bmod n$.

Lemme 5.2.1 *Soit $a \in \mathbb{Z}_n$. a est inversible dans $\mathbb{Z}_n$ si et seulement si $pgcd(a, n) = 1$*

Preuve : Inversement supposons $pgcd(a, n) > 1$. Puisque $ab \equiv 1 \bmod n$, il existe $k \in \mathbb{Z}$ tel que $ab + nk = 1 \bmod n$. D'après le théorème de Bezout a et n sont premiers entre eux. Ce qui est absurde.

Supposons $pgcd(a, n) = 1$ alors d'après le théorème de Bezout il existe u et v dans $\mathbb{Z}$ tels que $au + nv = 1$ d'où $au = 1 \bmod n$. Il suffit de prendre $b = u \bmod n$

L'unicité est évidente.

En appliquant le lemme on a : Pour a tel que $0 < a < (p-1)(q-1)$ et premier avec $(p-1)(q-1)$, il existe un b (unique) tel que $0 < b < n$ et $ab = 1 \mod (p-1)(q-1)$. L'existence d'un tel a est triviale par exemple : 1 et $(p-1)(q-1)-1$.

En prenant a et b comme ci-dessus on obtient

Proposition 5.2.2 *Pour tout $m \in \mathbb{Z}_n$: $(m^b)^a \equiv m \ mod\ n$.*

Preuve :

On montre d'abord que $(m^b)^a = m \mod p$ et $(m^b)^a = m \mod q$.

Si p divise m : $(m^b)^a = m = 0 \mod p$

Si q divise m : $(m^b)^a = m = 0 \mod q$

Si m n'est ni mulitple de p ni mulitple de q on a $pgcd(m,p) = pgcd(m,q) = 1$ et :

$(m^b)^a = m^{ba} = m^{1+k(p-1)(q-1)} = m(m^{(p-1)(q-1)})^k$

Le petit théorème de Fermat donne alors : $m^{(p-1)} \equiv 1 \mod p$ d'où $m^{(p-1)(q-1)k} = 1 \mod p$ donc $(m^b)^a = m \mod p$.

et on montre de même : $m^{(q-1)} = 1 \mod q$ donc $(m^b)^a = m \mod q$

Enfin, $(m^b)^a = m \mod pet (m^b)^a = m \mod q$ implique $(m^b)^a = m(modn)$.

$(m^b)^a = m \mod p$ d'où p divise $(m^b)^a - m$ d'où $kp = (mb)a - m$ $(m^b)^a = m \mod q$ d'où q divise $(m^b)^a - m$ d'où $k'q = (mb)a - m$ D'où $kp = k'q$ ce qui entraîne p divise k' et q divise k car $pgcd(p,q) = 1$ (lemme de Gauss). D'où $k = k"q$, puis finalement $kp = k"qp = (m^b)^a - m$ et donc $pq = n$ divise $(m^b)^a - m$

5.2.3 Algorithme d'Euclide étendu

L'algorithme d'Euclide donne le pgcd de deux nombres. En le prenant à l'envers, on peut l'utiliser pour trouver les coefficients de Bézout.

Exemple 5.2.3 $pgcd(38,17) = 1$
$38 = 2 \times 17 + 4$
$17 = 4 \times 4 + 1$
$4 = 2 \times 2 + 0$. *D'où*
$1 = 17 - 4 \times 4 = 17 - 4 \times (38 - 2 \times 17) = 9 \times 17 - 4 \times 38.$

5.2.4 Calcul des puissances modulo n

Comment calculer $123456^{10000} \mod 456456789789$? par exemple.

Calcul de $z = M^e \mod n$. On note $e(i)$ le ième bit dans la décomposition binaire de

$$e = \sum_{i=0}^{t-1} e(i)2^i$$

On décompose e en base 2 : $e = \sum_{i=0}^{t-1} e(i)2^i$ Il y a [$log_2 n$] opérations.

Ce qui revient à calculer pour $i = 1 \cdots t$

Algorithm 1 Algorithme

z :=1 ;
Pour i=t-1 à 0 faire
z :=z2 mod n ;
si e(i)=1 alors z :=z.M mod n finsi
finpour

$a^{2^i} \mod n = x_{i-1}^2 = x_i$

$$a^e = a^{\sum_{i=0}^{t-1} e(i)2^i} = \prod a^{e(i)2^i} = \prod x_i^{c_i} \mod n$$

Il suffit de multiplier entre eux les x_i dont les $e(i)$ correspondants sont non-nuls : maximum [$log_2 n$] opérations.

Deuxième méthode :

On utilise le fait que $a \equiv b \mod n$ implique $a^2 \equiv ba \mod n$:

i ¡- 1
j ¡- 0
tant que (k¡e) faire :
j ¡- j*a // on multiplie j par a
j ¡- j mod n // on affecte à j la valeur du reste de la D.E. de j par n
k ¡- k+1 // on incrémente la variable compteur k
retourner j

Cet algorithme fait n itérations de la boucle dont le temps d'exécution de chacune est similaire. Cet algorithme est donc en temps polynomial, mieux : linéaire.

5.2.5 Tests probabilistes de primalité

Les algorithmes probabilistiques de test de primalité ont vu le jour dans les années 1970. Celui de Miller-Rabin en 1977. Il est un raffinement du test de Solovay-Strassen. Miller-Rabin est utilisé dans presque toutes les implémentations de RSA.

Comment générer de grands nombres premiers ?

Ce test repose sur l'exponentiation modulaire. Il consiste en :

- choisir un nombre au hasard entre 2 et $n-1$
- si $a^{n-1} \equiv 1 \mod n$, déclarer que n est premier.

c'est une sorte de réciproque (probabiliste) du petit théorème de Fermat.

En 1989, Su Hee Kim et Carl Pomerance, ont montré que le risque d'erreur $E(n)$, par exemple que pour $n = 10^{100}$, si le test probabiliste de Fermat le déclare premier, alors la probabilité que n soit effectivement premier est supérieure à 99,9999972%.

Rappelons que si $\pi(n)$ est le nombre de nombres premier et $\leq n$ alors nous avons

$$\pi(n) \sim \frac{n}{\ln n} \tag{5.4}$$

Définition 5.2.4 *Soit p un nombre premier > 2 et x un entier tel que $1 \leq x \leq p$. On dit que x est un résidu quadratique mod p si il existe $y \in \mathbb{Z}_p$ tel que $x \equiv y^2$ mod p.*

Théorème 5.2.5 (Critère d'Euler) *Soit p un nombre premier > 2. x est un résidu quadratique mod p si et seulement si $x^{(p-1)/2} \equiv 1$ mod p.*

On suppose que qu'il existe y tel que $x \equiv y^2 \bmod p$ d'où $x^{(p-1)/2} = (y^2)^{(p-1)/2} = y^{p-1} \equiv 1 \bmod p$.

Preuve :

Inversement. Soit g un générateur de de $\mathbb{Z}_p^*$ alors il existe un entier i tel que $x^{(p-1)/2} = (g^i)^{(p-1)/2} = y^{i(p-1)/2} \bmod p$. D'où l'ordre p de g divise $i(p-1)/2$ d'où i est pair et $x = g^{i/2}$.

Définition 5.2.6 (Symbole de legendre) *Soit p un nombre premier > 2. Pour tout entier a*

$$L(a,p) = \begin{cases} 0 & si \ \ a \equiv 0 \ mod \ p \\ 1 & si \ a \ est \ un \ résidu \ quadratique \ mod \ p \\ -1 & si \ a \ est \ un \ non\text{-}résidu \ quadratique \ mod \ p \end{cases}$$

Théorème 5.2.7 *Soit p un nombre premier > 2 . On a $L(a,p) = a^{(p-1)/2}$ mod p.*

Définition 5.2.8 *Soit n un entier impair dont la décomposition en nombre premiers est $n = \prod_{i=1}^{k} p_i^{e_i}$ et a un entier. Le symbole de Jacobi est défini par $n = \prod_{i=1}^{k} L(a,p_i)^{e_i}$.*

Algorithme de Solovay-Strassen

Mis au point en 1976.

1) on tire aléatoirement un entier a tel que $1 \leq a \leq n-1$.

2) si $L(a,n) \equiv a^{(n-1)/2} \bmod n$ alors n est premier,

sinon n est décomposable.

L'algorithme de Solovay-Strassen est un algorithme polynômial en $\mathcal{O}((lnn)^3)$

Test de Miller-Rabin

L'idée est surtout de combiner plusieurs tests afin d'avoir une probabilité très forte (99,999 ?%) d'avoir un nombre premier.

Proposition 5.2.9 *L'algorithme de Miller-Rabin est polynomial, de complexité $\mathcal{O}((lnn)^3)$.*

Lemme 5.2.10 *Soit $n \geq 3$ impair $a \in \mathbb{Z}_n^*+$. On pose $n-1 = 2^k m$ où m est impair. Si l'une des conditions suivante est vraie alors n est décomposable.*

i) $a^{n-1} \not\equiv 1$ mod n ;

ii) $a^{n-1} \equiv 1$ mod n, $a^m \not\equiv 1$ mod n et aucun des éléments de la suite a^m, a^{2m}, a^{4m}, $\cdots$, $a^{2^k m}$ n'est congru à -1 mod n.

Preuve :

i) c'est la contraposée du petit théorème de Fermat.

ii) Soit b le dernier entier dans la suite a^m, a^{2m}, a^{4m}, $\cdots$, qui n'est pas congru à 1 mod n, alors $b^2 \equiv 1 \bmod n$ or $b \not\equiv \pm 1 \bmod n$ donc $b-1$ et $b+1$ sont des facteurs non triviaux de n.

Si $a \in \mathbb{Z}_n^*+$ satisfait la condition :

i) du lemme **??** a est appelé témoin de Fermat pour la non décomposabilité de n. On note

$$F_n = \{a \in \mathbb{Z}_n^* + / a \text{ est un témoin de Fermat}\}$$

Tout $a \in \mathbb{Z}_n^*+$ qui n'est pas premier avec n est un témoin de Fermat pour n

ii) du lemme **??** a est appelé témoin de Miller.

Un entier n composé dont les seuls témoins de Fermat sont les nombres premiers avec n est appelé nombre de Carmichael.

Le pus petit nombre de Carmichael est $561 = 3 \times 11 \times 17$.

Proposition 5.2.11 *Si n est un nombre composé mais n'est pas de Carmichael alors $|F_n| > n/2$.*

On considère $B = Z_n^*\mathbb{F}_n$, on a $B = \{a \in Z_n^* | a^{n-1} = 1 \bmod n\}$. On vérifie facilement que B est un sous groupe de Z_n^* et qu'il est propre puisque n est un nombre composé mais n'est pas de Carmichael. D'après le théorème de Lagrange $|B|$ divise Z_n^*. D'où $|B| \leq (n-1)/2$. Donc $|F_n| = |Z_n^*| - |B| > \frac{n}{2}$.

L'algorithme dit de Fermat suivant permet de tester si un nombre est probablement premier en temps polynomial

Algorithme The Fermat "Almost Prime" Test.

entrée : un entier $n > 2$.

Algorithme :

choisir $a \in Z_n^*$

if $a^{n-1} \equiv 1 \bmod n$

alors sortir ?premier ?

sinon sortir ?composé ?.

Si l'entrée est un nombre premier alors cet algorithme affirme certaimement qu'il est premier. Mais si n est un nombre composé mais n'est pas de Carmichael alors la proposition donne composé avec la probabilité au moins $\frac{1}{2}$.

Cet algorithme est utilisé dans certains cryptosystèmes puisqu'il y a plus de nombre premier que de nombre de Carmichael.

Théorème 5.2.12 (Alford, Granville and Pomerance (1994)) *Le nombre des nombres de Carmichael plus petits ou égaux à n est supérieur ou égale à $n^{2/7}$. En particulier il y a un nombre infini de nombres de Carmichael.*

L'algorithme suivant utilise les témoins de Fermat et de Miller.

Algorithme de Test de primalité de Miller-Rabin . Algorithme : test de primalité de Miller-Rabin sur n
Choisir $a \in \{1, \cdots, n-1\}$
Si $PGCD(a, n) \neq 1$, retourner "composé"
Écrire $n-1$ sous la forme $m.2^k$ avec m impair
Si $a^m \equiv 1 \bmod n$, retourner "premier"
De $i = 0$ à $k-1$, faire :
si $a^{m.2^i} \equiv -1\ mod\ n$, retourner "premier"
i=i+1
Retourner "composé"

Théorème 5.2.13 *L'algorithme de test de primalité de Miller-Rabin est probabiliste et polynomial. Soit l'entré n.*
i) si n est premier alors l'algorithme donne tjrs premier.
ii) si n est composé alors la probabilité pour que l'algorithme donne composé est $\geq frac12$

Preuve : i) Supposons que l'entrée n est premier. pour tout $a \in \mathbb{Z}_n^*$ on a $pgcd(a, n) = 1$. L'algorithme ne peut donner composé en ligne 2. Le seul cas où il peut sortir composé est si $a \not\equiv 1 \bmod n$ et $a^{m.2^i} \not\equiv -1 \bmod n$ pour tout $0 \leq i \leq k-1$. Dans ce cas on a soit :
$a^{n-1} \not\equiv 1 \bmod n$ d'où a est un témoin de Fermat pour n.
ou $a^{n-1} \equiv 1 \bmod n$ d'où a est un témoin de Miller pour n.

Ce qui est impossible puisque n est premier d'après le Lemme 5.2.10.

ii) On considère deux cas :

1) n est composé et n'est pas un nombre de Carmichael

Supposons que l'algorithme donne "premier". On a alors soit : $a^m \equiv 1 \bmod n$ ou $a^{m.2^i} \equiv -1 \bmod n$ pour un certain $0 \leq i \leq k-1$. Dans les 2 cas $a^{n-1} \equiv 1 \bmod n$, d'où a n'est pas un témoin de Fermat pour n. Mais d'après la Proposition 5.2.11 on a $|F_n| \geq \frac{n}{2}$. Donc Pr(l'algorithme donne composé $\geq \frac{1}{2}$).

2) n est est un nombre de Carmichael

On considère deux sous-cas :

a) n n'est pas une puissance d'un nombre premier

On pose

$$t = max\{0 \leq i \leq k-1 | \exists a \in \mathbb{Z}_n^* \text{ tel que } a^{m.2^i} = -1 \bmod n\}$$

et

$$B_t = \{a \in \mathbb{Z}_n^* | a^{m.2^i} = \pm 1 \bmod n\}$$

si $a \notin B_t$ l'algorithme donne composé.

5.3 Remarques sur RSA

Souvent on utilise RSA en combinaison avec avec un crypto-système à clef privée. (PGP)

5.4 Attaques

Il y a deux types d'attaques pour un système à clé publique :

i) attaque sur la clé : étant donnée la clé publique, retrouver la clé secrète. Pour RSA, il s'agit, étant donné (N, e), de retrouver (p, q, d). On peut montrer que c'est équivalent à la factorisation de N.

ii) attaque sur le message : étant donnée un message chiffré, retrouver le message clair M correspondant. Pour RSA, comme $c = M^e \bmod n$, il s'agit d'extraire des racines e-èmes modulo n.

On montre que le calcul d'une des clefs à partir de l'autre est équivalent au problème de la factorisation.

Il n'est pas encore établi que la cryptanalyse du RSA est équivalente au problème de la factorisation.

5.4.1 Sécurité

Le record actuel de factorisation est de 200 chiffres décimaux (RSA-200), soit 663 bits. Cette factorisation a été annoncée par Bahr, Boehm, Franke et Kleinjung le 9 mai 2005. Il est recommandé d'utiliser une clé d'au moins 1024 bits. Aussi, p et q doivent être des nombres premiers forts, i.e. tels que $p-1$, $p+1$, $q-1$, $q+1$ ont un grand facteur premier. De même, si $r = (p-1)/2$ et $s = (q-1)/2$, $r-1$ et $s-1$ doivent avoir un grand facteur premier. L'exposant privé d ne doit pas être choisi trop petit ; par contre, on peut prendre e petit pour accélérer le chiffrement (e = 65537 est classique).

les clés 1 024 bits ne seront bientôt plus un standard. Dans une publication spéciale de mai 2006, le (NIST) National Institute of Standards and Technology avait recommandé que cette clé ne soit plus utilisée après 2010 [**?**]. Le même mois, les Laboratoires RSA avaient publié des recommandations invitant à passer aux clés de 2 048 bits. De quoi être tranquille jusqu'en... 2030, selon RSA [**?**]. D'autres prévisions prévoient, puisque la puissance des ordinateurs double tous les 18 mois (loi de Moore), une clé de 2048 bits devrait tenir jusqu'à ... 2079.

Mais il faut également prendre en considération la possibilité d'apparition que de nouveaux algorithmes de factorisations soient découverts dans l'avenir et permettent de réduire le temps de factorisation nécessaire sur grands nombres.

Il a été prouvé théoriquement qu'un modèle d'ordinateur, dit quantique, permettrait de factoriser très rapidement des entiers. Dans le cas de la mise en pratique de tel modèle, le système RSA deviendrait obsolète ainsi que le problème du logarithme discret et donc Diffie-Hellman et la cryptographie à courbe elliptique.

5.4.2 vitesse de RSA

Compte tenu de la complexité des traitements, le DES est environ 100 fois à 1000 fois plus rapide que le RSA.

5.5 Exemple d'application de RSA

Sécurisation de transactions sur l'Internet. Cartes à de crédit bancaires

Le numéro de carte de crédit est un numéro de 16 chiffres auquel on ajoute les 4 chiffres de la date d ?expiration, soit au total 20 chiffres.

On utilise RSA pour la transmission du numéro de carte de crédit sur Internet.

on choisit p et q deux grands nombres premiers $p = 976095975111111112041886431$, $q = 8345523998678341256$

on calcule $n = pq81460323853031154412157864943449033559900223014841$

et $\phi(n) = (p-1)(q-1) = 814603238530311544121578468369652837704469246373 00$

on choisit sa clé de chiffrement $e = 45879256903$ et on calcule son inverse d $(mod\phi(n))$:

$d = 614249316518661714502675899921801756121674757401 67$

Un client dont le numéro de carte de crédit est 1234 5678 9098 7654 et la date d'expiration est le 01/06 enverra donc le message $M = 12345678909876540106$. L'application d'envoi calcule $M' \equiv M^e$ $(modn)$ soit

$M' = 625176510626059110979407460361990023455526694 6485$.

Le nombre M' est transmis. À la réception on calcule : $(M')^d \equiv 12345678909876540106$ $(modn)$. Qui correspond donc bien au numéro de la carte de crédit ainsi que sa date d ?expiration.

5.6 Exercices

Exercise 5.7 *1. Si n est un entier impair et $m_1 \equiv m_2$ mod n alors $L(m_1, n) = L(m_2, n)$*

2. Si n est un entier impair alors $L(2,n) = \begin{cases} 1 & si\ n \equiv \pm 1\ mod\ 8 \\ -1 & si\ n \equiv \pm 3\ mod\ 8 \end{cases}$

3. Si n est un entier impair alors $L(m_1 m_2, n) = L(m_1, n)L(m_2, n)$. En particulier si $m = 2^k t$ où t est impair alors $L(m,n) = \begin{cases} -L(n,m) & si\ m \equiv n \equiv 3\ mod\ 4 \\ L(m,n) & si\ sinon \end{cases}$

4. Si m et n sont des entiers impairs alors $L(m,n) = \begin{cases} -L(n,m) & si\ m \equiv n \equiv 3\ mod\ 4 \\ L(m,n) & si\ sinon \end{cases}$

5.8 Théoreme de Fermat

Théorème 5.8.1 (Petit théorème de Fermat) *Soit p un nombre premier. Pour tout entier a premier avec p on a : $a^{p-1} \equiv 1$ mod p.*

Autrement dit si p est premier alors pour tout entier n on a : $a^p \equiv a$ mod p.

On peut diminuer la taille des exposants dans les calculs :

Théorème 5.8.2 *Soit p un nombre premier. Pour tout entier a premier avec p et pour tout exposant entier d on a la relation : a^d mod $p = a^{d \bmod p-1}$ mod p.*

Théorème 5.8.3 *Soit p un nombre premier. Pour tout entier a on a la relation : $a^p \equiv a$ mod p.*

Théorème 5.8.4 *Pour tout entier a premier avec n on a la relation : $a^{\varphi(n)} \equiv 1$ mod n.*

Théorème 5.8.5 (Théorème de Wilson) *: un entier p est premier si et seulement si $(p-1)! \equiv -1$ mod p*

Inutilisable en pratique pour savoir si un nombre est premier : trop lent

5.9 Méthodes de factorisation

5.9.1 Le crible d'Eratostène

C'est la méthode naïve qui consiste à diviser n par tous les nombres $< \sqrt{n}$. Il est vieux de plus de 2000 ans.

Proposition 5.9.1 *Le crible d'Erastosthène est de complexité* $\mathcal{O}(\sqrt{n}(\ln n)^2)$

Preuve : Il faut faire $\sqrt{n}$ divisions dont le temps de calcul est $\mathcal{O}(\ln n)^2)$

Imaginons un ordinateur capable de réaliser 10^9 divisions par seconde ! Si n est de l'ordre de 2^{1024} , $\sqrt{n} \backsimeq 10^{150}$. Il faudrait donc au crible d'Eratostème près de 10^{140} secondes ce qui dépasse de très loin l'âge de l'univers !

5.9.2 La méthode de Fermat

Proposition 5.9.2 *Soit* $n = p.q$, *avec* $p \geq q > 0$ *des entiers impairs. Alors*

$$n = p.q = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2 = u^2 - v^2$$

Autrement dit, $u^2 - n = v^2$ *avec* $u = (p+q)2$ *et* $v = (p-q)2$ *si* p *et* q *sont proches alors* v *est petit et* u *est très voisin de* $\sqrt{n}$.

Méthode de Fermat :
Initialiser u à $\sqrt{n}$
Tester si $u^2 - n$ est un carré
Si oui, $u^2 - n = v^2$ alors $n = (u - v)(u + v)$
Sinon, incrémenter

Exemple :
n = 387400807. On a : $[\sqrt{n} = 19683$
- $19683^2 - 387400807 = 19682$ n'est pas un carré
- $19684^2 - 387400807 = 59049 = 243^2$ est bien un carré !

On obtient ainsi la factorisation de n : $n = (19684 - 243)(19684 + 243) = 19927.19441$ On vérifie que 19927 et 19441 sont des nombres premiers.

Si p et q pas très proches, le processus peut être long.

Cet algorithme est de complexité $\mathcal{O}((\ln n)^3)$, en effet, le seul calcul à effectuer est une exponentiation modulaire de complexité $\mathcal{O}(\ln(n?1)(\ln n)^2) = \mathcal{O}((\ln n)^3)$.

5.9.3 La méthode $p-1$ de Pollard

Adapté pour la factorisation de nombres n friables. La friabilité (en anglais smoothness).

Définition 5.9.3 (Nombre B-lisse) *Soit* $n \in \mathbb{N}$ *et* $\prod_{i=1} kp_i^{e_i}$ *sa décomposition en facteurs premiers.*
n *est dit B-lisse si pour tout* $1 \leq i \leq k$ *on a* $p_i \leq B$.
n *est dit B-superlisse si pour tout* $1 \leq i \leq k$ *on a* $p_i \leq B \leq p_i^{e_i}$

Définition 5.9.4 (Nombre friable) *Soit $n \in \mathbb{N}$ et p un facteurs de n. p est dit friable si $p-1$ est B-superlisse avec B "petit".*

Principe : trouver un multiple Q de $p-1$ sans connaître p.
Fermat : pour tout $a \in \mathbb{N}$ et premier avec n on a $a^Q \equiv 1 \bmod p$.
En particulier, p divise $aQ-1$ et p divise n.
Si n ne divise pas $aQ-1$ alors $pgcd(aQ-1, n) = p$ ainsi on trouve un facteur p de n.

Impact sur cryptosystèmes à clefs publiques : Choix des paramètres p et q dans RSA et Rabin ($q \leq p$) $p-1$ et $q-1$ ne sont pas B-lisse pour B "petit".

Problème : Comment trouver Q adéquat tel que $(p-1)|Q$?

Soit π_B l'ensemble des nombres premiers $\leq B$

Si $p-1$ est supposé B-superlisse :
1) $p-1$ divise $Q_1 = ppcm\{q^l / q \in \pi_B \text{ et } q^l \leq B\}$, Plus précisément : $Q_1 = \prod_{q \in \pi_B} q^{[\ln B / \ln q]}$.

2) Si $p-1$ est supposé B-lisse :
$p-1$ divise $Q_1 = ppcm\{q^l / q \in \pi_B \text{ et } q^l \leq n\}$, Plus précisément : $Q_1 = \prod_{q \in \pi_B} q^{[\ln n / \ln q]}$.

En général, $B << n$ et donc $Q_1 << Q_2$.

Exemple : Soit à factoriser $n = 969169$. On pose B = 10 d'où $\pi_B = \{2, 3, 5, 7\}$ et $Q_1 = 23.32.5.7$

On choisit $a = 3$: $pgcd(3, n) = 1$, $aQ1 = 323.32.5.7 = 613986$, Calcul de $d = pgcd(613986 - 1, n) = 281$. Autre facteur de n : n , $d = 3449$, $969169 = 281.3449$. Remarque : $d - 1 = 280 = 23 \times 5 \times 7$ est 10-superlisse.

5.9.4 La méthode ρ de Pollard

Nous utilisons une fonction $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ "simple" pour générer une suite aléatoire de la façon suivante :

$$\begin{cases} x_0 \in \mathbb{Z}_n \\ x_{i+1} = f(x_i) \text{ pour tout } i \in \mathbb{N} \end{cases}$$

Il y a forcément des collisions c'est à dire il existe des entiers $i \neq i'$ et $x_i = x_{i'}$.

En pratique on prend pour f une fontion polynômiale de degré 2 $f(x) \equiv (x^2 + c)\ mod\ n$ avec $c \in \mathbb{Z}^*$

Si $p|n$, les x_i distincts $\bmod n$ le seront souvent $\bmod p$. On calcule les $\{x_i\}_{i \geq 0}$ jusqu'à obtenir x_j et x_k $(j < k)$

$$\begin{cases} x_j \not\equiv x_k \bmod n \\ x_j \equiv x_k \bmod p \end{cases}$$

Où p est un facteur non trivial de n en fait $p = pgcd(|x_k - x_j|, n)$

Méthode ρ de Pollard : On évalue les $\{x_k\}_{k \geq 0}$. Pour chaque nouvel élément x_k :
- Evaluer $pgcd(|x_k - x_j|, n)$ pour tous les $\{x_i\}_{0 \leq j \geq k}$. - Si aucun facteur non-trivial n'est trouvé, passer à x_{k+1}

Variante de la méthode ρ de Pollard : On évalue les $\{x_k\}_{k \geq 0}$ pour chaque k, on détermine m tel que $2^m \leq k < 2^m + 1$, (m+1 = nombre de bits de k) On pose alors $j = 2^m - 1$) (plus grand entier de

m bits) On calcule alors $pgcd(|x_k - x_j|, n)$ si on obtient un facteur non-trivial : on a gagné, sinon, on passe à x_{k+1}

Avantages :

1) un seul pgcd à calculer à chaque étape ;

2) un seul élément supplémentaire stocké en mémoire

On ne détecte pas la 1ère collision mais on n'attendra pas trop. On triple au pire le nombre d'étapes nécessaires.

Exemple : Soit $n = 20467$. On utilise $f(x) = x^2 + 1 \bmod n$ avec $x_0 = 1$.

i	x_i	j	x_j	$\lvert x_i - x_j \rvert$	$pgcd(x_i - x_j, n)$
0	1				

un facteur non trivial de n : 97 on a n = 97 . 211 Remarque : on a pas detecté la première collision ! $pgcd(x^5 - x^2, n) = pgcd(8051, 20467) = 97$.

Théorème 5.9.5 *L'algorithme ρ de Pollard a plus d'une chance sur deux de se terminer en $\mathcal{O}(\sqrt{p})$ étapes.*

Permet de factoriser des nombres de 25 chiffres avec des facteur de 12 chiffres. Choix des paramètres p et q dans RSA ou Rabin. Les facteurs p et q sont suffisamment grand !

5.9.5 La méthode du crible quadratique de Pomerance

Méthode très efficace pour les nombres ≤ 129 chiffres.

Pour $n \in \mathbb{N}$ composé, trouver x et y tels que

$$\begin{cases} x^2 \equiv y^2 \bmod n \\ x \not\equiv \pm y \bmod n \end{cases}$$

Dans ce cas : n divise $(x-y)(x+y)$ et n ne divise pas $(x \pm y)$ d'où $pgcd(x-y, n)$ est un diviseur strict de n.

Problème : Comment construire x et y ?

Construction de x et y satisfaisant $x^2 \equiv y^2 \bmod n$

On pose $m = [\sqrt{n}]$ et $f(X) = (X+m)^2 - n$. On a donc pour tout $t \in \mathbb{Z}$: $(t+m)^2 \equiv f(t) \bmod n$. On évalue cette relation pour plusieurs valeurs de t :

$$\begin{cases} (t_1+m)^2 \equiv f(t_1) & \bmod n \\ (t_2+m)^2 \equiv f(t_2) & \bmod n \\ \cdots & \\ (t_s+m)^2 \equiv f(t_s) & \bmod n \end{cases}$$

On choisit r relations $\{i_1, \cdots, i_r\}$ tel que $f(t_{i_1}) \cdots f(t_{i_r})$ soit un carré, d'où $x = (t_{i_1}+m) \cdots (t_{i_r}+m)$ et $y = \sqrt{f(t_{i_1}) \cdots f(t_{i_r})}$ conviennent.

Soit $B \geq 0$ et $F(B) = \{-1\} \cap \{p \text{ premier} / p \leq B\} = \{p_1, p_2, \cdots, p_k\}$ On suppose avoir $s > k$ relations $\{i_1, \cdots, i_s\} / f(t_{i_j})$ est B-lisse : Pour tout $i \in \{i_1, \cdots, i_s\} / f(t_{i_j})$ on a $f(t_{i_j}) = \prod_{j=1} k p_j^{\alpha_{ij}}$ on note $a_{ij} = \alpha_{ij} \bmod 2$

$s > k$ alors les lignes v_i de la matrice $(a_{ij})_{1\leq i,j\leq k}$ sont liés dans $\mathbb{F}_2^k$.
Pivot de Gauss donne la relation de dépendance linéaire sur $\mathbb{F}_2$ $v_{i_1} + \cdots + v_{i_r} = 0$ pour $r \leq s$

En particuier

$$f(t_{i_1}) \cdots f(t_{i_r}) = \left(\prod_{j=1} kp_j^{\frac{\alpha_{i_1,j}+\cdots+\alpha_{i_1,j}}{2}} \right)$$

Il reste à montrer comment trouver t tel que $f(t)$ soit B-lisse :
Tester pour $t \in \{0, \pm 1, \pm 2, \cdots\}$ si $f(t)$ est B-lisse. Trop long : il faut diviser $f(t)$ par tous les p premier $\leq B$.

Méthode du crible : On fixe $c \in \mathbb{N}^*$ et un intervalle $T_c = \{-c, \cdots, 0, \cdots, c\}$ appelé intervalle de crible.
On calcule $f(t)$ pour tous les $t \in T_c$ Pour $p \in F(B)$, on divise $f(t)$ par la plus grande puissance de p divisant $f(t)$. $f(t)$ est B-lisse si on obtient ± 1 à la fin de ce processus.

Exemple

5.9.6 La méthode GNFS

5.10 Résolution du problème du logarithme discret

5.10.1 Méthode naïve

Recherche exhaustive : Tester, pour $x = 0, 1, 2, \cdots$, si $g^x = h$ est vérifiée dans G.

Coût mémoire : $\mathcal{O}(1)$ $(x, g, gxeth)$ Coût en nombre d'opérations : $\mathcal{O}(1)$ Il faut trouver des méthodes plus efficaces !

Notamment $\mathcal{O}(\sqrt{n})$ opérations

5.10.2 Méthode Baby Step Giant Step de Shanks

Description de la méthode Soit $m = [n]$. On fait la division euclidienne de x par m : il existe $q, r \in \mathbb{Z} : x = qm + r$ où $0 \leq r \leq m$.

Principe : trouver q et r pour en déduire x en utilisant : $hg^{-r} = (g^m)^q$.

1) Baby Step : Déterminer l'ensemble $B = (hg^{-r}, r)$ où $0 \leq r \leq m$. si il existe $r \in [0, m[$ tel que $(1, r) \in B$ alors $x = r$.

2) Giant Step : Soit $t = g^m$. Pour $q = 1, 2, \cdots$ faire si il existe $r \in [0, m[$ tel que $(t^q, r) \in B$, alors $x = qm + r$.

Complexité : mémoire $\mathcal{O}(\sqrt{n})$; opérations $\mathcal{O}(\sqrt{n})$.

Exemple :

5.10.3 Méthode ρ de Pollard

$\mathcal{O}(\sqrt{n})$ opérations mais seulement $\mathcal{O}(1)$ place mémoire !

5.10.4 Méthode de réduction de Pohlig-Hellman

5.10.5 Méthode de calcul d'indices

Exercice : Écrire un programme utilisant cette méthode.

5.11 Exercices

Exercice 1. Bob utilise RSA et publie sa clé publique $N = 187$ et $e = 3$.
1. Encoder le message $m = 15$ avec la clé publique de Bob.
2. En utilisant le fait que $\phi(N) = 160$, retrouver la factorisation de N, puis la clé privée de Bob.

Exercice 2. Soit $N = 39$ et $e = 29$.
1. Calculer d.
2. Coder le message $m = 2$ et vérifiez le résultat en le décodant.

Exercice 3. La clef publique est $(N, e) = (35, 5)$, on reçoit le message $M = 10$, retrouver le message original M.

Exercice 4. Bob1 et Bob2 ont pour clé publique RSA respectivement (N, e_1) et (N, e_2) avec e_1 et e_2 premiers entre eux. Alice envoie le même message M crypté par les clés publiques RSA de Bob1 et Bob2 en c_1 et c_2. Expliquer comment Eve, qui intercepte les deux messages chiffrés et qui connaît les clés publiques de Bob1 et Bob2, peut retrouver le message clair M.

Exercice 5. Soit le message $M = 11$ à chiffrer avec le crypto-système RSA défini avec les clés publiques $e = 3$ et $N = 187$. Donner le chiffré C de M. Sachant que $N = pq$, avec $p = 11$ et $q = 17$, déchiffrer $c' = 23$

Exercice 6. Soit $N = pq$ impair avec $p > q$ sont des entiers premiers.
1. Vérifier que $N = t^2 - s^2 = (t + s)(t - s)$ avec $t = \frac{p+q}{2}$ et $s = \frac{p-q}{2}$
2. On suppose que p est très proche de q, montrer que t est supérieur à $\sqrt{N}$ et très proche de N.
3. Utiliser ces remarques pour factoriser $N = 4397231$.
4. Conclure

Exercice 7. a) On suppose que Alice et Bob utilisent l'entier n et RSA avec deux clés publiques e_A et e_B premières entre elles. On suppose que $Caroline$ envoie le même message chiffré m^{e_A} et m^{e_B} à Alice et à Bob. Montrer que Eve qui écoute les communications peut retrouver facilement le message m.

b) À fin d'améliorer la sécurité des messages Bob choisit deux exposants e_1 et e_2 et demande à Alice de chiffrer d'abord son message par e_1 , pour obtenir $c_1 = m^{e_1}$ puis de re-chiffrer par e_2 pour obtenir $c_2 = c_1^{e_2}$ et d'envoyer c_2. Est-ce que ce double chiffrement améliore la sécurité. Si oui pourquoi, si non pourquoi.

Chapitre 6

Cryptographie à clef publique : ElGamal

Sommaire

ElGamal est un cryptosystème asymétrique basé sur le problème du logarithme discret, comme le protocole de Diffie-Hellman . Ce crypto-système est inventé en 1984 par Taher Elgamal, un cryptographe égyptien.

Son inconvénient par rapport à RSA est que le message chiffré est deux fois plus long que le message clair.

Cet algorithme est utilisé par le logiciel libre PGP, et d'autres systèmes de chiffrement. Il n'est pas breveté contrairement à RSA. Il peut être utilisé pour le chiffrement et la signature électronique (voir plus tard).

Le calcul de logarithmes sur le corps de nombres réels est facile. Mais ce calcul est difficile sur les corprs de Galois $\mathbb{F}_p$.

Courbes elliptiques

ElGamal Signature

6.1 Description de ElGamal

génération de la clef

Destinataire :

- choisit deux paramètres publiques : un nombre premier p et un générateur g du groupe $(\mathbb{Z}_p^*, .)$
- choisit aléatoirement un nombre a dans $[1, \cdots, p-2]$ et calcule $\alpha = g^a \mod p$.
- garde secrète sa clé a et publie (p, g, α).

Emetteur : chiffrement

- exprime son message sous la forme d'un nombre M entre 0 et $p-1$ (quitte à le décomposer).

- choisit aléatoirement un nombre b dans l'intervalle $[1, \cdots, p-2]$ et calcule $\beta = g^b \bmod p$.
- chiffre alors son message M en $C = \alpha^b.M \bmod p$.
- envoie au destinataire le couple (β, C).

Destinataire : déchiffrement
- Pour déchiffrer le message, le destinataire calcule $x = p - 1 - a$, calcule $\beta^x.C \bmod p$ et retrouve le message M. En effet :

$$\beta^x.C = (g^b)^x C = g^{b(p-1-a)}.\alpha^b.M = g^{b(p-1-a)}.g^{ab}.M \equiv g^{b(p-1)}.g^{-ab}.g^{ab}.M \equiv M \bmod p$$

car $g^{b(p-1)} \equiv 1 \bmod p$ d'après le petit Théorème de Fermat.

6.2 Remarques

1) Il est conseillé de changer la valeur de son paramètre b pour chaque nouveau message. Supposons, que deux messages M et M' soient cryptés avec la même valeur de b et qu'une tierce personne connaisse le texte clair M : $C = \alpha^b.M \bmod p$ et $C' = \alpha^b M' \bmod p$ Alors,

$$C'C^{-1} = \alpha^b M M'^{-1}(\alpha^b)^{-1} \bmod p = M'M^{-1} \bmod p$$

et

$$M' = C'(M')^{-1}M \bmod p$$

et on déduit M'.

2) L'utilisation du paramètre aléatoire a renforce la sécurité car le même message M chiffré à 2 moments différents donnera deux messages chiffrés différents.

Exemple

Supposons qu'Alice et Bob choisissent le nombre premier $p = 1259$ et $g = 3$.

Alice choisit $x = 144$ et calcule $3^{144} = 572$ [1259].

Alice envoie 572 à Bob.

Bob choisit $y = 731$ et calcule $3^{731} = 900$ [1259] Bob envoie 900 à Alice.

Alice calcule $900^{144} = 572^{731} = 26$ [1259] Donc Alice et Bob peuvent utiliser la clé $K = 26$.

6.3 Exercices

Exercice 1. Alice choisit $p = 97$ et $g = 13$.

(a) Elle choisit aléatoirement un nombre a, disons 45, dans l'intervalle $[1, \cdots, 95]$.

(b) Elle calcule $\alpha = (13^{45} \ mod \ 97) = 20$.

(c) Elle publie sa clé (97, 13, 20) et garde secrète sa clé 45.

Bob veut envoyer le message RAS à Alice.

(a) En utilisant le code ASCII, son message est 118 101 119.

(b) Il le découpe en nombres entre 0 et 97 : 11 81 01 11 09.

(c) Il choisit aléatoirement un nombre b, disons 35, dans l ?intervalle $[1, \cdots, 95]$.
(d) Il calcule $\beta = 13^{35}\ mod\ 97 = 71 mod 97$.
1. Vérifier que le chiffré de son message est (71, 21 40 46 21 26). 2. Comment Alice déchiffre-t-elle le message de Bob ? Déchiffrer-le.

Exercice 2. Soit G un groupe cyclique, soit x un ?élément d'ordre r et y un élément d'ordre s.

1. Montrer que le sous-groupe engendré par x et y a pour cardinal $ppcm(r, s)$.
2. Comment peut-on choisir un générateur de la forme $g = x^i y^j$?
3. Soit $G = (\mathbb{Z}/41\mathbb{Z})^?$; calculer l'ordre de 2 (resp. l'ordre de 3) et en déduire un générateur en utilisant la question précédente.
4. Combien de générateurs le groupe $(\mathbb{Z}/41\mathbb{Z})^?$ possède-t-il ?

Exercice 3. Danny veut partager un secret n entre Alice, Bob et Charlie, sans que deux d'entre eux puissent le reconstruire. Il fabrique un groupe G , un générateur g de grand ordre dans G , et une décomposition $n = a + b + c$, puis donne g^a à Alice, g^b à Bob, et g^c à Charlie. Ils peuvent ainsi reconstruire g^n en multipliant leurs valeurs : $g^n = g^a g^b g^c$. 1) La donnée de g^n ne permet pas facilement de retrouver la valeur de n . Comment contourner ce problème ?

2) En supposant que le secret est maintenant g^n, Alice et Bob peuvent-ils le reconstruire sans l'aide de Charlie ?

3) En supposant encore que le secret est n, qu'Alice connaît a, que Bob connaît b, et qu'ils connaissent tous les deux g^c, peuvent-ils retrouver n ?

Exercice 4.

Chapitre 7

Fonction de Hachage

Sommaire

Objectif de la cryptographie :

- la confidentialité des données ; (réalisée par les algorithmes de chiffrement et de déchiffrement, DES, IDEA, RC4, RSA, ElGamal etc)

- l'intégrité des données : prévention de modification non autorisée de données. (réalisée par les fonctions de hachage).

- l'authentification : vérifier que le message provient bien de celui qui prétend en être l'émetteur. (réalisée par les fonctions de hachage).

- la non-répudiation (ou non-désaveu) : prouver qu'un message a bien été émis par son expéditeur et que ce dernier ne peut nier l'avoir transmis. (Se résout par la signature électronique.)

Une fonction de hachage transforme un message de taille arbitraire en une chaîne de taille fixe (typiquement entre 128 et 512 bits).

Sous Linux la commande `md5sum` permet de calculer le haché d'un fichier. Pour l'utiliser `md5sum nomFichier` et pour plus de détail sur cette commande `man md5sum` .

Sous Windows on peut télécharger le programme `MD5Summer` de [1] Il est gratuit et à interface graphique.

Applications des fonctions de hachage : signature électronique, recherche dans une table stockage de mots de passe, vérification de l'intégrité d'un fichier : téléchargement, confirmation de connaissance, etc.

Protection de mots de passe : au lieu de stocker tous les mots de passe dans le serveur pour l'authentification d'utilisateurs, il vaut mieux stocker le hache des mots de passe.

Confirmation de connaissance : si quelqu'un veut prouver qu'il connaît un secret sans le révéler dans l'immédiat, il peut publier le haché de ce secret. Une fois le secret révélé, il est facile de vérifier ses dires.

7.1 Définitions

Définition 7.1.1 (Fonction à sens unique) *Une fonction $H : X \longrightarrow Y$ est une fonction à sens unique si pour tout x il est facile de calculer $H(x)$ mais, sachant $H(x)$, il est très difficile de trouver x.*

Exemple 7.1.2 *Soit p et q deux nombres premiers entre eux et grands :*
- $H(p,q) = p.q$ et ;
- $H(x) = x^2 \ mod \ n$ où $n = pq$;
sont des fonctions à sens unique. La factorisation de grands entiers La fonction logarithme discret : Soit p un grand nombre premier et g une racine primitive modulo p, il s'agit de retrouver a connaissant A et g / $g^a = A \ mod \ p$ avec $0 \leq a \leq p-2$.

Définition 7.1.3 (Fonction de Hachage) *Une fonction de hachage H est une application facilement calculable qui transforme une chaîne binaire de taille quelconque t en une chaine binaire de taille fixe n (n petit entre 100 et 200), appelée empreinte, haché ou condensé de hachage.*

Définition 7.1.4 *Une fonction est dite :*
- résistante aux collisions si est difficile de trouver deux messages M et M' distincts tels que $H(M) = H(M')$.
- résistante aux pré-images si étant donné un haché y, il est difficile de trouver un message M tels que $H(M) = y$.
- résistante aux secondes pré-images si étant donné un message M, il est difficile de trouver M' tel que $H(M) = H(M')$.

Une fonction de hachage est sécurisée si elle est à la fois résistante aux collisions, aux pré-images et aux secondes pré-images :

Remarques :

1. www.md5summer.org/

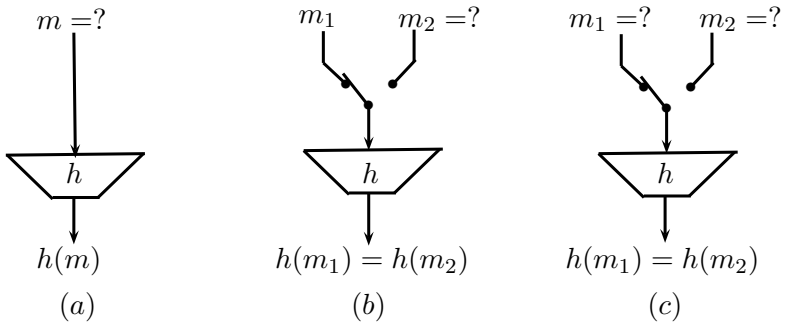


FIGURE 7.1 – (a) : résistance à la pré-image, (b) : résistance à seconde pré-image, (c) : résistance aux collisions

Résistance à la collision $\Longrightarrow$ résistance à la seconde pré-image $\Longrightarrow$ résistance à la pré-image.

La résistance à la pré-image est équivalente à fonction à sens unique.

Soit $h(x) = x^2 \ mod \ p$ pour p premier. Ce n'est pas une fonction à sens unique car le calcul de racine carré modulo p est facile. Pour $n = pq$ avec p, q premiers, $h(x) = x^2 \ mod \ n$ est à sens unique, car le calcul de racine carré modulo n est r réputé difficile. Par contre $h(x) = (x+n)^2 \ modn = x^2$ n'est pas résistante aux collisions (faibles ou fortes), car $h(x) = h('x)$.

La résistance aux collisions n'implique pas la résistance à la pré-image. En effet, soit f une fonction résistante aux collisions dont le haché est de longueur n. On définit une fonction de hachage h de longueur $n+1$ par

$$h(x) = \begin{cases} 1||x & \text{si } \ |x| = n \\ 0||f(x) & \text{sinon} \end{cases}$$

h est aussi résistante aux collisions. Mais si on se donne $1||x$ alors $h(x) = 1||x$.

7.1.1 Paradoxe des anniversaires

Dans un groupe de n personnes, quelle est la probabilité pour que deux d'entre-elles aient leur anniversaire le même jour ? (c-à-d même jour et mois, mais pas forcément l'année).

Par exemple pour $n = 23$, Intuitivement cette probabilité est-elle beaucoup proche de 0 ou de $\frac{1}{2}$?

$$P(pas\ de\ collusion\ entre\ 2\ personnes) = \left(1 - \frac{1}{365}\right)$$

$$P(pas\ de\ collusion\ entre\ 3\ personnes) = \left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)$$

$$P(pas\ de\ collusion\ entre\ n\ personnes) = \left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)\cdots\left(1 - \frac{n-1}{365}\right)$$

pour $n = 365$ il y aura sûrement collision.

$$P(au\ moins\ une\ collusion) = 1 - P(pas\ de\ collusion) = \left(1 - \frac{1}{365}\right)\cdots\left(1 - \frac{23-1}{365}\right) = 0,507 \approx \frac{1}{2}$$

Pour 40 personnes cette probabilité est d'environ 90%.

On cherche cette probabilité dans un cas plus général. Pour une fonction de hachage il n'y a pas 365 valeurs mais 2^n où n est la longueur de $h(m)$.

Question : combien de messages $\{m_1, m_2, \cdots, m_k\}$ à hacher pour avoir une chance raisonnable d'avoir $h(m_i) = h(m_j)$ où $i, j \in \{1, 2, \cdots, k\}$?

$$P(\textit{pas de collusion}) = \left(1 - \frac{1}{2^n}\right)\left(1 - \frac{2}{2^n}\right)\cdots\left(1 - \frac{n-1}{2^n}\right) = \prod_{i=1}^{k-1}\left(1 - \frac{i}{2^n}\right)$$

Rappelons que $e^{-x} = 1 - x + \sum_{j=2}^{+\infty}(-1)^j \frac{x^j}{j!}$ pour tout $x \in \mathbb{R}$, et pour $x << 1$ on a $e^{-x} \approx 1 - x$

$$P(\textit{pas de collusion}) \approx e^{-\frac{i}{2^n}} \approx e^{-\frac{1+2+\cdots+k-1}{2^n}} \approx e^{-\frac{(k-1)k}{2.2^n}}$$

Le but est de trouver combien de messages x_i faut-il pour avoir une collision ? on cherche alors k.

$$p = 1 - e^{-\frac{(k-1)k}{2.2^n}}$$

$$(k-1)k \approx 2^{n+1} \ln\left(\frac{1}{1-p}\right)$$

on suppose $k >> 1$ d'où $k^2 \approx (k-1)k$ d'où

$$k \approx \sqrt{2^{n+1} \ln\left(\frac{1}{1-p}\right)}$$

$$k \approx 2^{\frac{n+1}{2}} \sqrt{\ln\left(\frac{1}{1-p}\right)} \tag{7.1}$$

Pour $p = \frac{1}{2}$ on a $k \approx 2^{\frac{n}{2}}$.

Remarque 7.1.5 *Comme conséquence du paradoxe des anniversaires, pour une fonction de hachage $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ le nombre de messages à hacher pour trouver une collusion avec une probabilité $\frac{1}{2}$ est de $2^{\frac{n}{2}}$.*

La recherche brutale de collisions a plus d'une chance sur 2 d'aboutir après seulement $\mathcal{O}(2^{\frac{m}{2}})$ essais !

On est sûr d'aboutir après $\mathcal{O}(2^m)$ essais.

Si $n = 80$, $k \approx 2^{40,2}$.

C'est pour cette raison que toutes les fonctions de hachage sont de longueur ≥ 128.

7.2 Construction de fonctions de hachage

Une fonction de compression est une fonction qui transforme toute chaîne d'une taille fixée n en une chaîne de taille fixée m avec $n > m$: $f : \{0,1\}^n \longrightarrow \{0,1\}^m$.

La construction de fonction de hachage nécessite deux ingrédients importants :
1) une fonction de compression de $\{0,1\}^t \longrightarrow \{0,1\}^n$ où t et n sont des entiers fixes avec $t > n$.

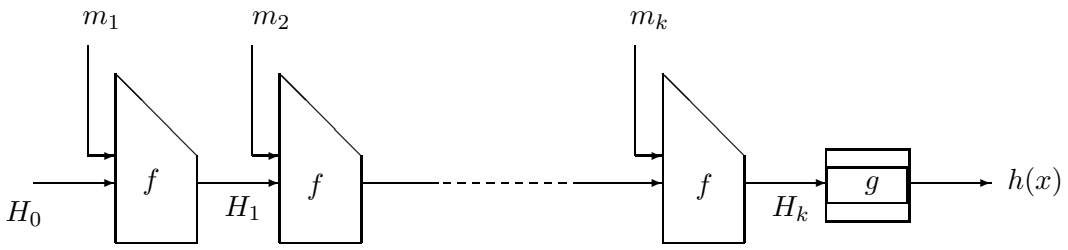


FIGURE 7.2 – Construction d'une fonction de hachage h à partir d'une fonction de compression f

2) un extenseur de domaine, qui à partir d'une fonction de compression donné, produit une fonction à entrée arbitraire.

7.2.1 Construction de Merkle-Damgård

Cette construction a été proposée indépendamment par R. C. Merkle [6] et I. B. Damgård [5] fin des années 1989.

La construction de Merkle-Damgård (1989) est la méthode la plus répondu pour fabriquer des fonctions de hachage. Elle se base sur une fonction de compression.

Comme fonction de compression on peut considérer une fonction de chiffrement, car celle-ci prend en entrée un bloc clair de longueur n et la clé de longueur k et le bloc chiffré en sortie est de longueur n.

Elle permet de réduire le problème de la construction d'une fonction de hachage résistante à la recherche de collision/preimage à celui de la construction d'une fonction de compression résistante à la recherche de collision/preimage : si l'on ne trouve pas de collision/preimage pour h, alors on ne peut trouver de collision/preimage pour H.

La plus part des fonctions de hachage l'utilise. Elle se fait à partir d'une fonction de compression.

Théorème 7.2.1 (Damgård) *Si la fonction de compression f est résistante aux collisions alors la fonction de hachage obtenue est résistante aux collisions.*

Se base sur une fonction de compression $h : \{0,1\}^b \times \{0,1\}^n \longrightarrow \{0,1\}^n$

Application au calcul de $H(M)$:

Application d'un padding à M pour avoir $|M| = k.b$ bits : on ajoute un 1 à droite de M puis suffisamment de 0. On ajoute le 1 et $b-1$ zéros même si $|M|$ est déjà multiple de b.

Découpage du message M obtenu après le padding en blocs de taille b

$M = m_1 m_2 \cdots m_{k-1} m_k$ avec $|M_i| = b \in i \in [1,k]$. Itération de la fonction h . Voir Figure 7.2.

Itération sur les blocs : $H_0 = IV$: Valeur initiale

$$H_0 = 0 \cdots 0, \quad n \text{ fois}$$

$$H_i = f(H_{i-1}||m_i)$$

$$h(M) = H_k$$

Remarque : On peut utiliser les chiffrements par bloc comme fonction de compression.

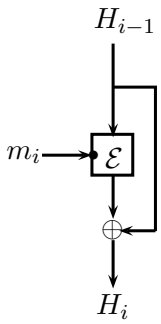


FIGURE 7.3 – Construction de Davies-Meyer

7.2.2 Construction de Davies-Meyer

Voir Figure 7.3 Soit la fonction de chiffrement $\mathcal{E}$. On partage le message m à hacher $m = m_0 m_1 \cdots m_k$ (on complète si besoin) m_i est de même longueur que la longueur de la clé de $\mathcal{E}$. le message m à hacher Les blocs d'entrée m_i sont les clés de la fonction de chiffrement $\mathcal{E}$. H_i est le bloc à chiffrer par $\mathcal{E}$.

H_0 valeur initiale.

$$H_i = \mathcal{E}_{m_i}(H_{i-1}) \oplus H_{i-1}$$

et

$$h(m) = m_k$$

7.2.3 Construction de Matyas-Meyer-Oseas

La construction générique de Matyas-Meyer-Oseas (appelé aussi CBC-MAC) Voir Figure 7.4 est basée sur une fonction de chiffrement par bloc $\mathcal{E}_k$: Elle sera améliorée par Miyaguchi-Preneel.

Soit la fonction de chiffrement $\mathcal{E}$. On partage le message m à hacher $m = m_0 m_1 \cdots m_k$ (on complète si besoin) m_i est de même longueur qu'un bloc à chiffrer de $\mathcal{E}$. Le message m à hacher Les blocs d'entrée m_i sont à chiffrer par $\mathcal{E}$. H_i sont les clés de $\mathcal{E}$. On considère une fonction g qui transforme n bits en chaîne de même longueur que la clé.

H_0 valeur initiale.

$$H_i = \mathcal{E}_{g(H_{i-1})}(m_i) \oplus m_i$$

et

$$h(m) = m_k$$

7.2.4 Miyaguchi-Preneel

Voir Figure 7.5

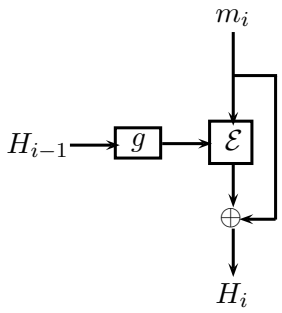


FIGURE 7.4 – Construction de Matyas-Meyer-Oseas

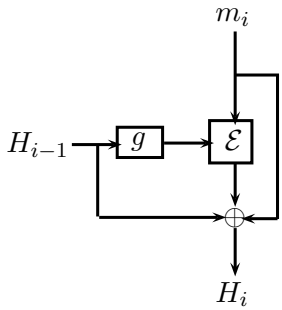


FIGURE 7.5 – Construction de Miyaguchi-Preneel

7.3 Applications des fonctions de hachage : MDC et MAC

Deux types d'utilisation

Une fonction de hachage est aussi appelée MDC (Modification Detection Code). Elle est sans clef. On peut l'utiliser pour s'assurer uniquement de l'intégrité de messages. Par exemples MD4, MD5, SHA-1.

HMAC est utilisé dans les protocoles TLS (Transport Layer Security) et IPsec

Par contre pour l'authentification on utilise des fonctions de hachage avec clef, appelé aussi MAC (Message Authentication Code). Les MAC permettent de vérifier l'intégrité et l'authentification du message en même temps. Elle prouve que l'expéditeur possède la clé

Construction de MAC Idée : concaténer la clé avec le message à authentifier. HMAC (standard) Paramètres K clé jusqu'à 512 bits.

h fonction de hachage. M message à authentifier. $||$ symbole de concaténation.

$$MAC(K, M) = h(M||K)$$

ou

$$MAC(K, M) = h(K||M)$$

Cette construction est faible.

Une construction plus sécurisée est la suivante.

7.3.1 Construction HMAC

Construction de M. Bellare, R. Canetti et H. Krawczyk : Soit h une fonction de hachage qui produit des hachés de longueur ℓ. Soit M le message à authentifier et K la clé partagée par les deux correspondants. La longueur de K doit être inférieure à 64 octets. On complète K par des zéros à droite pour avoir une longueur de 64 octets.

On définit deux chaînes fixes (i=inner, o=outer)

ipad= l'octet 0x36 répété 64 fois,

opad = l'octet 0x5C répété 64 fois,

$$HMAC_K(M) = h(K \oplus \ opad||h(K \oplus \ ipad||M)) \tag{7.2}$$

La longueur recommandée de K est au moins ℓ. Une grande longueur n'augmente pas la sécurité de façon significative.

Optionnellement, HMAC permet la troncature du résultat final à 80 bits.

7.4 Preuve sans transfert de connaissance

Exemple : Un jeu de Pile ou face par Téléphone.

Ils se donnent un ensemble E par exemple $E = \{0, 1, \cdots, n\}$ et une partition $E = X_0 \cup X_1$ de E, $X0$ sera les entiers pairs de E et $X1$ les entiers impairs de E. Puis ils se mettent d'accord sur une fonction de hachage, h, de E dans un ensemble $\{0, 1\}^n$. On considère le protocole suivant :

1. Alice choisit un élément $x \in E$ aléatoirement (c'est le jet de la pièce), calcule $y = h(x)$ et communique y à Bob (Bob ne peut pas retrouver x à partir de y car h est à sens unique).

2. Bob choisit son bit aléatoire $b \in \{0, 1\}$ et l'annonce à Alice.

3. Alice déclare qui a gagné suivant que $x \in X_b$ ou non : elle prouve sa bonne foi en révélant x.

7.5 Fonction de hachage MD5

La première fonction de hachage cryptographique a été est développée par RSA Security, Inc, nommée MD (message digest) propriétaire et jamais publiée. Par contre la version MD2 a été publiée et la première fonction de hachage largement utilisé.

Quand Merkle en 1990 a proposé SNEFRU qui était beaucoup plus rapide que MD2, RSA Security, Inc a répondu par MD4. MD3 a été développée mais jamais publiée ou utilisé. SNEFRU a été attaqué en 1991 par la cryptanalyse différentielle. MD5 est venu pour combler certaines failles découverte dans MD4, considéré actuellement non sécurisé. MD5 est légèrement moins rapide que MD4. Depuis 2004, MD5 est considéré partiellement cassé, car il est connu que la fonction de compression qu'utilise admet des collisions. MD5 n'est plus considérée comme sûr aujourd'hui.

L'algorithme de hachage MD5 (Message Digest Algorithm) a été développé par Ron Rivest en 1991. C'est une version renforcée de MD4 (1990) pour être plus rapide sur les machines 32 bits. Elle prend en entrée un message de longueur arbitraire et produit en sortie une empreinte de 128 bits. Conçue pour les processeurs 32 bits.

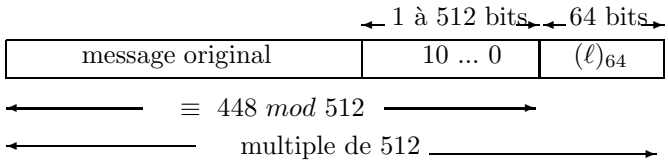


FIGURE 7.6 – Padding de la fonction de hachage MD5

Une faille a été trouvée en 1996 et des collision trouvées en quelques heures en 2004. Elle est encore largement utilisée en pratique. Mais à éviter !

Notations : "+" denote l'addition de mots (mots=32 bits) modulo-2^{32}.
$X \wedge Y$: X and Y (bit à bit).
$X \vee Y$: X or Y (bit à bit).
$X \oplus Y$: X xor Y (bit à bit).
$\bar{X}$: complement de X (bit à bit).
$X \hookleftarrow s$: décalage circulaire à gauche de X par s positions ($0 \leq s \leq 31$).

Fonctions primitives : On définit 4 fonctions, dites primitives, dont les arguments sont des mots de 32 bits et produisent des mots de 32 bits aussi.

$$f(X,Y,Z) = (X \wedge Y) \vee (\bar{X} \wedge Z)$$
$$g(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \bar{Z})$$
$$h(X,Y,Z) = X \oplus Y \oplus Z$$
$$i(X,Y,Z) = Y \oplus (X \vee \bar{Z})$$

X	Y	Z	f	g	h	i
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

TABLE 7.1 – Table de vérité des fonctions f, g, h, i

Tableau T MD5 utilise un tableau T de 64 éléments où $T[i]$ est le i^{eme} élément : $T[i] = [4294967296|sin(i)|]$ où $[\,]$ désigne la partie entière et i est en radians. Puisque $4294967296 = 2^{32}$ chaque élément de T peut être représenté sur 32 bits.

MD5 se fait en cinq étapes :

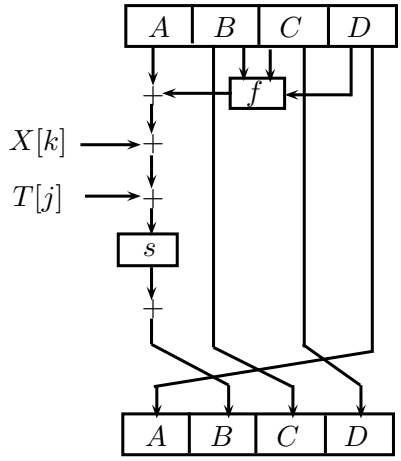


FIGURE 7.7 – Fonction ? ? ? de hachage MD5

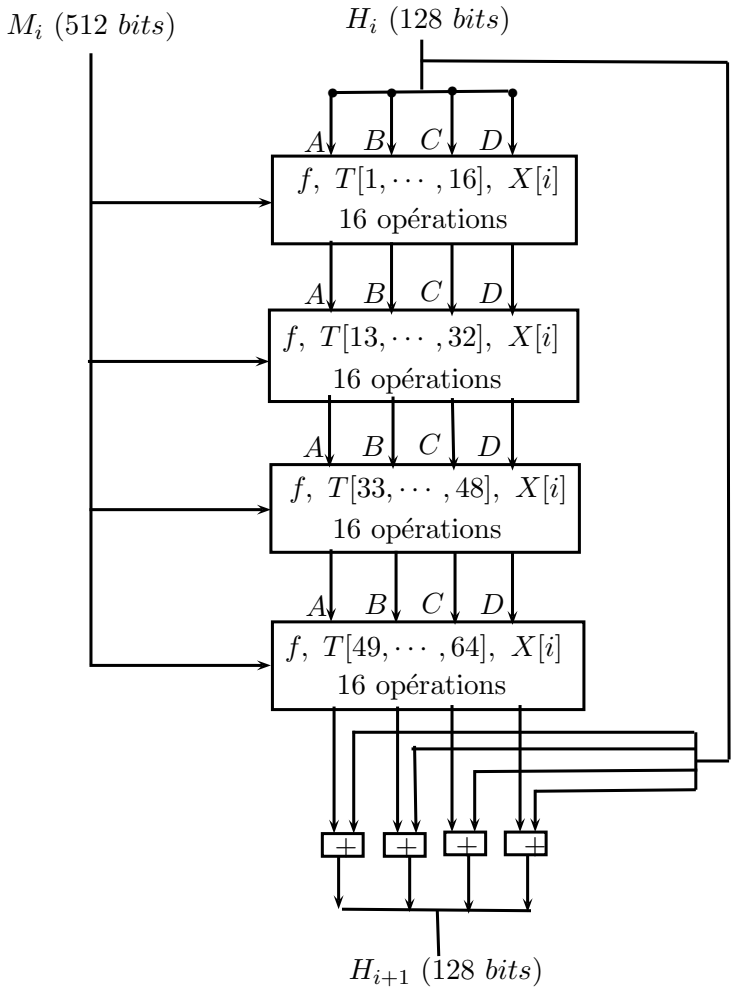


FIGURE 7.8 – Fonction de hachage MD5

Groupe t	tour j	Constante K_t	Fonction f_t
1	$0 \cdots 19$	$K_1=$	$f_1(B,C,D) =$
2	$20 \cdots 39$	$K_2=$	$f_2(B,C,D) =$
3	$40 \cdots 59$	$K_3=$	$f_3(B,C,D) =$
4	$60 \cdots 79$	$K_4=$	$f_4(B,C,D) =$

TABLE 7.2 – Fonctions et constantes dans SHA1.

Première étape : padding Soit un message M de longueur $\ell \leq 2^{64}$ bits. On complète à droite M par un 1, et suffisamment de 0 pour que le message étendu ait une longueur congruente à 448, modulo 512. Cette opération est faite même si ℓ est déjà congru à 448 mod 512.

Deuxième étape : appending Puis on ajoute à ce message la valeur de ℓ, codée en binaire sur 64 bits. Si $\ell > 2^{64}$, les 64 bits à gauche sont utilisés. On obtient donc un message M dont la longueur totale est un multiple de 512 bits, ou encore multiple de 16 mots (1 mot =32 bits)

Soit $M = M[0] \cdots M[N-1]$ le message obtenu, où chaque $M[i]$ désigne un mot (32 bits). Puisque la longueur de M est multiple de 512, N est divisible par 16.

On va travailler itérativement sur chacun des blocs $M[i]$

Troisième étape : Initialisation On considère les valeurs initiales suivantes qui sont des mots de 32 bits chacun écrit en hexadécimal.

```
A: 01 23 45 67 ;
B: 89 ab cd ef ;
C: fe dc ba 98 ;
D: 76 54 32 10 .
```

Quatrième étape : calcul itératif Traitement des blocs

```
For i = 0 to N/16-1 do

/* Copier le bloc i dans  X. */
     For j = 0 to 15 do
       Set X[j] to M[i*16+j].
     end  fin de la boucle j

/*  Affecter A à AA, B à BB, C à CC, et D à DD.  */
     AA = A
     BB = B
     CC = C
     DD = D
```

Tour 1. Soit $[ABCD\ k\ s\ j]$ dénote l'opération $A = B + ((A + f(B,C,D) + X[k] + T[j]) \hookleftarrow s)$, (voir Figure 7.7) faire les 16 opérations :

```
[ABCD  0  7  1]    [DABC  1 12  2]    [CDAB  2 17  3]    [BCDA  3 22  4]
[ABCD  4  7  5]    [DABC  5 12  6]    [CDAB  6 17  7]    [BCDA  7 22  8]
[ABCD  8  7  9]    [DABC  9 12 10]   [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]
```

Tour 2. Soit $[ABCD\ k\ s\ j]$ dénote l'opération $A = B + ((A + g(B,C,D) + X[k] + T[j]) \hookleftarrow s)$, (voir Figure 7.7) faire les 16 opérations :

```
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]
```

Tour 3. Soit $[ABCD\ k\ s\ j]$ dénote l'opération $A = B + ((A + h(B,C,D) + X[k] + T[j]) \hookleftarrow s)$, (voir Figure 7.7) faire les 16 opérations :

```
[ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
[ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
[ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
[ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]
```

Tour 4. Soit $[ABCD\ k\ s\ j]$ dénote l'opération $A = B + ((A + i(B,C,D) + X[k] + T[j]) \hookleftarrow s)$, (voir Figure 7.7) faire les 16 opérations :

```
[ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
[ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
[ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
[ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
```

Faire les additions suivantes :

```
A = A + AA
B = B + BB
C = C + CC
D = D + DD

Fin de la boucle i
```

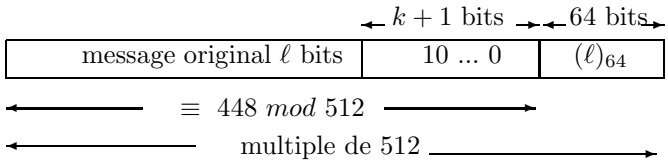


FIGURE 7.9 – Padding de la fonction de hachage SHA1

Cinquième étape Le haché est produit comme A, B, C D

7.6 Fonction de hachage SHA-1

SHA-1 (Secure Hash Algorithm 1), comme MD5, est basé sur MD4. (donc sur la construction de Merkle-Damgård). Son fonctionnement très similaire à MD5. Mis au point et publié en 1993 par l'agence de sécurité nationale américaine (NSA). C'est un standard officiel aux US pour usage avec le schéma de signature DSA depuis 1995 et jusqu'en 2012.

L'algorithme SHA-1 traite par blocs de 512 bits et produit une empreinte de 160 bits en sortie. Il s'effectue en 80 tours qui sont divisés en 4 groupes . Le SHA-1 traite les messages d'au plus 2^{64} bits en entrée.

SHA-1 a été cassé en février 2005 par Wang, Yin et Yu, qui ont montré que des collisions pouvaient être trouvées en 2^{69} essais au lieu de 2^{80}. Puis 2^{63} d'après des travaux récents, ce qui rend SHA-1 plus vulnérable.

SHA-2 est destiné à remplacer SHA-1. Les différences principales résident dans les tailles de hachés possibles 256, 384 ou 512 bits, qui sont désignés respectivement par SHA256, SHA384 ou SHA512 bits. En février 2004, le NIST a introduit SHA-224. SHA-224 est identique à SHA-256, mais utilise des valeurs initiales différentes et tronque le haché final en prenant les 224 bits à gauche.

1) Padding : Complément de M. Soit M un message de longueur ℓ bits. On ajoute le bit "1" à la fin du message M, puis k zéros, où $k \in \mathbb{N}^*$ est tel que : $\ell + 1 + k \equiv 448 \bmod 512$. L'ajout de 1 se fait même si $\ell \equiv 448 \bmod 51$.

Puis on ajoute ℓ écrit en binaire sur 64 bits.

2) Division du paddé : découper le message complété en blocs de 512 bits Le message complété est découpé en n blocs de 512 bits, notés $M_1, M_2, \cdots, M_n$. Chaque bloc M_i est ensuite découpé en 16 mots de 32 bits, notés $M_i = (M_i^{(0)}, M_i^{(1)}, \cdots, M_i^{15})$, où les $M_i^{(k)}$ sont des mots de 32 bits.

Initialisation des variables

Les cinq variables suivantes sont affectées de valeurs initiales : (les 4 premières sont les mêmes que dans MD4 et MD5)

- $A = H_0^{(0)} =$ 0x67452301
- $B = H_0^{(1)} =$ 0xefcdab89
- $C = H_0^{(2)} =$ 0x98badcfe
- $D = H_0^{(3)} =$ 0x10325476
- $E = H_0^{(4)} =$ 0xc3d2e1f0

et $H_0 = H_0^{(0)} H_0^{(1)} H_0^{(2)} H_0^{(3)} H_0^{(4)}$ de longueur 160 bits.

H_i consiste en 5 mots de 32-bit $H_i^{(0)}, H_i^{(1)}, H_i^{(2)}, H_i^{(3)}, H_i^{(4)}$

H_n est le haché.

Les M_i sont traités dans l'ordre et chaque bloc M_i subit un traitement de 80 tours, comme montré par la Figure 7.10.

On construit 80 mots de 32 bits chacun W_j si $0 \leq j \leq 79$, pour chacun des 80 tours.

$$W_j = \begin{cases} M_i^{(j)}, & 0 \leq j \leq 15 \\ (W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16}) \hookleftarrow 1, & 16 \leq t \leq 79 \end{cases} \tag{7.3}$$

Les quatre groupes de tours ont même structure mais utilisent différentes fonctions f_t et constantes K_i où $1 \leq t \leq 4$. Chaque groupe est composé de 20 tours.

Notations :

$\wedge$ = opération binaire AND ,
$\vee$ = opération binaire OR,
$\oplus$ = opération binaire XOR,
$\bar{X}$ = complément binaire de X,
$\boxplus$= addition modulo 2^{32},
$x \hookleftarrow s$ = décalage circulaire à gauche de s bits où $0 \equiv s \leq 31$.

Fonctions utilisées lors du calcul des valeurs de hachage. SHA-1 utilise une succession de fonctions logiques $f_0, f_1, \cdots, f_{79}$. Chaque fonction f_t, où $0 \leq t \leq 79$, opère sur trois mots de 32 bits, x, y, z et génère un mot de 32 bits en sortie. La fonction f_t est définie comme suit :

$$f_t(X,Y,Z) = \begin{cases} Ch(X,Y,Z) = (X \wedge Y) \oplus (\bar{X} \wedge Z), & \text{si } 0 \leq t \leq 19 \\ Parity(X,Y,Z) = X \oplus Y \oplus Z, & \text{si } 20 \leq t \leq 39 \\ Maj(X,Y,Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z), & \text{si } 40 \leq t \leq 59 \\ Parity(X,Y,Z) = X \oplus Y \oplus Z, & \text{si } 60 \leq t \leq 79 \end{cases} \tag{7.4}$$

x	y	z	Ch	$Parity$	Maj
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	1	0	1
1	1	1	1	1	1

TABLE 7.3 – Table de vérité des fonctions Ch,$Parity$ et Maj

Table de vérité des fonctions utilisées dans SHA-1 SHA-1 utilise 80 constantes notées $K_0, K_1, ..., K_{79}$ et définies par :

$$K_t = \begin{cases} [2^{30}\sqrt{2}] = \text{0x5a827999}, & \text{si } 0 \leq t \leq 19 \\ [2^{30}\sqrt{3}] = \text{0x6ed9eba1}, & \text{si } 20 \leq t \leq 39 \\ [2^{30}\sqrt{5}] = \text{0x8f1bbcdc}, & \text{si } 40 \leq t \leq 59 \\ [2^{30}\sqrt{10}] = \text{0xca62c1d6}, & \text{si } 60 \leq t \leq 79 \end{cases} \tag{7.5}$$

L'opération dans le tour j du groupe t est donné par :

$$A, B, C, D, E = (E + f_t(B, C, D) + (A) \hookleftarrow 5 + W_j + K_t), A, (B) \hookleftarrow 30, C, D \tag{7.6}$$

Voir Figure 7.11

SHA1 est formé de 4 groupes de tours :

Groupe 1 formé des 20 premiers tours où on utilise la fonction f_1, la constante K_1 et les mots $W_0, \cdots W_{19}$.

Groupe 2 formé des 20 tours qui suivent où on utilise la fonction f_2, la constante K_2 et les mots $W_{20}, \cdots W_{39}$.

Groupe 3 formé des 20 tours qui suivent où on utilise la fonction f_3, la constante K_3 et les mots $W_{40}, \cdots W_{59}$.

Groupe 4 formé des 20 derniers tours où on utilise la fonction f_4, la constante K_4 et les mots $W_{60}, \cdots W_{79}$.

Algorithme SHA-1 .

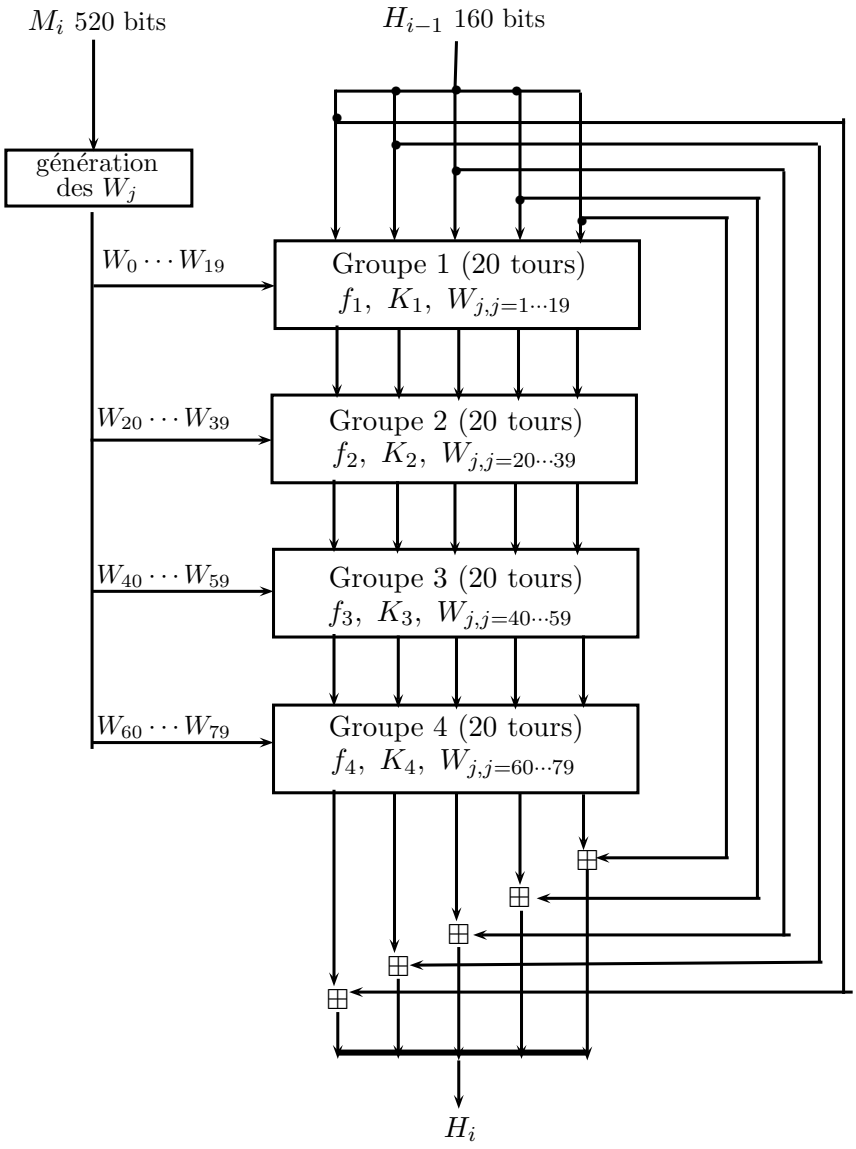


FIGURE 7.10 – Fonction de hachage SHA1

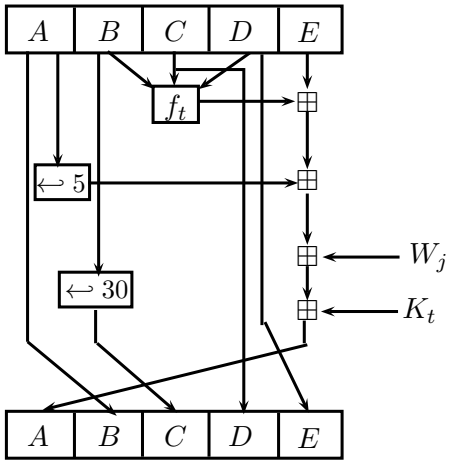


FIGURE 7.11 – Tour j du groupe t dans SHA1

$message\ m = m_0 m_1 \cdots m_{s-1}$
$Construct\ M = M[0]M[1] \cdots M[N-1]$
$A \longleftarrow 0x67452301$
$B \longleftarrow 0xefcdab89$
$C \longleftarrow 0x98badcfe$
$D \longleftarrow 0x10325476$
$E \longleftarrow 0xc3d2e1f0$
$for\ i = 0\ to\ N\ do$
    $PreparerW$
    $A \longleftarrow A$
    $B \longleftarrow B$
    $C \longleftarrow C$
    $D \longleftarrow D$
    $E \longleftarrow E$
    $for\ t = 0\ to\ 79\ do$
        $T \longleftarrow (A \hookleftarrow 5) + f_t(B,C,D) + E + K_t + W_t$
        $E \longleftarrow D$
        $D \longleftarrow C$
        $C \longleftarrow B \hookleftarrow 30$
        $B \longleftarrow A$
        $A \longleftarrow T$
$A \longleftarrow A + A$
$B \longleftarrow B + B$
$C \longleftarrow C + C$
$D \longleftarrow D + D$
$E \longleftarrow E + E$
$(h(M) = ABCDE)$

Calcul de l'empreinte

On traite successivement les N blocs de M comme il suit :

Pour $i = 1, \cdots, N$

2) On initialise a, b, c, d et e avec les valeurs de hachage du tour précédent

— $a = H_0^{(i-1)}$
— $b = H_1^{(i-1)}$
— $c = H_2^{(i-1)}$
— $d = H_3^{(i-1)}$
— $e = H_4^{(i-1)}$

3) Pour $t = 0, \cdots, 79$

— $T = ROTL^5(a) + f_t(b, c, d) + e + K_t + W_t$
— $e = d$
— $d = c$
— $c = ROTL^{30}(b)$
— $b = a$
— $a = T$

4) Calcul des valeurs de hachage intermédiaires

— $H_0(i) = a + H_0^{(i-1)}$
— $H_1(i) = b + H_1^{(i-1)}$
— $H_2(i) = c + H_2^{(i-1)}$
— $H_3(i) = d + H_3^{(i-1)}$
— $H_4(i) = e + H_4^{(i-1)}$

Après répétition des quatre étapes ci-dessus pour les N blocs du message M, le condensé de 160 bits de M est obtenu par concaténation des valeurs

$$H_0^{(N)}, H_1^{(N)}, H_2^{(N)}, H_3^{(N)}, H_4^{(N)}$$

Méthode 1 Le résumé de message est calculé en utilisant le message bourré comme décrit à la section 4. Le calcul est décrit avec l'utilisation de deux mémoires tampon, chacune consistant en cinq mots de 32 bits, et une séquence de quatre vingt mots de 32 bits. Les mots de la première mémoire tampon de cinq mots sont étiquetés A, B, C, D, E. Les mots de la seconde sont étiquetés H_0, H_1, H_2, H_3, H_4. Les mots de la séquence de 80 mots sont étiquetés $W(0)$, $W(1)$,..., $W(79)$. Une mémoire tampon TEMP d ?un seul mot est aussi employée. Pour générer le résumé de message, les blocs $M(1)$, $M(2)$,..., $M(n)$ de 16 bits définis à la section 4 sont traités dans cet ordre. Le traitement de chaque M(i) implique 80 étapes. Avant de traiter un bloc, les H sont initialisés comme suit, en hexadécimal : H0 = 67452301 H1 = EFCDAB89 H2 = 98BADCFE H3 = 10325476 H4 = C3D2E1F0.

Ensuite $M(1), M(2), \cdots, M(n)$ sont traités. Pour traiter $M(i)$, on procède comme suit :

a. Diviser $M(i)$ en 16 mots $W(0), W(1), \cdots, W(15)$, où $W(0)$ est le mot le plus à gauche.

b. Pour 16 ? t ? 79 soit

$$W(t) = S^1(W(t-3)OUXW(t-8)OUXW(t-14)OUXW(t-16))$$

c. Soit A = H0, B = H1, C = H2, D = H3, E = H4.

d. Pour 0 ? t ? 79 faire `TEMP = S^5(A) + f(t;B,C,D) + E + W(t) + K(t)` ; E = D ; D = C ; `C = S^30(B)` ; B = A ; A = TEMP ; e.

Soit H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E. Après le traitement de M(n), le résumé de message est la chaîne de 160 bits représentée par les 5 mots H0 H1 H2 H3 H4

7.7 SHA-1 vs MD5

- l'attaque par force brute est plus difficile (160 contre 128 bits pour MD5)
- non vulnérable à toutes les attaques connues (comparées à MD4/5)
- un peu plus lent que MD5 (80 contre 64 étapes)

Fonction	Empreinte	Complexité requise	Résistance aux collisions	Compléxité de l'attaque
MD5	128 bits	$\mathcal{O}(2^{64})$	Cassé [Sasaki al.05]	$\mathcal{O}(2^{30})$
SHA-1	160 bits	$\mathcal{O}(2^{80})$	Cassé (Crypto05 - Wang al.])	$\mathcal{O}(2^{63})$
HAVAL	256 bits	$\mathcal{O}(2^{128})$	Cassé (Asiacrypt 04)	$\mathcal{O}(2^{10})$
SHA-256	256 bits	$\mathcal{O}(2^{128})$	Sûr	
Whirlpool	512 bits	$\mathcal{O}(2^{256})$	Sûr	

TABLE 7.4 – Comparaison de fonctions de hachage

fonction	taille du Bloc en bits	taille du mot en bits	empreinte en bits	nombre de Tours	année
MD4	512	32	128	48	1990
MD5	512	32	128	64	1992
SHA-0	512	32	160	80	1993
SHA-1	512	32	160	80	1995
SHA-224	512	32	224	64	2004
SHA-256	512	32	256	64	2002
SHA-384	1024	64	384	80	2002
SHA-512	1024	64	512	80	2002
Whirlpool	512	-	512	10	2003

TABLE 7.5 – Fonctions de hachage standards

7.8 conclusion

SHA-3 est un appel à candidature du NIST

61 candidats soumis en octobre 2008. 51 propositions acceptes (les autres étant incomplets).

- pour l'instant 10 candidats déjà cassés, et une dizaine d'autres 'blessés'.
- il reste une trentaine de candidats potentiellement finalistes.
- 3 ou 4 candidats 'stars' (équipe renommée, algorithme médiatisé, etc.)
- 5 retenus.

Fonctions de hachage basées sur des chiffrements par blocs

7.9 Exercices

Exercice 1. Soit $f : \mathbb{F}_2^{2m} \longrightarrow \mathbb{F}_2^m$ une fonction de hachage et h une deuxième fonction de hachage définie par :

$$\begin{array}{rccc} h: & \mathbb{F}_2^{4m} & \longrightarrow & \mathbb{F}_2^m \\ & x_1||x_2 & \longrightarrow & f(f(x_1)||f(x_2)) \end{array}$$

où $||$ désigne l'opération de concaténation. Montrez que si f est résistante aux collisions, alors h est aussi résistante aux collisions.

Exercice 2. Soit une fonction de hachage $h : \{0,1\}^* \longrightarrow \{0,1\}^m$.
Montrer que la recherche exhaustive de collisions a plus d'une chance sur deux d'aboutir après seulement $\mathcal{O}(2^{\frac{m}{2}})$.

Exercice 3. Les systèmes d'authentification usuels vérifient les mots de passe à l'aide de leur haché stocké dans des fichiers protégés.

1. Quelle est l'utilité de stocker les hachés des mots de passe plutôt que les mots de passe eux-mêmes ?
2. Pourquoi doit-on protéger l'accès aux hachés des mots de passe ?
3. Sous quelle condition cette précaution ne serait-elle pas nécessaire ?

Exercice 4. Indiquer pour chacune des fonctions suivantes (i) si elles sont à sens unique ; (ii) si elles sont résistantes aux collisions.
- $h_1(x) = x^3 mod p$ pour p premier de 1024 bits ;
- $h_2(x) = x^3 mod n$ pour $n = pq$, avec p et q deux nombres premiers de 512 bits ;
- $h_3(x) = 3^x mod p$ pour p premier de 1024 bits.

Exercice 5. Soit $p = 1 + 2q$ un grand nombre premier tel que q soit aussi premier. Soit α et β deux éléments primitifs de $\mathbb{Z}_p^*$. La valeur de $\log_\alpha \beta$ n'est pas publique et l'on suppose qu'elle est calculatoirement difficile à obtenir.

a) Montrer que la fonction de hachage

$$\begin{array}{rcl} h: & \mathbb{Z}_q \times \mathbb{Z}_q \longrightarrow \mathbb{Z}_p^* \\ & (x_1, x_2) \longrightarrow \alpha^{x_1} \beta^{x_2} \end{array}$$

résiste aux collisions si le calcul de $\log_\alpha \beta$ est difficile.

b) Que pensez vous de cette fonction de hachage ?

Bibliographie

[1] Menezes A. J., Vanstone S. A. and Oorschot P. C. V., Handbook of Applied Cryptography, Computer Sciences Applied Mathematics Engineering, CRC Press, Inc., 1st edition, 1996, http ://www.cacr.math.uwaterloo.ca/hac/

[2] Schneier B., Cryptographie Appliqué Vuibert, Wiley and International Thomson Publishing, NY, 2nd edition, 1997. http ://www.schneier.com/book-applied.html

[3] Stinson D.R, Cryptography : Theory and Practice, Chapman & Hall/CRC Press, 2nd edition, 2002. http ://www.cacr.math.uwaterloo.ca/ dstinson/CTAP2/CTAP2.html

[4] Rolf Oppliger, Contemporary Cryptography, ARTECH HOUSE, INC. 2005.

[5] I. Damgård. A Design Principle for Hash Functions. Advances in Cryptology – CRYPTO'89, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[6] R. Merkle. One way hash functions and DES. Advances in Cryptology – CRYPTO'89, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[7] R. Rivest, The MD5 message-digest algorithm. IETF RFC 1321 (April 1992).

[8] http ://www.faqs.org/rfcs/rfc3174.html
http ://abcdrfc.free.fr/rfc-vf/pdf/rfc3174.pdf