Cryptographie

par **Jean-Pierre TUAL**
Directeur de l'unité cartes à microprocesseurs
Bull/CP8 Transac

*Le développement des transactions électroniques dans un nombre croissant de domaines économiques pose de manière de plus en plus critique le problème de la **sécurité** de telles transactions, notamment vis-à-vis des questions de confidentialité, d'authentification et de certification qui s'y rattachent. Pour tous ces aspects, la **cryptographie** est appelée à devenir une technique de plus en plus fondamentale pour la **protection des informations** possédées et/ou échangées par des individus ou des organisations.*

Les principes fondamentaux de la cryptographie moderne sont abordés dans les paragraphes 1, 2 et 3, avec notamment une présentation détaillée des deux types de cryptosystèmes les plus utilisés dans les applications pratiques, à savoir les ***systèmes à clefs symétriques,*** *dont le DES est l'exemple le plus caractéristique, et les* ***systèmes asymétriques*** *ou* ***à clefs publiques****, largement popularisés par le système RSA.*

Les paragraphes suivants abordent certaines des questions d'actualité (voire les plus controversées !) liées à la mise en œuvre pratique de la cryptographie symétrique ou asymétrique dans les applications, en particulier :

— le point en 1995 sur la ***sécurité*** *des systèmes DES et RSA (§ 3.5) ;*

— la description des deux systèmes standards de ***signature digitale*** *les plus utilisés aujourd'hui : DSS adopté par l'administration fédérale américaine et PKCS principalement utilisé par les industries travaillant sous Internet (§ 4) ;*

— une introduction aux systèmes de ***protocoles interactifs à connaissance nulle*** *(§ 5) ;*

— une présentation succincte de deux produits de sécurité maintenant largement utilisés dans les réseaux informatiques de type Internet : ***PGP*** *et* ***PEM*** *(§ 6).*

En raison de leur caractère particulièrement spécifique et spécialisé, les aspects juridiques liés à la mise en œuvre de la cryptographie ne sont pratiquement pas abordés ici. Il s'agit d'un domaine particulièrement délicat et qui est loin de trouver une réponse uniforme tant les législations en vigueur dans les différents pays peuvent être complexes voire contraignantes. On se contente de mentionner ici que, assimilée le plus souvent à une arme de guerre, la cryptographie est soumise à une multitude de restrictions, tant du point de vue utilisation qu'import/export de produits de sécurité pour des applications civiles. Avec la mondialisation de réseaux de type Internet ou Compuserve et le développement quasi exponentiel du commerce électronique et des communications mobiles, il est vraisemblable que cette question se trouvera rapidement au tout premier plan de débats particulièrement complexes et délicats, dans lesquels les enjeux socio-économiques, stratégiques ou sécuritaires devront être appréhendés à la mesure de leur caractère complémentaire ou contradictoire.

1. Généralités

1.1 Définitions et principes de la cryptographie moderne

Non formalisés jusqu'à la fin de la première moitié du XX^{e} siècle, les fondements de la cryptographie (du grec κρνπταω, cacher et γραφειν écrire), **science de la protection de l'information**, ont reçu une modélisation mathématique précise, en 1947, à la suite des travaux de Shannon [1] sur la théorie de l'information. Avant de décrire ce formalisme, donnons une idée précise du problème traité par la cryptographie et quelques définitions associées.

Lorsque deux individus souhaitent partager de l'information, ils sont amenés à s'**échanger**, le plus souvent par un **canal de transmission** (téléphone, télex, réseau d'ordinateurs...), des **messages** de longueur quelconque mais finie. Si l'émetteur de l'information souhaite que celle-ci ne soit accessible en clair qu'à son destinataire, alors il utilisera un procédé cryptographique si, par définition, il est capable de mettre en œuvre une transformation mathématique, agissant sur le message émis, rendant ce dernier inintelligible pour toute tierce partie illégalement branchée sur le canal de transmission. Cela suppose évidemment qu'à l'arrivée le destinataire du message dispose d'une autre transformation mathématique, éventuellement différente de la première, lui permettant de reconstituer le texte clair à partir du texte brouillé (ou chiffré) (figure 1).

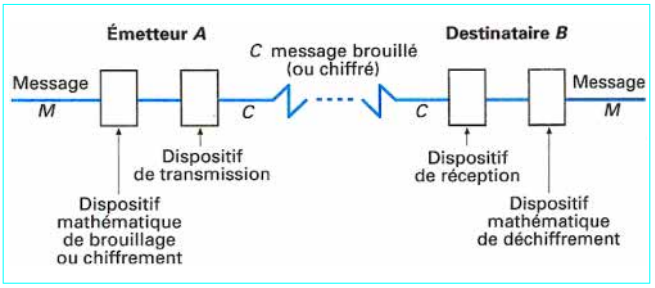


Figure 1 – Schéma général de la communication chiffrée entre un émetteur et un récepteur

Dans toute la suite de cet article, on utilise les définitions suivantes.

- On appelle **message** une suite de n symboles, pris dans un alphabet Z de q symboles. Dans la pratique, le plus souvent, Z est soit l'ensemble {0,1}, soit l'ensemble $Z_m = \{0, 1, ..., m - 1\}$ des entiers modulo m.

- On appelle ***N*-gramme**, un message M de longueur N, soit une suite $M = (Z_1, Z_2, ..., Z_N)$ où chaque Z_i appartient à Z.

L'ensemble des bijections de Z_m dans Z_m, ou ensemble des permutations d'ordre m, est noté S_m. Le cardinal de cet ensemble vaut exactement $m!$. Muni de la composition des applications, notée $\circ$, S_m est un groupe fini, c'est-à-dire un ensemble muni d'une loi vérifiant les propriétés :

— **de composition interne** : $\sigma \in S_m$, $\tau \in S_m \Rightarrow \sigma \circ \tau \in S_m$

— **d'associativité** : $\sigma \circ (\tau \circ \rho) = (\sigma \circ \tau) \circ \rho$

— **d'élément neutre** : l'élément $i \in S_m$, défini par $i(q) = q$ $\forall\, q \in Z_m$ vérifie

$$i \circ \sigma = \sigma \circ i = \sigma \quad \forall\, \sigma \in S_m$$

— **d'inversibilité** : $\forall\, \sigma \in S_m$, $\exists\, \tau \in S_m$ tel que

$$\sigma \circ \tau = \tau \circ \sigma = i$$

La notion de permutation est extrêmement importante et va permettre de préciser la typologie des systèmes cryptographiques.

Il faut noter également que l'on peut toujours supposer que l'alphabet de référence est l'un des ensembles Z_m, puisque, par isomorphisme, on peut toujours se ramener à ce cas ; ainsi, l'alphabet usuel $\mathcal{A} = \{A, B, ..., Z\}$ correspond à Z_{26}, l'alphabet ASCII à Z_{128}, l'alphabet ASCII étendu à Z_{256}, etc.

■ De manière précise, nous pouvons définir, avec Shannon, un **cryptosystème** comme étant un ensemble :

$$\mathcal{U} = (\mathcal{M}, \mathcal{C}, \mathcal{K}\{E_k\}_{k \in \mathcal{K}}, \{D_k\}_{k \in \mathcal{K}})$$

où $\mathcal{M}$ représente l'ensemble des messages clairs,

$\mathcal{C}$ représente l'ensemble des messages chiffrés,

$\mathcal{K}$ représente un espace de paramètres appelés clefs cryptographiques,

E_k représente, pour tout $k \in \mathcal{K}$, une transformation de chiffrement $\mathcal{M} \xrightarrow{E_k} \mathcal{C}$,

D_k représente, pour tout $k \in \mathcal{K}$, une transformation de déchiffrement $\mathcal{C} \xrightarrow{D_k} \mathcal{M}$ telle que, pour tout message M, on ait :

$$D_k \circ E_k(M) = M$$

Dans la pratique, si l'on désigne par Z_m l'espace des N-grammes construits sur Z_m, on a, par exemple, $\mathcal{M} = Z_{m,N}$, $\mathcal{C} = Z_{q,R}$ pour des entiers m, q, N, R bien choisis. Tous les systèmes de cryptographie moderne peuvent être décrits par ce formalisme à la fois simple, élégant et puissant.

La notion de **clef paramétrée** ou **clef cryptographique** est fondamentale : elle sous-tend tous les mécanismes de paramétrisation des algorithmes de cryptage ainsi que toutes les techniques de protocoles d'échanges cryptographiques.

Dans la pratique, on dispose en effet d'algorithmes génériques, dépendant de paramètres de base (ou clefs).

À chaque choix de paramètres correspond un algorithme de chiffrement et l'algorithme dual de déchiffrement. Dans un réseau d'ordinateurs, avec un grand nombre d'utilisateurs (quelques milliers, par exemple), il est exclu d'utiliser, pour des raisons techniques et économiques évidentes, plus d'un ou deux algorithmes génériques de chiffrement/déchiffrement. Ceux-ci sont en général soigneusement implantés dans quelques millimètres carrés de silicium, dans l'une de ces puces dont les coûts de développement ou d'achat (à cause de leur généralisation) sont loin d'être négligeables. La communication entre utilisateurs, en mode chiffré, est rendue possible par la distribution et la gestion sur le réseau des clefs cryptographiques, paramètres essentiellement volatils et peu encombrants, sur lesquels repose, en général, 90 % de la sécurité des mécanismes d'encryption.

1.2 Cryptanalyse et théorie de l'information

Avant de donner quelques exemples, complétons encore notre ensemble de définitions.

■ On appelle **cryptanalyse** l'opération consistant, sur la base de messages reçus $C = T_k(M)$ où T_k est une transformation mathématique, connue ou inconnue, dépendant du paramètre k, lui, toujours inconnu, à reconstituer les messages originaux M.

■ On appelle **cryptanalyste** l'individu s'adonnant de manière scientifique (et/ou perverse) à l'exercice de la cryptanalyse. En pratique, la cryptanalyse revient toujours à la recherche de la clef k de paramétrage.

La cryptanalyse est une notion complémentaire mais indissociable de la cryptographie : construire un système de cryptographie inviolable passe toujours par la vérification qu'il résiste aux assauts de la cryptanalyse. Ainsi un bon cryptographe est-il, par nécessité, un bon cryptanalyste (et réciproquement !).

De manière pratique, on rencontre quatre situations de cryptanalyse :

— en mode « *attaque sur texte chiffré seulement* », le cryptanalyste n'a, à sa disposition, pour retrouver la clef de chiffrement, qu'un nombre fini de textes chiffrés au moyen de cette clef (appelés aussi cryptogrammes) ;

— en mode « *attaque sur texte clair connu* », le cryptanalyste s'efforce de reconstituer la clef à partir de couples (clairs, chiffrés) correspondants : cette situation se rencontre par exemple lorsque, branché illégalement sur un réseau, le cryptanalyste sait que tous les premiers messages émis par un terminal commencent par une bannière standard (par exemple : LOGIN) ;

— en mode « *attaque sur texte clair choisi* », le cryptanalyste est capable de produire le cryptogramme correspondant au texte clair de son choix ; cela est le cas le plus favorable et on le visualise par un système de base de données où chaque utilisateur est capable d'ajouter des articles à la base et d'observer leurs cryptogrammes correspondants ;

— en mode « *attaque sur texte chiffré choisi* », le cryptanalyste est capable de produire le texte clair (éventuellement dépourvu de toute sémantique) correspondant au texte chiffré de son choix. Cette situation, quoique étrange à première vue, illustre parfaitement le cas des systèmes à clefs publiques, qui seront décrits au paragraphe 3.4.

Un cryptosystème est dit **inconditionnellement sûr** si, pour toute portion de texte chiffré intercepté, on ne peut reconstituer de manière unique le texte clair correspondant.

Un cryptosystème est dit **calculatoirement sûr** si, étant donné une portion de texte chiffré, on ne peut reconstituer le texte clair correspondant par analyse exhaustive, à partir des moyens mis à la disposition du cryptanalyste. Il s'agit ici essentiellement d'une définition pratique, qui se déplace dans le temps... comme les limites technologiques des ordinateurs et dispositifs de transmission.

La **distance d'unicité** d'un cryptosystème est définie comme la longueur de texte chiffré théoriquement nécessaire pour reconstituer un seul message en clair correspondant. Cette distance d'unicité n'apprend rien de particulier sur la longueur moyenne des cryptogrammes à intercepter pour autoriser une cryptanalyse facile : elle ne constitue qu'une indication subjective (mais utile) pour mesurer la sécurité *a priori* d'un cryptosystème. On peut montrer, dans le cas de cryptosystèmes classiques, que cette distance d'unicité vaut :

$$d = H(k)/D$$

où $H(k)$ représente l'entropie de la clef et D la redondance moyenne des messages en clair.

Exemple : dans le cas de messages en anglais, encryptés au moyen d'une clef (ici égale à l'entropie) de 56 bits, la distance d'unicité vaut, avec une bonne approximation $d = 56/3{,}2 = 17{,}5$ caractères, puisque la redondance moyenne de l'anglais vaut environ 3,2.

2. Évolution de la cryptographie

2.1 Typologie des cryptosystèmes

Nous ne présentons ici que quelques systèmes de cryptographie parmi les plus simples. Bien entendu, il serait illusoire de prétendre à toute exhaustivité, et nous renvoyons le lecteur intéressé aux références [2] [3] [4].

Les cryptosystèmes que nous allons brièvement décrire appartiennent à l'une des catégories suivantes.

Cryptosystèmes par permutations

Dans ce cas, les transformations cryptographiques considérées opèrent sur les N-grammes.

Pour tout $T_k \in \mathcal{K}, T_k$ (noté plus simplement T) envoie $Z_{m,N}$ sur $Z_{m,N}$.

Au message :

$$M = (X_0, X_1, ..., X_N, X_{N+1}, ..., X_{2N-1} ...)$$

correspond le cryptogramme :

$$C = E_T(M) = (X_{T(0)}, X_{T(1)}, ..., X_{T(N-1)}, X_{N+T(0)}, ..., X_{N+T(N-1)} ...)$$

Cryptosystèmes par substitutions

Dans ce cas, si $M = (X_0, X_1, ..., X_N ...)$ est le message en clair à encrypter, le cryptogramme associé $C = (Y_0, Y_1, ..., Y_N ...)$ est du type :

$$Y_i = \pi_i(X_i) \text{ où } \pi_i \in S_m$$

pour tout i (et bien sûr, avec la convention habituelle, $X_i \in Z_m$).

Si π_i dépend de i, on parle de **cryptosystème à substitution polyalphabétique** ; dans le cas inverse, on parle de **cryptosystème à substitution monoalphabétique**.

2.2 Système de César

C'est le plus naïf de tous les cryptosystèmes et, aussi, historiquement le plus ancien. César s'en est servi pour envoyer des messages au Sénat romain pendant ses campagnes militaires. Son modèle, au sens de Shannon, est le suivant :

— alphabet : Z_{26}

— espace des messages : $\mathcal{M} = (Z_{26})^n$

— espace des cryptogrammes : $\mathcal{C} = (Z_{26})^n$

— espaces des clefs : $\mathcal{K} = Z_{26}$

Un message $M = (X_1, X_2, ..., X_n)$ est encrypté, étant donné la clef de $k \in \mathcal{K}$, en le cryptogramme

$$C = [T_k(X_1), T_k(X_2), ..., T_k(X_n)]$$

avec $T_k(X_i) = X_i + k$ (modulo 26) pour tout i.

Exemple : si $k = 3$, le message clair
CECI EST UN CRYPTOGRAMME donne le texte chiffré FHFL HVW XQ FUBSWRJUDPPH.

2.3 Système de Vigenère

Il doit son origine à B. de Vigenère, qui le décrivit dans son *Traité des chiffres* [5] paru en 1587. Ce système utilise la notion de substitution polyalphabétique décrite précédemment ; il fut utilisé avec succès pendant près d'un siècle et garda même, pendant longtemps, une réputation d'inviolabilité... bien usurpée. Le modèle de Shannon du système de Vigenère est le suivant :

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

Figure 2 – Table du chiffre de Vigenère

— alphabet : Z_m

— espace des messages : $\mathcal{M} = (Z_m)^n$

— espace des cryptogrammes : $\mathcal{C} = (Z_m)^n$

— espaces des clefs : $\mathcal{K} = (Z_m)^r$

Un message $M = (X_1, X_2, ..., X_n)$ est encrypté, étant donné la clef $k = (k_1, ..., k_r) \in \mathcal{K}$, en le cryptogramme

$$C = (Y_1, ..., Y_n)$$

avec $Y_i = X_i + k_i$ (modulo r) (modulo m).

Une façon pratique de réaliser l'encryption est d'ulitiliser une grille, telle celle décrite en figure 2.

Étant donné le texte clair, on écrit immédiatement au-dessous la clef, reproduite autant de fois que nécessaire par périodicité. L'encryption de la i^e lettre se fait en regardant l'intersection de la colonne de la table correspondant à la lettre du texte clair et de la ligne de la table correspondant à la lettre de la clef.

Exemple : le texte clair CRYPTOGRAMME est encrypté comme suit avec la clef MINET :

— clair : C R Y P T O G R A M M E

— clef : M I N E T M I N E T M I

— cryptogramme : O Z L T M A O E E F Y M

2.4 Système Playfair

Ce cryptosystème, mis au point en 1854 par les savants anglais L. Playfair et C. Wheaststone, illustre la notion de chiffrement par polygrammes ; son modèle de Shannon, pour les messages à 2 n caractères, est le suivant :

— alphabet : Z_m (avec $m = a^2$ où a est un entier)

— espace des messages : $\mathcal{M} = (Z_m \times Z_m)^n$

— espace des cryptogrammes : $\mathcal{C} = (Z_m \times Z_m)^n$

— espace des clefs : $\mathcal{K} = B_{ij}(Z_m, \mathcal{M}_a(Z_m))$

où B_{ij} représente l'ensemble des bijections de Z_m dans $\mathcal{M}_a(Z_m)$ (matrices carrées d'ordre a à coefficients dans Z_m).

Ainsi une clef $k \in \mathcal{K}$ est une matrice $a \cdot a$ à cœfficients dans Z_m, telle que tous les éléments de Z_m apparaissent une et une seule fois dans k.

L'encryption du message $M = (X_1, X_2, ..., X_n)$ se fait de la manière suivante.

■ On partage M en bigrammes, de la manière suivante :

$$M \to \tilde{M} = (X_1, X_2)\ (X_3, X_4) \ldots\ (X_{n-1}, X_n)$$

— au besoin, on ajoute une lettre neutre (X ou Z) en fin de message, de sorte que le nombre de caractères du message soit pair ;
— on interdit les configurations de bigrammes où $X_{i-1} = X_i$; si tel est le cas, on intercale un X ou un Z après X_i.

■ On encrypte alors, bigramme par bigramme, de la manière suivante :

soit $U = (X_p, X_{p+1})$ le bigramme à encrypter. Par définition de k, il existe (i, j) et (k, l) tels que $X_p = k_{ij}$ et $X_{p+1} = k_{kl}$ (k_{ij} est l'élément situé à l'intersection de la i^e ligne et de la j^e colonne de k).

On pose alors, en identifiant k_{ij} avec (i, j) :

$$E_k(X_p, X_{p+1}) = E_k((i, j), (k, l))$$

$$E_k((i,j),(k,l)) = \begin{cases} ((i,l),(k,j)) \text{ si } i \neq k \text{ et } j \neq l \\ ((i,(j+1) \bmod m),(k,(l+1) \bmod m)) \text{ si } i = k \\ (((i+1) \bmod m, j),((k+1) \bmod m, l)) \text{ si } j = l \end{cases}$$

Les opérations de décryption s'en déduisent immédiatement. Le déchiffrement du bigramme (i, j) (k, l) s'effectue par :

$$D_k((i,j),(k,l)) = \begin{cases} ((i,(j-1) \bmod m),(k,(l-1) \bmod m)) \text{ si } i = k \\ (((i-1) \bmod m, j),((k-1) \bmod m, l)) \text{ si } j = l \\ ((i,l),(k,j)) \text{ si } i \neq k \quad j \neq l \end{cases}$$

Exemple :

avec la clef $k =$

P	A	T	R	I
C	B	D	E	F
G	H	J	K	L
M	N	O	Q	S
U	V	X	Y	Z

le texte « ELLE A UNE FERME ET DES CHAMPS », qui conduit à la suite de bigrammes

(EL) (LE) (AU) (NE) (FE) (RM) (EX) (ET) (DE) (SC) (HA) (MP) (SX)

s'encrypte en :

(FK) (KF) (PV) (QB) (CF) (PQ) (DY) (DR) (EF) (MF) (AU) (UC) (OZ)

2.5 Système ADFG(V)X

Ce système de chiffrement, inventé par l'Allemand F. Nebel durant la Seconde Guerre mondiale, présente la particularité de mixer de manière très astucieuse transpositions et substitutions.

Son modèle de base est le suivant :
— alphabet : Z_5 (ADFGX) ou Z_6 (ADFGVX)
— espace des messages : $\mathcal{M} = (Z_m)^n$ avec $m = a^2$
— espace des cryptogrammes : $\mathcal{C} = (Z_a)^{2n}$
— espace des clefs : $\mathcal{K} = B_{ij}(Z_m, \mathcal{M}_a(Z_m)) \times S_N$

Ici, N est un nombre donné, *a priori* inférieur à la longueur des messages.

La faible longueur de l'alphabet des cryptogrammes tient au mode de transmission codée utilisé (en l'occurrence l'alphabet Morse, où les lettres utilisées (A, D, F, G, V, X) sont très facilement reconnaissables : A. _ ; D _ .. ; F .. _ . ; G _ _ . ; V ... _ ; X _ .. _ .

Exemple : le chiffrement du texte CRYPTOGRAMME avec la clef suivante :

$$k = Q \times \pi \text{ où } Q = \begin{bmatrix} C & R & Y & P & T \\ O & G & H & A & M \\ B & D & E & F & I \\ K & L & N & Q & S \\ U & V & W & X & Z \end{bmatrix}$$

$$\pi : \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 1 & 5 & 6 & 4 \end{pmatrix}$$

On suppose $N = 6$, par définition.

Le chiffrement procède en trois étapes.

■ **ADFG(V)X 1** : on associe à tout élément de Q des « coordonnées », à valeur dans {A, D, F, G, X} de la manière suivante :

	A	D	F	G	X
A	C	R	Y	P	T
D	O	G	H	A	M
F	B	D	E	F	I
G	K	L	N	Q	S
X	U	V	W	X	Z

Ainsi R, par exemple, correspond à (A, D), Y à (A, F), Z à (X, X), etc.

On encrypte alors le texte clair « CRYPTOGRAMME » lettre par lettre, soit :

(CRYPTOGRAMME) → (AAADAFAGAXDADDADDGDXDXFF)

■ **ADFG(V)X 2** : le texte précédent est partagé en 4 blocs de 6 lettres, rangés dans la matrice (4, 6) suivante :

$$Z = \begin{bmatrix} A & A & A & D & A & F \\ A & G & A & X & D & A \\ D & D & A & D & D & G \\ D & X & D & X & F & F \end{bmatrix}$$

■ **ADFG(V)X 3** : on permute les colonnes de Z par rapport à la permutation π ; on obtient :

$$Z \to \overline{Z} = \begin{bmatrix} A & A & A & F & D & A \\ A & A & G & A & X & D \\ A & D & D & G & D & D \\ D & D & X & F & X & F \end{bmatrix}$$

Le texte encrypté s'obtient en lisant en séquence les colonnes de $\overline{Z}$ soit :

C = (AAADAADDAGDXFAGFDXDXADDF)

La sécurité de ce cryptosystème a été étudiée par A. Konheim [4].

2.6 Machine à rotors

Donnons ici quelques principes de base.

Schématiquement, une machine à N rotors se compose de N dispositifs (ou rotors), physiquement implantés sous forme de disques isolants (par exemple en bakélite) à la périphérie desquels,

des deux côtés, se trouvent uniformément répartis 26 contacts électriques, chacun correspondant à une lettre de l'alphabet. Les contacts situés de part et d'autre du disque sont reliés 2 à 2 par un chemin électrique.

Dans un assemblage à N rotors, les contacts électriques de la face externe du i^e rotor sont reliés aux contacts électriques de la face interne du $(i+1)^e$ rotor, et ainsi de suite, suivant le schéma donné figure **3**.

Un courant électrique émis sur l'un des contacts externes du rotor N traverse suivant un chemin unique l'ensemble des N rotors, jusqu'à la face interne du rotor n° 1.

Chaque rotor, *via* ses chemins électriques internes, implémente une substitution monoalphabétique π des 26 lettres de l'alphabet.

Si l'on tourne le rotor de k positions dans le sens trigonométrique, alors le rotor implémente une substitution différente, définie par :

$$T_{\pi,k}(X) = \pi(X+k) - k$$

Il est facile de voir que $T_{\pi,k}$ est en fait la transformation :

$$T_{\pi,k} = C_{-k} \circ \pi \circ C_k$$

où C_k est la substitution de César (décalage de k places modulo 26)

L'idée de base des systèmes à rotors est que, à chaque encryption de lettre, chaque dispositif peut tourner indépendamment suivant des fonctions de déplacement bien définies.

Cela veut dire qu'à chaque rotor i est associée une fonction de déplacement $r_s(i)$ dépendant du i^e caractère à encrypter.

La clef d'un système à rotors consiste donc en :

— la donnée des substitutions monoalphabétiques $(\pi_0, ..., \pi_{N-1})$ implémentées par chaque rotor ;
— les déplacements initiaux $(k_0, ..., k_{N-1})$;
— les fonctions de déplacement $\{r_s(i),\ 0 \leqslant s < N\}$.

L'équation chiffrante d'une machine à N rotors associe donc au texte clair $(X_0, X_1, ..., X_n ...)$ le cryptogramme $(Y_0, Y_1, ..., Y_n ...)$ obtenu par composition des N rotors $C_{-r_s(i)} \circ \pi_s \circ C_{r_s(i)}$ avec $0 \leqslant s < N$ soit :

$$Y_i = \pi_{N-1}(r_{N-1})(i) + ... + \pi_1(r_1(i) + \pi_0(r_0(i) + X_i) - r_0(i)) - r(i) ...) - r_{N-1}(i)$$

Ces types de système de cryptage ne sont plus inviolables avec les systèmes de calcul utilisés de nos jours [6] [7] [4]. Pour mémoire, mentionnons que la commande *crypt* du système UNIX a été initialement basée sur un système (logiciel !) à rotors.

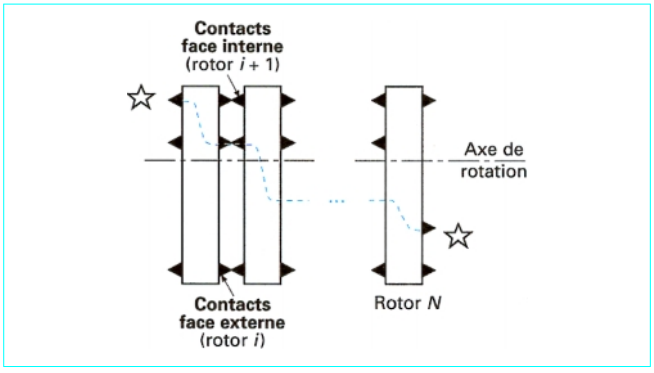


Figure 3 – Principe d'une machine à rotors

3. Cryptosystèmes actuels

3.1 Principaux types de cryptosystèmes

Les principaux types de cryptosystèmes utilisés aujourd'hui se répartissent en deux grandes catégories : les cryptosystèmes par flots et les crytosystèmes par blocs. Nous illustrons ci-après les différences fondamentales entre ces deux types de systèmes d'encryption.

■ Dans un **cryptosystème par flots**, l'encryption des messages se fait caractère par caractère, au moyen de substitutions de type César (§ 2.2) générées aléatoirement selon une loi de répartition équiprobable sur l'alphabet Z_m.

En d'autres termes, la lettre X_n du message à transmettre est encryptée en $Y_n = X_n + k_n$, où k_n est une variable aléatoire à valeurs dans l'alphabet de référence Z_m.

La loi de k_n est telle que toutes les valeurs $P(k_n = k)$ sont équiprobables $(0 \leqslant k < m)$ et que les $\{k_i\}$ $1 \leqslant i < n$ sont indépendantes, cela quelle que soit la longueur n du message.

Dans la pratique, les valeurs aléatoires des k_i sont générées par un processus déterministe, simulant une loi uniforme. Ce processus est un algorithme dépendant d'un vecteur d'initialisation Z. Le schéma d'encryption est donné sur la figure **4**.

L'intérêt de ce système, au moins du point de vue théorique, est qu'il présente un secret absolu, dès lors que la clef est effectivement parfaitement aléatoire et n'est jamais réutilisée (sinon, si deux messages M et M' sont encryptés avec la même clef k, et si $C = M \oplus k$ et $C' = M' \oplus k$ sont les cryptogrammes correspondants, on a $C \oplus C' = M \oplus k \oplus M' \oplus k = M \oplus M'$ puisque $k \oplus k = 0$).

Dans la pratique, il pose des problèmes délicats : canaux sûrs de distribution des clefs et surtout caractère aléatoire des générateurs de bits de clefs k_n utilisés. Notons de plus qu'une synchronisation parfaite doit exister, bit à bit, entre émetteur et récepteur, cela à toute émission de caractère, pour que le processus puisse fonctionner. En revanche, un des avantages du système est qu'il est insensible aux phénomènes de propagation d'erreurs : un bit erroné donne une erreur à la réception ou à l'émission, mais est sans incidence sur les bits suivants.

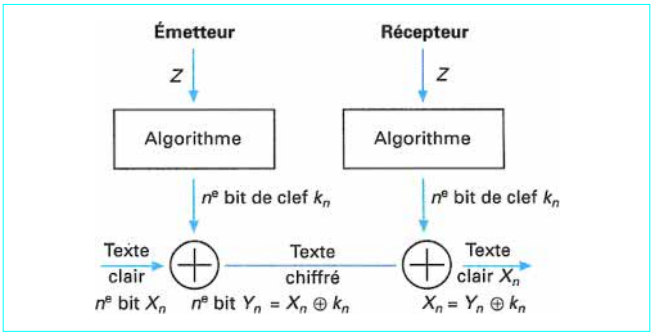


Figure 4 – Système de chiffrement par flots

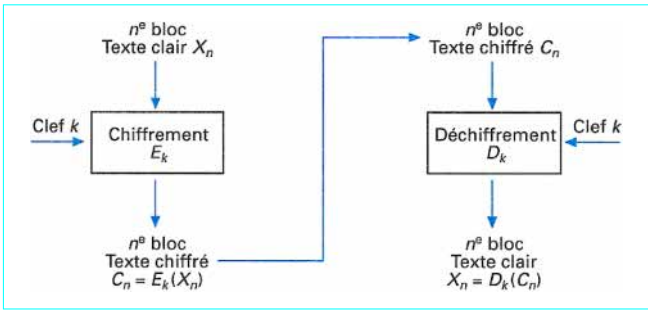


Figure 5 – Système de chiffrement par blocs

La deuxième classe de cryptosystèmes utilisée aujourd'hui est celle des **cryptosystèmes par blocs**. Dans ce mode d'encryption, le texte clair est fractionné en blocs de même longueur à l'aide d'une clef unique k. Émetteur et récepteur disposent de deux algorithmes d'encryption/décryption parfaitement réciproques, paramétrés par k (figure **5**).

Les caractéristiques de ces systèmes sont en général liées à leur très forte sensibilité à la dépendance intersymboles (en clair, si X_n et X'_n sont deux messages « voisins », il faut que les cryptogrammes associés Y_n et Y'_n soient non corrélables), ainsi qu'à leur mécanisme de propagation d'erreurs. Comme nous le verrons avec le DES (§ 3.3), toute erreur commise sur un bloc de texte clair ou chiffré peut perturber gravement le chiffrement/déchiffrement de ses voisins. Nous sommes maintenant en mesure de décrire quelques-uns des cryptosystèmes des deux catégories présentées dans ce paragraphe.

3.2 Systèmes d'encryption par flots

Dans ce type de cryptosystèmes, la difficulté pratique consiste à simuler, sur une période aussi longue que possible, un comportement quasi aléatoire d'une suite de bits (ou nombres de l'alphabet de référence Z_m) destinés à fournir la clef d'encryption.

La **génération de nombres aléatoires** est un problème très délicat du point de vue de la cryptographie ; tous les algorithmes déterministes connus présentent des faiblesses, du point de vue de l'inférabilité de la suite à partir d'observations localisées. Cela signifie que, étant donné une observation d'un nombre fini de tirages aléatoires $x_1, ..., x_k$, on est souvent en mesure d'en déduire les valeurs $..., x_{-n}, x_{-n+1}, ..., x_{-1}, x_0, ..., x_k, x_{k+1}, ..., x_m$ pour n et m aussi grands que l'on veut.

Dans la pratique, les algorithmes utilisés sont de l'un des types suivants :

— **algorithmes directs**, où le n^e nombre aléatoire x_n est obtenu par une relation directe, du type $x_n = f(x)$;

— **algorithmes itératifs**, où le n^e nombre aléatoire x_n est obtenu par une relation du type $x_n = f(x_{n-1}, x_{n-2}, ..., x_{n-k})$.

Seule la deuxième classe donne des résultats utilisables en contexte sécuritaire. Les congruences du type :

$$x_{n+1} = (a x_n + b) \text{ modulo } m \qquad x_0 \text{ fixé} \qquad \textbf{(1)}$$

où $1 \leqslant a < m$, $0 \leqslant b < m$ sont fréquemment utilisées.

Toute séquence de ce type est périodique, c'est-à-dire qu'il existe un entier q et un entier p, tels que l'on ait $x_{p+q} = x_q$, d'où l'on tire $x_{p+q+k} = x_{q+k}\ \forall\ k \leqslant p-1$.

En choisissant a tel que le plus grand commun diviseur PGCD $(a, m) = 1$, on peut montrer que la longueur du cycle de la suite pseudo-aléatoire définie par (1) est égale à :

$$w = \frac{m}{\text{PGCD}\ ((a-1)\ x_0 + b, m)}$$

On peut de même démontrer qu'il existe un entier t, appelé **période effective du générateur**, tel que :

- $x_q, ..., x_{q+t-1}$ sont tous distincts ;
- $\forall\ k \in \mathbb{N}\ \forall\ i \in \{0, ..., m-1\}$

on a : $x_{kt+i+q} = k x_t + x_{i+q}$ (modulo m) **(2)**

- $w = rt$ avec $a^t = 1$ (modulo m)
  $ar = 0$ (modulo m).

La période effective t du générateur caractérise la plus grande longueur de suite pseudo-aléatoire construite à partir de (1). Bien qu'étant tous distincts, les x_{kt+m} $(kt + m < p)$ se déduisent des $\{x_q, ..., x_{q+t-1}\}$ par la relation (2).

Exemples :

– Sur les calculettes Hewlett-Packard, la relation (1) est du type :

$$x_{n+1} = (9\,821\ x_n + 211\,327) \text{ modulo } 10^6$$

avec x_0 quelconque.

La période du générateur est 10^6, sa période effective 50 000.

Le niveau de sécurité offert par une encryption par flots à partir de ce générateur est tout à fait minimal.

– Sur le calculateur IBM 360/370, le générateur suivant a été longtemps employé :

$$x_{n+2} = (3 + 2^{16})\ x_n \text{ (modulo } 2^{31})$$

Sa période est 2^{29} pour tout x_0 impair, et sa période effective 2^{28}, ce qui constitue déjà un niveau de sécurité respectable.

– Il existe beaucoup d'autres générateurs de nombres aléatoires ; mentionnons-en un nettement plus robuste *cryptographiquement* que les précédents.

Soit $n = p\,q$ le produit de deux nombres premiers du type $4k + 3$.

Soit x_0 un résidu quadratique modulo n ; c'est-à-dire qu'il existe $y \in Z_n$, tel que $x_0^2 = y$ (modulo n).

Alors, si on pose :

$$x_{i+1} = x_i^2 \pmod{n}$$

$$b_{i+1} = \begin{cases} 1 & \text{si } x_{i+1} \text{ est impair} \\ 0 & \text{si } x_{i+1} \text{ est pair} \end{cases}$$

on définit ainsi une suite b_i pseudo-aléatoire, dont on peut démontrer qu'il n'existe aucun algorithme en temps polynomial capable de l'inférer et dont on peut calculer la période par :

$$p = \lambda\ (\lambda\ (n))$$

où λ est la **fonction de Carmichael** définie par :

$$\lambda\ (2^s) = \begin{cases} 2^{s-1} & \text{pour } s = 1,2 \\ 2^{s-2} & \text{pour } s > 2 \end{cases}$$

$$\lambda\ (2^E \cdot p_1^{\alpha_1} \ldots p_q^{\alpha_q}) = \text{PPCM}\left[\lambda\ (2^E), \left\{(p_i - 1) \cdot p_i^{\alpha_i - 1}\right\}\right]$$

où $2^E \cdot p_1^{\alpha_1} \ldots p_q^{\alpha_q}$ est la décomposition en facteurs premiers d'un nombre n quelconque de $\mathbb{N}$.

Cette fonction a une croissance très rapide vers l'infini.

Les mécanismes d'encryption par flots s'implémentent très bien, dans le cas où le générateur est du type $x_{n+1} = f(x_n, ..., x_{n-k})$, avec f linéaire, par des structures informatiques de type registre à décalage avec rebouclage (LFSR).

Un tel registre est défini par le contenu initial (0 à 1) de ses n cellules, notées $(t_n, t_{n-1}, ..., t_1)$. L'état à l'instant $p+1$ du registre, noté $(r_n^{p+1}, ... r_1^{p+1})$ se déduit de l'état à l'instant p par :

$$\begin{cases} r_k^{p+1} = r_{k+1}^p & k = 1, 2, ..., n-1 \\ r_n^{p+1} = \sum_{i=1}^{n} t_i r_i^p \bmod 2 \end{cases}$$

La méthode d'encryption est présentée sur la figure **6**.

On peut montrer que la période d'un tel registre divise $2^n - 1$ et en particulier lui est égale si le polynôme défini par $T(x) = t_n x^n + t_{n-1} x^{n-1} + ... + t_1 x + 1$ est primitif. Il est d'autre part possible de disposer en cascade plusieurs dispositifs de ce type pour augmenter la période du générateur [8].

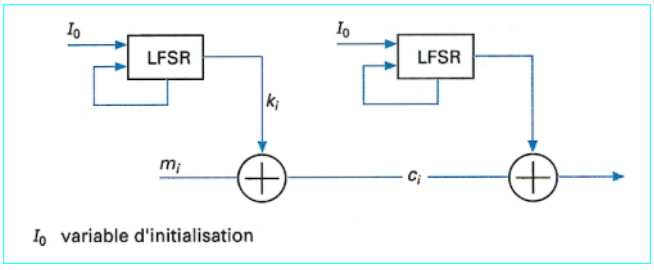


Figure 6 – Méthode d'encryption des registres à décalage avec rebouclage LFSR

3.3 Algorithme Data Encryption Standard (DES)

Publié en 1977 par le NBS (National Bureau of Standards), le DES est un algorithme de chiffrement de données recommandé pour les organisations à caractère fédéral, commercial ou privé.

Le DES tire son origine des travaux menés par le groupe cryptographique d'IBM dans le cadre du projet LUCIFER [9] [10].

Le DES a été l'objet de nombreuses implémentations, à la fois en matériel et en logiciel, depuis sa publication.

Après une décennie de succès, pendant laquelle les moyens et techniques de cryptanalyse mis en œuvre pour en étudier les caractéristiques n'ont pas permis d'en découvrir des faiblesses rédhibitoires, le DES a, depuis peu, révélé des sensibilités à des attaques nouvelles et puissantes, parfois réalisées sur un simple micro-ordinateur. Aussi l'ISO (International Organization for Standardization) a-t-il récemment refusé la normalisation du DES, ce qui n'empêche pas cet algorithme d'être, de loin, aujourd'hui encore comme le moyen d'encryption le plus sûr (et le plus largement utilisé) pour des données non militaires.

3.3.1 Principe

Le DES est un **algorithme de chiffrement symétrique** (émetteur et récepteur partagent la même clef), qui permet d'encrypter des mots de 64 bits à partir d'une clef K de 56 bits.

Le DES est le produit de 33 transformations :

$$\text{DES} = (IP)^{-1} \circ \pi_{T_{16}} \circ \theta \circ \pi_{T_{15}} \circ ... \circ \theta \circ \pi_{T_1} \circ IP$$

avec IP permutation initiale,
π_{T_k} transformation de $Z_{2,64} \rightarrow Z_{2,64}$,
θ involution d'échange.

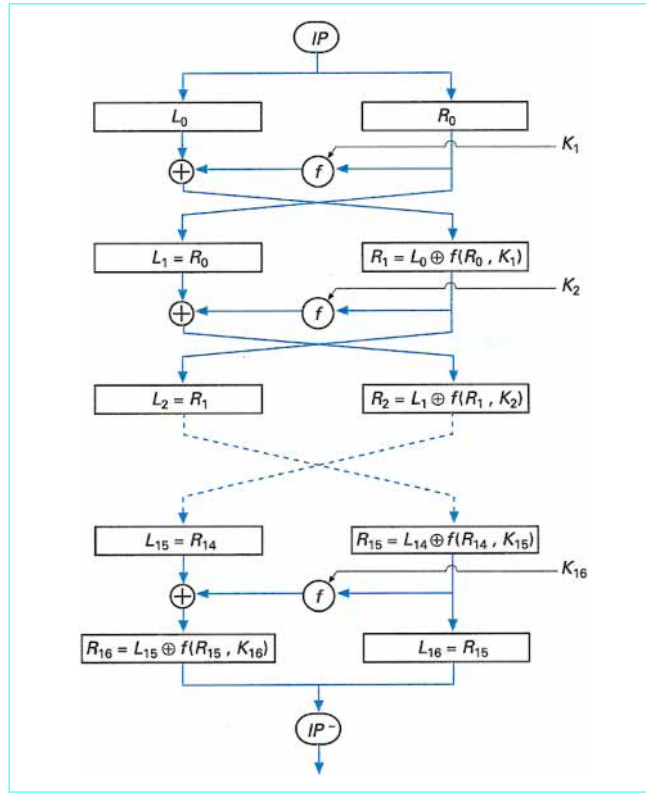


Figure 7 – Schéma général du DES

La structure globale de l'algorithme est décrite sur la figure **7**.

Le fonctionnement de l'algorithme est alors décrit par les étapes suivantes.

■ **Étape 1** : permutation des données : les 64 bits du texte clair $\tilde{x} = (x_0, ..., x_{63})$ sont soumis à la permutation initiale I/P pour produire le texte brouillé $x^0 = (x_{57}, x_{49}, ..., x_6)$.

■ **Étape 2** : à la i^e itération de l'algorithme, l'entrée (sur 64 bits) de l'algorithme $x^i = (x_{i,0}, ..., x_{i,63})$ est partagée en deux blocs de 32 bits :

$$L = (x_{i,0}, ..., x_{i,31}), \quad R = (x_{i,32}, ..., x_{i,63}).$$

Le bloc de gauche, L, vu comme 8 blocs de 4 bits, à savoir $(x_{i,0}, ..., x_{i,3})$, $(x_{i,4}, ..., x_{i,7})$, ..., $(x_{i,28}, ..., x_{i,31})$ est expansé en 8 blocs de 6 bits, chacun obtenu par addition, aux deux positions extrêmes, du bit adjacent (modulo 31) du bloc suivant. Ainsi le résultat de l'expansion E de L est la suite des blocs $\tilde{L}$ définie par :

$$\tilde{L} = (x_{i,31}, x_{i,0}, ..., x_{i,3}, x_{i,4}), (x_{i,3}, x_{i,4}, ..., x_{i,7}, x_{i,8}) ... (x_{i,27}, x_{i,28}, ..., x_{i,31}, x_{i,0})$$

■ **Étape 3** : la clef $K_{(i)}$ dérivée de K à la i^e itération est ajoutée (modulo 2) au bloc $\tilde{L}$. Le résultat, sur 48 bits, est donc le vecteur :

$$\tilde{M} = (x_{i,31} \oplus k_{i,0}, x_{i,0} \oplus k_{i,1}, ..., x_{i,4} \oplus k_{i,5}, ..., x_{i,27} \oplus k_{i,43}, ..., x_{i,8} \oplus k_{i,47})$$

■ **Étape 4** : $\tilde{M}$ peut être vu à son tour comme une suite de 8 blocs de 6 bits. Chacun de ces blocs est à son tour transformé en un bloc de 4 bits au moyen de la transformation non linéaire suivante.

Si le k^e bloc de $\tilde{M}$ se présente sous la forme $M_k = (z_0, z_1, z_2, z_3, z_4, z_5)$, alors les mots binaires (z_0, z_5) et (z_1, z_2, z_3, z_4) représentent respectivement l'indice de ligne (entre 1 et 4) et de colonne (entre 1 et 16) d'une matrice $S(k)$ dont toutes les entrées sont des nombres compris entre 0 et 15.

Pour le k^e bloc de $\tilde{M}$, le résultat de la transformation non linéaire est tout simplement, dans la lecture de la matrice $S(k)$, le nombre, codé sur 4 bits, situé à l'intersection de la ligne codée par (z_0, z_5) et de la colonne codée par (z_1, z_2, z_3, z_4).

Les matrices $S(k)$ sont indiquées sur le tableau 1. Par exemple, pour le bloc : $M_0 = (0, 0, 1, 0, 1, 1)$, le résultat de la transformation non linéaire est 2 [donc (0, 0, 1, 0)] puisque 2 [donc le nombre de 4 bits (0 0 1 0)] est situé à l'intersection de la 01 = première ligne et de la 0 1 0 1 = 5^e colonne de $S(0)$.

Tableau 1 – Fonction $S(i)$ avec $0 \leqslant i < 8$

	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	13	1	10	6	12	11	9	5	3	8	***S* (0)**
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	
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	***S* (1)**
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	
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	***S* (2)**
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	
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	***S* (3)**
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	
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	***S* (4)**
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	
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	13	8	***S* (5)**
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	
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	***S* (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	
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	***S* (7)**
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	

Étape 5 : les 8 blocs de 4 bits ainsi créés constituent un mot de 32 bits, $\tilde{y} = (y_{i,0}, ..., y_{i,31})$, soumis à une permutation P (décrite tableau 2), pour fournir en sortie le mot permuté $\bar{y} = (y_{i,15}, ..., y_{i,24})$.

Tableau 2 – Fonction P : permutation de 32 bits

15	6	19	20
28	11	27	16
0	14	22	25
4	17	30	9
1	7	23	13
31	26	2	8
18	12	29	5
21	10	3	24

Étape 6 : le bloc $\bar{y}$ est ajouté modulo 2 au mot $L = (x_{i,0}, ..., x_{i,31})$ pour former le mot $(x_{i,0} \oplus y_{i,15} ...)$.

L'entrée $\tilde{x} = (x_{i+1,0}, ..., x_{i+1,63})$ de la $(i+1)^e$ itération du DES est alors obtenue en échangeant parties gauche et droite du mot $(x_{i,0} \oplus y_{i,15}, ..., x_{i,31} \oplus y_{i,24}, x_{i,32}, ..., x_{i,63})$.

On a donc :

$$\tilde{x}_{i+1} = (x_{i,32}, ..., x_{i,63}, x_{i,0} \oplus y_{i,15}, ..., x_{i,31} \oplus y_{i,24})$$

Il y a en tout 16 itérations de ce type ; à la dernière, l'échange des moitiés gauche et droite est omis et remplacé par l'application de la permutation IP^{-1}.

À chaque étape, 48 bits de clef $k_i = (k_{i,0}, ..., k_{i,47})$ sont requis. Ces clefs k_i sont dérivées de la clef initiale par le processus suivant.

Étape 1' : initialement, les 56 bits de clef sont permutés suivant la permutation $PC\,1$ donnée tableau 3. Le vecteur de 56 bits résultant est vu comme composé d'une moitié gauche C_0 et d'une moitié droite D_0, de 28 bits chacune.

Tableau 3 – Permutation *PC* 1

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

Étape 2' : suivant le numéro i de l'itération, C_0 et D_0 sont décalés de $\sigma(i)$ places vers la gauche où $\sigma(i)$ est la fonction donnée tableau 4. Soit :

$$C_1 = g^{\sigma(i)}\, C_0,\ D_1 = g^{\sigma(i)}\, D_0 \text{ les blocs résultants.}$$

$g^{\sigma(i)}$ est l'opération de décalage circulaire de $\sigma(i)$ places vers la gauche.

Tableau 4 – Nombre de décalage $\sigma(i)$

i	$\sigma(i)$	i	$\sigma(i)$
1	1	9	1
2	1	10	2
3	2	11	2
4	2	12	2
5	2	13	2
6	2	14	2
7	2	15	2
8	2	16	1

Étape 3' : la fonction $PC\,2$ sélectionne 48 des 56 bits, dans l'ordre indiqué tableau 5.

Tableau 5 – Permutation *PC* 2

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

Ces 48 bits constituent la clef k_1 utilisée à la première itération.

Par la suite, étant donné les blocs C_i et D_i de k_i, la fonction σ décale vers la gauche C_i et D_i de $\sigma(i+1)$ places à la $(i+1)^{e}$ itération. $PC\,2$, vu alors comme une permutation de 48 bits, appliquée au mot de 48 bits $\tilde{k}_{i+1} = (C_{i+1}, D_{i+1}) = (g^{\sigma(i+1)} C_i, g^{\sigma(i+1)} D_i)$.

Cela permet alors de générer k_{i+1} par $k_{i+1} = PC\,2\,(\tilde{k}_{i+1})$.

L'algorithme DES ainsi décrit peut paraître effectivement extrêmement complexe et difficile à cryptanalyser. Il l'est en effet et nous reportons le lecteur aux comptes rendus des conférences Crypto et Eurocrypt pour la découverte des propriétés connues du DES, qui, par ailleurs, possède encore de nombreux mystères. Les principales faiblesses supposées du DES concernent :

— **la longueur de la clef** : il n'est pas certain que 56 bits (soit un espace de recherche de 2^{56}) soit une longueur suffisante pour résister aux moyens matériels et logiciels maintenant disponibles ;

— **les fonctions *S* (*i*)** : classifiés par la National Security Agency (NSA), les critères de réalisation des fonctions $S(i)$ n'ont jamais été rendus publics. Aussi est-il difficile de se prononcer sur le caractère fortement non linéaire de leur comportement.

3.3.2 Modes d'utilisation

Le DES peut être utilisé suivant quatre modes principaux que nous allons décrire à présent.

3.3.2.1 Mode Électronic Code Book (ECB)

C'est le mode décrit précédemment (figure **8*a***). Le message M est partagé en blocs de texte clair de 64 bits $P_1\,P_2 \ldots P_k$. Chaque bloc est encrypté par le processus :

$$P_k \mapsto C_k = \mathrm{DES}_K\,(P_k)$$

Le déchiffrage s'effectue par :

$$C_k \mapsto P_k = \mathrm{DES}_K^{-1}\,(P_k)$$

Nota : l'inverse du DES est l'application composée, dans l'ordre inverse, des 33 applications constituant le DES.

3.3.2.2 Mode Cipher Block Chaining (CBC)

On a ici, en plus, besoin d'un vecteur d'initialisation I (le même pour émetteur et récepteur) de 64 bits (figure **8*b***). M est partagé en blocs de 64 bits : $M = P_1\,P_2 \ldots P_k$. On encrypte par le processsus :

$$C_1 = \mathrm{DES}_K\,(P_1 \oplus I)$$

$$C_k = \mathrm{DES}_K\,(P_k \oplus C_{k-1})$$

Dans ces conditions, le cryptogramme C_k dépend non seulement de P_k, mais aussi de C_{k-1}. Le système est beaucoup plus difficile à briser. Le déchiffrage s'obtient immédiatement par :

$$C_0 = I$$

$$P_k = \mathrm{DES}_K^{-1}\,(C_k) \oplus C_{k-1}$$

3.3.2.3 Mode Cipher Feedback Chaining (CFC)

Dans ces conditions, on a encore une valeur d'initialisation I (figure **8*c***).

Le message est partagé en blocs de m bits, où m est un entier compris entre 1 à 64.

Le cryptogramme C_1 s'obtient comme suit. On calcule les m premiers bits de $\mathrm{DES}_K\,(I)$. On a alors,

si $M = P_1 \ldots P_k$ (chaque P_i faisant m bits de long)

$$C_1 = P_1 \oplus (\mathrm{DES}_K\,(I))_m$$

où $(\mathrm{DES}_K\,(I))_m$ représente la fonction prenant les m premiers bits (poids forts) de $\mathrm{DES}_K\,(I)$.

On modifie I en I_1 comme suit : on décale I de m places vers la gauche. I_1 s'obtient alors en concaténant le résultant avec C_1, d'où les équations d'encryption :

$$C_k = P_k \oplus [\mathrm{DES}_K\ (\mathrm{left}_m(I_{k-1}) \,\|\, C_{k-1})]_m$$

Ici, left_m est le décalage à gauche de m places, et $\|$ désigne la concaténation.

Le mode CFC est très utilisé en mode encryption de caractère (et alors m vaut 7 ou 8 le plus souvent).

3.3.2.4 Mode Output-Feedback Chaining (OFC)

Dans ce dernier mode (figure **8*d***), le DES est utilisé comme générateur pseudo-aléatoire. Partant d'une valeur d'initialisation I, et pour m fixé à l'avance, on calcule la suite de vecteurs aléatoires de m bits définis par :

$$\tilde{C}_k = (\mathrm{DES}_K\ (\mathrm{left}_m\ I_{k-1} \,\|\, [\mathrm{DES}_k\ (I_{k-1})]_m))_m$$

Le cryptogramme C_k est alors simplement $C_k = \tilde{C}_k \oplus P_k$. La question de la période du DES dans ce mode de fonctionnement est ouverte ; statistiquement, on peut montrer qu'avec une forte probabilité elle avoisine 2^{63} sous certaines conditions [41]. La figure **8** résume les quatre modes de fonctionnement du DES.

Les figures **9** et **10** donnent le schéma général de l'algorithme DES pour le calcul de la fonction non linéaire $f(R_{i-1}, K_i)$ et le calcul de la clef K_i à la i^{e} itération.

3.4 Cryptosystèmes à clefs publiques

3.4.1 Principe

L'idée de base des cryptosystèmes à clefs publiques a été proposée dans un article fondamental de Diffie et Hellman en 1976 [11]. Le principe fondamental est d'utiliser des clefs de chiffrement et déchiffrement différentes, non reconstructibles l'une à partir de l'autre. Les avantages fondamentaux de ce type de cryptosystème sont les suivants.

■ Si K_s et K_p sont les deux clefs possédées par un utilisateur, l'une peut être rendue publique, et l'autre secrète ; d'où un premier avantage, celui de la **confidentialité**. Connaissant la clef publique de l'utilisateur A, disons K_{pA}, chacun peut encrypter sous cette clef, mais seul A peut déchiffrer, grâce à la clef secrète K_{sA}.

■ **Authenticité** : si un message est encrypté par un utilisateur avec la clef K_{sA}, alors chacun peut le déchiffrer avec K_{pA}, mais seul A est capable de l'avoir chiffré, car il est le seul à connaître K_{sA}.

Pour mettre en œuvre un algorithme de chiffrement à clefs publiques, il est donc nécessaire de disposer de deux transformations, P (dépendant de K_p) et S (dépendant de K_s) telles que $P[S(M)] = M$ et $S[P(M)] = M$ pour tout message M soumis à chiffrement. Nous donnons ci-après les deux systèmes fondamentaux de chiffres à clefs publiques publiés dans la littérature.

3.4.2 Cryptosystèmes de type Knapsack (sac à dos)

Ces systèmes sont basés sur le fait que le problème suivant est NP complet (c'est-à-dire non soluble en temps polynomial par tout algorithme déterministe).

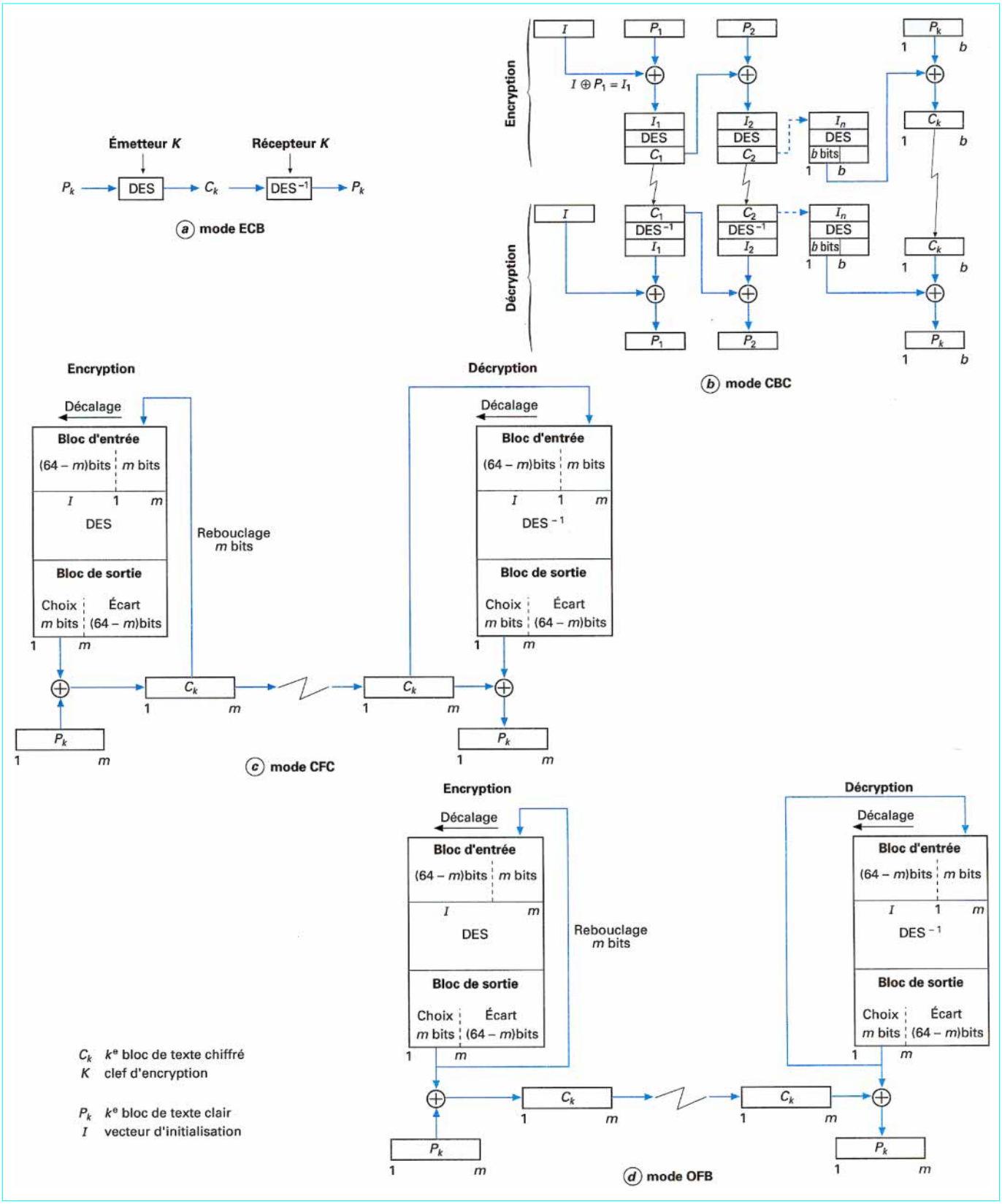


Figure 8 – Modes de fonctionnement du DES

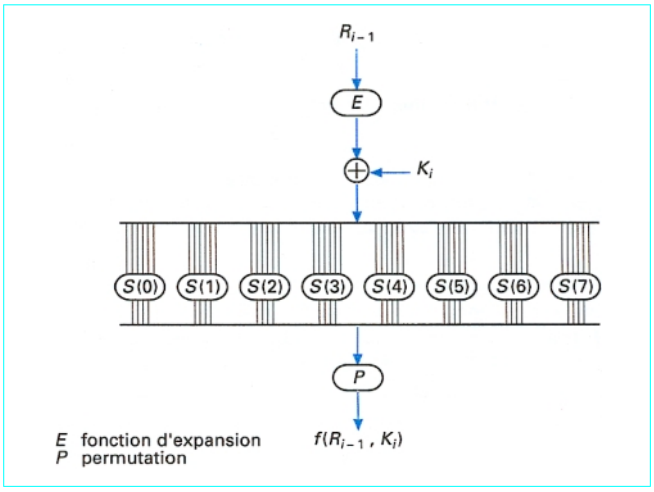


Figure 9 – Description globale du calcul de $f(R_{i-1}, K_i)$

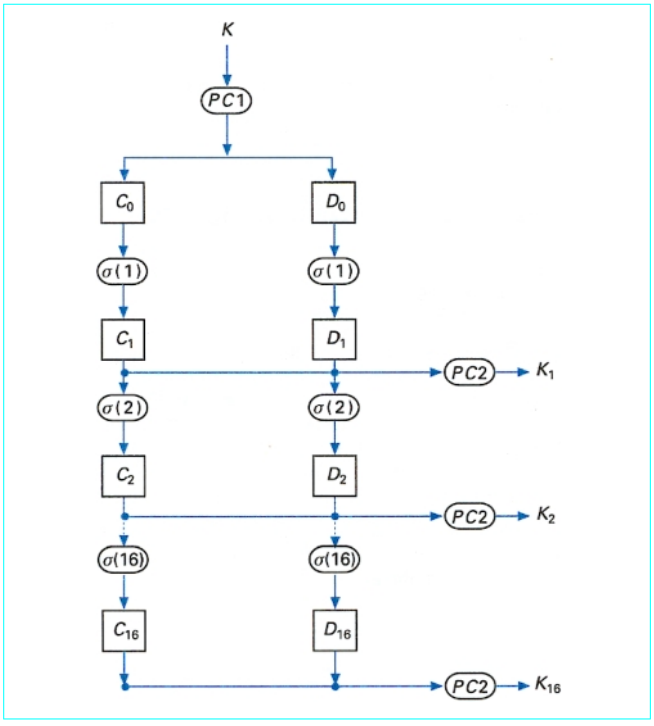


Figure 10 – Algorithme général de calcul de la clef à la i^e itération

Étant donné $n+1$ entiers $S, a_1, ..., a_n$, on recherche n nombres $x_1, x_2, ..., x_n$ dans {0, 1} tels que $S = \sum_{i=1}^{n} a_i x_i$.

Dans un cryptosystème de type Knapsack, chaque utilisateur possède une clef publique $(T, a_1, ..., a_n)$ formée de $n+1$ nombres entiers et une clef secrète $(W, s_1, ..., s_n)$ formée de $n+1$ nombres entiers également soumis aux conditions suivantes :

- la suite s_i est surcroissante : $\forall i,\ s_{i+1} > \sum_{j=1}^{i} s_j$
- $T > \sum_{i=1}^{n} s_i$
- $1 \leqslant W < T$ et PGCD $(W, T) = 1$
- $a_i = W^{-1} s_i$ (modulo T).

Pour envoyer un message chiffré à A, B lit dans la table $(T^A, a_1^A, ..., a_n^A)$.

Le message $M = (t_1, ..., t_n) \in \{0, 1\}^n$ est encrypté par :

$$M \mapsto \text{Knap}_A(M) = \sum_{i=1}^{n} a_i^A t_i \pmod{T^A} = U$$

La décryption se fait, côté A, par le processus suivant :

A calcule $V = U W^A \pmod{T^A}$

On a $V = \sum_{i=1}^{n} a_i^A W^A t_i \pmod{T^A} = \sum_{i=1}^{n} s_i t_i \pmod{T^A}$.

Comme la suite s_i est surcroissante, les t_i peuvent être reconstitués en temps linéaire.

Historiquement le premier cryptosystème de ce type à avoir été utilisé, le Knaspsack, possède de nombreuses faiblesses cryptographiques. Son utilisation directe n'est pas recommandée, comme le lecteur pourra s'en convaincre en parcourant les articles des comptes rendus des conférences Crypto et Eurocrypt qui lui ont été consacrés [12] [13].

3.4.3 Cryptosystème RSA (Rivest, Shamir, Adleman)

C'est un système de chiffrement exponentiel, dans lequel encryption et décryption sont réalisées par des fonctions du type [14] :

$$M \mapsto C = a^A \pmod{M}$$

Dans le cryptosystème RSA, chaque utilisateur dispose :

— **d'une clef publique**, constituée d'un nombre n très grand, produit de deux nombres premiers p et q, et d'un nombre E choisi au hasard dans l'intervalle $[2, (p-1)(q-1)-1]$;

— **d'une clef secrète** formée du produit $\varphi(n) = (p-1)(q-1)$ des facteurs premiers de n (φ appelé fonction totient d'Euler) et d'un nombre D tel que $ED \equiv 1$ (modulo $\varphi(n)$) où $\equiv$ représente la congruence.

Les messages à encrypter sont découpés en blocs de telle façon que chaque bloc M_i soit une représentation binaire d'un chiffre de l'intervalle $[0, n-1]$.

Quand un utilisateur B veut envoyer un message à A, il le chiffre au moyen de la clef publique de A, définie par n_A, E_A, comme suit :

$$M \mapsto C = M^{E_A} \pmod{n_A}$$

A décrypte C au moyen de sa clef secrète D_A *via* :

$$C \to C^{D_A} = M^{E_A D_A} = M^{1 + K\varphi(n_A)} \pmod{n_A}$$

Un théorème classique d'arithmétique, connu sous le nom de *petit théorème de Fermat*, affirme en effet que, si PGCD $(M, p) = 1$, alors :

$$M^{\varphi(p)} \equiv 1 \bmod n_A$$

On en déduit aisément que $C^{D_A} = M$, d'où l'équation de déchiffrage du RSA.

La sécurité du cryptosystème RSA repose sur le problème suivant de théorie des nombres, connu pour ne pas avoir de solution efficace : étant donné un nombre très grand, il est très difficile de calculer sa décomposition en facteurs premiers. Autrement dit, connaissant $n = p\,q$, il est très difficile, sinon impossible, de retrouver p et q à partir de n. Dans la pratique, n est un nombre dont la représentation binaire est de l'ordre de grandeur de 350 à 400 bits.

De bons choix de p et q ont été proposés : ils doivent être de tailles comparables et les facteurs premiers de $p-1$ et $q-1$ les plus grands possible.

La cryptanalyse du système RSA est, au mieux, aussi difficile que le problème de la factorisation de n, ces deux questions n'étant pas équivalentes.

La factorisation des grands nombres ($\geqslant$ 100 chiffres décimaux) est un problème très difficile ; les meilleurs algorithmes connus (cf., par exemple, [15]), utilisent la méthode dite du **crible quadratique**, qui énonce le résultat suivant :

$$\text{si } X^2 \equiv Y^2 \text{ (modulo } n\text{) et si } X \neq \pm Y \text{ (modulo } n\text{)}$$

alors le PGCD de $(X \pm Y, n)$ divise n.

Il est ici impossible, dans un espace aussi limité, de donner toute considération théorique sur la factorisation des grands nombres. Signalons simplement que des développements récents, utilisant la théorie des courbes elliptiques, ont permis des progrès spectaculaires en la matière, *ressuscitant* ainsi une théorie pratiquement oubliée depuis un siècle [16].

Pour conclure, mentionnons que le RSA est maintenant largement utilisé, en particulier pour l'encryption de messages courts (distributions de clefs DES par exemple). Les circuits intégrés implémentant l'algorithme ont des performances assez modestes (de l'ordre de 10 à 100 kbit/s, contre 10 à 20 Mbit/s, pour des circuits implémentant le DES). Cela est dû essentiellement à la lenteur des opérations (même câblées) de multiplication et d'exponentiation modulo un grand nombre. Il paraît cependant évident, malgré cette faiblesse intrinsèque, que le système RSA doit jouer, dans un futur proche, un rôle de plus en plus important dans les applications.

3.5 Sécurité des systèmes classiques

3.5.1 Sécurité du DES

Le DES est aujourd'hui le cryptosystème le plus étudié au niveau théorique et le plus utilisé dans les applications civiles. En particulier, il est bien adapté, du point de vue vitesse des calculs, à l'encryption de grandes masses de données. Sa longueur de clef relativement faible, jointe à l'absence de certitude sur certains critères de conception de l'algorithme (en particulier ceux liés aux *boîtes S* qui pourraient présenter certaines faiblesses induites intentionnellement par la National Security Agency) ont longtemps laissé planer quelques doutes sur sa sécurité. De fait, et même si l'algorithme DES a été recertifié comme standard fédéral pour 5 ans en 1993 par le National Institute for Standards and Technology (NIST), cette recertification pourrait bien être la dernière. Depuis 1990, de nouvelles et puissantes techniques de cryptanalyse des systèmes de types DES sont en effet apparues, telles celles basées sur la cryptanalyse **différentielle** ou **linéaire**.

La première technique de cryptanalyse du DES plus efficace que la méthode exhaustive (recherche parmi les 2^{55} configurations de clef possibles) date de 1990. Lors de la conférence *Crypto 90*, Biham et Shamir [24] présentèrent le concept de cryptanalyse différentielle. L'idée de base de la technique est que, si des textes clairs choisis au hasard produisent, une fois encryptés par le DES, des cryptogrammes de nature apparemment aléatoire, cela n'est plus nécessairement vrai dès lors que l'on s'intéresse aux cryptogrammes produits à partir de textes clairs ***voisins***. Deux textes clairs P et P^*, dont le OU exclusif bit à bit $P' = P \oplus P^*$ est choisi de manière convenable, peuvent produire des cryptogrammes correspondants C et C^* tels que $C' = C \oplus C^*$ prend des valeurs prédéterminées en certaines positions binaires avec une probabilité non uniforme. À côté du cas trivial $P' = 0$ (d'où $C' = 0$), Biham et Shamir découvrirent par exemple que sur 5 itérations bien déterminées du DES, le texte clair d'expression hexadécimale P' = 405C000004000000 produit le cryptogramme C' = 04000000405C0000 avec une probabilité 1/10 486, très supérieure à la valeur attendue d'une distribution uniforme des bits du cryptogramme. À l'aide de remarques de ce type, ils mirent en œuvre une stratégie d'attaque d'une version à 8 itérations du DES, utilisant la correspondance entre 5 000 textes clairs *voisins* et les textes chiffrés associés ; à partir de cette remarque, ils démontrèrent que le concept de cryptographie différentielle permettait d'envisager une attaque du DES complet à partir d'une correspondance entre 2^{47} textes clairs choisis et les cryptogrammes associés (soit un facteur d'accélération de 2^8 par rapport à une attaque exhaustive sur les 56 bits de clef).

A. Shamir améliora encore la technique en 1992, lors de la conférence *Crypto* annuelle, et démontra qu'il était possible, en théorie, de casser le DES complet avec un nombre de pas de calcul de l'ordre de 2^{37}, concentrés sur l'analyse de 2^{36} cryptogrammes choisis avec soin parmi les 2^{47} issus de la méthode proposée en 1990 [25].

En 1994, lors de la conférence annuelle *Crypto*, un chercheur de Mitsubishi Electric Corporation, Mitsuru Matsui, a présenté la première cryptanalyse pratique du DES complet, démontrant qu'il était possible, avec une forte probabilité de succès, de casser l'algorithme dès lors que 2^{43} correspondances **textes clairs** $\leftrightarrow$ **textes chiffrés** étaient disponibles (une situation maintenant réalisable, compte tenu des moyens et puissances de calcul disponibles). Il publiait à cette occasion le premier résultat de cryptanalyse *linéaire* : pour retrouver les 56 bits de clefs, il lui avait fallu 50 jours de calcul effectués en parallèle sur douze stations Hewlett-Packard *HP9735/PARISC* à 99 MHz de fréquence d'horloge. Le lecteur intéressé par le descriptif complet de la méthode utilisée est renvoyé à l'article original de Matsui [22].

Le principe de la cryptanalyse linéaire consiste à observer que quelques relations linéaires binaires entre certains bits du texte clair, certains bits du texte chiffré, et certains bits de la clef ont une probabilité d'occurrence différente de celle qui devrait être observée si le DES était un système de chiffrement *purement aléatoire*.

De manière un peu plus précise, deux équations de ce type suffisent à Matsui pour récupérer 26 des 56 bits de clefs.

Reprenons les notations classiques du DES :

— P représente les 64 bits de texte clair après permutation initiale IP ;

— C représente les 64 bits du cryptogramme associé avant la permutation IP^{-1} ;

— P_H, P_L représentent les 32 bits de poids fort et faible de P respectivement ;

— C_H, C_L représentent les 32 bits de poids fort et faible de C respectivement ;

— K représente la clef DES cherchée, après permutation $PC1$;

— K_r représente la clef de 48 bits dérivée de K à l'itération r du DES ;

— $f_r(x_r, K_r)$ est la fonction f non linéaire du DES lors de l'itération r ;

— $A[i]$ est le bit de position i du vecteur binaire A ;

— $A[i, j, ..., k] = A[i] \oplus A[j] \oplus ... \oplus A[k]$ (OU exclusif).

Matsui remarque alors que les deux équations ci-après ont une probabilité d'occurrence sensiblement égale à $1/2 - (1{,}19) \times 2^{-21}$:

$$P_L[7, 18, 24] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] = K_2[22] \oplus K_3[44] \oplus K_4[22] \oplus K_6[22] \oplus K_7[44] \oplus K_8[22] \oplus K_{10}[22] \oplus K_{11}[44] \oplus K_{12}[22] \oplus K_{14}[22] \quad \textbf{(3)}$$

$$C_L[7, 18, 24] \oplus P_H[7, 18, 24, 29] \oplus P_L[15] = K_{13}[22] \oplus K_{12}[44] \oplus K_{11}[22] \oplus K_9[22] \oplus K_8[44] \oplus K_7[22] \oplus K_5[22] \oplus K_4[44] \oplus K_3[22] \oplus K_1[22] \quad \textbf{(4)}$$

En appliquant ces équations pour les itérations 2 à 15 incluses du DES, on arrive aux deux équations linéaires suivantes, elles aussi réalisées avec la même probabilité :

$$P_H[7, 18, 24] \oplus f_1(P_L, K_1)[7, 18, 24] \oplus C_H[15] \oplus C_L[7, 18, 24, 29] \oplus f_{16}(C_L, K_{16})[15] = K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] \oplus K_8[44] \oplus K_9[22] \oplus K_{11}[22] \oplus K_{12}[44] \oplus K_{13}[22] \oplus K_{15}[22] \quad \textbf{(5)}$$

$$C_H[7, 18, 24] \oplus f_{16}(C_L, K_{16})[7, 18, 24] \oplus P_H[15] \oplus P_L[7, 18, 24, 29] \oplus f_1(P_L, K_1)[15] = K_{14}[22] \oplus K_{13}[44] \oplus K_{12}[22] \oplus K_{10}[22] \oplus K_9[44] \oplus K_8[22] \oplus K_6[22] \oplus K_5[44] \oplus K_4[22] \oplus K_2[22] \quad \textbf{(6)}$$

Chacune des équations (5) et (6) met en œuvre deux des boîtes S du DES et permet, par examen d'un nombre significatif de correspondances texte clair ↔ texte chiffré, de reconstituer, par résolution au maximum de vraisemblance, les configurations les plus probables pour 13 bits de la clef K inconnue. On construit ainsi, en rassemblant les résultats des deux équations, une suite (W_m) $(m = 0, ..., 2^a)$ de vecteurs de 26 bits de clefs recherchés. Pour chacun des vecteurs obtenus, on effectue alors une recherche exhaustive sur les 30 bits de clefs restant, jusqu'à trouver, dans les correspondances texte clair ↔ texte chiffré la valeur correcte de la clef secrète DES. Pour $a = 13$, donc avec 2^{13} vecteurs W_m, la probabilité de succès de l'étape de recherche exhaustive est alors voisine de 0,86.

Une telle recherche exhaustive sur 2^{30} configurations est maintenant à la portée d'un réseau de stations de travail moderne, sur lequel il est facile de *paralléliser* les calculs (par exemple les correspondances texte clair ↔ texte chiffré). Comme l'indique Matsui, le temps de préparation (c'est-à-dire le calcul des correspondances clair ↔ chiffré pour 2^{43} messages, voisin de 40 jours de calculs sur 13 stations, est largement supérieur au temps d'exécution de l'algorithme (10 jours). Cela démontre donc, en attendant sûrement d'autres améliorations des techniques de cryptanalyse, que le DES est en théorie cassable par une organisation qui voudrait s'en donner les moyens et investir, par exemple, dans la création de grands dictionnaires de correspondances clair ↔ chiffré. Ces moyens étant cependant encore largement hors de portée de l'immense majorité des particuliers et entreprises, le DES reste encore considéré comme sûr pour la **plupart des applications non sensibles**.

Par ailleurs, un résultat récent de Campbell et Wiener, présenté à la conférence *Crypto 92* [23], démontre que l'ensemble des transformations de type DES ne forme pas un groupe, au sens mathématique du terme. De manière plus précise, étant donné 2 clefs de 56 bits K_1 et K_2, la composée des deux transformations $\mathrm{DES}_{K_1} \circ \mathrm{DES}_{K_2}$ n'est pas une transformation du type DES_{K_3}, en général. On en déduit des mécanismes de renforcement de la sécurité du DES, par encryption triple par exemple, où, étant donné deux clefs de 56 bits K_1 et K_2, un message M sera encrypté en le cryptogramme $C = (\mathrm{DES}_{K_1})^{-1} \circ \mathrm{DES}_{K_2} \circ \mathrm{DES}_{K_1}(M)$, étendant ainsi considérablement le nombre de configurations de clefs possibles. Malgré toutes ces améliorations, un réexamen de la situation d'ici cinq ans environ semble cependant plus que nécessaire.

3.5.2 Sécurité du système RSA

3.5.2.1 Généralités

Nota : pour la description de l'algorithme, le lecteur se reportera aux références [22] [14] et au paragraphe 3.4.3.

La sécurité de l'algorithme RSA tient à la conjonction de plusieurs facteurs :

— la non-divulgation des nombres p_A et q_A intervenant dans la décomposition du nombre n_A choisi par (ou attribué à) l'utilisateur A ; la connaissance accidentelle ou forcée de l'une de ces deux quantités par *intercepteur* suffit en effet pour retrouver la clef secrète D_A de l'utilisateur A à partir de sa clef publique E_A et de la factorisation de n_A ;

— la difficulté pratique de la factorisation de n_A. Tous les algorithmes connus aujourd'hui ont un temps de calcul prohibitif en pratique, dès lors que la taille de n est choisie suffisamment grande (de l'ordre de 512 à 1 024 bits) ;

— l'absence de méthode algébrique connue permettant de calculer la clef secrète D_A de A à partir du nombre n et de sa clef publique E_A.

Les attaques contre le système RSA sont *a priori* de plusieurs types possibles :

— améliorer les techniques actuelles de factorisation de très grands nombres ;

— attaquer directement le générateur de nombres premiers (nécessairement basé sur un générateur de nombres pseudo-aléatoires initialisé par une souche donnée) utilisé dans le système.

En pratique, ce sont les techniques de factorisation qui intéressent le plus les chercheurs, d'autant qu'elles se prêtent à une implémentation relativement aisée sur un réseau d'ordinateurs. L'état de l'Art en 1996 est que les algorithmes actuels permettent la factorisation de nombres de l'ordre de 130 chiffres décimaux. Un succès spectaculaire a d'ailleurs été obtenu en 1994, lorsqu'un chercheur de la société Bellcore, Arjen Lenstra, aidé d'un groupe d'algorithmiciens parmi lesquels Derek Atkin, Michael Graff et Paul Leyland coordonnèrent les travaux d'environ 600 volontaires pour relever le défi lancé en 1976 par R. Rivest dans un article publié par M. Gardner au *Scientific American*, où il exposait les principes du système RSA et offrait une prime symbolique de 100 $ à la première personne qui lui enverrait la factorisation complète d'un nombre de 129 chiffres décimaux, connu depuis sous le nom de RSA-129. La preuve de cette factorisation consistait en le déchiffrement d'un message encrypté par la méthode RSA avec le module RSA-129. En 1976, R. Rivest estimait que la charge de travail nécessaire pour factoriser un tel nombre équivalait à 40 millions de milliards d'années de calcul sur les machines informatiques existant à l'époque (de puissance de l'ordre du MIPS ou Million d'Instructions Par Seconde). Utilisant les techniques de factorisation les plus avancées, A Lenstra et son équipe invitèrent des chercheurs du monde entier à relever le challenge et à participer à l'exécution des calculs nécessaires en se procurant, *via* le réseau Internet, des fichiers sources à compiler puis à lancer sur leurs machines. Chaque volontaire (il y en eut au total plus de 600, de 25 pays différents, le tout représentant environ 1 500 machines sollicitées ainsi à travailler durant leur période de disponibilité) reçut ainsi un sous-ensemble du problème de factorisation et réexpédiait les résultats au site central du MIT qui assurait la consolidation des résultats. Après 7 mois d'efforts, la factorisation de RSA-129 fut annoncée lors d'une réunion tenue le 26 avril 1994 à New York. Le nombre choisi par R. Rivest s'écrit, en base 10 :

n = 114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541

(en notation binaire, n est représentable par 406 digits).

Ses facteurs premiers sont :

p = 3490529510847650949147849619903898133417764638493387843990820577

q = 3276913299326670954996198819083446141317764296799294253979828853 3

Ce résultat spectaculaire permit à R. Rivest de relire le message encrypté 18 ans plus tôt (dont l'intitulé en clair *the magic words are squeamish ossifrage* n'est cependant pas des plus limpides !), soit légèrement en avance par rapport à ses prévisions. En fait, l'unique erreur de R. Rivest fut d'avoir au départ (mais qui pourrait le lui reprocher ?) sous-estimé la fantastique évolution des puissances de calcul depuis les deux dernières décennies, puisque son estimation initiale de 40 millions de milliards d'années correspond grosso modo au nombre d'opérations élémentaires effectuées sur

les 1 600 machines utilisées dans le monde lors de l'expérience conduite par A. Lenstra.

Ce résultat ne représente pas en lui-même une menace pour l'algorithme RSA. En effet, même si les machines progressent rapidement et si les algorithmes de factorisation les plus efficaces permettent approximativement de traiter des nombres 100 fois plus gros tous les deux ans en moyenne, il suffit de choisir des nombres n de plus en plus grands pour maintenir l'écart entre sécurité pratique et efficacité de la cryptanalyse.

Dans un article récent [27], R. Rivest estime que le nombre d'opérations requises (en MIPS année : unité de mesure signifiant qu'un ordinateur de 1 MIPS travaille en continu pendant une année) pour factoriser de très grands nombres, est donné par les estimations du tableau **6**.

Tableau 6 – Nombre d'opérations requises pour factoriser de très grands nombres [14]

Taille de n bits	Borne inférieure	Charge moyenne	Borne supérieure
512	59,23	$5{,}2 \times 10^5$	$5{,}2 \times 10^5$
1 024	$1{,}7 \times 10^7$	$3{,}8 \times 10^{12}$	$3{,}8 \times 10^{12}$
2 048	$2{,}6 \times 10^{14}$	$4{,}3 \times 10^{21}$	$4{,}3 \times 10^{21}$
4 096	$7{,}7 \times 10^{23}$	$3{,}4 \times 10^{33}$	$3{,}4 \times 10^{33}$
8 192	$2{,}0 \times 10^{36}$	$1{,}4 \times 10^{49}$	$1{,}4 \times 10^{49}$

Il en tire les estimations du tableau **7** sur la taille des nombres (en bits) pouvant être factorisés par les algorithmes actuels, jusqu'à l'horizon 2020.

Tableau 7 – Taille des nombres (en bits) pouvant être factorisés avec les algorithmes actuels

Année	Taille minimale	Taille moyenne	Taille maximale
1995	405	579	1 341
2000	425	619	1 451
2005	447	661	1 567
2010	469	705	1 689
2015	493	751	1 915
2020	515	799	1 947

Ces estimations ne doivent bien entendu pas être considérées comme absolues, mais donnent cependant une idée assez vraisemblable des scénarios de choix des nombres à implémenter dans le système RSA. Bien sûr, toute révolution soit en matière de puissance de calcul, soit en matière de techniques de factorisation, pourra amener à une reconsidération partielle ou complète de ce tableau.

Les résultats ci-dessus doivent cependant être interprétés avec précaution et l'on doit écarter le préjugé trop hâtif qui conduirait aux conclusions que le RSA est une méthode d'encryption beaucoup plus sûre que le DES, et qu'il vaut mieux, en pratique, utiliser des cryptosystèmes à clefs publiques plutôt que des cryptosystèmes symétriques. Le RSA souffre en effet d'un problème inhérent à sa structure arithmétique (calculs modulo de très grands nombres), à savoir qu'il est très lent, donc pratiquement inopérant sur des grosses masses de données ou des fichiers importants. Implémenté en logiciel, le DES est en général 1 000 fois plus rapide que le système RSA. Avec des tailles de l'ordre de 512 bits, les circuits intégrés les plus rapides implémentant le RSA ne dépassent guère un débit en encryption/décryption de l'ordre de 1 Mbit/s, alors que des débits 100 fois plus élevés peuvent être atteints pour des implémentations en hardware du DES. Pour cette raison, DES et RSA sont presque toujours utilisés en association (DES pour l'encryption des messages, RSA pour l'encryption des clefs DES devant être distribuées de manière particulièrement sécuritaire sur le réseau). Un exemple de telle association est illustré par le système PGP décrit au paragraphe 6.

3.5.2.2 Aspects algorithmiques du système RSA

Deux principaux types de problèmes sont associés au système RSA :

— être capable de générer de très grands nombres premiers (et, en particulier, certifier que ces nombres sont effectivement premiers) ;

— trouver un compromis optimal entre taille des nombres n à factoriser (et donc, robustesse vis-à-vis des techniques de factorisation), niveau de sécurité requis et temps d'exécution des calculs au niveau des encryptions/décryptions.

■ Concernant le premier problème, la majorité des algorithmes utilisés en pratique reposent sur des **tests de primalité probabilistes ou déterministes**. Les nombres premiers générés sont, en général, choisis parmi les termes d'une progression arithmétique. On sait en effet depuis le XIX^e siècle (résultat du mathématicien allemand G. Lejeune-Dirichlet) que, si a et m sont deux entiers premiers entre eux, alors la progression arithmétique $G_{a,\,m} = \{n \in \mathbb{N},\ n = ak + m,\ k \geqslant 0\}$ contient une infinité de nombres premiers. Les tests de primalité déterministes les plus efficaces ressuscitent une théorie déjà ancienne (et qui, en partie pour cette raison, mais aussi à cause de ses liens profonds avec le grand théorème de Fermat, s'est retrouvée l'objet de recherches très actives), à savoir la théorie des courbes elliptiques. Il existe aujourd'hui deux principaux tests de primalité basés sur cette théorie, celui dit GK (ou de Goldwasser et Killian) et celui dit ECPP (*Elliptic Curve Primality Proving*). Pour une description complète de ces algorithmes le lecteur intéressé pourra consulter les références [17] [18].

Nous nous contenterons ici d'indiquer le schéma de base de l'algorithme GK. Donnons d'abord la définition suivante.

Soit N un entier différent de 1 et premier avec 6. Soit $A = \mathbb{Z}/N\mathbb{Z}$ l'anneau (au sens mathématique du terme) des classes d'équivalences d'entiers relatifs modulo N (rappel : m et n dans $\mathbb{Z}$ sont dits équivalents modulo N si leur différence $n - m$ est un multiple de N. A représente alors l'ensemble des classes d'équivalence pour cette relation et s'identifie à l'ensemble $\{1, 2, ..., N-1, 0\}$. L'addition et la multiplication sur $\mathbb{Z}$ se transportent alors naturellement sur A et lui confèrent une structure d'anneau).

Définition. Une **courbe elliptique** sur A est constituée des triplets (X, Y, T) de A^3 satisfaisant les conditions suivantes :

(1) X, Y, T ne sont pas simultanément nuls ;

(2) X, Y, Z vérifient une équation du type

$$Y^2\,T = X^3 + a\,X^2\,T + b\,T^3$$

avec a et b dans l'anneau A avec la relation

$$4\,a^3 + 27\,b^2 \in A^*$$

où A^* désigne le groupe des éléments inversibles pour la multiplication dans A.

On notera par la suite $E_{a,\,b}$ une telle courbe.

Le résultat essentiel pour la suite est que, si N est premier, l'ensemble des points de coordonnées (X_p, Y_p, T_p) appartenant à A^3 satisfaisant l'équation $E_{a,\,b}$ est un groupe commutatif d'élément neutre le point $O_E = (0, 1, 0)$.

Le principe de l'algorithme *GK* est alors le suivant. Soit N un très grand nombre entier ; on cherche à savoir si N est premier. On dispose pour cela du résultat arithmétique suivant.

Théorème. On suppose donnés un entier m et un point $P(x, y)$ d'une courbe elliptique $E_{a,b}$ construite sur A et satisfaisant les conditions suivantes :

(1) il existe un diviseur premier q de m tel que $q > (\sqrt[4]{N} + 1)^2$

(2) $m \cdot P = O_E$

(3) $(m/q) \cdot P = (x, y, t)$, avec $t \in A^*$

Alors N est un nombre premier.

L'algorithme GK s'implémente alors de la façon suivante :

1) choisir une courbe elliptique $E_{a,b}$ sur $A = \mathbb{Z}/N\mathbb{Z}$ dont le nombre total de points est du type $m = 2q$, avec q nombre premier ;

2) tester si $(E_{a,b}, m)$ satisfont les conditions du résultat précédent, avec $s = m$; sinon boucler en 1 ;

3) vérifier de la même manière que q est bien premier ; sinon boucler en 1 ;

4) fin de l'algorithme.

On peut montrer que le temps d'exécution de cet algorithme est en $\mathcal{O}\left((\ln N)^{12}\right)$ c'est-à-dire a un temps d'exécution plus petit, en nombre d'étapes élémentaires, que la puissance douzième du logarithme de N, à une constante multiplicative près. En pratique, l'algorithme GK permet de déterminer la primalité de nombres à un millier de digits binaires (800 chiffres décimaux environ).

■ Concernant le problème de la factorisation des très grands nombres, il existe trois grands types d'algorithmes efficaces en pratique :

1) ceux basés sur la théorie des **courbes elliptiques** ;

2) ceux basés sur des variantes de l'**algorithme du crible quadratique**, en particulier la variante dite MPQS (*Multiple Polynomial Quadratic Sieve*) ;

3) ceux basés sur des **cribles dans les corps de nombres** ou *Number Field Sieve* (NFS), en théorie les plus efficaces mais qui ne marchent bien que sur des nombres de forme particulière (du type $n = r^k \pm s$, où r, k, s sont des entiers).

La vitesse de convergence des algorithmes de type 1 et 2 est bornée en nombre d'opérations élémentaires par un terme en $\mathcal{O}\left(\exp\sqrt{\ln N(\ln\ln N)}\right)$, ce qui les rend utilisables, en pratique, sur des nombres de l'ordre de 110 à 120 chiffres décimaux.

L'algorithme NFS (type 3) est en théorie plus rapide, demandant un nombre d'opérations élémentaires borné par $\mathcal{O}\left(\sqrt[3]{\ln N^3}\sqrt{(\ln\ln N)^2}\right)$. Compte tenu des moyens informatiques actuels, l'algorithme NFS permet de factoriser des nombres de la forme particulière décrite ci-dessus, ayant de l'ordre de 130 chiffres décimaux.

Cela démontre qu'aujourd'hui, en pratique, il a environ un ordre de grandeur entre la taille des nombres dont on sait certifier la primalité et la taille des nombres qu'on sait factoriser.

Pour plus de détails, nous renvoyons le lecteur intéressé aux références [17] [18]. Signalons enfin, pour mémoire, qu'il existe un nombre important de logiciels implémentant ces algorithmes, disponibles sur les sites Internet et distribués librement. Nous renvoyons également à la référence [17] pour la liste de tels sites.

4. Signatures digitales

Le développement du commerce électronique pose des problèmes particulièrement difficiles et subtils en matière d'authentification de documents générés et gérés par voie purement électronique et/ou informatique. Compte tenu des incidences financières énormes que peuvent présenter certains types de transactions (échanges interbancaires, ordre d'achat/vente en bourse...), les techniques de signature digitales doivent fournir des garanties (notamment en matière d'authentification du contenu et d'identification) absolument identiques aux signatures manuelles formalisant les contrats *papier* traditionnels.

Dans les paragraphes précédents, nous avons présenté différentes techniques cryptographiques pour la génération de signatures digitales, basées sur des algorithmes symétriques ou asymétriques. Dans la pratique, les schémas de signature utilisant des algorithmes asymétriques sont les plus utilisés. Pour des raisons évidentes liées au développement des transactions électroniques, la nécessité de disposer de standards industriels efficaces de signature et/ou d'authentification électronique est apparue très rapidement. Cette question a fait l'objet de travaux importants depuis les dernières années, aux États-Unis en particulier, et a débouché sur les deux standards que nous présentons à présent, DSS et PKCS.

4.1 Spécification du standard de signature digitale DSS

Le schéma de signature que nous décrivons ci-après est particulièrement important puisqu'il constitue la première tentative d'élaboration d'un standard *de jure* d'authentification et de signature digitale. Ce schéma a été sélectionné par le National Institute of Standards and Technology pour les transactions gouvernementales américaines. Son adoption comme standard officiel au niveau des autorités fédérales (Federal Information Publication-Standards Publication 186 du 19 mai 1994) fait encore l'objet d'un débat d'opinion assez vif aux États-Unis.

Ce standard, appelé DSS ou *Digital Signature Standard*, spécifie un algorithme de signature digitale (dénommé *Digital Signature Algorithm* ou DSA) basé sur un problème bien connu en théorie des nombres et appelé le problème du *logarithme discret*. L'énoncé de ce problème est particulièrement simple :

étant donné un nombre premier p et $\mathbb{F}_q$ un corps à q éléments, où $q = p^n$ est une puissance de ce nombre premier, étant donné d'autre part g un générateur du groupe cyclique de $\mathbb{F}_q$ et deux éléments quelconques non nuls x et y, dans $]0, q^{n-1}[\times \mathbb{F}_q$, il est :

— très facile de calculer g^x dans $\mathbb{F}_q$;

— infaisable calculatoirement de trouver un élément a de $[0, q^{n-1}]$ tel que $g^a = y$.

(a joue alors l'équivalent du logarithme de y dans $\mathbb{F}_q$, d'où la terminologie employée).

Le problème du *logarithme discret* a été utilisé dans de nombreux schémas de signature digitale, notamment ceux popularisés par Schnorr [28] et El Gamal [29]. En fait le standard DSS est une variante des cryptosystèmes correspondants ; ce qui rend l'algorithme DSS particulièrement intéressant tient à deux raisons assez contradictoires :

— du point de vue **sécurité** du système cryptographique sous-jacent, le problème du *logarithme discret* est le seul qui paraisse aujourd'hui capable de rivaliser avec le système RSA proposé par Rivest, Shamir et Adleman et décrit au paragraphe 3.4, basé lui, sur la difficulté extrême de factoriser de très grands nombres ;

— du point de vue de **l'implémentation proposée**, DSS est assez révélateur des types de problèmes pouvant intervenir lors de l'implémentation pratique de la cryptographie dans des applications civiles. La publication du DSS a en effet généré un nombre impressionnant de critiques, en raison notamment de l'influence visible de la NSA (National Security Agency) dans la conception fine de l'algorithme, laissant penser au plus grand nombre que des limitations sécuritaires y auraient été volontairement introduites, afin d'en permettre une cryptanalyse aisée par la NSA elle-même. La même suspicion existe d'ailleurs, depuis l'origine, concernant les critères de conception des *boîtes S* du DES, qui n'ont jamais été publiés. En ce sens, DSS a reçu un accueil plutôt très défavorable de l'industrie ; celle-ci avait en effet misé sur une standardisation officielle de l'algorithme RSA, par ailleurs très souvent implémenté comme technique d'authentification et de signature digitale dans les réseaux informatiques classiques (comme le prouve d'ailleurs la popularité grandissante du programme de chiffrement PGP décrit au paragraphe 6).

L'algorithme DSA utilise les paramètres suivants.

— p est un nombre premier choisi dans l'intervalle $]2^{L-1}, 2^L[$, où L est un nombre compris entre 512 et 1 024 et est un multiple de 64 ;

— q est un diviseur premier de $p-1$, satisfaisant les inégalités : $2^{159} < q < 2^{160}$;

— $g = h^{(p-1)/q}$ (modulo p), où h est un entier vérifiant $1 < h < p-1$, tel que $h^{(p-1)/q}$ (modulo p) > 1 ;

— x est un nombre choisi aléatoirement dans l'intervalle $]0, q[$;

— $y = g^x$ (modulo p) ;

— k est un entier choisi de matière aléatoire ou pseudo-aléatoire dans l'intervalle $]0, q[$.

Dans l'algorithme DSA, les entiers p, q et g sont publics et peuvent, le cas échéant, être partagés par un groupe d'utilisateurs. Chaque utilisateur possède une clef secrète x et une clef publique y construites suivant les procédures décrites (4 et 5 ci-dessus). Les données x et y de chaque utilisateur sont en principe valables durant une période de temps limitée.

Dans l'algorithme DSA, les paramètres x et k sont utilisés uniquement lors de la génération de signatures, et le nombre k doit être **régénéré à chaque signature**.

Le protocole de signature associé à l'algorithme DSA peut alors être décrit comme suit.

Génération de signature

Si l'utilisateur A souhaite envoyer le message M à l'utilisateur B, alors A calcule deux quantités entières r et s obtenues comme suit :

à partir des nombres p, q, g (publics) et de sa clef privée (x_A, k_A), A calcule successivement :

- $r = (g^{k_A}$ modulo $p)$ modulo q ;
- $s = (k^{-1}$ (SHA $(M) + x_A^r)$ modulo q.

Ici k^{-1} désigne l'inverse multiplicatif de k (modulo q), c'est-à-dire l'unique entier de l'intervalle $]0, q[$ tel que $k^{-1}\, k = 1$ (modulo q).

La valeur SHA (M) est le résultat de l'application de la fonction de hachage sécurisée définie dans la publication 180 (11 mai 1993) du National Institute of Standards and Technology. Pour un message donné, SHA (M) est une chaîne de 160 bits, qui est à son tour convertie en un entier suivant un processus standard.

Les nombres entiers r et s constituent la signature digitale associée au message M, et sont envoyés au destinataire avec le message M.

Vérification de la signature

Le destinataire B du message M et des entiers r et s reçoit ces quantités (par exemple *via* un réseau) sous la forme M', r', s'. Il dispose également de l'identité A de l'émetteur et de sa clef publique y_A. Pour vérifier la signature du message transmis, B effectue successivement les opérations suivantes :

— si l'une des deux inégalités $0 < r' < q$ et $0 < s' < q$ n'est pas satisfaite, alors la signature est invalidée ;

— si les deux inégalités ci-dessus sont satisfaites, alors B calcule les quantités suivantes :

$w = (s')^{-1}$ (modulo q),

$u_1 = ((\text{SHA}\ (M')) \cdot w)$ (modulo q),

$u_2 = ((r') \cdot w)$ (modulo q),

$v = ((g^{u_1})\ (y_A)^{u_2})$ (modulo p) (modulo q).

Si $v = r'$, alors la signature est vérifiée, et le vérificateur peut acquérir une quasi-certitude que le message adressé a bien été envoyé par le possesseur de la clef secrète y_A.

Si, au contraire, $v \neq r'$, alors les possibilités suivantes sont envisageables :

1) le message M a été modifié ;
2) le message M a été signé de manière incorrecte par l'émetteur ;
3) le message a été signé par un contrefacteur.

Bien entendu, dans chacun des cas 1, 2 et 3 ci-dessus, le message M doit être considéré comme invalide.

La preuve mathématique du fait que $v = r'$ lorsque $M = M'$, $r' = r$ et $s' = s$, peut être trouvée dans [30, appendice 1] et [21].

4.2 Spécification d'un algorithme de hachage sécurisé

L'utilisation de l'algorithme SHA (*Secure Hasch Algorithm*) de hachage sécurisé est requise pour toutes les applications fédérales et plus précisément pour toutes les opérations de signature digitale basées sur le standard DSS.

Étant donné un message de longueur inférieure à 2^{64} bits, l'algorithme SHA produit un compressé de 160 bits de ce message, appelé **condensé**. Ce condensé est utilisé à la fois dans l'opération de signature électronique du message et dans le processus de vérification de cette signature. L'algorithme est conçu de telle manière que toute altération du message durant sa transmission produira, avec une probabilité très forte, un condensé différent du condensé original lorsque le destinataire vérifiera la signature de l'expéditeur.

L'algorithme SHA tel que spécifié dans la publication FIPS Pub. 180 du National Institute of Standards and Technology [Specifications for the secure Hash Standard (SHS)] est basé sur le principe des fonctions à sens unique ou *one way* ; ces fonctions f sont caractérisées par la propriété suivante : étant donné une entrée M pour la fonction f, il est, du point du vue calculatoire, extrêmement facile de calculer $f(M)$. Par contre, étant donné une valeur C de l'image de f, il est infaisable, toujours du point de vue calculatoire, de trouver une valeur d'entrée M du domaine de f telle que $f(M) = C$, ou bien de trouver deux entrées M et M' telles que $f(M) = f(M')$.

Cela étant, l'algorithme de hachage sécurisé spécifié dans le standard SHS opère suivant les principes décrits ci-après.

4.2.1 Préliminaires

L'algorithme SHA opère sur des blocs de 512 bits. Un tel bloc de 512 bits peut être considéré comme une suite de 16 mots de 32 bits. Un mot de 32 bits peut, à son tour, être considéré comme une suite de 8 symboles hexadécimaux, c'est-à-dire appartenant à l'ensemble {0, 1, ..., 8, 9, A, B, ... F}, cela en convertissant chaque groupe de 4 bits du mot en son équivalent hexadécimal.

Exemple : le mot W = (00011101011001010111001001001 0011) est converti en la suite de 8 groupes de 4 bits (0001) (1101) (0110) (0111) (0010) (1001) (0011) qui donne, en final, la chaîne de 8 symboles hexadécimaux Y = 1D6572B3.

4.2.2 Opérations sur les entiers et sur les mots

Étant donné des mots binaires X et Y (par exemple de 32 bits), on définit les opérations suivantes opérant sur ces mots.

Opérations booléennes élémentaires :

- $X \wedge Y$ (**ET logique**) : par exemple, $(0011) \wedge (0001) = (0001)$;
- $X \vee Y$ (**OU logique**) : par exemple, $(0011) \vee (1010) = (1010)$;
- X XOR Y (**OU exclusif**) : par exemple (0101) XOR (0010) = (0111) ;
- $\tilde{X}$ (**complément logique**) : par exemple $(\widetilde{0110}) = (1001)$.

Somme de deux mots binaires :

si X et Y sont deux mots de n bits, ils représentent deux entiers x et y, chacun compris entre 0 et $2^n - 1$ (issus à partir de cette représentation binaire). On définit alors leur somme comme étant le mot de n bits Z représenté par le nombre entier $z = (x + y)$ modulo 2^n.

Décalage circulaire vers la gauche :

si X est un mot de n bits, on définit $S^m(X)$ $(0 \leqslant m \leqslant n)$ comme étant le mot de n bits défini par :

$$S^m(X) = (X \ll m) \vee (X \gg n - m)$$

où $X \ll m$ est le mot n bits obtenu en enlevant les m bits les plus à gauche de X et en ajoutant m zéros à droite au mot résultant,

$X \gg n - m$ est le mot obtenu en enlevant les $n - m$ bits les plus à droites de X et en rajoutant $n - m$ zéros à gauche au mot résultant.

Exemple : si X = (100011100001), on a $S^4(X)$ = (111000011000).

4.2.3 Description de l'algorithme SHA

Remplissage des messages

Tout message M peut toujours, après un codage adéquat, être considéré comme une suite de bits. En pratique, on peut même supposer que la longueur ℓ d'une telle suite est un entier strictement inférieur à 2^{64}.

La première étape de l'algorithme SHA est une étape de remplissage (*padding* en anglais) consistant à rajouter des bits au message M de telle sorte que la longueur du nouveau message (en bits) soit un multiple exact de 512.

- On ajoute d'abord un 1 à la droite du message. Soit M' le message obtenu en rajoutant le bit 1 à la fin de M.
- On calcule ensuite l'entier m le plus petit tel que $(1 + \ell + 64 + m)$ soit un multiple entier de 512.
- On ajoute alors m bits tous égaux à 0 à la fin de M'. Soit M'' le message résultant et PAD (M'') son expression en caractères hexadécimaux (§ 4.2.1).
- Pour terminer le remplissage, on rajoute à la fin de PAD (M'') les deux mots de 8 caractères hexadécimaux dérivés du mot de 64 bits codant la longueur initiale du message M. On notera N le message *rempli* résultant.

Exemple :

M = 01111000 11110000 01010101 11110101 01000110 11001101

On a $\ell = 48$ pour ce message

On déduit d'abord :

M' = 01111000 11110000 01010101 11110101 01000110 11001101 1

On constate que le plus petit entier m tel que $48 + 1 + m + 64$ soit un multiple de 512 est 399.

On en déduit l'expression de M'' :

M'' = 01111000 11110000 01010101 11110101 01000110 11001101 10.....0 (399 zéros)

On en déduit :

PAD(M'') = 78F055F5 46CD8000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Comme $48 = 32 + 16 = 2^5 + 2^4$, dont la représentation sur 64 bits est le mot :

(0000000000000000000000000000000000000000000000000000000000 00 110000), soit encore 00000000 00000030 en hexadécimal, on en déduit l'expression de N :

N = 78F055F5 46CD8000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000030

Description de l'algorithme

L'algorithme SHA utilise 80 fonctions logiques définies sur des mots de 32 bits et 80 constantes de 32 bits.

Les 80 fonctions logiques f_t, produisent en sortie des mots de 32 bits et prennent chacune en entrée trois mots de 32 bits, B, C, D. Leur expression est donnée par :

$f_t(B, C, D) = (B \wedge C) \vee (\tilde{B} \wedge D)$ pour $0 \leqslant t \leqslant 19$

$f_t(B, C, D) = B$ XOR C XOR D pour $20 \leqslant t \leqslant 39$

$f_t(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ pour $40 \leqslant t \leqslant 59$

$f_t(B, C, D) = B$ XOR C XOR D pour $60 \leqslant t \leqslant 79$

L'expression en hexadécimal des 80 constantes K_t utilisées par l'algorithme SHA est donnée par :

K_t = 5A827999 pour $0 \leqslant t \leqslant 19$

K_t = 6ED9EBA1 pour $20 \leqslant t \leqslant 39$

K_t = 8F1BBCDC pour $40 \leqslant t \leqslant 59$

K_t = CA62C1B6 pour $60 \leqslant t \leqslant 79$

Pour calculer le condensé d'un message M, l'algorithme SHA opère sur le message *rempli* N obtenu par la procédure décrite précédemment. Le message N peut donc être considéré comme une suite de n blocs de 512 bits, M_1, ..., M_n, où chaque M_i est lui-même considéré comme une suite de 16 mots de 64 bits.

Pour calculer le condensé du message M, l'algorithme SHA traite en séquence chacun des blocs M_i, le traitement de chacun de ces blocs demandant 80 étapes.

L'algorithme utilise deux mémoires tampons de 5 mots de 32 bits chacune et une suite de 80 mots de 32 bits. Traditionnellement, les mots de la première mémoire tampon sont notés A, B, C, D, E, et ceux de la deuxième H_0, H_1, H_2, H_3, H_4. Les 80 mots de la suite sont notés W_0, ... W_{79}. Enfin une dernière mémoire tampon *TEMP* de 32 bits est utilisée.

L'algorithme SHA est alors exécuté de la manière suivante.

- **Initialisation**

Le contenu des mots H_i est initialisé comme suit :

H_0 = 67452301

H_1 = EFCDAB89

H_2 = 98BADCFE

H_3 = 10325476

H_4 = C3D2E1F0

● **Traitement**

La procédure suivante est exécutée successivement pour M_1, ..., M_n.

— Division du bloc M_i en 16 mots de 32 bits, W_0, ..., W_{15}, où W_0 est constitué des 32 bits les plus à gauche de M_i.

— Pour $16 \leqslant t \leqslant 79$, soit $W_t = S^1 (W_{t-3}$ XOR W_{t-8} XOR W_{t-14} XOR $W_{t-16})$.

— Poser $A := H_0$, $B := H_1$, $C := H_2$, $D := H_3$, $E := H_4$.

— Pour $0 \leqslant t \leqslant 79$, exécuter la boucle :

$TEMP = S^5 (A) + f_t (B, C, D) + E + W_t + K_t$

$E := D$; $D := C$; $C := S^{30} (B)$; $B := A$; $A := TEMP$;

— Poser $H_0 := H_0 + A$, $H_1 := H_1 + B$; $H_2 := H_2 + C$; $H_3 := H_3 + D$; $H_4 := H_4 + E$.

À la fin du traitement de M_n, le condensé du message M est tout simplement la chaîne de 160 bits représentée par les 5 mots H_0, H_1, H_2, H_3, H_4.

Remarque : la procédure ci-dessus est particulièrement performante du point de vue du temps des calculs. En revanche, elle est assez gourmande du point de vue de l'espace mémoire demandé. On trouvera en [21] une procédure alternative de calcul du condensé, utilisant uniquement comme espace mémoire une liste circulaire représentée sous forme d'un unique tableau de 16 mots de 32 bits.

4.3 Standard PKCS

À l'inverse de DSS, PKCS (*Public Key Cryptography Standards*) est un standard de facto issu des travaux d'une compagnie américaine, RSA Data Security, en coopération avec le Massachusetts Institute of Technology et un consortium de compagnies informatiques incluant Apple, Microsoft, DEC, Lotus, Sun Microsystems. PKCS est compatible avec l'implémentation des standards de messagerie sécurisée d'Internet, PEM (*Privacy Enhanced Mail*), (§ 6). À la différence de PEM, PKCS peut prendre en compte non seulement des fichiers ASCII, mais également des fichiers binaires. Enfin PKCS est comptatible avec le standard de certification X.509 du CCIT, et fournit des spécifications d'implémentation d'encryption et d'authentification de messages et fichiers à partir du système RSA qui ne figurent pas dans le standard X.509.

PKCS comporte à la fois des standards reliés à des algorithmes spécifiques et des standards de présentation indépendants des algorithmes d'encryption choisis. Au niveau des algorithmes et protocoles spécifiques, PKCS supporte en particulier le DES en mode CBC, le système RSA, le protocole d'échange de clefs de Diffie-Hellmann et les algorithmes MD2, MD4 et MD5 de compression de messages développés par R. Rivest. Chacun de ces algorithmes est construit suivant le principe des fonctions *one way* [42], ce qui rend pratiquement impossible, en pratique, de trouver deux messages ayant même compressé ou un message ayant un compressé prédéterminé. Les syntaxes de signature et enveloppe digitale ainsi que les certificats d'authentification spécifiés dans PKCS sont, par contre, indépendants des algorithmes d'encryption choisis.

Du point de vue des fonctionnalités offertes, PKCS propose des méthodes de signature digitale, d'enveloppe digitale (c'est-à-dire le moyen de *sceller* électriquement un message de telle sorte que seul le destinataire du message puisse l'ouvrir électroniquement), de certification digitale, d'échange de clefs mutuellement agréées entre deux utilisateurs.

PKCS est décrit dans un certain nombre de documents de référence, nommés PKCS #1, #3, #5, #6, #7, #8, #9, #10. Les documents PKCS #6 à PCKCS #10 sont essentiellement des documents spécifiant des syntaxes standards pour différentes opérations liées à la cryptographie, aussi n'en parlerons-nous pas ici, renvoyant le lecteur intéressé à l'ensemble des références [36].

Les trois autres documents de PKCS sont ceux décrivant directement la mise en œuvre pratique d'algorithmes cryptographiques.

■ **PKCS #1** (RSA encryption standard). Ce premier document fournit les recommandations d'implémentation pratique de signature et d'enveloppe digitale d'un message en utilisant le système RSA. Pour la fonction signature digitale, le message M à signer est tout d'abord condensé sur 128 bits, en utilisant l'un des algorithmes de compression MD2, MD4 ou MD5. Ce condensé c(M) est ensuite encrypté avec la clef secrète RSA du signataire. L'ensemble formé du message M et de l'encrypté de son condensé constitue la signature digitale.

Pour la fonction d'enveloppe digitale, le message à expédier est d'abord encrypté en mode CBC par le DES, produisant le cryptogramme $E_K(M)$, et la clef DES utilisée est encryptée suivant $P_{RSA}(K)$ avec la clef publique RSA du destinataire. L'ensemble formé de $E_K(M)$ et $P_{RSA}(K)$ constitue l'enveloppe digitale du message M.

Concernant les modalités pratiques d'implémentation, PKCS #1 recommande que les modules publics de chaque utilisateur, du type $n = p \cdot q$ avec p et q premiers, aient au moins une longueur binaire de 328 bits, soit 41 octets. Si la longueur de n est définie comme l'entier k tel que $2^{8(k-1)} \leqslant n < 2^{8k}$, k doit avoir une longueur d'au moins 12 octets. Afin d'augmenter encore la sécurité en transmission, la donnée D à encrypter (il y a plus d'une donnée D seulement si le message M est très long) est formatée de la manière suivante : la chaîne d'octets représentant D est transformée en le bloc d'encryption EB formé de la chaîne d'octets (séparés par le symbole ||) :

$EB = 00 \| BT \| PS \| 00 \| D$ (en représentation hexadécimale)

Le premier octet (00) est destiné à s'assurer que la donnée à encrypter est inférieure au module n de l'utilisateur.

Le deuxième octet, BT, indique la structure du bloc d'encryption et a la valeur 00 ou 01 pour des opérations d'encryption par clef secrète, 02 pour des opérations d'encryption par clef publique.

PS est un groupe de $k - 3 - |D|$ octets ; où $|D|$ désigne le nombre d'octets de D. Ces octets de *remplissage* sont mis à 00 pour des blocs de type $BT = 00$, à FF pour des blocs de type $BT = 01$ et générés pseudo-aléatoirement pour le type $BT = 02$.

Pour une justification précise de ces modifications, le lecteur intéressé pourra se reporter aux références [36].

Avant encryption, le bloc EB est alors transformé en l'entier x défini par :

$$x = \sum_{i=1}^{b} 2^{8(k-i)} EB_i$$

où EB_1, ..., EB_k représentent la suite d'octets de EB, EB_1 étant l'octet de poids fort de EB, EB_k l'octet de poids faible.

Remarque : comme $EB_1 = 00$, on a bien $0 \leqslant x < n$, puisque $2^{8(k-1)} \leqslant n$.

L'entier x est alors encrypté suivant la méthode RSA standard, produisant le cryptogramme $y = x^c$ (modulo n), avec $0 \leqslant y < n$.

Le destinataire effectue l'opération de décryptage avec la clef appropriée et produit alors le décrypté $x' = y^e$ (modulo n) (c et e représentent les clefs publiques et secrètes du destinataire). Il effectue alors une conversion inverse, entier ⇒ chaîne d'octets, suivant le calcul $x' = \sum_{i=1}^{k} 2^{8(k-i)} ED_i$. Il vérifie alors que la structure des octets produits par ce procédé est bien du type EB ci-dessus.

Bien entendu, le destinataire récusera le message (signature ou enveloppe) si :

— le premier octet n'est pas du type 00 ;
— le contenu de BT n'est pas de l'un des types 00, 01 à 02 ;
— la structure des octets PS n'est pas compatible avec le type BT ;
— le processus de décryption étant obtenu par clef publique, le type BT n'est pas 00 ou 01 ;

— le processus de décryption étant obtenu par clef secrète, le type *BT* n'est pas 02.

■ **PKCS #3** (Diffie-Hellmann key-agreement standard). Ce standard décrit un protocole permettant à deux utilisateurs de pouvoir partager une clef commune d'encryption, de manière rapide et sûre. Le principe utilisé est basé sur le problème du *logarithme discret*, tel qu'exposé pour le standard DSS.

Pour la mise en œuvre de ce protocole, l'autorité centrale gérant le réseau génère un très grand nombre premier p et α une racine primitive modulo p, c'est-à-dire un nombre compris entre 1 et $(p-1)$, tel que tous les nombres de la forme α^m, avec m décrivant l'intervalle entier $[1, p-1]$, soient distincts (en d'autres termes, α est un générateur du groupe cyclique des éléments inversibles de l'anneau $\mathbb{Z}/p\mathbb{Z}$).

p et α sont alors rendus publics et constituent la base du protocole d'échange de clefs mutuellement agréées de PKCS.

Pour échanger une clef secrète entre eux, deux utilisateurs A et B procèdent alors de la manière suivante.

- A choisit un nombre X_A au hasard dans l'ensemble $]0, p-1[$.

Il garde ce nombre secret et envoie à B la quantité $Y_A = \alpha^{X_A}$ (modulo p).

- De la même façon, B choisit un nombre X_B au hasard dans l'ensemble $]0, p-1[$, garde ce nombre secret et envoie à A le nombre $Y_B = \alpha^{X_B}$ (modulo p).
- B calcule $K_{AB} = (Y_A)^{X_B} = (\alpha^{X_A})^{X_B} = \alpha^{X_A X_B}$ (modulo p).
- A calcule $K_{BA} = (Y_B)^{X_A} = (\alpha^{X_B})^{X_A} = \alpha^{X_A X_B}$ (modulo p).

Comme $K_{AB} = K_{BA}$ de manière évidente, K_{AB} est alors choisi comme clef d'encryption commune entre A et B. Les quantités X_A et X_B étant gardées secrètes, toute tierce partie cherchant à reconstituer K_{AB} à partir de Y_A ou Y_B n'a d'autre solution que de chercher à résoudre l'un des deux problèmes de *logarithmes discrets* suivants : $Y_A = \alpha^x$ (modulo p) ou $Y_B = \alpha^x$ (modulo p), ce qui, en pratique, s'avère infaisable calculatoirement, dès lors que p est un nombre premier dont l'écriture en base 2 est de l'ordre de 1 000 bits.

L'implémentation pratique de l'algorithme d'échange de Diffie-Hellmann dans le standard PKCS s'effectue suivant des mécanismes de transformation entier ⇒ chaîne d'octets et chaîne d'octets ⇒ entier exactement similaires à ceux décrits dans le paragraphe précédent.

■ **PKCS #5** (password-based encryption standard). Ce standard précise la méthode à utiliser pour encrypter une chaîne d'octets avec un algorithme symétrique, dont la clef est directement dérivée d'un mot de passe. Bien que ce standard puisse *a priori* être utilisé pour un message arbitraire, il est destiné principalement à l'encryption de clefs secrètes RSA devant être transférées sur un réseau (par exemple, lors d'un changement de machine d'un utilisateur ou d'un changement de lieu de travail de celui-ci). PKCS #5 décrit deux algorithmes d'encryption de clefs : MD2 avec DES en mode CBC et MD5 avec DEC en mode CBC (qui, sur le principe, sont fondamentalement identiques).

Chaque utilisateur dispose ici d'une quantité secrète P, s'exprimant en un nombre arbitraire d'octets, son **mot de passe**. Il choisit en plus une souche S sur 8 octets et un entier c. Chacune des quantités P, S, c peut varier à chaque encryption.

Chaque encryption/décryption s'effectue de la manière suivante.

- **Génération de la clef DES**

— La chaîne d'octets concaténée $P \| S$ est condensée, en utilisant c itérations de l'algorithme de condensation choisi (MD2 ou MD5). On calcule donc la quantité : $KC = \mathrm{MD}x \circ \mathrm{MD}x \circ \ldots \circ \mathrm{MD}x\ (P \| S)$ (c fois). Par exemple, si $x = 5$ et $c = 2$, $KC = \mathrm{MD5}\ (\mathrm{MD5}\ (P \| S))$. Le résultat précédent, sur 128 bits, est partagé en 16 octets, dont on sélectionne les 8 premiers.

— Le premier bit de chacun des 8 octets conservés est ajusté de sorte que chaque octet ait une parité impaire. Les 8 octets ainsi modifiés constituent la clef DES_K utilisée.

— Les 8 octets du condensé KC constituent alors la souche d'initialisation IV du DES pour le mode CBC (cf. (§ 3.3.2.2)).

- **Formatage en mode encryption-bloc**

Le message M à encrypter est augmenté d'une certaine chaîne PAD d'octets de *remplissage*, pour former le bloc $EB = M \| \mathrm{PAD}$. PAD est obtenue de la manière suivante :

soit $|M|$ le nombre d'octets de M. Alors, PAD consiste en $(8 - (|M| \text{ modulo } 8))$ octets de valeur hexadécimale $(8 - |M|)$.

Exemple :

si $|M| = 7$ modulo 8, PAD = 01 et $EB = M \| 01$

si $|M| = 0$ modulo 8, PAD = 08 08 08 08 08 08 08 08 et $EB = M \| 08\ 08\ 08\ 08\ 08\ 08\ 08\ 08$

- **Encryption**

Le bloc EB (dont la longueur en octets est toujours un multiple de 8) est alors encrypté par le DES utilisé en mode CBC avec la souche IV et la clef K, produisant le cryptogramme $C = \mathrm{DES}_K(EB, \mathrm{IV})$.

En mode décryption, le même utilisateur, sur la machine d'arrivée, regénérera la clef K et la souche IV à partir des mêmes données P, S et c. Il vérifie d'abord que le cryptogramme C, reçu sur la machine d'arrivée, a bien une longueur en octets multiple de 8. Il calcule ensuite, à l'aide de K et IV, $EB' = \mathrm{DES}_K^{-1}(C) \cdot EB'$ se présente alors sous la forme d'un message du type $M' \| \mathrm{PAD}'$. Bien entendu si PAD' n'est pas du type prévu, l'utilisateur récusera le transfert de clefs et en déduira qu'un incident ou une malveillance est intervenu lors du changement de machine.

4.4 DSS contre PKCS

La comparaison entre les deux techniques de signature digitale doit, très sommairement, être exprimée à la fois en termes techniques et en termes économiques ou juridiques.

■ Sur le plan technique, et malgré quelques aménagements apportés par le NIST par rapport à la version initiale de l'algorithme (notamment vis-à-vis de la longueur des clefs), DSS reste l'objet d'un certain nombre d'interrogations fondamentales de la part des spécialistes :

— le fait que plusieurs utilisateurs puissent partager un module commun (paramètres p, g et q de l'algorithme) reste considéré comme un risque important sur le plan de la sécurité. La littérature récente regorge de méthodes d'attaques efficaces de tels schémas à *secrets partagés* et bon nombre de cryptologues considèrent que l'étude théorique de la sécurité de DSS est loin d'être terminée ;

— même s'il n'existe pas d'algorithme efficace connu pour résoudre le problème du *logarithme discret*, certains auteurs [30] ont récemment observé que, pour certains types de nombres premiers, le problème du *logarithme discret* était sensiblement plus facile que dans le cas général. Les possibilités d'attaque cryptographique du schéma DSS le plus général liées à l'existence de tels nombres premiers n'ont pas encore été analysées ;

— beaucoup plus récent que PKCS, le DSS n'a pu être mis autant à l'épreuve de la cryptologie théorique et pratique que son concurrent. De ce fait, il est considéré par beaucoup comme insuffisamment testé vis-à-vis des attaques potentielles ;

— la génération d'une clef à chaque signature (paramètre k de l'algorithme) pose un problème de sécurité essentiel. Il est en effet possible de démontrer que la corruption d'une seule de ces clefs k peut compromettre la totalité des clefs secrètes de l'utilisateur (ou du groupe d'utilisateurs). La question d'une gestion efficace des paramètres de signature k est donc particulièrement cruciale ; même si des propositions techniques ont été faites en ce sens, elles n'ont aucun caractère contraignant, ouvrant donc la porte à des choix malencontreux de la part des utilisateurs, susceptibles de compromettre gravement toute la sécurité du système ;

— enfin, et c'est un point essentiel, DSS ne traite absolument pas de distribution des clefs ni de l'encryption des messages proprement dite, ce qui, en toute rigueur, ne peut lui conférer que le statut d'un *demi-standard* selon R. Rivest.

Pour l'algorithme RSA, beaucoup mieux connu, les réserves techniques concernent essentiellement la vitesse avec laquelle les algorithmes de factorisation, jointe à l'accélération de la puissance de traitement des ordinateurs, déplacent la taille des nombres à factoriser. La factorisation récente de RSA-129 (§ 3), contrairement aux prédictions de R. Rivest, illustre assez bien le phénomène.

■ Sur le plan économique, la standardisation de DSS est intervenue au moment où pratiquement toute l'industrie américaine s'était ralliée au standard de fait RSA (Apple, AT&T, IBM, Lotus, Microsoft, SUN...) posant, de fait, d'importants problèmes de compatibilité de matériel et de logiciel avec les pratiques industrielles communes (par exemple les systèmes PEM ou PGP largement utilisés aujourd'hui sur Internet).

Enfin, et ce n'est pas forcément le moindre des problèmes, DSS reste entaché du manque de transparence du processus ayant donné lieu à sa naissance (pas d'appel d'offres du NIST, *pilotage* en sous-main par la NSA) et du manque de certitude juridique quant à la validité des brevets protégeant l'algorithme. Pour ces questions le lecteur intéressé est renvoyé à [30] et [31].

Nota : pour un traitement beaucoup plus complet de cette question, le lecteur intéressé trouvera une analyse assez développée dans [28] ainsi que dans les compte rendus récents des conférences *Crypto* et *Eurocrypt*.

5. Compléments sur les protocoles cryptographiques

5.1 Protocoles *Zero Knowledge*

Dans de nombreuses applications faisant intervenir la cryptographie, on se trouve confronté à des situations où un individu est censé prouver à un centre de services (un guichet bancaire, un système de point de vente électronique...), de manière interactive, qu'il est bien qu'il prétend être et qu'il possède bien le droit de solliciter le service en question (ce qu'on appelle communément son *accréditation*). La résolution pratique de ce problème se heurte à trois types de contraintes partiellement antagonistes :

— faire en sorte que l'information échangée entre l'individu et le centre de services effectuant la vérification ou entre le centre de service et l'individu fournisse le moins possible d'informations susceptibles, en cas d'interception, de corrompre la sécurité des éléments du système (individu et centre de services) ;

— faire en sorte que le centre de services acquière un niveau de certitude quasi absolu quant à l'identité du demandeur et à ses droits pour demander accès à ces services ;

— faire en sorte que le protocole d'échanges d'informations entre individu et centre de services permette le traitement de demandes en temps réel.

La recherche de protocoles cryptographiques satisfaisant aux contraintes précédentes a fait l'objet de recherches très actives, tant sur le plan théorique que sur le plan pratique, depuis 1984, aboutissant au concept de protocole à *connaissance nulle*.

Dans un protocole interactif à un prouveur (P, V), un prouveur P et un vérifieur V partagent une donnée d'entrée x commune, disposent chacun de deux sources privées et non biaisées de nombres aléatoires (r_P et r_V respectivement), ainsi que d'un canal de transmission d'information. Sur la base de l'entrée x, la traduction du protocole est une suite d'informations $(\alpha_1, \beta_2, ..., \alpha_{2t-1}, \beta_{2t})$, où $t = t(n)$ est une fonction à croissance bornée par un polynôme en $n = |x|$ (nombre de bits de x). De même, il existe un polynôme q tel que chaque $|\alpha_{2i-1}|$ et chaque $|\beta_{2i}|$ soient bornés par $q(n)$, pour $1 \leqslant i \leqslant t$. Chaque α_{2i-1} est un message du vérifieur vers le prouveur, et est une fonction calculable en temps polynomial de x, r_V et $(\alpha_1, \beta_2, ..., \alpha_{2i-3}, \beta_{2i-2})$. De même, chaque β_{2i} est un message du vérifieur vers le prouveur, et est une fonction calculable en temps polynomial de x, r_P, $(\alpha_1, \beta_2, ..., \beta_{2i-2}, \alpha_{2i-1})$. Après calcul de β_{2t}, le vérifieur calcule en temps polynomial une fonction de x, r_V et de la suite des α_{2i-1} et des β_{2i}. Le résultat de ce calcul est alors ACCEPT si le vérifieur considère que le prouveur a bien réussi à certifier son identité, REJECT dans le cas contraire.

Dans un protocole interactif à connaissance nulle (*Zero-Knowledge Interactive Protocol* ou ZKIP dans la littérature anglosaxonne), le prouveur P cherche à convaincre (en temps polynomial) le vérifieur V de la validité d'un théorème (son *accréditation*), sans rien (ou presque) révéler sur le théorème lui-même.

Nota : le lecteur intéressé par l'aspect théorique de ces questions pourra utilement se reporter à la référence [32].

D'un point de vue pratique, les ZKIP sont basés sur des problèmes considérés comme calculatoirement difficiles, tels le problème du calcul du *logarithme discret* évoqué plus haut, le problème de la factorisation des très grands nombres, l'extraction de racines carrées modulo un très grand nombre ou encore le problème des résidus quadratiques.

Donnons deux exemples pratiques de tels schémas.

5.2 Protocole de Fiat-Shamir

Cet exemple, dérivé de Feige, Fiat et Shamir [33], est conçu pour permettre à une communauté d'utilisateurs de s'identifier chacun vis-à-vis des autres sans divulgation d'informations confidentielles relatives à ces utilisateurs.

Dans ce système, chaque utilisateur dispose d'une paire de clefs, I et S.

L'information I est publique et peut être considérée comme l'identité de l'utilisateur ; S est sa clef secrète. Le but du système de preuve d'identification est qu'un utilisateur i d'identité I parvienne à convaincre un autre utilisateur du système qu'il *connaît* sa clef secrète S, sans rien révéler d'autre sur l'information S que le simple fait qu'il la connaît effectivement. L'efficacité pratique du système est basée sur l'hypothèse (vraisemblable !) qu'il est pratiquement infaisable calculatoirement de calculer des racines carrées modulo un très grand nombre composé, sans en connaître la factorisation.

La mise en pratique d'un tel système s'effectue de la manière suivante.

■ Une autorité de confiance génère deux très grands nombres premiers p et q ; tous deux congruents à 3 modulo 4. Le produit $m = p \cdot q$ de ces deux nombres est publié, mais bien entendu p et q sont gardés secrets.

Comme p et q sont tous deux du type $4k + 3$, un résultat mathématique classique assure que (-1) n'est pas un résidu quadratique modulo m, c'est-à-dire que l'équation $a^2 = -1$ modulo m n'a pas de solution dans l'intervalle $[0, m-1]$. Dans toute la suite, on notera $\mathcal{Z}_m^*$ (1) l'ensemble des entiers compris entre 1 et m, premiers avec m et dont le symbole de Jacobi par rapport à m vaut 1.

Définition : si p est un nombre premier > 2, et $a \in]0, p[$, on note (a/p) le symbole de Legendre de a par rapport à p, à savoir + 1 si a est un résidu quadratique modulo p, (– 1) si a n'est pas un résidu quadratique modulo p. Si $m = p \cdot q$ est le produit de deux nombres premiers et si $a \in]0, m[$, alors le symbole de Jacobi de a par rapport à m est défini par $(a/m) = (a/p)\,(a/q)$.

■ Chaque utilisateur choisit alors au hasard t_1 nombres $S_1, ..., S_{t_1}$ dans $\mathcal{Z}_m^*$ (1) et t_1 bits $(b_1, ..., b_{t_1})$. Il calcule alors les t_1 quantités $I_j = (-1)^{b_j} / S_j^2$ modulo m, pour $1 \leqslant j \leqslant t_1$. Il définit alors son identité I_A comme la collection des I_j, soit $I_A = (I_1, ..., I_{t_1})$ et sa clef secrète S_A comme la collection des S_j, soit $S_A = (S_1, ..., S_{t_1})$.

■ Pour prouver son identité vis-à-vis de l'utilisateur B, l'utilisateur A d'identité I_A et de clef secrète S_A fait appel au protocole suivant, répété t_2 fois :

— A choisit un nombre au hasard R dans $\mathcal{Z}_m^*$ (1), un bit aléatoire c, et envoie $X = (-1)^c R^2$ (modulo m) à B ;

— B choisit au hasard un vecteur $(E_1, ..., E_k)$ de digits binaires et l'envoie à A ;

— A renvoie $Y = R \prod_{E_j = 1} S_j$ (modulo m) à B ;

— B vérifie que $Y^2 \prod_j E_j I_j$ (modulo m) est égal à $\pm X$.

B considère alors que A s'est correctement identifié auprès de lui, si l'étape précédente du protocole s'est déroulée correctement pour chacun des t_2 essais de vérification.

Ce protocole est effectivement efficace parce que, si la probabilité qu'un imposteur (ignorant S_A) puisse tromper B en choisissant un Y et en faisant le choix d'un vecteur E^* vraisemblable de telle sorte que les deux quantités $Y^2 \prod_j E_j I_j$ et $Y^2 \prod_j E_j^* I_j$ soient égales modulo m est de l'ordre de 2^{-k}, la probabilité pour que cette opération frauduleuse soit réussie t_2 fois n'est plus que de 2^{-kt_2}. En pratique, le système devient sûr dès que le couple (k, t_2) est voisin de (6,5) ou (9,8). L'intérêt de ce système d'identification est la très faible puissance de calcul qu'il demande, qui le rend pratiquement implémentable dans une carte à mémoire. Au niveau théorique, on peut démontrer [33] que ce schéma fournit un schéma de protocole à connaissance nulle dès que t_1 est très grand devant ln (ln m) et t_2 est de l'ordre de ln m.

5.3 Protocole de Quillou-Quisquater

Ce protocole [34] est particulièrement adapté à un implémentation dans des systèmes à base de cartes à mémoire, en raison du volume de stockage et du nombre d'échanges absolument réduits au minimum : ce protocole ne demande en effet qu'une accréditation (donnée permettant d'authentifier une valeur et de certifier une identité) et une interaction de la carte avec le monde extérieur.

Par rapport au protocole précédent, il demande environ trois fois plus de temps de calcul mais reste très acceptable en pratique.

Dans ce protocole, et aux fins d'identification, l'émetteur de cartes à mémoires choisit 2 très grands nombres premiers p et q (d'environ 256 bits chacun) et forme le produit $n = p\,q$. Cet entier n est une constante publique, également connue du vérifieur (point de vente, guichet bancaire...). Bien entendu, les nombres premiers sont gardés secrets et connus uniquement de l'organisme émetteur de cartes. En plus de n, un exposant v est également publié et porté à la connaissance du vérifieur. La taille de cet exposant est en général choisie de l'ordre de 30 bits, afin de représenter un compromis acceptable entre sécurité et rapidité.

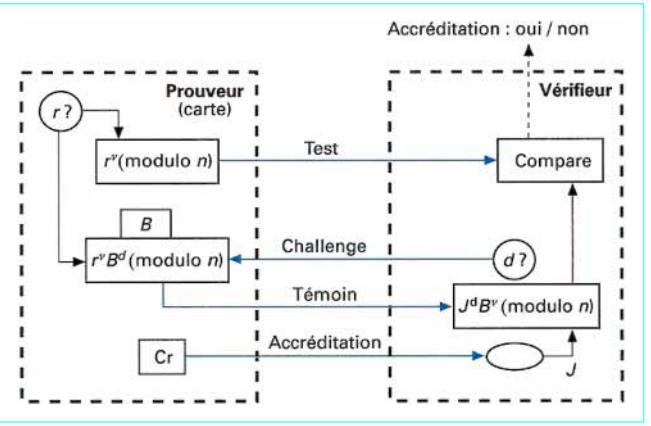


Figure 11 – Description du protocole de Quillou / Quisquater

À chaque carte à mémoire est associée un identificateur I, permettant, par des procédés publics, de construire un identifiant J du même ordre de grandeur que n. Un nombre secret B, servant d'authentifiant, est alors inscrit dans la carte au moment de sa fabrication. Ce nombre B est choisi de manière à satisfaire l'équation arithmétique $J\,B^v \equiv 1$ (modulo n).

Le protocole défini par Quillou et Quisquater permettant de prouver l'accréditation d'une carte est alors décrit par les étapes suivantes symbolisées figure **11**.

■ Le processeur de la carte à mémoire choisit un nombre au hasard r dans l'intervalle $]0, n-1[$ et calcule la donnée test $T \equiv r^v$ (modulo n). La carte transmet T et son identité I au vérifieur.

■ Le vérifieur choisit au hasard un entier d entre 0 et $v - 1$, appelé le *challenge* et l'envoie au prouveur (la carte !).

■ Le processeur de la carte calcule le produit $t = r\,B^d$ (modulo n). Ce nombre appelé ***témoin*** est ensuite transmis par la carte au vérifieur.

■ Pour vérifier la validité de t, le vérifieur calcul $c = J^d\,t^v$ (modulo n) et compare cette quantité avec le nombre test T reçu en b. Si $c = t$, le vérifieur considère que le prouveur a correctement prouvé son accréditation ; dans le cas inverse, le vérifieur récuse l'accréditation. Le fait que $c = t$ résulte de la relation mathématique évidente :

$$J^d t^v \equiv J^d\,(r\,B^d)^v \equiv (JB^v)^d r^v \equiv r^v = T \text{ (modulo } n)$$

Les trois remarques ci-après sont cruciales pour expliquer les conditions imposées au protocole, et pourquoi le vérifieur n'apprend en fait rien sur la donnée secrète d'accréditation B de la carte.

■ Si un imposteur pouvait deviner le challenge d, alors il pourrait facilement en déduire une stratégie d'identification frauduleuse. Il est en effet facile, si l'on peut anticiper la valeur de d avant envoi du nombre test, de générer un nombre test T^* passant le test $T^* = t$ sans aucune connaissance de B.

■ En cas de litige, tout organisme judiciaire d'arbitrage est incapable de faire la distinction entre une suite de données inscrites correspondant à une suite réussie de vérifications, d'une suite de contrefaçons où les challenges auraient été demandés avant de fixer les nombres témoins.

■ La connaissance de deux réponses t_1 et t_2 à deux challenges d_1 et d_2 associés au même nombre test T est équivalente à la connaissance de la puissance B^k de l'accréditation B, où k est le PGCD de v et $d_2 - d_1$.

6. Sécurité dans les réseaux : nouveaux produits

6.1 PGP *Pretty Good Privacy*

6.1.1 Description de PGP

PGP [20] est un logiciel de chiffrement destiné à préserver en priorité la confidentialité des messages éléctroniques et les fichiers stockés dans les mémoires de masse des ordinateurs ou stations de travail personnels. PGP est en fait un système de chiffrement complet, offrant les fonctionnalités suivantes.

Signature électronique et vérification d'intégrité de messages

PGP offre une fonction de signature digitale et d'authentification basée sur l'emploi simultané d'une fonction de hachage et le système RSA. La fonction de hachage utilisée dans PGP est la fonction *one way* MD5 de R. Rivest qui, étant donné un message quelconque en produit un condensé sur 128 bits. Comme 2^{128} n'est autre que le nombre 340282366920938463463374607431768211456, qui est plus grand que le nombre total de documents que l'humanité a produit depuis l'origine et produira pour les quelques millénaires à venir, la probabilité que deux documents différents produisent le même condensé (collision) est extrêmement faible. Par contre, deux documents identiques produiront toujours le même condensé. Pour signer et authentifier un message M émis par A, PGP procède comme suit.

Il calcule d'abord le condensé MD5(M) du message et encrypte les 128 bits résultants avec la clef secrète de A. Le récepteur du message, à partir de la clef publique de A, calculera une quantité MD5'(M). Il calculera ensuite, à partir du message décrypté M', le condensé correspondant MD5(M'). Si MD5(M') = MD5(M), alors il en déduira que $M' = M$ et que c'est bien A qui lui a envoyé le message puisque :

— seul A connaît sa clef secrète et est donc capable de générer le message E_{S_A} (MD5(M)) où E_{S_A} est l'encryption RSA avec la clef secrète de A ;

— la probabilité de collision entre M et M' étant quasi nulle, le fait que MD5(M) = MD5(M') prouve que le message M n'a pas été altéré au cours de sa transmission.

Compression de message

PGP offre en standard une fonction de compression de message, dont l'algorithme sous-jacent est identique au programme ZIP 2.0, familier de pratiquement tous les utilisateurs MS-DOS et Unix, et depuis longtemps dans le domaine public. ZIP 2.0 a un taux de compression voisin de 2, c'est-à-dire réduit en moyenne par 2 le volume d'octets correspondant au message initial. À noter que PGP utilise toujours la compression *avant* encryption et *après* signature.

Chiffrement de fichiers

Pour ce faire, PGP utilise un algorithme de type DES, appelé IDEA avec une clef de chiffrement sur 128 bits. IDEA est un algorithme assez récent, inventé par des chercheurs de l'ETH de Zurich, Lai Xuejia et J.L. Massey et présenté pour la première fois lors du congrès *Eurocrypt'90* [35]. Comme le DES, IDEA travaille sur des blocs de 64 bits. Cet algorithme n'a, à ce jour, révélé aucune faiblesse du point de vue de la cryptanalyse, et est présenté brièvement ci-dessous.

Dans PGP, IDEA est utilisé en mode CBC. À chaque nouvelle encryption, PGP génère une nouvelle clef IDEA sur 128 bits ainsi qu'une souche d'initialisation sur 64 bits pour le mode CBC. La clef d'encryption IDEA et la souche utilisée sont ensuite encryptées avec la clef publique RSA du destinataire, et le cryptogramme correspondant est ajouté en en-tête du message chiffré sous IDEA, l'ensemble étant adressé au destinataire.

Dans le cas d'un message devant être signé, PGP encrypte le compressé par ZIP du message à transmettre augmenté de son condensé par MD5, et transmet la clef IDEA encryptée sous la clef publique du destinataire de la même façon que dans le cas de messages non signés. Ainsi, sous PGP, il est impossible de vérifier la signature d'un message encrypté sans déchiffrement préalable.

Le fait qu'une clef différente IDEA soit générée pour chaque transmission fait qu'un même message sera en général chiffré différemment avec une même clef publique, ce qui complique encore le processus de cryptanalyse. Une option très intéressante de PGP est également la possibilité d'expédier un message chiffré à plusieurs destinataires.

Génération de clefs publiques et privées

Chaque utilisateur chiffre ses messages à l'aide de clefs privées IDEA qui sont communiquées à ses correspondants suivant les techniques décrites précédemment. Le transfert de clefs électroniques IDEA utilise le système RSA, aussi PGP offre-t-il donc des mécanismes de génération de clefs adaptés à ce système. La taille des clefs RSA est proposée suivant trois niveaux de sécurité : 512, 768 ou 1 024 bits. Cette génération est effectuée de la manière suivante.

PGP demande d'abord à l'utilisateur un identifieur (par exemple, son nom ou son adresse électronique...) puis, ensuite, un mot de passe qui devra être mémorisé pour toujours et jamais révélé à toute tierce personne ou organisation. À partir de ces données, PGP demande à l'utilisateur de saisir des caractères au hasard sur son clavier. Le temps de saisie et le mot de passe sont alors utilisés pour créer la *souche* du générateur aléatoire qui créera les deux grands nombres premiers p et q dont le produit constituera le nombre n associé à l'utilisateur. Un nombre E tiré au hasard dans l'intervalle [2, $(p-1)(q-1)-1$] étant choisi, la clef publique de l'utilisateur est alors constituée du couple (n, E), la clef secrète de l'utilisateur étant alors formée des deux nombres (D, $\varphi(n)$), où $\varphi(n) = (p-1)(q-1)$ et $DE \equiv 1$ (modulo $\varphi(n)$).

Ces deux clefs sont stockées dans deux fichiers : *secring.pgp* pour la clef secrète, *pubring.pgp* pour la clef publique. En complément, la souche du générateur utilisé est stockée dans un troisième fichier *randseed.bin*. Bien entendu, le fichier *secring.pgp* ne contient qu'une version binaire encryptée (sous IDEA) de la clef secrète de l'utilisateur. La clef IDEA correspondante est générée à partir du mot de passe fourni et n'est jamais stockée en clair dans le système, toujours recalculée à la volée, sur fourniture du mot de passe.

En fait, les fichiers *pubring.pgp* et *secring.pgp* peuvent contenir les clefs de plusieurs utilisateurs ou plusieurs types de clefs (par exemple, une par application) d'un même utilisateur. On trouve ici la notion de *trousseau* éléctronique.

Gestion des clefs

PGP permet l'extraction d'une clef publique du fichier *pubring.pgp* afin d'en assurer la distribution aux correspondants habituels de l'utilisateur qui souhaiteraient lui envoyer des messages chiffrés. Cette extraction peut se faire soit en mode binaire, soit en mode encapsulation ASCII, qui est en général mieux adapté à la messagerie électronique. Bien entendu, des facilités de modification d'attributs (changement d'identificateur, de mots de passe, suppression...) sont aussi fournies par le système.

Certification de clefs

PGP offre la possibilité d'ajouter un sceau numérique garantissant l'authenticité des clefs publiques. Il s'agit ici d'une mesure protectrice supplémentaire, destinée à éviter la production dans le réseau de clefs valides mais non authentifiées. Lors d'une opération d'encryption de message, par exemple, PGP doit récupérer la clef publique du destinataire, et donc s'assurer de la validité de cette clef. À cet effet, PGP utilise un certificat digital associé à chaque clef publique, sorte de *signature* donnant à la fois des informations sur la validité de la clef et le niveau de confiance qui lui est associé. Ce dernier point constitue l'une des grandes originalités de PGP, en basant le concept de ***confiance*** sur la notion de ***proximité sociale*** plutôt que sur celle d'***autorité centrale de certification***. En clair, il n'y a pas d'autorité absolue certifiant les clefs de tous les utilisateurs PGP. Chaque utilisateur peut signer des clefs publiques et construire

lui-même de manière incrémentale son réseau de certification, en accordant d'autant plus de confiance aux clefs reçues qu'elles ont été signées par des individus ou organismes en qui il a lui-même pleine confiance.

Révocation, désactivation, enregistrement notarial de clefs

En cas de perte ou de vol du fichier *secring.pgp* et de son mot de passe associé, l'utilisateur de PGP peut produire des certificats de révocation, invalidant les clefs publiques correspondantes. Cette opération est définitive, sans possibilité de retour en arrière. Signalons que, de manière tout à fait classique, une entreprise ou un individu peuvent décider d'effectuer un dépôt légal sous enveloppe scellée auprès d'un notaire ou de toute autorité acceptée des différents utilisateurs, des éléments suivants :

— une disquette contenant le fichier *secring.pgp*, enregistré et protégé par sa clef IDEA ainsi que le fichier *pubring.pgp* correspondant ;

— une trace en clair du mot de passe permettant de régénérer la clef IDEA.

Par la suite, en cas de démission, licenciement, voire mort d'un employé de la société, celle-ci sera en mesure, par cette procédure, de récupérer les informations confidentielles manipulées par l'employé en question.

6.1.2 Distribution de PGP

PGP est aujourd'hui distribué sur une multitude de sites Internet. Les versions compressées pour machine Unix et MS-DOS sont en général directement récupérables par *ftp* anonyme (protocole de transfert de fichiers, permettant à un utilisateur de transférer un fichier distant sur son ordinateur). À noter cependant que certains sites, tel celui du MIT, refusent la distribution à des utilisateurs ne possédant pas la nationalité américaine et ne travaillant pas aux États-Unis. L'exportation de PGP en dehors des États-Unis est en effet interdite par la législation américaine, mais un mouvement violent de contre-réaction à cette interdiction des *cypherpunks* d'Internet fait qu'il est possible aujourd'hui de se procurer le logiciel à peu près sur n'importe quel site universitaire. En ce sens, PGP est un exemple représentatif de ce que l'on pourrait appeler le *guerilla freeware*.

Il existe deux grands types de versions de PGP utilisées : celles utilisant la boîte à outils RSAREF de la société américaine RSADSI et, *a priori*, protégée par un brevet (version 2.6 et suivantes), et celles s'affranchissant de cette restriction (versions 2.6.ui et suivantes, ui signifiant *unofficial international*), principalement utilisées en Europe. À noter que, en France, l'utilisation d'un logiciel de cryptage tel PGP est *a priori* interdite (les autorisations éventuelles sont à demander au SCSSI (Service Central de la Sécurité des Systèmes d'Information).

Pour finir cette brève présentation du sytème PGP, signalons que son inventeur, P. Zimmermann, fervent défenseur de la cryptographie privée et opposant à la législation américaine en cours, fait actuellement l'objet d'une procédure judiciaire pour violation de la législation sur les exportations de matériels et logiciels cryptographiques.

6.1.3 Présentation de l'algorithme IDEA

IDEA est un système de chiffrement par blocs, opérant sur des mots de 64 bits et utilisant une clef à 128 bits. L'algorithme à la base d'IDEA est utilisé à la fois en mode encryption et en mode décryption. Sur le principe, IDEA est tout à fait comparable au DES, et peut opérer suivant les mêmes modes que le DES [c'est-à-dire ECB, CBC, OFB et CFC (§ 3.3.2)]. La philosophie de conception à la base de l'algorithme consiste à mélanger de manière astucieuse des transformations algébriques de nature différente, faciles à implémenter en hardware comme en software. Les trois opérations de base d'IDEA sont les suivantes :

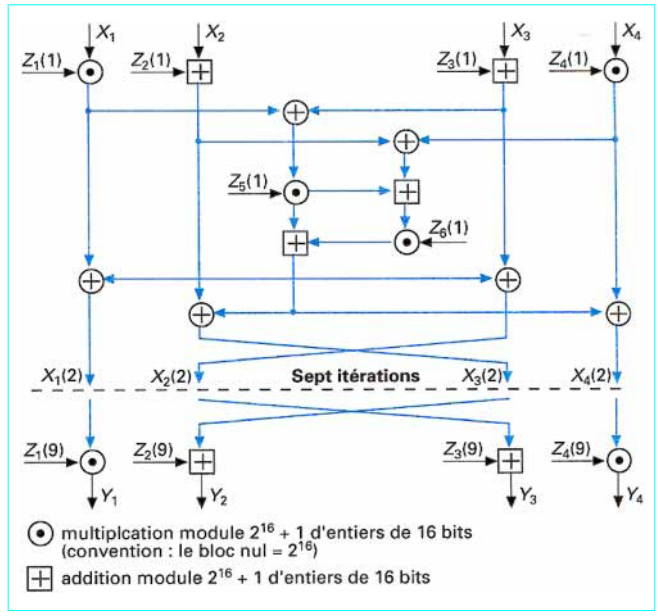


Figure 12 – Structure générale de l'algorithme IDEA

— le Ou exclusif bit à bit de blocs binaires (noté XOR) ;

— l'addition modulo 2^{16} (notée $\oplus$ dans ce paragraphe) ;

— la multiplication modulo $2^{16} + 1$ (notée $\otimes$), qui joue un peu le même rôle que les boîtes *S* dans le DES.

Fonctionnement général de l'algorithme

La figure 12 représente un descriptif général de l'algorithme IDEA. Les 64 bits du bloc d'entrée à encrypter, *X*, sont divisés en 4 mots de 16 bits, notés X_1, X_2, X_3, X_4. Ces quatre blocs sont alors présentés en entrée de la première itération de l'algorithme, qui en comporte huit de même nature plus une étape terminale. Chacune des huit itérations de l'algorithme utilise 6 blocs de 16 bits, extraits de la clef initiale *K* de 128 bits et utilisés comme sous-clefs d'encryption. Ces 6 blocs sont notés, à la *i*-ème itération, $Z_1(i)$, ..., $Z_6(i)$.

On note $X_1(i)$, $X_2(i)$, $X_3(i)$, $X_4(i)$ les entrées de l'algorithme à la *i*-ème itération [de sorte que $X_1(1) = X_1$, ..., $X_4(1) = X_4$]. À chaque itération, la suite d'opérations suivante est exécutée :

— calculer $M_1(i) = X_1(i) \otimes Z_1(i)$

— calculer $S_1(i) = X_2(i) \oplus Z_2(i)$

— calculer $S_2(i) = X_3(i) \oplus Z_3(i)$

— calculer $M_2(i) = X_4(i) \otimes Z_4(i)$

— calculer $R_1(i) = M_1(i)$ XOR $S_2(i)$

— calculer $R_2(i) = M_2(i)$ XOR $S_1(i)$

— calculer $M_3(i) = R_1(i) \otimes Z_5(i)$

— calculer $S_3(i) = R_2(i) \oplus M_3(i)$

— calculer $M_4(i) = S_3(i) \otimes Z_6(i)$

— calculer $S_4(i) = M_3(i) \oplus M_4(i)$

— calculer $Y_1 = M_1(i)$ XOR $M_4(i)$; $Y_2 = S_2(i)$ XOR $M_4(i)$
$Y_3 = S_2(i)$ XOR $S_4(i)$; $Y_4 = M_2(i)$ XOR $S_4(i)$

— si $i \leqslant 7$, faire $i = i + 1$ et poser :

Tableau 8 – Correspondance entre sous-clefs d'encryption et sous-clefs de décryption pour l'algorithme IDEA

Itération	Clefs d'encryption	Sous-clefs de décryption					
1	$Z_i(1)$, $i = 1, ..., 6$	$Z_1(9)^{-1}$	$-Z_2(9)$	$-Z_3(9)$	$Z_4(9)^{-1}$	$Z_5(8)$	$Z_6(8)$
2	$Z_i(2)$, $i = 1, ..., 6$	$Z_1(8)^{-1}$	$-Z_3(8)$	$-Z_2(8)$	$Z_4(8)^{-1}$	$Z_5(7)$	$Z_6(7)$
3	$Z_i(3)$, $i = 1, ..., 6$	$Z_1(7)^{-1}$	$-Z_3(7)$	$-Z_2(7)$	$Z_4(7)^{-1}$	$Z_5(6)$	$Z_6(6)$
4	$Z_i(4)$, $i = 1, ..., 6$	$Z_1(6)^{-1}$	$-Z_3(6)$	$-Z_2(6)$	$Z_4(6)^{-1}$	$Z_5(5)$	$Z_6(5)$
5	$Z_i(5)$, $i = 1, ..., 6$	$Z_1(5)^{-1}$	$-Z_3(5)$	$-Z_2(5)$	$Z_4(5)^{-1}$	$Z_5(4)$	$Z_6(4)$
6	$Z_i(6)$, $i = 1, ..., 6$	$Z_1(4)^{-1}$	$-Z_3(4)$	$-Z_2(4)$	$Z_4(4)^{-1}$	$Z_5(3)$	$Z_6(3)$
7	$Z_i(7)$, $i = 1, ..., 6$	$Z_1(3)^{-1}$	$-Z_3(3)$	$-Z_2(3)$	$Z_4(3)^{-1}$	$Z_5(2)$	$Z_6(2)$
8	$Z_i(8)$, $i = 1, ..., 6$	$Z_1(2)^{-1}$	$-Z_3(2)$	$-Z_2(2)$	$Z_4(2)^{-1}$	$Z_5(1)$	$Z_6(1)$
Finale	$Z_i(9)$, $i = 1, ..., 4$	$Z_1(1)^{-1}$	$-Z_2(1)$	$-Z_3(1)$	$Z_4(1)^{-1}$		

$X_1(i+1) = Y_1$, $X_2(i+1) = Y_3$, $X_3(i+1) = Y_2$, $X_4(i+1) = Y_4$ revenir à la première étape ;

si $i = 8$, calculer :

$Y_1 = X_1(8) \otimes Z_1(9)$, $Y_2 = X_2(8) \oplus Z_2(9)$, $Y_3 = X_3(8) \oplus Z_3(9)$, $Y_4 = X_4(8) \otimes Z_4(9)$

Y_1, Y_2, Y_3, Y_4 constituent, une fois réassemblés, le cryptogramme Y associé à X.

Génération des clefs

La clef initiale K de l'algorithme IDEA comporte 128 bits.

À chacune des 8 itérations principales de l'algorithme, 6 blocs de 16 bits, dérivés de K sont utilisés comme sous-clefs d'encryption. L'étape finale ci-dessus n'en utilise que 4. La procédure pour générer les clefs d'encryption est particulièrement simple.

- La clef K est initialement partitionnée en 8 blocs de 16 bits, $K_1, ..., K_8$. À l'itération 1, on a $Z_i(1) = K_i$ pour $i = 1, ..., 6$.
- À l'itération 2, on a $Z_1(2) = K_7$, $Z_2(2) = K_8$. On décale alors la clef K de 25 bits vers la gauche, pour obtenir une nouvelle clef K', elle aussi formée de 8 blocs de 16 bits, notés $K'_1 ... K'_8$. On pose alors : $Z_{i+2}(2) = K'_1$ pour $i = 1, ..., 4$.
- Les quatre premières sous-clefs correspondant à l'itération 3 sont alors définies par $Z_i(3) = K'_{i+4}$ pour $i = 1, ... 4$. On permute alors à nouveau K', de 25 bits vers la gauche, pour obtenir 8 nouveaux blocs de 16 bits, $K''_1, ..., K''_8$. Les deux premiers blocs sont utilisés pour obtenir les deux sous-clefs $Z_5(3)$ et $Z_6(3)$, les six derniers constituent les sous-clefs de l'itération 4, $Z_1(4), ..., Z_6(4)$. On permute alors à nouveau K'' de 25 bits vers la gauche pour calculer les 6 blocs de 16 bits constituant les sous-clefs de l'itération 5 et ainsi de suite jusqu'à la fin de l'algorithme.

Décryption

Pour décrypter un cryptogramme IDEA, il suffit alors d'exécuter l'algorithme en sens inverse, ce qui revient à exécuter 8 itérations principales et une étape finale comme ci-dessus, mais avec des sous-clefs différentes, qui s'expriment en fonction des sous-clefs d'encryption $Z_i(k)$, de leurs inverses pour l'addition modulo 2^{16} [notées $-Z_i(k)$] ou de leurs inverses pour la multiplication modulo $2^{16} + 1$ (notées $Z_i(k)^{-1}$, avec la restriction que le bloc nul est censé représenter $2^{16} = -1$ si bien que l'inverse multiplicatif de 0 est 0).

Le tableau **8** fournit la correspondance entre sous-clefs d'encryption et sous-clefs de décryption pour l'algorithme IDEA.

Implémentations de l'algorithme IDEA

IDEA se prête très bien à une implémentation en software, et a un taux d'encryption en général deux fois supérieur à celui du DES. Les taux d'encryption observés sur une machine à base de processeur Intel 386 à 33 MHz sont de l'ordre de 880 kbit/seconde, cela passant à environ 2,5 Mbit/seconde sur une machine à base Intel 486 à 66 MHz. Il existe, par ailleurs, des implémentations hardware de l'algorithme IDEA, tel le processeur dédié développé par l'ETH de Zurich, et qui offre des taux d'encryption de 177 Mbit/seconde, ce qui constitue déjà un débit tout à fait raisonnable.

Cryptanalyse de l'algorithme IDEA

IDEA est aujourd'hui un algorithme considéré comme très robuste, et qui a remarquablement résisté à toutes les tentatives de cryptanalyse. Tout au plus a-t-on remarqué l'existence de quelques clefs faibles, pouvant donner lieu à des attaques sur texte clair choisi permettant de retrouver aisément de telles clefs d'encryption. La probabilité d'occurrence de telles clefs étant cependant de l'ordre $1/2^{96}$, cela n'entame en rien, du point de vue pratique, la sécurité d'IDEA. On manque cependant aujourd'hui de données précises sur le comportement de l'algorithme vis-à-vis de techniques d'attaque du genre de celles utilisées pour le DES, telles la cryptanalyse différentielle ou linéaire. L'opinion générale est cependant que, en raison même de la longueur de clef utilisée (128 bits contre 56 pour le DES) et des fondations mathématiques particulièrement solides de l'algorithme, IDEA est probablement aujourd'hui le meilleur algorithme d'encryption par bloc utilisable en pratique. Cela rend bien entendu l'utilisation de PGP particulièrement attractive pour les applications.

6.2 PEM (*Internet Privacy Enhanced Mail*)

PEM est le standard de messagerie électronique sécurisée soutenu par l'*Internet Architecture Board*, comité officiellement en charge des évolutions architecturales d'Internet. Les travaux sur PEM ont été initialisés dès 1985 et ont abouti à la publication de toute une série de spécifications, dont l'ensemble de documents les plus récents (*Requests for Comments* ou RFC 1421-1424) sont proposés comme standards officiels Internet. PEM offre toute une série de services [21] dont les principaux sont les suivants.

Confidentialité

PEM inclut tout un ensemble de fonctions cryptographiques, offrant les fonctions d'encryption et signatures de messages, génération et distribution de clefs cryptographiques. *A priori*, PEM supporte à la fois les algorithmes symétriques et à clefs publiques pour la distribution des clefs, mais encourage surtout ces derniers, en particulier le système RSA. Les clefs publiques sont accompagnées d'un certificat conforme à la recommandation X.509 du CCITT, et constitué d'un certain nombre d'attributs associés à la clef [n° de version, n° de série, type algorithme de signature utilisé (par exemple RSA avec condensé MD5...), émetteur de la clef, validité...]. L'encryption des messages proprement dite est réalisée à partir du DES en mode CBC.

Authentification

Les facilités correspondantes permettent à un utilisateur recevant un message de déterminer avec certitude l'identité du transmetteur du message (ou du retransmetteur, lors de messages redirigés en cascade).

Intégrité en mode sans connection

Les fonctions correspondantes de PEM servent à garantir que le contenu reçu d'un message est bien identique au contenu envoyé. Le terme *sans connection*, dérivé de la terminologie OSI, signifie que l'ordre d'arrivée des messages est sans importance au niveau du service d'intégrité.

Support pour la non-réfutation

Un problème important en messagerie électronique est la redirection des messages, souvent à travers une cascade de nœuds informatiques. Aussi est-il important, en particulier pour les problèmes de commerce électronique, de pouvoir certifier à la fois l'intégrité du contenu et l'identité de l'émetteur (et pas uniquement du transmetteur) d'un message (par exemple, lors d'une commande ou d'une facturation électronique).

La soumission d'un message sous PEM s'effectue de la manière suivante :

— l'utilisateur spécifie en clair un préambule standard Internet de message comprenant l'adresse du récepteur et un certain nombre de paramètres optionnels tels que le sujet du message, la date... Ces divers champs se retrouveront en général dans l'en-tête standard du message électronique reçu. À la fin de la saisie de ces paramètres, l'utilisateur rentre alors dans la partie sécuritaire de PEM ;

— PEM produit alors un en-tête de sécurité, comprenant tous les champs nécessaires pour effectuer les services d'authentification, d'intégrité, d'encryption. Ces services peuvent d'ailleurs être utilisés de manière complètement découplée : on distingue ainsi plusieurs niveaux :

- MIC-CLEAR, dans lequel tous les destinataires du message transmis seront capables de le lire, mais seuls ceux utilisant PEM seront capables d'en vérifier l'intégrité et l'authenticité,
- MIC-ONLY permettant en plus d'assurer le passage du message à travers n'importe quelle passerelle de réseau sans altération d'intégrité ou d'authenticité,
- ENCRYPTED enfin, dans lequel seuls les utilisateurs PEM seront capables de déchiffrer le message transmis, avec les mêmes garanties d'intégrité et d'authenticité que précédemment.

Trois champs spécifiques de l'en-tête sécurisé (*Originator-Certificate, Key-Info, Issuer-Certificate*) permettront au destinataire de vérifier l'intégrité et l'authenticité du message transmis. Un champ MIC-info précise les algorithmes d'encryption et de calcul du condensé utilisés pour les procédures d'intégrité et authenticité (par exemple RSA et MD5). La méthode employée, dans le cas d'emploi d'un système type RSA, est exactement identique à celle de PGP. Le certificat X.509 de l'émetteur est encrypté avec sa clef privée et le compressé du message chiffré avec la clef d'encryption DES utilisée.

Au niveau encryption, un champ KEK-Info spécifie la souche d'initialisation sur 64 bits du DES en mode CBC. La clef DES utilisée pour chiffrer le message est encryptée avec la clef publique du destinataire et placée dans un champ *Key-info*, précédé d'un champ *Recipient-ID* spécifiant l'identité du récepteur.

— Le message encrypté proprement dit est alors ajouté à l'en-tête de sécurité.

L'ensemble préambule/en-tête sécurisé/message encrypté est alors encodé conformément au standard SMTP (*Internet Simple Mail Protocol*) et envoyé sur le réseau.

Internet offre enfin une hiérarchie de certification destinée à offrir toutes les garanties possibles aux utilisateurs quant à la validité et à la sémantique des différents certificats utilisés sous PEM. Cette hiérarchie est en fait un arbre d'habilitation, dont la racine, appelée *Internet Policy Certification Registration Authority* ou IPRA constitue le point de référence pour la validation de tous les certificats émis sous PEM. L'IPRA à son tour produit des certificats d'authenticité à des organisations de deuxième niveau, appelées *Policy Certification Authorities* ou PCA, dûment enregistrées auprès de l'IPRA et pouvant pratiquer des procédures de certification spécifiques vers un troisième niveau communément appelé *Certification Authority* ou CA. Les différents CA de PEM peuvent être de nature :

— organisationnelle (par exemple une compagnie industrielle habilitant certains de ses employés à travailler sous PEM) ;

— résidentielle, par exemple au niveau d'une municipalité ;

— privée.

Pour ces questions assez délicates, le lecteur intéressé est renvoyé aux références [22] et [26].

Il existe aujourd'hui plusieurs implémentations de PEM : citons par exemple les produits RIPEM, développés par Mark Riordan, et TIS-PEM (essentiellement pour systèmes UNIX) de la société Trusted Information Systems. Comme la majeure partie des programmes cryptographiques, RIPEM et TIS-PEM ne sont pas exportables hors des États-Unis et du Canada.

6.3 PGP versus PEM

Du strict point de vue des fonctionnalités, PGP et PEM peuvent ne pas paraître très différents *a priori*, et ces deux produits offrent effectivement des services très semblables. Malgré toutes les restrictions liées à son exportation, PGP est un produit très répandu et en général très apprécié des utilisateurs. Le succès de PEM n'apparaît pas encore aussi évident mais la pénétration sur le marché des standards de messageris X400 et X500 devrait en accélérer la diffusion. Une différence visible entre les deux approches est cependant liée au fait que, dans PGP, il est impossible de vérifier une signature portant sur un message encrypté, contrairement à ce qui est possible sous PEM. En ce sens, on pourrait dire que **PGP** provilégie le **secret** sur l'authentification, alors que **PEM** est d'abord un système d'**authentification**.

Sur la philosophie générale en revanche, les deux approches sont très différentes ; alors que PEM utilise une approche très centralisée pour la certification des clefs et la gestion/validation des certificats associés, PGP prend exactement l'optique inverse : chaque utilisateur de PGP construit progressivement son réseau d'autorités de confiance pour la certification des clefs. À partir du moment où un utilisateur *a confiance* dans une autorité A pour l'émission de clefs et la production de certificats associés, alors il est capable de vérifier que les clefs publiques produites par A sont bien à ceux qui déclarent les posséder.

On retrouve dans cette dualité PGP/PEM l'essence même du débat très âpre engagé aux États-Unis sur le role des organismes fédéraux dans l'établissement des politiques sécuritaires liées aux informations possédées par chaque citoyen. Comme le dit P. Zimmermann lui-même de manière très imagée, PGP est pour ceux qui préfèrent *plier leur parachute eux-mêmes*.

7. Implémentation de la cryptographie dans une architecture

7.1 Modèle original SNA

On retrouvera une description pratiquement exhaustive de ce modèle dans [37]. Le modèle SNA est applicable (et généralisable) aux réseaux informatiques hiérarchisés, distinguant les concepts de machine hôte et de terminal connecté. Dans l'implémentation initiale SNA, chaque système hôte possède une clef maître *KM* 0, dont sont dérivées deux variantes *KM* 1 et *KM* 2. Chaque terminal connecté au système hôte possède lui, une clef maître *KMT*.

Le système hôte stocke et met à jour l'ensemble de toutes les clefs *KMT*, sous forme d'un fichier d'articles, chaque article étant constitué d'éléments du type $E_{KM\,1}\,(KMT)$. (Dans le cas SNA, l'algorithme d'encryption *E* n'est autre que le DES).

La clef maître *KM* 0 est stockée dans une mémoire non volatile d'un module de sécurité spécial, dispositif matériel connecté au système hôte. Les clefs *KM* 1 et *KM* 2 servent à l'encryption de clefs d'encryption temporaires (fichiers ou communication) avant distribution aux utilisateurs du réseau.

Dans l'architecture SNA, tant le système hôte que les terminaux connectés sont pourvus de capacités cryptographiques destinées, soit à la gestion des clefs, soit aux opérations d'encrytion/décrytion.

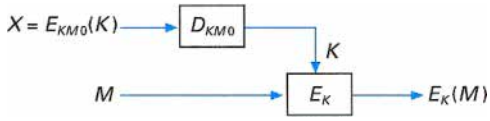
■ **Au niveau du système hôte,** les fonctions suivantes sont implémentées.

1 – Set_master_key (*smk*) : cette fonction, tout à fait particulière et nécessitant des pratiques sécuritaires spéciales, installe la clef maître *KM* 0 dans le module de sécurité du système hôte.

2 – Encipher_under_master_key (*emk*) : cette fonction encrypte une clef *K* sous *KM* 0.

$$emk\,(K) = E_{KM\,0}\,(K)$$

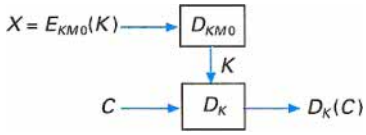
3 – Encipher (*ecph*) : cette fonction encrypte un message *M* sous la clef *K*. Pour ce faire, la clef *K* est passée au module de sécurité encryptée sous *KM* 0, suivant le schéma :



ainsi, on a :

$$ecph\,(M, X) = E_{D_{KM0}(X)}(M)$$

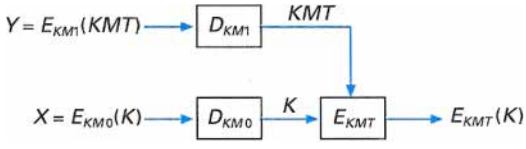
4 – Decipher (*deph*) : cette fonction est l'inverse de la précédente et permet, étant donné le cryptogramme *C* et la clef *K* passée au module de sécurité sous la forme $E_{KM\,0}\,(K)$, de produire $D_K\,(C)$. On a ici le schéma fonctionnel suivant :



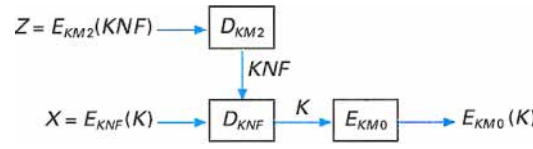
5 – Reencipher_from_master_key (*rfmk*) : cette fonction permet de réencrypter une clef encryptée sous *KM* 0 en une clef encryptée sous une clef terminal *KMT* [*KMT* étant stocké sous la forme $Y = E_{KM\,1}\,(KMT)$].

Le diagramme fonctionnel est le suivant :

$$rfmk\,(Y, X) = E_{KMT}\,(K)$$



6 – Reencipher_to_master_key (*rtmk*) : cette dernière fonction permet, étant donné une clef d'encryption elle-même encryptée sous une clef secondaire *KNF*, de la récupérer encryptée sous *KM* 0. Les clefs *KNF* sont stockées dans un fichier encrypté sous *KM* 2. Cela conduit au diagramme fonctionnel suivant :



ainsi $rtmk\,(Z, X) = E_{KM\,0}\,(K)$

■ Au niveau du terminal, seules trois opérations sont autorisées.

1 – Decipher_from_master_key (*dmk*) : étant donné la clef *K* transmise au terminal encryptée sous sa clef maître *KMT*, cette opération déchiffre *K* et la stocke dans un registre temporaire du module sécuritaire du terminal.

2 – Encipher (*ecph*) : cette opération encrypte un message *M* avec la clef stockée dans le registre temporaire du module sécuritaire du terminal.

3 – Decipher (*dcph*) : cette opération décrypte le message chiffré *C*, avec la clef stockée dans le registre temporaire du module sécuritaire.

Ainsi décrit, le schéma des différentes fonctions représente, dans ses grandes lignes, le modèle de base d'implémentation de la cryptographie dans l'architecture SNA d'IBM. Cette implémentation n'est plus réalisée sur les sytèmes IBM actuels mais tous les systèmes à clefs secrètes en ont repris les concepts fondamentaux. Il présente les grands avantages suivants :

— séparation des fonctionnalités d'encryption pour les communications et les fichiers (*via* les variantes *KM* 1 et *KM* 2 de *KM* 0) ;
— gestion dynamique des clefs primaires ;
— impossibilité de récupérer, par combinaison des fonctions du système hôte ;
- les clefs temporaires *K* ou maîtres *KMT*,
- les mêmes données encryptées sous *KM* 0 [c'est-à-dire les quantités $E_{KM\,0}\,(KMT)$ ou $E_{KM\,0}\,(K)$], ce qui corromprait irrémédiablement la sécurité du réseau,

— extensibilité et adaptabilité de l'architecture à un réseau multi-hôtes [38] ;
— localisation de la sécurité : une information acquise par effraction ne permet pas de déduire des informations adjacentes (sauf en ce qui concerne *KM* 0, dont la corruption entraîne la vulnérabilité de tout le réseau : cette éventualité est cependant exclue, compte tenu des procédures associées à la fonction *smk* [38].

7.2 Gestion des clefs cryptographiques

Le problème de la gestion des clefs cryptographiques dans un réseau correspond aux préoccupations suivantes :
— génération des informations élémentaires ;
— stockage temporaire ou permanent ;
— distribution.

Un traitement exhaustif de ces questions est en dehors du sujet de cet article. Nous présenterons dans ce paragraphe quelques solutions implémentées dans la pratique, renvoyant le lecteur intéressé à l'abondante littérature dédiée à ces sujets [3] [37] [38].

7.2.1 Génération des clefs dans le réseau IBM

Deux modèles de génération sont proposés en pratique.

■ **Utilisation du DES comme générateur de nombres pseudo-aléatoires couplé à un générateur de nombres aléatoires**

Cela suppose installés *KM* 0 et les variantes *KM* 1 et *KM* 2.

Un nombre *RN* étant tiré au hasard, on procède en deux étapes.

● **Étape 1** : on pose $RN = E_{KM\,1}\,(a)$

Étant donné une quantité connue, par exemple *TOD* (*Time of Day* : la date du jour), on calcule :

$$A_i = rfmk\,(RN,\ TOD + i)$$

c'est-à-dire $$A_i = \mathrm{DES}_a\,(\mathrm{DES}^{-1}_{KM\,0}\,(TOD + i))$$

avec $a = \mathrm{DES}^{-1}_{KM\,1}(RN)$

● **Étape 2** : la i^{e} clef temporaire, K_i, est obtenue par :

$$Y_i = rfmk\,(RN,\ A_i)$$

$$K_i = \mathrm{Parity}\,(Y_i)$$

c'est-à-dire que K_i s'obtient en ajoutant, après chaque octet de Y_i, un bit de parité.

■ **Appel à un générateur de nombres pseudo-aléatoires externe**

Dans ce cas, la i^{e} clef d'encryption temporaire de communication KS_i (respectivement de fichiers, KF_i) est déterminée, à partir d'un générateur de nombres pseudo-aléatoires $RN\,(k)$, par :

$$RN\,(i) = \mathrm{DES}_{KM\,0}\,(KS_i)$$

$$RN\,(i) = \mathrm{DES}_{KM\,2}\,(KF_i)$$

On remarque que, par ce procédé, jamais KS_i ni KF_i ne sont produits en clair dans le réseau.

Bien sûr, la question de disposer d'un *bon* générateur de nombres pseudo-aléatoires se pose. Mathyas et Meyer [37] proposent le générateur suivant.

Soit une quantité u_0 donnée, et une quantité déterministe $Z\,(i)$, ils obtiennent $RN\,(i)$ par l'application successive de deux opérations *rtmk*.

● **Étape 1** : connaissant $u\,(i-1)$ et $Z\,(i)$, on calcule $u\,(i)$ par :

$$u\,(i) = rtmk\,(u\,(i-1),\ Z\,(i))$$

soit $$u\,(i) = \mathrm{DES}_{KM0}\,[\mathrm{DES}^{-1}_{\mathrm{DES}^{-1}_{KM2}(u\,(i-1))}(Z(i))]$$

● **Étape 2** : $RN\,(i)$ est obtenu à partir de $u\,(i)$ par :

$$RN\,(i) = rtmk\,(u\,(i),\ u\,(i))$$

soit $$RN\,(i) = \mathrm{DES}_{KM0}[\mathrm{DES}^{-1}_{\mathrm{DES}^{-1}_{KM2}(u(i))}(u(i))]$$

7.2.2 Stockage des clefs

Dans tous les cas, les clefs secrètes ne doivent jamais être stockées en clair. Il est donc très important de sécuriser, en général par encryption, les fichiers contenant les clefs d'encryption temporaire de communication ou de fichiers. Dans l'approche IBM, par exemple, les clefs de communication sont stockées encryptées sous $KM\,1$, les clefs d'encryption de fichiers stockées encryptées sous $KM\,2$.

7.2.3 Distribution de clefs cryptographiques

Nous proposons ici, par deux approches très différentes, de montrer le type de procédure à mettre en œuvre, dès lors que deux usagers A et B d'un réseau informatique désirent partager une clef commune pour s'échanger, de manière cryptée, des messages ou des fichiers.

■ **Dans la première approche**, nous supposerons le réseau basé sur un algorithme complètement symétrique (par exemple le DES). A est connecté au système hôte H_A, et B au système hôte H_B. Chaque hôte H possède un module de sécurité où sont stockées les clefs maîtres et leurs variantes ($KM\,0_A$, $KM\,1_A$, $KM\,2_A$, KMT_A ... pour H_A, $KM\,0_B$, $KM\,1_B$, $KM\,2_B$, KMT_B ... pour H_B). De plus, H_A et H_B partagent une clef de communication secondaire, K_H, stockée sous forme $W_A = E_{KM1_A}\,(K_{AB})$ pour H_A, sous forme $W_B = E_{KM1_B}\,(K_{AB})$ pour H_B.

Supposons donc que A veuille échanger avec B des informations encryptées sous la clef K. K a été généré par un des procédés précédents ; par exemple, on a :

$$R = E_{KM0_A}\,(K)$$

Le protocole d'échange se déroule alors de la manière suivante :

1 – H_A transmet R à H_B *via*

$$rfmk\,(W_A,\ R) = E_{K_{AB}}\,(R)$$

2 – H_B obtient R en calculant

$$rtmk\,(W_B,\ E_{K_{AB}}\,(R)) = R$$

3 – H_A envoie à A la quantité Z_A

$$Z_A = rfmk\,(E_{KM1_A}\,(KMT_A),\ R) = E_{KMT_A}\,(K)$$

4 – A obtient K en déchiffrant Z_A à l'aide de sa clef KMT_A

5 – de même, H_B envoie à B la quantité Z_B

$$Z_B = rfmk\,(E_{KM1_B}\,(KMT_B),\ R) = E_{KMT_B}\,(K)$$

6 - B déchiffre de même Z_B avec sa clef KMT_B

■ **Dans la seconde approche**, nous supposerons que le réseau est mixte, c'est-à-dire qu'il utilise un système à clefs publiques, pour la distribution de clefs symétriques, par exemple de clefs DES. Ce type de système tend à se répandre de plus en plus en pratique, car il offre une sécurité importante tout en réduisant les coûts de sécurité au niveau des hôtes. L'approche décrite ci-après a été développée par MITRE Corporation [40], sur la base d'un schéma de distribution initialement proposé par Diffie et Hellman [11]. Dans ce schéma, un nombre premier p très grand est rendu public.

Chaque utilisateur du réseau A choisit au hasard un nombre x_A dans l'intervalle $[0,\ p-1]$.

Un nombre a, également dans l'intervalle $[0,\ p-1]$, est rendu public, et chaque utilisateur calcule une quantité publique, $Y_A = a^{x_A}$ (modulo p).

Quand A veut partager une clef DES K avec B, le protocole suivant est appliqué.

1) • A lit dans le répertoire public la quantité y_B et calcule

$$K_{AB} = (y_B)^{x_A} = (y_A)^{x_B}\ (\text{modulo } p),$$

• A génère deux nombres aléatoires R et R_{AB},

• A définit une valeur d'initialisation $I_{AB} = \mathrm{DES}^{-1}_{k}\,(R_{AB})$, où K est la clef DES à partager avec B, définie par

$$K = \mathrm{DES}^{-1}_{K_{AB}}\,(R)$$

2) A envoie à B le message

$$M_1 = (A,\ R,\ R_{AB})$$

3) B obtient y_A à partir du répertoire public et calcule

$$K_{AB} = (y_A)^{x_B} = (y_B)^{x_A}\ (\text{modulo } p),$$

4) • B calcule $K = \mathrm{DES}^{-1}_{K_{AB}}\,(R)$, $I_{AB} = \mathrm{DES}^{-1}_{K}\,(R_{AB})$

• B modifie I_{AB} de manière prédéfinie, le transformant en I'_{AB}, et calcule $X = \mathrm{DES}^{-1}_{K}\,(I'_{\mathrm{AB}})$

• B genère un nombre aléatoire S_{AB} et calcule :

$$I_{BA} = \mathrm{DES}^{-1}_{K}\,(S_{AB})$$

5) *B* envoie à *A* le message

$$M_2 = (X, S_{AB})$$

6)• *A* calcule $\mathrm{DES}_K^{-1}(X) = I'_{AB}$

• *A* transforme I'_{AB} suivant le processus inverse de celui appliqué par *B*. Si la quantité correspondante $\bar{I}_{AB}$ est identique à I_{AB}, alors *A* authentifie *B* comme l'auteur du message ;

• *A* calcule $\bar{I}_{BA} = \mathrm{DES}_K^{-1}(S_{AB})$ et le modifie de manière prédéfinie en $\overline{I'_{BA}}$;

• *A* calcule $Y = \mathrm{DES}_K(\overline{I'_{BA}})$ et l'envoie à *B* ;

7) *B* calcule $\mathrm{DES}_K^{-1}(\overline{I'_{BA}})$ et le modifie suivant la procédure inverse (prédéfinie) de celle utilisée par *A*.

Si la quantité obtenue, I''_{BA} est identique à I'_{BA}, alors *B* authentifie *A*, et les échanges encryptés peuvent alors commencer. On est sûr en effet, à cette étape, que *A* et *B* partagent une clef commune et se sont authentifiés mutuellement. Toute la sécurité de ce protocole réside dans le fait que, connaissant $y_A = a^{x_A}$ (modulo p) et p, il est très difficile, sinon impossible, d'en déduire x_A. Ce problème, connu en théorie des nombres sous le nom de problème du « logarithme discret », continue de susciter un intérêt soutenu des chercheurs en arithmétique (§ 4).

8. Conclusion

Cet article a tenté de donner un aperçu rapide des principes fondamentaux de la cryptographie moderne ainsi que de l'évolution des techniques cryptographiques civiles durant les cinq dernières années. Le développement des réseaux mondiaux et les immenses possibilités offertes par les transactions électroniques posent aujourd'hui de manière cruciale le problème de la protection des informations échangées sur les réseaux informatiques mondiaux. Les individus et les organisations sont ainsi devenus de plus en plus sensibles à la sécurité des systèmes mis à leur disposition, ainsi qu'aux aspects de standardisation liés à la diffusion des techniques cryptographiques. Au-delà de ces aspects techniques, la cryptographie pose cependant, de plus en plus, un véritable problème de société et devient, au moins aux États-Unis, l'objet d'un débat central associé à deux préoccupations contradictoires :

— le désir légitime des individus ou organisations commerciales de **protéger** les informations essentielles à leur vie privée ou à leurs activités économiques ;

— l'exigence *a priori* justifiée de l'autorité publique de pouvoir en permanence **contrôler les informations** du ressort de la sécurité publique ou d'État (il est clair que l'utilisation incontrôlée et à des fins illégales de la cryptographie par des individus ou organisations à but maffieux ou politique, par exemple, constitue une réelle menace pour la plupart des pays industrialisés).

Le congrès américain a récemment demandé au National Research Council de conduire une étude d'ensemble sur la cryptographie, en partie suite à la vague de protestations résultant de la tentative de mise en place en 1993 par le gouvernement américain d'un projet d'étude de système de cryptographie *avec tiers de confiance*.

En fait, il ne s'agissait ni plus ni moins que d'imposer, à tous les individus ou organisations souhaitant utiliser la cryptographie, l'utilisation d'un algorithme classifié conçu par la NSA, nommé Skipjack, et implémenté en hardware dans des circuits intégrés nommés *Clipper* et *Campstone* destinés à équiper tous les appareils devant transmettre des informations de manière chiffrée (fax, modems, téléphones portables, ordinateurs personnels...). Dans cette vision très influencée par la NSA, le gouvernement américain (et sans doute la NSA elle-même) aurait la possibilité d'accéder à toutes les clefs de chiffrement inscrites dans les puces *Clipper* ou *Campstone* et aussi (sous contrôle légal) à toutes celles générées pour des transactions électroniques.

Bien que des experts civils aient reconnu l'algorithme Skipjack comme sûr et que le gouvernement ait depuis admis la possibilité d'implémenter un tel système en software, le tollé fut général et remonta jusqu'au Congrès, déclenchant l'étude demandée au NRC. En conséquence, la mise en œuvre généralisée d'un système cryptographique *avec tiers de confiance* aux États-Unis ne pourra sans doute pas avoir lieu avant la remise des conclusions du rapport du NRC, prévue pour fin 1996.

Sans doute s'agit-il ici de la première manifestation du désaccord entre pouvoir de l'autorité et droit du citoyen à la protection de sa vie privée, dans un pays traditionnellement et constitutionnellement attaché à la liberté de l'individu. Il n'en reste pas moins vrai qu'il préfigure vraisemblablement le type de débat qui se posera dans la plupart des pays fortement impliqués dans la mise en œuvre d'une société de l'information, qui devrait entraîner, à moyenne ou longue échéance, une révision drastique des législations en vigueur. Pour le bien des citoyens, on ne peut qu'espérer que de telles révisions s'effectueront dans le cadre d'une large concertation, avec le double souci de préserver les intérêts publics et privés. D'ailleurs, des discussions sur la mise en œuvre de systèmes cryptographiques *avec tiers de confiance* sont maintenant également envisagées en Europe (et même en France, où le concept est discuté !).

Cryptographie

par **Jean-Pierre TUAL**
Directeur de l'unité cartes à microprocesseurs
Bull/CP8 Transac

Références bibliographiques

[1] SHANNON (C.E.). – *Communication theory of secrecy systems.* Bell Syst. Tech. J. *vol. 28*, p. 656-71 (1949).

[2] ROUBATY (R.). – *ABC de cryptographie.* Masson (1984).

[3] DENNING (D.). – *Cryptography and data security.* Addison Wesley (1983).

[4] KONHEIM (A.J.). – *Cryptography : a primer.* John Wiley & Sons (1981).

[5] DE VIGENÈRE (B.). – *Traictés des chiffres ou secrètes manières d'escrire.* Paris (1586).

[6] DEAVOURS (C.A.) et KRUH (L.). – *Machine cryptography and modern cryptanalysis.* Artech House (1985).

[7] DEAVOURS (C.A.), KRUH (L.), KAHN (D.), MELLEN (G.) et WINKEL (B.). – *Cryptology yesterday, today and tomorrow.* Artech House (1987).

[8] GOLLMANN (D.). – *Pseudo random properties of cascade connections of clock controlled shift registers.* Proceedings EUROCRYPT 84, Springer Verlag, LN in Comp. Science n° 209, p. 93-109.

[9] DIFFIE (W.) et HELLMAN (M.E.). – *Privacy and authentication : an introduction to cryptography.* Proceedings of the IEEE, *vol 67*, n° 3, p. 397-427 (1979).

[10] DIFFIE (W.) et HELLMAN (M.E.). – *Exhaustive cryptanalysis of the NBS data encryption standard.* Computer, *vol. 10*, n° 6, p. 74-80 (1977).

[11] DIFFIE (W.) et HELLMAN (M.E.). – *New directions in cryptography.* IEEE Trans. on Inf. Theory, *vol. II-22*, n° 6, p. 644-51 (1976).

[12] SHAMIR (A.). – *A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem.* Proceedings of CRYPTO 82, Plenum Press, p. 275-88 (1983).

[13] BRICKELL (E.F.), DAVIS (J.A.) et SIMMONS (G.J.). – *A preliminary report on the cryptanalysis of the Markle-Hellman Knapsack cryptosystem.* Proceedings of CRYPTO 82, Plenum Press, p. 289-301 (1983).

[14] RIVEST (R.), SHAMIR (A.) et ADLEMAN (L.). – *On digital signatures and public key cryptosystems.* Commun. ACM, *vol. 21*, p. 120-1 (1978).

[15] RIESEL (H.). – *Prime numbers and computer methods for factorization.* Birkhauser, Progress in Mathematics n° 57 (1985).

[16] BRESSOUD (D.M.). – *Factorization and primality testing.* Springer Verlag, Undergraduate Texts in mathematics (1989).

[17] COHEN (H.). – *A course in computational algebraic number theory.* Graduate Texts in Mathematics n° 138, Springer Verlag (1994).

[18] ATKIN (A.) et MORAIN (F.). – *Elliptic curves and primality proving.* Rapport de recherche n° 1152, INRIA (1990).

[19] FAHN (P.). – *Frequently asked questions about today's cryptography.* RSA Laboratories (1992).

[20] GARFINKEL (S.). – *PGP.* Éditions O'Reilly International Thomson (1995).

[21] HOFFMAN (L.J.) (Éd). – *Building in big brother.* Springer-Verlag (1995).

[22] MATSUI (M.). – *The first experimental cryptanalysis of the data encryption standard.* Advances in cryptology-CRYPTO'94, Lecture Notes, in Computer Science n° 839, Springer-Verlag (1995).

[23] CAMPBELL (K.W.) et WIENER (M.J.). – *Proof that DES is not a group.* Advances in cryptology-CRYPTO'92, Lecture Notes, in Computer Science n° 839, Springer-Verlag (1993).

[24] BIHAM (E.) et SHAMIR (A.). – *Differential cryptanalysis of DES-like cryptosystems.* Advances in cryptology-CRYPTO'90, Lecture Notes, in Computer Science n° 536, Springer-Verlag (1991).

[25] BIHAM (E.) et SHAMIR (A.). – *Differential cryptanalysis of the full 16-round DES, like cryptosystems.* Advances in Cryptology-CRYPTO'92, Lecture Notes in Computer Science n° 740, Springer-Verlag (1993).

[26] CCITT Recommendation X.509. – *The Directory-Authentication framework* (1988).

[27] RIVEST (R.). – *Ciphertext : the RSA Newsletter* (1993).

[28] SCHNORR (C.P.). – *Efficient identification and signatures for smart cards.* Advances in Cryptology-CRYPTO'89, Lecture Notes in Computer Science n° 435, Springer-Verlag (1990).

[29] El GAMAL (T.). – *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE transactions on Information Theory, *vol. IT 31*, p. 469-472 (1985).

[30] RIVEST (R.). – *Responses to NIST's proposal.* Communications of the ACM, *vol. 35*, n° 7 (1992).

[31] HELLMANN (M.). – *Responses to NIST's proposal.* Communications of the ACM, *vol. 35*, n° 7 (1992).

[32] GOLDWASSER (S.), MICALI (S.) et RACKHOFF (C.). – *The knowledge complexity of interactive proof systems.* SIAM Journal on Computing, *vol. 18*, p. 186-208 (1989).

[33] FEIGE (U.), FIAT (A.) et SHAMIR (A.). – *Zero knowledge proof of identity.* Journal of Cryptology, *vol. 1*, p. 77-94 (1988).

[34] GUILLOU (L.C.) et QUISQUATTER (J.J.). – *A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory.* Advances in Cryptology-EUROCRYPT'88, Lecture Notes in Computer Science, Springer-Verlag (1989).

[35] LAI (X.) et MASSEY (J.L.). – *A proposal for a next block encryption standard.* Advances in Cryptology-EUROCRYPT'90, Lecture Notes in Computer Science n° 473, Springer-Verlag (1992).

[36] PKCS 1-10. – RSA-DI Publications, novembre (10 documents au total) (1993).

[37] MEYER (C.H.) et MATYAS (S.M.). – *Cryptography.* Wiley Interscience (1983).

[38] MEYER (C.H.) et MATYAS (S.M.). – *Generation, distribution and installation of cryptographic keys.* IBM Syst. J. *vol. 17*, n° 2 (1979).

[39] AKL (S.G.). – *Digital signatures : a tutorial survey.* Computer, *vol. 16*, n° 2, p. 15-24 (1983).

[40] SCHANNING (B.). – *Data encryption with public key distribution.* Eascon 79, Conf. Record, p. 653-60, (1979).

[41] KALISKI (B.), RIVEST (R.L.) et SHERMAN (A.T.). – *Is DES a pure cipher.* Proceedings CRYPTO 85, Springer Verlag. LN in Computer Science n° 218, p. 212-26.

[42] SCHEIER (B.). – *Applied cryptography.* John Wiley (1995).

POUR EN SAVOIR PLUS

Conférences

Crypto (USA) - Tous les ans
Eurocrypt (Europe) - Tous les ans
Le lieu et le sujet de ces conférences changent tous les ans.

Organismes officiels

SCSSI Service Central de la sécurité des systèmes d'information
CCITT Comité consultatif international télégraphique et téléphonique (instance normative internationale)