Chapitre 7

Cryptographie et procédés de chiffrement

L'essence même de la cryptographie est de permettre à deux entités de communiquer via un réseau public de sorte qu'un tiers à l'écoute soit dans l'impossibilité de comprendre le contenu des messages échangés. Par réseau public on entend un médium de communication qui ne comporte aucune restriction ni contrôle d'accès. Cela peut être le réseau téléphonique ou Internet par exemple. L'utilisation de moyens cryptographiques doit rendre inexploitables les informations illégitimement recueillies sur le canal public.

L'usage de la cryptographie est fondamental lorsque la confidentialité des communications ne doit souffrir d'aucune faille. C'est le cas par exemple de certains messages à caractère militaire ou, bien entendu, de transactions bancaires dans le cadre d'une activité de commerce électronique. Remarquons cependant que la préservation de la confidentialité des échanges n'est qu'une des fonctionnalités des moyens cryptographiques ; il en existe d'autres comme on le verra dans la suite de ce chapitre mais pour le moment nous nous focalisons sur cet aspect particulier.

D'un point de vue très général, un échange de type cryptographique entre deux personnes, disons Alice et Bob, s'élabore de la façon suivante. Alice, avant de transmettre son information, appelée *message* (ou *texte*) *clair*, à Bob via le réseau, la transforme, par l'emploi d'un *système cryptographique*, en un nouveau message dit *texte chiffré* ou *cryptogramme* dont la propriété essentielle est de dissimuler le sens du message

Chapitre rédigé par Sami HARARI, Laurent POINSOT.

d'origine à tout individu autre que Bob. Celui-ci, interlocuteur légitime d'Alice, récupère et décode le texte chiffré afin de retrouver le message clair qu'Alice souhaitait lui envoyer. Une tierce personne peut évidemment intercepter le message chiffré lors de sa transmission puisque le réseau sur lequel il transite est libre d'accès. Néanmoins, puisque chiffré **pour Bob**, ce message lui paraît être complètement incompréhensible et de fait inutilisable. C'est ainsi qu'en principe la confidentialité est assurée.

Tous les procédés de chiffrement actuels et passés possèdent des caractères communs, une forme similaire, et donc, en quelque sorte, font partie du même *genre*. Aussi la première partie de ce chapitre, telle une classification phylogénétique, est consacrée à la description générale de ces traits partagés : nous y retrouvons donc la définition précise de la notion de cryptosystème et la description des services relevant de l'emploi de moyens cryptographiques. Par ailleurs si la cryptographie existe c'est bien parce qu'un ensemble de menaces pèse sur nos moyens de communication ; c'est la raison pour laquelle le concept de cryptanalyse est aussi introduit dans cette première partie. Bien que d'apparences sensiblement similaires, les cryptosystèmes sont classés de manière systématique au sein de deux *espèces* distinctes : les cryptosystèmes symétriques ou à clef secrète et les cryptosystèmes à clef publique. À chacune de ces espèces est consacrée une partie de ce chapitre. Dans la deuxième partie sont présentés l'architecture générale des cryptosystèmes à clef secrète ainsi que les algorithmes historiquement pertinents tels que le DES, IDEA ou l'AES, l'accent étant mis sur l'évolution des technologies employées. La troisième et dernière partie du présent chapitre détaille RSA, un exemple représentatif de système à clef publique, puis le rôle essentiel joué par la notion de primalité dans les procédés de chiffrement à clef publique est mis en avant par la présentation de différentes techniques permettant d'obtenir des nombres premiers, de factoriser des entiers ou encore de prouver la primalité.

7.1. Principes généraux de la cryptographie

7.1.1. *Les systèmes de chiffrement*

La notion empirique de systèmes cryptographiques peut se modéliser rigoureusement. Nous présentons une définition qui est une légère modification de celle donnée dans [STI 03]. Mathématiquement un *procédé* (ou *algorithme*) *de chiffrement*, aussi appelé *cryptosystème* ou encore *système de chiffrement*[1], se présente comme la donnée de trois ensembles finis et non vides, $\mathcal{M}$, $\mathcal{C}$ et $\mathcal{K}$, respectivement appelés ensemble

1. Dans la suite on s'autorisera à employer indifféremment les expressions « cryptographique » et « de chiffrement » afin de rompre la monotonie.

des *messages clairs*, des *messages chiffrés* et des *clefs secrètes*, et de deux applications[2] $E : \mathcal{K} \to \mathcal{C}^{\mathcal{M}}$, qui à toute clef $k \in \mathcal{K}$ associe une *fonction de chiffrement* $E_k : \mathcal{M} \to \mathcal{C}$, et $D : \mathcal{K} \to \mathcal{M}^{\mathcal{C}}$, qui à toute clef $k \in \mathcal{K}$ associe une *fonction de déchiffrement* $D_k : \mathcal{C} \to \mathcal{M}$, le tout satisfaisant la règle dite de *déchiffrement* : quel que soit $m \in \mathcal{M}$,

$$D_k(E_k(m)) = m \ .$$

Cette règle, ainsi que l'indique son nom, permet de réaliser le décodage d'un message chiffré. En effet supposons que $c \in \mathcal{C}$ soit le message chiffré $E_k(m)$ pour un certain message clair $m \in \mathcal{M}$. La règle de déchiffrement indique que l'on retrouve le message clair originel en calculant $D_k(c) = D_k(E_k(m)) = m$. On déduit de cette formalisation qu'Alice doit avoir connaissance de la fonction E_k et D_k doit être connu de Bob afin que les opérations symétriques de chiffrement et de déchiffrement puissent être réalisées. Si la fonction D_k est portée à la connaissance d'une tierce personne, celle-ci sera apte à retrouver tout message clair chiffré avec la clef k et sera donc en mesure d'écouter de façon illégitime les communications confidentielles entre Alice et Bob.

Exemple 7.1 Le cryptosystème à clef secrète suivant fut inventé par son éponyme Vernam en 1917 et publié en 1926 [VER 26]. Notons $\mathbb{Z}_2$ l'ensemble $\{0, 1\}$ des bits (un bit est un symbole binaire : un « 0 » ou un « 1 ») et $\oplus$ l'addition modulo 2 des bits, aussi appelée *ou-exclusif* ou *XOR*, donnée par la table suivante :

$\oplus$	0	1
0	0	1
1	1	0

Le procédé de chiffrement de Vernam se formalise comme suit : $\mathcal{M} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^\ell$ où ℓ est un entier supérieur ou égal à 1. Les messages clairs, chiffrés et les clefs sont donc des ℓ-uplets de bits. Pour chaque clef $k = (k_1, \ldots, k_\ell)$, on définit :

$$\begin{array}{rccl} E_k : & (\mathbb{Z}_2)^\ell & \to & (\mathbb{Z}_2)^\ell \\ & m = (m_1, \ldots, m_\ell) & \mapsto & m \oplus k = (m_1 \oplus k_1, \ldots, m_\ell \oplus k_\ell) \ . \end{array}$$

Le chiffré par la clef k correspondant au clair m est donc la somme modulo 2 bit-à-bit, que nous conviendrons d'appeler encore, par abus de langage, XOR ou ou-exclusif, de m et de k. La fonction de déchiffrement D_k est identique à E_k. On peut facilement vérifier que la règle de déchiffrement est satisfaite. Notons tout d'abord que quels que

2. Rappelons qu'une application $f : X \to Y$ est une fonction partout définie sur l'ensemble X et à valeurs dans l'ensemble Y. L'ensemble de toutes les applications de X dans Y est noté Y^X.

soient $x, y \in (\mathbb{Z})^\ell$, on a $(x \oplus y) \oplus y = x$. Il s'ensuit que :

$$\begin{aligned} D_k(E_k(m)) &= D_k(m_1 \oplus k_1, \ldots, m_\ell \oplus k_\ell) \\ &= ((m_1 \oplus k_1) \oplus k_1, \ldots, (m_\ell \oplus k_\ell) \oplus k_\ell) \\ &= (m_1, \ldots, m_\ell) \\ &= m\ . \end{aligned}$$

7.1.2. *Deux grandes classes de cryptosystèmes*

Il existe deux grandes classes de cryptosystèmes discriminées par la manière dont est géré le secret sur E_k et D_k.

Les *cryptosystèmes à clef secrète* ou *symétriques* ou *conventionnels* dans lesquels la clef secrète k n'est connue que des seuls Alice et Bob. Les fonctions E_k et D_k sont alors des ressources secrètes partagées par les deux interlocuteurs. L'emploi de la cryptographie conventionnelle nécessite *de facto* un échange de clef secrète, entre Alice et Bob, *a priori* de toutes communications. En clair, Alice et Bob doivent se mettre d'accord sur la clef secrète qu'ils utiliseront. Pour effectuer ce choix, Alice et Bob doivent soit se rencontrer physiquement dans une zone sûre, soit l'un transmet à l'autre la clef sur un canal de communication sécurisé afin d'éviter toute interception par une tierce personne. Le chiffrement de Vernam est un cryptosystème à clef secrète.

L'autre classe de procédés de chiffrement est celle des *cryptosystèmes à clef publique* ou *asymétriques*. La clef k et la fonction D_k sont des secrets détenus par le seul Bob alors que E_k est connue publiquement. Dans ce cadre, si Alice veut communiquer de façon confidentielle avec Bob, elle utilise la fonction publique de chiffrement E_k pour coder les messages. Bob est alors le seul individu apte à décoder puisque personne en dehors de lui ne connaît la fonction de déchiffrement D_k associée.

Dans la suite de ce chapitre nous verrons en détail certains des principaux représentants de ces deux types de cryptosystèmes mais pour l'heure nous poursuivons la présentation générale de la cryptographie.

7.1.3. *Fonctionnalités*

Le cadre d'utilisation de procédés cryptographiques ne se limite pas au chiffrement et donc à la seule garantie de confidentialité des messages. On peut en fait distinguer

quatre fonctionnalités primitives que l'on souhaite assurer par l'emploi d'un moyen cryptographique : confidentialité, authenticité, intégrité et non-répudiation. Chacune de ces fonctionnalités représente une défense, une sécurité contre un certain type d'attaques.

7.1.3.1. *Confidentialité*

La confidentialité permet de garantir qu'un message chiffré n'est compréhensible que par les protagonistes légitimes d'une communication cryptographique. Un attaquant interceptant un message codé doit être dans l'incapacité de le décrypter.

7.1.3.2. *Intégrité*

Dans les communications (chiffrées ou non) on espère évidemment que le destinataire reçoive le message tel qu'il a été envoyé, c'est-à-dire sans modification d'aucune sorte. De plus, si le message reçu est différent du message envoyé, alors son destinataire devrait s'en rendre compte. On résume cela en disant que l'on souhaite garantir l'intégrité des messages contre toutes modifications.

7.1.3.3. *Authenticité*

La mission d'authentification est de garantir que le message reçu provient bien de l'entité qui affirme l'avoir envoyé. Si un ennemi envoie un message à Bob en se faisant passer pour Alice, l'authenticité du message doit être remise en cause par le service d'authentification. Dans le cas contraire, Bob, croyant s'adresser à Alice, pourrait envoyer des informations confidentielles qui tomberaient donc dans des mains ennemies.

7.1.3.4. *Non-répudiation*

La non-répudiation est le moyen qui empêche le destinataire d'un message de contester l'avoir reçu ou l'emetteur de nier l'avoir envoyé. Cette protection est fondamentale par exemple dans le cadre d'une transaction commerciale sur Internet.

Dans ce chapitre nous ne nous étendons que sur la seule notion d'intégrité - qui est fondamentale en cryptologie - et n'abordons pas les autres concepts cryptographiques.

7.1.4. *Cryptanalyse*

Dans le monde de la cryptographie deux catégories d'entités co-existent : les interlocuteurs légitimes d'une communication chiffrée et un adversaire (aussi indifféremment appelé un ennemi, assaillant, attaquant, opposant) qui tente de découvrir la clef employée ; s'il réussit dans sa tâche, on dit alors que l'attaquant a « cassé » ou encore « cryptanalysé » le cryptosystème. On doit donc modéliser les forces de l'assaillant pour mesurer la résistance d'un cryptosystème donné. À ce niveau on fait généralement l'hypothèse que l'attaquant connaît le procédé de chiffrement employé : il s'agit

du *principe de Kerckhoffs*, du nom d'Auguste Kerckhoffs, qui l'a énoncé à la fin XIXe siècle dans un article en deux parties ([KER 83a, KER 83b]).

Il existe une attaque (ou cryptanalyse) de base, dénuée de toute forme de subtilité, dite de *recherche exhaustive* ou *attaque par force brute*, qui permet, en théorie, de casser n'importe quel cryptosystème. Elle consiste simplement à déchiffrer un message codé avec toutes les clefs possibles jusqu'à obtenir un message clair intelligible. En moyenne un attaquant retrouvera la clef au bout de $\frac{|\mathcal{K}|}{2}$ essais[3]. Clairement le nombre de clefs possibles $|\mathcal{K}|$ est le paramètre fondamental pour la résistance contre cette attaque. L'emploi de clefs secrètes de 128 bits, comme spécifié pour de nombreux cryptosystèmes modernes, rend en pratique les procédés de chiffrement peu vulnérables à une recherche exhaustive sachant que 2^{127} est un nombre beaucoup trop élevé.

À l'évidence on rencontre des attaques bien plus sophistiquées dont l'objectif commun est de déterminer la clef utilisée. Les cryptanalyses les plus courantes sont classées selon le pouvoir mis à disposition de l'adversaire. Elles sont énumérées ci-dessous dans l'ordre croissant de puissance de l'attaquant.

1) **Texte chiffré connu** : L'attaquant connaît plusieurs textes chiffrés avec la même clef secrète ;

2) **Texte clair connu** : L'attaquant dispose de textes clairs et des chiffrés correspondant ;

3) **Texte clair choisi** : La fonction de chiffrement E_k est connue de l'attaquant (bien évidemment la clef k, elle, ne l'est pas) et il peut à sa discrétion chiffrer des messages clairs ;

4) **Texte chiffré choisi** : La fonction de déchiffrement D_k est connue de l'opposant qui décode les messages chiffrés selon sa volonté.

Cette classification permet d'établir plusieurs niveaux de robustesse cryptographique. On peut par exemple montrer que le système de Vernam résiste à une attaque à texte chiffré connu mais peut être très facilement cryptanalysé par texte clair connu (supposons que l'on ait connaissance d'un message m et de son chiffré $c = m \oplus k$, alors on récupère k puisque $m \oplus c = k$).

7.1.5. *Critères de résistance*

Il existe trois approches pour mesurer la sécurité d'un procédé de chiffrement. En 1949 Claude Shannon, père fondateur de la cryptographie mathématisée, posa les

3. $|X|$ représente le nombre d'éléments ou *(nombre) cardinal* ou *cardinalité* d'un ensemble fini X.

bases théoriques de la cryptographie moderne dans son article [SHA 49]. Il introduisit alors les deux premiers critères de sécurité : la *confidentialité parfaite* (ou *sûreté inconditionnelle*) et la *sécurité statistique*.

Un cryptosystème est dit *inconditionnellement sûr* ou *assure une confidentialité parfaite* lorsque la connaissance d'un message chiffré n'apporte aucune information sur le clair correspondant. À titre d'exemple on peut montrer que le cryptosystème de Vernam possède la propriété de sûreté inconditionnelle à condition qu'une nouvelle clef secrète soit tirée au hasard à chaque action de chiffrement. Cette propriété très forte assure au passage la non-vulnérabilité aux attaques à texte chiffré connu.

Cependant le système de chiffrement de Vernam est à clef secrète et nécessite donc un échange au préalable de la clef entre les deux interlocuteurs légitimes. Or la confidentialité parfaite les contraint à se transmettre une nouvelle clef à chaque communication chiffrée. On voit donc que cela limite, dans la pratique, l'utilisation d'un tel cryptosystème. Devant la difficulté d'emploi des cryptosystèmes à confidentialité parfaite, Shannon établit un critère de résistance statistique affaibli. Génie visionnaire, il comprit que la *solidité statistique* repose essentiellement sur deux propriétés informelles, la diffusion et la confusion, que doivent posséder les procédés de chiffrement. Sous le terme de *diffusion*, Shannon entend la propriété selon laquelle chaque lettre d'un texte chiffré doit dépendre à la fois de toutes les lettres du texte clair correspondant et de la clef secrète utilisée. Ainsi deux textes chiffrés dont l'un est produit après une modification, même minimale, du texte clair ou de la clef, doivent être radicalement différents. On peut interpréter cela comme une dépendance forte des textes chiffrés aux « conditions initiales ». Le but étant qu'une légère différence en entrée du cryptosystème ait une grande influence sur la valeur du chiffré.
La *confusion* signifie quant à elle qu'un texte clair doit impérativement dépendre de son clair et de la clef d'une manière très complexe de façon à détruire d'éventuelles structures statistiques remarquables au cours du processus de chiffrement, lesquelles sans cela pourraient être utilisées afin de décoder un cryptogramme sans la connaissance de la clef. C'est ainsi par exemple que l'on souhaite masquer les statistiques d'une langue à travers le chiffrement afin de les rendre inexploitables par un adversaire.
Nous verrons que ces deux propriétés déterminent l'architecture des cryptosystèmes symétriques modernes.

La denière approche de la sécurité cryptographique, dite *sécurité calculatoire*, introduite par Diffie et Hellmann dans l'article [DIF 76], concerne les cryptosystèmes à clef publique. On dit qu'un procédé de chiffrement est *calculatoirement sûr* si la meilleure attaque connue pour le casser nécessite un nombre d'opérations beaucoup trop élevé pour être réalisé en pratique. En général on montre que casser un procédé de chiffrement donné est équivalent à la résolution d'un problème réputé être difficile

au sens où la construction explicite d'une solution est impossible en pratique (mais pas nécessairement en théorie !).

7.2. Cryptographie symétrique

7.2.1. *Principe de fonctionnement*

Rappelons brièvement la mise en œuvre d'un procédé de chiffrement conventionnel. En cryptographie symétrique, Alice et Bob et eux seuls connaissent la clef secrète utilisée. Ainsi Alice chiffre son message avec cette clef et l'envoie à Bob. Celui-ci le décode à l'aide de la clef secrète. Un adversaire ayant intercepté le message chiffré ne peut retrouver le texte clair initial car même s'il connaît le cryptosystème — d'après le principe de Kerckhoffs — il ne dispose pas de la clef secrète.
Il est utile de remarquer que le choix de la clef secrète par Alice et Bob est potentiellement délicat. En effet soit ils se rencontrent physiquement soit l'un envoie la clef à l'autre sur un canal de communication. Il faut donc dans ce cas sécuriser la transmission afin d'assurer la confidentialité.

7.2.2. *Chiffrement par blocs et structure itérée*

En pratique le chiffrement s'effectue numériquement. Les messages clairs ou chiffrés sont alors traités comme des blocs de bits de longueur finie. En règle générale, les textes clairs et chiffrés sont des blocs de même longueur. La plupart des algorithmes de chiffrement symétrique possède la même architecture dite *itérée* ; structure permettant d'assurer à la fois la confusion et la diffusion. Ils disposent d'une *fonction de tour* ou de *ronde* T qui s'applique à deux arguments : un message m et une *sous-clef secrète* (encore appelée *clef de tour*) k, qui est aussi un bloc de bits. La sous-clef est obtenue à partir de la clef secrète, dite *clef principale* (ou *clef maîtresse*), à l'aide d'un *algorithme de dérivation* (ou *cadencement* ou encore de *diversification*) *de clef* à propos duquel nous ne nous étendrons pas. À clef de tour k fixée, la fonction $T_k : m \mapsto T(m,k)$ doit être inversible pour permettre le décodage. m est appelé le *message clair du tour courant* et $T(m,k)$ le *chiffré du tour courant*. La fonction de tour est généralement constituée d'une série d'opérations mathématiques suffisamment complexes pour rendre inintelligible le message chiffré $T(m,k)$; cette fonction doit ainsi posséder les propriétés de diffusion et de confusion. En particulier le bloc produit en sortie d'une telle fonction doit dépendre d'un nombre important de bits de clair et de sous-clef, et la complexité des opérations utilisées doit être en mesure de réaliser la confusion. Afin d'accentuer la diffusion et la confusion, la fonction de tour est itérée un certain nombre de fois, disons r, comme suit. Soit m le message à chiffrer.

$$\begin{aligned} m_0 &= m; \\ m_{i+1} &= T_{k_{i+1}}(m_i) \text{ pour } 0 \leq i \leq r-1 \end{aligned}$$

où k_i désigne la sous-clef secrète associée au tour numéro i. Le message chiffré finalement obtenu est le bloc de bits c donné par les égalités suivantes :

$$\begin{aligned} c &= m_r \\ &= T_{k_r}(m_{r-1}) \\ &= T_{k_r} \circ T_{k_{r-1}} \circ \cdots \circ T_{k_2} \circ T_{k_1}(m) \ . \end{aligned}$$

où le symbole « $\circ$ » désigne la composition des applications. L'architecture itérée des procédés de chiffrement par blocs s'impose dès lors que l'on souhaite obtenir de bons niveaux de diffusion et de confusion afin de garantir la sécurité statistique.

Examinons maintenant le processus de déchiffrement. Rappelons que pour une sous-clef de tour k fixée, l'application T_k est inversible, c'est-à-dire qu'il existe une fonction notée T_k^{-1} telle que pour tout bloc x on ait $T_k^{-1}(T_k(x)) = x$. Cette propriété est essentielle pour garantir le décodage. Cette phase s'effectue alors naturellement « en faisant tourner l'algorithme à l'envers » c'est-à-dire en utilisant comme fonction de tour T_k^{-1} au lieu de T_k et en parcourant la suite des clefs de tour en sens inverse. Plus formellement, on obtient le texte clair m à partir du chiffré c comme suit :

$$\begin{aligned} c_0 &= c; \\ c_{i+1} &= T^{-1}_{\widehat{k}_{i+1}}(c_i) \text{ pour } 0 \leq i \leq r-1 \end{aligned}$$

où l'on a noté $\widehat{k}_i$ la sous-clef k_{r+1-i} de sorte que l'on ait :

$$\begin{aligned} \widehat{k}_1 &= k_r \\ \widehat{k}_2 &= k_{r-1} \\ &\cdots \\ \widehat{k}_r &= k_1 \ . \end{aligned}$$

D'après la propriété d'inversibilité de la fonction de tour (à sous-clef fixée), le bloc c_r ainsi calculé correspond bien au message clair original m.

7.2.3. *Algorithmes historiques*

La grande majorité des algorithmes de chiffrement symétrique historiquement pertinents fait usage de cette structure itérée avec quelques modifications mineures en général aux niveaux des premier et dernier tours. Ces cryptosystèmes ne diffèrent ainsi que par la taille des données (message clair et chiffré, clef secrète, sous-clefs secrètes), par le nombre d'itérations (de tours), par la fonction de tour utilisée ainsi que par l'algorithme de dérivation de sous-clef. Nous allons maintenant en présenter trois parmi les plus connus, à savoir le DES, IDEA et l'AES.

7.2.3.1. *DES : Data Encryption Standard*

Le Standard de Chiffrement de Données, le DES, fut développé par la société IBM au milieu des années 1970 et fut adopté en 1977 par le gouvernement fédéral des États-Unis d'Amérique comme standard de chiffrement pour une durée de cinq ans. Il fut reconduit tous les cinq ans, après réévaluation ; le dernier renouvellement datant de 1999.

Le document [FIP 99] décrit entièrement le DES alors que le document [FIP 87] précise ses différents modes d'utilisation. Cet algorithme de chiffrement symétrique opère sur des messages clairs et chiffrés de 64 bits et utilise une clef secrète principale de 64 bits également. En fait, 8 bits de la clef sont utilisés comme bits de parité : le huitième bit de chaque *octet* (huit bits consécutifs) de la clef prend une valeur qui rend pair le nombre de bits valant 1 dans cet octet. Une sous-clef est consituée de 48 bits de la clef principale — hors ceux de parité — disposés dans un certain ordre. Le chiffrement d'un message est obtenu en seize tours.

Étudions maintenant la fonction de tour particulière au DES. Elle est formellement définie comme une *structure* (ou *schéma*) *de Feistel* du nom de son découvreur le cryptologue américain Horst Feistel [FEI 73]. Dans un tel schéma, les blocs ont un nombre pair de bits 2ℓ ($\ell = 32$ dans le cas du DES). On désigne par G les ℓ premiers bits d'un bloc B et par D les ℓ suivants, de telle sorte que l'on a $B = (G, D)$. Soit f une application qui prend deux blocs de bits comme arguments ; la longueur du premier bloc étant ℓ. Cette fonction calcule un bloc également de taille ℓ. La fonction de tour T définie par la structure de Feistel associée à cette fonction f opère sur le bloc (G, D) en permutant la place de G et D, et en transformant G en $f(D, k) \oplus G$ où k désigne la sous-clef du tour courant. Son fonctionnement est décrit plus rigoureusement ci-dessous :

$$\begin{aligned} T(B, k) &= T((G, D), k) \\ &= (D, f(D, k) \oplus G) \end{aligned}$$

On peut facilement démontrer que quelle que soit la fonction f utilisée, la fonction de tour T définie de la sorte est inversible à k fixée. Cette propriété est essentielle car nécessaire à la réalisation du décodage dans un tel cryptosystème. Effectuons donc cette démonstration. On définit l'inverse T_k^{-1} de T_k par :

$$T_k^{-1}(G, D) = (f(G, k) \oplus D, G) \ .$$

Remarquons que si l'on désigne par σ la permutation $\sigma(G, D) = (D, G)$, alors $T_k^{-1} = \sigma \circ T_k \circ \sigma$. Il devient facile de vérifier que $T_k^{-1}(T_k(G, D)) = (G, D)$. Pour cela posons

$G' = D$ et $D' = f(D,k) \oplus G$.

$$\begin{aligned} T_k^{-1}(T_k(G,D)) &= T_k^{-1}(D, f(D,k) \oplus G) \\ &= T_k^{-1}(G', D') \\ &= (f(G',k) \oplus D', G') \\ &= (f(D,k) \oplus (f(D,k) \oplus G), D) \\ &= (G, D). \end{aligned}$$

Un tel schéma de Feistel est donc apte à servir de fonction de tour d'un algorithme de chiffrement symétrique itéré.

Afin de compléter la présentation de la fonction de tour du DES, il nous faut décrire la fonction f particulière à ce procédé de chiffrement. Elle est très importante car c'est à elle que revient la charge d'assurer la confusion du système. Le premier argument de f est un bloc de 32 bits (les 32 premiers ou 32 derniers bits d'un bloc à chiffrer) noté X. Le second argument de cette fonction est une sous-clef, soit ici un bloc de 48 bits que l'on note Y. Le résultat $f(X,Y)$ est lui-même un bloc de 32 bits (d'après les spécifications d'un schéma de Feistel). L'opération qu'effectue f se décompose en quatre étapes :

1) X traverse une fonction E qui prend un bloc de 32 bits en entrée et renvoie un bloc de 48 bits en sortie de sorte que $E(X)$ soit constitué de tous les bits de X — réordonnés — et seize d'entre eux apparaissant deux fois. Les 48 bits de $E(X)$ sont obtenus en sélectionnant les bits de X dans l'ordre donné par la table suivante :

Fonction E

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

Ainsi par exemple les 4 premiers bits de $E(X)$ sont les bits 32, 1, 2 et 3 de X alors que ses 3 derniers bits sont les bits 31, 32 et 1 de X ;

2) Le résultat de $E(X) \oplus Y$ est ensuite calculé et écrit sous la forme d'une juxtaposition de 8 sous-blocs de 6 bits chacun, soit :

$$E(X) \oplus Y = B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8$$

où pour chaque $i \in \{1, \ldots, 8\}$, B_i est de longueur 6 ;

3) Pour chaque $i = 1, \ldots, 8$, B_i traverse une fonction S_i de *substitution* dite *boîte-S* (ou « S-box» en anglais). Une telle boîte prend donc en entrée un bloc de 6 bits et produit un bloc de 4 bits en sortie. Le résultat de cette étape est la concaténation des $S_i(B_i)$, c'est-à-dire le bloc de 32 bits :

$$S = S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8).$$

Chacune des boîtes-S S_i est représentée sous la forme d'un tableau de 4 lignes et 16 colonnes (les lignes étant numérotées de haut en bas de 0 à 3 et les colonnes de gauche à droite de 0 à 15), chaque case contenant un entier compris entre 0 et 15. Par exemple la boîte-S S_1 est donnée par la table suivante :

S_1

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

Étudions en détail la manière dont agit S_i. Soit B_i un bloc de 6 bits. Les premier et dernier bits de B_i représentent, en base deux, un entier compris entre 0 et 3. Soit a cet entier. Les quatre bits du milieu de B_i représentent, en base deux, un entier, que l'on note b, compris entre 0 et 15. L'entier contenu dans la case (a, b) de S_i, c'est-à-dire au niveau de la $a^{\text{ème}}$ ligne et de la $b^{\text{ème}}$ colonne, puisque compris entre 0 et 15, est représentable de manière unique par un bloc de 4 bits. Ce bloc est justement la sortie $S_i(B_i)$ de S_i pour l'entrée B_i. Voici un petit exemple pour clarifier la situation. Supposons que B_1 soit le bloc 011011. Le numéro de la ligne de S_1 est représentée par 01 en base deux, soit $0 \times 2^1 + 1 \times 2^0 = 1$ en base dix. Le numéro de la la ligne de S_1 est représentée par 1101 en base deux, soit $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$ en notation décimale. Il en résulte que $S_1(B_1)$ est l'écriture en base deux de l'entier 5 contenu dans la case disposée sur la ligne 1 et la colonne 13. Or $5 = 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$, ce qui signifie que 0101 représente l'entier 5 écrit en binaire. Finalement $S_1(B_1) = 0101$.
Ces boîtes-S sont en particulier des fonctions non linéaires au sens où en général $S_i(B_i \oplus B_i') \neq S_i(B_i) \oplus S_i(B_i')$. Elles permettent donc de briser la structure algébrique et réalisent de ce fait la confusion au sein du cryptosystème ;

4) À la sortie de ces huit boîtes-S nous disposons donc d'un bloc S de 32 bits. La dernière étape de la fonction f consiste alors à réordonner les bits de S en utilisant

une permutation P. Celle-ci est représentée par la table ci-dessous :

Permutation P

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

La sortie de $P(S)$ est obtenue à partir de S en prenant le seizième bit de S comme premier bit de $P(S)$, le septième bit de S comme deuxième bit de $P(S)$ et ainsi de suite, jusqu'à ce que le vingt-cinquième bit de S soit utilisé comme trente-deuxième bit de $P(S)$. $P(S)$ ainsi constitué est le résultat $f(X, Y)$ de la fonction de tour.

Pour résumer la situation, afin de calculer $f(X, Y)$ on définit d'abord $B_1, \ldots, B_8$ comme des blocs de 6 bits chacun par

$$B_1 B_2 \ldots B_8 = E(X) \oplus Y$$

puis le bloc $f(X, Y)$ est défini par

$$f(X, Y) = P(S_1(B_1) S_2(B_2) \ldots S_8(B_8)).$$

Nous avons vu en détail la fonction de tour de DES, c'est-à-dire le schéma de Feistel et la fonction f. Il nous reste maintenant à décrire le processus complet de chiffrement d'un message par le DES. Avant d'effectuer les 16 itérations de la fonction de tour, le message clair à coder est sujet à la permutation IP, appelée *permutation initiale*, dont le fonctionnement est donné par la table suivante.

Permutation IP

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

Ainsi le bloc permuté a le bit 58 du bloc initial comme premier bit, puis le bit 50 pour son deuxième bit et ainsi de suite. Cette étape initiale est suivie par les 16 itérations de la fonction de tour. Enfin si on désigne par (G_{16}, D_{16}) le bloc de 64 bits produit à l'issue du seizième tour, on termine le chiffrement en appliquant à $\sigma(G_{16}, D_{16}) = (D_{16}, G_{16})$ la permutation IP^{-1} inverse de IP. Nous pouvons donner un version synthétique du processus de codage. Notons k la clef secrète principale et k_i la sous-clef du tour i dérivée de k, et soit $E_k^{\mathbf{DES}}$ la fonction de chiffrement du DES. Le chiffré $E_k^{\mathbf{DES}}(m)$ d'un message clair m est alors calculé comme suit.

$$\begin{array}{lcl} (G_0, D_0) & = & IP(m); \\ (G_{i+1}, D_{i+1}) & = & T_{k_{i+1}}(G_i, D_i) \text{ pour } i = 0, \ldots, 15; \\ E_k^{\mathbf{DES}}(m) & = & IP^{-1}(D_{16}, G_{16}). \end{array}$$

Soit en utilisant une notation plus condensée :

$$E_k^{\mathbf{DES}}(m) = \left(IP^{-1} \circ \sigma \circ T_{k_{16}} \circ \cdots \circ T_{k_1} \circ IP\right)(m).$$

Notons que la permutation finale n'est pas appliquée à (G_{16}, D_{16}) mais effectivement à $\sigma(G_{16}, D_{16})$ soit encore (D_{16}, G_{16}). Cette permutation, IP^{-1}, est l'inverse de la permutation IP. Par conséquent pour décoder il est simplement nécessaire d'appliquer exactement le même algorithme de chiffrement à $E_k^{\mathbf{DES}}$ en parcourant les sous-clefs de k_{16} à k_1. En d'autres termes la fonction de déchiffrement est définie par

$$D_k^{\mathbf{DES}}(c) = (IP^{-1} \circ \sigma \circ T_{k_1} \circ \cdots \circ T_{k_{16}} \circ IP)(c)\ .$$

Pour s'en persuader il suffit de remarquer que puisque $T_k^{-1} = \sigma \circ T_k \circ \sigma$ et que quel que soit (G, D), $(\sigma \circ \sigma)(G, D) = (G, D)$, alors on a :

$D_k^{\mathbf{DES}}(E_k^{\mathbf{DES}}(m)) =$
$(IP^{-1} \circ \sigma \circ T_{k_1} \circ \cdots \circ T_{k_{16}} \circ IP)(E_k^{\mathbf{DES}}(m)) =$
$IP^{-1} \circ (\sigma \circ T_{k_1} \circ \sigma) \circ \cdots \circ (\sigma \circ T_{k_{16}} \circ \sigma) \circ \sigma \circ IP(E_k^{\mathbf{DES}}(m)) =$
$IP^{-1} \circ T_{k_1}^{-1} \circ \cdots \circ T_{k_{16}}^{-1} \circ \sigma \circ IP(E_k^{\mathbf{DES}}(m)) =$
$IP^{-1} \circ T_{k_1}^{-1} \circ \cdots \circ T_{k_{16}}^{-1} \circ \sigma \circ IP \circ IP^{-1} \circ \sigma \circ T_{k_{16}} \circ \cdots \circ T_{k_1} \circ IP(m) = m$
(en éliminant deux à deux une fonction et son inverse).

Le document [FIP 99] contient aussi la description d'un autre algorithme de chiffrement symétrique, le TDEA (Triple Data Encryption Algorithm), encore appelé le *triple DES*. Il s'agit d'une utilisation itérée du DES original que l'on présente brièvement. Soient $k^{(1)}$, $k^{(2)}$ et $k^{(3)}$ trois clefs secrètes principales. Soit m un bloc de 64 bits à coder.

1) L'opération de chiffrement : on transforme le bloc m en un nouveau bloc c de 64 bits comme suit :

$$c = E_{k^{(3)}}^{\mathbf{DES}}(D_{k^{(2)}}^{\mathbf{DES}}(E_{k^{(1)}}^{\mathbf{DES}}(m)));$$

2) L'opération de déchiffrement : on retrouve le bloc m à partir de son cryptogramme c de la manière suivante :

$$m = D_{k^{(1)}}^{\mathbf{DES}}(E_{k^{(2)}}^{\mathbf{DES}}(D_{k^{(3)}}^{\mathbf{DES}}(c)))$$

7.2.3.2. *IDEA : International Data Encryption Algorithm*

L'algorithme IDEA, inventé par Xuejia Lai et James L. Massey, est décrit dans [LAI 90] et [LAI 92]. Il est protégé par un brevet, est commercialisé par la société suisse MediaCrypt.

IDEA fut explicitement conçu par ses auteurs pour réaliser concrètement les exigences de diffusion et de confusion. Comme DES, il met en œuvre une structure itérée, cependant il n'utilise pas la technologie des schémas de Feistel mais une autre méthode permettant de produire des fonctions inversibles, propriété essentielle pour le déchiffrement. La fonction de tour d'IDEA repose sur l'exploitation d'opérations mathématiques intervenant dans certaines structures algébriques : les *groupes*. Une *loi de composition interne*, notée $*$, sur un ensemble E est une application qui à un couple (x, y) d'éléments de E associe un élément z de E que l'on note $x * y$. Un *groupe* est la donnée d'un ensemble non vide G et d'une loi de composition interne $*$ sur G vérifiant les axiomes suivants :

1) Associativité : quels que soient x, y, z dans G, $x * (y * z) = (x * y) * z$;

2) Élément neutre : il existe un élément distingué e dans G tel que quel que soit x dans G, on ait $x * e = e * x = x$;

3) Inverse : pour tout x dans G il existe un élément noté x^{-1} dans G tel que $x * x^{-1} = x^{-1} * x = e$.

Par exemple si p est un nombre *premier* — c'est-à-dire un entier naturel différent de 1 qui n'est divisible que par 1 et lui-même (tel que $2, 3, 5, 7, 11$, etc.) — la multiplication modulo p des entiers non nuls est une loi de groupe sur l'ensemble $\{1, 2, \ldots, p-1\}$. Il en est de même pour un entier naturel n, différent de 0, de l'addition modulo n sur l'ensemble $\{0, 1, \ldots, n-1\}$. Enfin l'ensemble des blocs de n bits muni de l'addition modulo 2 bit-à-bit, c'est-à-dire le XOR, est un autre exemple de groupe. Ces trois structures sont justement celles mises en œuvre dans le fonctionnement d'IDEA.

Afin de présenter la fonction de tour d'IDEA, nous employons les notations suivantes. Soit n un entier tel que $2^n + 1$ soit un entier premier (par exemple $n = 1, 2, 4, 8, 16$).

– L'opération XOR entre deux blocs de 2^n bits est dénotée comme d'habitude par le symbole « $\oplus$ ». Pour $n = 2$, on a par exemple $(0, 1, 1, 0) \oplus (1, 1, 0, 1) = (1, 0, 1, 1)$;

– Chaque bloc de 2^n bits peut être identifié à un et un seul entier entre 0 et $2^n - 1$ en notation binaire. On peut donc réaliser l'addition modulo 2^n de ces blocs via cette identification. Cette opération est notée par « $\boxplus$ ». Pour $n = 4$, $(0, 1, 1, 0)$ représente l'entier 6 et $(1, 1, 0, 1)$ l'entier 13. L'addition modulo $2^4 = 16$ de 6 et 13 donne 3, représenté en binaire par $(0, 0, 1, 1)$. Il résulte de cela que $(0, 1, 1, 0) \boxplus (1, 1, 0, 1) = (0, 0, 1, 1)$;

– Chaque bloc de 2^n bits non tous nuls représente de manière unique un entier entre 1 et $2^n - 1$. Le bloc dans lequel tous les bits valent 0 est quant à lui identifié au nombre 2^n. Comme $2^n + 1$ est supposé être premier, l'ensemble $\{1, 2, \ldots, 2^n\}$ équipé de la multiplication des entiers modulo $2^n + 1$ est un groupe. Au travers de l'identification on peut effectuer cette multiplication, notée par « $\odot$ », entre deux blocs de bits. C'est ainsi que l'on a $(0,1,1,0) \odot (1,1,0,1) = (1,0,1,0)$ puisque en effet 6 multiplié par 13 modulo $2^4 + 1 = 17$ est égal à 10, représenté par $(1,0,1,0)$ en base deux.

Nous sommes maintenant en mesure de décrire la fonction de tour d'IDEA. Cet algorithme manipule des blocs de 64 bits pour les messages clairs et chiffrés et utilise une clef principale de 128 bits. Le processus de dérivation produit à chaque tour, à partir de cette clef maîtresse, des sous-clefs de 96 bits chacune. À la ronde numéro i, le bloc en entrée m_{i-1} de la fonction de tour est divisé en quatre sous-blocs de 16 bits alors que la sous-clef k_i est partitionnée en 6 sous-blocs de 16 bits. Ainsi $m_{i-1} = m_{i-1}^1\ m_{i-1}^2\ m_{i-1}^3\ m_{i-1}^4$ et $k_i = k_i^1\ k_i^2\ k_i^3\ k_i^4\ k_i^5\ k_i^6$ où m_{i-1}^j et k_i^l sont des blocs de 16 bits chacun pour chaque $j = 1,2,3,4$ et chaque $l = 1,2,3,4,5,6$. Notons ici que $2^{16} + 1 = 65537$ est un entier premier. On peut donc utiliser les lois de groupe définies précédemment sur des blocs de 16 bits. La fonction de tour utilise une opération MA, dite de *multiplication-addition* ou *structure-MA*, qui prend quatre blocs x_1, x_2, y_1, y_2 de 16 bits chacun en entrée et produit deux blocs, notés $MA_1(x_1,x_2,y_1,y_2)$ et $MA_2(x_1,x_2,y_1,y_2)$, de 16 bits également. Les relations mathématiques entre les entrées et les sorties de la fonction MA sont données ci-dessous :

$$\begin{array}{lcl} MA(x_1,x_2,y_1,y_2) & = & MA_1(x_1,x_2,y_1,y_2)\ MA_2(x_1,x_2,y_1,y_2) \\ MA_1(x_1,x_2,y_1,y_2) & = & MA_2(x_1,x_2,y_1,y_2) \boxplus (x_1 \odot y_1) \\ MA_2(x_1,x_2,y_1,y_2) & = & ((x_1 \odot y_1) \boxplus x_2) \odot y_2 \end{array}$$

où le second membre de la première égalité représente la juxtaposition des blocs $MA_1(x_1,x_2,y_1,y_2)$ et $MA_2(x_1,x_2,y_1,y_2)$. La structure MA est donc composée d'une utilisation entremêlée de deux des trois opérations de groupe, la multiplication $\odot$ modulo $2^{16} + 1$ et l'addition $\boxplus$ modulo 2^{16}.
La structure MA joue le même rôle que la fonction f du DES dans l'établissement de la confusion et de la diffusion, mais contrairement à cette dernière, la structure MA est inversible lorsque sont fixées les entrées y_1 et y_2. À partir de la connaissance des sorties $z_1 = MA_1(x_1,x_2,y_1,y_2)$, $z_2 = MA_2(x_1,x_2,y_1,y_2)$ de la fonction MA ainsi que de y_1, y_2, on est en mesure de retrouver x_1, x_2. De l'équation $z_1 = MA_2(x_1,x_2,y_1,y_2) \boxplus (x_1 \odot y_1)$ on peut en déduire la valeur de x_1. Notons en effet a^{-1} (respectivement $-a$) l'inverse de a par rapport à la loi $\odot$ (respectivement $\boxplus$).

$$\begin{array}{lcl} z_1 & = & z_2 \boxplus (x_1 \odot y_1) \\ \Leftrightarrow\ -z_2 \boxplus z_1 & = & x_1 \odot y_1 \\ \Leftrightarrow\ (-z_2 \boxplus z_1) \odot y_1^{-1} & = & x_1. \end{array}$$

Puis en injectant cette valeur pour x_1 dans l'équation $z_2 = ((x_1 \odot y_1) \boxplus x_2) \odot y_2$, on récupère la valeur de x_2. En effet,

$$\begin{array}{lll} & z_2 & = & ((x_1 \odot y_1) \boxplus x_2) \odot y_2 \\ \Leftrightarrow & z_2 \odot y_2^{-1} & = & (x_1 \odot y_1) \boxplus x_2 \\ \Leftrightarrow & -(x_1 \odot y_1) \boxplus (z_2 \odot y_2^{-1}) & = & x_2 \\ \Leftrightarrow & -(((-z_2 \boxplus z_1) \odot y_1^{-1}) \odot y_1) \boxplus (z_2 \odot y_2^{-1}) & = & x_2. \end{array}$$

IDEA n'utilise pas de structure de Feistel, qui *de facto* est inversible, néanmoins nous allons voir plus loin que la fonction de tour d'IDEA est aussi, par construction, inversible. Il s'avère cependant que l'inversibilité de la structure MA ne joue en fait aucun rôle dans le processus de déchiffrement.

Examinons maintenant, avec précision, une ronde quelconque de IDEA. La ronde numéro i produit un bloc c_i de 64 bits divisé en quatre blocs de 16 bits chacun, que l'on convient de noter c_i^1, c_i^2, c_i^3 et c_i^4 de telle sorte que $c_i = c_i^1\ c_i^2\ c_i^3\ c_i^4$. Mathématiquement un tour de IDEA est donné par les formules ci-dessous

$$\begin{array}{lll} c_i^1 & = & MA_2 \oplus (m_{i-1}^3 \boxplus k_i^3); \\ c_i^2 & = & MA_1 \oplus (m_{i-1}^4 \boxplus k_i^4); \\ c_i^3 & = & MA_2 \oplus (m_{i-1}^1 \odot k_i^1); \\ c_i^4 & = & MA_1 \oplus (m_{i-1}^2 \odot k_i^2) \end{array} \tag{7.1}$$

où l'on a posé

$$\begin{array}{lll} MA_1 & = & MA_1((m_{i-1}^1 \odot k_i^1) \oplus (m_{i-1}^3 \boxplus k_i^3), (m_{i-1}^2 \odot k_i^2) \oplus (m_{i-1}^4 \boxplus k_i^4), k_5^i, k_6^i); \\ MA_2 & = & MA_2((m_{i-1}^1 \odot k_i^1) \oplus (m_{i-1}^3 \boxplus k_i^3), (m_{i-1}^2 \odot k_i^2) \oplus (m_{i-1}^4 \boxplus k_i^4), k_i^5, k_i^6). \end{array} \tag{7.2}$$

Le processus de chiffrement d'IDEA est constitué d'une séquence de 8 telles rondes où la sortie c_i du tour i est choisie comme entrée du tour suivant. Le cryptogramme correspondant au clair $m = m_0$ n'est pas le bloc c_8 produit au terme de la huitième ronde. En effet il y a l'étape finale suivante : on calcule le chiffré $c_9 = c_9^1\ c_9^2\ c_9^3\ c_9^4$ par

$$\begin{array}{lll} c_9^1 & = & c_8^1 \odot k_9^1; \\ c_9^2 & = & c_8^2 \odot k_9^2; \\ c_9^3 & = & c_8^3 \boxplus k_9^3; \\ c_9^4 & = & c_8^4 \boxplus k_9^4 \end{array} \tag{7.3}$$

où $k_9 = k_9^1\ k_9^2\ k_9^3\ k_9^4$ est une sous-clef de 64 bits (chacun des k_9^i étant composé de 16 bits), elle aussi issue du processus de diversification à partir de la clef principale.

Étudions le processus de déchiffrement. Regardons tout d'abord comment retrouver les entrées m_{i-1}^j pour $j = 1, \ldots, 4$ de la ronde i (pour $1 \leq i \leq 8$) à partir des sorties

c_i^j et des sous-clefs k_i^l ($l = 1, \ldots, 6$). Rappelons tout d'abord que l'inverse d'un bloc x de bits pour la loi de groupe $\oplus$ est x lui-même. En particulier $x \oplus x$ est égal au bloc dans lequel tous les bits valent zéro, élément neutre pour la loi $\oplus$. À partir des définitions de c_i^j données par les égalités (7.1), on peut vérifier les résultats suivants.

$$\begin{array}{rrcl} & c_i^1 \oplus c_i^3 & = & MA_2 \oplus (m_{i-1}^3 \boxplus k_i^3) \oplus MA_2 \oplus (m_{i-1}^1 \odot k_i^1) \\ \Leftrightarrow & c_i^1 \oplus c_i^3 & = & (m_{i-1}^3 \boxplus k_i^3) \oplus (m_{i-1}^1 \odot k_i^1). \end{array}$$

De même

$$\begin{array}{rrcl} & c_i^2 \oplus c_i^4 & = & MA_1 \oplus (m_{i-1}^4 \boxplus k_i^4) \oplus MA_1 \oplus (m_{i-1}^2 \odot k_i^2) \\ \Leftrightarrow & c_i^2 \oplus c_i^4 & = & (m_{i-1}^4 \boxplus k_i^4) \oplus (m_{i-1}^2 \odot k_i^2). \end{array}$$

On remarque alors que $(c_i^1 \oplus c_i^3)$ (respectivement $c_i^2 \oplus c_i^4$) est le premier (respectivement deuxième) argument de la fonction MA d'après les égalités (7.2). Il résulte de cela que la connaissance des c_i^j et de k_i^5, k_i^6 nous permet de recalculer MA_1 et MA_2. Enfin en utilisant les formules (7.1) on est en mesure de déduire les entrées du tour numéro i à savoir les m_{i-1}^j puisque l'on connaît aussi les sous-clefs k_i^l. Par exemple,

$$\begin{array}{rrcl} & c_i^1 & = & MA_2 \oplus (m_{i-1}^3 \boxplus k_i^3) \\ \Leftrightarrow & c_i^1 \oplus MA_2 & = & m_{i-1}^3 \boxplus k_i^3 \\ \Leftrightarrow & (c_i^1 \oplus MA_2) \boxplus (-k_i^3) & = & m_{i-1}^3. \end{array}$$

Il nous reste enfin à étudier comment, à partir des c_9^j et des sous-clefs $k_9^1, k_9^2, k_9^3, k_9^4$, on retrouve $c_8 = c_8^1\ c_8^2\ c_8^3\ c_8^4$. Pour ce faire on utilise les égalités (7.3). On montre, après quelques calculs faciles, que l'on a :

$$\begin{array}{rcl} c_9^1 \odot (k_9^1)^{-1} & = & c_8^1; \\ c_9^2 \odot (k_9^2)^{-1} & = & c_8^2; \\ c_9^3 \boxplus (-k_9^3) & = & c_8^3; \\ c_9^4 \boxplus (-k_9^4) & = & c_8^4. \end{array}$$

Ainsi que cela a été précisé précédemment, la propriété d'inversibilité de la structure MA n'intervient pas dans le processus de déchiffrement. En d'autres termes, si f est une fonction quelconque prenant 4 blocs de 16 bits en entrée et produisant deux blocs de 16 bits en sortie, le cryptosystème obtenu en remplaçant dans l'algorithme d'IDEA la structure MA par f reste inversible et permet donc le décodage des cryptogrammes. On peut alors s'interroger sur le choix et l'utilité de cette fonction. En fait MA participe activement à la réalisation de la diffusion au cours du processus de chiffrement. En effet, on peut montrer que chacun des sous-blocs en sortie de MA dépend de tous les sous-blocs en entrée, ce qui permet d'assurer la diffusion, en un nombre de rondes moindre que dans le DES. La confusion quant à elle est obtenue par le mélange des trois opérations de groupes qui sont, dans un sens bien précis, mutuellement incompatibles entre elles et permettent ainsi de cacher les structures algébriques utilisées. En effet on peut par exemple citer les propriétés suivantes. Soit $\#$ et $\bigstar$ deux

opérations distinctes parmi $\oplus$, $\boxplus$ et $\odot$ (par exemple $\#$ est $\oplus$ et $\bigstar$ est $\boxplus$ ou encore $\#$ est $\boxplus$ et $\bigstar$ est $\odot$).

1) Aucune paire d'opérations $\#$, $\bigstar$ ne satisfait la loi de distributivité, c'est-à-dire qu'il existe nécessairement trois blocs x, y, z de 16 bits chacun tels que :

$$x\#(y\bigstar z) \neq (x\#y)\bigstar(x\#z);$$

2) Aucune paire d'opérations $\#$, $\bigstar$ ne satisfait la loi d'associativité, c'est-à-dire qu'il existe nécessairement trois blocs x, y, z de 16 bits chacun tels que :

$$x\#(y\bigstar z) \neq (x\#y)\bigstar z.$$

7.2.3.3. *AES : Advanced Encryption Standard*

Le 2 septembre 1997 le NIST (National Institute of Standards and Technology), agence du département américain du commerce, organisa un appel d'offre public pour le remplacement du DES comme standard de chiffrement pour les agences fédérales des États-Unis d'Amérique. Le cahier des charges était grossièrement le suivant : développer un algorithme de chiffrement symétrique, de nom de code AES (Advanced Encryption Standard), supportant des blocs de taille 128 bits et des clefs de longueurs 128, 192 et 256 bits. Le 20 août 1998, le NIST annonça la candidature de quinze algorithmes provenant de douze pays. Un an plus tard, après une analyse détaillée des candidats, le NIST n'en retint plus que cinq, à savoir MARS, RC6, Rijndael, Serpent et Twofish. Une seconde étape d'étude de la sécurité des finalistes fut alors menée. Elle ne fut pas conduite par le seul NIST mais par une grande partie de la communauté cryptographique internationale. Il est remarquable de noter qu'aucune attaque en mesure de mettre en défaut de manière significative la sécurité de l'un des finalistes ne fut découverte à cette occasion [NEC 01]. Cependant d'autres critères, comme la complexité algorithmique ou les caractéristiques d'implantation, permirent, en 2000, de choisir l'algorithme Rinjdael, conçu par deux cryptologues belges, Joan Daemen et Vincent Rijmen, comme standard de chiffrement avancé ou AES. Le document officiel [FIP 01], daté du 26 novembre 2001, approuva l'AES comme moyen cryptographique de protection des données électroniques sensibles (non classifiées) des agences fédérales et des départements du gouvernement américain. Dans ce même document sont présentées en détail les spécifications complètes de l'AES.

L'AES est capable d'utiliser des clefs de 128, 192 ou 256 bits pour coder et décoder des blocs de données de 128 bits. Le choix des clefs dépend du niveau de protection que l'on souhaite attribuer aux documents ; c'est ainsi que dans une note du gouvernement américain [NSA 03], l'agence nationale de sécurité (NSA) recommande l'emploi des clefs de 192 ou 256 bits pour des documents top-secrets. Notons au passage que l'algorithme originel Rinjdael est conçu pour manipuler d'autres longueurs additionnelles de blocs et de clefs, mais celles-ci ne sont pas adoptées dans le standard

qu'est l'AES. Comme son prédesseur, le DES, l'AES opère sur un certain nombre de tours qui dépend de la taille choisie pour la clef comme le décrit le tableau suivant.

Clef	Nombre de tours
128	10
192	12
256	14

Contrairement au DES et à IDEA, l'unité de base sur laquelle opère l'AES n'est pas le bit mais l'octet. Ainsi les messages clairs et chiffrés sont vus comme des blocs de 16 octets. Plus précisément les messages à l'entrée et à la sortie d'une ronde de l'AES, appelés *états*, sont représentés non comme une chaîne de 16 octets mais comme une matrice — un tableau — de quatre lignes et quatre colonnes dont chacune des cases contient un octet. Ainsi un état est donné sous la forme suivante.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

où $a_{i,j}$ est un octet. Un tel état représente le bloc

$$a_{0,0}\ a_{1,0}\ a_{2,0}\ a_{3,0}\ a_{0,1}\ a_{1,1}\ a_{2,1}\ a_{3,1}\ \ldots\ a_{0,3}\ a_{1,3}\ a_{2,3}\ a_{3,3}.$$

Quelle que soit la taille choisie pour la clef secrète, les sous-clefs de tour sont aussi représentées par de telles matrices d'octets de taille 4×4 et sont donc constituées de 128 bits.
Avant d'entrer dans le détail de l'architecture de la fonction de tour, il est nécessaire d'introduire des notions mathématiques sur lesquelles s'appuie le fonctionnement de l'AES. Pour la présentation de ces notions nous reprenons les notations officielles, ainsi que certains exemples, du document [DAE 99] disponible à l'URL :

`http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf`.

L'AES met en œuvre des opérations définies sur un corps fini. Un *corps* (commutatif) $\mathbb{K}$ est un ensemble possédant deux éléments distincts et distingués, notés 0 et 1, qui est muni de deux lois de composition interne, notées $+$ et $\times$ (la seconde loi pourra aussi être notée par juxtaposition comme une multiplication classique), pour lesquels on a :

1) $\mathbb{K}$ muni de $+$ est un groupe dont l'élément neutre est 0 (l'inverse de x pour cette loi est noté $-x$, c'est-à-dire que $x + (-x) = (-x) + x = 0$) ;

2) L'*addition* $+$ est commutative, c'est-à-dire quels que soient x, y dans $\mathbb{K}$, $x+y = y + x$;

3) L'ensemble $\mathbb{K}^*$ des éléments de $\mathbb{K}$ qui sont différents de 0 est un groupe lorsqu'on le munit de la *multiplication* $\times$ et son élément neutre est 1 (l'inverse de x non nul dans $\mathbb{K}$ pour la multiplication est noté x^{-1} et donc $xx^{-1} = x^{-1}x = 1$) ;

4) 0 est absorbant pour la multiplication : quel que soit x dans $\mathbb{K}$, $x0 = 0x = 0$;

5) La multiplication est également commutative : $xy = yx$ quels que soient x, y dans $\mathbb{K}$;

6) La multiplication se distribue sur l'addition : $x(y + z) = xy + xz$ quels que soient x, y, z dans $\mathbb{K}$.

Parmi les corps il en est dont la cardinalité est infinie et d'autres, plus intéressants d'un point de vue pratique pour la cryptographie, qui n'ont qu'un nombre fini d'éléments. Assez naturellement ces derniers sont appelés les *corps finis*. Un octet peut être représenté comme un élément du corps fini $\mathsf{GF}(2^8)$ à $2^8 = 256$ éléments (« GF » est une abbréviation du terme anglais « Galois Field » signifiant « corps de Galois »). Ainsi un état peut être identifié à un tableau de taille 4×4 dont chaque case contient un élément du corps fini à 256 éléments.

Nous sommes maintenant en mesure de détailler le fonctionnement d'une ronde de l'AES. La fonction de tour est constituée de quatre applications inversibles ; les trois premières étant indépendantes de la sous-clef de tour, alors que la quatrième est justement l'addition case-par-case entre les octets de l'état courant et ceux de la sous-clef. Chacune de ces opérations agit de manière spécifique sur un état et favorise la diffusion. Un tour s'effectue donc en quatre étapes de la manière suivante :

1) La fonction `ByteSub` (de l'Anglais *Byte Substitution* soit en français *substitution d'octet*) est appliquée à la matrice A représentant l'état en entrée du tour. Cette fonction agit indépendamment sur chacune des cases de A via une transformation inversible $\mathsf{S}_{\mathtt{RD}} : \mathsf{GF}(2^8) \to \mathsf{GF}(2^8)$. Cette fonction est essentiellement définie par :

$$\mathsf{inv} : x \mapsto \begin{cases} 0 & \text{si } x = 0, \\ x^{-1} & \text{si } x \neq 0 \end{cases}$$

où x est un octet vu comme un élément de $\mathsf{GF}(2^8)$. Plus précisément $\mathsf{S_{RD}} = \lambda \circ \mathsf{inv}$ où λ est une transformation des octets qui est *affine* et inversible ; affine signifiant dans ce contexte qu'il existe une application α, telle que quels que soient les octets x, y, $\alpha(x + y) = \alpha(x) + \alpha(y)$ (on dit que α est *linéaire*), et un élément β de $\mathsf{GF}(2^8)$ tels que $\lambda(x) = \alpha(x) + \beta$. L'application linéaire α est également inversible, et on a $\lambda^{-1}(x) = \alpha^{-1}(x - \beta)$. Ainsi l'inverse de $\mathsf{S_{RD}}$ est obtenu par ${\mathsf{S_{RD}}}^{-1} = \mathsf{inv} \circ \lambda^{-1}$ puisque, comme il est facile de le vérifier, inv est son propre inverse. Graphiquement

ByteSub se comporte comme suit :

$$\texttt{ByteSub}\left[\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}\right] = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}$$

où chaque $b_{i,j} = \mathsf{S_{RD}}(a_{i,j})$.
Des résultats d'ordre théorique indiquent que cette fonction permet d'obtenir un bon niveau de confusion [DAE 02] ; en particulier $\mathsf{S_{RD}}$, tout comme les boîtes-S du DES, est non linéaire (au niveau des octets) : en général $\mathsf{S_{RD}}(x+y) \neq \mathsf{S_{RD}}(x) + \mathsf{S_{RD}}(y)$. Cette première étape permet donc de « disloquer » la structure de groupe de l'ensemble $\mathsf{GF}(2^8)$ muni de l'addition. Elle prend donc une part essentielle dans la réalisation de la confusion au sein du cryptosystème. Notons que la fonction ByteSub est inversible : pour retrouver une entrée A à partir de $B = \mathsf{ByteSub}(A)$, il suffit d'appliquer sur chacune des cases de B la transformation ${\mathsf{S_{RD}}}^{-1}$ inverse de $\mathsf{S_{RD}}$; en d'autres termes on applique simplement à B la transformation ByteSub dans laquelle on a remplacé $\mathsf{S_{RD}}$ par ${\mathsf{S_{RD}}}^{-1}$;

2) Les lignes de la matrice $B = \mathsf{ByteSub}(A)$, sortie de la transformation ByteSub, subissent chacune un décalage (*shift* en anglais) vers la gauche du contenu de leurs cases. Ces décalages sont *cycliques* : par exemple après décalage de deux crans sur la gauche de la ligne

x_1	x_2	x_3	x_4

on obtient la ligne

x_3	x_4	x_1	x_2

Cette opération sur les lignes de l'état courant est appelée ShiftRow. Le décalage appliqué à une ligne dépend de son numéro. La première ligne n'est ainsi pas décalée alors que la deuxième est décalée d'un cran sur la gauche, la troisième de deux et la quatrième de trois. Graphiquement ShiftRow opère comme suit sur la matrice B dont les éléments ont été notés $b_{i,j}$.

$$\texttt{ShiftRow}\left[\begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}\right] = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{pmatrix}.$$

À l'instar de la première transformation, ByteSub, ShiftRow est inversible. Pour retrouver la matrice avant décalage à partir de la matrice décalée il suffit d'appliquer la transformation ShiftRow en changeant comme suit la valeur des décalages appliqués : la première ligne n'est toujours pas décalée, la deuxième est décalée maintenant de trois crans sur la gauche, la troisième de deux et la quatrième d'un seul. Cette transformation a pour but de favoriser la diffusion ;

3) La troisième étape de la ronde est une fonction qui opère au niveau des colonnes de l'état courant. Il s'agit d'une multiplication matricielle d'une matrice inversible M, de 4 lignes et 4 colonnes, par chacune des colonnes de l'état. Voyons plus précisément le fonctionnement de cette opération notée $\mathsf{MixColumn}$. Soit $C = \mathsf{ShiftRow}(B)$ l'état courant, ses cases étant notées $c_{i,j}$ ($i = 0, \ldots, 3$, $j = 0, \ldots, 3$). Notons C_0, C_1, C_2, C_3 les quatres colonnes de C de telle sorte que l'on peut voir C comme la juxtaposition, notée $[C_0 \mid C_1 \mid C_2 \mid C_3]$, de ses colonnes. Il s'ensuit que la colonne j (pour $j = 0, \ldots, 3$) est de la forme :

$$C_j = \begin{pmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{pmatrix}.$$

Si l'on multiplie la matrice M (dont les éléments, les $m_{i,j}$, appartiennent à $\mathsf{GF}(2^8)$) par la colonne C_j on obtient une autre colonne $D_j = \begin{pmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{pmatrix}$. On a donc $D_j = \mathsf{M}C_j$, soit en notation matricielle :

$$D_j = \mathsf{M}C_j \Leftrightarrow \begin{pmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{pmatrix} = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \begin{pmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{pmatrix}.$$

La définition d'un produit matriciel conduit au résultat suivant quant aux valeurs des $d_{i,j}$ pour $i = 0, \ldots, 3$.

$$d_{i,j} = m_{i,0}c_{0,j} + m_{i,1}c_{1,j} + m_{i,2}c_{2,j} + m_{i,3}c_{3,j}.$$

La multiplicaton matricielle de chacune des colonnes de l'état C produit ainsi quatre colonnes D_0, D_1, D_2, D_3. Il en résulte que l'on a construit la matrice D de 4 lignes et 4 colonnes constituée par juxtaposition $D = [D_0 \mid D_1 \mid D_2 \mid D_3]$. En résumé, la transformation $\mathsf{MixColumn}$ est formellement définie par

$$D = \mathsf{MixColumn}(C) = [\mathsf{M}C_0 \mid \mathsf{M}C_1 \mid \mathsf{M}C_2 \mid \mathsf{M}C_3].$$

Nous avions précisé que la matrice M est *inversible*. Cela signifie qu'il existe une autre matrice de 4 lignes et 4 colonnes, qui est assez naturellement notée M^{-1}, telle que quelle que soit la matrice X constituée d'une seule colonne et de quatre lignes (donc du même format que l'une des colonnes C_j par exemple) on ait :

$$\mathsf{M}^{-1}(\mathsf{M}X) = X. \tag{7.4}$$

En d'autres termes si l'on multiplie M^{-1} par la colonne résultant du produit $\mathsf{M}X$ on obtient à nouveau X. Cette propriété permet d'inverser l'opération $\mathsf{MixColumn}$.

En effet notons $\mathsf{MixColumn}^{-1}$ l'opération MixColumn dans laquelle la matrice M utilisée pour réaliser la multiplication matricielle est remplacée par M^{-1}. Vérifions qu'en appliquant $\mathsf{MixColumn}^{-1}$ à $D = \mathsf{MixColumn}(C)$ on récupère C.

$$\begin{aligned}
\mathsf{MixColumn}^{-1}(D) &= [\mathsf{M}^{-1}D_0 \mid \mathsf{M}^{-1}D_1 \mid \mathsf{M}^{-1}D_2 \mid \mathsf{M}^{-1}D_3] \\
&\quad \text{(puisque } D = [D_0 \mid D_1 \mid D_2 \mid D_3]\text{)} \\
&= [\mathsf{M}^{-1}(\mathsf{M}C_0) \mid \mathsf{M}^{-1}(\mathsf{M}C_1) \mid \mathsf{M}^{-1}(\mathsf{M}C_2) \mid \mathsf{M}^{-1}(\mathsf{M}C_3)] \\
&\quad \text{(car } D = \mathsf{MixColumn}(C)\text{)} \\
&= [C_0 \mid C_1 \mid C_2 \mid C_3] \\
&\quad \text{(par la propriété (7.4)).}
\end{aligned}$$

4) Pour clore un tour de l'AES, on effectue une addition de $\mathsf{GF}(2^8)$ case-par-case entre le résultat D de MixColumn, obtenu à l'étape précédente, et la sous-clef de tour k, représentée, comme nous l'avons précisé, sous la forme d'une matrice d'octets de 4 lignes et 4 colonnes. La traduction de ceci en représentation matricielle est présentée ci-dessous :

$$\begin{aligned}
D + k &= \begin{pmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{pmatrix} + \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix} \\
&= \begin{pmatrix} d_{0,0} + k_{0,0} & d_{0,1} + k_{0,1} & d_{0,2} + k_{0,2} & d_{0,3} + k_{0,3} \\ d_{1,0} + k_{1,0} & d_{1,1} + k_{1,1} & d_{1,2} + k_{1,2} & d_{1,3} + k_{1,3} \\ d_{2,0} + k_{2,0} & d_{2,1} + k_{2,1} & d_{2,2} + k_{2,2} & d_{2,3} + k_{2,3} \\ d_{3,0} + k_{3,0} & d_{3,1} + k_{3,1} & d_{3,2} + k_{3,2} & d_{3,3} + k_{3,3} \end{pmatrix}.
\end{aligned}$$

De manière plus condensée, le résultat de $D + k$ est une matrice E de 4 lignes et 4 colonnes dont les éléments sont les octets $e_{i,j}$ donnés par

$$e_{i,j} = d_{i,j} + k_{i,j}.$$

Cette opération d'addition de sous-clef est aussi inversible car on a

$$E = D + k \;\Leftrightarrow\; D = E + k$$

puisque l'addition dans $\mathsf{GF}(2^8)$ représente en réalité un simple ou-exclusif.

Ainsi $(D + k) + k = D$.

7.3. Nombres premiers et cryptographie à clef publique

7.3.1. *Introduction*

Les cryptosystèmes qui utilisent des nombres premiers sont nombreux et très utilisés. Ils appartiennent bien souvent à la classe des procédés de chiffrement à clef

publique. Dans cette section nous présentons très brièvement le cryptosystème RSA, représentant le plus largement employé de cette classe, ainsi que la problématique liée à l'obtention et au choix des nombres premiers « cryptographiques ».

Le principe même du chiffrement à clef publique fut introduit par Diffie et Hellman en 1976 [DIF 76]. Bien que ces derniers furent incapables à l'époque de fournir un exemple concret de tels cryptosystèmes, cet article fonda une nouvelle branche de la cryptographie.

Les procédés de chiffrement à clef publique emploient deux genres distincts de clefs : une clef **publique** et une clef **privée**. La première est utilisée en chiffrement alors que la seconde intervient dans le processus de déchiffrement. Examinons les principales étapes d'une communication chiffrée entre Alice et Bob. Supposons ici que Alice souhaite transmettre un message confidentiel à Bob. Pour ce faire, elle récupère la clef publique de Bob qui, comme son nom l'indique, est disponible dans un répertoire public (par exemple un site internet). Alice chiffre son message avec cette clef puis transmet le chiffré à Bob. Ce dernier utilise sa propre clef secrète, évidemment connue de lui seul, afin de déchiffrer le message reçu et retrouver ainsi le message clair originel. Remarquons que pour que Bob puisse envoyer un message chiffré à Alice (par exemple un acquittement) il doit à son tour employer la clef publique d'Alice. Celle-ci utilise sa propre clef secrète pour déchiffrer comme l'avait fait Bob. Dans ce processus de communication, tout intervenant a accès aux clefs publiques alors que les clefs secrètes sont conservées par chaque destinataire de messages.

La sécurité des cryptosystèmes à clef publique est basée sur la difficulté voire l'infaisabilité pratique de résoudre certains problèmes mathématiques. En particulier le problème sur lequel est fondée la sûreté de RSA est celui de la factorisation des grands entiers.

7.3.2. *Cryptosystème RSA*

L'un des premiers cryptosystèmes à clef publique, RSA, fut développé par Rivest, Shamir et Adleman [RIV 78].

Étant donnés deux entiers premiers p et q il est aisé de calculer leur produit $n = pq$. Il est en revanche très difficile de **factoriser** cet entier n, c'est-à-dire de retrouver ses facteurs premiers p et q lorsque les entiers p et q sont suffisamment grands (de l'ordre du millier de chiffres décimaux). La sécurité RSA est basée sur cette constation.

7.3.2.1. *Création des clefs*

Soient p et q deux entiers premiers distincts. Soit $n = pq$ le *module RSA*. Un entier e premier avec $(p-1)(q-1)$, et strictement inférieur à $(p-1)(q-1)$, est choisi. Il s'agit

de l'*exposant de chiffrement*. On calcule ensuite l'inverse d de e modulo $(p-1)(q-1)$, c'est-à-dire que le produit ed satisfait l'équation suivante :

$$ed = a(p-1)(q-1) + 1 \tag{7.5}$$

où a est un certain entier. Le nombre d est alors l'*exposant de déchiffrement*. Le couple (n, e) est la clef publique et le couple (p, q, d) est la clef secrète.

7.3.2.2. *Chiffrement d'un message*

Un message clair m est alors un entier positif et strictement inférieur au module n. Le message chiffré correspondant est l'entier c, $0 \leq c < n$, donné par

$$c = m^e \ mod \ n \ .$$

7.3.2.3. *Déchiffrement d'un message*

Pour retrouver le message clair à partir du chiffré c il suffi d'élever ce dernier à la puissance d (modulo n) :

$$m = c^d \ mod \ n \ .$$

On peut en effet démontrer la suite d'égalités suivante :

$$\begin{aligned} c^d \ mod \ n &= (m^d)^e \ mod \ n \\ &= m^{ed} \ mod \ n \\ &= mm^{a(p-1)(q-1)} \ mod \ n \text{ (d'après (7.5))} \\ &= m(m^{(p-1)(q-1)})^a \ mod \ n \\ &= m \ mod \ n \\ &= m \end{aligned}$$

car on peut vérifier que l'on a $m^{(p-1)(q-1)} = m \ mod \ n = m$.

7.3.3. *Primalité et pseudo primalité*

L'utilisation de nombres premiers dans les algorithmes de chiffrement à clef publique — comme dans RSA — est très classique. Il s'ensuit que la notion de primalité est essentielle au développement de tels procédés. Nous terminons ainsi ce chapitre par une présentation de méthodes et de techniques utilisées pour déterminer si un nombre est premier ou non, ou encore pour produire des nombres premiers.

La mise en oeuvre de cryptosystèmes à clef publique mettant en jeu des nombres premiers suppose résolus deux problèmes distincts. Dans le cas des cryptosystèmes utilisant un unique nombre premier p il faut être sûr que l'entier est bien premier. Cette assurance est fournie par les tests de primalité et une stratégie associée qui seront passés en revue.

Dans le cas où la solidité de l'algorithme repose sur la difficulté de factoriser un entier RSA $n = pq$ des contraintes sont requises sur les entiers premiers p et q pour que la complexité de la factorisation de n soit aussi élevée que prévue. L'obtention de clefs RSA sera aussi examinée.

Pour savoir si un nombre est premier ou pas, la méthode qui consiste à le factoriser fournit cette information. Le crible d'Eratosthène est une des plus anciennes méthodes pour y parvenir. Toutefois les temps de calculs pour arriver à trouver des premiers deviennent très vite importants et les besoins mémoire également, ce qui les rend inutilisables pour les entiers décimaux de 1000 à 2000 chiffres que l'on souhaite obtenir.

Les algorithmes de factorisation les plus performants parviennent à factoriser des entiers de l'ordre de 200 chiffres décimaux. Ces nombres sont bien inférieurs à ceux dont on a besoin et il faut donc rechercher des premiers par d'autres critères que la factorisation.

Le théorème de Fermat affirme que si p est un entier premier, pour tout entier a avec $1 < a < p - 1$ on a

$$a^{p-1} = 1 \; mod \; p$$

Sa réciproque est fausse mais permet de mettre au point des tests de pseudo primalité faible et forte qui sont utiles [MEN 97]. Des tests de primalité sûrs ont été mis au point à partir d'une réciproque due à Lehmer et rendue utilisable par Pocklington, qui sera examinée. Toutefois la mise en œuvre est assez ardue et ne sera utilisée que pour des entiers qui ont déjà passé avec succès des tests de pseudo primalité.

Les résultats obtenus sont entachés d'une probabilité d'erreur, très faible mais non nulle, due aux imperfections informatiques sur lesquels ces algorithmes sont exécutés, imperfections logicielles ou matérielles.

7.3.4. *Test de pseudo primalité*

Le réciproque du théorème de Fermat est fausse : pour un entier n non premier il existe au moins un entier a tel que

$$a^{n-1} = 1 \; mod \; n$$

Ceci permet de définir un test de pseudo primalité faible de base a pour un entier n.

1) Calculer $a^{n-1} mod \; n$;

2) Si le résultat ne vaut pas 1 l'entier n est composite (c'est-à-dire non premier) ;

3) S'il est égal à 1 il est dit pseudo premier faible de base a.

Il faut remarquer qu'un entier premier est un pseudo premier de toute base.

Il existe un algorithme de calcul de cette exponentielle de complexité $O(log_2\ n)$. Il s'agit de l'algorithme de Strassen [SCH 71], version multiplicative de l'algorithme dit « des paysans russes ». Il s'agit de la forme récursive du calcul de l'exponentielle, qui relie le calcul de a^n à celui de $a^{n/2}$.

Pour les longueurs de n envisagées en cryptographie ce calcul s'effectue en quelques millisecondes sur les machines de bureau actuelles. C'est cet algorithme qui est utilisé dans les implémentations de RSA (et d'autres cryptosystèmes).

7.3.4.1. *Probabilité d'erreur et primalité*

Si l'on se donne une base a et un entier n avec $a < n$ et que n est pseudo premier de base a, l'assertion n est premier est une assertion qui sera fausse dans le cas d'un entier n non premier.

Il y a une certaine probabilité d'erreur dûe à l'algorithme employé et non pas à une quelconque erreur de programmation.

Pour mesurer cette probabilité d'erreur, il faut arriver à évaluer le nombre d'entiers pseudo premiers mais non premiers.

Il faut remarquer qu'un entier pseudo premier de base a n'est en général pas pseudo premier de base b. Ainsi pour être sûr de réduire la probabilité d'erreur sur la primalité d'un entier n il faut essayer de tester la primalité sur plusieurs bases. Dès qu'un entier pseudo premier sur une base n'est pas pseudo premier sur une autre base il faut le rejeter.

Bien que les entiers premiers soient des pseudo premiers faibles de toute base, ils ne sont toutefois pas les seuls. Il existe de pseudo premiers faibles de toute base non premiers, il s'agit, notamment, des entiers de Carmichael dont le plus petit est 561. Rapellons qu'un entier de Carmichael [CAR 12] est un entier n composite qui vérifie la propriété suivante : pour tout entier a, a^n est égal à a modulo n.

Les calculs de densité de pseudo premiers faibles non premiers parmi les pseudo premiers ont été faits aussi bien sur le plan théorique que pratique. Cette densité diminue avec la taille de n et on dispose de résultats assez précis.

Soit $P_2(x)$ l'ensemble des pseudo premiers de base 2 non premiers inférieurs à x. Soit $\pi(x)$ l'ensemble des entiers premiers inférieurs à x. Soit enfin $C(x)$ l'ensemble des entiers de Carmichael inférieurs à x.

Pour $x \approx 10^{20}$ le quotient $P_2(x)/\pi(x) = 1 \cdot 43.10^{-2}$ et $C(x)/\pi(x) = 7 \cdot 2.10^{-5}$. Ces probabilités diminuent avec le nombre de chiffres de n et passent à :

Pour $x \approx 10^{50}$ le quotient $P_2(x)/\pi(x) = 7 \cdot 4.10^{-7}$ et $C(x)/\pi(x) = 5 \cdot 16.10^{-16}$

7.3.4.2. *Nombre de bases*

La question pratique qui se pose lors de la mise en œuvre d'un tel protocole est le choix des « bases » et de leur nombre pour tester la pseudo primalité.

Si l'on n'effectue qu'un seul test la qualité sera de l'ordre de 10^{-7} pour des entiers à 50 chiffres. Si l'on augmente le nombre de tests on abaissera cette probabilité sans tomber au dessous de 10^{-16}.

Pour les valeurs qui nous concernent, il suffit en général d'une seule base, surtout si l'on doit confirmer la primalité par d'autres méthodes.

Rappelons que le taux d'erreur dû aux défaillances du matériel est de l'ordre de 10^{-12}.

7.3.4.3. *Pseudo primalité forte*

Le concept de pseudo primalité forte est apparu à la suite de la remarque suivante dûe à Euler et renforcée depuis.

Soit n un entier (pseudo premier de base a). Soit $n - 1 = d.2^s$ avec d impair.

Si $a^d = 1 \ mod \ n$ ou bien $a^{d.2^r} = 1 \ mod \ n$ avec $r < s$ alors n est dit pseudo premier fort de base a.

On a constaté, de manière expérimentale, que les entiers pseudo premiers forts non premiers étaient moins nombreux que les pseudo premiers faibles non premiers.

Par exemple pour les entiers inférieurs à 25.10^9 il n'y a que 13 pseudo premiers forts de base 2, 3, 5 non premiers. De ce fait la probabilité d'erreur intrinsèque du test de pseudo primalité forte utilisé en tant que test de primalité est plus faible que pour les pseudo premiers faibles dans cet intervalle, pour ces mêmes bases, qui eux sont de l'ordre de 2600.

De plus il y a très peu de pseudo premiers forts de plusieurs bases. Les nombres de Carmichael ne sont pas pseudo premiers forts ; on ne sait pas s'il existe une classe infinie de pseudo premiers forts de toute base.

7.3.5. *Les tests de primalité sûrs*

Il existe un test de primalité sûr sur le plan théorique qui est utilisable. Il s'agit d'une réciproque au théorème de Fermat conjecturée et démontrée par Lehmer [LEH 35]. Une version exploitable informatiquement à été mise au point par Pocklington [POC 14].

Soit N l'entier dont il faut prouver la primalité.

Soit $N-1 = R.F$ une factorisation partielle de $N-1$, F étant une partie complètement factorisée de $N-1$, et R une partie non factorisée.

Supposons que R et F soient premiers entre eux, et $R < F$.

Soit donc $F = \prod_{j=1}^{n} q_j^{\beta_j}$ avec les q_j premiers.

S'il existe un entier a tel que $a^{(N-1)/q_j} - 1$ et N sont premiers entre eux, c'est-à-dire n'ayant aucun facteur premier en commun, pour tout $j = 1, \ldots, n$ et tel que

$$a^{N-1} = 1 \; mod \; N$$

alors l'entier N est premier.

La mise en œuvre de l'algorithme associé est rendue difficile par le choix de l'entier a qui n'est pas obtenu par un algorithme déterministe.

Le point fort de cette méthode est qu'il ne faut connaître qu'un nombre partiel de facteurs pour parvenir au résultat.

L'entier premier sera obtenu par un algorithme ne comportant pas d'erreur.

7.3.5.1. *L'algorithme de construction associé*

Il est possible d'associer à ces considérations un algorithme de construction d'entiers premiers utilisables en cryptographie.

On part d'un entier R qui est premier de la plus grande taille possible, fourni par un algorithme de type crible. On choisit des petits premiers pour constituer la partie F qui comporte l'entier 2. En utilisant le théorème et en modifiant la partie F si nécessaire, on obtient un entier premier de longueur double.

En itérant ce processus il est possible d'obtenir des entiers premiers p de longueurs arbitrairement grandes avec la propriété que $p-1$ comporte toujours un grand facteur. Cette dernière propriété doit absolument être vérifiée pour que les algorithmes de factorisation les plus performants aient une probabilité de succès qui soit conforme aux prévisions.

Au vu de ce qui a été dit la méthode d'obtention des entiers premiers de qualité cryptographique est la suivante.

1) Poser $j = 1$;

2) Partir d'un entier premier p_j de longueur de 10 chiffres environ obtenu par un crible ;

3) Construire un ensemble de premiers de petite tailles et de certaines de leur puissance entière, appelé base de premiers ;

4) Tirer au hasard dans la base de premiers des entiers dont le produit F_j sera un entier pair tel que $F_j > p_j$;

5) Tester la pseudo primalité faible de $p_{j+1} = F_j.p_j + 1$;

6) Si la pseudo primalité faible n'est pas vérifiée changer F_j et recommencer ;

7) Changer de base de pseudo primalité et vérifier la pseudo primalité faible pour cette nouvelle base ;

8) Itérer la ligne 7 dix fois ;

9) Incrémenter j. Itérer les lignes 2 à 8 jusqu'à obtenir un entier p_j de la taille souhaitée ;

10) Tester la pseudo primalité forte de p_j. Si elle n'est pas vérifiée aller en 2) ;

11) Appliquer l'algorithme de Pocklington à p_j.

L'entier p_j est un premier du nombre de chiffres désiré.

7.3.6. *La factorisation des entiers*

La factorisation des entiers est un problème qui a vu ses méthodes de résolution évoluer.

Les plus performantes sont le crible quadratique et ses variantes [POM 85, CRA 05] ainsi que les méthodes de Pollard (plus anciennes et un peu moins puissantes) : la méthode ρ [POL 75] et la méthode $p - 1$ [POL 74].

Ces méthodes ont un caractère probabiliste : si un facteur d'un entier n est trouvé, c'est un vrai facteur. Toutefois elles ne parviennent pas à factoriser tous les entiers de même taille, contrairement aux méthodes de crible. C'est dans cette incertitude que réside le caractère probabiliste et non pas dans la qualité du résulat, qui elle est vérifiable.

Soit $n = p \cdot q$ un entier produit de deux premiers de même taille. Les algorithmes de crible factorisent n de taille pouvant atteindre 300 chiffres. Les algorithmes de Pollard cessent de fournir des facteurs dans un temps raisonnable pour des entiers de 200 chiffres.

Toutefois si pour l'un des facteurs p de n les entiers $p-1$ ou $p+1$ ne sont constitués que de petits facteurs, alors ces derniers algorithmes permettent de factoriser jusqu'à des longueurs de l'ordre de 600 et même au delà. Ceci explique les contraintes sur la méthode utilisée pour l'obtention des premiers utilisés dans RSA.

7.3.7. *L'obtention des couples de clefs RSA*

Un autre problème à résoudre pour une mise en œuvre du cryptosystème RSA est celui de l'obtention des clefs. Connaissant n et sa factorisation, donc le nombre d'entiers inférieurs à n et premiers avec n que l'on note $\phi(n)$ il s'agit de touver un (ou de préférence des) couple(s) d'entiers k et k' satisfaisant

$$k \cdot k' = 1 \; mod \; \phi(n)$$

La méthode comporte deux étapes. La première consiste à choisir k premier de petite taille (3 ou 4 chiffres). L'équation à résoudre devient alors

$$1 + \lambda\phi(n) = 0 \; mod \; k$$

Par recherche systématique sur les k valeurs possibles de λ, on trouve la bonne valeur λ_0. L'entier k' est alors obtenu par

$$k' = (1 + \lambda_0\phi(n))/k$$

Le couple (k, k') est un couple de clefs RSA.

7.3.7.1. *Généralisation*

La deuxième étape consiste à en obtenir d'autres. En effet les couples de clefs (k, k') ainsi obtenus ne sont pas assez nombreux et sont très typés : une des clefs est très courte.

Il est possible d'en obtenir un très grand nombre en remarquant que pour entier positif m les entiers $k^m \; mod \; \phi(n)$ et $k'^m \; mod \; \phi(n)$ sont aussi des couples de clefs RSA.

De même si l'on résoud l'équation avec un autre entier premier k_1 et que l'on obtient un autre couple (k_1, k'_1) alors le produit $(k \cdot k_1, k' \cdot k'_1)$ est aussi un couple de clefs RSA.

7.4. Conclusion

L'idée directrice de ce chapitre consistait à présenter d'une part les principes généraux de la cryptographie et d'autre part divers algorithmes de chiffrement actuels ou passés. Le second point souligne notamment le fait que la cryptographie et sa face cachée la cryptanalyse évoluent de concert à l'instar de proies et de leurs prédateurs soumis aux règles de la sélection naturelle. C'est ainsi qu'un cryptosystème ne peut être considéré comme sûr qu'à un instant donné et jamais dans l'absolu. Par exemple, l'emploi de l'AES plutôt que du DES est actuellement (c'est-à-dire *a priori* le premier quart du vingt-et-unième siècle) recommandé.

Ce processus évolutif fut particulièrement mis en valeur dans le cadre de la description des cryptosystèmes à clef secrète, à savoir dans la deuxième partie de ce chapitre, où furent exposés, dans l'ordre chronologique de leur apparition, trois algorithmes historiques parmi les plus célèbres : DES, IDEA et l'AES. Cette dynamique darwiniste se calque sur les avancées mathématiques (la découverte de nouvelles structures algébriques et de nouveaux algorithmes de cryptanalyse) aussi bien que sur la progression des performances des moyens informatiques (logiciels et matériels). Il est donc fort probable, si ce n'est certain, que de nouveaux procédés de chiffrement apparaîtront pendant que d'autres soumis à la pression sélective de formes novatrices de cryptanalyses disparaîtront en ne laissant comme trace de leur existence que des descriptions fossilisées dans des revues et livres scientifiques.

Assez récemment est apparue la notion de cryptographie quantique [BEN 84]. Elle diffère de la cryptographie présentée dans ce chapitre, laquelle pourrait être qualifiée de « classique », en ce sens qu'elle repose sur les principes physiques tirés de la mécanique quantique plutôt que sur des théorèmes mathématiques. La sécurité de tels systèmes est garantie par l'impossibilité quantique de dupliquer une fonction d'onde inconnue (le principe d'incertitude de Heisenberg), soit, en termes moins pédants, l'impossibilité d'effectuer des mesures sur un système quantique sans le pertuber. Ainsi lorsqu'un adversaire tente d'écouter une communication quantiquement chiffrée, il lui est nécessaire de mesurer des grandeurs (comme par exemple le spin des photons transmis) ce qui pertube le système quantique et permet aux interlocuteurs légitimes de savoir qu'ils sont espionnés.

La cryptographie quantique et plus généralement l'informatique quantique semblent suffisamment prometteuses pour espérer supplanter dans un avenir plus ou moins proche les cryptosystèmes de type classique tels *Sapiens* remplaçant *Erectus*.

7.5. Bibliographie

[BEN 84] BENNETT C. H., BRASSARD G., « Quantum cryptography : Public key distribution and coin tossing », *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*, p. 175–179, 1984.

[CAR 12] CARMICHAEL R. D., « On composite numbers P which satisfy the Fermat congruence $a^{P-1} \equiv 1 \ mod \ P$ », *The American Mathematical Monthly*, vol. 19, n°2, p. 22–27, 1912.

[CRA 05] CRANDALL R. E., POMERANCE C., *Prime Numbers a Computational Perspective*, Springer, 2005.

[DAE 99] DAEMEN J., RIJMEN V., « AES proposal : Rijndael », 1999.

[DAE 02] DAEMEN J., RIJMEN V., *The design of Rijndael : AES - the Advanced Encryption Standard*, Springer, 2002.

[DIF 76] DIFFIE W., HELLMAN M., « New directions in cryptography », *Information Theory, IEEE Transactions on*, vol. 22, n°6, p. 644–654, 1976.

[FEI 73] FEISTEL H., *Cryptography and computer privacy*, Scientific American, 1973.

[FIP 87] FIPS P., « 81 : DES Modes of Operation », *National Bureau of Standards*, vol. 1, 1987.

[FIP 99] FIPS P., « 46-3 : Data Encryption Standard », *National Institute for Standards and Technology*, vol. 25, 1999.

[FIP 01] FIPS P., « 197 : Advanced Encryption Standard », *National Institute of Standards and Technology*, vol. 26, 2001.

[KER 83a] KERCKHOFFS A., « La cryptographie militaire (première partie) », *Journal des Sciences Militaires*, vol. IX, p. 5–38, 1883.

[KER 83b] KERCKHOFFS A., « La cryptographie militaire (seconde partie) », *Journal des Sciences Militaires*, vol. IX, p. 161–191, 1883.

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

[LAI 92] LAI X., MASSEY J. L., MURPHY S., « Markov ciphers and differential cryptanalysis », *Advances in Cryptology - Eurocrypt*, vol. 91, p. 17–38, Springer, 1992.

[LEH 35] LEHMER D. H., « On Lucas's test for the primality of Mersenne's numbers », *J. London Math. Soc.*, vol. 10, p. 162–165, 1935.

[MEN 97] MENEZES A. J., OORSCHOT P. C. V., VANSTONE S. A., *Handbook of applied cryptography*, CRC Press, 1997.

[NEC 01] NECHVATAL J., BARKER E., BASSHAM L., BURR W., DWORKIN M., FOTI J., ROBACK E., « Report on the development of the Advanced Encryption Standard (AES) », *Journal of Research of the National Institute of Standards and Technology*, vol. 106, n°3, p. 511–576, 2001.

[NSA 03] NSA, « 15, Fact Sheet No. 1 National policy on the use of the Advanced Encryption Standard (AES) to protect national security systems and national security information. CNSS », 2003.

[POC 14] POCKLINGTON H. C., « The determination of the prime or composite nature of large numbers by Fermat's theorem », *Proc. Cambridge Phil. Soc.*, vol. 18, p. 29–30, 1914.

[POL 74] POLLARD J. M., « Theorems on factorization and primality testing », *Proc. Camb. Phil. Soc.*, vol. 76, n°2, p. 521–528, 1974.

[POL 75] POLLARD J. M., « A monte carlo method for factorization », *BIT Numerical Mathematics*, vol. 15, n°3, p. 331–334, Springer, 1975.

[POM 85] POMERANCE C., « The quadratic sieve factoring algorithm », *Proc. of the EUROCRYPT 84 workshop on Advances in cryptology : theory and application of cryptographic techniques table of contents*, p. 169–182, Springer, 1985.

[RIV 78] RIVEST R. L., SHAMIR A., ADLEMAN L., « A method for obtaining digital signatures », *Commun. ACM*, vol. 21, p. 120–126, 1978.

[SCH 71] SCHONHAGE A., STRASSEN V., « Schnelle Multiplikation grosser Zahlen », *Computing*, vol. 7, n°3-4, p. 281–292, 1971.

[SHA 49] SHANNON C. E., *Communication theory and secrecy Systems*, Bell Telephone Laboratories, 1949.

[STI 03] STINSON D. R., *Cryptographie : théorie et pratique*, Vuibert, 2003.

[VER 26] VERNAM G. S., « Cipher printing telegraph systems for secret wire and radio telegraphic communications », *Journal of the American Institute of Electrical Engineers*, vol. 45, p. 109–115, 1926.