

CHAPITRE 1

GENERALITE SUR LA CRYPTOGRAPHIE

.1 Principaux systèmes de chiffrement

.1.1 Historique

L'histoire de la cryptographie est déjà longue. On rapporte son utilisation en Egypte il y a 4000 ans. Toutefois, pendant des siècles, les méthodes utilisées étaient restées souvent très primitives. D'autre part, sa mise en œuvre était limitée aux besoins de l'armée et de la diplomatie.

Les méthode de chiffrement et de cryptanalyse ont connu un développement très important au cours de la seconde guerre mondiale et ont eu une profonde influence sur le cours de celle-ci.

Mais la prolifération actuelle des systèmes de communication a fait sortir la cryptographie du domaine militaire. De plus, elle a diversifié la demande et provoqué le développement de nouvelles techniques cryptographiques. Elle est à l'origine d'un développement rapide depuis les dernières décennies, qui ne semble pas s'essouffler aujourd'hui, bien au contraire.

Nombreux systèmes de chiffrement différents ont été imaginés pour se protéger contre la curiosité et la malveillance des ennemis depuis des siècles. On peut classer ces systèmes en trois grandes classes que nous allons représenter sur la *Figure 1.01*.

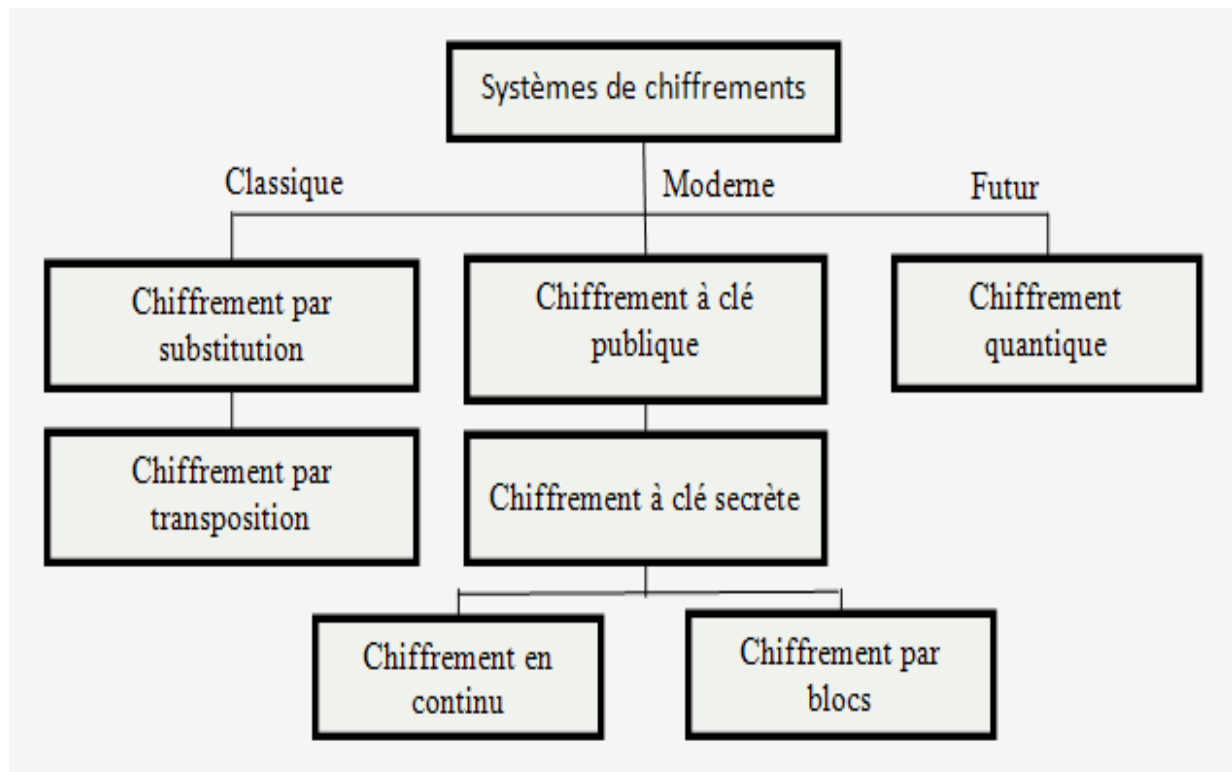


Figure 1.01 : *Les principales techniques de chiffrement*

.1.2 Systèmes classiques

Avant l'avènement des ordinateurs, l'opération de chiffrement était basée sur des caractères. L'idée était de transposer ou de remplacer les caractères d'un texte par d'autres. Les meilleurs systèmes répètent ces deux opérations de base plusieurs fois.

.1.2.1**.1.2.2 Substitution**

Historiquement, c'est le premier type de chiffrement utilisé. C'est un chiffrement dans lequel chaque caractère du texte en clair est remplacé par un autre caractère dans le texte chiffré.

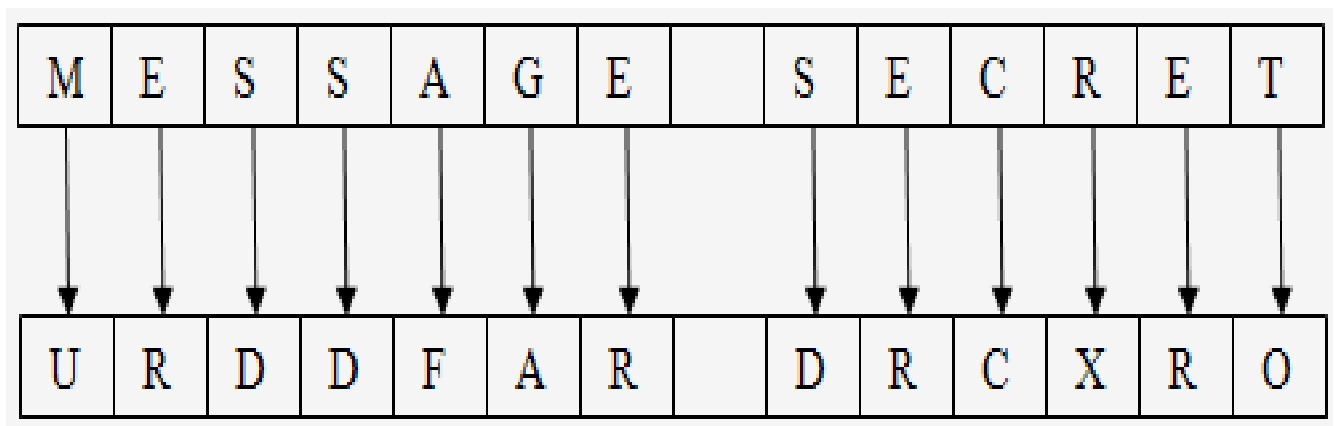


Figure 1.02 : Principe de la substitution

.1.2.3 Transposition

Un chiffrement par transposition est un chiffrement dans lequel les caractères du texte en clair demeurent inchangés mais dont les positions respectives sont modifiées.

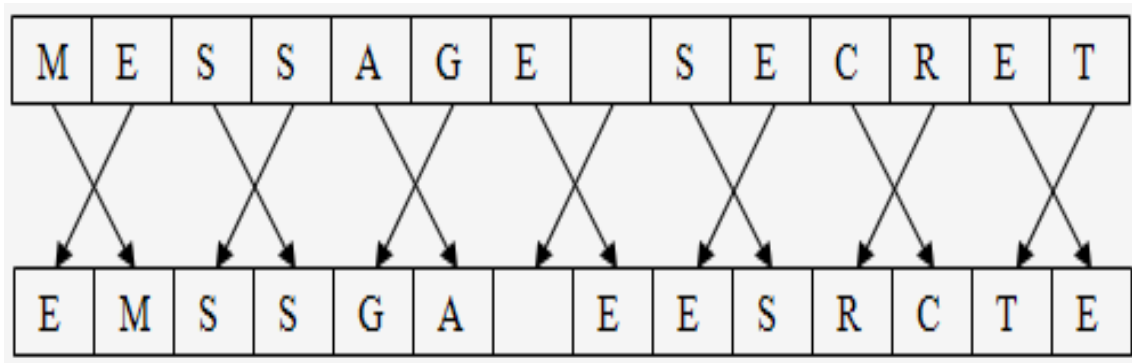


Figure 1.03 : Principe de la transposition

La substitution et la transposition peuvent être facilement cassées car elles ne cachent pas les fréquences des différents caractères du texte en clair. D'ailleurs, les procédures de chiffrement et de déchiffrement doivent être gardées secrètes.

.1.3 Systèmes modernes

Les systèmes modernes sont plus complexes, cependant la philosophie reste la même. La différence fondamentale est qu'ils exploitent la puissance des ordinateurs modernes en manipulant directement des bits, par opposition aux anciennes méthodes qui s'opèrent sur des caractères alphabétiques. Ce n'est donc qu'un changement de taille ou de représentation.

On distingue deux classes de chiffrement à base de clés.

.1.3.1 Chiffrement à clé privé

Le chiffrement à clé privé consiste à utiliser la même clé pour le chiffrement et le déchiffrement.

Par analogie c'est le principe d'une serrure d'une porte : tous les utilisateurs autorisés ont une clé identique. On distingue le système de chiffrement en continu et chiffrement par bloc.

.1.3.2 Chiffrement à clé publique

Les problèmes de distribution des clés sont résolus par la cryptographie à clé publique ou cryptographie asymétrique. Ce concept a été introduit par WhitfieldDiffie et Martin Hellman en 1975.

La cryptographie à clé publique est un procédé asymétrique utilisant une paire de clés asymétrique associé : une clé publique qui crypte des données et une clé privée ou secrète correspondante pour le décryptage. Vous pouvez ainsi publier votre clé publique tout en conservant votre clé privée secrète. Il est basé sur une méthode mathématique garantissant un encryptage facile et rapide, et un décryptage difficile. S'il fallait aussi une analogie, considérons que l'on crypte le message avec un cadenas (clé publique) que le détenteur de la clé privée peut ouvrir pour lire le cryptogramme. Il est impossible de retrouver la clé privée à partir de la clé publique.

Les principaux services offerts par la cryptographie moderne sont les suivantes :

- Confidentialité : assure que les données concernées ne pourront être dévoilées qu'aux personnes autorisées.

- Intégrité : assure que les données ne seront pas altérées (intentionnellement ou non) pendant leur transmission ou leur stockage.
- Authentification/Identification : Prouver l'origine d'une donnée ou l'identité d'une personne
- Signature proprement dite (undeniability ou non répudiation) : permet à une personne de prendre part à un contrat avec impossibilité de renier ensuite ses engagements.

.1.4 Systèmes de chiffrement quantique

Les systèmes de chiffrement quantique sont des systèmes fondés sur la mécanique quantique et les propriétés particulières de la matière dans ce domaine. Ils reposent sur le principe d'incertitude d'Heisenberg [10], selon lequel la mesure d'un système quantique perturbe ce système. Une oreille indiscrete sur un canal de transmission quantique engendre des perturbations inévitables qui alertent les utilisateurs légitimes. Ce système résout ainsi les problèmes de distribution de clé.

.2 Principe général du chiffrement

Le texte en clair est noté M . Ce peut être une suite de bits, un fichier de texte, une voix numérisé, ou une image vidéo numérique, ...

Le texte en clair peut être transmis ou stocké. Le texte chiffré est noté C , qui a la même taille que M , parfois plus grand.

La fonction de chiffrement, notée E , transforme M en C .

$$E(M) = C \quad (1.26)$$

La fonction inverse, notée D , de déchiffrement transforme C en M :

$$D(C) = M$$

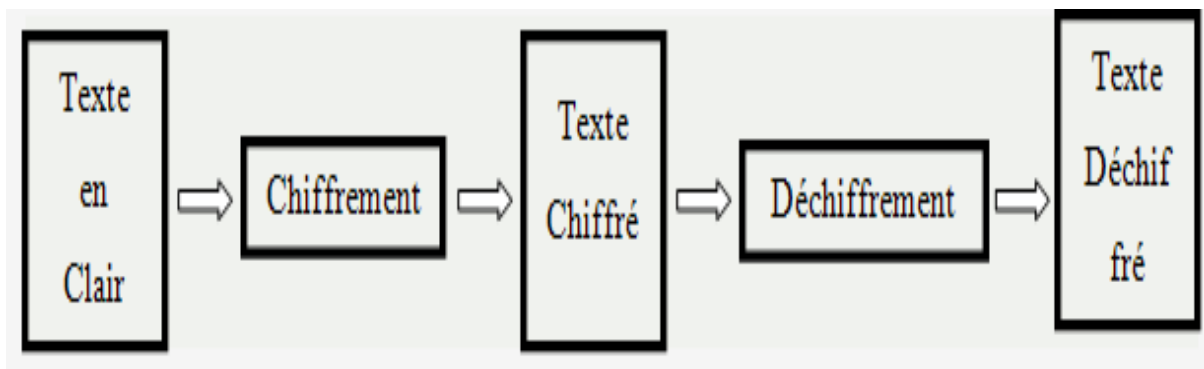


Figure 1.04 : *Chiffrement et déchiffrement*

.3 Clé de chiffrement

Une clé est une valeur qui est utilisée avec un algorithme cryptographique pour produire un texte chiffré spécifique. La cryptographie moderne utilise souvent une clé, notée K . Cette clé K peut prendre une valeur parmi un grand nombre de valeurs possibles. L'ensemble des valeurs possibles d'une clé est appelé espace des clés.

.3.1 Chiffrement avec une clé

Dans ce type de chiffrement, les opérations de chiffrement et de déchiffrement utilisent toutes les deux la clé K , aussi, les fonctions s'écrivent de la même manière suivante :

$$E_k(M) = \text{Cet}D_k(C) = M$$

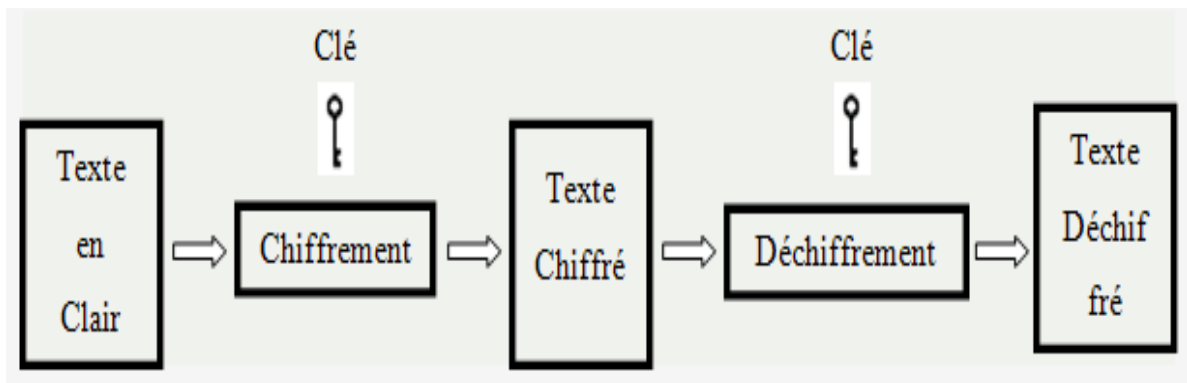


Figure 1.05 : Chiffrement avec une clé

.3.2 Chiffrement avec deux clés

Certains algorithmes utilisent des clés différentes pour le chiffrement et le déchiffrement. Dans ce cas, la clé de chiffrement, notée k_1 est différente de la clé de déchiffrement, notée k_2

$$E_{k_1}(M) = \text{Cet}D_{k_2}(C) = M$$

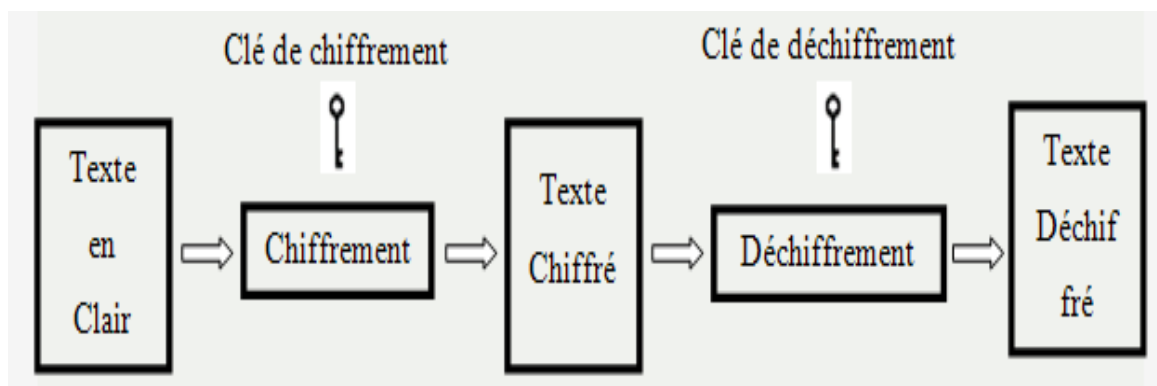


Figure 1.06 : *Chiffrement et déchiffrement avec deux clés*

.4 Algorithme cryptographique

Il y a deux types principaux d'algorithmes à base de clés : algorithme à clé secrète et algorithme à clé publique.

.4.1 Algorithme à clé secrète

Les algorithmes à clé secrète sont des algorithmes où la clé de chiffrement peut être calculée à partir de la clé de déchiffrement ou vice versa. Dans la plupart des cas, la clé de chiffrement et la clé de déchiffrement sont identiques. Pour de tels algorithmes, l'émetteur et le destinataire doivent se mettre d'accord sur une clé à utiliser avant d'échanger des messages. Cette clé doit être gardée secrète. La sécurité d'un algorithme à clé secrète repose ainsi sur la clé.

Les algorithmes à clé secrète peuvent être classés en deux catégories. Certains opèrent sur le message en clair un bit ou un octet à la fois. Ceux-ci sont appelés algorithmes de chiffrement en continu. D'autres opèrent sur le message en clair par groupes de bits de taille supérieure à un bit. Ces groupes de bits sont appelés blocs, et les algorithmes correspondants sont appelés algorithmes de chiffrement par blocs. La taille typique des blocs est de 64 bits.

.4.2 Algorithme à clé publique

Les algorithmes à clé publique sont conçus de telle manière que la clé de chiffrement soit différente de la clé de déchiffrement. De plus, la clé de déchiffrement ne peut pas être calculée à partir de la clé de chiffrement. De tels algorithmes sont appelés algorithmes « à clé publique » parce que la clé de chiffrement peut être rendue publique : n'importe qui peut l'utiliser pour chiffrer un message mais seul celui qui possède la clé de déchiffrement peut déchiffrer le message chiffré résultant. Dans de tels systèmes, la clé de chiffrement est appelée clé publique et la clé de déchiffrement est appelée clé privée. La clé privée est aussi appelée clé secrète.

Parfois, les messages seront chiffrés avec la clé privée et déchiffrés avec la clé publique ; une telle technique est utilisée pour les signatures numériques.

.4.3 *Choix d'algorithme*

La cryptographie à clé publique et à clé secrète sont deux choses différentes ; elles résolvent des problèmes de types différents. La cryptographie à clé secrète est meilleure pour chiffrer un message car elle est infiniment plus rapide. La cryptographie à clé publique peut faire des choses que la cryptographie à clé secrète ne permet pas ; elle est adoptée pour la gestion des clés. Il y a deux raisons à cela :

- Les algorithmes à clé publique sont lents. Les algorithmes à clé secrète sont généralement au moins 1000 fois plus rapide que les algorithmes à clé publique.
- Les cryptosystèmes à clé publique sont vulnérables aux attaques à texte en clair choisi. Si

- $C =$ où M est un texte en clair parmi n textes en clair possibles, alors, il suffit à un cryptanalyste de chiffrer les n messages et de comparer les textes chiffrés résultants avec C (la clé de chiffrement est publique). Il ne pourra pas trouver la clé de déchiffrement de cette manière, mais il pourra déterminer M .

.4.4 *Cryptosystèmes hybrides*

Dans la plupart des applications pratiques, la cryptographie à clé publique est utilisée pour protéger et distribuer les clés de session, et ces clés de session sont utilisées dans des algorithmes à clé secrète pour protéger les messages transmis. Cela est parfois appelé un cryptosystème hybride.

Voici le protocole correspondant :

- 1) Le destinataire B envoie sa clé publique à l'émetteur A ;
- 2) A engendre une clé de session aléatoire, k , la chiffre avec la clé publique de B et envoie le résultat à B :

$$E_B(k)$$

- 3) B utilise sa clé privée pour déchiffrer le message de A et ainsi retrouver la clé de session :

$$D_B(E_B(k)) = k$$

- 4) A et B utilisent alors la même clé de session pour chiffrer leur conversation.

Avec ce protocole, la clé de chiffrement est créée au moment de son utilisation pour chiffrer les communications et elle est détruite dès qu'on n'en a plus besoin. Cela réduit considérablement le risque de compromettre la clé de session.

.5 Générateurs aléatoires et pseudo-aléatoires

La cryptographie a souvent recours à des nombres aléatoires. Ainsi, lorsqu'une personne génère une clé secrète ou privée, elle doit faire intervenir le hasard de façon à empêcher un adversaire de deviner la clé. De même, certains protocoles cryptographiques nécessitent, pour éviter la rejouabilité par exemple, l'utilisation d'aléas imprévisibles par les opposants.

Malheureusement, il est impossible de produire des suites aléatoires à l'aide uniquement d'un ordinateur : le générateur sera toujours périodique, donc prévisible.

On a donc recours à des générateurs dits pseudo aléatoires et cryptographiquement sûrs. Un tel générateur doit présenter les caractéristiques suivantes :

- 1) La période de la suite doit être suffisamment grande pour que les sous suites finies utilisées avec l'algorithme ou le protocole cryptographique ne soient pas périodiques.
- 2) Ces sous suites doivent, sur le plan statistique, sembler aléatoires.
- 3) Le générateur doit être imprévisible, au sens où il doit être impossible de prédire le prochain aléa à partir des aléas précédents.

La plupart des générateurs pseudo-aléatoires sont construits en utilisant des registres à décalage (*shift registers* en anglais) et, en particulier, les registres à décalage à rétroaction linéaire (*Linear Feedback Shift Registers, LFSR*). Ces derniers présentent l'inconvénient de générer des suites linéaires, si bien que des grands nombres générés à partir de sous-suites sont fortement corrélés.

C'est pourquoi les générateurs pseudo-aléatoires sont généralement construits en combinant, à l'aide d'une fonction non linéaire, plusieurs registres à décalage de tailles différentes. Ce type de générateur est très utilisé par les algorithmes de chiffrement en continu.

Si l'on veut vraiment générer des suites aléatoires, au sens où ces suites sont de plus non reproductibles, on a généralement recours à des éléments extérieurs comme les déplacements de la souris, la vitesse de frappe, l'entrée d'un micro enregistrant le bruit atmosphérique,...

.6 Intégrité et authentification de l'origine des données

Parmi les problèmes auxquels s'attaque la cryptographie, on trouve l'authentification de l'origine des données et l'intégrité : lorsque l'on communique avec une autre personne au travers d'un canal peu sûr, on aimerait que le destinataire puisse s'assurer que le message émane bien de l'auteur auquel il est attribué et qu'il n'a pas été altéré pendant le transfert.

Les fonctions de hachage à sens unique interviennent dans la résolution de ces problèmes. Si l'on dispose d'un canal sûr (mais plus coûteux) en parallèle du canal de communication normal, on peut communiquer l'empreinte des messages par l'intermédiaire de ce canal. On assure ainsi l'intégrité des données transférées.

Sans canal sûr, le problème se complique : si l'on transfère l'empreinte sur un canal de communication non sûr, un intercepteur peut modifier les données puis recalculer l'empreinte. Il convient donc de trouver une méthode pour s'assurer que seul l'expéditeur est capable de calculer l'empreinte. Pour cela, on peut utiliser, par exemple, une fonction de hachage à sens unique qui fonctionne de plus avec une clé secrète ou privée. On remarquera qu'on fournit également l'authentification de l'origine des données.

Inversement, si on désire fournir l'authentification de l'origine des données et que l'on utilise pour cela un moyen qui ne garantit pas l'intégrité des données authentifiées, un intrus peut modifier le message et donc faire accepter comme authentifiées des données qu'il a choisies. C'est pourquoi intégrité et authentification de l'origine des données sont généralement fournies conjointement par un même mécanisme. On utilise parfois le terme *d'authenticité* pour désigner l'intégrité jointe à l'authentification des données.

.6.1 Fonctions de hachage à sens unique

Aussi appelée fonction de condensation, une fonction de hachage est une fonction qui convertit une chaîne de longueur quelconque en une chaîne de taille inférieure et généralement fixe. La chaîne résultante est appelée empreinte ou condensé de la chaîne initiale.

Une fonction à sens unique est une fonction facile à calculer mais difficile à inverser. La cryptographie à clé publique repose sur l'utilisation de fonctions à sens unique à brèche secrète : pour qui connaît le secret (i.e. la clé privée), la fonction devient facile à inverser.

Une fonction de hachage à sens unique est une fonction de hachage qui est en plus une fonction à sens unique : il est aisé de calculer l'empreinte d'une chaîne donnée, mais il est difficile d'engendrer des chaînes qui ont une empreinte donnée, et donc de déduire la chaîne initiale à partir de l'empreinte. On demande généralement en plus à une telle fonction d'être sans collision, c'est-à-dire qu'il soit impossible de trouver deux messages ayant la même empreinte. On utilise souvent le terme fonction de hachage pour désigner une fonction de hachage à sens unique sans collision.

La plupart des fonctions de hachage à sens unique sans collision sont construites par itération d'une fonction de compression : le message M est décomposé en n blocs m_1, \dots, m_n puis une fonction de compression est appliquée à chaque bloc et au résultat de la compression du bloc précédent ; l'empreinte $h(M)$ est le résultat de la dernière compression.

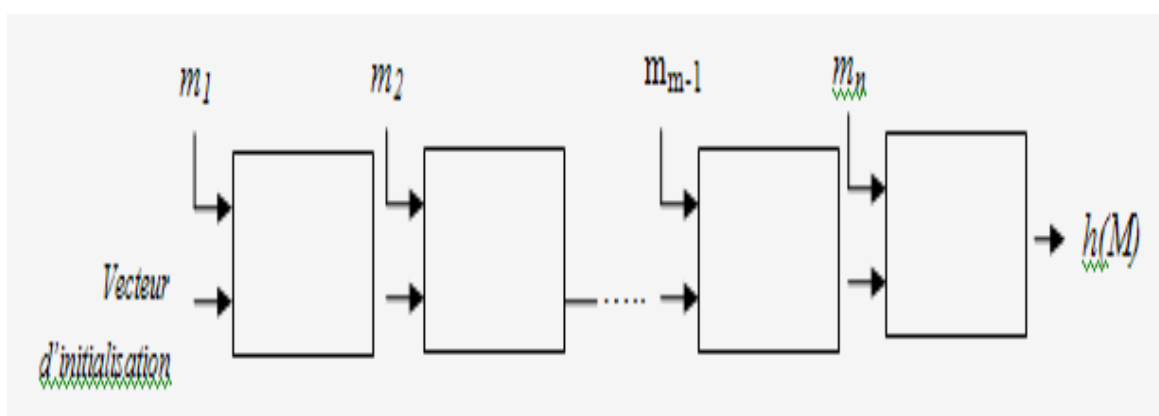


Figure 1.07 : Fonction de hachage itérative

.6.1.1 MD5 (Message Digest 5)

Développé par Rivest en 1991, MD5 produit une empreinte de 128 bits à partir d'un texte de taille arbitraire. MD5 manipule le texte d'entrée par blocs de 512 bits.

.6.1.2 SHA (Secure Hash Algorithm)

SHA est la fonction de hachage utilisée par SHS (*Secure Hash Standard*), la norme du gouvernement Américain pour le hachage. SHA-1 est une amélioration de SHA publiée en 1994. SHA-1 produit une empreinte de 160 bits à partir d'un message de longueur maximale 2^{64} bits. Tout comme MD5, SHA-1 travaille sur des blocs de 512 bits.

.6.1.3 RIPE-MD

Développée dans le cadre du projet **RIPE** (*RACE Integrity Primitives Evaluation*) de la communauté Européenne, RIPE-MD fournit une empreinte de 128 bits. RIPE-MD-160 est une version renforcée de RIPE-MD qui fournit une empreinte de 160 bits.

.6.2 Signature numérique

La norme ISO 7498-2 définit la signature numérique comme des "données ajoutées à une unité de données, ou transformation cryptographique d'une unité de données, permettant à un destinataire de prouver la source et l'intégrité de l'unité de données et protégeant contre la contrefaçon (par le destinataire, par exemple). La mention "protégeant contre la contrefaçon" implique que seul l'expéditeur doit être capable de générer la signature.

Une signature numérique fournit les services d'authentification de l'origine des données, d'intégrité des données et de non-répudiation. Ce dernier point la différencie des *codes d'authentification de message*, et a pour conséquence que la plupart des algorithmes de signature utilisent la cryptographie à clé publique.

Sur le plan conceptuel, la façon la plus simple de signer un message consiste à chiffrer celui-ci à l'aide d'une clé privée : seul le possesseur de cette clé est capable de générer la signature, mais toute personne ayant accès à la clé publique correspondante peut la vérifier. Dans la pratique, cette méthode est peu utilisée du fait de sa lenteur.

Une autre méthode utilisée pour signer consiste à calculer une empreinte du message à signer et à ne chiffrer que cette empreinte. Le calcul d'une empreinte par application d'une fonction de hachage étant rapide et la quantité de données à chiffrer étant fortement réduite, cette solution est bien plus rapide que la précédente.

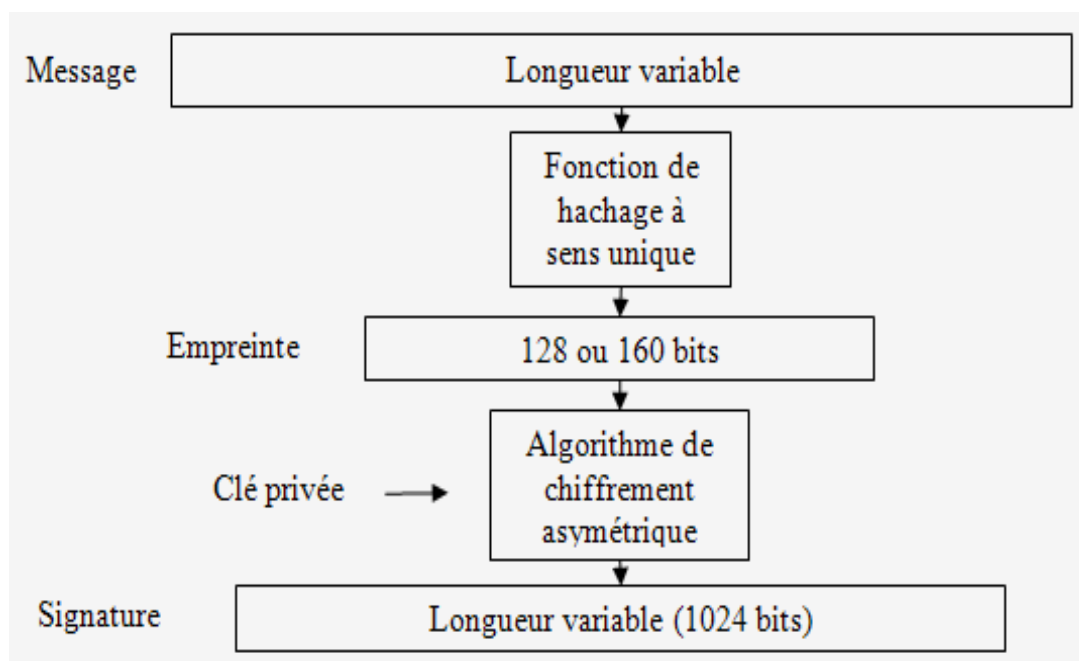


Figure 1.08 : Signature

.6.2.1 DSS

Proposé par le NIST en 1991, puis finalement adopté en 1994, DSS (*Digital Signature Standard*) est la norme de signature numérique du gouvernement Américain. Elle se base sur l'algorithme DSA (*Digital Signature Algorithm*), qui utilise SHA comme fonction de hachage à sens unique et El Gamal pour la génération et la vérification de la signature.

.6.2.2 RSA

Si DSS est la norme officielle aux U.S.A., RSA est en revanche une norme de fait, qui est en pratique beaucoup plus utilisé pour la signature que DSA.

.6.3 Scellement

Tout comme la signature numérique, le scellement fournit les services d'authentification de l'origine des données et d'intégrité des données, mais il ne fournit pas le non répudiation. Ceci permet l'utilisation de la cryptographie à clé secrète pour la génération du sceau ou *code d'authentification de message*.

Un code d'authentification de message (MAC : *Message Authentication Code*) est le résultat d'une fonction de hachage à sens unique dépendant d'une clé secrète : l'empreinte dépend à la fois de l'entrée et de la clé. On peut construire un MAC à partir d'une fonction de hachage ou d'un algorithme de chiffrement par blocs. Il existe aussi des fonctions spécialement conçues pour faire un MAC.

Une façon courante de générer un code d'authentification de message consiste à appliquer un algorithme de chiffrement symétrique en mode CBC au message ; le MAC est le dernier bloc du cryptogramme.

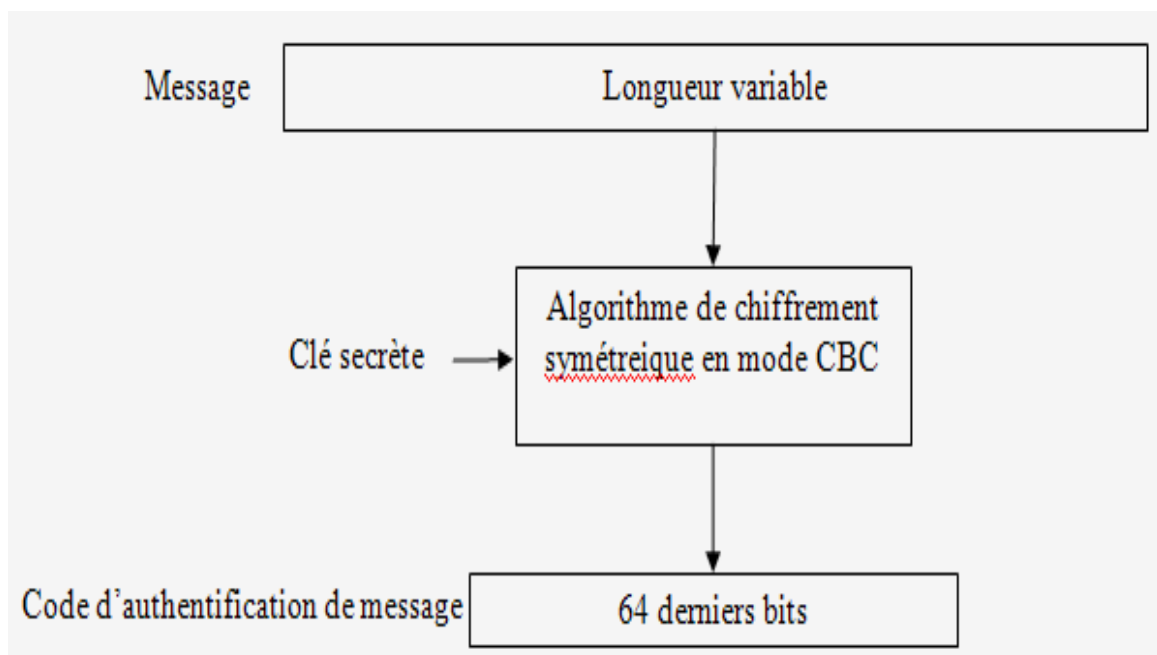


Figure 1.09 : *Scellement à l'aide d'un algorithme de chiffrement symétrique*

Un moyen simple de transformer une fonction de hachage à sens unique en un MAC consiste à chiffrer l'empreinte avec un algorithme à clé secrète. Une autre méthode consiste à appliquer la fonction de hachage non pas simplement aux données à protéger, mais à un ensemble dépendant à la fois des données et d'un secret.

.6.3.1 Keyed-Hash

Un exemple simple de cette façon de procéder est de prendre pour MAC des valeurs du type $H(\text{secret}, \text{message})$, $H(\text{message}, \text{secret})$ ou $H(\text{secret}, \text{message}, \text{secret})$. Ces méthodes, présentées en 1992 par Gène Tsudiks s'appellent respectivement méthode du préfixe secret, du suffixe secret et de l'enveloppe secrète. Elles ont une sécurité limitée.

.6.3.2 HMAC

Une méthode de calcul de MAC à base de fonction de hachage plus élaborée et plus sûre est HMAC. La méthode HMAC peut être utilisée avec n'importe quelle fonction de hachage itérative telle que MD5, SHA-1 ou encore RIPE-MD.

Soit H une telle fonction, K le secret et M le message à protéger. H travaille sur des blocs de longueur b octets (64 en général) et génère une empreinte de longueur l octets (16 pour MD5, 20 pour SHA et RIPE-MD-160). Il est conseillé d'utiliser un secret de taille au moins égale à l octets.

On définit deux chaînes, ipad (inner pudding data) et opad (outer pudding data), de la façon suivante : $\text{ipad} = l'octet\ 0 \times 36 \text{ répété } b \text{ fois}$, $\text{opad} = l'octet\ 0 \times 5 \text{ répété } b \text{ fois}$. Le MAC se

calcule alors suivant la formule suivante :
$$HMAC_K(M) = H(K \parallel \text{opad} \parallel H(K \parallel \text{ipad} \parallel M)).$$

Une pratique courante avec les fonctions de calcul de MAC est de tronquer la sortie pour ne garder comme MAC qu'un nombre réduit de bits. Avec HMAC on peut ainsi choisir de ne retenir que les t bits de gauche, où t doit être supérieur à $l/2$ et 80. On désigne alors sous la forme

HMAC-H- t l'utilisation de HMAC avec la fonction H , tronque à t bits (par exemple, HMAC-SHA1-96).

.7 Protocoles cryptographiques

Tout comme les protocoles de communication, les protocoles cryptographiques sont une série d'étapes prédéfinies, basées sur un langage commun, qui permettent à plusieurs participants (généralement deux) d'accomplir une tâche. Dans le cas des protocoles cryptographiques, les tâches en question sont bien sûr liées à la cryptographie : ce peut être une authentification, un échange de clé,....

Une particularité des protocoles cryptographiques est que les tiers en présence ne se font généralement pas confiance et que le protocole a donc pour but d'empêcher l'espionnage et la tricherie.

.7.1 Protocoles à apport nul de connaissance

Une situation fréquente est celle où un tiers doit prouver à un autre la connaissance d'un secret sans rien révéler sur ce secret. Les protocoles qui permettent de répondre à ce problème sont appelés protocoles à apport nul de connaissance ou preuves à divulgation nulle (*zero-knowledge*).

Les protocoles à apport nul de connaissance prennent la forme de protocoles interactifs au cours desquels Bob, le vérificateur, pose à Alice, une série de questions. Si Alice connaît le secret, elle saura répondre correctement à toutes les questions ; sinon elle aura 50% de chances de répondre correctement à chaque question. La probabilité qu'Alice connaisse effectivement le secret après n

réponses correctes est donc de $1 - \frac{1}{2^n}$. Après 10 questions, cette probabilité sera donc de 0,999.

Soit n un module aléatoire, produit de deux grands nombres premiers. La clé publique d'Alice est

v , un résidu quadratique modulo n (i.e. $x^2 = v \bmod n$ admet une solution et $v^{-1} \bmod n$ existe). La

clé privée correspondante est le plus petit s tel que $s \equiv \sqrt{v^{-1}} \bmod n$

Le déroulement d'une ronde du protocole est le suivant :

1) Alice choisit un nombre aléatoire $r \leq n$, puis calcule $x = r^2 \bmod n$ et envoie x à Bob.

2) Bob envoie un bit aléatoire $b = 0$ ou 1 à Alice.

3) Alice envoie $y = rs^b \bmod n$ à Bob

Si $b = 0$, $y = r$, donc Bob vérifie que $y^2 = x \bmod n$; cela prouve qu'Alice connaît

$$r = \sqrt{x}$$

Si $b = 1$, $y = rs$, donc Bob vérifie que $y^2 v \equiv x \bmod n$ cela prouve qu'Alice connaît s .

Si Alice ne connaît pas s , pour tromper Bob lorsque $b = 1$, elle doit lui envoyer $x = v^{-1}$; mais

cela implique que $r = s$, donc Alice ne saura pas répondre si $b = 0$.

Inversement, si Alice envoie effectivement une valeur x générée par $x = r^2 \bmod n$, elle saura répondre lorsque $b = 0$ ($y = r$), mais elle ne pourra pas répondre lorsque $b = 1$, car il lui faudrait connaître s . Dans chaque cas, Alice a donc une chance sur deux de savoir répondre à la question. Le principal problème des protocoles à apport nul de connaissance est le nombre important d'échanges nécessaires, qui les rend peu utilisés en pratique.

.7.2 La notion de tiers de confiance

Beaucoup de protocoles cryptographiques, notamment ceux visant à sécuriser des environnements distribués, ont recours à la notion de *tiers de confiance*.

Dans un tel cadre, pour établir une communication sécurisée, Alice et Bob se font aider de Norbert le notaire, qui est une personne en qui ils ont confiance. Norbert va jouer un rôle dans la sécurisation des échanges entre Alice et Bob en participant à la mise en œuvre de mécanismes de sécurité, notamment en intervenant dans les protocoles d'authentification et d'échange de clé.

.7.3 Echange de clé

Les deux méthodes les plus courantes d'établissement de clé sont le transport de clé et la génération de clé. Un exemple de transport de clé est l'utilisation d'un algorithme à clé publique pour chiffrer une clé de session générée aléatoirement. Un exemple de génération de clé est le protocole Diffie-Hellman [1] qui permet de générer un secret partagé à partir d'informations publiques

Le seul protocole décrit ici est le principe de génération de clé de Diffie-Hellman et les façons de l'étendre pour contrer les attaques de l'intercepteur. Diffie-Hellman est à la base de nombreux protocoles d'échange de clé.

.7.3.1 Diffie-Hellman

Inventé en 1976 par Diffie et Hellman [1], ce protocole permet à deux tiers de générer un secret partagé sans avoir aucune information préalable l'un sur l'autre. Il est basé sur la cryptologie à clé publique, car il fait intervenir des valeurs publiques et des valeurs privées. Sa sécurité dépend de la difficulté de calculer des logarithmes discrets sur un corps fini. Le secret généré à l'aide de cet algorithme peut ensuite être utilisé pour dériver une ou plusieurs clés (clé secrète,...).

Voici le déroulement de l'algorithme :

- 1) Alice et Bob se mettent d'accord sur un grand entier n tel que $(n-1)/2$ soit premier et sur un entier g primitif par rapport à n . Ces deux entiers sont publics.
- 2) Alice choisit de manière aléatoire un grand nombre entier a , qu'elle garde secret, et calcule sa valeur publique, $A = g^a \bmod n$. Bob fait de même et génère b et $B = g^b \bmod n$.
- 3) Alice envoie A à Bob ; Bob envoie B à Alice.
- 4) Alice calcule $K_{AB} = B^a \bmod n$; Bob calcule $K_{BA} = A^b \bmod n$.

$K_{AB} = K_{BA} = g^{ab} \bmod n$ est le secret partagé par Alice et Bob.

Une personne qui écoute la communication connaît g , n , A et B , ce qui ne lui permet pas de calculer $g^{ab} \bmod n$: il lui faudrait pour cela calculer l'algorithme de A ou B pour retrouver a ou b .

En revanche, Diffie-Hellman est vulnérable à l'attaque active dite attaque de l'intercepteur.

Une façon de contourner le problème de l'attaque de l'intercepteur sur Diffie-Hellman est

d'authentifier les valeurs publiques utilisées pour la génération du secret partagé. On parle alors de Diffie-Hellman authentifié.

L'authentification peut se faire à deux niveaux :

- En utilisant des valeurs publiques authentifiées, à l'aide de certificats par exemple. Cette méthode est notamment à la base du protocole SKIP.
- En authentifiant les valeurs publiques après les avoir échangées, en les signant par exemple.

L'inconvénient, dans les deux cas, est que l'on perd un gros avantage de Diffie-Hellman, qui est la possibilité de générer un secret partagé sans aucune information préalable sur l'interlocuteur.

.7.3.2 Echange de clé et authentification mutuelle

Pour établir une communication sécurisée, on procède généralement, en premier lieu, à une authentification à des fins de contrôle d'accès.

Puis, un échange de clé permet l'utilisation d'un mécanisme de sécurisation des échanges : l'authentification est ainsi étendue à la suite de la communication. L'exemple de Diffie-Hellman montre comme il est important que l'échange de clé soit authentifié. On parle alors de protocole d'authentification mutuelle avec échange de clé.

.7.3.3 Propriétés des protocoles d'échange de clé

Diffie, Van Oorschot et Wiener définissent, la notion de protocole d'authentification mutuelle avec échange de clé sûr.

Un protocole est dit sûr si les deux conditions suivantes sont valables dans chaque instance du protocole où l'un des deux tiers, par exemple Alice, exécute le protocole honnêtement et accepte l'identité de l'autre tiers :

- Au moment où Alice accepte l'identité de Bob, les enregistrements des messages échangés par les deux tiers se correspondent (i.e. les messages n'ont pas été altérés en route).
- Il est matériellement impossible pour toute personne autre que les tiers en présence de retrouver la clé échangée.

La propriété évoquée ci-dessus représente le minimum requis pour tout protocole. Cependant, d'autres propriétés des protocoles d'authentification avec échange de clé peuvent être souhaitables :

- La propriété dite de *PFS (PerfectForwardSecrecy)* est garantie si la découverte par un adversaire du ou des secrets à long terme ne compromet pas les clés de session générées précédemment : les clés de session passées ne pourront pas être retrouvées à partir des secrets à long terme.

On considère généralement que cette propriété assure également que la découverte d'une clé de session ne compromet ni les secrets à long terme ni les autres clés de session.

A ce sujet, on parle d'attaque de Denning-Sacco lorsqu'un attaquant récupère une clé de session et utilise celle-ci, soit pour se faire passer pour un utilisateur légitime, soit pour rechercher les secrets à long terme (par attaque exhaustive ou attaque par dictionnaire).

- La propriété dite de BTP (*Back Traffic Protection*) est fournie si la génération de chaque clé de session se fait de manière indépendante : les nouvelles clés ne dépendent pas des clés précédentes et la découverte d'une clé de session ne permet ni de retrouver les clés de session passées ni d'en déduire les clés à venir.
- On dit qu'il y a authentification directe (*Direct Authentication*) si, à la fin du protocole, les valeurs servant à générer le secret partagé sont authentifiées ou si chaque tiers a prouvé qu'il connaissait la clé de session.

Par opposition, l'authentification est dite indirecte (*Indirect authentication*) si elle n'est pas garantie à la fin du protocole, mais dépend de la capacité de chaque tiers à utiliser, dans la suite des échanges, la ou les clés mises en place précédemment.

- On parle de protection de l'identité (*Identity Protection*) lorsque le protocole garantit qu'un attaquant espionnant les échanges ne pourra pas connaître les identités des tiers communiquant.
- Enfin, l'utilisation du temps (*Timestamps*) afin d'éviter la rejouabilité est très controversée du fait, principalement, de sa trop grande dépendance d'horloges synchronisées.

.7.3.4 Hiérarchie des clés

On peut définir plusieurs types de clés suivant leurs rôles :

- Clé de chiffrement de clés : situées en haut de la hiérarchie, ces clés servent exclusivement à chiffrer d'autres clés et ont généralement une durée de vie longue. On notera que leur utilisation, restreinte au chiffrement de valeurs aléatoires qui sont des clés, rend les attaques par cryptanalyse plus difficiles à leur niveau. La cryptographie à clé publique est souvent utilisée pour le transport de clés en chiffrant la clé à transporter à l'aide d'une clé publique.

- Clés maîtresses : les clés maîtresses sont des clés qui ne servent pas à chiffrer mais uniquement à générer d'autres clés par dérivation. Une clé maîtresse peut ainsi être utilisée, par exemple, pour générer deux clés : une pour le chiffrement et une pour la signature.
- Clés de session ou de chiffrement de données. D'une durée de vie généralement faible (elle peut aller jusqu'à changer à chaque message), une telle clé, par opposition aux précédentes, sert à chiffrer des données et se situe donc au bas de la hiérarchie. Du fait de leur lenteur, les algorithmes asymétriques sont très peu utilisés en chiffrement de données, et les clés de session sont donc généralement des clés secrètes.

Il est à noter que des abus de langage ont souvent lieu avec ces termes. Ainsi, on parle parfois de clé de session pour désigner en fait une clé maîtresse de même durée de vie et servant à générer plusieurs clés : une pour l'authentification, une pour le chiffrement,...

CHAPITRE 1

PANORAMA SUR LA CRYPTOGRAPHIE MODERNE

Ce chapitre donne la synthèse des systèmes cryptographiques qui ont marqué l'histoire même du crypto système.

.8 Les systèmes de chiffrement classique

De l'antiquité à la renaissance, l'art de la cryptographie met en œuvre, pour cacher la substance d'un texte, une combinaison plus ou moins élaborée de substitutions et de permutations.

Dès l'origine, la cryptographie a été utilisée à des fins diplomatiques puis militaires

.8.1 Les techniques traditionnels

- En Egypte, 2000 ans avant notre ère un scribe avait gravé des hiéroglyphes transformés sur la pierre tombale de Khumhotep II pour rendre inintelligible la description de sa vie.
- Jules César dans la Guerre des Gaules décrit un procédé de substitution aujourd'hui bien connu: Il consiste pour chiffrer à décaler d'un nombre de rangs convenu la lettre « claire » dans l'alphabet usuel. Si la clé est 3, a est remplacé par d et b par e ...

Alphabet	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
Substitution	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

Tableau 2.01 : Chiffrement de César

- A Sparte, au IV^{ème} siècle avant notre ère, les communications entre les chefs des armées et les commandants étaient chiffrées à l'aide d'une Scytale, un bâton sur lequel on enroule une lanière en spires jointives
- En Crète, un disque de Phaïstos, datant de 1700 avant notre ère, comportant un texte chiffré sur les deux faces à été retrouvé. Ce texte n'est encore à ce jour décrypté de manière sûre.
- Il existe plusieurs manières d'assurer la confidentialité des informations échangées :

La cryptographie consistant à crypter le message transmis est sans doute la technologie qui est et a été la plus utilisée.

La stéganographie est une autre technique consistant à dissimuler le message dans un contexte innocent. Impossible donc de trouver un message si on ignore jusqu'à son existence.

La stéganographie est une discipline très ancienne qui date de l'antiquité mais dont la formalisation est récente :

- Ainsi, en 440 avant notre ère, on rase la tête d'un esclave, puis on y tatoue un message qui devint invisible après que les cheveux aient repoussés
- Cette méthode était encore utilisée par les espions allemands au début du XX siècle
- Une ancienne technique chinoise de nature différente : un texte imprimé sur de la soie était enfermé dans une boulette de cire que le messenger avalait
- L'encre sympathique est la plus connue des méthodes de stéganographie
- La dissimulation d'images ou de texte à l'intérieur d'autres images est devenue un processus de plus en plus complexe de nos jours.
- L'ère numérique ouvre de nouvelles possibilités: La stéganographie à clé secrète permet d'extraire un message d'un support numérique, le « stégano-médium ».

.8.2 L'éveil cryptographique

Avec l'invention de l'imprimerie, vers 1450, l'emploi du chiffre s'impose et se généralise dans les relations diplomatiques et au plus haut niveau du commandement militaire. La science du chiffre progresse et l'art du déchiffrement progresse également.

- Le chiffrement de Vigenère

Le procédé de Vigenère consiste à changer l'alphabet de substitution à chaque chiffrement d'une lettre, ce qui fait que l'on ne peut tenter de décrypter le message en utilisant la fréquence des lettres.

- Pour cela, on construit un carré constitué de tous les alphabets décalés d'une lettre.
- Le chiffrement part d'un mot clé. Par exemple TRIAGE. Pour coder la première lettre C du message en clair COULER, on considère la ligne 10 commençant par T, indiquant que le C doit être chiffré par son correspondant sur cette ligne, soit V. La seconde lettre O doit être chiffrée en prenant la ligne commençant par R et ainsi de suite.

Clair	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
1	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
2	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
3	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
4	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
5	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
6	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
7	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
8	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
9	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
10	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
11	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
12	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
13	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
14	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
15	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
16	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
17	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
18	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
19	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
20	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
21	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
22	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
23	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
24	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
25	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
26	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

Tableau 2.02 : Carré de Vigenère

- Le chiffrement ADFVGX

Cette technique de chiffrement fut utilisée à partir de 1918 par les allemands. Elle utilise un carré de 36 symboles (alphabet et chiffres) pour convertir ces symboles en des couples de lettres.

	A	D	F	G	V	X
A	c	1	o	f	w	j
D	y	m	t	5	b	4
F	i	7	a	2	8	s
G	p	3	0	q	h	x
V	k	e	u	l	6	d
X	v	r	g	z	n	9

Tableau 2.03 : *Tableau ADFVGX*

Six lettres sont utilisées pour construire ces couples : A, D, F, G, V et X.

Message	C	e	c	i	e	s	t	u	n	e	s	s	a	i	d	e	c	h	i	f	f	r	e	m	e	n	t
Message codé	AA	VD	AA	FA	VD	FX	DF	VF	XV	VD	FX	FX	FF	FA	VX	VD	AA	GV	FA	AG	AG	XD	VD	DD	VD	XV	DF

Tableau 2.04: *Exemple de message chiffré*

Une fois la substitution effectuée, le résultat est soumis à une permutation des colonnes du carré. On place les caractères dans un tableau à l'aide d'une clé qu'on a pris le soin de choisir avec des caractères tous différents des uns des autres.

Puis, on trie les colonnes selon les lettres de la clé avant de procéder à la lecture de haut en bas puis de gauche à droite

A	C	O	D	E	R	A	C	D	E	O	R
A	A	V	D	A	A	A	A	D	A	V	A
F	A	V	D	F	X	F	A	D	F	V	X
D	F	V	F	X	V	D	F	F	X	V	V
V	D	F	X	F	X	V	D	X	F	F	X
F	F	F	A	V	X	F	F	A	V	F	X
V	D	A	A	G	V	V	D	A	G	A	V
F	A	A	G	A	G	F	A	G	A	A	G
X	D	V	D	D	D	X	D	D	D	V	D
V	D	X	V	D	F	V	D	V	D	X	F

Tableau 2.05: *Classement et Permutation*

Ainsi, on a le texte chiffré :

AFDVFVXV AAFDFDADD DDFXAAGDV AFXFVGADD VVVFFAAVX AXVXXVGDF

- Le disque Mexicain

L'utilisation de ce disque, utilisé autrefois par l'armée mexicaine, est très facile :

Tout d'abord, il faut choisir un nombre pour ajuster l'orientation des différents disques qui sont libres les uns par rapport aux autres.

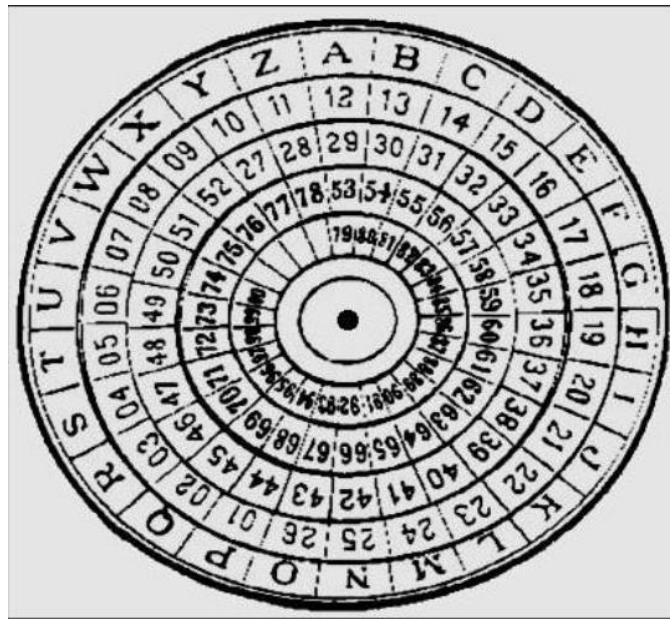


Figure 2.01 : *Disque Mexicain*

Ce nombre est composé de 8 chiffres selon le format aabbccdd, sachant que aa est compris entre 01 et 26, bb entre 27 et 53, cc entre 54 et 78 et enfin dd entre 79 et (1)00.

Alignez les quatre disques avec la lettre A en fonction des quatre nombres retenus. Chacun de ces quatre nombres qui sont situés sur des disques différents doivent être alignés avec la lettre A. le choix de l'image correspond à 12295379.

Pour crypter un message, il suffit de remplacer la lettre par l'une des 3 ou 4 valeurs numériques situées en face de la lettre à l'intérieur du disque.

Message:	C	e	c	i	e	s	t	u	n	e	s	s	a	i	d	e	c	h	i	f	f	r	e	m	e	n	t
Message codé:	14	33	55	87	16	47	72	99	25	57	71	97	12	37	15	83	31	36	20	34	58	70	16	41	83	42	05

Tableau 2.06 : Chiffrement

.8.3 L'arme cryptographique

L'enseignement que les états major tirent de la guerre est qu'il faut automatiser les opérations de chiffrement.

L'armée allemande s'équipe de machines Enigma dès 1926.

L'armée française s'équipe au milieu des années 1930, de matériel suédois: La machine C36 et la machine B211 qui resteront en service environ 20 ans.

- La machine Enigma

La machine Enigma possède 3 rotors, chaque rotor contenant les 26 lettres de l'alphabet. L'ordre de grandeur de la combinatoire est donc de $(26)^3$, c'est à dire 17576 positions initiales des rotors.

- La combinatoire est encore augmentée par le fait que les rotors sont permutables
- Vers la fin de 1938 les 3 rotors sont choisis parmi 5, ce qui fait passer les possibilités d'arrangement des rotors de 6 à 60.

- La machine de Turing (Notion de théorie de la complexité)

Le but est de classifier les problèmes algorithmiques, en fonction de leur difficulté (i.e. du nombre d'opérations élémentaires nécessaires pour les résoudre)

Pour faire fonctionner la machine de Turing, on inscrit des données sur la bande, sous la forme de mots de Σ^* (alphabet). La taille totale des données est finie. Leurs différents éléments sont

séparés par des blancs ^{*b*}. Les cases non-utilisées par les données sont aussi remplies par des

blancs. On place la tête de lecture sur le premier symbole de la donnée (extrémité gauche) et on met la machine dans l'état initial. Ensuite, la machine exécute le programme et s'arrête lorsqu'elle ne possède pas d'instruction correspondant au symbole lu et à l'état où elle se trouve. Le résultat du programme est constitué des mots alors inscrits sur la bande.

Son principe est le suivant : tous les problèmes qui peuvent être résolus à l'aide d'un ordinateur réel, actuel ou futur, programmé dans n'importe quel langage, peuvent en principe être résolus par une machine de Turing. Bien que ce fait ne puisse être prouvé rigoureusement (le concept d'ordinateur réel est trop informel, et l'intérêt principal des machines de Turing est justement de les modéliser) de nombreux travaux ont étayé cette hypothèse. En particulier, de nombreuses autres tentatives de modélisation d'ordinateurs ont vu le jour, et elles se sont ensuite révélées être équivalentes au modèle de Turing.

De plus, on peut imaginer des dispositifs apparemment plus puissants que les machines de Turing, en autorisant plusieurs bandes et/ou plusieurs têtes de lecture/écriture notamment. Mais l'examen montre que les problèmes que peuvent résoudre ces dispositifs peuvent aussi l'être simulés par une machine de Turing classiques. Inversement, on montre que toute machine de Turing peut être simulée par des machines de Turing dont l'alphabet est réduit à $\{0,1\}$. On montre aussi qu'on ne perd pas de potentialités si on impose que la bande soit finie dans l'un des deux sens. Bref, la machine de Turing, malgré son apparence simpliste et difficulté à programmer, s'est largement imposée comme modèle théorique pour analyser les algorithmes et leur complexité.

On peut remarquer qu'une machine de Turing peut être décrite à l'aide d'un nombre fini de symboles. On adoptant un encodage convenable, on peut même représenter chaque machine de Turing par un mot de $\{0,1\}^*$. Rien n'empêche alors d'utiliser la représentation d'une machine de Turing dite universelle qui, lorsqu'on lui fournit comme données la description d'une machine M et d'autres données D , le résultat obtenu étant identique à celui qu'on aurait en appliquant la machine M aux données D .

.8.4 Inconvénients

Avec la méthode de substitution et de transposition, ces systèmes classiques présentent comme faille la limitation de nombre de clés et de permutation. Ces problématiques rendent ces cryptosystèmes vulnérable à l'analyse fréquentielle. Et avec l'évolution de l'électronique, la technologie et les mathématiques, le système de chiffrement classique est devenu obsolète et cède la place au système de chiffrement moderne.

.9 Le système de chiffrement moderne

.9.1 Chiffrement à clé secrète

D'après le chapitre 1, il existe deux modèles de base d'algorithmes à clé secrète : algorithme de chiffrement par blocs et algorithme de chiffrement en continu. Quelques notions de base à propos de ces deux types de chiffrement seront décrites précédemment.

.9.1.1 Communications à l'aide d'un cryptosystème à clé secrète

Nous allons voir comment l'émetteur A peut envoyer un message chiffré au destinataire B :

- 1) A et B choisissent un cryptosystème ;
- 2) A et B choisissent une clé ;
- 3) A chiffre son texte en clair à l'aide de l'algorithme choisi et avec la clé sélectionnée. Cette étape produit un texte chiffré ;
- 4) A envoie le texte chiffré à B ;
- 5) B déchiffre le texte chiffré avec le même algorithme et la même clé, et finalement lit le message.

Avec un algorithme à clé secrète, A et B peuvent réaliser l'étape 1 en public mais ils doivent réaliser l'étape 2 en secret. La clé doit être restée secrète avant, pendant et après le protocole, sinon, le message ne serait plus confidentiel.

.9.1.2 Système de chiffrement en continu

Les systèmes de chiffrement en continu sont spécifiquement utiles pour chiffrer des communications continues. Le même bit ou octet du texte en clair sera chiffré en un bit ou un octet différent à chaque chiffrement.

a. Principe de base

La plupart des conceptions d'algorithmes de chiffrement en continu tournent autour des Registres de Décalage à Rétroaction Linéaire ou RDRL.

b. Le RDRL

Le RDRL est un circuit combinatoire simple composé de deux parties :

- Un registre à décalage : c'est une suite de bits et chaque fois qu'un bit est nécessaire, tous les bits du registre sont décalés vers la droite de un bit.
- La fonction de rétroaction : c'est une simple combinaison par *ou exclusif* de certains bits du registre. La liste de ces bits s'appelle séquence de dérivation.

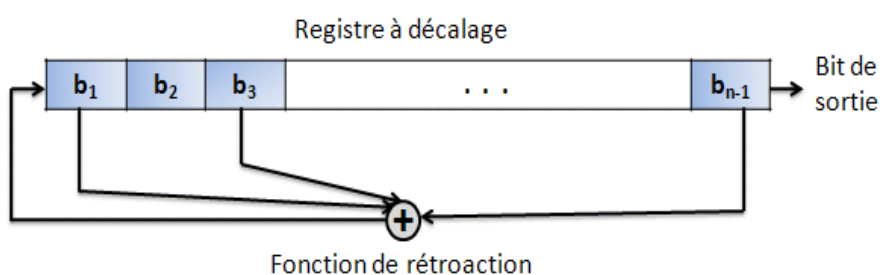


Figure 2.02 : Principe du RDRL

c. Générateur de codons

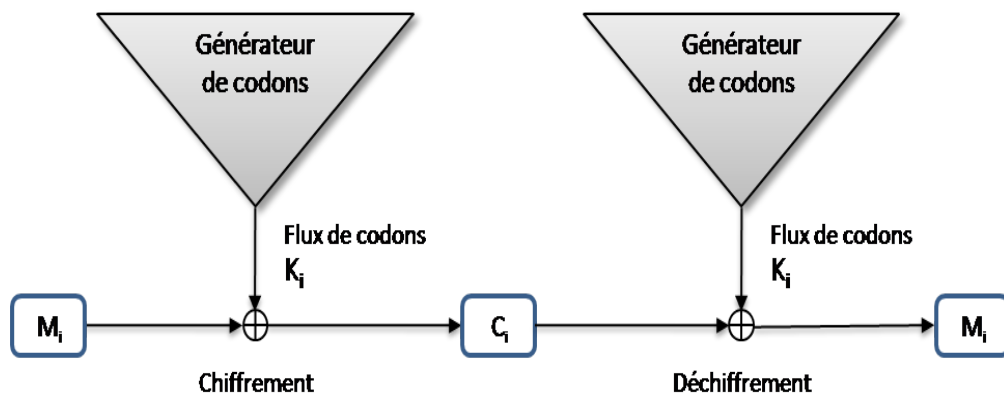


Figure 2.06: Principe du générateur de codons

La réalisation la plus simple d'un algorithme de chiffrement en continu est illustrée par la *Figure 2.06*. Un générateur de codons engendre un flux de bits k_1, k_2, \dots, k_i . Ce flux est combiné par *ou exclusif* avec le flux de bits du texte en clair m_1, m_2, \dots, m_i , pour produire le flux de bits du texte chiffré.

$$c_i = m_i \oplus k_i$$

Du côté de déchiffrement, les bits du texte chiffré sont combinés par *ou exclusif* avec un flux identique de codons pour retrouver les bits du texte en clair :

$$m_i = c_i \oplus k_i$$

d. Générateur périodique

Comme le générateur de codons doit engendrer le même flux au chiffrement et au déchiffrement, il doit être déterministe. Comme il est réalisé par une machine ou matériel à états finis, la suite finira par se répéter. Ces générateurs de codons sont appelés générateurs périodiques.

Un générateur de codons doit avoir une longue période, bien plus longue que le nombre de bits dont le générateur devra produire entre deux changements de clé. Si la période est plus courte que le texte en clair, alors différentes parties du texte en clair seront chiffrées de la même manière. Si un cryptanalyste connaît une partie du texte en clair, il peut reconstituer une partie du flot de clés et l'utiliser pour en savoir plus sur le texte en clair.

Même si le cryptanalyste n'a que le texte chiffré, il peut combiner par *ou exclusif* les parties chiffrées avec le même flot de clé et obtenir les combinaisons par *ou exclusif* du texte en clair avec ce dernier.

Message crypté : $c_i = m_i \oplus k_i$;

Cryptanalyste : $c_i \oplus k_i = m_i \oplus k_i$.

C'est juste un algorithme de combinaison par *ou exclusif* avec une très longue clé.

e. RC4

Conçu par Ron Rivest, RC4 est un algorithme de chiffrement en continu, pouvant utiliser des clés de taille variables jusqu'à 2048 bits ce qui réduit bien évidemment la possibilité d'attaques.

Le flux de codons ne dépend pas du texte en clair. Il a une table-S à 8×8 bits : $T[0], \dots, T[255]$. Il y a deux compteurs x et y initialisés à zéro. Pour générer un octet aléatoire b , voici l'algorithme :

$x \leftarrow x + 1 \text{ modulo } 256$

$y \leftarrow T[x] + y \text{ modulo } 256$

échanger $T[x]$ et $T[y]$

$b \leftarrow T[x] + T[y] \text{ modulo } 256.$

L'octet b est combiné par *ou exclusif* avec le texte en clair pour produire le texte chiffré ou bien avec le texte chiffré pour produire le texte en clair.

L'initialisation est facile, on commence avec l'identité : $T[0] = 0, T[1] = 1, \dots, T[255] = 255$. Notons que la clé est devenue une suite de 256 octets en concaténant autant de copies nécessaires.

Voici l'algorithme correspondant :

$c \leftarrow 0$

Pour i de 0 à 255

$c \leftarrow K[i] + T[i] + c \text{ modulo } 256$

Echanger $T[i]$ et $T[c]$

Fin *pour* i

f. Sécurité d'un système de chiffrement

La sécurité d'un système de chiffrement en continu dépend entièrement des détails internes du générateur de codons. Si le générateur de codons engendre un flux continu de zéros, le texte chiffré résultant sera identique au texte en clair et toute l'opération sera inutile.

Le générateur de codons engendre un flux de bits qui ont l'air aléatoires, mais en fait, il est déterministe et il peut être reproduit de façon infaillible au moment du déchiffrement. Plus la sortie du générateur est aléatoire, plus il est difficile pour le cryptanalyste de le casser. Si, néanmoins, le générateur engendre le même flot de bits chaque fois qu'on l'active, le système cryptographique résultant sera facile à casser.

Si le cryptanalyste a un texte chiffré et le texte en clair correspondant, il peut combiner par *ou exclusif* le texte en clair avec le texte chiffré pour retrouver le flux de codons. Maintenant, chaque fois qu'il intercepte un message chiffré, il a les codons nécessaires pour déchiffrer le message. De plus, il peut déchiffrer et lire tout texte chiffré qu'il aurait intercepté avant.

Quand le cryptanalyste obtient une seule paire « texte en clair, texte chiffré », il peut tout lire. C'est pourquoi tous les algorithmes de chiffrement en continu utilisent des clés. La sortie du générateur de codons est une fonction de la clé. Maintenant, s'il a une paire « texte en clair, texte chiffré », il peut seulement lire les messages chiffrés avec une seule clé. Lorsqu'on change la clé, l'adversaire se retrouve à la case de départ.

.9.1.3 Système de chiffrement par bloc

a. Principe de base

Le but des cryptanalystes est de déterminer la clé K , le texte en clair M , ou les deux. La plupart des cryptanalystes du monde réel ont quelques informations probabilistes concernant M avant de commencer. Ils utilisent la redondance naturelle du langage pour réduire le nombre possible de

textes en clair.

b. *La confusion et la diffusion*

Selon Shannon, les deux techniques de base pour effacer les redondances dans un texte en clair sont la confusion et la diffusion. Les algorithmes par blocs utilisent à la fois ces deux techniques.

La confusion sert à cacher les relations entre le texte en clair, le texte chiffré et la clé. Une bonne confusion rend les relations statistiques si compliquées que même les outils de cryptanalyses puissants ne marchent pas. Le moyen le plus simple de réaliser cela est la substitution que nous avons exposée dans le paragraphe précédent

La diffusion disperse les redondances du texte en clair en les répartissant dans le texte chiffré. Un cryptanalyste qui cherche ces redondances aura plus de difficulté à les trouver. Le moyen le plus simple de provoquer cela est la transposition.

c. *Réseaux de Feistel*

La plupart des algorithmes de chiffrement par blocs sont des réseaux de Feistel. Le principe c'est qu'un bloc de longueur n bits est divisé en deux moitiés de longueur $n/2$: L (Left) et R (Right).

Notons que n doit être pair. On peut définir un algorithme de chiffrement par blocs itératif où la sortie de la i -ème ronde est déterminée d'après la sortie de la ronde précédente :

$$L_i = R_{i-1} \text{ et } R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \quad (2.03)$$

K_i est la sous clé utilisée dans la i -ème ronde et f est une fonction quelconque.

d. *Table de substitution ou table-S*

La solidité d'une variété de réseaux de Feistel est directement reliée à leurs tables-S. Une table-S est une simple substitution : une application de m bits d'entrée vers n bits de sortie. La table avec m bits en entrée et n bits en sortie s'appelle une table-S à $m \times n$ bits. Les tables-S constituent en général la seule étape non linéaire d'un algorithme ; elles sont ce qui lui donne sa sécurité. En général, plus elles sont grandes, meilleurs elles sont.

$\begin{smallmatrix} c \\ r \end{smallmatrix}$	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
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

r: rang

c: colonne

Tableau 2.07 : *Détail d'une table-S de l'algorithme DES*

e. *Mode de chiffrement*

Un mode cryptographique combine en général un algorithme avec une sorte de rétroaction et des opérations simples.

- Mode ECB (ElectronicCodeBook)

Le mode du carnet de codage électronique ou ECB est la méthode la plus évidente pour utiliser un algorithme de chiffrement par blocs : un bloc de texte en clair se chiffre en un bloc de texte chiffré.

Comme un même bloc de texte en clair sera toujours chiffré en un même bloc de texte chiffré, il est théoriquement possible de créer un carnet de codage de textes en clair et de textes chiffrés correspondants. C'est le mode opératoire le plus simple à utiliser et chaque bloc est chiffré indépendamment des autres blocs.

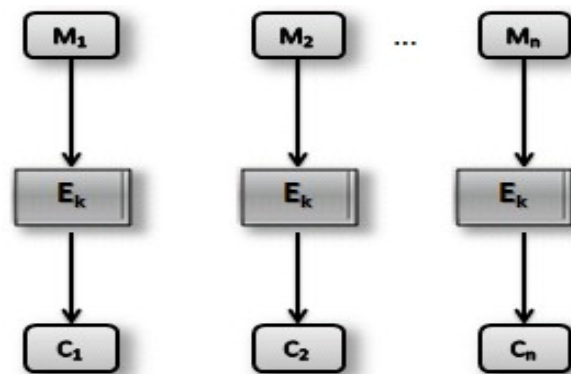


Figure 2.07: Mode ECB

Le défaut du mode ECB est que si un cryptanalyste obtient le texte en clair et le texte chiffré de plusieurs messages, il peut commencer à construire un carnet de codage sans connaître la clé. Pourtant, si la taille du bloc est de 64 bits, le carnet de codage aura 2^{64} entrées, ce qui est assez grand pour être précalculé et stocké ; de plus, chaque clé a un carnet différent.

- Mode CBC (Cipher Block Chaining)

En mode chiffrement avec chaînage de blocs ou CBC, le texte en clair est combiné par *ou exclusif* avec le bloc chiffré précédent avant d'être chiffré.

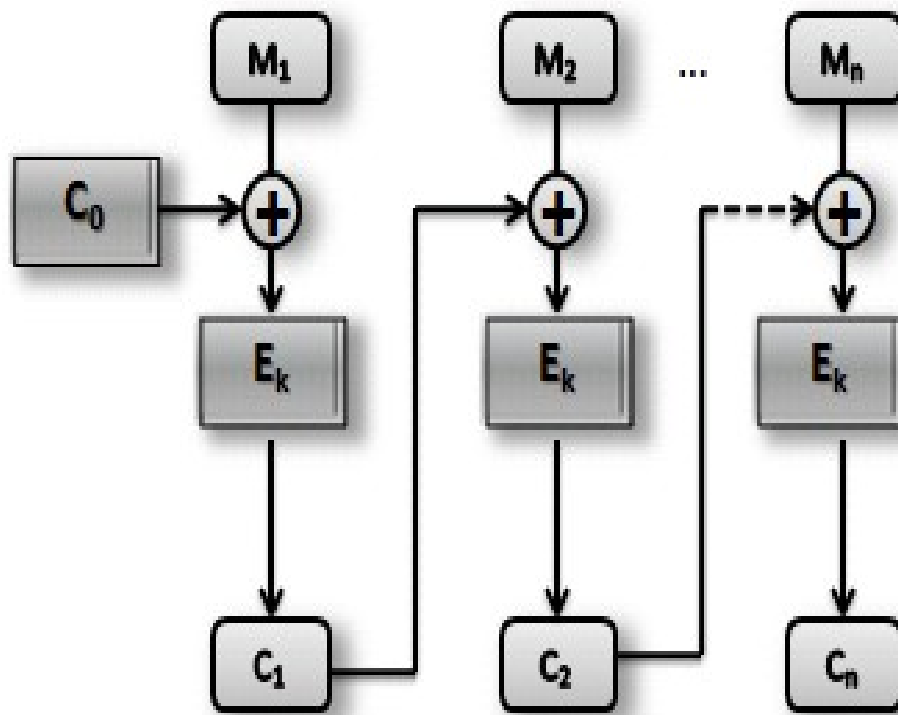


Figure 2.08 : Mode CBC

Après qu'un bloc de texte en clair a été chiffré, le texte chiffré correspondant est aussi stocké dans un registre de rétroaction. Avant que le bloc du texte en clair suivant soit chiffré, il est combiné par *ou exclusif* avec le registre de rétroaction pour devenir la nouvelle entrée de l'algorithme de chiffrement. Le texte chiffré résultant est à nouveau stocké dans le registre pour être combiné par *ou exclusif* avec le bloc de texte en clair suivant, et ainsi de suite jusqu'à la fin du message. Le chiffrement de chaque bloc dépend de tous les blocs précédents.

Un bloc de texte chiffré est déchiffré normalement, et aussi sauvé dans le registre de rétroaction. Une fois que le bloc suivant a été déchiffré, il est combiné par *ou exclusif* avec le contenu du registre de rétroaction. Ensuite, le bloc suivant de texte chiffré est sauvé dans le registre de

rétroaction, et ainsi de suite jusqu'à la fin du message.

Le mode CBC présente deux inconvénients : d'abord, il impose de déchiffrer la totalité du fichier avant de pouvoir accéder à une partie de ce fichier ; ensuite, une erreur sur un bloc va se répercuter sur tous les blocs suivants. Ainsi, avec ce mode, on sera très attentif au contrôle d'erreurs.

- Mode CFB (CipherFeedBack)

En mode CFB, les données peuvent être chiffrées par unités plus petites que la taille d'un bloc. On peut utiliser un mode CFB à n bit où n est inférieur ou égal à la taille du bloc. Ce mode manipule une file d'attente de la taille d'un bloc d'entrée.

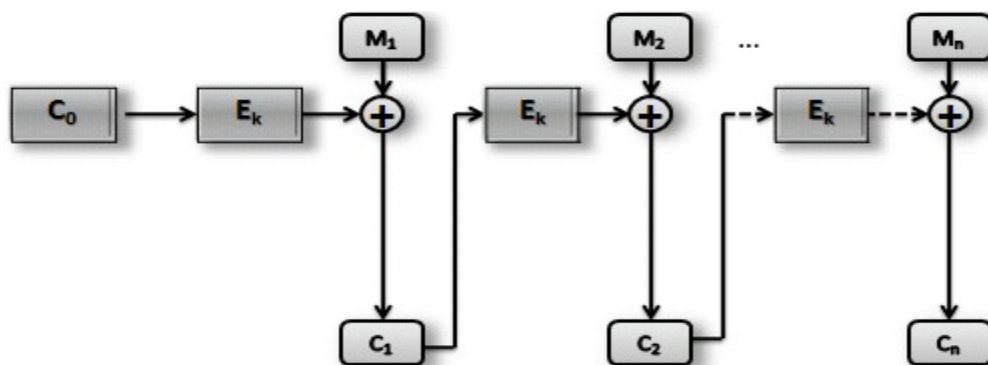


Figure 2.09: Mode CFB

Comme le mode CBC, une erreur dans le texte chiffré en mode CFB affecte tous les textes chiffrés suivants. En mode CFB, l'algorithme de chiffrement par blocs est utilisé comme un algorithme de chiffrement autosynchrone en continue que nous allons voir dans la suite.

- Mode OFB (Output-FeedBack)

Le mode de rétroaction de sortie ou OFB est similaire au mode CFB, sauf que le bloc de sortie précédent est mis dans la file d'attente. C'est une méthode qui consiste à utiliser un algorithme de chiffrement par blocs comme un algorithme de chiffrement synchrone en continu.

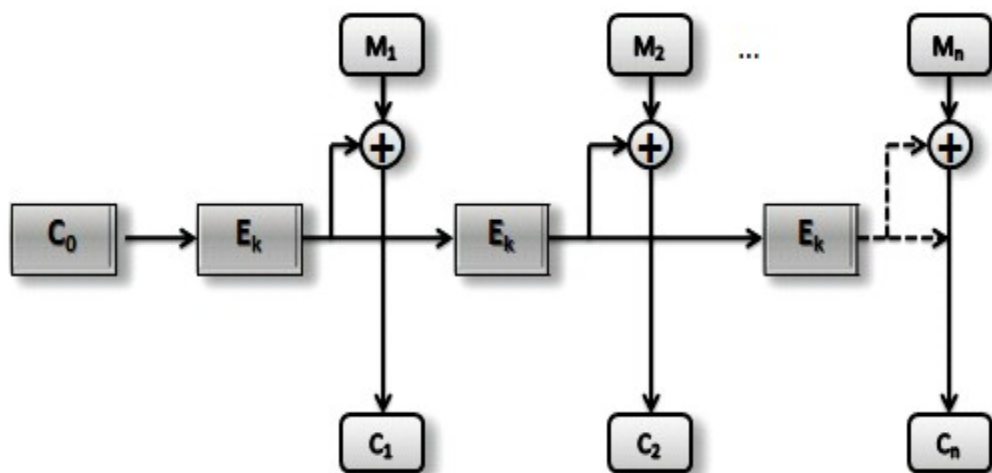


Figure 2.10 : Mode OFB

Avec le mode OFB, il n'y a pas d'amplification d'erreur c'est-à-dire, une erreur d'un seul bit dans le texte chiffré n'occasionne qu'une erreur d'un seul bit dans le texte en clair récupéré.

- Choix du mode

Si la simplicité et la vitesse sont les critères principaux, le mode ECB est le mode le plus rapide et facile à utiliser. Pour le chiffrement de données aléatoires, l'ECB est aussi un bon mode à utiliser tandis que pour un texte en clair normal, on utilise souvent les modes CBC, CFB, ou OFB. Le mode CBC convient en général pour chiffrer des fichiers. D'ailleurs, si l'application est une réalisation logicielle, ce mode est presque toujours le meilleur choix.

Le mode CFB est généralement le mode de choix pour chiffrer des flots de caractères quand chaque caractère doit être traité individuellement. Le mode OFB est généralement utilisé pour les systèmes synchrones à grande vitesse où la propagation d'erreurs est inacceptable. Le mode OFB est le mode de choix dans un environnement où il y a beaucoup d'erreurs, car il ne les propage pas.

.9.1.4 Le DES et son successeur

a. Historique.

Jusque dans les années 1970, seuls les militaires possédaient des algorithmes à clé secrète fiables. Devant l'émergence des besoins civils, le NBS (National Bureau of Standards) lança le 15 mai 1973 un appel d'offre dans le Fédéral Register (l'équivalent du journal Officiel américain) pour la création d'un système cryptographique. Le cahier des charges était le suivant

- l'algorithme repose sur une clé relativement petite, qui sert à la fois au chiffrement et au déchiffrement.

- l'algorithme doit être facile à implémenter, logiciellement et matériellement, et doit être très rapide.

Le chiffrement doit avoir un haut niveau de sûreté, uniquement lié à la clé, et non à la confidentialité de l'algorithme.

Les efforts conjoint d'IBM, qui propose Lucifer (fin 1974), et de la NSA (National Security Agency) conduisent à l'élaboration du DES (Data Encryption Standard), l'algorithme de chiffrement le plus utilisé au monde durant le dernier quart du XX^e Siècle.

b. Schéma général.

Le DES utilise une clé K de 56 bits, pour chiffrer des blocs de 64 bits, les blocs chiffrés obtenus ayant aussi 64 bits. Le bloc de texte clair subit d'abord une permutation initiale. Puis on itère 16 fois une procédure identique décrite ci-après, où la moitié droite est recopiée telle qu'elle à gauche, et la moitié gauche est transmise à droite en subissant au passage une modification dépendante de la clé. A la fin, on inverse les moitiés gauches et droites (ou bien, comme sur le schéma, on supprime le croisement de la dernière étape), et on applique l'inverse de la permutation initiale pour obtenir le bloc chiffré. Le schéma général de DES est donc le suivant (on a seulement représenté quelques-unes des 16 étapes) :

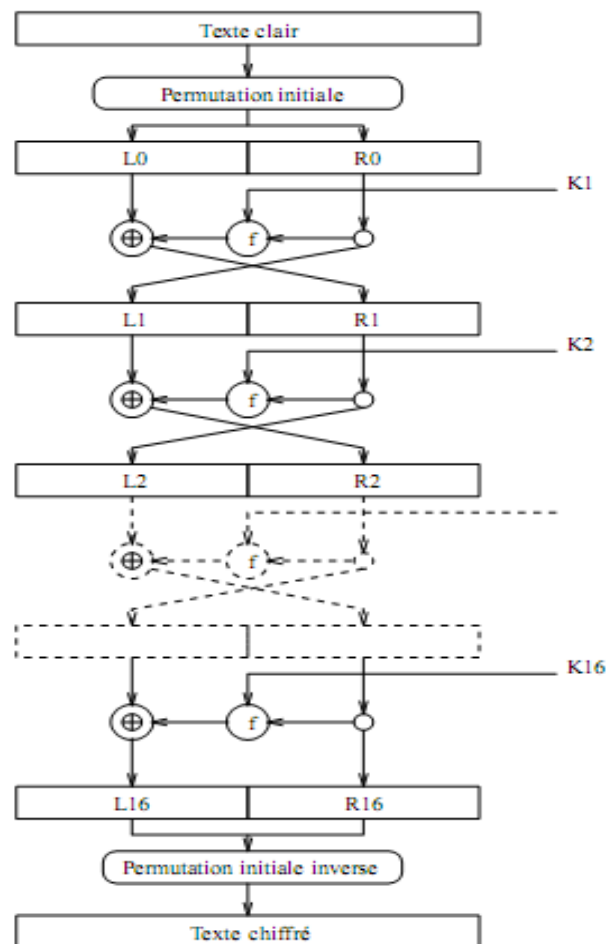


Figure 2.31 : DES : Schéma général

Le tableau suivant montre la permutation initiale et son inverse

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

Tableau 2.08 : La permutation initiale et son inverse

La permutation initiale et son inverse sont décrits par le tableau 2.08. Les tableaux se lisent de gauche à droite et de haut en bas, le n-ième nombre est la position avant permutation du bit qui se trouve en n-ième position après permutation.

Après la permutation initiale, le message est séparé en deux moitiés de 32 bits, désignées par L_0 et R_0 . A chaque itération de la procédure, on détermine deux groupes de 32 bits L_j et R_j en fonction de L_{i-1} et R_{i-1} obtenus précédemment. Pour cela, on utilise une clé intermédiaire K , de 48 bits, calculée à partir de K et on applique les formules suivantes :

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

La fonction f est schématisée par la figure ci-dessus. Tout d'abord, l'argument de gauche qui possède 32 bits est expansé en 48 bits en redoublant certains bits. Ensuite, on calcule le *ou exclusif* de cet argument expansé avec le deuxième argument (qui n'est autre que la clé K_i).

Le résultat possède 48 = 8x6 bits et est transformé en une chaîne de 32 = 8 x 4 bits en utilisant des dispositifs appelés boîte-S qui calculent un bloc de 4 bits à partir d'un bloc de 6 bits. Enfin on applique la permutation décrite par la figure à ces 32 bits pour obtenir la valeur de f

c. Les boîtes-S

Il y a huit boîtes-S différentes. On les représente par deux tableaux à 4 lignes et 16 colonnes. Les premiers et derniers bits de l'entrée déterminent une ligne du tableau, les autres bits déterminent une colonne. La valeur numérique trouvée à cet endroit indique la valeur des quatre bits de sortie.

.9.1.5 Le triple DES

Puisque la faiblesse de DES est la faible longueur de sa clé, il est naturel de chercher à combiner plusieurs chiffrements pour obtenir un système de chiffrement global avec une clé plus longue.

La première idée qui vient à l'esprit est de composer deux chiffrements DES avec des clés différentes. Mais on peut alors monter contre ce « double-DES » une attaque à message clair dite « par le milieu » parce qu'elle s'appuie sur le message intermédiaire apparaissant entre les deux chiffrements DES successifs. Cette attaque consiste à construire la liste des messages intermédiaires possibles en chiffrant par DES un clair avec les 2^{56} clés possibles.

En déchiffrant par DES le chiffré correspondant avec des clés différentes, on obtient une autre liste de messages intermédiaires possibles et le véritable message intermédiaire est dans l'intersection des deux listes. Le coût en mémoire de cette attaque est très important mais son coût en temps n'est pas significativement plus élevé que l'attaque exhaustive sur DES.

Triple-DES consiste à composer deux chiffrements DES de même clé séparées par un déchiffrement DES avec une autre clé. Plus précisément :

$$Triple\ e - DES_{K_1, K_2} = DES_{K_1} \circ DES_{K_2}^{-1} \circ DES_{K_1}$$

Cette façon de faire est préférée à trois chiffrements parce qu'elle généralise DES qui se trouve être le cas particulier où $AT, = K_2$. Bien sûr, le déchiffrement est :

$$Triple - DES_{K_1, K_2}^{-1} = DES_{K_1}^{-1} \circ DES_{K_2} \circ DES_{K_1}^{-1}$$

Une clé triple DES est donc composée de deux clés DES et fait 112 bits ce qui met Triple DES largement hors de portée d'une attaque exhaustive. On peut aussi concevoir une variante à trois clés DES différentes mais elle reste vulnérable à une attaque de coût en 2^{112} s'appuyant sur l'un des deux messages intermédiaires.

.9.1.6 L'IDEA

Développé à Jurich en Suisse par Xuejia Lai et James Massey en 1992, L'International Data Encryption Algorithm (IDEA) a été proposé pour remplacer le DES. Il utilise une clé de 128 bits, réalise un chiffrement par blocs de 64 bits, en opérant 8 rondes d'une même fonction. La description de cette fonction est un peu compliquée, elle utilise seulement trois opérations :

- Ou exclusif
- Addition module 2^{16}
- Multiplication modulo 2^{16}

IDEA est particulièrement adapté aux réalisations logicielles. Il est aussi efficace même sur des processeurs de 16 bits.

a. Algorithme.

Le bloc de 64 bits en entrée est tout d'abord divisé en quatre blocs de 16 bits : $X1, X2, X3, X4$. La

clé K est divisée en 8 blocs de 16 bits, puis décalée circulairement sur la gauche de 25 bits, et redécoupée, et ainsi de suite jusqu'à obtenir 52 clés. Ces clés formeront 8 groupes de 6 clés (un groupe par ronde) : $K1, K2, K3, K4, K5, K6$, un groupe de 4 clés pour la ronde finale :

$K1, K2, K3, K4.$

Etapes des 8 rondes :

$$Etape1 = X1 * K1$$

$$Etape2 = X2 * K2$$

$$Etape3 = X3 * K3$$

$$Etape4 = X4 * K4$$

$$Etape5 = Etape1 \oplus Etape3$$

$$Etape6 = Etape2 \oplus Etape4$$

$$Etape7 = Etape5 \oplus K5$$

$$Etape8 = Etape6 + Etape7$$

$$Etape9 = Etape8 * K6$$

$$Etape10 = Etape7 + Etape9$$

$$Etape11 = Etape1 \oplus Etape9 \Rightarrow \text{X1 de la ronde suivante}$$

$$Etape12 = Etape3 \oplus Etape9 \Rightarrow \text{X3 de la ronde suivante}$$

$$Etape13 = Etape2 \oplus Etape10 \Rightarrow \text{X2 de la ronde suivante}$$

$$Etape14 = Etape4 \oplus Etape10 \Rightarrow X4 \text{ de la ronde suivante}$$

Pour finir, on applique une étape supplémentaire après la huitième ronde :

$$C1 = X1 * K1$$

$$C2 = X2 + K2$$

$$C3 = X3 + K3$$

$$C4 = X4 * K4$$

Les 4 blocs C1, C2, C3, C4, forment alors le message chiffré.

Les 52 sous clés générées à partir de la clé de 128 bits sont produits comme suit :

- La clé de 128 bits est divisée en 8 blocs. Ces 8 blocs sont en fait les 8 premiers sous clés utilisées dans le chiffrement.
- La clé de 128 bits est ensuite cycliquement décalée de 25 positions et divisée en 8 blocs de 16 bits. Ces 8 blocs sont les 8 sous clés suivantes utilisées dans le chiffrement.
- Le cycle est répété jusqu'à ce que les 52 sous clés soient toutes générées.

b. Déchiffrement :

Le déchiffrement s'effectue essentiellement à la même manière que le chiffrement. La seule différence est que les 52 sous clés sont générées de façon inverse au chiffrement. Aussi les blocs

de textes chiffrés doivent être traités dans l'ordre inverse de chiffrement pour inverser parfaitement le processus de chiffrement.

.9.1.7 L'AES ou Rijndael.

L'Advanced Encryption Standard est issu d'une compétition et d'une expertise qui s'échelonne de l'appel à candidature par le NIST en 1997 jusqu'à la sélection finale du candidat Rijndael en octobre 2000. Tout comme son prédécesseur, ce standard a été l'occasion de formaliser des critères de conception réunissant l'état d'art des dernières connaissances en cryptographie.

Trois critères principaux ont été respectés dans sa conception :

- Résistance face à toutes les attaques connues
- Rapidités du code sur la plus grande variété de plateforme possible.
- Simplicité dans la conception.

La structure globale de l'AES à 128 bits de clé est décrite à la figure 2.13. Dans l'AES, le bloc comporte 128 bits. La clé peut comporter 128, 196 ou 256 bits, ce qui influence le nombre de tours du chiffrement. La version à 128 bits, comporte 10 itérations.

L'algorithme de cadencement de clé produit des sous clés de la même taille que la clé maîtresse.

La première itération est précédée par l'addition de la sous-clé numéro 0 qui correspond à la clé-maitre et la dernière itération ne comporte pas de *MixColumns* ou fonction de brouillage de colonne. La fonction itérée se compose du quadruplet de fonctions (*SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*), cette dernière fonction étant simplement l'addition bit-à-bit de la sous clé au bloc courant. On y retrouve trois étapes, conformément aux principes fondamentaux de confusion et de diffusion énoncés par Shannon. La première étape, dite de confusion, la fonction *SubBytes*,

consiste à appliquer à chacun des 16 octets de l'entrée une même permutation S . Cette fonction correspond (à une application affine près) à la fonction inverse dans le corps fini à 2^8 éléments ; elle assure la résistance de l'algorithme aux attaques différentielle et linéaire.

La phase de diffusion est composée des fonctions *ShiftRow* et *MixColumns* qui représentent des opérations simples sur le corps à 2^8 éléments. Enfin on effectue un Ou exclusif bit-à-bit entre le résultat et la sous-clé de l'itération.

Les sous clés de 128 bits, numérotées de 0 à 10, sont dérivées de la clé secrète de la manière suivante : le sous-clé numéro 0 correspond à la clé secrète ; ensuite, la sous-clé numéro i (utilisée à la i -ème itération) est obtenu à partir de la sous-clé numéro $(i-1)$ grâce à l'algorithme décrit à la

figure 2.13. On permute de manière circulaire, par la fonction *RotWord*, les quatre derniers octets de la clé numéro $(i-1)$, puis on leur applique la fonction *Sub Word* composée de 4 permutations

S . Après avoir ajouté une constante (dépendant de i) au premier octet (les trois autres octets de la constante $C[i]$ du schéma sont nuls), on effectue une addition bit-à-bit entre les quatre octets ainsi obtenus et les quatre premiers octets de la sous-clé précédente.

Les trois autres blocs de quatre octets de la clé numéro i sont ensuite simplement le résultat d'un ou exclusif entre le bloc correspondant de la sous-clé $(i-1)$ et le bloc précédent de la sous-clé i .

Toutes les opérations réalisées lors de chiffrage sont réversibles à condition d'avoir la clé. Pour déchiffrer, on procède à l'extension de la clé de la même manière qu'un chiffrage. Les additions

par Ou exclusifs lors de l'opération d'addition de la clé de la tour sont réversibles. L'opération de transformation *SubBytes* est inversée en utilisant la table-S inversée. Les décalages de l'opération de décalage (*ShiftRows*) sont inversés, c'est-à-dire vers la droite. La manipulation matricielle de l'opération de brouillage (*MixColumns*) nécessite une inversion de la matrice. Une fois la matrice inversée obtenue, la manipulation est la même que l'opération de brouillage des colonnes.

.9.1.8 Blowfish

Blowfish est un algorithme qui a été conçu par Bruce Schneider. C'est un algorithme de chiffrement par blocs de 64 bits à longueur de clé variable. L'algorithme se fait en deux phases : expansion de la clé et chiffrement des données. L'expansion de la clé convertit une clé de taille inférieure à 448 bits en plusieurs tableaux de sous-clés d'une taille totale de 4168 octets.

Le chiffrement est obtenu en itérant 16 fois une fonction simple. Chaque ronde consiste en une permutation dépendante de la clé et une substitution dépendante de la clé et des données. Toutes les opérations sont des additions et des *ou exclusif* sur des nombres de 32 bits. La seule opération supplémentaire est la consultation des quatre tableaux indexés par ronde.

Blowfish utilise un grand nombre de sous-clés. Ces sous-clés doivent être précalculées avant tout chiffrement ou déchiffrement.

Le tableau-P est un tableau constitué de 18 sous-clés de 32 bits :

$$P_1, P_2, \dots, P_{18}$$

Il y a 4 tables-S de 32 bits ayant 256 entrées chacune :

$$S_{1,0}, S_{1,1}, \dots, S_{1,255}$$

$$S_{2,0}, S_{2,1}, \dots, S_{2,255}$$

$$S_{3,0}, S_{3,1}, \dots, S_{3,255}$$

$$S_{4,0}, S_{4,1}, \dots, S_{4,255}$$

Blowfish est un réseau de Feistel constitué de 16 rondes. L'entrée est un élément de données x de 64 bits. Pour chiffrer, procéder de la manière suivante :

Partager x en deux moitiés de 32 bits : L et R ;

Pour i variant de 1 à 16, effectuer :

$$L \leftarrow L \oplus P_i$$

$$R \leftarrow f(L) \oplus R$$

Echanger L et R

Echanger L et R (annuler le dernier échange)

$$R \leftarrow R \oplus P_{17}$$

$$L \leftarrow L \oplus P_{18}$$

Remettre L et R bout à bout.

La fonction f est la suivante :

Diviser L en quatre parts de 8 bits a, b, c, d .

$$f(L) = \left((S_{1,a}, S_{2,b} \bmod 2^{32}) \oplus S_{3,c} \right) + S_{4,d} \bmod 2^{32}$$

Le déchiffrement s'effectue exactement de la même manière en inversant l'ordre des P_i .

.9.2 Chiffrement à clé publique

.9.2.1 RSA

En 1978, l'algorithme à clé publique de Ron Rivest, A Shamir, Léonard Adleman apparaît. C'est un système à clé publique car l'algorithme n'est pas caché, ni la clé de chiffrement.

Son fonctionnement est basé sur la difficulté de factoriser de grands entiers.

a. Fonctionnement.

Pour chiffrer le message l'émetteur va utiliser la clé publique que le destinataire a préalablement publiée. La clé publique est un ensemble de lettre et chiffre qui vont permettre à quiconque veut lui envoyés des messages confidentiels. Cela veut dire que tout le monde peut connaître la clé publique qui permet de crypté»les messages uniquement au destinataire qui a publié la clé mais seul ce dernier pourra déchiffrer ce qui a été codé avec sa clé publique grâce à sa clé privée.

b. Génération de clé :

- Choisir deux nombres entier premier p et q, de l'ordre de 100 chiffres au minimum, pour rendre la factorisation hors de porter, même avec les meilleurs ordinateurs.

1) Calculer $n = p * q$

2) Calculer $m = (p - 1)(q - 1)$

3) Choisir un nombre entier e tel que $e > 2$ et $\text{pgcd}(e, n) = 1$

4) Calculer d tel que $d * e \bmod m = 1$

On prendra comme clé publique e et n et comme clé privée d et n .

c. Chiffrement

Connaissant la clé publique e et n , et le message à chiffrer M . On peut le chiffrer de la manière suivante, le message doit être remplacé par un chiffre. Ensuite on découpera le message par bloc de x longueurs avec $x < n$. Le bloc B est chiffré par la formule :

$$C = B^e * \text{mod}(n)$$

C'est un bloc de message chiffré à envoyer vers le destinataire.

d. Déchiffrement

Pour le déchiffrement on va pratiquer quasiment de la même façon que le chiffrement mais avec la formule inverse :

$$b = C^d * \text{mod}(n)$$

Ce qui permettra au destinataire de trouver le message clair.

e. Illustration.

Soit l'exemple suivant

$p = 47$, $q = 59$ et $n = pq = 2773$, d est premier avec $(p-1)(q-1) = 2668$ est choisi égal à 157.

On trouve alors $e = 17$. En codant avec la règle: espace=00, A=01, B=02,...,Z=26 le message :

IT'S ALL GREEK TO ME devient $M=0902\ 1900\ 0112\ 1200\ 0718\ 0505\ 1100\ 2015\ 0013\ 0500$

Découpé en bloc de quatre chiffres. Le premier bloc $M_1=0920$ donne $C_1 = 920^{17} = 948[2773]$.

On répète l'opération pour les blocs suivants et on obtient à la fin :

$C = 0948\ 2342\ 1084\ 1444\ 2663\ 2390\ 0778\ 0774\ 0919\ 1655$

Pour déchiffrer C_1 , on calcule $948^{157}[2773]$ et on obtient bien 920, soit 09 20, c'est-à-dire IT.

Ce système est simple, facile à programmer et relativement rapide pour coder et décoder. Les algorithmes de factorisation sont beaucoup plus long que le test de primalité ce qui rend le système RSA relativement sûr et couramment utilisé.

.9.2.2 Cryptosystème d'El Gamal

Dans ce système, on suppose que les blocs de message clair sont numérisés dans les entiers

modulo p. On commence par choisir un grand nombre premier p , et un nombre $g \pmod{p}$, qui

sont tous deux connus de tous. L'utilisateur A choisit un grand nombre $a \pmod{p-1}$ qui sera sa

clé secrète de déchiffrement. La clé publique de A est le nombre $g^a \pmod{p}$.

Pour envoyer un message m à A, l'utilisateur B choisit aléatoirement un grand nombre entier

$k \pmod{p}$, et il l'envoie à A la paire

(K, M) , où $K = (g^k \pmod{p})$, et $M = (mg^{a-k} \pmod{p})$.

Le receveur A, qui connaît la clé secrète a , récupère le message m à partir de cette paire de la

façon suivante. Il calcule d'abord $(K^{-a} \bmod p) = (g^{-aK} \bmod p)$, à partir du premier élément du couple reçu ; puis il multiplie M par ce résultat pour obtenir :

$$Mg^{aK} \equiv (mg^{aK})g^{-aK}(\text{mod } p)$$

$$\equiv mg^{aK-aK}(\text{mod } p)$$

$$\equiv m$$

Intuitivement, le message chiffré C envoyé à A est une version masquée de m obtenue par la multiplication par g^{ak} . Le nombre K , qui accompagne le message chiffré C , est un indice qui permet à A de retirer le masque. Cet indice $K = (g^K \text{ mod } p)$ ne peut être utilisé que par quelqu'un qui connaît la clé a . Il semble que pour qu'un cryptanalyste puisse casser le cryptosystème de El Gamal, il doive retrouver la clé a à partir de la clé publique g^a . C'est donc dire qu'il aura trouvé une solution efficace au problème du calcul du logarithme discret.

.9.2.3 PGP

Le PGP (Pretty Good Privacy) est un cryptosystème inventé par Phil Zimmermann en 1991. Il combine à la fois les meilleures fonctionnalités de la cryptographie à clé privée et de la cryptographie à clé publique. PGP est donc un système hybride

a. Chiffrement.

Quand un utilisateur chiffre du texte clair avec PGP, PGP compresse d'abord le texte clair. La compression de données économise le temps de transmission des données et de l'espace disque et, ce qui est important, renforce la sécurité cryptographique. La plus part des techniques de cryptanalyse exploitent les redondances trouvés dans le texte clair pour craquer le texte chiffré. La

compression réduit ces redondances dans le texte clair, ce qui augmente grandement la résistance à la cryptanalyse.

PGP crée ensuite une clé de session, qui est une clé secrète qui ne sert qu'une fois. Cette clé est un nombre aléatoire généré à partir des mouvements aléatoires de votre souris et des touches du clavier sur lesquelles vous tapez. Cette clé de session fonctionne avec un algorithme de chiffrement conventionnel très sûr et rapide qui chiffre le texte clair ; le résultat est le texte chiffré. Une fois que les données sont chiffrées, la clé de session est elle-même chiffrée avec la clé publique du destinataire. Cette clé de session chiffrée par la clé publique est transmise avec le texte chiffré au destinataire.

b. Déchiffrement.

Le déchiffrement fonctionne de la manière inverse. La copie de PGP du destinataire utilise la clé privée de celui-ci pour retrouver la clé de session temporaire, que PGP utilise ensuite pour déchiffrer le texte chiffré de manière conventionnelle.

La combinaison des deux méthodes de chiffrement (IDEA, RSA) associe la commodité du chiffrement à clé publique avec la vitesse du chiffrement conventionnel. Le chiffrement conventionnel est environ 1000 fois plus rapide que le chiffrement à clé publique. Le chiffrement à clé publique fournit quant à lui une solution aux problèmes de distribution de la clé et de transmission des données. Utilisées toutes les deux, la performance et la distribution de la clé sont améliorées sans aucun sacrifice sur la sécurité.

.9.2.4 Le cryptosystème de RABIN

La cryptanalyse totale d'une instance du cryptosystème RSA revient à factoriser N . Par contre, personne n'a pu prouver que cette factorisation est nécessaire pour une cryptanalyse partielle (déchiffrer un message). La sécurité du système suivant, contre une attaque à message clair

choisie, est en revanche équivalente à la factorisation de son module. En gros, ce système consiste à remplacer l'exposant du système de chiffrement de RSA par 2.

Toutefois, puisque $\text{pgcd}(2, (p-1)(q-1))$ est toujours strictement supérieur à 1, l'élévation au carrée modulo N n'est plus une fonction injective.

Le destinataire Bob choisit un entier N qui soit le produit de deux grand nombres entiers distincts, congrus à 3 modulo 4. Il choisit aussi un entier b dans l'intervalle [0, N-1]. Il diffuse l'entier N et b, tout en gardant secret n et p. Il lui sera utile de calculer les coefficients s de Bézout u, v telles que

$$up + vq = 1$$

Pour chiffrer l'élément $m \in \mathbb{Z}_N$, Alice calcul $c = m(m + b) \bmod N$. Lorsque Bob reçoit c, il

déchiffre en calculant les quatre racines carrées modulo N de $c + b^2/4$. Pour cela, il peut calculer

d'abord les racines r_p et r_q modulo p et q, ce qui est particulièrement facile puisque :

$$p \equiv q \equiv 3 \bmod 4.$$

$$r_p = c^{(p+1)/4} \bmod p, \quad r_q = c^{(p+1)/4} \bmod q$$

Puis, il utilise le théorème chinois pour obtenir les quatre racines modulo N :

$$r = \pm ur_q \pm vqr_p \bmod N$$

Le message initial est l'une des quatre valeurs $r - b/2$. En effet on a

$$\left(r - \frac{b}{2}\right)\left(r - \frac{b}{2} + b\right) \equiv \left(r - \frac{b}{2}\right)\left(r + \frac{b}{2}\right) \equiv r^2 - \frac{b^2}{4} \equiv \left(c + \frac{b^2}{4}\right) - \frac{b^2}{4} \equiv c \pmod{N}$$

Décrypter suivant le schéma de Rabin revient à calculer une racine carrée de $c - \frac{b^2}{4} \pmod{N}$

Quelqu'un qui saurait comment décrypter saurait donc calculer des racines carrées modulo N , donc pourrait facilement factoriser N

Un inconvénient évident du schéma de Rabin est que le déchiffrement est ambigu. Précisément

pour $m \in \mathbb{Z}_N$, les quatre messages

$$m, \quad -m - b, \quad \omega m + \frac{(\omega - 1)b}{2}, \quad -\omega m - (\omega + 1)b/2$$

donnent le même message chiffré, ω étant une racine carrée non triviale de 1 modulo N . Au déchiffrement, il faut pouvoir lever l'ambiguïté sur les quatre messages clairs possibles. Cela peut se faire en pratique en ajoutant de la redondance aux messages clairs (par exemple redoubler les 64 premiers bits). Avec une très grande probabilité, une seule des quatre solutions possible au déchiffrement respectera cette redondance.

Si le schéma de Rabin est sûr (si on admet que le problème de la factorisation est difficile) vis-à-vis d'une attaque à message clairs choisis, il est par contre vulnérable contre une attaque à message chiffrés choisis. En effet, si un attaquant choisit un message m au hasard, calcule $c = m(m + b)$ et

obtient un déchiffrement m' de c , alors il y a une chance sur deux pour que m' soit différent de m et de $-m - b$. Il peut dans ce cas calculer ω et factoriser N . L'ajout de redondance rend cette attaque inutilisable. Toutefois, on ne sait pas rigoureusement prouver que la sécurité du schéma de Rabin avec redondance est assujettie à la factorisation.

.9.2.5 Cryptosystème basé sur les codes correcteurs

- Le cryptosystème de Mc Eliece

On choisit un code correcteur sur un corps K pour lequel on connaît un algorithme efficace de décodage (par exemple un code de Goppa). Ce code C est donné par une matrice génératrice G de taille $k \times n$, où n et k sont la longueur et la dimension de C .

On choisit aussi une automorphisme de K^k , donnée par une matrice inversible U , et une isométrie

(pour la distance de Hamming) de K^n , donnée par une matrice de permutation P , qui vont servir à

masquer la nature du code initial C qui lui, restera secret. La clé publique sera constituée de la

matrice $\hat{G} = UGP$, et la clé privée des trois matrices U, G, P .

L'espace des messages clairs est K^k , est chaque message m sera chiffré en l'encodant avec la

matrice \hat{G} et en ajoutant une erreur aléatoire e de poids t , la capacité correctrice de C :

$$c = m\hat{G} = e \text{ avec } \omega(e) = t$$

Le message chiffré c obtenu est un élément de K^n . Pour le déchiffrer, le destinataire lui applique la

permutation définie par P^{-1}

$$cP^{-1} = mUG + eP^{-1}$$

et décode le mot obtenu. Il obtient mU puisque eP^{-1} est de poids t . Il ne lui reste plus qu'à

appliquer la transformation définie par U^{-1} pour retrouver m .

Il est important que l'expéditeur du message respecte le protocole de chiffrement, en ajoutant un mot aléatoire e de poids après encodage. S'il ne le fait pas, son message clair m peut être retrouvé à partir d message chiffré, sans toutefois que la clé privée soit compromise. Il suffit pour cela qu'un attaquant sélectionne k colonnes de la matrice \hat{G} formant une matrice carrée \hat{G}_k inversible.

Le seul message clair m' tel que l'encodé correspondant $m\hat{G}_k$ coïncide avec c sur les k positions sélectionnées est m et il est facile de le retrouver en résolvant $m\hat{G}_k = c_k$, où c_k désigne les k symboles de c correspondant aux colonnes sélectionnées. Si au contraire, l'expéditeur ajoute un mot aléatoire e après chaque encodage, la méthode de décryptage ci-dessus ne fonctionne que si les k positions sélectionnées sont en dehors du support de e . Ceci est hautement improbable lorsque les paramètres sont choisis convenablement.

Les paramètres suggérés à l'origine par Mc Eliece étaient $n=1024$, $t=50$ et $k \geq 524$. Lorsqu'on base ce cryptosystème sur un code de Goppa, un choix optimal pour la sécurité est $n=1024$, $t=38$ et $k \geq 644$. Un inconvénient de ce cryptosystème est la très grande taille de la clé publique (environ 2^{19} bits dans le cas des paramètres précédents).

Un autre inconvénient est un taux de transmission k/n nettement inférieur à 1. Ces inconvénients ont empêché ce cryptosystème d'être utilisé en pratique. D'autre part l'étude profonde de sa sécurité n'a encore bénéficié que de peu d'effort comparativement aux cryptosystèmes usuels.

- Cryptosystème de Niederreiter

Sa sécurité est équivalente à celle du précédent. Toutefois, avec de paramètres judicieux, la taille des clés est un peu raisonnable que dans le cas de Mc Eliece [19]. Le code C peut être identique à celui utilisé pour le cryptosystème de Mc Eliece mais il est ici donné par une matrice de contrôle H (de taille $(n - k) \times n$). On masque la matrice H en utilisant une matrice inversible V de taille

$(n - k) \times (n - k)$ et une matrice de permutation P de taille $n \times n$. La matrice $\hat{H} = VHP$ obtenue est une matrice de contrôle du code \hat{C} . Elle est rendue publique et H est gardée secrète.

L'espace des messages clairs est cette fois ci l'ensemble des mots de longueurs n et de poids t (donc des erreurs). Le chiffré correspondant au message m est son syndrome relativement à la matrice publique :

$$c = \hat{H}m^T \text{ avec } \omega(m) = t$$

Pour déchiffrer, on remarque que $V^{-1}c$ est le syndrome de Pm^T relativement à la matrice secrète :

$$V^{-1}c = HPm^T$$

L'algorithme de décodage dans C (il faut que C possède un algorithme de décodage par symbole efficace) permet de retrouver l'« erreur » Pm^T de poids t . Il ne reste plus qu'à appliquer P^{-1} pour retrouver m .

.9.2.6 Cryptosystème utilisant les courbes elliptiques

Nous verrons que la résolution du logarithme discret dans les corps finis est accessible sur des corps de taille moyenne grâce à des logarithmes relativement récents (algorithme sous exponentiels). L'existence de ces algorithmes oblige à des modules assez grands dans les cryptosystèmes RSA et El Gamal pour préserver leur sécurité. Mais les problèmes du logarithme discret et de Diffie-Hellman ont des homologues naturels en termes de courbes elliptiques.

Par exemple : Soient E une courbe elliptique sur un corps fini et G un point d'ordre h (grand) sur E .
Le problème du logarithme discret sur E est celui de trouver, étant donné un point A du sous groupe de E engendré par G , l'entier a vérifiant :

$$aG = A \quad \text{avec } 0 \leq a \leq h-1$$

Or, on ne dispose pas actuellement de méthode sous exponentielle pour résoudre le logarithme discret sur les courbes elliptiques. Le problème du logarithme discret sur les courbes elliptiques est donc (provisoirement) plus difficile, à taille égale, que le problème du logarithme discret classique. Il est donc tentant de transcrire les cryptosystèmes basés sur le logarithme discret en termes de courbe elliptiques, pour obtenir des cryptosystèmes de sécurité équivalente avec des clés plus petites.

Une transcription directe du cryptosystème d'El Gamal en termes de courbe elliptiques serait imaginable mais pose quelques problèmes techniques (grande taille du message chiffré par rapport à celle du message clair, génération de point de E en fonction du message à transmettre, ...). On lui préfère donc la méthode suivante

- Cryptosystème de Menezes/Vanstone

Soit E une courbe elliptique sur un corps fini et G un point d'ordre h (grand) sur E, rendu publics.

Le destinataire Bob choisit un entier $b \in [0, h-1]$ et diffuse la valeur du point $B=bG$. L'expéditeur

Alice peut alors chiffrer un message $M = (x_M, y_M) \in \mathbb{F}_q * \mathbb{F}_q$ de la manière suivante (noter que le point M n'est pas nécessairement sur E).

Elle choisit secrètement un entier $k \in [0, h-1]$, calcul $K = kG$ et $kB = (x_{kB}, y_{kB})$. Elle transmet à

Bob les données (K, C) où $C = (x_c, y_c) = (x_M x_{kB}, y_M y_{kB})$.

A la réception, Bob peut retrouver M en calculant $bK = (x_{bK}, y_{bK})$ et $(x_M, y_M) = (x_C x_{bK}^{-1}, y_C y_{bK}^{-1})$.



.10 La cryptographie quantique





« Cryptographie quantique » est une expression médiatique, mais quelque peu trompeuse : en effet, il ne s'agit pas de chiffrer un message à l'aide de la physique quantique, mais d'utiliser celle-ci pour s'assurer que la transmission de la clé n'a pas été espionnée. La transmission d'un message, chiffré ou non, peut se faire en utilisant les deux états de polarisation linéaire orthogonaux d'un photon, par exemple $|x\rangle$ et $|y\rangle$. On peut décider d'attribuer par convention la valeur 1 à la polarisation $|x\rangle$ et la valeur 0 à la polarisation $|y\rangle$: chaque photon transporte donc un bit d'information.

Tout message, chiffré ou non, peut être écrit en langage binaire, comme une suite de 0 et de 1, et le message 1001110 sera codé par Alice grâce à la séquence de photons $xyyxxxy$, qu'elle expédiera à Bob par exemple par une fibre optique. A l'aide d'une lame biréfringente, Bob sépare les photons de polarisation verticale et horizontale, et deux détecteurs placés derrière la lame lui permettent de décider si le photon était polarisé horizontalement ou verticalement : il peut donc reconstituer le message.

S'il s'agissait d'un message ordinaire, il y aurait bien sûr des façons bien plus simples et efficaces de le transmettre ! Remarquons simplement que si Eve s'installe sur la fibre, détecte les photons et renvoie à Bob des photons de polarisation identique à ceux expédiés par Alice, Bob ne peut pas savoir que la ligne a été espionnée. Il en serait de même pour tout dispositif fonctionnant de façon classique (c'est-à-dire sans utiliser le principe de superposition) : si l'espion prend suffisamment de précautions, il est indétectable.

C'est ici que la mécanique quantique et le principe de superposition viennent au secours d'Alice et de Bob, en leur permettant de s'assurer que leur message n'a pas été intercepté. Ce message n'a pas besoin d'être long (le système de transmission par la polarisation est très peu performant). Il s'agira en général de transmettre une clé permettant de chiffrer un message ultérieur, clé qui pourra être remplacée à la demande. Alice envoie vers Bob quatre types de photons : polarisés suivant Ox : \uparrow et Oy : \rightarrow comme précédemment, et polarisés suivant des axes inclinés à $\pm 45^\circ$ Ox' :

 et Oy' : , correspondant respectivement aux valeurs 1 et 0 des bits. De même Bob analyse les photons envoyés par Alice à l'aide d'analyseurs pouvant prendre quatre directions, verticale/horizontale, et $\pm 45^\circ$.

Une possibilité serait d'utiliser un cristal biréfringent orienté aléatoirement soit verticalement, soit à 45° de la verticale et de détecter les photons sortant de ce cristal. Cependant, au lieu de faire tourner l'ensemble cristal+détecteurs, on utilise plutôt une cellule de Pockels, qui permet de transformer une polarisation donnée en une polarisation orientée de façon arbitraire et de maintenir fixe l'ensemble cristal+détecteur. La figure 1.5 donne un exemple : Bob enregistre 1 si le photon est polarisé  ou , 0 s'il est polarisé  ou . Après enregistrement d'un nombre suffisant de photons, Bob annonce publiquement la suite des analyseurs qu'il a utilisés, mais non ses résultats. Alice compare sa séquence de polariseurs à celle de Bob et lui donne toujours publiquement la liste des polariseurs compatibles avec ses analyseurs.

Les bits qui correspondent à des analyseurs et des polariseurs incompatibles sont rejetés (-) et, pour les bits restants, Alice et Bob sont certains que leurs valeurs sont les mêmes : ce sont les bits qui serviront à composer la clé, et ils sont connus seulement de Bob et Alice, car l'extérieur ne connaît que la liste des orientations, pas les résultats ! Le protocole décrit ci-dessus est appelé BB84, du nom de ses inventeurs Bennett et Brassard.

polariseurs d'Alice									
séquences de bits	1	0	0	1	0	0	1	1	1
analyseurs de Bob									
mesures de Bob	1	1	0	1	0	0	1	1	1
bits retenus	1	–	–	1	0	0	–	1	1

Tableau 2.9 : *Transmission de photons polarisés entre Bob et Alice.*

Il reste à s'assurer que le message n'a pas été intercepté et que la clé qu'il contenait peut être utilisée sans risque. Alice et Bob choisissent au hasard un sous-ensemble de leur clé et le comparent publiquement. La conséquence de l'interception de photons par Eve serait une réduction de la corrélation entre les valeurs de leurs bits : supposons par exemple qu'Alice envoie un photon polarisé suivant Ox. Si Eve l'intercepte avec un polariseur orienté suivant Ox', et que le photon est transmis par son analyseur, elle ne sait pas que ce photon était initialement polarisé suivant Ox ; elle renvoie donc à Bob un photon polarisé dans la direction Ox', et dans 50% des cas Bob ne va pas trouver le bon résultat. Comme Eve a une chance sur deux d'orienter son analyseur dans la bonne direction, Alice et Bob vont enregistrer une différence dans 25% des cas et en conclure que le message a été intercepté. Cette discussion est bien sûr simplifiée : elle ne tient pas compte des possibilités d'erreurs qu'il faut corriger, et d'autre part il faut réaliser des impulsions à un seul photon et non des paquets d'états cohérents qui ne seraient pas inviolables. Néanmoins la méthode est correcte dans son principe et un prototype a été réalisé récemment pour des transmissions dans l'air sur plusieurs kilomètres. Il est difficile avec une fibre optique de contrôler la direction de la polarisation sur de longues distances, et c'est pourquoi on utilise un support physique différent pour mettre en œuvre le protocole BB84 avec des fibres. Dans ces conditions la transmission a pu être effectuée sur une centaine de kilomètres.

CHAPITRE 3

CRYPTANALYSE ET THEORIE DE LA COMPLEXITE

.11 La cryptanalyse

Si le but de la cryptographie est d'élaborer des méthodes de protection, le but de la cryptanalyse est au contraire de casser ces protections. Une tentative de cryptanalyse d'un système est appelée une attaque, et elle peut conduire à différents résultats :

- Cassage complet : le cryptanalyste retrouve la clé de chiffrement
- Obtention globale : le cryptanalyste retrouve un algorithme de déchiffrement, mais qui ne nécessite pas de connaissance de la clé de chiffrement
- Obtention locale : le cryptanalyste retrouve le texte en clair correspondant à un message chiffré
- Obtention d'information : le cryptanalyste obtient quelques indications sur le texte en clair ou la clé (certains bits de la clé, un renseignement sur la forme du texte...).

.12 Attaques des fonctions de chiffrement

Les fonctions de chiffrement sont supposées rendre impossible le décryptement, c'est-à-dire la récupération d'un message clair sans clé. A fortiori, elles doivent protéger le secret des clés.

.12.1 Classement des attaques en fonction des données dont dispose le cryptanalyste

On distingue plusieurs types d'attaques suivant les informations que peut obtenir le cryptanalyste.

Ce sont :

- *L'attaque à texte chiffré* seulement, où le cryptanalyste connaît qu'un ensemble de textes chiffrés ; il peut soit retrouver seulement les textes en clair, soit retrouver la clé. En pratique, il est très souvent possible de deviner certaines propriétés du texte en clair (format ASCII, présence d'un mot particulier,...), ce qui permet de valider ou non le décryptement.
- *L'attaque à texte en clair connu*, où le cryptanalyste connaît non seulement les textes chiffrés, mais aussi les textes en clair correspondants ; son but est alors de retrouver la clé. Du fait de la présence, dans la plupart des textes chiffrés, de parties connues (en-têtes de paquets, champs communs à tous les fichiers d'un type donné,...), ce type d'attaques est très courant.
- *L'attaque à texte en clair choisi*, où le cryptanalyste peut, de plus, choisir des textes en clair à chiffrer et donc utiliser des textes apportant plus d'informations sur la clé. Si le cryptanalyste peut de plus adapter ses choix en fonction des textes chiffrés précédents, on parle d'attaque adaptative.
- *L'attaque à texte chiffré choisi*, qui est l'inverse de la précédente : le cryptanalyste peut choisir des textes chiffrés pour lesquels il connaîtra le texte en clair correspondant ; sa tâche est alors de retrouver la clé. Ce type d'attaques est principalement utilisé contre les systèmes à clé publique, pour retrouver la clé privée.

.12.2 Attaques sur les algorithmes symétriques

- Attaques au niveau des clés

L'attaque la plus simple sur le plan conceptuel est l'attaque exhaustive ou attaque en force brute, qui consiste à essayer toutes les clés possibles. Avec une clé de 56 bits par exemple, cela nécessite 2^{56} tests.

Si la clé recherchée est en fait un mot de passe ou si elle dérive d'un mot de passe, on a des chances de trouver la clé en testant des mots susceptibles d'avoir été choisis comme mot de passe. On parle alors d'attaque par dictionnaire, car le cryptanalyste se constitue un dictionnaire de mots à tester (ce peut être une liste de prénoms, l'ensemble des mots d'une langue donnée,...). Ce type d'attaques est bien sûr beaucoup plus rapide qu'une attaque exhaustive.

- Cryptanalyse différentielle

La cryptanalyse différentielle est un type d'attaques qui peut-être utilisé contre les algorithmes de chiffrement par blocs itératifs. Elle utilise une attaque à texte en clair choisi et se base sur l'observation de l'évolution des différences entre deux textes lorsqu'ils sont chiffrés avec la même clé. En analysant ces différences entre paires de textes, il est possible d'attribuer des probabilités à chaque clé possible, à force d'analyser des paires de textes, on finit soit par trouver la clé recherchée, soit par réduire suffisamment le nombre de clés possibles pour pouvoir mener une attaque exhaustive rapide.

La cryptanalyse différentielle peut également être utilisée pour attaquer d'autres types d'algorithmes comme les fonctions de hachage.

Ce type d'attaques a été utilisé pour la première fois par Murphy en 1990 contre l'algorithme FEAL-4. Biham et Shamir ont mené des attaques différentielles sur le DES, mais leur efficacité reste limitée du fait de la conception des tables-S, qui avaient été optimisées contre ce type d'attaques. La meilleure attaque différentielle contre le DES complet à 16 rondes nécessite 2^{47} textes en clair choisis. Cela reste énorme et demande beaucoup trop de chiffrements, si bien qu'il est couramment admis que le DES est encore sûr par rapport à la cryptanalyse différentielle.

- Cryptanalyse linéaire

La **cryptanalyse linéaire** utilise une attaque à texte en clair connu et consiste à modéliser l'algorithme de chiffrement par blocs par une approximation linéaire. Avec un nombre suffisant de paires (texte en clair, texte chiffré), on peut deviner certains bits de la clé.

La cryptanalyse linéaire fut utilisée pour la première fois en 1992 par Matsui et Yamagishi pour attaquer FEAL.

Elle fut étendue par Matsui en 1993 pour attaquer DES. Les tables-S du DES ne sont pas optimisées pour contrer la cryptanalyse linéaire, et la meilleure attaque différentielle actuelle contre le DES nécessite 2 textes en clair connus en moyenne.

Une réalisation logicielle de cette attaque a découvert une clé DES en 50 jours avec 12 stations de travail. En 1994, Langford et Hellman introduisirent une attaque appelée cryptanalyse différentielle linéaire qui combine des éléments des deux méthodes précédentes.

.12.3 Attaques sur les algorithmes asymétriques

- Attaques au niveau des clés

Avec les algorithmes asymétriques, le problème n'est pas de trouver la bonne clé par attaque

exhaustive, mais de **dériver la clé secrète à partir de la clé publique**.

La plupart des algorithmes asymétriques reposant sur des problèmes mathématiques difficiles à résoudre, **cela revient généralement à résoudre ce problème**.

C'est pourquoi les paramètres qui influencent la difficulté de résolution du problème sont les éléments déterminant la sécurité. Généralement, cela se traduit par la nécessité d'utiliser de grands nombres, la taille de ces nombres ayant une répercussion sur la longueur des clés.

Cela explique que les clés utilisées par la cryptographie à clé publique soient généralement bien plus longues que celles utilisées par la cryptographie à clé secrète.

Par exemple, dans le cas de RSA, l'élément déterminant est la taille du module. La factorisation d'un module de 512 bits est à la portée d'une agence gouvernementale, 1024 bits est considéré comme sûr actuellement, et 2048 bits garantit une sécurité à long terme.

- Attaque à texte en clair deviné et problème de la faible entropie

Un point faible des algorithmes à clé publique est le caractère public de la clé de chiffrement : le cryptanalyste ayant connaissance de cette clé, il peut mener une **attaque à texte en clair deviné**, qui consiste à tenter de deviner le texte en clair et à le chiffrer pour vérifier son exactitude. Cette particularité implique donc une **restriction sur l'utilisation des algorithmes asymétriques** : il faut absolument éviter de les utiliser avec un ensemble de textes en clair possibles restreint (i.e. de faible entropie). En effet, si tel était le cas, un attaquant pourrait aisément se constituer une liste exhaustive des textes en clair possibles et des textes chiffrés correspondants. Il n'aurait alors aucun mal à retrouver le texte en clair correspondant à un cryptogramme donné.

- Attaque à texte chiffré choisi

D'autre part, la plupart des algorithmes asymétriques sont sensibles aux attaques à texte chiffré choisi. Il convient donc, si l'on utilise le même algorithme pour le chiffrement et pour la signature, de s'assurer que les protocoles employés ne permettent pas à un adversaire de faire signer n'importe quel texte. Mieux, on peut avoir recours à des couples (clé publique, clé privée) distincts pour ces deux opérations.

- Attaque temporelle

Un type d'attaque apparu récemment est l'**attaque temporelle**, qui utilise la mesure du temps pris par des opérations cryptographiques pour retrouver les clés privées utilisées. Cette attaque a été réalisée avec succès contre des cartes à microcircuits, des calculettes de sécurité et contre des serveurs de commerce électronique à travers l'Internet.

La société CounterpaneSystems, entre autres, a généralisé ces méthodes pour y inclure des **attaques sur des systèmes en mesurant la consommation, les émissions radioélectriques, et autres "canaux latéraux"** ; ils les ont mises en œuvre contre plusieurs types d'algorithmes à clé publique ou à clé secrète utilisés dans des calculettes. Une recherche voisine s'est intéressée à l'introduction délibérée d'erreurs dans les processeurs cryptographiques pour tenter d'obtenir des informations sur la clé.

.12.4 Attaques des générateurs pseudo-aléatoires

Les générateurs pseudo-aléatoires sont supposés être imprévisibles. Un modèle d'attaque courant consiste à observer les aléas engendrés pendant un certain temps pour deviner la suite, voire l'état du générateur.

.12.5 Attaques des fonctions de hachage à sens unique

Les deux attaques exhaustives de base que l'on peut mener contre une fonction de hachage à sens unique consistent à tester des entrées aléatoires avec la fonction jusqu'à :

- Trouver une entrée qui donne en sortie une empreinte donnée (contredisant de ce fait la propriété "à sens unique") ;
- Trouver deux entrées qui produisent la même sortie (contredisant de ce fait la propriété "sans collision").

Supposons que la fonction de hachage génère une empreinte de n bits. Si l'on veut trouver une entrée qui produira une valeur de sortie h donnée, alors, chaque sortie étant équiprobable, il faudra essayer 2^n valeurs possibles d'entrée pour avoir une bonne chance de la trouver.

.12.6 Attaque des anniversaires (paradoxe des anniversaires)

Le paradoxe des anniversaires est un résultat célèbre du problème de probabilité suivant : combien faut-il de personnes dans une pièce pour qu'il y ait plus d'une chance sur deux pour que deux personnes au moins soient nées le même jour de l'année.

La réponse est surprenante : 23 personnes suffisent ! D'une façon plus générale, considérons une fonction qui, lorsqu'on lui fournit une entrée aléatoire, retourne, de façon équiprobable une valeur parmi k valeurs possibles (k grand).

Si l'on applique cette fonction à différentes valeurs d'entrée, on a plus d'une chance sur deux de retomber sur une sortie déjà obtenue après avoir effectué un nombre d'essais de l'ordre de \sqrt{k}

Si l'on applique ce résultat à une fonction de hachage générant une empreinte de n bits, il y a $k = 2^n$

valeurs possibles en sortie, donc il faudra essayer seulement de l'ordre de $2^{n/2}$ valeurs d'entrées.

Au lieu d'être en $O(2^n)$, la complexité de l'attaque n'est plus qu'en $O(2^{n/2})$.

L'attaque des anniversaires est surtout intéressante avec les algorithmes de signature basés sur une fonction de hachage : elle permet de faire signer à un tiers un message correct, mais auquel correspond un message frauduleux ayant la même empreinte. Yuval a proposé la stratégie suivante

- L'adversaire choisit un message frauduleux qu'il désire faire signer, et un message inoffensif qu'Alice acceptera sûrement de signer.
- L'adversaire génère $2^{n/2}$ variantes du message inoffensif (en faisant, par exemple, des modifications rédactionnelles mineures) et les empreintes correspondantes. Il génère ensuite un nombre égal de variantes du message frauduleux.
- La probabilité qu'une des variantes du message inoffensif et une des variantes du message frauduleux donnent la même empreinte est supérieure à $1/2$ selon le paradoxe des anniversaires.
- L'adversaire fait alors signer la variante du message inoffensif à Alice.
- La signature du message inoffensif est alors retirée et attachée à la variante du message frauduleux qui donne la même empreinte. L'adversaire a donc forgé un faux message signé sans avoir besoin de connaître les clés de chiffrement.

Pour éviter de telles attaques, il convient de choisir une longueur d'empreinte assez élevée. Les empreintes de 128 bits générées par MD5 sont désormais trop justes ; les 160 bits de SHA-1 et RIPE-MD-160 sont plus sûrs.

.12.7 Attaques des protocoles cryptographiques

On distingue deux types d'attaques sur les protocoles : les **attaques passives** et les **attaques actives**.

Dans le premier cas, l'attaquant ne peut qu'espionner les données échangées par les tiers communicants, alors que dans le second cas il peut modifier ou supprimer des messages, en ajouter des nouveaux ou des anciens qu'il rejoue.

- Attaque par mascarade

On parle d'attaque par mascarade (*spoofing attack*) lorsqu'un attaquant **essaye de se faire passer pour un utilisateur légitime** en exécutant un **protocole** à la place de celui-ci.

La conception d'un **protocole** a bien sûr pour but principal de contrecarrer ce type d'attaques.

- Attaque par rejeu

Une attaque par rejeu (*replay attack*) consiste à envoyer, dans une communication, des messages interceptés au cours d'une autre communication ou plus tôt dans la communication.

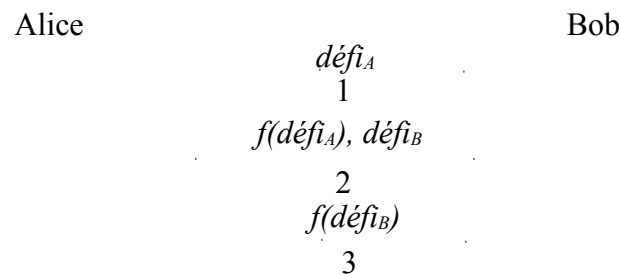
Ce type d'attaques permet de contourner des protocoles simples comme, par exemple, une authentification par mot de passe : il suffit à un adversaire d'avoir espionné un échange pour connaître le mot de passe et donc pour pouvoir se faire passer pour un utilisateur légitime.

Les méthodes pour se prémunir contre ce type d'attaques sont l'utilisation de marquages temporels (*timestamps*) ou de défis imprévisibles et à usage unique.

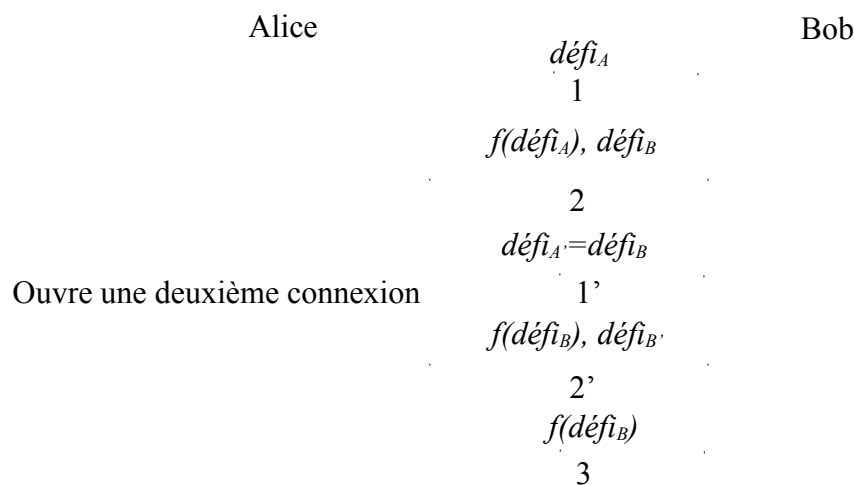
- Attaque par réflexion

Une attaque par réflexion (*reflection attack*) est une attaque pour laquelle **l'adversaire exploite le caractère symétrique d'un protocole** pour répondre aux défis de son interlocuteur en utilisant des réponses fournies par l'interlocuteur lui-même.

Considérons par exemple le **protocole** d'authentification mutuelle suivant :



Pour se faire passer pour Alice auprès de Bob, il suffit à un adversaire d'établir deux connexions en parallèle avec Bob, et d'envoyer $\text{défi}_{A'} = \text{défi}_B$ comme second défi à Bob. Celui-ci fournira alors la réponse attendue à son premier défi :



- Attaque de l'intercepteur

Le principe de l'attaque de l'intercepteur (*man-in-the-middle attack*) est que, pendant que deux tiers procèdent à un échange de clé, en utilisant un algorithme du type Diffie-Hellman par exemple, **un adversaire se positionne entre les deux tiers et intercepte les échanges.**

De cette façon, il procède à un échange de clé avec chaque tiers, à la fin du **protocole**, chaque tiers utilisera donc une clé différente, chacune de ces clés étant connue de l'intercepteur.

Pour chaque message transmis par la suite, l'intercepteur procédera à son déchiffrement avec la clé correspondante puis le rechiffra avec l'autre clé avant de l'envoyer à son destinataire : les deux tiers croiront communiquer de façon sûre alors que l'intercepteur pourra en fait lire tous les messages, voire même forger de faux messages.

Voici comment se déroule cette attaque dans le cas de Diffie-Hellman :

- 1) Alice envoie sa valeur publique $A = g^a \bmod n$ à Bob ; Ingrid l'intercepteur remplace cette valeur publique par la sienne. Bob reçoit donc $I = g^i \bmod n$.
- 2) Bob envoie sa valeur publique à Alice ; Ingrid remplace là aussi cette valeur par la sienne.
- 3) Alice génère le "secret" $K_{AI} = I^a \bmod n$. Ingrid génère le même secret en calculant $A^i \bmod n$.
- 4) Bob génère le "secret" $K_{BI} = I^b \bmod n$. Ingrid génère le même secret en calculant $B^i \bmod n$.

Alice et Bob croient tous les deux être en possession d'un secret partagé, mais en fait chacun d'eux partage un secret différent avec Eve.

.13 Théorie de la complexité

La théorie de la complexité s'attache à connaître la difficulté (ou la complexité) d'une réponse par algorithme à un problème, dit algorithmique, posé de façon mathématique. Pour pouvoir la définir, il faut présenter ces trois concepts que sont les problèmes algorithmiques, les réponses algorithmiques aux problèmes, la complexité des problèmes algorithmiques.

.13.1 *Problème algorithmique*

Un problème algorithmique est un problème posé de façon mathématique, c'est-à-dire qu'il est énoncé rigoureusement dans le langage des mathématiques – le mieux étant d'utiliser le calcul des prédicats. Il comprend des *hypothèses*, des *données* et une *question*. On distingue :

- les **problèmes de décision** : ils posent une question dont la réponse est *oui* ou *non* ;
- les **problèmes d'existence ou de recherche d'une solution** : ils comportent une question ou plutôt une injonction de la forme « *trouver un élément tel que ...* » dont la réponse consiste à fournir un tel élément.

.13.2 *Réponse algorithmique*

Dans chaque catégorie de problèmes ci-dessus, on dit qu'un problème a une réponse algorithmique si sa réponse peut être fournie par un algorithme. Un problème est *décidable* s'il s'agit d'un problème de décision – donc d'un problème dont la réponse est soit *oui* soit *non* – et si sa réponse peut être fournie par un algorithme. Symétriquement, un problème est *calculable* s'il s'agit d'un problème d'existence et si l'élément calculé peut être fourni par un algorithme. La théorie de la complexité ne couvre que les problèmes décidables ou calculables et cherche à évaluer les ressources – temps et espace mémoire – mobilisées pour obtenir algorithmiquement la

réponse.

.13.3 Complexité d'un problème algorithmique

La théorie de la complexité vise à savoir si la réponse à un problème peut être donnée très efficacement, efficacement ou au contraire être inatteignable en pratique (et en théorie), avec des niveaux intermédiaires de difficulté entre les deux extrêmes ; pour cela, elle se fonde sur une estimation – théorique – des temps de calcul et des besoins en mémoire informatique. Dans le but de mieux comprendre comment les problèmes se placent les uns par rapport aux autres, la théorie de la complexité établit des hiérarchies de difficultés entre les problèmes algorithmiques, dont les niveaux sont appelés des « *classes de complexité* ». Ces hiérarchies comportent des ramifications, suivant que l'on considère des calculs déterministes – l'état suivant du calcul est « déterminé » par l'état courant – ou non déterministes.

.13.4 Modèle de calcul

L'analyse de la complexité est étroitement associée à un **modèle de calcul**. L'un des modèles de calcul les plus utilisés est celui des machines abstraites dans la lignée du modèle proposé par Alan Turing en 1936.

Les deux modèles les plus utilisés en théorie de la complexité sont :

- La machine de Turing ;
- La machine RAM (Random Access Machine).

Dans ces deux modèles de calcul, un calcul est constitué d'étapes élémentaires ; à chacune de ces étapes, pour un état donné de la mémoire de la machine, une action élémentaire est choisie dans un ensemble d'actions possibles. Les *machines déterministes* sont telles que chaque action possible est unique, c'est-à-dire que l'action à effectuer est dictée de façon unique par l'état courant de celle-ci.

S'il peut y avoir plusieurs choix possibles d'actions à effectuer, la *machine* est dite *non déterministe*. Il peut sembler naturel de penser que les machines de Turing non déterministes sont plus puissantes que les machines de Turing déterministes, autrement dit qu'elles peuvent résoudre en un temps donné des problèmes que les machines déterministes ne savent pas résoudre dans le même temps.

.13.5 Complexité en temps et en espace

Sans nuire à la généralité on peut supposer que les problèmes que nous considérons n'ont qu'une donnée. Cette donnée a une taille qui est un nombre entier naturel. La façon dont cette taille est mesurée joue un rôle crucial dans l'évaluation de la complexité de l'algorithme.

Ainsi si la donnée est elle-même un nombre entier naturel, sa taille peut être appréciée de plusieurs façons : on peut dire que la taille de l'entier p vaut p , mais on peut aussi dire qu'elle vaut

$\log(p)$ parce que l'entier a été représenté en numération binaire ou décimale ce qui raccourcit la représentation des nombres, ainsi 1 024 peut être représenté avec seulement onze chiffres binaires et quatre chiffres décimaux et donc sa taille est 11 ou 4 et non pas de l'ordre de 1 000.

Le but de la complexité est de donner une évaluation du temps de calcul ou de l'espace de calcul nécessaire en fonction de cette taille, qui sera notée n . L'évaluation des ressources requises permet de répartir les problèmes dans des classes de complexité.

Pour les machines déterministes, on définit la classe $TIME(t(n))$ des problèmes qui peuvent être résolus en temps $t(n)$. C'est-à-dire pour lesquels il existe au moins un algorithme sur une machine

déterministe résolvant le problème en temps $t(n)$.

Le temps est le nombre de transitions sur machine de Turing ou le nombre d'opérations sur machine RAM, mais en fait ce temps n'est pas une fonction précise, mais c'est un ordre de grandeur, on parle aussi d'évaluation asymptotique, ainsi pour un temps qui s'évalue par un polynôme ce qui compte c'est le degré du polynôme, si ce degré est 2, on dira que l'ordre de grandeur est en $O(n^2)$, que la complexité est *quadratique* et que le problème appartient à la classe

$TIME(n^2)$.

Notation	Type de complexité
$O(1)$	complexité constante (indépendante de la taille de la donnée)
$O(\log(n))$	complexité logarithmique
$O(n)$	complexité linéaire
$O(n \cdot \log(n))$	complexité quasi-linéaire
$O(n^2)$	complexité quadratique
$O(n^3)$	complexité cubique
$O(n^p)$	complexité polynomiale
$O(n^{\log(n)})$	complexité quasi-polynomiale
$O(2^n)$	complexité exponentielle
$O(n!)$	complexité factorielle

.13.6 Les quatre familles de classes de complexité en temps et en espace

Suivant qu'il s'agit de temps et d'espace, de machines déterministes ou non déterministes, on

distingue quatre classes de complexité.

$TIME(t(n))$ est la classe des problèmes de décision qui peuvent être résolus en temps de l'ordre de grandeur de $t(n)$ sur une machine déterministe.

$NTIME(t(n))$ est la classe des problèmes de décision qui peuvent être résolus en temps de l'ordre de grandeur de $t(n)$ sur une machine non déterministe.

$SPACE(s(n))$ est la classe des problèmes de décision qui requièrent pour être résolus un espace de l'ordre de grandeur de $s(n)$ sur une machine déterministe.

$NSPACE(s(n))$ est la classe des problèmes de décision qui requièrent pour être résolus un espace de l'ordre de grandeur de $s(n)$ sur une machine non déterministe.

.13.7 Classes de complexité

Dans ce qui suit nous allons définir quelques classes de complexité parmi les plus étudiées en une liste qui va de la complexité la plus basse complexité à la complexité la plus haute. Il faut cependant avoir à l'esprit que ces classes ne sont pas totalement ordonnées.

Commençons par la classe constituée des problèmes les plus simples, à savoir ceux dont la réponse peut être donnée en *temps constant*. Par exemple, la question de savoir si un nombre entier est positif peut être résolue sans vraiment calculer, donc en un temps indépendant de la taille du nombre entier, c'est la plus basse des classes de problèmes.

La classe des *problèmes linéaires* est celle qui contient les problèmes qui peuvent être décidés en un temps qui est une fonction linéaire de la taille de la donnée. Il s'agit des problèmes qui sont en

$O(n)$.

Souvent au lieu de dire « un problème est dans la classe C » on dit plus simplement « le problème

est dans C ».

.13.7.1 Classes L et NL

Un problème de décision qui peut être résolu par un algorithme déterministe en espace *logarithmique* par rapport à la taille de l'instance est dans L .

Avec les notations introduites plus haut, $L = SPACE(log(n))$. La classe NL s'apparente à la classe L mais sur une machine non déterministe ($NL = NSPACE(log(n))$).

Par exemple savoir si un élément appartient à un tableau trié peut se faire en espace logarithmique.

.13.7.2 Classe P

Un problème de décision est dans P s'il peut être décidé sur une machine déterministe en temps *polynomial* par rapport à la taille de la donnée. On qualifie alors le problème de polynomial, c'est un problème de complexité $O(n^k)$ pour un certain k .

Un exemple de problème polynomial est celui de la connexité dans un graphe. Étant donné un graphe à s sommets (on considère que la taille de la donnée, donc du graphe est son nombre de sommets), il s'agit de savoir si toutes les paires de sommets sont reliées par un chemin.

Un algorithme de parcours en profondeur construit un arbre couvrant du graphe à partir d'un sommet. Si cet arbre contient tous les sommets du graphe, alors le graphe est connexe. Le temps nécessaire pour construire cet arbre est en au plus $c \cdot s^2$ (où c est une constante), donc le problème est dans la classe P .

On admet, en général, que les problèmes dans P sont ceux qui sont facilement solubles.

.13.7.3 Classe NP et classe Co-NP (complémentaire de NP)

La classe NP des problèmes **Non-déterministes Polynomiaux** réunit les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial.

De façon équivalente, c'est la classe des problèmes qui admettent un algorithme polynomial capable de tester la validité d'une solution du problème, on dit aussi capable de construire un certificat. Intuitivement, les problèmes dans NP sont les problèmes qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant à l'aide d'un algorithme polynomial.

Par exemple, la recherche de cycle Hamiltonien dans un graphe peut se faire à l'aide de deux algorithmes :

- le premier engendre l'ensemble des cycles (en temps exponentiel, classe *EXPTIME*, voir ci-dessous) ;
- le second teste les solutions (en temps polynomial).

Ce problème est donc de la classe *NP*.

La classe duale de la classe *NP*, quand la réponse est *non*, est appelée *Co-NP*.

.13.7.4 Classe PSPACE

La classe *PSPACE* est celle des problèmes décidables par une machine déterministe en espace

polynomial par rapport à la taille de sa donnée. On peut aussi définir la classe *NSPACE* ou

NPSPACE des problèmes décidables par une machine non déterministe en espace polynomial par

rapport à la taille de sa donnée. Par le théorème de Savitch, on a $PSPACE = NPSPACE$, c'est

pourquoi on ne rencontre guère les notations $NSPACE$ ni $NPSPACE$.

.13.7.5 Classe EXPTIME

La classe *EXPTIME* rassemble les problèmes décidables par un algorithme déterministe en temps exponentiel par rapport à la taille de son instance.

.13.7.6 Classe NC (Nick's Class)

La classe **NC** est la classe des problèmes qui peuvent être résolus en temps **poly-logarithmique** (c'est à dire résolus plus rapidement qu'il ne faut de temps pour lire séquentiellement leurs entrées) sur une **machine parallèle** ayant un nombre **polynomial** (c'est à dire raisonnable) de processeurs.

Intuitivement, un problème est dans **NC** s'il existe un algorithme pour le résoudre qui peut être

parallélisé et qui gagne à l'être. C'est à dire, si la version parallèle de l'algorithme (s'exécutant sur plusieurs processeurs) est significativement plus efficace que la version séquentielle.

Par définition, **NC** est un sous ensemble de la classe P ($NC \subseteq P$) car une machine parallèle peut être simulée par une machine séquentielle.

Mais on ne sait pas si $P \subseteq NC$ (et donc si $NC = P$). On conjecture que non, en supposant qu'il existe dans **P** des problèmes dont les solutions sont intrinsèquement non parallélisables.

.13.8 Problèmes C-complets ou C-difficiles

Soit **C** une classe de complexité (comme **P**, **NP**, etc.). On dit qu'un problème est **C-complet** ou **C-difficile** si ce problème est au moins aussi difficile que tous les problèmes dans **C**. Formellement on définit une notion de réduction : soient Π et Π' deux problèmes ; une réduction de Π' à Π est un

algorithme (ou une machine) d'une complexité qu'on sait être inférieure à celle de la classe **C** transformant toute instance de Π' en une instance de Π . Ainsi, si l'on a un algorithme pour résoudre

Π , on sait aussi résoudre Π' , mais de plus, si la complexité de Π est au moins celle de la classe **C**,

on peut dire que Π est donc au moins aussi difficile à résoudre que Π' .

Π est alors **C-complet** ou **C-difficile** si pour tout problème Π' de **C**, Π' se réduit à Π .

Pour les problèmes **NP-complet** ou **NP-difficile** on s'autorise uniquement des réductions dans **P**, c'est-à-dire que l'algorithme qui calcule le passage d'une instance de Π' à une instance de Π est polynomial. Quand on parle de problèmes **P-complet** ou **P-difficile** on s'autorise uniquement des réductions dans **LOGSPACE**.

On qualifie de **NP-complet** les problèmes décisionnels, c'est-à-dire que la réponse est de type binaire (oui/non, vrai/faux, 1/0,...). On qualifie de **NP-difficile** les problèmes d'optimisation, c'est-à-

dire que la réponse est de type numérique. A un problème d'optimisation **NP-difficile**, est associé un problème de décision **NP-complet**, mais dire qu'un problème **NP-difficile** est aussi **NP-complet** est un abus de langage car il n'y a pas de comparaison possible.

.13.9 Réduction de problèmes et problèmes NP-complets

Pour montrer qu'un problème Π est **C**-difficile pour une classe **C** donnée, il y a deux façons de procéder : ou bien montrer que tout problème de **C** se réduit à Π , ou bien montrer qu'un problème **C**-difficile se réduit à Π . Par exemple, démontrons que le problème de la recherche de circuit hamiltonien dans un graphe orienté est NP-Complet.

Le problème est dans *NP*, car on peut trouver un algorithme pour le résoudre à l'aide d'une machine non déterministe, par exemple en engendrant (de façon non déterministe) un circuit, puis en testant au final s'il est hamiltonien.

On sait que le problème de la recherche d'un cycle hamiltonien dans un graphe non orienté est **NP-difficile**. Or un graphe non orienté peut se transformer en un graphe orienté en créant deux arcs opposés pour chaque arête. Il est donc possible de ramener le problème connu, NP-difficile, à savoir chercher un cycle hamiltonien dans un graphe non orienté, au problème que nous voulons classer, à savoir chercher un circuit hamiltonien dans un graphe orienté. **Le problème de l'existence d'un circuit hamiltonien est donc NP-difficile.**

.13.10 Le problème ouvert ^{1.1.1} *P*

On a clairement $P \subseteq NP$ car un algorithme déterministe est un algorithme non déterministe particulier, ce qui, dit en mots plus simples, signifie que si une solution peut être calculée en temps polynomial, alors elle peut être vérifiée en temps polynomial. En revanche, la réciproque : $P \subseteq NP$

, qui est la véritable difficulté de l'égalité $P = NP$ est un problème ouvert central d'informatique théorique. La plupart des spécialistes conjecturent que les problèmes NP-complets ne sont pas solubles en un temps polynomial. À partir de là, plusieurs approches ont été tentées.

Des algorithmes d'approximation (ou heuristiques) permettent de trouver des solutions approchées

de l'optimum en un temps raisonnable pour un certain nombre de programmes. Dans le cas d'un problème d'optimisation on trouve généralement une réponse correcte, sans savoir s'il s'agit de la meilleure solution.

Des algorithmes stochastiques : en utilisant des nombres aléatoires on peut «forcer» un algorithme à ne pas utiliser les cas les moins favorables, l'utilisateur devant préciser une probabilité maximale admise que l'algorithme fournisse un résultat erroné.

Des algorithmes par séparation et évaluation permettent de trouver la ou les solutions exactes. Le temps de calcul n'est bien sûr pas borné polynomialement mais, pour certaines classes de problèmes, il peut rester modéré pour des instances relativement grandes.

L'approche de la complexité paramétrée consiste à identifier un paramètre qui, dans le cas où il reste petit, permet de résoudre rapidement le problème. En particulier les algorithmes FPT en un paramètre k permettent de résoudre un problème en temps proportionnel au produit d'une fonction quelconque de k et d'un polynôme en la taille de l'instance, ce qui fournit un algorithme polynomial quand k est fixé.

On peut restreindre la classe des problèmes considérés à une sous-classe suffisante, mais plus facile à résoudre.