TABLE DES MATIERES

NOTATIONS

a, b, c, d, e, r, p, n, u, v, q	: sont des entiers
C	: Cryptogramme ou texte chiffré
C_i , c_i	: Cryptogramme élémentaire
D_{k_i}	: Processus de décryptage avec la clé k_i
$D_{k_i}(x)$, $D_k(x)$	: Fonctions inverses paramétrées respectivement par les clés k_i et k
E_{k_i}	: Processus d'encryptage avec une clé k_i
$E_{k_i}(x)$, $E_k(x)$	: Fonctions paramétrées respectivement par les clés k_i et k
f	: Une fonction quelconque
$f_t(X,Y,Z)$	: Fonction à trois variables non linéaire de SHA
h	: Longueur fixe du résultat de $H(x)$
$H(x)$	: Fonction de hachage à sens unique
K, k	: Clé de cryptage
K_i	: Sous-clé
M	: Message clair
M_i, m_i	: Message élémentaire
$S_k(x)$	: Fonction paramétrée par la clé k
$S_k^{-1}(x)$	: Fonction inverse paramétrée la clé k
AES	: Advanced Encryption Standard
AKA	: Authentification and Key Agreement
ANSI	: American National Standardisation Institute
CBC	: Cipher Block Chaining

CCITT	: Comité Consultatif International pour la Téléphonie et le Télégraphe
CFB	: Cipher Feedback
CRC	: Cyclic Redundancy Algorithm
DEA	: Data Encryption Algorithm
DES	: Data Encryption Standard
DSA	: Digital Signature Algorithm
DSS	: Digital Signature Standard
ECB	: Electronic Code Book
EKE	: Encrypted Key Exchange
ETSI	: European Telecommunication Standardization Institute
GSM	: Global System Mobile
IDE	: Interface Development Environment
IDEA	: International Data Encryption Algorithm
ISO	: International Standard Organisation
IP	: Internet Protocol
JDK	: Java Development Kit
JCE	: Java Cryptography Extension
JRE	: Java Runtime Environment

LFSR	: Linear feedback Shift Register
MAC	: Message Authentification Code
MD5	: Message Digest 5
NBS	: National Bureau of Standard
NIST	: National Institute of Standard and Technologies
NSA	: National Security American
OFB	: Output Feedback
OSI	: Open System Interconnection
PC	: Personal Computer
PGCD	: Plus Grand Commun Diviseur
PGP	: Pretty Good Privacy
PPCM	: Plus Petit Commun Multiple
RAND	: Nombre aléatoire
RDRL	: Registre de Décalage à Rétroaction Linéaire
RSA	**:** Ron RIVEST Adi SHAMIR Leonard ADLEMAN
RSADSI	: RSA DATA SECURITY, INC.
SHA	: Secure Hash Algorithm
SHS	: Secure Hash Standard
TCP/IP	: Transfert Communication Protocole/Internet Protocole

TPDU	: Transport Protocol Data Unit
addRoundKey	: Fonction d'addition de la clé de tour
Mixculoms	: Fonction de brouillages des colonnes
ShiftRows	: Fonction de décalage ligne
Subbytes	: Fonction de transformation non linéaire
$\oplus$	: Ou exclusif
$\mathbb{N}$	: Ensemble des entiers naturels
$\mathbb{Z}$	: Ensemble des entiers relatifs
mod	: modulo

INTRODUCTION

Durant ces vingt dernières années, il y a eu une explosion de recherche académique publique en cryptographie. Alors que la cryptographie classique est utilisée depuis longtemps par des citoyens ordinaires, la cryptographie par ordinateur était le domaine réservé des militaires depuis la seconde guerre mondiale. De nos jours, la cryptographie à la pointe de l'art est pratiqué en dehors de la protection des mûrs des agents militaires. Le profane peut maintenant employer des techniques cryptographiques qui le protègent contre les adversaires les plus puissants, à un niveau de sécurité qui pourrait même le protéger des agences militaires pour plusieurs années à venir.

Est-ce que l'individu moyen a besoin de ce type de sécurité ? Il peut préparer une campagne politique. Il peut concevoir un nouveau produit, discuter d'une stratégie de commercialisation ou préparer une prise de pouvoir commercial agressive. Il peut vivre dans un pays qui ne respecte pas le droit à la vie privée de ses citoyens. Il peut faire quelque chose qu'il estime ne pas être illégal mais qui l'est. Peu importe les raisons, ses données et ses communications sont personnelles, privées et ce n'est affaire des personnes d'autre à part lui.

L'énorme volume de données personnelles et confidentielles traitées, la protection des informations est devenue une priorité, en même temps que la surveillance pour la prévention des délits et crimes. Cette exigence est générale car elle concerne les intérêts des nations, des entreprises et la protection de la vie privée.

Des systèmes de chiffrement peuvent toutefois assurer la confidentialité des informations. On mélange les informations de manière tellement complexe qu'elle devient inaccessible, excepté pour le destinataire légitime, qui connaît comment le rendre intelligible

Actuellement la cryptographie, considérée comme étant l'art des mots cachés, est extrêmement répandue dans la politique, le commerce et l'industrie. Elle permet d'assurer la confidentialité et l'authenticité des messages sensibles, transmis à travers des systèmes de Télécommunications au moyen des algorithmes paramétrés par des clés.

Conscient de la sécurité que la cryptographie peut leur apporter, beaucoup sont les entreprises, qu'y accordent de l'importance et y mènent des recherches avec les Universités quant à la méthode de cryptage performante et incassable, qu'elles doivent mettre en œuvre, surtout ceux

qui opèrent dans les domaines de la Télécommunication (E-commerce et E-mail via Internet, Téléphonie mobile, …) et de l'informatique (gestion de la sécurité des systèmes informatiques).

Aussi les buts de cette étude sont :

- de présenter, en premier lieu, la notion de la cryptographie ;
- de montrer, en second lieu, comment l'utiliser en exploitant les algorithmes standards existants à fin d'assurer la sécurité des systèmes ou la confidentialité et l'authenticité des messages électroniques ;
- de réaliser un système de cryptage combiné avec les algorithmes RSA/AES.

CHAPITRE 1 : INTRODUCTION SUR LA CRYPTOGRAPHIE

1.1 Présentation [1]

Etymologiquement, le mot « cryptographie » vient du mot grecs « kruptos », qui signifie caché, et « graphein », qui signifie écrire. Par définition, la cryptographie est l'ensemble des techniques permettant de dissimuler une information à l'aide d'un code secret. Même si de tels procédés existent depuis fort longtemps, ils se sont considérablement multipliés depuis l'essor des télécommunications modernes.

Dans son sens le plus large, la cryptographie se traduit par une manipulation de chiffres, de codes ou de messages cachés. Ces derniers, écrits à l'encre invisible ou dissimulés dans des textes apparemment quelconques, n'ont d'intérêt que s'ils restent insoupçonnés : une fois qu'ils sont découverts, il n'est généralement pas très difficile de les déchiffrer. Les codes, dans lesquels les mots, les phrases ou les messages complets sont représentés par des expressions ou symboles prédéfinis, sont généralement impossibles à lire sans l'annuaire contenant les clés des codes, mais encore faut-il pouvoir transmettre cet annuaire de façon confidentielle. Enfin, le chiffrement consiste à transformer les symboles d'un texte en cryptogramme au moyen d'un calculateur ou d'une machine à chiffrer, le décryptage s'obtenant par la transformation inverse.

1.2 Historiques [1] [2]

Dès l'Antiquité, les peuples employèrent des codes secrets dans certains de leurs textes : Vers 1900 av. J.-C., un scribe égyptien a employé des hiéroglyphes non conformes à la langue correcte dans une inscription. Kahn le qualifie de premier exemple documenté de cryptographie écrite.

Quatre siècles plus tard, vers 1500 av. J.-C., une tablette mésopotamienne contient une formule chiffrée pour la fabrication de vernis pour les poteries.

Cinq siècles avant notre ère, des scribes hébreux mettant par écrit le livre de Jérémie ont employé un simple chiffre de substitution connu sous le nom d'Atbash. C'était un des quelques chiffres hébreux de cette époque, avec Albam et Atbah.

En 487 av. J.-C., les grecs emploient un dispositif appelé la scytale, un bâton autour duquel une bande longue et mince de cuir était enveloppé et sur laquelle on écrivait le message. Le

cuir était ensuite porté comme une ceinture par le messager. Le destinataire avait un bâton identique permettant d'enrouler le cuir afin de déchiffrer le message.

Sur le champ de bataille, les Spartes communiquaient souvent avec leurs généraux par le biais de messages écrits sur un ruban de parchemin enroulé en spirale autour d'une règle. Une fois le ruban déroulé, on ne pouvait lire le message qu'en enroulant le ruban autour d'une règle identique. Jules César se servit également de codes secrets pour correspondre avec ses hommes, et laissa même son nom à un chiffre particulier selon lequel chaque lettre est décalée de quatre rangs par rapport à sa place dans l'alphabet (le « A » devenant un « D », le « B » un « E », etc.).

Le Kâma-Sûtra est un texte écrit au 5e siècle par le brahman Vatsayayana, mais fondé sur des manuscrits du 4e siècle avant J.-C. Le Kâma-Sûtra recommande que les femmes apprennent 64 arts, entre autres cuisiner, s'habiller, masser et élaborer des parfums. La liste comprend aussi des domaines moins évidents, comme la prestidigitation, les échecs, la reliure et la tapisserie. Le numéro 45 de la liste est le mlecchita-vikalpa, l'art de l'écriture secrète, qui doit leur permettre de dissimuler leurs liaisons.

Abu Bakr ben Wahshiyya publie, en 855, plusieurs alphabets secrets utilisés à des fins de magie, dans son livre Kitab shauk almustaham fi ma'arifat rumuz al aklam, le livre de la connaissance longuement désirée des alphabets occultes enfin dévoilée.

Au 9é siècle, Abu Yusuf Ya'qub ibn Is-haq ibn as-Sabbah Oomran ibn Ismaïl Al-Kindi rédige le plus ancien texte connu décrivant la technique de décryptement appelée analyse des fréquences.

A partir de 1226 une timide cryptographie politique apparait dans les archives de Venise, où des points ou des croix remplacent les voyelles dans quelques mots épars.

Gabriel de Lavinde compose, en 1379, un recueil de clés, dont plusieurs combinent code et substitution simple. En plus d'un alphabet de chiffrement, souvent avec des nulles, on trouve un petit répertoire d'une douzaine de noms communs et de noms propres avec leurs équivalents en bigrammes. C'est le premier exemple d'un procédé qui devait prévaloir pendant 450 ans en Europe et en Amérique : le nomenclateur.

Le premier grand cryptanalyste européen fut peut-être Giovanni Soro, nommé secrétaire chiffreur en 1506. Il devint secrétaire du chiffre de Venise. Le Vatican lui-même testa ses chiffres sur Soro, qui les perça à jour une première fois. Le Pape envoya d'autres textes chiffrés à Soro afin

de savoir si le meilleur cryptanalyste pouvait battre son chiffre. Soro renvoya les textes en écrivant qu'il n'avait pas réussi à les déchiffrer mais on ne sut jamais s'il avait dit la vérité, ou s'il avait menti pour pouvoir décrypter sans difficulté tout message émanant des autorités pontificales...

Blaise de Vigenère écrit son « Traité des chiffres ou secrètes manières d'écrire » en 1585. Il présente entre autres un tableau du type Trithême, que l'on dénomme aujourd'hui à tort carré de Vigenère. On considéra longtemps ce chiffre comme indécryptable, légende si tenace que même en 1917, plus de cinquante ans après avoir été cassé, le Vigenère était donné pour « impossible à déchiffrer » par la très sérieuse revue Scientific American.

En 1854, soit 269 ans après sa publication, Charles Babbage casse le chiffre de Vigenère, mais sa découverte resta ignorée, car il ne la publia pas. Ce travail ne fut mis en lumière qu'au vingtième siècle, lors de recherches effectuées sur l'ensemble des papiers de Babbage.

Gilbert S. Vernam, travaillant pour AT&T, invente en 1917 une machine de chiffre polyalphabétique pratique capable d'employer une clé qui est totalement aléatoire et ne se répète jamais - un masque jetable. C'est le seul chiffre, dans nos connaissances actuelles, dont on a prouvé qu'il était indécryptable en pratique et en théorie. Ce procédé ne fut cependant jamais utilisé par l'armée car il exigeait de devoir produire des millions de clés différentes (une par message), ce qui est impraticable. Par contre, il fut utilisé par les diplomates allemands dès 1921.

En 1918, Arthur Scherbius fait breveter sa machine à chiffrer Enigma. Le prix d'un exemplaire s'élevait à 20 000 livres sterling actuelle. Ce prix sembla décourager les acheteurs potentiels. La machine Enigma ne fut pas un succès commercial mais elle fut reprise et améliorée pour devenir la machine cryptographique de l'Allemagne nazie pendant la seconde guerre mondiale. Les Allemands communiquèrent ainsi par radio grâce au code Enigma, que les Britanniques réussirent à percer grâce à un ordinateur numérique baptisé Colossus. La machine Enigma a été cassée par le mathématicien polonais Marian Rejewski, qui s'est fondé seulement sur un texte chiffré et une liste des clés quotidiennes obtenues par un espion. De la même manière, les Américains parvinrent à décrypter certains chiffres employés par les Japonais. Ces divers succès, qui contribuèrent pour beaucoup à la victoire des Alliés, ont prouvé qu'aucun cryptage, aussi sophistiqué soit-il, ne doit être considéré comme inviolable, et qu'il est par conséquent dangereux de lui faire aveuglement confiance.

Whitfield Diffie et Martin Hellman publient, en 1976, New Directions in Cryptography, introduisant l'idée de cryptographie à clé publique. Ils donnent une solution entièrement nouvelle au problème de l'échange de clés.

En 1978, l'algorithme de chiffrement à clé publique « RSA » est publié par Ronald L. Revest, A. Shamir et Léonard M Adleman.

En 1987, le RC4 est développé par Ronald L. Revest pour la RSA Security et sera gardé secret jusqu'à 1994, où l'algorithme est rendu publique anonymement dans une liste de distribution de « cipher punk ».

En 1990, première publication des résultats expérimentaux de la cryptographie quantique par Charles H. Benêt et Gilles Brassard.

En 1991, Phil Zimmermann rend disponible sa première version de PGP.

En 1992, l'IDEA est inventé en suisse par Huejeya Laj et James Massey.

En 1993, Bruce Schneier conçoit Blowfish, et Don Choppers Schmith crée Seal.

En 1994, Ronald L. Revest, déjà auteur de RC1, RC4 publie RC5 et peu après, un standard régissant les signatures numériques voit le jour : « DSA ».

En 1995, le NIST développe le « SHA ».

En 1998, l'algorithme de Rijndael est finalisé et soumis au NIST pour devenir le nouveau standard de l'avancé : l'AES. Seize autres algorithmes font parties du groupe dont Mars, RC6, Serpent, et Twofish.

En 2000, Rijndael devient l'AES standard de chiffrement avancée.

Depuis 2000, des nouvelles versions de PGP apparaissent.

1.3 Généralités sur la cryptologie [3]

La cryptologie, étymologiquement « la science du caché » par extension « la science du secret », est composée de deux disciplines complémentaires : la cryptographie et la cryptanalyse. La cryptographie est l'art de cacher l'information. Elle répond aux besoins suivant :

- Confidentialités : garantir que les données échangées ne sont dévoilées qu'aux personnes voulues.

- Intégrités : garantir que les données échangées ne sont pas altérées intentionnellement ou non.

- Authenticités : prouver l'identité d'une personne ou d'un document.

- Signature : permettre à une personne de signer informatiquement un document sans qu'elle puisse le renier par la suite.

Quant à la cryptanalyse, c'est le pendant naturel de la cryptographie : lorsqu'un système cryptographique est utilisé, certains vont tenter de l'attaquer pour mettre en défaut sa sécurité. L'ensemble des attaques sur les cryptosystèmes est la cryptanalyse.

1.3.1 Les techniques de cryptographie

La cryptologie, science fondamentale qui régit la cryptographie, est essentiellement basée sur l'arithmétique. Ainsi dans le cas d'un texte, il s'agit de transformer les lettres qui composent le message en une succession de chiffres (sous forme de bits dans le cas de l'informatique pour permettre le fonctionnement binaire des ordinateurs), puis ensuite de faire des calculs sur ces chiffres pour:

- d'une part les modifier et les rendre incompréhensibles. Le résultat de cette modification (le message chiffré) est appelé cryptogramme.

- d'autre part, faire en sorte que le destinataire sache les déchiffrer en utilisant les outils préétablis ou joints aux données.

Le fait de coder un message de façon à le rendre secret s'appelle *chiffrement*. La méthode inverse consistant à retrouver le message original, est appelée *déchiffrement*.

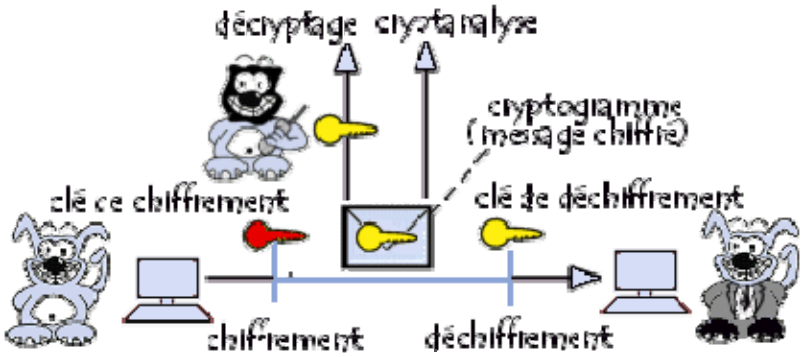

Figure 1.01: processus cryptographique

Le chiffrement se fait généralement à l'aide d'une *clé de chiffrement*, le déchiffrement avec une *clé de déchiffrement*. On distingue généralement deux types de clés :

- *Les clés symétriques*: on utilise des clés identiques à la fois pour le chiffrement et pour le déchiffrement. On parle alors de chiffrement symétrique ou de chiffrement à clé secrète. Il s'agit de la cryptographie à clé privée.

- *Les clés asymétriques*: on utilise de clés différentes pour le chiffrement et le déchiffrement. On parle alors de chiffrement asymétrique. Il s'agit de la cryptographie à clé publique.

Au cours des années soixante dix, un système de sécurisation basé sur la polarisation des photons est apparu: la cryptographie quantique. Cette technique est différente des autres cryptosystèmes à clé puisqu'elle fonctionne sur des propriétés physiques intrinsèques au système.

1.3.2 Chiffrement et déchiffrement

Le premier objectif de la cryptographie est d'assurer la confidentialité des données. Supposons que deux individus veuillent se communiquer entre eux sur un canal non sûr sans que personne d'autre ne puisse lire le contenu du message ; ces deux individus, que nous appellerons Alice et Bob, comme le veut la tradition cryptographique doivent se protéger des éventuelles attaques du dénommé Oscar utiliser un système cryptographique. Si Alice veut envoyer un message à Bob, elle chiffre le texte du message grâce à une fonction de chiffrement E utilisant une clé *k.* elle envoie alors le message chiffré que seul Bob sera capable de déchiffrer à l'aide d'une fonction de déchiffrement D et d'une donnée supplémentaire (la clé par exemple). On peut alors formaliser la notion de fonction de chiffrement et de déchiffrement de la façon suivante :

Soit M l'ensemble des textes clairs, C l'ensemble des textes chiffrés, K l'ensemble des clés. Une fonction de chiffrement E associe à tout couple (k, m) de K x M un élément c= E (k, m) de C. la fonction de déchiffrement correspondante, paramétrée par la clé de déchiffrement d, associe alors au couple (d, c) de K x C le message m de M, elle vérifie :

$$D(d, E(k, m)) = m. \qquad (1.01)$$

1.3.3 Deux méthodes pour échanger de l'information

Jusqu'en 1976, la seule méthode connue pour échanger de l'information était la cryptographie à clé secrète. Pour que deux personnes puissent échanger un message chiffré, elles devaient partager un secret connu d'elles seules. En 1976 est apparues une nouvelle forme de chiffrement dite à clé publique où les protagonistes ne sont plus tenus de partager une même clé.

1.3.3.1 Chiffrement symétrique

C'est le plus facile à comprendre, c'est aussi la méthode de chiffrement la plus facile à réaliser et qui consomme le moins de ressources de calcul et de bande passante.

Les deux hôtes qui doivent échanger des données confidentielles (secrètes) disposent tous les deux d'une clé identique. L'émetteur chiffre les données avec, puis les envoie au récepteur. Ce dernier déchiffre avec la même clé pour récupérer des données lisibles.

Cette méthode assure la confidentialité des données, celui qui intercepterait la communication ne pourra pas lire les données échangées tant qu'il n'aura pas pu se procurer la clé. Il n'y a aucune authentification de faite sur l'émetteur comme sur le récepteur, sauf si deux personnes seulement disposent de la clé. Le principal souci avec cette méthode, c'est qu'il faut s'échanger la clé et lors de cet échange, sans précautions particulières, n'importe quoi peut se produire. Si Alice et Bob décident d'utiliser pour communiquer un principe de chiffrement symétrique, ils doivent partager la même clé k, connue d'eux seuls qui est identique pour le chiffrement et le déchiffrement.

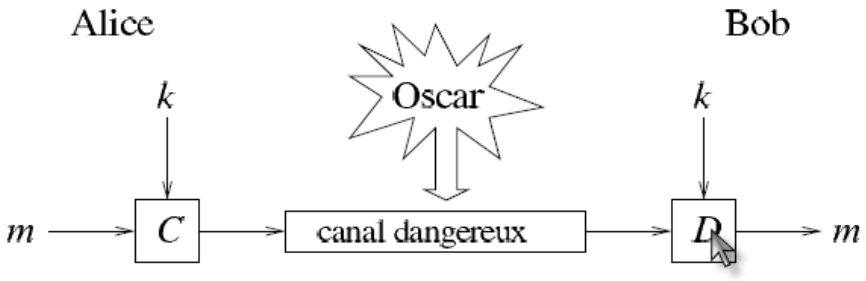

Figure 1.02 : chiffrement symétrique

Un de exemples les plus connues de la cryptographie à clé secrète est le chiffrement de Vernam, inventé par ce dernier en 1917. C'est un des seuls chiffrements inconditionnellement sûrs (c'est-à-dire pour lequel la connaissance du chiffré ne permet pas d'obtenir aucune information sur

le texte clair). Si Alice veut envoyer un message m de longueur n à Bob en utilisant cette méthode, elle doit générer une clé aléatoire k de longueur n et calculer le chiffré à envoyer c= m ⊕ k, où ⊕ représente le ou exclusif bit à bit. Pour déchiffrer le message c, Bob qui possède la même clé k qu'Alice n'a alors qu'a calculer m= c ⊕ k. pour que ce chiffrement soit inconditionnellement sûr, il ne faut utiliser la clé k qu'une seule fois, d'où le deuxième nom de chiffrement de Vernam : « One Time Pad ». L'inconvénient majeur de cet algorithme est la longueur de la clé. On doit avoir k≥taille(m).

1.3.3.2 Chiffrement asymétrique

En 1976, Diffie et Hellman décrivent le principe d'un nouveau type d'échange d'information : la cryptographie asymétrique ou à clé publique. De nombreux algorithmes permettant de réaliser un cryptosystème à clé publique ont été proposés depuis. Ils sont le plus souvent basés sur des problèmes mathématiques difficiles à résoudre, donc leur sécurité est conditionnée sur ces problèmes, sus lequel on a maintenant une vaste expertise. Mais, si quelqu'un trouve un jour le moyen de simplifier la résolution d'un de ces problèmes, l'algorithme correspondant s'écroulera.

Tous les algorithmes actuels présentent l'inconvénient d'être bien plus lent que les algorithmes à clé secrète ; de ce fait, ils sont souvent utilisés non pour chiffrer directement des données, mais pour chiffrer une clé de session secrète. Certains algorithmes asymétriques ne sont adaptés qu'au chiffrement, tandis que d'autres ne permettent que la signature. Seuls trois algorithmes sont utilisables à la fois pour le chiffrement et la signature : RSA, El Gamal et Rabin.

Si Alice veut envoyer un message confidentiel à Bob, elle se procure (par exemple dans un annuaire public) la clé publique k_p de Bob et chiffre le message *m* à l'aide d'un algorithme de chiffrement asymétrique et la clé k_p. Elle envoie ensuite sur le canal non sûr le message chiffré *c* que seul Bob peut déchiffrer grâce à la clé secrète correspondante k_s, connue de lui seul.

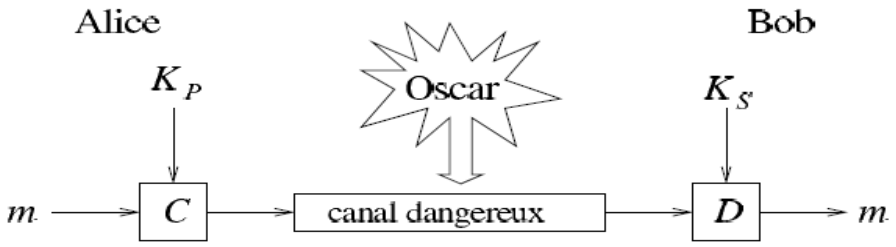

Figure 1.03 : chiffrement asymétrique

A priori, la cryptographie à clé publique apparait comme la meilleure méthode de chiffrement, car pour que deux personnes communiquent sur un canal non sûr, ils n'ont pas besoin de pré-échanger une clé. Cependant, les algorithmes à clé publique sont lents. Par exemple, l'algorithme RSA chiffre au moins mille fois plus lentement que le DES (Data Encryption Standard), ancien standard de chiffrement à clé secrète. La méthode la plus efficace et la plus usitée pour échanger de l'information sur un canal non sûr est de ce fait une combinaison des deux méthodes de chiffrement: on chiffre le message à envoyer à l'aide d'un algorithme symétrique et d'une clé secrète, clé transmise au destinataire à l'aide d'un algorithme asymétrique.

1.3.4 Deux méthodes pour chiffrer en clé secrète

Les deux méthodes les plus usitées en cryptographie à clé secrète sont le chiffrement à flot et le chiffrement par blocs.

1.3.4.1 Chiffrement à flot

Le principe des algorithmes à flots informatiquement sûr repose sur le partage d'une clé K de longueur constante et non sur le partage d'une clé de longueur égale à celle du message à chiffrer. L'algorithme utilisé chiffre « à la volée » le flot de message clair par un ou-exclusif bit à bit (ou une opération similaire) avec une suite chiffrante (k_i) pour produire un flot de texte chiffré, comme dans le cas du chiffrement de Vernam. La suite chiffrante $k= (k_i)$ est produite par un générateur pseudo-aléatoire G à partir de la clé K et d'un vecteur d'initialisation *IV*.

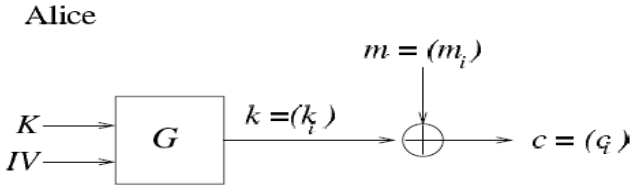

Figure 1.04 : Principe du chiffrement à flot

1.3.4.2 Chiffrement par blocs

Le chiffrement par bloc s'oppose à celui par flot. Contrairement à ce dernier, il ne chiffre pas bit à bit, mais découpe des blocs entiers de bits pour les passer à la moulinette des algorithmes. On peut ainsi schématiser le circuit emprunté par les données. Pour calculer la valeur chiffrée C, le bloc de bits M est utilisé avec une clé K de longueur définie, par l'algorithme E.

Ce circuit peut être plus ou moins complexe en fonction du mode employé. Il en existe quatre pour le chiffrement par bloc. Ces modes ont été définis lors de la mise en place de Data Encryption Standard.

Il est possible d'en implémenter plusieurs sur un même support mais cela n'est pas une obligation.

1.3.4.2.1 Mode dictionnaire

L'Electronic Code Book mode est le plus simple des quatre présentés. Il traite chaque bloc indépendamment des autres, ce qui permet d'insérer un ordre aléatoire de traitement. En contre partie, il se voit plus sensible aux attaques par regroupement et analyse statistique.

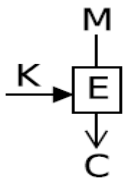

Figure 1.05 : Mode dictionnaire

1.3.4.2.2 Mode chainage de blocs chiffrés

Mode le plus couramment utilisé, le Cipher Block Chaining mode permet d'augmenter la sécurité en introduisant une complexité supplémentaire dans le processus de chiffrement en créant une dépendance entre les blocs successifs.

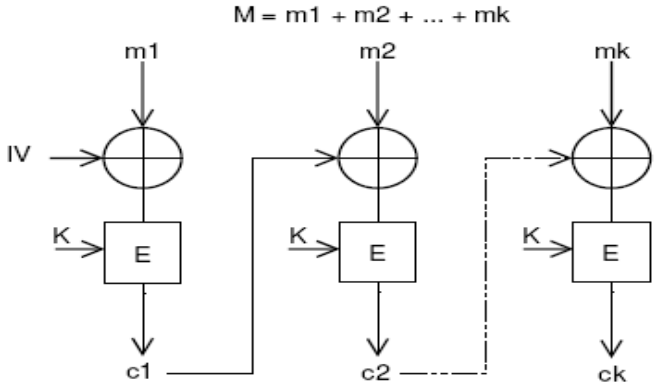

Figure 1.06 : Cipher Block Chaining mode

Comme le montre la figure 1.06, chaque m[i] est transformée avant le chiffrement. On applique pour le Chiffrement :

$$C[0]=E(m[0]\text{^}IV) \quad (1.02)$$

$$C[n] = E\,(m[n] \text{^} C\,[n-1]),\ si\ (n > 0) \quad (1.03)$$

Où représente ^ désigne le ou exclusif (XOR), et C [i-1] le résultat du bloc précédent venant d'être chiffré. Pour le premier bloc, il est nécessaire d'employer un vecteur d'initialisation noté IV, puisqu'il n'y a pas de valeur C_{-1} existante. Ce vecteur est constitué d'un bloc de bits aléatoires qui sera transmis au destinataire du message, afin de pouvoir effectuer le déchiffrement. Il peut être transmis publiquement sans que cela n'affecte la sécurité.

1.3.4.2.3 Mode de rebouclage chiffré

Le Cipher Feedback mode se rapproche d'un algorithme par flot. Le message M est divisé en sous blocs m[i] de taille allant de 1 à n bits. Contrairement au mode précédent, L'opération XOR est appliquée entre le bloc de texte clair et le résultat précédent chiffré à nouveau par la fonction de chiffrement.

Pour le Chiffrement: I[0] = VI

$$I[n]=C[n-1],si(n>0) \quad (1.04)$$

$$C[n] = m[n] \text{^} E\,(I[n]) \quad (1.05)$$

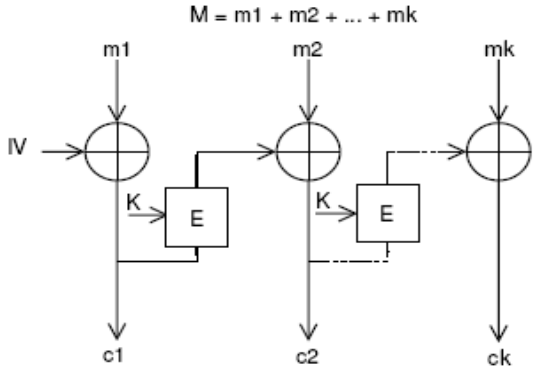

Figure 1.07 : Cipher Feedback mode

Le chiffrement E ne s'effectue pas après le XOR du bloc en cours et n'influe pas sur le Ci courant, mais sur le suivant. Pour le premier bloc de texte clair, on génère un vecteur d'initialisation.

1.3.4.2.4 Mode rebouclage sur la sortie

Dans ce quatrième et dernier mode, nommé Output Feedback, l'opération effectuée est indépendante des données, qu'elles soient en clair ou chiffrées.

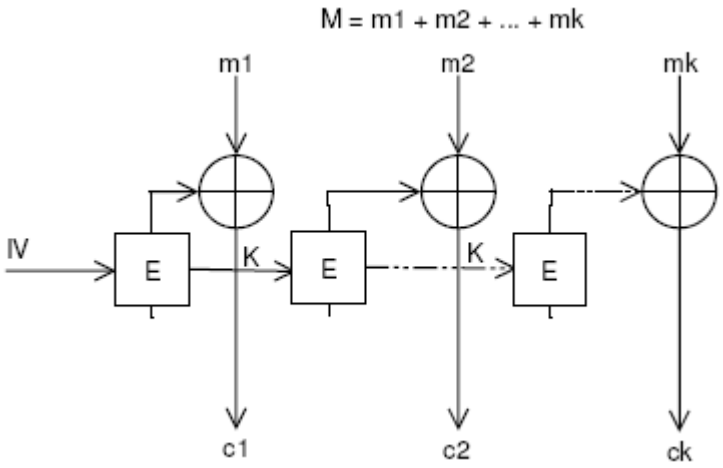

Figure 1.08 : Output Feedback

1.4 Implémentation de la cryptographie dans le modèle OSI [4]

Au début des années 70, chaque constructeur a développé sa propre solution réseau autour d'architecture et de protocoles privés et il s'est vite avéré qu'il serait impossible d'interconnecter ces différents réseaux « propriétaires » si une norme internationale n'était pas établie.

1.4.1 Le système OSI

Cette norme établie par *l'International Standard Organization* (ISO) est la norme Open System Interconnection (OSI : interconnexion de systèmes ouverts). Un système ouvert est un ordinateur, un terminal, un réseau, n'importe quel équipement respectant cette norme et donc apte à échanger des informations avec d'autres équipements hétérogènes et issus de constructeurs différents.

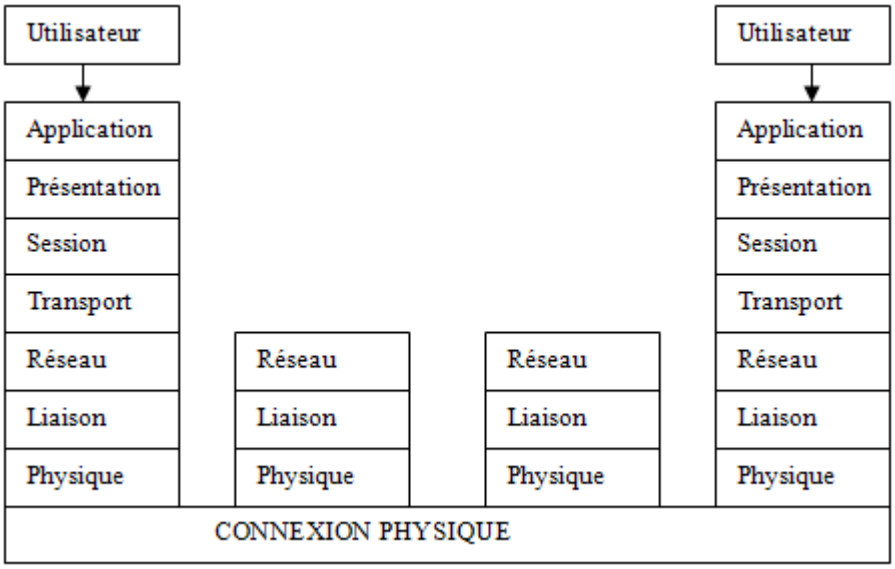

Figure 1.09 : Architecture OSI

Le premier objectif de la norme OSI a été un modèle de toute architecture de réseau basé sur un découpage en 7 couches (cf. Figure 1.09), chacune des couches correspondant à une fonctionnalité particulière d'un réseau. Les couches 1, 2, 3, 4 sont dites basses, et les couches 5, 6, 7 sont dites hautes.

Chaque couches est constituée d'éléments matériels et logiciels et offre un service à la couche située immédiatement au dessus d'elle en lui épargnant les détails d'implémentation nécessaires.

Nous allons maintenant détailler les caractéristiques de chacune de ces couches en précisant d'abord que les fonctions et services définis dans les couches du modèle OSI peuvent se retrouver dans d'autres couches dans les systèmes opérationnels disponibles sur le marché.

1.4.1.1 La couche physique

Par définition, la couche physique fournit les moyens mécaniques, électriques, fonctionnels et procéduraux nécessaire à l'activation, au maintien et à la désactivation des connexions physiques destinées à la transmission de bits entre deux entités de liaisons de données.

Ici, on s'occupe donc de la transmission de bit de façon brute, l'important est que l'on soit sûr que si l'émetteur envoie un bit à 1 alors que le récepteur reçoit un bit à 1. Les normes et standards de la couche physique définissent le type de signaux émis (modulation, puissance, portée,…), la nature et la caractéristique des supports (câble, fibre optique,…)

1.4.1.2 La couche liaison de donnée

Par définition, la couche liaison fournit les moyens fonctionnels et procéduraux nécessaires à l'établissement, au maintient, et à la libération des connexions de liaisons de données entre entités du réseau. Elle détecte et corrige, si possible, les erreurs dues au support physique et signale à la couche réseau les erreurs irrécupérables. Elle supervise le fonctionnement de la transmission et définit la structure syntaxique des messages, la manière d'enchainer les échanges selon un protocole normalisé ou non.

Une connexion de liaison de données est réalisée à l'aide d'une ou plusieurs liaisons physiques entre deux machines adjacentes dans le réseau donc sans nœuds intermédiaires entre elles.

1.4.1.3 La couche réseau

Par définition, la couche réseau assure toutes les fonctionnalités de relai et d'améliorations de services entre entité du réseau, à savoir : l'adressage, le routage, le contrôle de flux et la détection et correction d'erreurs non réglées à la couche 2.

À ce niveau là de l'architecture OSI il s'agit de faire transiter une information complète (un fichier par exemple) d'une machine à une autre à travers un réseau de plusieurs ordinateurs. Il existe deux grandes possibilités pour établir un protocole de niveau réseau : le mode avec connexion et le mode sans connexion. Le premier cas est celui adopté par la norme X25.3 et le second est celui du protocole IP du réseau internet.

1.4.1.4 La couche transport

Par définition, la couche transport assure un transfert de données transparents entre entités de session et en les déchargeant des détails d'exécutions. Elle a pour rôle d'optimiser l'utilisation des services de réseau disponibles afin d'assurer au moindre coût les performances requises par la couche session.

C'est la première couche à résider sur les systèmes d'extrémités. Elle permet aux deux applications de chaque extrémité de dialoguer directement indépendamment de la nature des sous-réseaux traversés et comme si le réseau n'existait pas. Au niveau inferieur de la couche réseau seule la phase d'établissement de la liaison physique s'effectue de bout en bout, alors que les transferts d'informations se font de proche en proche.

La couche transport doit assurer, en mode connecté ou non connecté, un transfert transparent de données entre utilisateurs de service réseau en leur rendant invisible la façon dont les ressources de communication sont mises en œuvres. Cette notion de transparence impliqua par exemple de pouvoir acheminer de bout en bout des TPDU (Transport Protocol Data Unit) dont la taille peut varier de 128 à 8192 octets. Cette taille est cependant fixe une fois que la valeur à été négociée.

1.4.1.5 La couche session

Les couches session, présentation et applications constituent les couches hautes du modèle OSI et offrent les services orientés vers les utilisateurs alors que les couches basses sont concernées par la communication fiable de bout en bout. Elles considèrent que la couche transport fournit un canal fiable de communication et ajoutent des caractéristiques supplémentaires pour les applications.

Par définition, la couche session fournit aux entités de la couche présentation les moyens d'organiser et synchroniser les dialogues et les échanges de données.

1.4.1.6 La couche présentation

Par définition, la couche présentation s'occupe de la syntaxe et de la sémantique des informations transportées en se chargeant notamment de la représentation des données.

C'est à ce niveau de la couche présentation que peuvent être implantées des techniques de compression (exemple code de Huffman) et de chiffrement de données (RSA, DES, etc.)

1.4.1.7 La couche application

Par définition, la couche application donne au processus d'application le moyen d'accéder à l'environnement OSI et fournit tout les services directement utilisables par l'application, a savoir :

- Le transfert d'informations
- L'allocation des ressources
- L'intégrité et la cohérence des données accédées
- La synchronisation des applications coopérantes

En fait, la couche application gère les programmes de l'utilisateur et définit des standards pour que les différents logiciels commercialisés adoptent les mêmes principes, comme par exemple :

- Notion de fichier virtuel représenté sous forme d'arbres pour les applications de transfert de fichiers, opérations permises sur un fichier, accès concurrentiels,…
- Découpages des fonctions d'une application de courrier électroniques qui se compose d'un contenu (en-tête et corps) et d'une enveloppe. Une fonctionnalité de l'application gère le contenu et une autre le transfert en interprétant l'enveloppe.

1.4.2 La cryptographie dans le modèle OSI [5]

L'implémentation de la cryptographie dans le modèle OSI peut s'effectuer en deux types : dans les couches basses et dans les couches hautes

Si le chiffrement a lieu dans les couches basses, on parle de chiffrement lien par lien, les données qui passent par le lien seront chiffrés.

Si le chiffrement a lieu dans les couches hautes, on parle de chiffrement de bout en bout. Les données restent chiffrées jusqu'à ce que le destinataire le déchiffre.

1.4.2.1 Chiffrement lien par lien

L'endroit le plus simple pour effectuer le chiffrement se trouve dans la couche physique. Le dispositif de chiffrement est placé à cet endroit. Il chiffre ensuite toutes les informations qui passent entre lui et l'autre dispositif reversent. La sécurité de cette méthode réside sur le fait que les données qui circulent dans le réseau sont cryptées, l'interception des données ne peut corrompre personne (on ne connait ni le contenu ni sa destination).

1.4.2.2 Le chiffrement de bout en bout

Dans ce cas, on place le dispositif de chiffrement entre la couche réseau et la couche transport. Les données chiffrées sont les unités de transport.

1.5 Cryptanalyse [6] [7] [8] [9]

C'est l'étude des systèmes cryptographiques, en particulier de leurs faiblesses, dans le but de retrouver des messages clairs correspondant à des messages chiffrés dont on n'est pas destinataire.

Un attaquant est donc une personne qui tente de décrypter des messages, c'est-à-dire de retrouver des clairs à partir de chiffrés sans connaitre la clé. On réserve généralement le verbe déchiffrer à l'action du destinataire légitime qui effectue l'opération inverse de chiffrement.

Si le but de la cryptographie est d'élaborer des méthodes de protections, 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 eut conduire à différents résultats :

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

1.5.1 Attaques des fonctions de chiffrements

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

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

- L'attaque à texte chiffré seulement, où le cryptanalyste ne connait 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, ce qui permet de valider ou non le déchiffrement d'un cryptanalyste.

- L'attaque à texte en clair connu, où la cryptanalyste connait 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, ce type d'attaque est très courant.

- L'attaque à texte en clair choisi, où le cryptanalyste peut choisir des textes en clair à chiffrer et donc utiliser des textes apportant plus d'informations sur la clé. S'il peut de plus adapter ses choix en fonction des textes chiffrés précédents, on parle aussi d'attaque adaptive.

- L'attaque à texte chiffré choisi, qui est l'inverse de la précédente : le cryptanalyste peut choisir des textes chiffrés pour lesquels il connaitra le texte en clair correspondant, sa tâche est alors de retrouver la clé. Ce type d'attaque est principalement utilisé contre les systèmes à clé publiques, pour retrouver la clé privée.

1.5.2 Attaques sur les algorithmes symétriques

1.5.2.1 Attaques au niveau des clés

- L'attaque la plus simple sur le plan conceptuel est l'attaque exhaustive ou attaque en force brute, ce qui consiste à essayer toutes les clés possibles. Le nombre de test à effectuer est dépendant de la longueur de la clé.

- L'attaque par dictionnaire est la méthode par laquelle le cryptanalyste se constitue un dictionnaire de mots à tester. Ce type d'attaque est beaucoup plus rapide qu'une attaque exhaustive.

1.5.2.2 La cryptanalyse différentielle

Elle utilise une attaque à texte clair choisi et basé sur l'observation de l'évolution des différences entre deux textes lorsqu'ils sont chiffrés avec la même clé. A force d'analyser des paires de textes, on finit par trouver la clé recherchée en réduisant suffisamment le nombre de clés possibles pour pouvoir mener une attaque exhaustive rapide.

Elle peut également être utilisée pour attaquer d'autres types d'algorithmes comme les fonctions de hachages.

1.5.2.3 La cryptanalyse linéaire

La cryptanalyse linéaire utilise une attaque à texte clair connu et consiste à spécifier 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.

1.5.3 Attaques sur les algorithmes asymétriques

1.5.3.1 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 reposent sur des problèmes mathématiques difficiles à résoudre. C'est pourquoi les paramètres qui influencent la difficulté de la résolution du problème déterminent 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.

1.5.3.2 L'attaque à texte clair deviné

Un point faible des algorithmes à clé publique est le caractère publique de la clé de chiffrement : le cryptanalyste ayant connaissance de cette clé peut mener une attaque à texte clair deviné, qui consiste à tenter de deviner le texte en clair et à le chiffrer pour vérifier son exactitude. En effet, un attaquant pourrait se constituer une liste exhaustive des textes en clairs possible et des textes chiffrés correspondants. Il n'aurait alors aucun mal à retrouver le texte en clair correspondant à un cryptogramme donné.

1.5.3.3 Attaque à texte chiffré choisi

La plupart des algorithmes asymétriques sont sensibles aux attaques à texte chiffré choisi. Il convient donc de, si on utilise le même algorithme pour le chiffrement et pour la signature, 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 distincts pour ces deux opérations.

1.5.3.4 Attaque temporelle

Ce type d'attaque utilise la mesure du temps pris par des opérations cryptographiques pour retrouver les clés privées utilisées. Une recherche s'est intéressée à l'introduction délibérée d'erreurs dans les processeurs cryptographiques pour tenter d'obtenir des informations sur la clé.

CHAPITRE 2 : THEORIE DES NOMBRES ET CRYPTOGRAPHIE

2.1 Introduction

Le but de ce paragraphe est de donner les quelques notions de base d'arithmétique, qui sont nécessaire quand on parle de cryptographie. Tout bien précisant que ce ne sont que des notions, mais à l'heure actuelle les prés requis mathématiques pour faire de la cryptographie sérieusement sont d'un autre niveau.

2.2 Cryptographie classique

Quand on parle de cryptographie classique, ce qui vient tout de suite à l'esprit c'est la cryptographie mise en œuvre par les anciens civilisations.

2.3 Cryptographie moderne [10]

2.3.1 Un peu de formalisme mathématique

Comme nous l'avons dit dans l'introduction, l'informatique a considérablement élargi le domaine d'application de la cryptographie. Nous sommes dans un monde qui échange énormément de données et le besoin de sécurité est à la mesure des enjeux économiques énormes de ces échanges. La cryptographie a cessé d'être un art, une sorte de hobby pour officier en retraite (Kasiski écrivit son livre après la fin de sa carrière militaire), pour devenir partie intégrante de différentes disciplines que recouvre le mot informatique.

C'est la raison pour laquelle il est nécessaire de développer un certain formalisme mathématique, évité jusque-là, mais qui sera utile pour la suite de l'article. Néanmoins, le vocabulaire précédemment introduit va nous aider à donner des définitions précises. Pour rester le plus simple possible, nous allons limiter notre formalisation au problème du chiffrement qui consiste, comme nous l'avons expliqué dans l'introduction, à assurer la confidentialité d'une communication. Nous aurons à manipuler trois ensembles :

- Un ensemble de messages (que nous notons M) ;
- Un ensemble de cryptogrammes (que nous notons C) ;
- Un ensemble de clés (que nous notons K).

$$\begin{cases} M \times K \longrightarrow C \\ (M, K) \longrightarrow E(M, K). \end{cases}$$

On note souvent E_K (M) l'élément E (M, K) de sorte que, pour chaque clé K fixée, nous ayons une application E_K :

$$\begin{cases} M \longrightarrow C \\ M \longrightarrow E_K(M) \end{cases}$$

Nous disons alors que cette application E est un algorithme de chiffrement si, pour toute clé *k* dans K, l'application E_K est inversible (en fait, pour éviter la perte d'information au cours de la transmission, il suffit d'exiger que E_K soit injective c'est-à-dire que deux éléments distincts de M aient des images distinctes dans C par exemple E_K).

L'application inverse de E_K est notée D_K. Ainsi, pour tout message M, on a :

$$D_K \circ E_K(M) = M \qquad (2.01)$$

2.3.2 Sécurité des algorithmes

Pour simplifier, et pour le moment, nous allons parler seulement de chiffrement. Notre introduction des algorithmes à clé publique s'est faite en disant qu'une certaine fonction inversible E_K possédait une application réciproque D_K qui n'était pas calculable pratiquement à partir d'E_K seule. Ce type de considération pose le problème de savoir comment évaluer la sécurité d'un algorithme.

Nous allons distinguer deux types de sécurité :

- Les algorithmes inconditionnellement sûrs ;
- Les algorithmes sûrs d'un point de vue informatique.

2.3.2.1 La sécurité inconditionnelle ou secret parfait signifie que la connaissance d'un cryptogramme ne donne aucune indication sur le message clair correspondant. Cela se définit de manière parfaitement rigoureuse. Il s'agit d'un concept lié à la théorie de l'information qui a été introduit dans les travaux de Claude Shannon, dans les années

1940. On connaît un seul exemple de schéma cryptographique inconditionnellement sûr en ce sens. Il s'agit du procédé (connu en anglais sous le nom de « one time pad ») dont l'idée remonte à Vernam. Pour simplifier, supposons que le message est une suite binaire de longueur N :

$$m_0\, m_1 \ldots m_N\ ;$$

La clé secrète doit être une suite binaire aléatoire de longueur N également :

$$k_0\, k_1 \ldots k_N.$$

Cela suppose que les deux correspondants aient pu auparavant échanger de manière secrète cette clé. On sait, par ailleurs, fabriquer ce type de clé à partir de phénomènes physiques aléatoires. Le cryptogramme est de nouveau une suite binaire de longueur N :

$$c_0\, c_1 \ldots c_N$$

Obtenue par « ou exclusif » (langage des informaticiens ou électroniciens) des deux suites précédentes :

$$c_i = m_i \oplus k_i \qquad (2.02)$$

C'est-à-dire par addition modulo 2 (langage des mathématiciens) :

$$c_i = m_i + k_i \text{ modulo } 2 \qquad (2.03)$$

Le destinataire qui connaît la clé (la suite aléatoire) n'a plus qu'à la combiner de la même manière au cryptogramme reçu pour obtenir le message clair car :

$$c_i + k_i \equiv m_i + k_i + k_i \equiv m_i \text{ modulo } 2 \qquad (2.04)$$

La clé $k_0\, k_1 \ldots k_N$ est évidemment employée une seule fois et la connaissance du message clair et du message chiffré correspondant ne peut être d'aucune utilité pour décrypter un message futur (qui sera chiffré avec une nouvelle clé aléatoire). Il est impossible à mettre en œuvre dans les besoins actuels en informatique, car il suppose l'échange préalable d'une quantité d'information aussi grande que celle que l'on veut chiffrer.

2.3.2.2 Après avoir évoqué la notion de secret parfait et donné une illustration de ce concept, venons-en à l'autre concept de sécurité. Un algorithme est dit « informatiquement » (computationally en anglais) sûr si le temps de calcul nécessaire pour trouver la clé est hors de portée avec les moyens de calculs disponibles. Ce type de considération donne naissance à une hiérarchie dans la sécurité. On peut concevoir des algorithmes qui résisteraient à une capacité de calcul, mais pas à une autre. Des considérations économiques apparaissent aussi ; le cryptanalyste n'a pas intérêt à faire l'acquisition de moyens de calculs d'un coût supérieur au gain escompté. Comme dans le cas de la sécurité inconditionnelle, on a donné des fondations rigoureuses à ces notions ; il s'agit de la théorie de la complexité. Cette théorie construit des classes de problèmes en établissant des modèles abstraits de machines informatiques; elle fait appel à la logique mathématique.

Il y a également des approches quantitatives aux problèmes de complexité visant, par exemple, à donner des majorations (absolues ou le plus souvent asymptotiques) du temps ou de la place mémoire nécessaires à la mise en œuvre des algorithmes. Le plus souvent, ces méthodes font appel à l'analyse mathématique.

2.4 Eléments mathématiques appliqués à la cryptographie [11] [12]

2.4.1 Congruences

2.4.1.1 Définition

On dit que a est congru à b modulo n, si et seulement si $a = b + kn$ pour un certain entier k Si a > 0, b > 0 et b < n alors b est le reste de la division de a par n. La relation ci-dessous peut être encore noter $a \equiv b(\mathrm{mod}\, n)$.

On dit que b est le résidu a modulo n et l'on a $a \bmod n = b$.

L'ensemble des entiers de 0 à n-1 forme ce que l'on appelle l'ensemble de tous les résidus modulo n. Ce qui signifie que, le résidu de a modulo n est comprise entre 0 et n-1 bornes comprises.

2.4.1.2 Propriété

L'arithmétique modulo n est commutatif, associatif et distributif, d'où les relations :

$$(a+b)\bmod n = ((a \bmod n)+(b \bmod n))\bmod n \quad (2.05)$$

$$(a-b)\bmod n = ((a \bmod n)-(b \bmod n))\bmod n \quad (2.06)$$

$$(a\times b)\bmod n = ((a \bmod n)\times(b \bmod n))\bmod n \quad (2.07)$$

$$(a\times(b+c))\bmod n = (((a\times b)\bmod n)+((a\times c)\bmod n))\bmod n \quad (2.08)$$

2.4.1.3 Calcul des exponentielles en arithmétique modulo n

Il s'agit du calcul l'expression ayant la forme $a^x \bmod n$. Elever un nombre à une puissance entière modulo n se fait par une série de multiplications et divisions. Mais il existe des techniques pour effectuer cela efficacement. Une de ces techniques consiste à minimiser le nombre de multiplications modulo n.

Par exemple pour calculer $a^8 \bmod n$, effectuer 7 multiplications suivies d'une réduction module n seraient coûteuses en ressource mémoire. Il est avantageux d'effectuer 3 multiplications plus petites ainsi que 3 réductions modulo n également plus petites :

$$a^8 \bmod n = \left((a^2 \bmod n)^2 \bmod n\right)^2 \bmod n \quad (2.09)$$

Calculer $a^x \bmod n$ quand x n'est pas une puissance de 2 n'est qu'un petit peu difficile, exemple calculons $a^{25} \bmod n$ pour l'illustration.

$$a^{25} \bmod n = \left(a\times a^8\times a^{16}\right)\bmod n$$

$$= \left(a\times\left((a^2)^2\right)^2\times\left(\left((a^2)^2\right)^2\right)^2\right)\bmod n$$

$$a^{25} \bmod n = \left(\left(\left((a^2\times a)^2\right)^2\right)^2\times a\right)\bmod n \quad (2.10)$$

Par un réarrangement on obtient une expression qui ne contient que 6 multiplications à faire :

$$a^{25} \bmod n = \left(\left(\left(\left(\left(\left(a^2 \bmod n \right) \times a \right) \bmod n \right)^2 \bmod n \right)^2 \bmod \right)^2 \bmod n \right) \times a \bmod n \qquad (2.11)$$

Cette technique de calcul s'appelle la sommation en chaîne. Elle utilise une suite d'additions simples et évidentes qui est basée sur la représentation binaire.

2.4.2 Nombres premiers

Un nombre premier un nombre entier, strictement plus grand que un, dont les seuls facteurs sont 1 et lui-même. Le nombre 2 est un nombre premier, ainsi que 73, 2521, 2365347734339, et 2756839-1. Un nombre qui n'est pas premier est dit composé. Il existe une infinité de nombres premiers. En cryptographie, on utilise souvent de très grands nombres premiers (de plus de 512 bits ou plus)

2.4.3 Plus grand commun diviseur (PGCD)

Deux nombres sont premiers entre eux quand ils n'ont d'autre facteur en commun que 1. Autrement dit, si le plus grand commun diviseur de a et n est égal à 1, alors a et n sont dits premier entre eux, d'où la relation :

$$PGCD(a, n) = 1 \qquad (2.12)$$

2.4.4 Inverses modulo n

Le problème général consiste à chercher x tel que :

$$1 = (a \times x) \bmod n \qquad (2.13)$$

Ce qui est équivalent à écrire :

$$a^{-1} \equiv x \pmod n \qquad (2.14)$$

Si a et n sont premier entre eux, alors l'équation $a^{-1} \equiv x \pmod n$ a une solution unique x. Sinon elle n'aura pas de solution. Mais comment calcule-t-on l'inverse de a modulo n. Il existe plusieurs méthodes pour le calculer, l'algorithme d'Euclide, qui est un algorithme simple et très pratique en est une.

2.4.5 Fonction d'Euler

2.4.5.1 Petit théorème de Fermat

Si m est premier, et *a* n'est pas multiple de *m*, alors le petit théorème de Fermat indique que : $a^{m-1} \equiv 1(\mathrm{mod}\, n)$ (2.15)

2.4.5.2 Définition de la fonction d'Euler

La fonction d'Euler (appelée aussi l'indicateur d'Euler) est le nombre d'éléments de l'ensemble restreint des résidus modulo n. En d'autres termes, c'est le nombre d'entiers positifs plus petits que n et premier par rapport à n. Cette fonction est notée : $\varphi(n)$

Si n est premier, alors $\varphi(n) = n - 1$ (2.16)

Si $n = p \times q$, p et q sont premiers, alors

$$\varphi(n) = (p-1)(q-1) \quad (2.17)$$

D'après le petit théorème de Fermat, si n est premier et $PGCD(a,n) = 1$, alors $a^{n-1} \equiv 1(\mathrm{mod}\, n)$. Ce qui est équivalent à écrire : $a^{n-1} \bmod n = 1$. Et on en déduit

$$a^{\varphi(n)} \bmod n = 1 \quad (2.18)$$

Il est facile de calculer $x = a^{-1} \bmod n$ et l'on aura :

$$x = a^{\varphi(n)-1} \bmod n \quad (2.19)$$

On peut généraliser les deux méthodes de calcul d'inverse pour calculer x dans l'équation :

$$(a \times x) \bmod n = b \quad (2.20)$$

Par la généralisation d'Euler, on a :

$$x = \left(b \times a^{\varphi(n)-1}\right) \bmod n \quad (2.21)$$

Par la généralisation d'Euclide, on a :

$$x = \left(b \times \left(a^{-1} \bmod n\right)\right) \bmod n \tag{2.22}$$

En général, l'algorithme d'Euclide est une méthode plus rapide.

2.4.6 Théorème du reste chinois

Si on connaît la décomposition en facteurs premiers du nombre n, alors on peut utiliser le théorème du reste chinois pour résoudre un système d'équation particulier. La version de base du théorème a été découverte au premier siècle de notre ère par le mathématicien chinois SUN TSE.

En général, si la décomposition en facteur premier de n est

$$n = p_1 \times p_2 \times .... \times p_t \tag{2.23}$$

Alors le système d'équation :

$$\left(x \bmod p_i\right) = a_i \text{ , pour } i = 1,2,.....,t \tag{2.24}$$

admet une solution unique x, telle que x soit inférieur à n. Certains nombres premiers peuvent apparaître plusieurs fois dans le produit. Par exemple, p_1 peut être égal à p_2. Ainsi, pour a et b quelconques tels que $a < p$ et $b < q$ (p et q premiers), il existe un seul x tel que x soit inférieur à $p \times q$ et tel que : $x \equiv a(\bmod p)$ et (2.25)

$$x \equiv b(\bmod q) \tag{2.26}$$

Pour calculer ce x, on utilise l'algorithme d'Euclide pour calculer u tel que :

$$u \times q \equiv 1(\bmod p) \tag{2.27}$$

Ensuite on calcule

$$x = \left(\left((a-b) \times u\right) \bmod p\right) \times q + b \tag{2.28}$$

Un corollaire du théorème du reste chinois peut être utilisé pour résoudre un problème similaire : si p et q sont premiers et p est inférieur à q, alors il existe un x unique inférieur à $p \times q$ tel que :

$$a \equiv x(\mathrm{mod}\ p) \text{ et} \tag{2.29}$$

$$b \equiv x(\mathrm{mod}\, q) \tag{2.30}$$

Si $a \geq b \bmod p$, alors : $$x = (((a - (b \bmod p)) \times u) \bmod p) \times q + b \tag{2.31}$$

Si $a < b \bmod p$, alors : $$x = (((a + p - (b \bmod p)) \times u) \bmod p) \times q + b \tag{2.32}$$

2.4.7 Résidus quadratiques

Si p est premier et a est inférieur à p, alors a est un résidu quadratique modulo p si :

$$x^2 \equiv a(\mathrm{mod}\ p) \text{ , pour un certain x} \tag{2.33}$$

Une valeur quelconque de a ne satisfait pas forcement cette propriété. Par exemple, si $p = 7$, ses résidus quadratiques sont 1, 2 et 4 :

$$1^2 = 1 \equiv 1(\mathrm{mod}\, 7)$$

$$2^2 = 4 \equiv 4(\mathrm{mod}\, 7)$$

$$3^2 = 9 \equiv 2(\mathrm{mod}\, 7)$$

$$4^2 = 16 \equiv 2(\mathrm{mod}\, 7)$$

$$5^2 = 25 \equiv 4(\mathrm{mod}\, 7)$$

$$6^2 = 36 \equiv 1(\mathrm{mod}\, 7)$$

Remarquons que chaque résidu quadratique apparaît deux fois dans la liste. Il n'existe pas de valeur de x qui satisfait le système :

$$x^2 \equiv 3(\mathrm{mod}\, 7)$$

$$x^2 \equiv 5(\mathrm{mod}\, 7)$$

$$x^2 \equiv 6(\mathrm{mod}\, 7)$$

Ce qui signifie que les nombres 3, 5 et 6 ne sont pas des résidus quadratiques modulo 7.

2.4.8 Symbole de Legendre

Le symbole de Legendre, noté $L(a, p)$, est défini quand a est un entier quelconque et p est un nombre premier plus grand que 2. Il vaut 0,1 et –1 :

$L(a, p) = 0$ si a est divisible par p

$L(a, p) = 1$ si a est un résidu quadratique modulo p

$L(a, p) = -1$ si a n'est un résidu quadratique modulo p

Un moyen simple de calculer $L(a, p)$ est donné par :

$$L(a, p) = a^{(p-1)/2} \bmod p \tag{2.34}$$

Un autre moyen de calculer $L(a, p)$ est donné par les formules récursives suivantes :

Si $a = 1$, alors $L(a, p) = 1$; (2.35)

Si a est pair, alors : $L(a, p) = L(a/2, p) \times (-1)^{(p^2-1)/8}$ (2.36)

Si a est impair, alors : $L(a, p) = L(p \bmod a, a) \times (-1)^{(a-1)\times(p-1)/4}$ (2.37)

2.4.9 Symbole de Jacobi

Le symbole de Jacobi, noté $J(a,n)$ est une généralisation du symbole de Legendre, seulement il est défini pour toute paire d'entier a et n. Cette fonction est utilisée dans les tests de primalité. Le symbole de Jacobi est une fonction de l'ensemble restreint des résidus des facteurs de n. Il peut être calculé au moyen de plusieurs formules et en voici une qui repose sur les définitions suivantes :

- $J(a,n)$ n'est défini que pour n impair

$$J(0,n) = 0 \tag{2.38}$$

- Si n est premier et divise a, alors

$$J(a,n) = 0 \tag{2.39}$$

- Si n est premier et si a est résidu quadratique modulo n, alors

$$J(a,n)=1 \tag{2.40}$$

- Si n est premier et si a n'est pas résidu quadratique modulo n, alors

$$J(a,n)=-1 \tag{2.41}$$

- Si n est un nombre composé, alors

$$J(a,n)=J(a,p_1)\times J(a,p_2)\times .....\times J(a,p_m) \tag{2.42}$$

où $p_1,.....,p_m$ est la décomposition en facteurs premiers de n.

Il existe des règles qui permettent de calculer récursivement le symbole de Jacobi :

$$J(1,n)=1 \tag{2.43}$$

$$J(a\times b,n)=J(a,n)\times J(b,n) \tag{2.44}$$

$J(2,n)=1$ si $(n^2-1)/8$ est pair, -1 sinon (2.45)

$$J(a,n)=J((a \bmod n),n) \tag{2.46}$$

$$J(a,p_1\times p_2)=J(a,p_1)\times J(a,p_2) \tag{2.47}$$

Si $PGCD(a,b)=1$ et a et b sont tous deux impairs (la loi de réciprocité n'est pas valable pour les entiers pairs) :

$J(a,b)=+J(b,a)$ si $(a-1)(b-1)/4$ est pair (2.48)

$J(a,b)=-J(b,a)$ si $(a-1)(b-1)/4$ est impair (2.49)

2.4.10 Génération de nombres premiers

La cryptographie à clé publique a besoin de nombres premiers. La méthode consiste à générer des nombres aléatoires et de tester s'ils sont premiers ou non. Il existe plusieurs tests probabilistes de primalités, ce sont des tests qui permettent de déterminer si un nombre est premier avec un certain niveau de confiance. En supposant que ce « niveau de confiance » est suffisant, ce type de tests et assez bon.

2.4.10.1 Solovay-Strassen

SOLOVAY et STRASSEN ont développé un algorithme probabiliste de test de primalité. Cet algorithme utilise le symbole de Jacobi pour tester si p est premier :

- Choisissez un nombre aléatoire *a* inférieur à *p*
- Si le $PGCD(a, p) \neq 1$, alors *p* échoue au test
- Calculer $j = a^{(p-1)/2} \bmod p$
- Calculer le symbole de Jacobi $J(a, p)$

  Si $j \neq J(a, p)$, alors *p* n'est certainement pas premier

  Si $j = J(a, p)$, alors la probabilité que *p* ne soit pas premier est au plus de 50%.

Un nombre *a* qui permet d'affirmer que *p* n'est certainement pas premier s'appelle *un témoin*. Si *p* est composé, il y a au moins 50% de chances pour qu'un nombre *a* soit un témoin. Répétons ce test *t* fois avec *t* valeurs différentes aléatoires pour *a*. Un nombre composé à une chance sur 2^t de passer les *t* tests.

2.4.10.2 Lehmann

Un autre test, plus simple, a été développé indépendamment par LEHMANN. Voici l'algorithme pour tester si t est premier :

- Choisissez un nombre aléatoire a inférieur à p
- Calculer $j = a^{(p-1)/2} \bmod p$
- Si $a^{(p-1)/2} \cong 1$ ou $-1 (\bmod p)$ pour tous les i, alors p n'est pas premier
- Si $a^{(p-1)/2} \equiv 1$ ou $-1 (\bmod p)$, alors p la probabilité que p soit composé est de 50% au plus

A nouveau, un nombre aléatoire a au moins 50% de chances d'être un témoin. On répète le test t fois. Si le calcul vaut 1 et –1, mais pas toujours 1, alors p est probablement premier avec un taux d'erreur de 1 sur 2^t.

2.4.10.3 Rabin-Miller

L'algorithme que tout le monde utilise a été développé par RABIN, en partir des idées de MILLER. On choisit d'abord un nombre aléatoire p à tester. Calculer b, où b est le nombre de fois

que 2 divise $p-1$ (c'est-à-dire que 2^b est la plus grande puissance de 2 qui divise $p-1$). Ensuite calculer m tel que : $p = 1 + 2^b m$ (2.50)

- Choisissez un nombre aléatoire a inférieur à p
- On pose $j = 0$ et $z = a^m \bmod p$
- Si $z = 1$ ou si $z = p-1$, alors p passe le test et peut être premier
- Si $j > 0$ et $z = 1$, alors p n'est pas premier
- On pose $j = j+1$. Si $j < b$ et $z \neq p-1$ on pose $z = z^2 \bmod p$ et on retourne à l'étape précédente.
- Si $z = p-1$, alors p passe le test et peut être premier
- Si $j = b$ et $z \neq p-1$, alors p n'est pas premier

Les chances d'un nombre composé de passer ce test décroissent plus vite qu'avec les algorithmes précédents. Trois quart des valeurs de a sont à coup sûr des témoins. Cela veut dire qu'un nombre composé passera le test moins d'une fois sur 4^t, où t est le nombre d'itérations. En réalité, ces estimations sont très pessimistes. Pour la plupart des nombres aléatoires, environs 99,9% des valeurs possibles de a sont des témoins.

2.4.11 Logarithmes discrets dans un corps fini

L'exponentiation modulaire est une autre fonction à sens unique fréquemment utilisée en cryptographie. Evaluer l'expression suivante est assez aisée :

$$a^x \bmod n \qquad (2.51)$$

Le problème inverse de l'exponentiation modulaire est celui de la recherche du logarithme discret d'un nombre. C'est un problème redoutable de trouver x tel que

$$a^x \equiv b \left(\bmod n\right) \qquad (2.52)$$

Par exemple : Si $3^x \bmod 17 = 15$, alors $x = 6$

Les algorithmes discrets n'ont pas tous des solutions (faut-il rappeler que les solutions doivent être des entières). Il est facile de voir qu'il n'y a pas de solution x pour l'équation :

$$3^x \bmod 13 = 7$$

Il est encore plus difficile de résoudre ce problème pour les nombres de 1024 bits.

CHAPITRE 3 : LES ALGORITHMES CRYPTOGRAPHIQUES

3.1 Introduction

En fait, les algorithmes cryptographiques sont des fonctions mathématiques utilisées pour le chiffrement et le déchiffrement. Si M est le texte en clair, alors le texte chiffré est noté C. La fonction de chiffrement, notée E, transforme M en C. Ce qui s'écrit, mathématiquement:

$$E(M) = C \tag{3.01}$$

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

$$D(C) = M \tag{3.02}$$

Comme le but de toutes ces opérations n'est rien d'autre que de retrouver le message en clair à partir de la version chiffré de ce message, donc l'identité suivante doit être vérifiée :

$$D(E(M)) = M \tag{3.03}$$

3.2 Les algorithmes cryptographiques à clé unique [11] [12] [13] [14]

La cryptographie moderne utilise une clé, notée k qui peut prendre une valeur parmi un grand nombre de valeurs possibles. Les opérations de chiffrement et déchiffrement utilisent, tous les deux, cette clé. Ainsi les fonctions s'écrivent :

$C = E_k(M)$ pour le chiffrement, et (3.04)

$M = D_k(C)$ pour le déchiffrement. (3.05)

Dans les deux fonctions :

C : désigne le texte chiffré

M : désigne le texte en clair

E_k : représente la fonction qui permet de chiffrer le texte en clair en utilisant la clé k

D_k : représente la fonction qui permet de déchiffrer le texte en clair en utilisant la clé k.

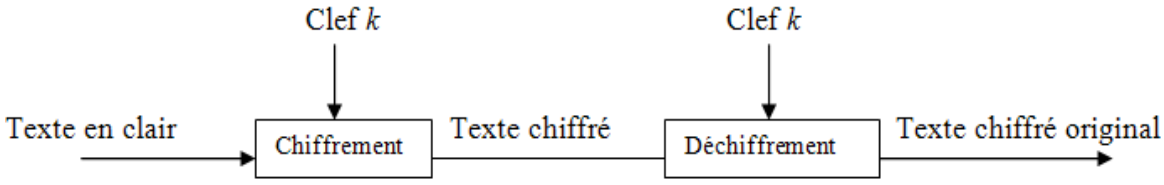

Figure 3.01: Cryptosystème avec une clé unique

En générale la fonction D_k n'est autre que l'inverse de la fonction E_k.

3.2.1 Algorithme de cryptage par bloc

L'algorithme de cryptage par bloc est fondé sur les principes de confusion et de diffusion de SHANNON.

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 compliquer que même ces outils cryptanalytiques puissants ne marcheront pas.

La diffusion répand l'influence d'un bit particulier du texte en clair ou de la clé aussi loin que possible dans le texte chiffré. Ceci camoufle aussi les relations statistiques et rend la cryptanalyse plus difficile.

3.2.1.1 Algorithme de chiffrement produit

C'est une astuce qui consiste à mélanger d'une manière répétitive la confusion et la diffusion avec un seul algorithme de chiffrement dans différentes combinaisons. C'est ce qu'on appelle *algorithme de chiffrement produit.*

3.2.1.2 Le réseau substitution-permutation

L'algorithme constitué par des couches de substitution et permutation est parfois appelé réseau substitution-permutation ou encore réseau SP.

3.2.1.3 Les réseaux de Feistel

C'est un algorithme de chiffrement par blocs itératif où la i^e rond est déterminée d'après la sortie de la ronde précédente. Donc pour un bloc de longueur n (pair) divisé en deux, on aura deux moitiés de longueur n/2, l'une appelé L et l'autre nommé R. D'où les relations suivantes :

$$L_i = R_{i-1} \tag{3.06}$$

$$R_i = L_{i-1} \oplus f\left(R_{i-1}, K_i\right) \text{ où} \tag{3.07}$$

Ki est la sous-clé utilisée dans le i^{e} ronde

f est une fonction quelconque.

Ceci est un concept important car il est garanti que la fonction (de chiffrement) est inversible. Comme la moitié droite et la sortie de la fonction de ronde sont combinées par ou exclusif, l'égalité suivante est forcement vérifiée :

$$R_i \oplus f\left(R_{i-1}, K_i\right) = L_{i-1} \oplus f\left(R_{i-1}, K_i\right) \oplus f\left(R_{i-1}, K_i\right) = L_{i-1} \tag{3.08}$$

La nature de f n'est pas importante ; f n'a pas besoin d'être inversible. Donc on peut concevoir une fonction f aussi compliquée que désiré, et la réalisation de deux algorithmes n'est plus nécessaire. En fait le réseau de Feistel prend tout cela en compte d'une manière automatique.

Pour la suite on va décrire et étudier le système DES (Data Encryption Standard) qui n'est autre que le système standard en matière d'algorithme de cryptage par bloc. Après nous allons faire la description de la méthode IDEA (International Data Encryption Algorithm), le prétendant pour remplacer le système DES qui est en train de vieillir et qui n'est plus fiable en matière de sécurité. Et enfin, nous allons parler de l'algorithme AES (Advanced Encryption Standard).

3.2.1.4 Le système DES (Data Encryption Standard)

3.2.1.4.1 Description générale du DES

Le DES est système de chiffrement par bloc de 64 bits. Il chiffre le texte en clair par bloc de 64 bits. Le chiffrement et le déchiffrement utilisent tous les deux le même algorithme (avec les différences au niveau de la génération des clés). La longueur de la clé est de 56 bits. Généralement, la clé est exprimée comme un nombre de 64 bits mais un bit sur huit utilisé comme un bit de contrôle de parité est ignoré. La clé peut être n'importe quel nombre de 56 bits et peut être changée à tout moment. Toute la sécurité réside dans la clé.

L'algorithme n'est rien d'autre que la combinaison de deux techniques de base de chiffrement par bloc : confusion et diffusion. Le DES a 16 rondes, c'est-à-dire qu'il applique 16

fois la même combinaison de techniques au bloc de texte en clair. L'algorithme n'utilise que des opérations arithmétiques et logiques standards sur des blocs de 64 bits.

3.2.1.4.2 Plan générale de l'algorithme

Le DES manipule le texte en clair par bloc de 64 bits. Après une permutation initiale, le bloc est coupé en une partie droite et une partie gauche, chacune d'une longueur de 32 bits. Après cela, il y a16 rondes d'opérations identiques, appelées « fonction f », qui consiste à combiner les données et la clé. Après la 16e ronde, les parties droite et gauche sont rassemblées et une permutation finale (l'inverse de la permutation initiale) termine l'algorithme.

A chaque ronde, les bits de la clé sont décalés et 48 bits sont alors sélectionnés, par une permutation compressive, parmi les 56 bits de la clé. La parité droite des données est étendue à 48 bits par une permutation expansive, puis elle est combinée avec 48 bits de la clé décalée par un ou exclusif. Le résultat est soumis à une opération de substitution qui le ramène à 32 bits. Et de nouveau il subit une simple permutation. La fonction f est constituée de ces quatre opérations. La sortie de la fonction f est alors combinée avec la moitié gauche par une ou exclusif. Le résultat de ces opérations devient la nouvelle moitié droite ; l'ancienne moitié de droite devient la nouvelle moitié gauche. Ces opérations sont répétées 16 fois, donnant le DES à 16 rondes.

Si le bloc Bi est le résultat de la ie itération, Li et Ri sont respectivement les moitiés gauche et droite de Bi, Ki est la clé de 48 bits pour la ie ronde, et f est la fonction qui fait toutes les substitutions, permutations et ou exclusif avec la clé, alors une ronde est décrit par :

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

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

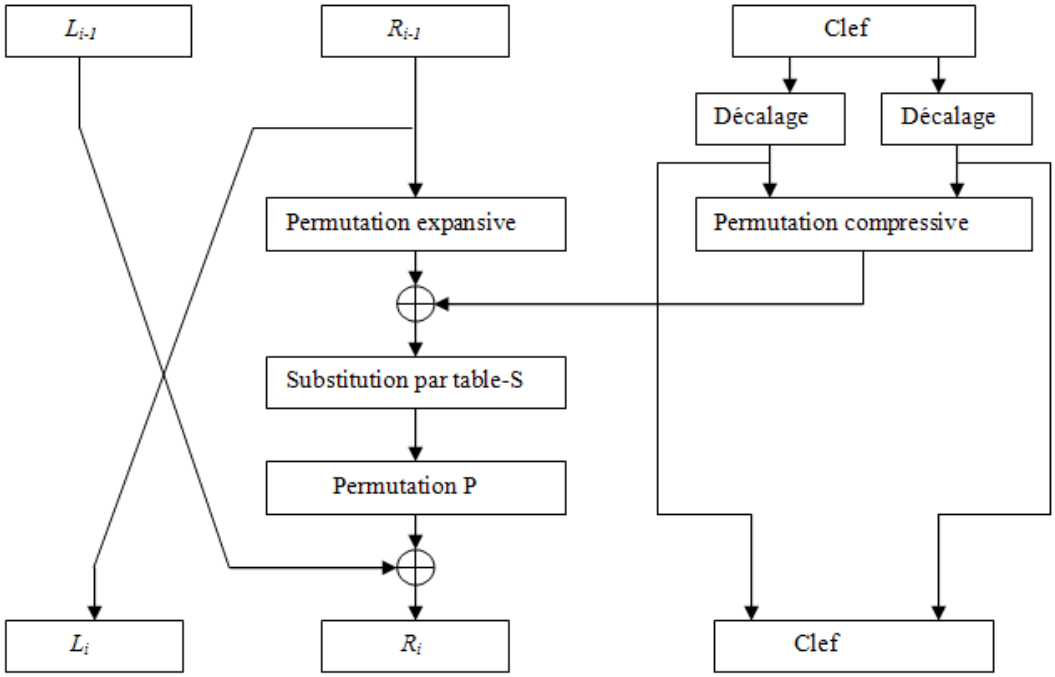

Figure 3.02 : Une ronde du DES

3.2.1.4.3 La permutation initiale

C'est une permutation qui transpose le bloc de 64 bits d'entrée avant la première ronde. Comme cette permutation bit à bit est difficile à réalisée en logiciel, de nombreuses réalisations logicielles du DES n'inclut pas les permutations initiale et finale.

3.2.1.4.4 Le plan de génération de clé

Les 64 bits de la clé DES sont réduits à 56 bits en ignorant un bit sur huit. Ces bits peuvent être utilisés comme bits de contrôle de parité pour vérifier qu'il n'y a pas eu d'erreur lors de la saisie de la clé.

Après que la clé de 56 bits a été extraite, une clé de 48 bits est engendrée pour chaque ronde du DES. Ces clés Ki, sont déterminées de la manière suivante :

La clé de 56 bits est divisée en deux moitiés de 28 bits.

Les moitiés sont décalées vers la gauche d'une ou deux positions, en fonction de la ronde. Les décalages ont pour effet de différencier les sous-ensembles de bits utilisés dans chacune des sous-clés. Voici le tableau qui donne le nombre de décalage en fonction de la ronde.

Ronde	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Nombre de décalages	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tableau 3.01 : Décalage de clé par ronde

48 bits parmi 56 bits sont sélectionnés. Cette opération est appelée permutation compressive ou encore choix permuté car elle combine une permutation des bits avec une sélection d'un sous-ensemble des bits.

3.2.1.4.5 La permutation expansive

C'est une opération qui étend la moitié droite des données, Ri, de 32 bits à 48 bits. Cette opération est appelée permutation expansive car elle change l'ordre des bits, et répète certains bits. Elle a deux buts :

Pour avoir un résultat de même taille que la clé pour l'opération ou exclusif.

Pour fournir un résultat plus long qui pourra être comprimé pendant l'opération de substitution. En permettant à un bit d'affecter deux substitutions, la dépendance entre les bits d'entrée et les bits de sortie se dilue plus vite. C'est ce qu'on appelle l'effet avalanche. Le DES est conçu afin de parvenir le plus vite possible à ce que chaque bit du texte chiffré dépende de chaque bit du texte en clair et de chaque clé.

3.2.1.4.6 La substitution par table-S

Après que la clé comprimée a été combinée par ou exclusif avec le bloc expansé, le résultat de 48 bits est soumis à une opération de substitution. Ces opérations sont réalisées à l'aide de huit tables de substitution ou table-S. Chaque table-S a six entrées et quatre sorties, et il y a huit tables différentes. Les 48 bits sont divisés en bloc de 6 bits. Chaque bloc est manipulé séparément par une table-S différente : le bloc 1 est manipulé par la table-S1, le bloc 2 est manipulé par la table-S2, etc.…

Chaque table-S a 4 rangs et 16 colonnes. Chaque cellule dans la table est un nombre de 4 bits. Les 6 bits du bloc d'entrée spécifient le rang et la colonne à exploiter pour obtenir la sortie. Les bits d'entrée spécifient une cellule de la table-S d'une manière particulière :

Considérons 6 bits d'entrée étiquetés b_1, b_2, b_3, b_4, b_5, b_6 ; les bits b1 et b6 sont combinés pour former un nombre de 2 bits, entre 0 et 3, qui correspond à un rang dans la table ; les 4 bits du milieu, b_2 à b_5, sont combinés pour former un nombre de 4 bits, entre 0 et 15, qui correspond à une colonne de la table.

Exemple : Supposons que l'entrée du sixième table-S (c'est-à-dire les bits 31 à 36 en sortie ou exclusif) soit 110010_2. Le premier et le dernier bit combinés donnent 10_2, indique qu'il faut prendre le 2^e rang de la table. Les 4 bits du milieu combinés forment 1001_2, indiquent qu'il faut prendre la 9^e colonne de la table. La cellule au 2^e rang, 9^e colonne de la table-S6 est 0.Donc la valeur 0000_2 remplace 110010_2 à la sortie.

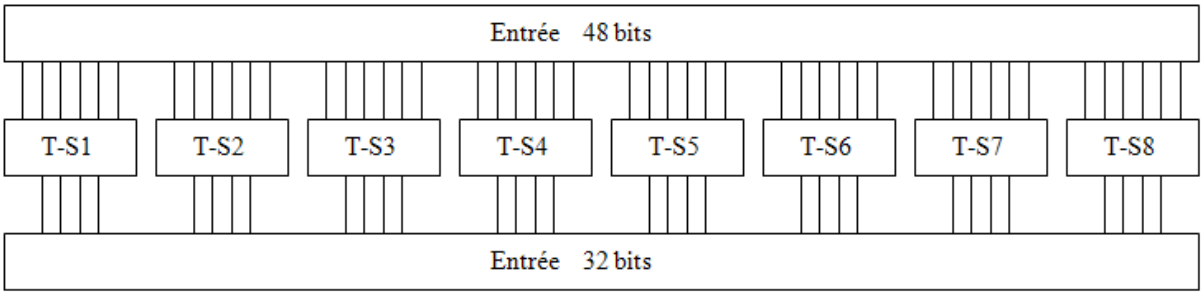

Figure 3.03 : Table-S de substitution

Cette phase de substitution donne 8 blocs de 4bits qui sont recombinés pour former un seul bloc de 32 bits. Ce même bloc qui va passer à l'étape suivante.

3.2.1.4.7 La permutation-P

Les 32 bits de sortie des substitutions par table-S sont permutés à l'aide d'une table-P. Cette table met en correspondance chaque bit de l'entrée avec un bit de sortie ; aucun n'est utilisé deux fois et aucun bit n'est ignoré. On parle de permutation pure

Finalement, le résultat de la permutation-P est combiné par ou exclusif avec la moitié gauche du bloc de 64 bits initial. Ensuite, les parties gauche et droite sont échangées est une nouvelle ronde commence.

3.2.1.4.8 La permutation finale

La permutation finale est l'inverse de la permutation initiale.

3.2.1.4.9 Déchiffrement du DES

Les différentes opérations ont été choisies pour offrir une propriété très utile : le même algorithme est utilisé pour le chiffrement et le déchiffrement. La seule différence est que les clés doivent être utilisées dans l'ordre inverse. Si la clé de chiffrement de chaque ronde sont K_1, K_2, K_3, .., K_{16}, alors les clés de déchiffrement sont respectivement K_{16}, K_{15}, K_1. L'algorithme qui engendre les clés pour chaque ronde est également circulaire, et le nombre de décalages(vers la droite) à effectuer devient 0,1,2,2,2,2,2,2,2,1,2,2,2,2,2,2,1.

3.2.1.4.10 Modes opératoires du DES

Les quatre modes opératoires ECB, CBC, OFB et CFB sont applicables au système DES. Par exemple le standard bancaire de l'ANSI spécifie ECB et CBC pour le chiffrement, et les modes CBC et CFB à n bits pour l'authentification.

A cause de sa simplicité le mode ECB est le plus utilisé dans des logiciels commerciaux classiques bien qu'il soit plus vulnérable aux attaques. Le mode de chiffrement avec chaînage de blocs (CBC) est utilisé occasionnellement, bien qu'il ne soit guère plus compliqué que le ECB et qu'il procure un niveau de sécurité nettement plus élevé.

3.2.1.4.11 Question de sécurité

Le DES est un système qui a existé déjà depuis plusieurs années et si répandu qu'il est naïf de croire que les agences comme le NSA et ses homologues n'ont pas construit une machine spécialement dédiée pour le casser. Le DES deviendra de moins en moins sûr avec le temps qui passe.

3.2.1.5 Le système IDEA

Le système IDEA est conçu par Xuejia LAI et James MASSEY vers les années 1990. C'est un système fondé sur des bases théoriques impressionnantes, et l'algorithme à l'aire solide.

IDEA est un algorithme de chiffrement par blocs ; il manipule des blocs de texte en clair de 64 bits. La clé est longue de 128 bits. L'algorithme de chiffrement et déchiffrement est le même.

Comme tous les algorithmes de chiffrement par bloc, IDEA utilise à la fois la confusion et la diffusion. Les opérations constituant l'IDEA est facilement réalisable à la fois en logiciel qu'en matériel :

- Ou exclusif
- Addition modulo 2^{16}
- Multiplication modulo $2^{16}+1$(cette opération est un peu la table-S de l'IDEA).

Toutes ces opérations manipulent des sous-blocs de 16 bits. Cet algorithme est efficace même sur des processeurs 16 bits.

3.2.1.5.1 Description du système IDEA

Le bloc de 64 bits de données est divisé en 4 sous-blocs de 16 bits ; X_1, X_2, X_3 et X_4. Ces 4 sous-blocs deviennent les entrées de la première ronde de l'algorithme et il y a 8 rondes au total. A chaque ronde, les 4 sous-blocs sont combinés par ou exclusif, additionnés, multipliés entre eux et avec les 6 sous-blocs de 16 bits dérivés de la clé. Entre chaque ronde, le deuxième et le troisième sous-bloc sont échangés. Enfin les 4 sous-blocs sont combinés avec les 4 sous-clés dans une transformation finale.

A chaque ronde, les procédures sont les suivantes :

(a) Multiplication de X1 et de la première sous-clé ;

(b) Addition de X2 et la deuxième sous-clé ;

(c) Addition de X3 et la troisième sous-clé ;

(d) Multiplication X4 et la quatrième sous-clé ;

(e) Combinaison par OU EXCLUSIF les résultats des étapes (a) et (c) ;

(f) Combinaison par OU EXCLUSIF les résultats des étapes (b) et (d) ;

(g) Multiplication le résultat de l'étape (e) avec la cinquième sous-clé ;

(h) Addition des résultats des étapes (f) et (g) ;

(i) Multiplication du résultat de l'étape (h) par la sixième sous-clé ;

(j) Addition des résultats des étapes (g) et (i) ;

(k) Combinaison par OU EXCLUSIF des résultats des étapes (a) et (i) ;

(l) Combinaison par OU EXCLUSIF des résultats des étapes (c) et (i) ;

(m) Combinaison par OU EXCLUSIF des résultats des étapes (b) et (i) ;

(n) Combinaison par OU EXCLUSIF des résultats des étapes (d) et (i) ;

La sortie de la ronde est constituée des 4 sous-blocs produits par les étapes (k), (m.), (l) et (n). Echanger les deux blocs intérieurs (sauf la dernière ronde) et cela donne l'entrée de la ronde suivante. Après la huitième ronde, il y a une transformation finale :

Multiplication de X1 et la première sous-clé ;

Addition de X2 et la deuxième sous-clé ;

Addition de X3 et la troisième sous-clé ;

Multiplication de X4 et la quatrième sous-clé ;

Enfin, les 4 sous-blocs sont rassemblés pour former le texte chiffré. La création des sous-clés est aussi facile. L'algorithme en utilise 52 (6 pour chacune des 8 rondes et 4 pour la transformation finale). D'abord, la clé de 128 bits est divisée en 8 sous-clés de 16 bits. Ce sont les 8 premières sous-clés pour l'algorithme (Les 6 de la première ronde et les 2 premières de la deuxième ronde). Ensuite la clé est décalée circulairement de 25 bits vers la gauche et à nouveau divisée en 8 sous-clé. Les 4 premières sont utilisées lors de la deuxième ronde et 4 autres lors de la troisième ronde. La clé est à nouveau décalée circulairement de 25 bits vers la gauche pour les sous-clés suivantes, et ainsi de suite jusqu'à la fin de l'algorithme.

Le déchiffrement est exactement le même, excepté que les sous-clés sont inversées et légèrement différentes. Les sous-clés de déchiffrement sont inversées par rapport à l'addition ou par rapport à la multiplication des sous-clés de chiffrement. Calculer ces inverses prend du temps

mais vous ne devez le faire qu'une fois par clé de déchiffrement. Voici la représentation de l'algorithme :

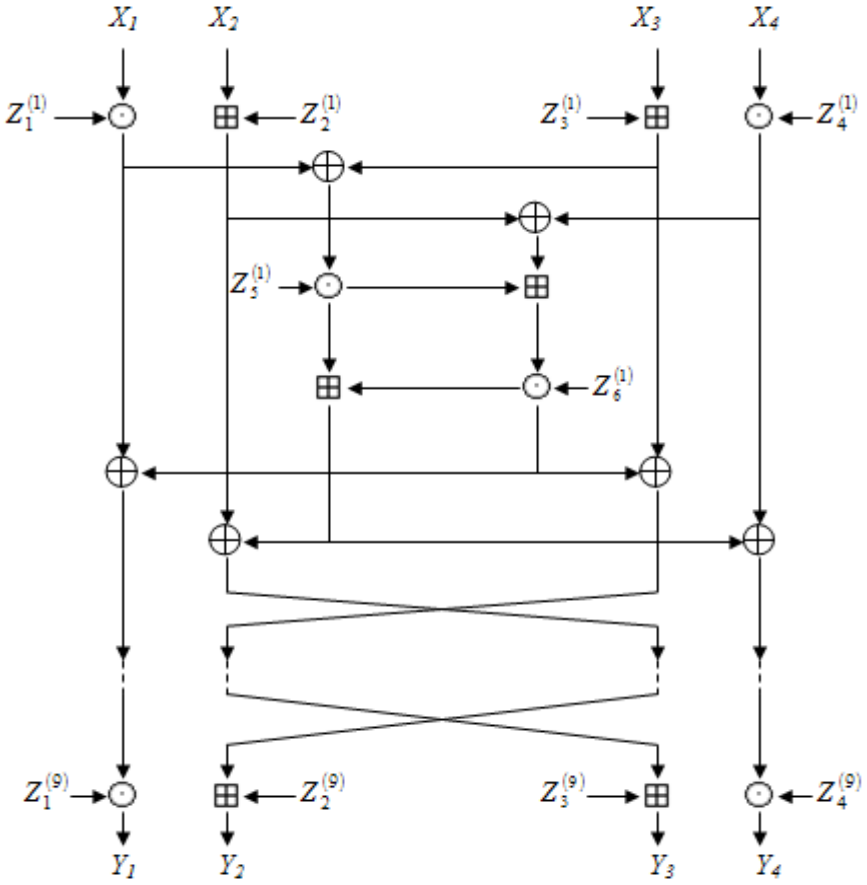

Figure 3.04 : Représentation du plan de l'algorithme

X_i : sous-bloc de 16 bits de texte en clair

Y_i : sous-bloc de 16 bits de texte chiffré

$Z_i^{(r)}$: Sous-bloc 16 bits de la clé

⊕ : ou exclusif bit à bit de sous-blocs de16 bits

⊞ : addition modulo 2^{16} d'entier de 16 bits

⊙ : multiplication modulo $2^{16}+1$ d'entiers de 16 bits

3.2.1.5.2 Modes opératoires d'IDEA

IDEA peut être utilisé avec n'importe quel mode opératoire de chiffrement.

3.2.1.5.3 Sécurité du système IDEA

Plusieurs groupes académiques et militaires tentent de casser l'algorithme IDEA mais aucun d'entre eux n'a réussi jusqu'ici. Mais il faut quand même se méfier car des algorithmes qui paraissent sûrs sont tombés devant des nouvelles formes de cryptanalyse, il ne faut pas oublier aussi que l'IDEA est encore un nouveau algorithme basé sur une théorie ferme.

Même si l'IDEA paraît nettement plus sûr que le DES, il n'est pas toujours facile de remplacer l'un par l'autre dans une application existence. Si la base de données et vos écrans d'interrogation sont conçus pour accepter une clé de 64 bits, il pourrait être impossible de mettre la clé de 128 bits d'IDEA à la place.

3.2.1.6 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 (National Institute of Standard and Technology) 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 tous types d'attaques
- Rapidités du code sur la plus grande variété de plateforme possible
- Simplicité dans sa conception

La structure globale de l'AES à 128 bits de clé est décrite à la figure 3.05. Dans l'AES, le bloc comporte 128 bits. La clé peut comporter 128, 196, ou 256 bits, ce qui influent 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 maitre. La première itération est précédée par l'addition de la sous-clé numéro 0 qui correspond à la clé maître et la dernière itération ne comporte pas de *Mixcolums* ou fonction de brouillage de colonne. La fonction itérée se compose du quadruplet de fonctions (*SubBytes, ShiftRows, MixColums, 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 (dans la pratique, elle est mise en table) ; elle assure la résistance de l'algorithme aux attaques différentielles et linéaires.

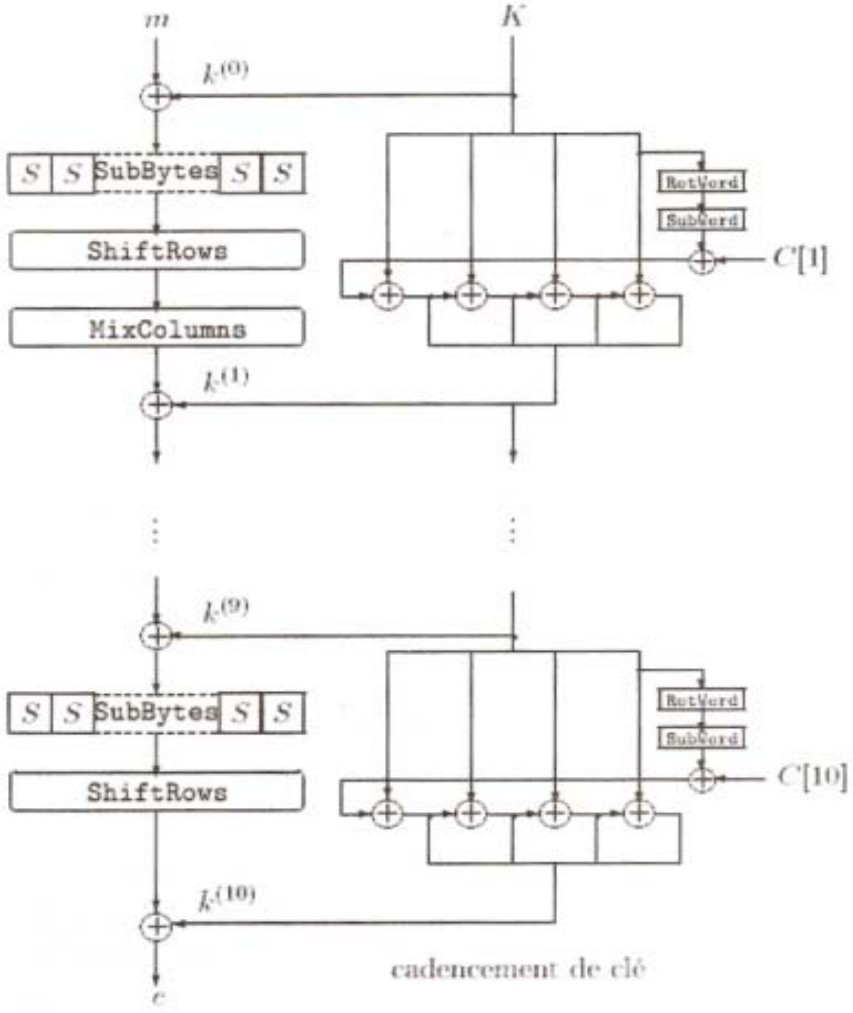

Figure 3.05 : L'AES

La phase de diffusion est composée des fonctions *ShiftRows* et *MixColums* 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 : la sous-clé numéro 0 correspond à la clé secrète ; ensuite, la sous-clé numéro *i* (utilisée à la ième itération) est obtenue à partir de la sous-clé numéro (*i-1*) grâce à l'algorithme décrit à la figure 3.05. 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 *SubWord* 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 4 octets ainsi obtenu et les 4 premiers octets de la sous-clé précédente. Les 3 autres blocs de 4 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.

3.2.1.6.1 Déchiffrement

Toutes les opérations réalisés lors d 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 (*ShiftRows*) sont inversés, c'est-à-dire vers la droite. La manipulation matricielle de l'opération de brouillage (*MixColums*) nécessite une inversion de la matrice. Une fois la matrice inversée obtenue, la manipulation est la même que l'opération de brouillage des colonnes.

3.2.2 Algorithme de chiffrement en continu

La plupart des conceptions pratiques des algorithmes de chiffrement en continu tournent autour des Registres à décalage à rétroaction linéaire ou RDRL, c'est pourquoi il est important d'étudier d'abord ce que c'est qu'un RDRL.

3.2.3 Registres à décalage à rétroaction linéaire

Un registre à décalage à rétroaction est composé de deux parties : un registre à décalage et une fonction de rétroaction.

Le registre à décalage est une suite de bits. Chaque fois qu'un bit est nécessaire, tous les bits du registre sont décalés vers la droite de un bit. Le nouveau bit le plus à gauche est calculé en

fonction des autres bits dans le registre. Le registre produit ainsi un bit en sortie, le moins significatif en général.

La période d'un registre à décalage est la longueur de séquence produit avant qu'elle ne commence à se répéter.

La figure 3.05 représente un registre à décalage. L'opérateur ou exclusif constitue la fonction de rétroaction du RDRL.

Un RDRL à n bits est dans l'un de ses $2^n - 1$ états. En théorie, il peut donc engendrer des suites pseudo-aléatoires de $2^n - 1$ bits de long avant de se répéter.

Les RDRL ayant les états internes passant par $2^n - 1$ états, sont appelés RDRL à période maximale et la suite des bits produite est alors appelée une m-suite.

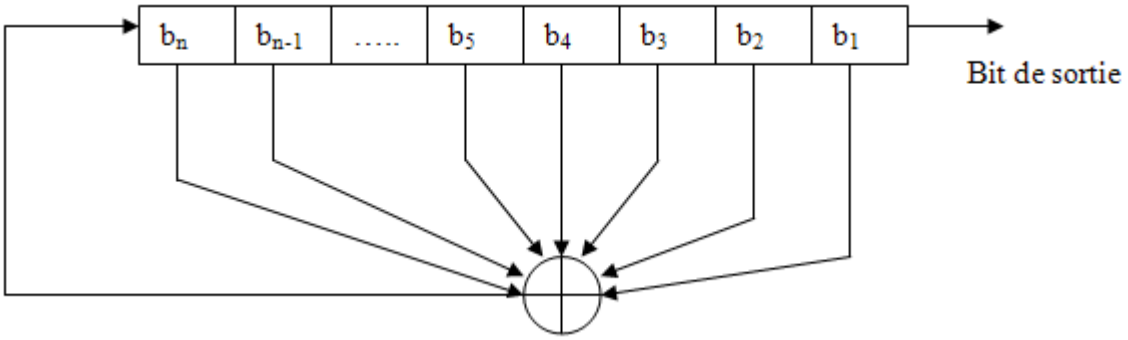

Figure 3.06 : Registre à décalage à rétroaction linéaire

Pour qu'un RDRL soit de période maximale, le polynôme formé par les éléments de la séquence de dérivation +1 doit être un polynôme primitif modulo 2. Le degré du polynôme est la longueur du registre à décalage

Un polynôme primitif de degré n est un polynôme irréductible qui divise $x^{2n-1} + 1$ mais pas $x^d + 1$ pour tout d qui divise $2^n - 1$.

En général, il n'existe pas de moyen de générer des polynômes primitifs modulo 2 de degré donné. Le moyen le plus facile consiste à choisir un polynôme aléatoirement et de vérifier s'il est primitif.

3.2.3.1 Chiffrement en continu à base de RDRL

L'approche classique de la conception d'un cryptosystème à base de RDRL est simple. On commence par prendre un ou plusieurs RDRL généralement de longueurs différentes et avec des polynômes primitifs distincts. La clé est l'état initial des RDRL. Chaque fois qu'un bit est nécessaire, le RDRL est décalé d'un cran.

3.2.3.2 Le système A5

L'algorithme de chiffrement en continu A5 sert à chiffrer les messages en téléphonie GSM. C'est le standard non américain pour les téléphones cellulaires numériques.

L'algorithme est utilisé pour chiffrer la communication entre le mobile et la borne reliée au réseau.

A5 est constitué de trois RDRL, de taille de registres 19, 22 et 23 ; tous les polynômes primitifs utilisés pour les rétroactions sont clairsemés, ce qui veut dire que la plupart de leurs coefficients sont nuls. Chaque registre est cadencé en fonction de son propre bit du milieu, combiné par ou exclusif avec l'inverse de la fonction seuil des bits du milieu des trois registres. Habituellement, deux des RDRL opèrent un cycle d'horloge à chaque ronde

Les idées fondamentales de A5 sont bonnes, car il est très efficace. Il réussit tous les tests statistiques connus ; sa seule faiblesse connue est la taille de ses registres qui sont assez petits pour permettre une recherche exhaustive.

3.2.3.3 Autre algorithme de chiffrement en continu : Le système RC4

C'est un algorithme de chiffrement en continu à clé de longueur variable développé en 1987 par Ron REVEST pour RSA DATA SECURITY, INC.

RC4 est simple à décrire. L'algorithme fonctionne en mode OFB : il a une table-S à 8*8 bits (S_0, S_1……, S_{255}). Les Si forment une permutation des nombres entre 0 et 255, et la permutation dépend de la clé de longueur variable. Il y a deux compteurs i et j, initialisés à zéro.

Pour générer un octet aléatoire, on procède ainsi :

- $i = (i + j) \bmod 256$ (3.09)

- $j = (j + S_i) \bmod 256$ (3.10)

  Echanger Si et Sj

  $t = (S_i + S_j) \bmod 256$ (3.11)

  $K = S_t$ (3.12)

L'octet K est combiné par ou exclusif avec le texte clair pour produire le texte chiffré ou bien avec le texte chiffré pour produire le texte clair. Le chiffrement est très rapide, environ 10 fois plus vite que le DES.

L'initialisation est facile aussi. On commence avec l'identité : $S_0 = 0, S_1 = 1, \ldots S_{255} = 255$. On remplit un autre tableau 256 octets avec la clé, en répétant celle-ci suffisamment de fois pour combler le tableau : K_0, K_1…, K_{255}.

On initialise l'index j à zéro et on effectue :

Pour i variant de 0 à 255, effectuer :

$j = (j + S_i + K_i) \bmod 256$ (3.13)

Echanger Si et Sj

RSADSI prétend que l'algorithme est immune aux attaques à texte en clair choisi. Il n'a pas de cycle court et il est hautement non linéaire. La table-S évolue lentement au cours de l'algorithme : i permet d'assurer que chaque élément change et j permet d'assurer que les éléments de l'algorithme changent aléatoirement. L'algorithme est simple que la plupart des programmeurs peuvent rapidement écrire le code de mémoire.

L'algorithme est aussi dans la spécification de téléphones modulaires « Cellular Digital Packet Data ».

3.3 Algorithme à clé publique : [11] [13] [15]

C'est in concept inventé par Whitfield DIFFIE, Martin HELLMAN et indépendamment par Ralph MERKLE. L'idée neuve dans le domaine était que les clés pouvaient être des paires, une clé de chiffrement et une clé de déchiffrement, et qu'il était impossible de générer une clé à partir de l'autre.

Nombreux algorithmes de cryptographie à clé publique ont été proposés. Certains ne sont adaptés qu'à la distribution de clés. D'autres ne sont adaptés qu'aux signatures numériques. Seuls trois algorithmes sont adéquats pour le chiffrement et les signatures numériques : RSA, ELGAMAL et RABIN. On va s'intéresser aux deux premiers car ce sont les algorithmes à clé publique les plus utilisés et les plus fiables.

Les algorithmes de chiffrement à clé publique utilisent deux clés différentes, l'une pour chiffrement et l'autre pour le déchiffrement. La clé de déchiffrement, notée k1, est différente de la clé de déchiffrement, notée k_2. Un tel cryptosystème est décrit par les relations suivantes :

$$E_{k_1}(M) = C \tag{3.14}$$

$$D_{k_2}(C) = M \tag{3.15}$$

$$D_{k_2}\left(E_{k_1}(M)\right) = M \tag{3.16}$$

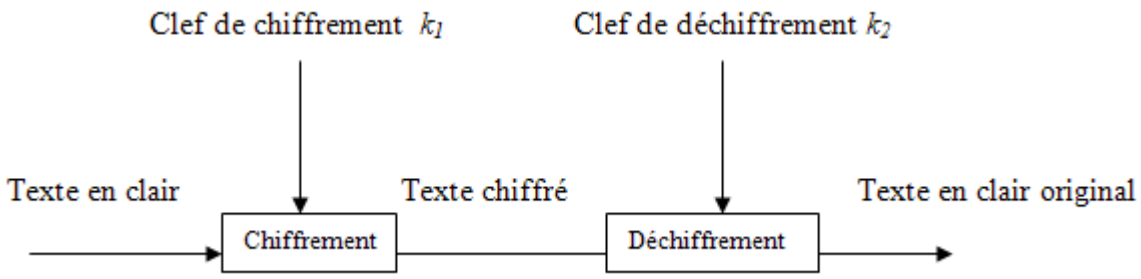

Figure 3.07: Cryptosystème à clé publique

Avec ces algorithmes, toute la sécurité réside dans la (ou les) clé(s), et dans les détails de l'algorithme. Ce qui fait que l'algorithme peut être publié et analysé. Peu importe qu'un espion connaisse l'algorithme, si elle ignore votre clé, elle ne pourra pas lire vos messages.

Les algorithmes de chiffrement à clé publique sont caractérisés par ses lenteurs. Les vitesses de chiffrement et de déchiffrement sont nettement plus faibles par rapport à celles des algorithmes à clé secrète.

3.3.1 Le système RSA

Le premier algorithme à clé publique de grandeur nature apparut peu après l'algorithme à empilement de MERKLE. Il fonctionne pour le chiffrement aussi bien que pour les signatures numériques.

Baptisé d'après les noms de ses inventeurs, Ron RIVEST, Adi SHAMIR, et Leonard ADLEMAN, il a résisté à des années de cryptanalyse intensive.

Le niveau de sécurité de RSA dépend de la difficulté de factoriser des grands nombres. Les clés publique et privée sont des fonctions d'une paire de grands nombres premiers (100 à 200 chiffres ou plus encore). Retrouver le texte en clair à partir d'une des clés et du texte chiffré est supposé équivalent à la factorisation du produit des deux nombres premiers.

Pour engendrer les deux clés, il faut deux grands nombres premiers, p et q. Calculer le produit :

$$n = p \times q \tag{3.25}$$

Choisir aléatoirement une clé de chiffrement e telle que e et $(p-1)(q-1)$ soient premiers entre eux. Après utiliser l'algorithme d'Euclide pour calculer la clé de déchiffrement d de telle manière que :

$$ed \equiv 1(\mathrm{mod}(p-1)(p-1)) \tag{3.26}$$

Autrement dit :

$$d \equiv e^{-1} \bmod((p-1)(q-1)) \tag{3.27}$$

n et d sont aussi premiers entre eux. Les nombre e et n constituent la clé publique et le nombre d n'est autre que la clé privée. Les nombres p et q ne sont plus nécessaires. Ils peuvent être jetés.

Pour chiffrer le message m, il faut le découper en blocs numériques tels que chaque bloc ait une représentation unique modulo n (avec les données binaires, choisir la plus grande puissance de 2 inférieurs à n). Ainsi, si p et q sont tous les deux des nombres premiers de 100 chiffres, alors n aura tout juste moins de 200 chiffres et chaque bloc de message mi doit avoir juste moins de 200 chiffres. Le message chiffré c sera formé, de manière similaire, par des blocs ci,

ayant à peu près la même longueur. Donc la formule de chiffrement est simplement :

$$c_i = m_i^e \bmod n \tag{3.28}$$

Pour déchiffrer un message chiffré, il suffit de prendre chaque bloc ci et calculer :

$$m_i = c_i^d \bmod n \tag{3.29}$$

En effet (sachant que toutes les opérations sont effectuées modulo n), on a

$$c_i^d = \left(m_i^e\right)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i \times m_i^{k(p-1)(q-1)} = m_i \times 1 = m_i \tag{3.30}$$

Donc on résume comme ci- dessous le système RSA

Clé publique : $n = p \times q$

où p et q sont deux nombres premiers qui doivent rester secrets

$$PGCD(e,(p-1)(q-1)) = 1, \tag{3.31}$$

e et $(p-1)(q-1)$ sont premier entre eux

Clé privée : $$d \equiv e^{-1} \bmod((p-1)(q-1)) \tag{3.32}$$

Chiffrement :

$$c = m^e \bmod n \tag{3.33}$$

Déchiffrement :

$$m = c^d \bmod n \tag{3.34}$$

Voici un petit exemple pour éclaircir tout cela. Si $p = 47$ et $q = 71$ alors :

$$n = p \times q = 3337 \tag{3.35}$$

La clé de chiffrement e ne doit pas avoir de facteur commun avec :

$$(p-1)(q-1) = 46 = 3220 \tag{3.36}$$

Choisissez e (aléatoirement) égal à 79. Dans ce cas :

$$d = 79^{-1} \bmod 3220 = 1019 \tag{3.37}$$

Ce nombre est obtenu par l'algorithme d'Euclide étendu. Publiez e et n, gardez secret d. Jetez p et q. Pour chiffrer le message :

$$m = 6882326879666683 \tag{3.38}$$

Divisons-le d'abord en petits groupe de trois chiffres, on obtient ainsi 6 blocs mi.

$$m_1 = 688\,,\ m_2 = 232\,,\ m_3 = 687\,,\ m_4 = 966\,,\ m_5 = 668\,,\ m_6 = 3 \tag{3.39}$$

Le premier bloc est chiffré par :

$$688^{79} \bmod 3337 = 1570 = c_1 \tag{3.40}$$

En effectuant la même opération pour tous les blocs, on obtient le message chiffré :

$$\text{c} = 1570\ 2756\ 2091\ 2276\ 2423\ 158 \tag{3.41}$$

Pour déchiffrer le message, il faut effectuer les même exponentiations mais en utilisant la clé de déchiffrement 1019. Donc :

$$1570^{1019} \bmod 3337 = 688 = m_1 \tag{3.42}$$

Le reste du message est obtenu de la même manière.

3.3.1.1 Sécurité de RSA

La sécurité de RSA dépend du problème de la factorisation de grands nombres. Mais il n'a jamais été prouvé mathématiquement qu'on doive factoriser n pour calculer m et e. Il est concevable qu'un tout autre moyen de cryptanalyser RSA soit découvert. Toutefois si cette nouvelle voie permettait aux cryptanalystes de retrouver d, cela pourrait aussi être utilisé comme une nouvelle voie pour la factorisation de grands nombres. Factoriser n est l'attaque la plus évidente. Si les adversaires ont à leur possession la clé publique e et le module n, alors pour trouver la clé de déchiffrement d, ils doivent factoriser n. Actuellement, la technologie de factorisation la mieux avancée, n'arrive à factoriser que 129 chiffres. Ainsi n doit être supérieur à cela. L'attaque exhaustive est encore moins efficace. Donc il n'y a pas à se tracasser à ce propos.

Il faut se rappeler qu'il ne suffit pas d'avoir un algorithme cryptographique sûr. Le système entier n'est pas sûr s'il y a moindre faille dans les trois zones suivantes, à savoir :

- L'algorithme cryptographique
- Le protocole cryptographique
- Le cryptosystème

3.3.2 Chiffrement ElGamal

Pour engendrer une paire de clés, il faut choisir d'abord un nombre premier p et deux nombres aléatoires g et x, tel que g et x soient tous deux inférieurs à p.

On calcule ensuite $y = g^x \bmod p$ (3.43)

La clé publique est faite de y, g et p. Les valeurs de g et p peuvent être toutes deux partagées par un groupe d'utilisateur. La clé privée est x.

Pour chiffrer un message M, choisir d'abord un nombre aléatoire k tel que k et $p-1$ soient premiers entre eux. Ensuite calculer :

$$a = g^k \bmod p \quad (3.44)$$

$$b = y^k M \bmod p \quad (3.45)$$

La paire a et b forme le texte chiffré. Donc la taille du texte chiffré est le double de celle du texte en clair.

Pour déchiffrer a et b, on calcule :

$$M = b / a^x \bmod p \quad (3.46)$$

En effet on a : $a^x \equiv g^{kx} \pmod p$ (3.47)

et $b / a^x \equiv y^k M / a^x \pmod p$ (3.48)

or $y^k \equiv g^{kx} \pmod p$ (3.49)

donc $b / a^x \equiv g^{kx} / M g^{kx} \pmod p$ (3.50)

d'où $b / a^x \equiv M \pmod p$ (3.51)

c'est-à-dire $M = b / a^x \bmod p$

On peut résumer comme suit le chiffrement ElGamal

Clé publique :

p premier (peut être partagé par un groupe d'utilisateurs)

$g < p$ (Peut être partagé par un groupe d'utilisateurs)

$$y = g^x \bmod p \tag{3.52}$$

Clé privée : $x < p$

Chiffrement :

k choisi aléatoirement et premier avec $p - 1$

$$a = g^k \bmod p \text{ (Texte chiffré)} \tag{3.53}$$

$$b = y^k M \bmod p \text{ (Texte chiffré)} \tag{3.54}$$

Déchiffrement :

$M = b / a^x \bmod p$ (Texte en clair)

3.3.2.1 Sécurité de ElGamal

L'algorithme ElGamal est incassable comme RSA si les conditions et critères d'utilisation sont respectés, surtout le protocole d'utilisation d'un algorithme à clé publique.

3.3.3 Algorithmes d'échange de clés

3.3.3.1 Diffie-Hellman

DIFFIE-HELLMAN est le premier algorithme à clé publique qui fut inventé. Sa sécurité dépend de la difficulté de calculer des algorithmes discrets sur un corps fini par rapport à la facilité de calcul d'exponentielles dans le même corps. DIFFIE-HELLMAN peut être utilisé pour la distribution des clés mais il ne peut pas être utilisé pour chiffrer et déchiffrer des messages. Alice et Bernard peuvent utiliser cet algorithme pour engendrer une clé secrète. Les mathématiques sont simples. Au départ, Alice et Bernard se mettent d'accord sur deux grands

nombres entiers, n et g, de telle manière que g soit primitif par rapport à n. Ces deux entiers n'ont pas à être secrets ; Alice et Bernard peuvent convenir de ces nombres sur un canal non sûr. Ils peuvent même être communs à un groupe d'utilisateurs. Peu importe.

Le protocole se déroule ainsi :

Alice choisit un grand nombre entier aléatoire x et envoie à Bernard le résultat du calcul :

$$X = g^x \bmod n \tag{3.55}$$

Bernard choisit un grand nombre entier aléatoire y et envoie à Alice le résultat du calcul :

$$Y = g^y \bmod n \tag{3.56}$$

Alice calcule : $k = Y^x \bmod n$ (3.57)

Bernard calcul : $k' = X^y \bmod n$ (3.58)

Les valeurs k et k' sont toutes deux égales à $g^{xy} \bmod n$. Personne ne peut, en écoutant la communication calculer cette valeur ; celui qui écoute ne connaît que n, g, X et Y. Ainsi k est la clé secrète qu'Alice et Bernard ont calculée indépendamment.

3.3.3.2 Diffie-Hellman avec trois participants et plus

Le protocole d'échange de clés DIFFIE-HELLMAN peut être facilement étendu à trois personnes et plus. Alice, Bernard et Christine engendrent ensemble une clé secrète :

Alice choisit un grand nombre entier aléatoire x et envoie à Bernard le résultat du calcul : $X = g^x \bmod n$ (3.59)

Bernard choisit un grand nombre entier aléatoire y et envoie à Christine le résultat du calcul : $Y = g^y \bmod n$ (3.60)

Christine choisit un grand nombre entier aléatoire z et envoie à Alice le résultat du calcul :

$$Z = g^z \bmod n \tag{3.61}$$

Alice envoie $Z' = Z^x \bmod n$ à Bernard (3.62)

Bernard envoie $X' = X^y \bmod n$ à Christine (3.63)

Christine envoie $Y' = Y^z \bmod n$ à Alice (3.64)

Alice calcul : $k = Y'^x \bmod n$ (3.65)

Bernard calcul : $k = Z'^y \bmod n$ (3.66)

Christine calcul : $k = X'^z \bmod n$ (3.67)

La clé secrète, k, est égale à $g^{xyz} \bmod n$, quiconque d'autre qui écoute la communication ne peut calculer cette valeur. Le protocole peut être aisément étendu à quatre personne et plus. Il suffit d'ajouter les calculs des participants supplémentaires et les rondes de calcul additionnelles.

3.3.3.3 Echange de clés chiffrées

Le protocole d'échange de clés chiffrées (EKE pour « Encrypted Key Exchange ») a été conçu par Steve BELLOVIN et Michael MERRITT. Il offre la confidentialité et l'authentification sur un réseau d'ordinateurs, en utilisant à la fois la cryptographie à clé secrète (unique) et la cryptographie à clé publique de façon originale : une clé secrète partagée est utilisée pour chiffrer une clé publique engendrée aléatoirement.

Alice et Bernard partagent un mot de passe commun P. En utilisant le protocole suivant, ils peuvent s'authentifier l'un à l'autre et engendrer une clé commune de session qu'on va noter K.

Alice engendre une clé publique aléatoire, K'. Elle chiffre celle-ci à l'aide d'un algorithme à clé sécrète et de la clé P : $E_P(K')$. Elle envoie à Bernard :

$$Alice, E_P(K')$$

Bernard connaît P. Il déchiffre le message pour obtenir K'. Ensuite, il engendre une clé de session aléatoire, K, et la chiffre avec la clé publique qu'il vient de recevoir d'Alice et la clé secrète P : $E_P(E_{K'}(K))$

Il envoie le résultat à Alice

Alice déchiffre le message pour obtenir K. Elle engendre une chaîne aléatoire RA et la chiffre avec K et envoie le résultat à Bernard : $E_K(R_A)$

Bernard déchiffre le message pour obtenir RA. Il engendre une autre chaîne RB, chiffre les deux chaînes avec K et envoie le résultat à Alice : $E_K(R_A, R_B)$

Alice déchiffre le message pour obtenir RA et RB. En faisant l'hypothèse que la RA qu'elle a reçue de Bernard est la même que celle qu'elle a engendrée à l'étape 3. Elle chiffre RB avec K et l'envoie à Bernard : $E_K(R_B)$

Bernard déchiffre le message pour obtenir RB. En faisant l'hypothèse que la RB qu'il a reçue d'Alice est la même que celle qu'il a engendrée à l'étape 4. Le protocole est terminé. Les deux participants peuvent communiquer en utilisant la clé de session K.

La portion « défi-réponse » du protocole, de l'étape 3 et 6, valide le protocole tandis que les étapes 3 et 5 prouvent à Alice que Bernard connaît K et les étapes 4 et 6 prouvent à Bernard qu'Alice connaît K.

EKE peut être réalisé grâce à plusieurs algorithmes à clé publique comme RSA, ELGAMAL et DIFFIE-HELLMAN.

CHAPITRE 4 : AUTHENTIFICATION ET SIGNATURE NUMÉRIQUE

L'authentification donne au destinataire d'un message chiffré la possibilité d'identifier et de certifier l'origine du message, c'est-à-dire de l'expéditeur. L'authentification d'un message utilise le principe de la signature numérique. Le mécanisme de la clé publique fournit un moyen de générer une signature.

4.1 Algorithme de signature numérique à clé publique : [11] [13] [16]

En août 1991, L'Institut national des standards et de la technologie américaine NIST (National Institut of Standards and Technology) proposa l'algorithme de signature numérique DSA (Digital Signature Algorithm) pour le nouveau standard de signature numérique DSS (Digital Signature Standard). Malgré des protestations contre cette initiative, le DSA s'est imposé comme étant le standard en matière de signature numérique. En fait l'algorithme de DSA a été conçu par NIST avec la collaboration de NSA, ce qui explique l'inquiétude des sociétés à l'adopter comme un standard. Elles ont peur que le NSA ait mis une petite brèche secrète dans le DSS laquelle leur permettra de le casser s'il le veut. Beaucoup sont les sociétés qui ont préféré investir dans la réalisation de l'algorithme RSA.

4.1.1 Algorithme de signature numérique de DSA:

Data Signature Algorithm (ou DSA) est un algorithme standard d'authentification conçu par l'Agence de Sécurité Américaine (ou NSA : National Security Agency)

4.1.1.1 Fonction de hachage :

Une fonction de hachage à sens unique, $H(M)$ opère sur un message M de longueur arbitraire et fournit une valeur de hachage de longueur fixe h.

$h = H(M)$, où h est de longueur m.

Les caractéristiques des fonctions de hachage à sens unique sont :

Etant donné M, il est facile de calculer h

Etant donné h, il est difficile et impossible de calculer M

Etant donné *M*, il est difficile et quasiment impossible de trouver un autre message *M'* tel que $H(M) = H(M')$

L'algorithme sûr de hachage SHA, conçu par NIST et la NSA pour être utiliser dans le DSA, est fondé sur le concept de la fonction de hachage.

4.1.1.2 Description de SHA :

Au départ le message est complété de manière à ce que sa longueur en bits soit un multiple de 512. Ajouter d'abord un 1 suivi d'autant de 0 que nécessaire pour que 64 bits de plus amènent la longueur à un multiple de 512 bits. Ceci s'explique par le fait que la manipulation du texte se fait par bloc de 512 bits et la sortie de l'algorithme est un ensemble de 5 blocs de 32 bits qui, joints ensemble, forment une seule empreinte de 160 bits.

Cinq variables sont initialisées de la façon suivante :

$A = 0x67452301$

$B = 0xEFCDAB89$

$D = 0x10325476$

$E = 0xC3D2E1F0$

La boucle principale de l'algorithme commence alors. Elle traite le message 512 bits à la fois et continue tant qu'il y a des blocs de 512 bits.

Pour commencer, les 5 variables sont copiées dans des variables différentes : A dans AA, B dans BB, C dans CC, D dans DD et E dans EE.

La boucle principale a 4 rondes de 20 opérations chacune. Chaque opération est effectuée au moyen d'une fonction non linéaire de trois variables $f_t(X,Y,Z)$, des décalages et des additions.

L'ensemble des fonctions non linéaires de SHA est donné par :

$f_t(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$ pour *t* entre 0 et 19

$f_t(X,Y,Z) = X \oplus Y \oplus Z$ pour *t* entre 20 et 39

$f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$ pour *t* entre 40 et 59

$f_t(X,Y,Z) = X \oplus Y \oplus Z$ pour t entre 60 et 79

L'algorithme utilise également 4 constantes :

$K_t = 0x5A827999$, pour t entre 0 et 19

$K_t = 0x6ED9EBA1$,pour t entre 20 et 39

$K_t = 0x8F1BBCDC$,pour t entre 40 et 59

$K_t = 0xCA62C1D6$, pour t entre 60 et 79

Le bloc du message découpé en 16 mots de 32 bits est transformé en 80 mots de 32 bits en utilisant l'algorithme suivant

$W_t = M_t$, pour t = 0 à 15

$W_t = W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}$, pour t = 16 à 79 (4.01)

Si t est le numéro de l'opération (1 à 80), M_j représente le j^e sous-bloc du message (j allant de 0 à15), et que $\lhd s$ représente un décalage circulaire à gauche de s bits, alors la boucle principale ressemble à :

$$TEMP = (A \lhd 5) + f_t(B,C,D) + E + W_t + K_t \quad (4.02)$$

$$E = D \quad (4.03)$$

$$D = C \quad (4.04)$$

$$C = (B \lhd 30) \quad (4.05)$$

$$B = A \quad (4.06)$$

$$A = TEMP \quad (4.07)$$

Après, A, B, C, D et E sont ajoutés à AA, BB, CC, DD et EE respectivement et l'algorithme passe au bloc de données suivant. La sortie est la juxtaposition de AA, BB, CC, DD et EE.

4.1.1.3 Description de l'algorithme:

L'algorithme utilise les paramètres suivants :

p= un nombre premier de L bits de long, où $512 < L < 1024$ et L est multiple de 64

q= un facteur premier de p-1, long de 160 bits.

$$g = h^{(p-1)/q} \bmod p \text{ , où} \tag{4.08}$$

h est n'importe quel nombre inférieur à p-1 tel que $h^{(p-1)/q} \bmod p$ soit plus grand que 1.

x= un nombre inférieur è q.

$$y = g^x \bmod p \tag{4.09}$$

De plus, l'algorithme utilise une fonction de hachage à sens unique $H(x)$. Les trois premiers paramètres p, q et g sont publiques et peuvent être communs à un réseau d'utilisateurs. La clé publique est y.

Pour signer un message M :

Alice engendre un nombre aléatoire k inférieur à q

Alice engendre

$$r = \left(g^k \bmod p\right) \bmod q \tag{4.10}$$

$$s = \left(k^{-1}\left(H(M) + xr\right)\right) \bmod q \tag{4.11}$$

Les paramètres r et s forment sa signature ; elle les envoie à Bernard

Bernard vérifie la signature en calculant :

$$w = s^{-1} \bmod q \tag{4.12}$$

$$u_1 = \left(H(M) \times w\right) \bmod q \tag{4.13}$$

$$u_2 = rw \bmod q \tag{4.14}$$

$$v = \left(\left(g^{u_1} \times y^{u_2}\right) \bmod p\right) \bmod q \tag{4.15}$$

Si $v = r$, alors la signature est vérifier

D'où le résumé

Clé publique :

p nombre premier compris entre 512 et 1024 bits

q facteur premier de p-1 de 160 bits

$$g = h^{(p-1)/q} \bmod p \text{ , où} \tag{4.16}$$

h est inférieur à p-1 et $h^{(p-1)/q} \bmod p > 1$

(p, q et g peuvent chacun être partagés par un groupe d'utilisateurs)

$$y = g^x \bmod p \tag{4.17}$$

Clé privée :

$x < q$ (un nombre de 160 bits)

Signature :

k choisi aléatoirement et inférieur à q

$$r = \left(g^k \bmod p\right) \bmod q \text{ (signature)} \tag{4.18}$$

$$s = \left(k^{-1}\left(H(M) + xr\right)\right) \bmod q \text{ (signature)} \tag{4.19}$$

Vérification :

$$w = s^{-1} \bmod q \tag{4.20}$$

$$u_1 = \left(H(M) \times w\right) \bmod q \tag{4.21}$$

$$u_2 = rw \bmod q \tag{4.22}$$

$$v = \left(\left(g^{u_1} \times y^{u_2}\right) \bmod p\right) \bmod q \quad * \tag{4.23}$$

Si $v = r$, alors la signature est vérifier

4.1.1.4 Génération des nombres premiers pour le DSA

Le NIST qui n'est autre que le concepteur, recommande la méthode suivante pour générer les deux nombres premiers p et q tels q divise p-1. Le nombre p a une longueur de 512 à 1024 bits et multiple de 64. Le nombre q a 160 bits de long.

Soit $L - 1 = 160n + b$ où L est la longueur de p, n et b sont deux nombres, et b est inférieur à 160.

Choisissez une chaîne d'au moins 160 bits et appelez-la S. Soit g la longueur de S en bits

Calculez $U = H(S) \oplus H\left((S+1) \bmod 2^g\right)$, *(4.24)*

où H est l'algorithme de hachage sûr

Formez q à partir de U en mettant à 1 le bit le plus significatif et le bit le moins significatif.

Vérifier si q est un nombre premier

Si q n'est pas un nombre premier, retournez à l'étape 1.

Posez C=0 et N=2.

Pour $k = 0,1,....,n$, on a :

$$V_k = H\left((S + N + k) \bmod 2^g\right) \tag{4.25}$$

Calculer

$$W = V_0 + V_1 \times 2^{160} + ... + V_{n-1} \times 2^{(n-1)\times 160} + \left(V_n \bmod 2^b\right) \times 2^{n\times 160} \tag{4.26}$$

et

$$X = W + 2^{L-1} \tag{4.27}$$

Remarquez que X est un nombre de L bits.

Calculez

$$p = X - \left((X \bmod 2q) - 1\right) \tag{4.28}$$

Si $p < 2^{L-1}$, alors allez à l'étape 13

Vérifiez si p est premier.

Si p est premier, allez à l'étape 15.

Posez

$C = C + 1$

$N = N + n + 1$

Si $C = 4096$, alors allez à l'étape 1., sinon allez à l'étape 7.

Sauvegardez les valeurs de S et C utilisées pour engendrer p et q.

4.1.1.5 Sécurité du DSA :

Avec 512 bits, le DSA n'était pas assez solide pour la sécurité à long terme. Avec 1024 bits, il l'est. Les chercheurs de NSA ont affirmé que les probabilités que quiconque, y compris la NSA eux même, de falsifier une signature avec le DSA, quand il est correctement utiliser et réaliser sont infinitésimales. De plus la présumée brèche secrète est vraie pour n'importe quel système d'authentification à clé public, y compris le RSA.

4.1.2 Algorithme de signatures ElGamal

Pour signer un message *M,* il faut d'abord choisir un nombre aléatoire *k*, tel que *k* et *p*-1 soit premiers entre eux. Ensuite il faut calculer :

$$a = g^k \bmod p \tag{4.29}$$

et utiliser l'algorithme d'Euclide étendu pour calculer la valeur de *b* qui satisfait l'équation suivante :

$$M = (xa + kb) \bmod (p-1) \tag{4.30}$$

La signature est la paire : *a* et *b*. La valeur aléatoire *k* doit être tenue secrète.

Pour vérifier une signature, il faut confirmer que :

$$y^a a^b \bmod p = g^M \bmod p \tag{4.31}$$

On peut résumer ainsi la procédure de signature avec l'algorithme ElGamal

Clé publique :

p premier (peut être partager par un groupe d'utilisateurs)

$g < p$ (peut être partager par un groupe d'utilisateurs)

$$y = g^x \bmod p \tag{4.32}$$

Clé privée :

$x < p$

Signature :

k choisi aléatoirement et premier avec *p*-1

$$a(signature) = g^k \bmod p$$

$b(signature)$ tel que

$$M = (xa + kb) \bmod (p-1) \tag{4.33}$$

Vérification :

La signature est valide si

$$y^a a^b \bmod p = g^M \bmod p \tag{4.34}$$

Il faut prendre une nouvelle valeur de *k* à chaque signature et cette valeur doit être choisie aléatoirement.

Pour illustration voici un exemple : choisissons $p = 11$ et $g = 2$. Choisissons la clé privée $x = 8$. Calculons : $y = g^x \bmod p = 2^8 \bmod 11 = 11$

La clé publique est faite de $y = 3, g = 2$ et $p = 11$.

Pour authentifier $M = 5$, choisissons un nombre aléatoire $k = 9$. Il faut vérifier que $PGCD(9,10) = 1$, calculons : $a = g^k \bmod p = 2^9 \bmod 11 = 6$

et utilisons l'algorithme d'Euclide étendu pour résoudre :

$$M = (ax + kb) \bmod (p-1) \tag{4.35}$$

$$5 = (8 \times 6 + 9 \times b) \bmod 10$$

La solution est $b = 3$ et la signature est la paire $a = 6$ et $b = 3$

Pour vérifier la signature, confirmons que :

$$y^a a^b \bmod p = g^M \bmod p$$

$$3^6 6^3 \bmod 11 = 2^5 \bmod 11 = 10$$

CHAPITRE 5 : CRYPTAGE COMBINE AVEC LES ALGORITHMES RSA/AES

Dans ce chapitre, nous allons réaliser un cryptage combiné avec les algorithmes RSA/AES. Pour le réaliser, nous avons utilisés JBuilder 9 pour le développement. Ce système possède trois types d'application : la partie application a cryptographie publique RSA utilisant uniquement sa propre clé publique et sa clé privée générée aléatoirement, une autre application utilisant uniquement la clé privée AES, et enfin l'application qui consiste à combiner ces deux types d'algorithmes c'est-à-dire a combiner la clé publique RSA et la clé privée AES avec en plus un algorithmes d'authentification SHA1.

5.1 Choix du langage de programmation

Nous avons choisi JAVA comme langage de programmation. En effet, nous avons étudiés au sein du département télécommunication durant trois ans. Ce choix s'explique comme suit :

- JAVA est un langage orienté objet
- Il est portable sur la plupart des plateformes (Windows, Linux, Solaris,…)
- C'est un langage généraliste ayant un vaste champ d'application (réseau, cryptographie, base de données, calcul scientifique,…). Il permet ainsi de développer des applications professionnelles de grandes tailles.
- Il intègre une interface graphique de haut niveau
- Il est plus sûr, car de nombreuses vérifications sont faites, aussi bien à la compilation qu'à l'exécution, pour limiter le nombre et la gravité des erreurs.
- Il existe de nombreuses bibliothèques de programmes dans des domaines très variés, de sorte que ce langage devient un langage professionnel de premier plan.
- Le code produit (il s'agit d'un pseudo-code ou byte-code) est indépendant de la plate-forme utilisée.

Même si JAVA présente tous ces avantages (cette liste est non exhaustive), il présente certains inconvénients. Il faut remarquer une certaine lenteur à l'exécution.

Pour mettre en œuvre la programmation JAVA, nous avons utilisés JBuilder 9, qui est un environnement de développement graphique IDE (Interface development Environment). JBuilder aide le programmeur au développement du code source, plus précisément minimise les codes écrits.

5.2 Choix de l'algorithme de chiffrement

Le choix de l'algorithme de chiffrement est le plus difficile des choix. Mais nous avons réussis par trouver ce qui est mieux pour notre réalisation. Tout d'abord, l'algorithme doit être un algorithme sûr et qui a fait ses preuves durant des années. Ensuite, il doit être facile à comprendre et à programmer (pas un logiciel fini). Et enfin, il doit avoir une capacité de chiffrement très avancé. Comme notre système est conçu pour Madagascar, l'utilisation de l'algorithme ne devra pas poser un problème d'acheminement de clé, d'où le besoin d'un algorithme à clé publique.

L'algorithme à clé publique RSA présente tous ces besoins, avec une existence de presque 30 ans ; et depuis, une résistance à la cryptanalyse très remarquable. En augmentant la taille de la clé RSA, elle devient difficile à casser, mais le chiffrement devient un peu long. En choisissant une taille de clé raisonnable, cet algorithme présente encore une barrière infranchissable pour certaine technologie.

L'algorithme à clé privée AES est l'algorithme standard de chiffrement actuel. Elle a été choisie à partir d'un concours organisé par le gouvernement Américain et élue à l'unanimité comme standard de chiffrement depuis 2000. Et aujourd'hui, elle résiste encore à différents types d'attaques. C'est pourquoi on a choisi dans notre réalisation de combiner ces deux types d'algorithmes pour le chiffrement de données.

Dans notre réalisation, pour l'algorithme AES, nous avons choisi une clé de longueur variable 128 bits à 432 bits ; et pour l'algorithme RSA, nous avons choisi une clé de longueur variant entre 1024 bits à 4096 bits. Ce qui explique que notre système est suffisamment sécurisé et un temps de chiffrement et de déchiffrement acceptable.

5.3 Conception et réalisation [2] [4] [8] [17]

Comme nous avons dits au début de ce chapitre, JAVA permet de programmer différents types d'application. Pour programmer l'algorithme RSA, JAVA nous offre deux méthodes

différentes. Soit en programmant directement en utilisant les formules qu'on a vues au chapitre 2, en utilisant le paquetage java.math.BigInteger, java.security, soit en utilisant le paquetage JCE (pour Java Cryptography Extensions). Ce dernier regroupe des algorithmes cryptographiques prêts à être employés.

5.3.1 Description du JCE

Le paquetage JCE fait partie intégrante du JDK standard version 1.4, s'il a été fourni comme extension auparavant.

5.3.2 La classe Provider

Il est possible de choisir n'importe quel fournisseur de service de cryptographie. Mais nous somme contentés du fournisseur par défaut du JDK.

5.3.3 La clé d'encryptage

Le paquetage JCE procure la gestion de clé. Une clé est spécifiée par l'interface Key. On confectionne une clé en utilisant un KeyGenerator ou KeyPairGenerator.

```
kGen = KeyGenerator.getInstance(algorithme) ;

Key sk = kGen.generateKey() ;

ou

kGen = KeyPairGenerator.getInstance(algorithme) ;

KeyPair sk = kGen.generateKeyPair() ;
```

5.3.4 La classe Cipher

La classe javax.crypto.Cipher implémente le cryptage d'un texte. Un objet Cipher est obtenu par getInstance doit être initialisé dans un des deux modes (encodage ou décodage), qui sont définis comme des entiers constants finaux dans la classe Cipher. Les noms symboliques pour ces modes sont :

```
ENCRYPT_MODE, DECRYPT_MODE
```

Chaque méthode d'initialisation prend un paramètre mode (opmode) et initialise l'objet Cipher pour ce mode. D'autres paramètres incluent la clé (*key*), paramètres de l'algorithme (params) et source aléatoire (random).

Il faut noter que lorsqu'un objet Cipher est initialisé, il perd tous les états précédents. En d'autres termes, initialiser un objet *Cipher* équivaut à créer une nouvelle instance de Cipher.

Pour notre application, cous avons choisis la programmation directe par la formule. Voici un exemple simple de code qui nous a inspiré ;

```
import java.io.*;
import java.math.*;
import java.lang.String;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
public class Rsa {
        public Rsa() { }
                boolean bCrypt;
                int nByte;
                String sExt;
                boolean bChkSum;
                SecretKeySpec Key;
                 String sFileSrc,sFileDst;
                 byte bCryptTag[] = {-48,-59,105,60,37,-7,72,-126};
                 BigInteger pfMod;
                 BigInteger pfKey;
    public String CryptFile(String sFileSrcIn, String sFileDstIn,
        String sKeyFile, boolean bChkSumIn) {
```

```
//   Crypte / Décrypte un fichier
                File fileSrc=new File(sFileSrcIn);
                IsCrypted(fileSrc);
        if(sFileDstIn != sFileDst) sFileDst=sFileDstIn;
                File fileDst=new File(sFileDst);
            try {
                    int i, j, n;
                    String sMsg;
                    FileInputStream in = new FileInputStream(fileSrc);
                     FileOutputStreamout=new FileOutputStream(fileDst);
                // Chargement des clés
                sMsg = SetKeyFile(sKeyFile);
                if(sMsg.length() > 0) return sMsg;
/////////////////////////// ENCRYPT - DECRYPT
int iBlockSize = 32;
if(bCrypt) {        // decrypt
   byte[] bOut = new byte[iBlockSize];
   byte[] b4 = new byte[4];
   byte[] bIn = new byte[40960];
   int nBlock = nByte / iBlockSize;
   int iReste = nByte - (nBlock * iBlockSize);
   int iCountBlock = 0;
   while (true) {
      i = in.read(b4);
      if(i == -1) break;
      j = BytesToInt(b4);
```

```
        i = in.read(bIn, 0, j);
        byte[] bIn1 = new byte[j];
        for(i=0; i<j; i++) bIn1[i] = bIn[i];
        bOut = Decrypt(bIn1);
        if(bOut.length != iBlockSize) return "Fichier corrompu? Bonne clé?";
        iCountBlock++;
        if(iCountBlock > nBlock) out.write(bOut, 0, iReste);
        else out.write(bOut);
    }
} else {        // encrypt
    byte[] bIn = new byte[iBlockSize];
    byte[] b4;
    byte[] bOut;
    i = in.read(bIn);
    while (i != -1) {
        bOut = Encrypt(bIn);
        b4 = IntToBytes(bOut.length);
        out.write(b4);
        out.write(bOut);
        i = in.read(bIn);
    }
}
//////// VALIDATION
if(bChkSum && bCrypt) {
    out.close(); in.close();
    in = new FileInputStream(fileDst);
```

```
            i = in.read(b1);
            while (i != -1) {
               m.update(b1);
               i = in.read(b1);
            }
            byte[] br = m.doFinal();
            for(j=0; j<20; j++) { if(br[j] != br2[j]) break; }
            if(j < 20) bErr = true;
         }
         // Validité
         String sMsgChkSum = "";
         if(bErr) sMsgChkSum = "ERREUR D' AUTHENTIFICATION   ";
         else if(bChkSum && bCrypt) sMsgChkSum = "AUTHENTIFICATION OK   ";
         in.close();
         if(! (bChkSum && bCrypt)) out.close();
         return sMsgChkSum + sMsg;
         } catch(Exception e) {
         return e.getMessage();
      }
   }
   private String SetKeyFile(String sKeyFile) {
      File f=new File(sKeyFile);
      try {
         FileInputStream in = new FileInputStream(f);
         byte[] bLen = new byte[4];
         in.read(bLen);
```

```
            int iLen = BytesToInt(bLen);
            byte[] bP = new byte[iLen];
            in.read(bP);
            BigInteger p = new BigInteger(bP);
            in.read(bLen);
            iLen = BytesToInt(bLen);
            bP = new byte[iLen];
            in.read(bP);
            BigInteger m = new BigInteger(bP);
            in.close();
            pfMod = m; pfKey = p;
            // Secret key pour authentification
            bP = "Pas utilisé...".getBytes();
            Key = new SecretKeySpec(bP,"AES");
        } catch(Exception e) {
            return e.getMessage();
        }
        return "";
    }
//création des paires de clés
    public String CreateKeyPairFile(int nBits, String sFilePublic, String sFilePrivate) {
            // Crée une paire de clé
        SecureRandom random = new SecureRandom();
         BigInteger P = BigInteger.probablePrime(nBits / 2, random);
         String sP = P.toString();
        BigInteger Q = BigInteger.probablePrime(nBits / 2, random);
```

```
String sQ = Q.toString();
BigInteger one = BigInteger.ONE;
BigInteger phi = (P.subtract(one)).multiply(Q.subtract(one));
BigInteger modulus = P.multiply(Q);

////////////////////////////////// ENCRYPTION ////////////////////////
public String Encrypt(String sMsg, String sFilePublic) {
      // Encrypte une String (max 42car. avec clé de 512bits)
        BigInteger modulus;
    BigInteger publicKey;
    // Charge la clé privée
          File f=new File(sFilePublic);
          try {
                    FileInputStream in = new FileInputStream(f);
              byte[] bLenPub = new byte[4];
                    in.read(bLenPub);
              int iLenPub = BytesToInt(bLenPub);
                    byte[] bPub = new byte[iLenPub];
                    in.read(bPub);
              byte[] bLenMod = new byte[4];
              in.read(bLenMod);
              int iLenMod = BytesToInt(bLenMod);
              byte[] bMod = new byte[iLenMod];
              in.read(bMod);
                    in.close();
                    BigInteger m = new BigInteger(bMod);
```

```
                BigInteger p = new BigInteger(bPub);
                        modulus = m; publicKey = p;
            } catch(Exception e) {
                return e.getMessage();
                                }
    // Crypte le message
                byte[] bMsg = sMsg.getBytes();
                byte[] bMsg1 = new byte[bMsg.length + 1];
                for(int i=0; i<bMsg.length; i++) bMsg1[i+1]=bMsg[i];
                bMsg1[0]=1; // Pour éviter les valeurs négatives...
                BigInteger msg = new BigInteger(bMsg1);
                BigInteger out = msg.modPow(publicKey, modulus);
                return out.toString();
            }
public byte[] Encrypt(byte[] bMsg, String sFilePublic) {
        // Encrypte une byte array  (max 42bytes avec clé de 512bits)
                BigInteger modulus;
                BigInteger publicKey;
            File f=new File(sFilePublic);
                try {
                    FileInputStream in = new FileInputStream(f);
                        byte[] bLenPub = new byte[4];
                        in.read(bLenPub);
                         int iLenPub = BytesToInt(bLenPub);
                        byte[] bPub = new byte[iLenPub];
                         in.read(bPub);
```

```
                    byte[] bLenMod = new byte[4];
                    in.read(bLenMod);
                    int iLenMod = BytesToInt(bLenMod);
                    byte[] bMod = new byte[iLenMod];
                in.read(bMod);
                in.close();
                    BigInteger m = new BigInteger(bMod);
                BigInteger p = new BigInteger(bPub);
                     modulus = m; publicKey = p;
            } catch(Exception e) {
                    return null;
            }

        // Décrypte le message
         BigInteger msg = new BigInteger(sMsg);
        BigInteger out = msg.modPow(privateKey, modulus);
        byte[] bMsg = out.toByteArray();
        byte[] bMsg1 = new byte[bMsg.length - 1];
        for(int i=0; i<bMsg1.length; i++) bMsg1[i]=bMsg[i+1];
            String sOut = new String(bMsg1);
            return sOut;
              }
```

5.4 Présentation du cryptage combiné avec les algorithmes RSA/AES

Dans ce paragraphe, nous allons présenter le fruit de notre étude.

5.4.1 La partie zone de saisie

Dans cette partie, l'utilisateur peut, soit :

- Créer son propre texte à crypter
- Charger les données à partir d'un fichier se trouvant dans son disque dur

5.4.1.1 En mode saisie

Comme le montre la figure 5.01, on a une console présenté en mode saisie. Il existe une zone de texte où l'utilisateur peut saisir un texte à crypter et ensuite cliquer sur le bouton sauvegarder pour enregistrer le texte saisie dans un répertoire.

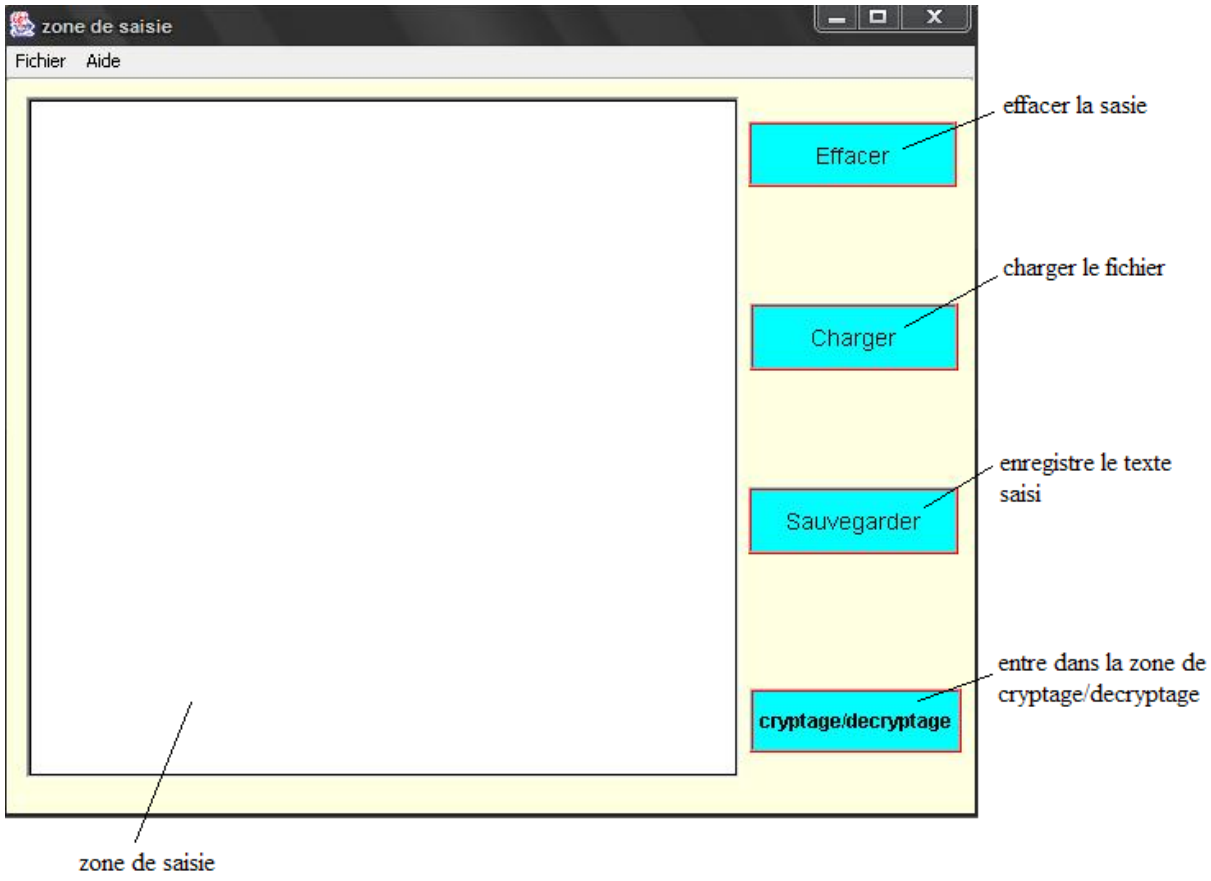

Figure 5.01 : schéma de la console en mode saisie

5.4.1.2 En mode chargement

Le bouton Charger permet par contre de charger un fichier texte ou crypté sur la console zone de saisie. Ceci offre à l'utilisateur le choix de modifier le contenu d'un texte sans avoir à le rédiger de nouveau.

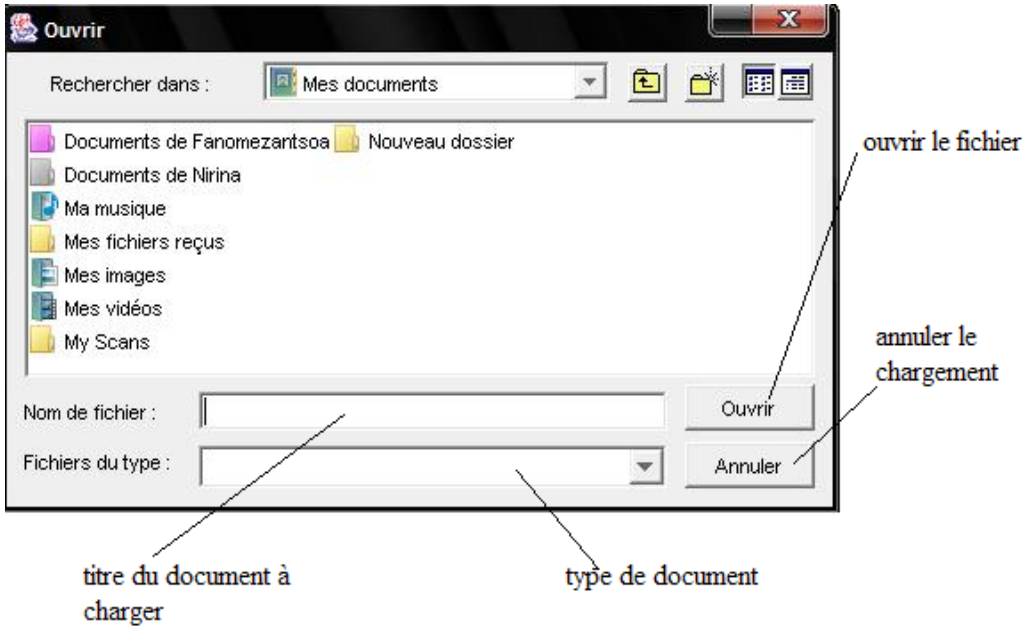

Figure 5.02 : schéma du chargement du fichier

Quand on clique sur le bouton « cryptage/decryptage », une boite de dialogue s'ouvre si on veut entrer dans la zone de cryptage ou revenir en mode saisie.

Figure 5.03 : fenêtre de dialogue

Deux cas se présentent alors :

Si on clique sur « oui », une nouvelle fenêtre s'ouvre, et on entre dans la zone de cryptage.

Si on clique sur « non », on retourne en mode saisie.

5.4.2 La partie zone de cryptage

Cette partie est réservée uniquement pour le cryptage/decryptage de donnée. Le logiciel gère automatiquement l'opération à effectuer c'est-à-dire que le logiciel sait auparavant que le fichier source est un fichier crypté ou non.

Voici un exemple de fichier non crypté :

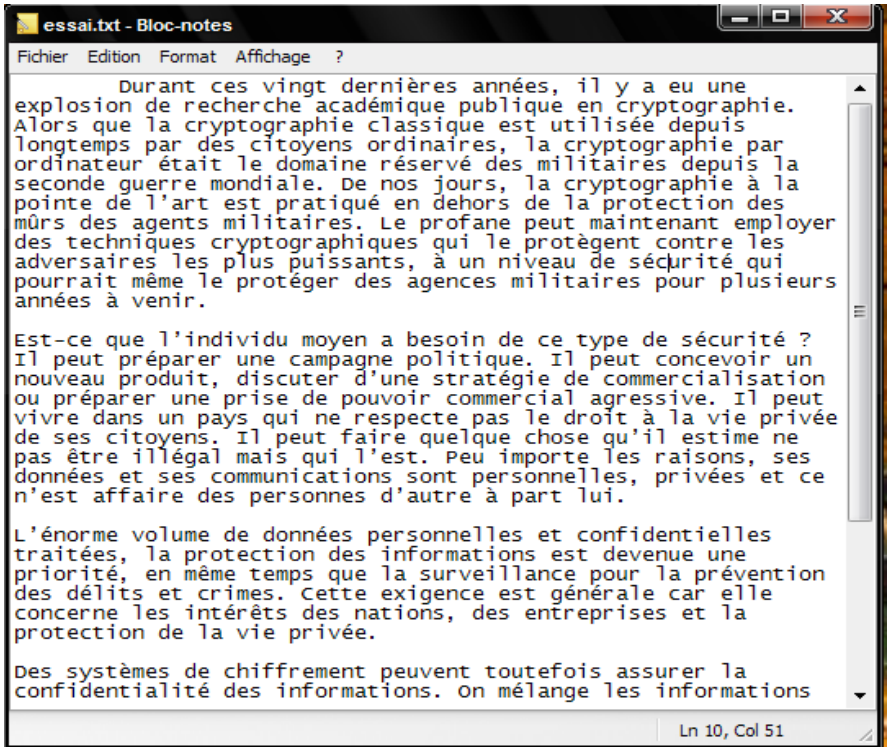

Figure 5.04 : exemple d'un fichier non-crypté

La figure suivante nous montre la zone de cryptage/decryptage :

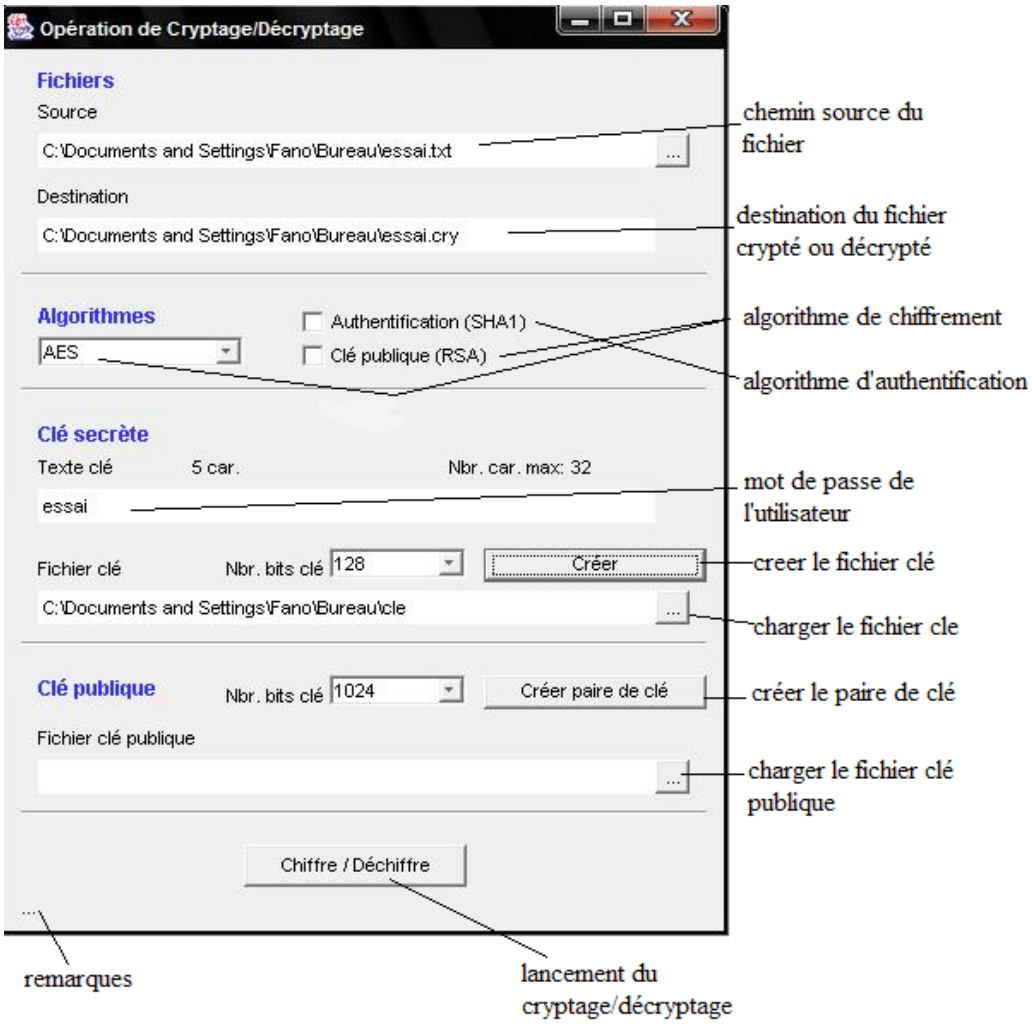

Figure 5.05 : schéma de la zone de cryptage

5.4.2.1 En mode cryptage

Le mécanisme de cryptage se déroule comme suit :

- Dans la zone supérieure, on clique sur le bouton « … » pour chercher le fichier à crypter. Quand on l'a trouvé, le chemin du fichier à crypter s'affiche sur la zone « source ». on notera aussi que le chemin du fichier crypté s'affiche sur la zone « Destination ». on peut remarquer que l'extension du fichier crypté est toujours « .cry ».

- Dans la seconde zone, on choisi l'algorithme de chiffrement à utiliser pour le cryptage de données. Dans le menu déroulant, on a le choix entre AES et RSA, le choix entre ces deux algorithmes nous guide vers quel zone on doit entrer pour la création de clé.

On notera qu'on a les trois choix suivants :

- Soit, on utilise uniquement l'algorithme à clé privée AES, dans ce cas, on ne coche pas la partie écrit « clé publique(RSA) »

- Soit, on utilise uniquement l'algorithme à clé publique RSA, dans ce cas, on choisi dans le menu déroulant l'algorithme RSA.

- Soit, on fait la combinaison des deux algorithmes, dans ce cas, on choisi dans le menu déroulant l'algorithme AES et on coche la zone RSA.

Figure 5.06 : choix de l'algorithme de chiffrement

A volonté, on peut aussi faire une authentification avec l'algorithme SHA1. Dans ce cas

Dans toute la suite, on prendra l'exemple où on utilise la combinaison de ces deux algorithmes.

Dans la troisième zone, on entre le texte clé ou mot de passe pour la génération de la clé secrète de l'algorithme AES. Toutefois, le nombre de caractère maximum est limité à 32 caractères, si on dépasse cette limite, une fenêtre de mise en garde s'affiche.

Figure 5.07 : fenêtre de mise en garde

Ensuite, on doit générer le fichier clé. Le nombre de bits de la clé est variable selon notre besoin. Elle varie de 128 bits à 432 bits. Il faut aussi cliquer sur le bouton créer pour créer notre clé privé. Le chemin de destination de notre fichier clé s'affiche sur la zone fichier.

Enfin, on doit aussi générer la paire de clé utilisé par l'algorithme RSA. Le mécanisme se fait comme suit :

On clique sur le bouton « créer paire de clé » une fenêtre s'ouvre indiquant le chemin du fichier clé publique.

Une deuxième fenêtre s'ouvre indiquant le chemin du fichier clé secrète. On cherche donc pour ce cas, le fichier clé privé généré avec AES, et c'est ce fichier qui sera notre clé secrète.

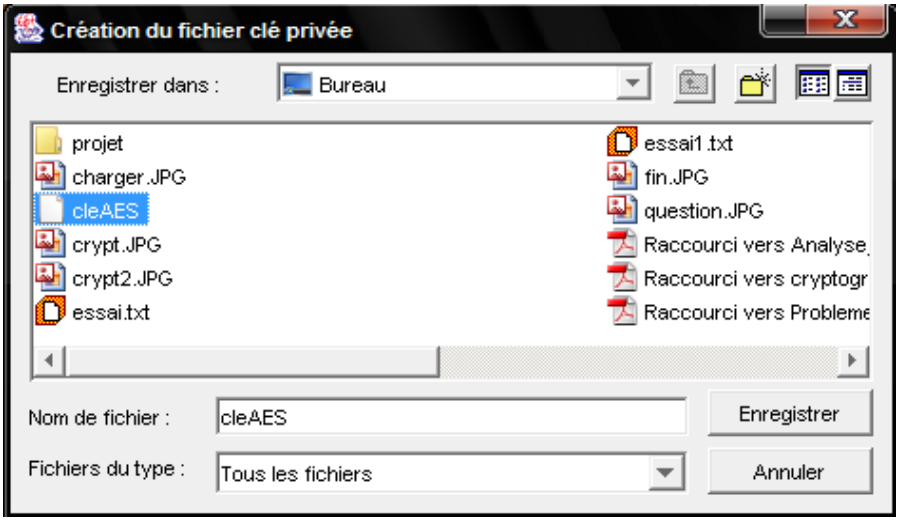

Figure 5.08 : création du fichier clé publique

Le nombre de bits de pour l'algorithme à clé publique AES est aussi variable allant de 1024 bits à 4096 bits suivant l'importance de l'application à utiliser.

Après avoir effectué ces étapes, on peut maintenant cliquer sur le bouton « Chiffre/Déchiffre » pour lancer le cryptage.

Le fichier crypté se présente donc comme la figure 5.09 suivante :

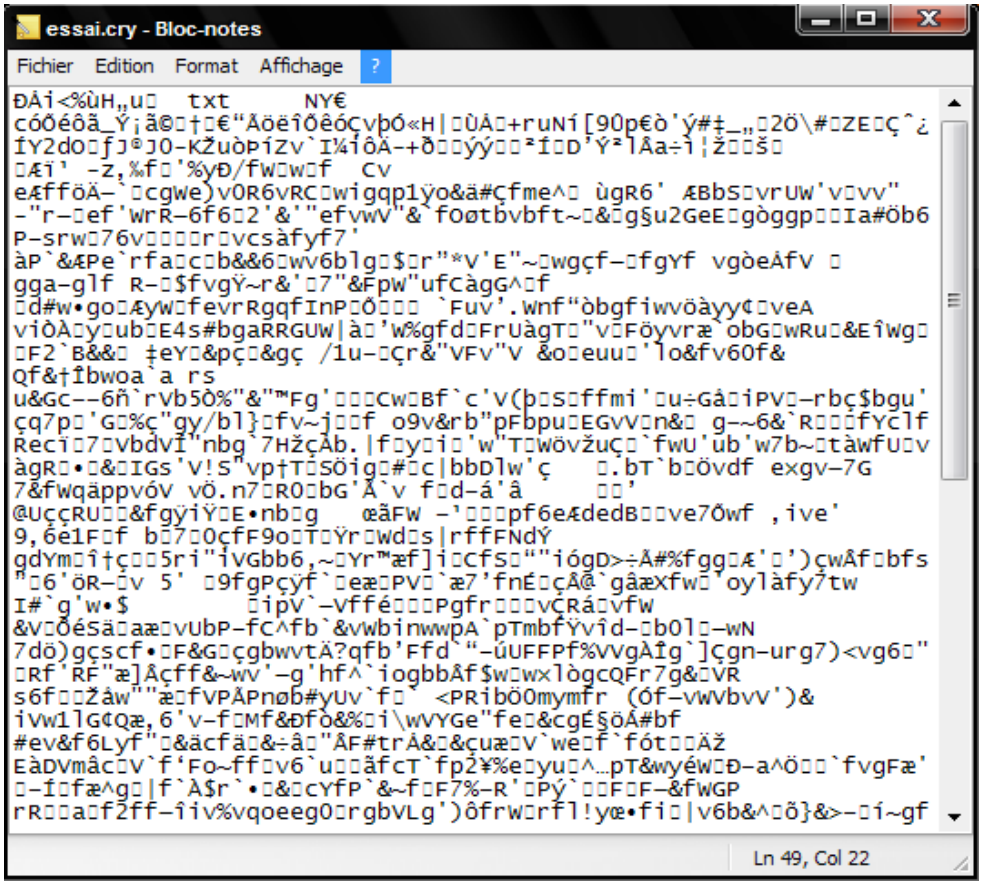

Figure 5.09 : aperçu d'un fichier crypté

A la fin du cryptage, une fenêtre de dialogue s'affiche si on veut encore crypter un autre fichier. Pour notre exemple, on choisi « non » pour terminer l'application.

5.4.2.2 En mode decryptage

Il est à noter que le mode décryptage se lance automatiquement. En effet lorsqu'on entre dans le fichier source un fichier dont l'extension est « .cry », il effectue automatiquement un décryptage et restitue son extension d'origine comme le montre la figure 5.10

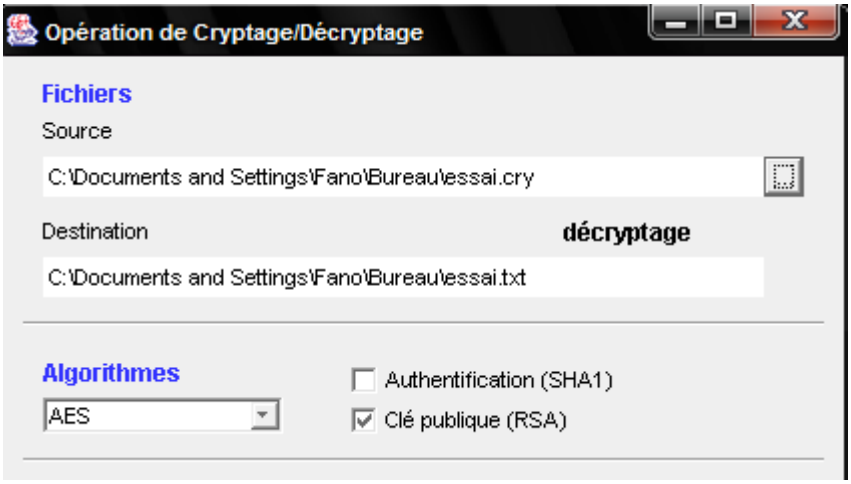

Figure 5.10 : opération de décryptage

On remarque ici que dans la partie algorithme, l'algorithme AES combiné avec RSA est sélectionné automatiquement aussi. Il ne nous reste plus donc que d'entrer le mot de passe saisi pour le cryptage, et de chercher respectivement les clés publiques et privés générées avant le cryptage. A la fin on clique sur « Chiffre/Déchiffre » pour lancer le décryptage.

Voici donc nos deux clé publique et privé en les visualisant à l'aide d'un bloc note.

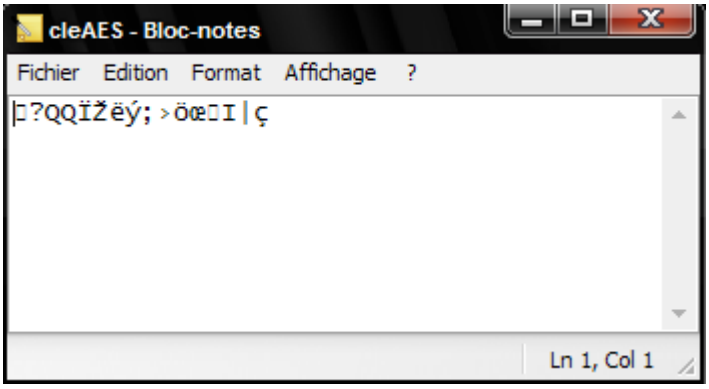

Figure 5.11 : aperçu d'une clé secrète AES

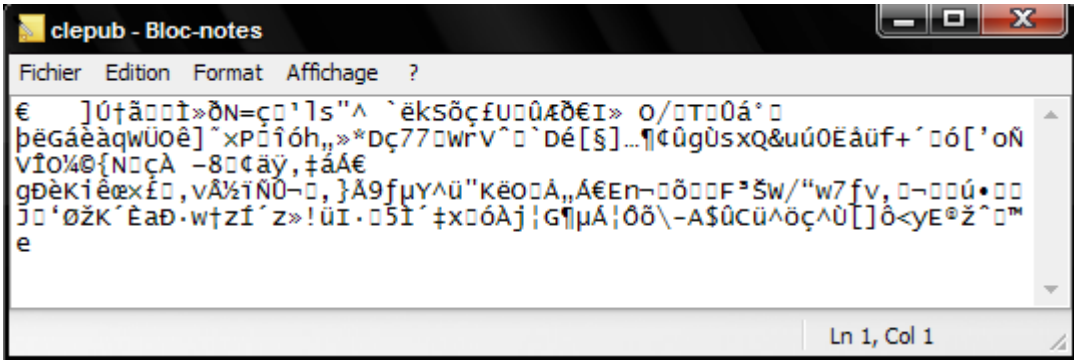

Figure 5.10 : aperçu d'une clé publique RSA

Remarque : Avant de clore cette présentation, nous aimerions laisser ces quelques remarques

Comme l'algorithme de chiffrement que nous avons choisis est un algorithme à clé publique, le chiffrement ou le déchiffrement sera un peu long pour un texte assez long.

L'utilisateur de ce système doit choisir un texte convenable pour être chiffré. Ceci afin d'éviter d'attendre longtemps.

Il est important de noter aussi que cette version de RAFCrypt 1.0 ne fonctionne qu'a la version de jdk 1.5 et plus

Comme nous avons utilisés JAVA, ce système peut s'exécuter sous plusieurs plateformes comme Windows, Linux, Mac-Os ou Solaris.

CONCLUSION

Grâce à la cryptographie à clé publique, la communication chiffrée est possible sans canal sécurisé pour l'échange des clés. C'est un procédé dit asymétrique qui repose sur un couple de clés complémentaires. L'une des deux clés, dit clé publique, sert à chiffrer le message et peut être divulguée, tandis que l'autre appelé clé secrète, sert à déchiffrer le message ainsi chiffré.

Malgré ces progrès, la cryptographie à clé publique à deux graves limitations. D'une part, la technique est relativement lente, de sorte qu'en pratique on ne peut pas chiffrer les messages trop longs. D'autre par, des structures caractéristiques du message en clair subsistent dans le message chiffré Ces fragments sont utilisables par des spécialistes, qui peuvent reconstituer la totalité des messages.

Pour ces raisons, le chiffrement est souvent assuré par des algorithmes à clé unique, plus rapides et plus sûrs et la cryptographie à clé publique ne sert que lors de l'étape d'échange des clés. Et c'est ceci le point fort de notre réalisation du fait que la combinaison des algorithmes RSA/AES donne une meilleure sécurité, et un plus rapide temps de chiffrement.

Dans le cas de l'authentification d'un message, après le calcul du condensé du message au moyen d'une fonction de hachage (l'algorithme SHA en est un exemple), on chiffre ce condensé avec la clé secrète et le résultat obtenu constituent la signature qui sera envoyé avec le texte chiffré. A l'arrivée on déchiffre la signature avec la clé publique correspondante pour confirmer l'identité de l'envoyeur du message reçu. La cryptographie à clé publique contribue beaucoup pour la gestion des clés.

La cryptographie est très utilisée aussi dans le domaine de la télécommunication. En effet pour protéger leurs bases de données contenant toutes les informations concernant leurs abonnés et l'accès au réseau les exploitants ont recours à la cryptographie.

Pour notre part nous avons conçu un cryptosystème pour mettre en œuvre les connaissances que nous avons acquises durant nos recherches concernant les algorithmes cryptographiques. Les principales difficultés que nous avons rencontrées durant le développement, se sont survenues durant l'implémentation du système. Il y a eu quelques programmes qui nous ont posé quelques problèmes. Mais grâce à la persévérance nous avons résolu les problèmes.

Nous laissons à ceux qui veulent continuer nos travaux le soin d'apporter leur contribution pour améliorer ce cryptosystème qui restera la propriété du département Télécommunications de l'Ecole Supérieure Polytechnique d'Antananarivo.

ANNEXE

ANNEXE 1 Comparaison des différentes sortes d'algorithmes de cryptographie

La cryptographie à clé publique et la cryptographie à clé secrète :

Cryptographie à clé unique ou cryptographie à clé publique ? Quelle est la meilleure ? Cette question n'a aucun sens mais fait l'objet d'un débat depuis que la cryptographie à clé publique a été inventée.

En fait les deux cryptographies sont deux choses différentes car elles résolvent des problèmes de types différents.

La cryptographie à clé unique est meilleure pour chiffrer des données. Ceci s'explique par le fait qu'elle est infiniment plus rapide et n'est pas prédisposée à des attaques à texte chiffré choisi.

La cryptographie à clé publique peut faire des choses que la cryptographie à clé unique ne permet pas. En effet elle est adaptée pour la gestion des clés et les protocoles.

Chiffrement par bloc et chiffrement en continu :

Biens que les systèmes de chiffrement par blocs et les systèmes de chiffrement en continues sont tous les deux des systèmes à clé secrète, ils sont très différents et complémentaires du point de vue conception. En effet les systèmes de chiffrement par blocs peuvent être utilisés en tant que chiffrement en continu et les systèmes de chiffrement en continu peuvent être utilisés en tant que chiffrement par bloc.

ANNEXE 2 Notion sur la théorie de complexité

La théorie de la complexité fournit une méthodologie pour analyser la complexité de calcul de différents algorithmes et techniques cryptographiques.

Elle consiste à comparer les algorithmes et les techniques cryptographiques pour déterminer le niveau de sécurité. Elle apprend ainsi si les algorithmes cryptographiques peuvent être cassés avant la fin du monde.

Complexités des algorithmes

La complexité d'un algorithme est déterminée par la puissance de calcul nécessaire pour l'exécuter.

La complexité calculatoire d'un algorithme est mesurée par deux paramètres :

- Le paramètre T pour la complexité en temps
- Le paramètre S pour la complexité en espace

En général T et S sont tous deux exprimés en fonction de n, où n est la taille de l'entrée.

La complexité calculatoire d'un algorithme est exprimée à l'aide de ce que l'on appelle «grand O ». L'ordre de grandeur de la complexité de calcul à savoir le terme de la fonction de complexité qui croît le plus vite avec n. Toutes les constantes et les termes d'ordre inférieur sont ignorés. Par exemple, si la complexité en temps d'un algorithme est de $1\mu s$, alors la complexité en calcul sera de l'ordre de, ce qui s'exprime $O(n^2)$.

De cette façon, la mesure de la complexité en temps est indépendante du système.

Cette notation vous permet de voir la façon dont les besoins en temps et en espace évoluent avec la taille des entrées. Par exemple,

si $T = O(n)$, alors doubler la taille de l'entrée double le temps de calcul de l'algorithme

si $T = O(n^2)$, alors ajouter 1 bit à la taille de l'entrée double le temps de calcul de l'algorithme.

En général, les algorithmes sont classés d'après leur complexité en temps et espace. Un algorithme est *constant* si sa complexité est indépendante de $n : O(1)$. Un algorithme est dit linéaire si sa complexité croît linéairement avec *n*. Les algorithmes peuvent aussi *quadratique, cubiques,* etc…., que l'on appelle aussi algorithmes *polynomiaux* leur complexité est $O(n^t)$ où *t* est une constante. Les classes des algorithmes qui ont une complexité en temps polynomiale sont appelées algorithmes *polynomiaux en temps.*

Les algorithmes dont la complexité est de $O(t^{f(n)})$, où *t* est une constante et $f(n)$ est une fonction polynomiale de *n*. Le tableau montre le temps d'exécution de différentes classes d'algorithmes pour *n* valant 10^6.

Classe	Complexité	Nombre d'opérations Pour n = 10^6	Temps pour 10^6 Opérations par seconde
Constants	$O(1)$	1	$1\mu s$
Linéaires	$O(n)$	10^6	$1s$
Quadratiques	$O(n^2)$	10^{12}	11,6j
Cubiques	$O(n^3)$	10^{18}	32.000 années
Exponentiels	$O(2^n)$	10^{301030}	10^{301030} fois l'âge de l'univers

La théorie de Shannon

Dès 1949, Claude Shannon tente de donner des fondements théoriques à la cryptologie. Il adopte le point de vue de la théorie de l'information et introduit la notion de sécurité inconditionnelle.

Considérons les messages clairs, chiffrés et les clés comme des variables aléatoires M, C, K. On suppose que chaque clé k définit une bijection : $e_k : M \longrightarrow C$.

On a:

$$P(C = c) = \sum_{k} P(K = k)P(M = e_k^{-1}(c))$$

Shannon définit un cryptosystème comme parfait, ou assurant une sécurité inconditionnelle, si la condition:

$H(M \mid C) = H(M)$ est réalisée.

Autrement dit, la connaissance du message chiffré n'apporte aucune information sur le message clair.

BIBLIOGRAPHIE

[1] Collection Microsoft Encarta 2006

[2] J.P. GAULIER, *Analyse des algorithmes finalistes concourant pour le futur standard AES*, Conservatoire National des Arts et Métiers Centre Régional Associé de Limoges.

[3] M. MINIER, *Preuves d'analyse et de sécurité en cryptologie à clés secrète*, Universités de Limoges, Ecole Doctorale STS, 30 septembre 2002

[4] P. NICOLAS, *Cours de réseau Maitrise de l'informatique*, Université d'Angers, Année 1999/2000.

[5] R.V.J ANDRIAMASY, *Conception d'un cryptosystème client/serveur basée sur RSA*, Ecole Supérieur Polytechnique d'Antananarivo, Département Télécommunication, Année 2005/2006.

[6] F. ARNAULT, *Théorie des Nombres & Cryptographie*, Université de Limoges, cours de DEA, mai 2002

[7] G. LABOURET, *Introduction à la cryptographie*, cours, 5 nov.1998

[8] J. RAZAKARIVONY, *Cryptographie et sécurité réseau*, cours 5ème année, dép. Tél.-ESPA, A.U. :2006-2007

[9] M.VIDEAU, *critères de sécurités des algorithmes de la cryptographie*, Rapport d'exposé, Faculté des sciences Unice : Centre Universitaire d'informatique, 4 janvier 2005

[10] G. CHASSÉ, *cryptographie mathématique*, maitre assistant de mathématique, Ecole des Mines de Nantes.

[11] B. SCHNEIER, *Cryptographie appliquée*, VUIBERT, 1996

[12] R.COHEN, M.GIRAULT, M.CAMPAMA, *Formule de dénombrement*, Annales des Télécommunications, 1992

[13] P. ZIMMERMAN, *Cryptographie et réseau*, Revue pour la science, N° 260, Juin 1999

[14] M. MAIMAN, *Télécom et réseau*, MASSON, 1996

[15] A. TANENBAUM, *Réseaux : Architecture, Protocoles, Applications*, Interédition 1992

[16] G.PUJOLLE, D.SERET, D.DROMARD, E.HORLAIT, *Réseaux et télématique*, EYROLLES, 1989

[17] J.DOUDOUX, *développons en java,* version 0.85 :2006

Nom : RAFILIPSON

Prénom : Andrianisaina Fanomezantsoa

Titre de mémoire :

CRYPTAGE COMBINE AVEC LES ALGORITHMES RSA/AES

Nombres de pages : 100

Nombres de tableaux : 2

Nombre de figures : 26

Mots clés : En *Français* :

- Cryptographie, Cryptologie, Cryptanalyse, Authenticité, Confidentialité, Algorithme, attaque, sécurité

En Anglais :

- Cipher, Encryption, Decryption, Algorithm.

Directeur de mémoire : RAZAKARIVONY Jules

Adresse de l'auteur : Lot II S 7 bis A, Anjanahary

ANTANANARIVO- 101

Tel : 03311 82122

E-mail : rafilipson@yahoo.fr

RESUME

Le monde évolue, le temps change, beaucoup de choses ne resteront pas comme elles étaient. Les progrès technologiques ont apportés des évolutions, mais aussi des méfaits comme l'insécurité. Et qui est la base de notre présent mémoire.

La cryptographie est l'une des meilleurs moyens d'assurer la sécurité des informations. Nous avons élaborés durant ce rapport l'étude des algorithmes cryptographique, ainsi que leurs implémentations dans la sécurisation de données.

Pour notre part, nous avons proposés un système de cryptage combiné de données avec les algorithmes RSA/AES. Nous avons ainsi lié les études théoriques à la réalisation pratique.

ABSTRACT

The world evolves, time changes, many things will not remain as they were. Technological progress brought evolutions, but also of the misdeeds like the insecurity. And which is the base of our present report.

Cryptography is one of the best means of ensuring the safety of information. We prepared during this report/ratio the cryptographic study of the algorithms, like their implementations in the securisation of data.

For our part, we proposed a system of combined encoding of data with algorithms RSA/AES. We thus bound the theoretical studies to the practical realization.