Université de Montréal

SAND : un chiffrement symétrique à très grandes S-Boxes

par
Patrick Baril-Robichaud

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

2 septembre, 2014

RÉSUMÉ

Nous avons développé un cryptosystème à clé symétrique hautement sécuritaire qui est basé sur un réseau de substitutions et de permutations. Il possède deux particularités importantes. Tout d'abord, il utilise de très grandes S-Boxes incompressibles dont la taille peut varier entre 256 Kb et 32 Gb bits d'entrée et qui sont générées aléatoirement. De plus, la phase de permutation est effectuée par un ensemble de fonctions linéaires choisies aléatoirement parmi toutes les fonctions linéaires possibles. Chaque fonction linéaire est appliquée sur tous les bits du bloc de message. Notre protocole possède donc une structure simple qui garantit l'absence de portes dérobées.

Nous allons expliquer que notre cryptosystème résiste aux attaques actuellement connues telles que la cryptanalyse linéaire et la cryptanalyse différentielle. Il est également résistant à toute forme d'attaque basée sur un biais en faveur d'une fonction simple des S-Boxes.

Mots clés : cryptographie symétrique, complexité de Kolmogorov, réseaux de substitutions et permutations, grande S-Boxe

ABSTRACT

We developed a new symmetric-key algorithm that is highly secure. Our algorithm is SPN-like but with two main particularities. First of all, we use very large random incompressible s-boxes. The input size of our s-boxes vary between 256 Kb and 32 Gb. Secondly, for the permutation part of the algorithm, we use a set of random linear functions chosen uniformly and randomly between every possible fonctions. The input of these functions is all the bits of the block of messages to encode. Our system has a very simple structure that guarantees that there are no trap doors in it.

We will explain how our algorithm is resistant to the known attacks, such as linear and differential cryptanalysis. It is also resistant to any attack based on a bias of the s-boxes to a simple function.

Keywords: symmetric cryptography, Kolmogorov's complexity, Substitution-Permutation networks, large s-boxes

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SIGLES

AES	Advance Encryption Standard
BIC	Bit Independance Criteria
CRYPTREC	Cryptography Research and Evaluation Committees
DES	Data Encryption Standard
Gb	Giga bits
GHz	Giga Hertz
Kb	Kilo bits
Mb	Mega bits
NESSIE	New European Scheme for Signatures, Integrity and Encryption
NIST	National Institute of Standard and Technology
NSA	National Security Agency
SAC	Strict Avalanche Criteria
SPN	Substitution Permutation Network
XSL	eXtended Sparse Linearisation

(dédicace) Je dédie ce mémoire à ma famille qui m'a soutenu pendant l'ensemble de mes études et qui a permis l'élaboration de ce document.

REMERCIEMENTS

Tout d'abord, je tiens à remercier mon directeur de thèse, Alain Tapp, pour sa supervision, sa collaboration et son soutien tout au long de la rédaction de ce mémoire. Je remercie également Raphaël Major pour son soutien dans l'ensemble de mes études, ainsi que son aide dans la correction de ce document. Je tiens à remercier l'ensemble des membres du LITQ dont, plus spécialement, Samuel Ranellucci pour son aide dans la vérification de certains des résultats de ce mémoire ainsi que Claude Gravel pour son aide avec les différentes notions de statistique.

CHAPITRE 1

INTRODUCTION

La cryptographie est un domaine qui passionne l'humanité depuis des millénaires. Les premières techniques consistaient à avoir un alphabet particulier, des mots codés spécifiques ou tout simplement à cacher les messages de différentes manières. Dans l'antiquité, Histiée aurait envoyé un esclave à Aristagoras avec un message lui indiquant de raser la tête du messager, car un tatouage indiquant à Aristagoras qu'il était temps de se révolter contre les Perses y était camouflé [24]. Depuis, plusieurs autres méthodes ont été développées telles que le chiffrement de César, le chiffrement de Vigenère, Enigma, et, finalement, la cryptographie actuelle avec AES et le chiffrement à clé publique RSA. Les concepteurs de codes secrets, les cryptographes, livrent une bataille millénaire contre les briseurs de codes, les cryptanalystes. Les cryptographes ont gagné une manche importante avec la conception du *one time pad* dont la sécurité et la résistance au déchiffrement ont été formellement démontrées. Malheureusement, le fait que la clé ne puisse être utilisée qu'une seule fois limite grandement son utilisation à grande échelle.

De nos jours, la cryptographie est divisée en deux domaines spécifiques, soit la cryptographie à clé privée et celle à clé publique. La cryptographie à clé privée correspond aux algorithmes de chiffrement dont la sécurité dépend de l'existence d'une clé secrète connue uniquement de deux interlocuteurs qui souhaitent communiquer ensemble. Un des aspects laissés sans solution par ces algorithmes est donc la distribution et le partage de telles clés entre deux personnes. Ces techniques sont tout à fait inadéquates lorsque deux personnes inconnues souhaitent communiquer secrètement pour la première fois sans avoir à traiter avec un intermédiaire les mettant en contact. La cryptographie à clé publique correspond à des algorithmes comportant deux types de clés, soit une clé privée et une clé publique. La clé publique est une clé qui est connue de tous et qui est utilisée pour chiffrer des messages à l'intention de l'émetteur de la clé. La clé privée, quant à elle, doit rester secrète et permet de déchiffrer les messages chiffrés à partir de

la clé publique. Malheureusement, ces algorithmes sont moins efficaces et significativement plus lents que les algorithmes de chiffrement à clé privée. De plus, la sécurité de ces cryptosystèmes nécessite d'utiliser de très grandes clés publiques. Or, pour tous ces cryptosystèmes, il n'existe aucune preuve qu'il n'existe pas d'algorithmes permettant de simplifier la complexité de ces éléments et de déterminer efficacement la valeur de la clé privée à partir de la clé publique. Par conséquent, les protocoles de cryptographie à clé publique utilisent les deux types de cryptographie dans des protocoles hybrides. Ils utilisent la cryptographie à clé publique pour échanger une première paire de clés entre deux interlocuteurs et, par la suite, ils utilisent cette paire de clés pour communiquer en utilisant un protocole de cryptographie privé.

Depuis la sélection du standard de chiffrement avancé *AES : Advance Encryption Standard* par le bureau des standards Américains aux États-Unis en 1997, il reste toujours un doute concernant l'existence de portes dérobées à l'intérieur d'AES qui ne serait connue que par la NSA. Par conséquent, dans ce mémoire, nous présenterons un nouvel algorithme de chiffrement à clé privée possédant une structure simple, mais qui, à notre avis, possède un très haut niveau de sécurité. Par sa structure simple, nous garantissons l'absence de portes dérobées dans notre système. Dans le chapitre 2, nous ferons un survol des algorithmes de cryptographie à clé symétrique et de trois concours importants, dont l'*Advance Encryption Standard*, qui ont eu lieu dans les dernières années. Dans le chapitre 3, nous présenterons les quatre caractéristiques essentielles que doivent posséder de bonnes S-Boxes pour pouvoir résister aux différents types de cryptanalyse. Le chapitre 4 décrit les principaux types de cryptanalyses utilisées actuellement et auxquels tous les cryptosystèmes doivent résister. Nous présentons en détail notre nouvel algorithme au chapitre 5. Le chapitre 6 contient l'analyse complète des propriétés de notre cryptosystème, dont sa résistance aux différentes techniques de cryptanalyse. Nous concluons en présentant les perspectives d'avenir concernant notre algorithme.

CHAPITRE 2

HISTORIQUE DES CRYPTOSYSTÈMES

Afin de situer nos travaux par rapport à la littérature actuelle, nous allons vous présenter ce qui s'est fait dans les dernières années dans le domaine des systèmes de cryptographie à clé symétrique. Ce n'est donc pas un historique complet de l'ensemble des cryptosystèmes qui ont existé mais plutôt des principaux cryptosystèmes à clé symétrique qui ont marqué la littérature des dernières années. Une liste plus complète peut être trouvée dans plusieurs thèses et mémoires, dont [5, 45, 59].

Cette section contient d'abord une description des deux structures sur lesquelles sont basés les cryptosystèmes que nous allons décrire, soit les réseaux de substitutions et de permutations et les réseaux de Feistel. Nous expliquerons ensuite comment le *Data Encryption Standard* et le *Advance Encryption Standard* furent décidés. Nous finirons par la présentation des recommandations de deux regroupements indépendants, soit le projet NESSIE (*New European Scheme for Signatures, Integrity and Encryption*) en Europe et CRYPTREC (*Cryptography Research and Evaluation Committees*) au Japon.

2.1 Réseau de substitutions et de permutations

Le premier élément est le réseau de permutations et de substitutions, soit un *Substitution and Permutation Network*. Un réseau de substitutions et de permutations est un réseau qui suit le schéma 2.1

Tel qu'illustré, un réseau de substitutions et de permutations est composé des trois étapes distinctes suivantes :

- Ou-exclusif entre l'entrée et la clé ;
- Permutation utilisant différents procédés comme le *Butterfly* ;
- S-Boxe réversible.

Ces trois étapes sont répétées plusieurs fois, et chacune de ces répétitions est considérée comme étant un *round*. Formellement, un réseau de substitutions et de permutations

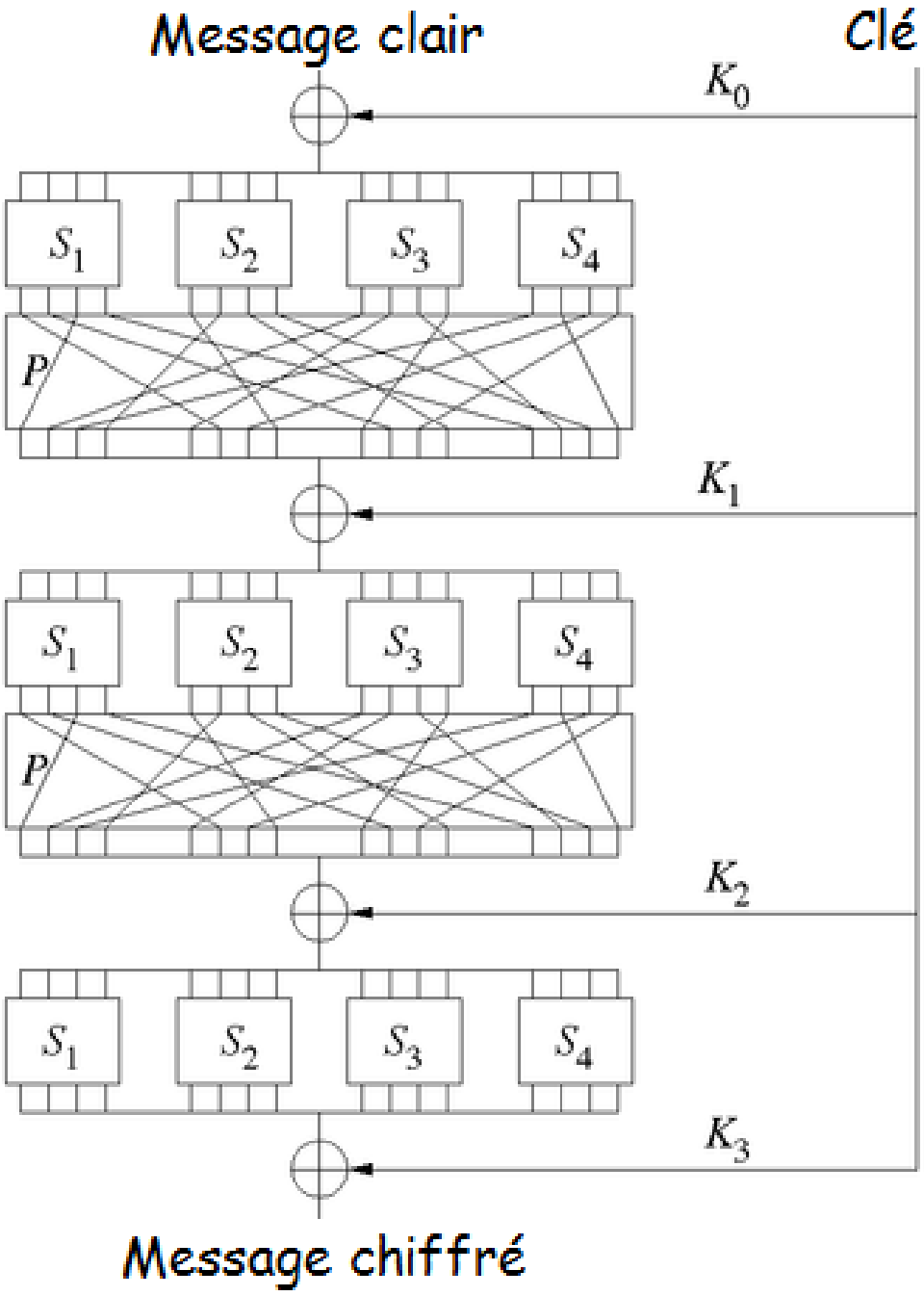

Figure 2.1 – Réseau de substitutions et de permutations [72]

est constitué des éléments suivants comme il est illustré dans la figure 2.1 :

- X_i correspond au ième bit d'entrée du système ;
- Y_i correspond au ième bit de sortie du système ;
- K correspond à la clé transmise avec le système ;
- K_i correspond à la ième clé intermédiaire générée à partir de K ;
- S_i correspond à la ième S-Boxe ;
- X correspond au message à chiffrer ;
- Y correspond au message chiffré.

Nous définissons également les S-Boxes de la manière suivante :

Définition 1. *S-Boxe n×m :* Soit une table de substitution S qui prend n bits en entrée et qui produit m bits en sortie, alors nous noterons que S est une S-Boxe $n \times m$ qui implémente une fonction $f := \{0,1\}^n \rightarrow \{0,1\}^m$.

2.2 Réseaux de Feistel

Un autre système souvent utilisé en cryptographie à clé privée est le système des réseaux de Feistel, qui est illustré dans la figure 2.2. Un réseau de Feistel est défini de la manière suivante :

Définition 2. *Une fonction de round F* est une fonction qui représente l'ensemble des opérations effectuées lors d'un *round* dans un cryptosystème.

Définition 3. *Réseau de Feistel :* soit F, une fonction de *round*, et K_i, les clés intermédiaires, déterminées à partir de K, une clé secrète partagée par les deux parties. L'opération de base est alors de séparer le message d'entrée X en deux sections de taille égale, soit X^L pour la moitié de gauche de X et X^R pour sa moitié de droite. Nous posons $L_0 = X^L$ et $R_0 = X^R$. Nous avons ensuite la relation de récurrence suivante :

- $L_{i+1} = R_i$;
- $R_{i+1} = L_i \oplus F(R_i, K_i)$.

où $\oplus$ correspond au ou-exclusif. Le chiffrement correspond à (R_{n+1}, L_{n+1}).

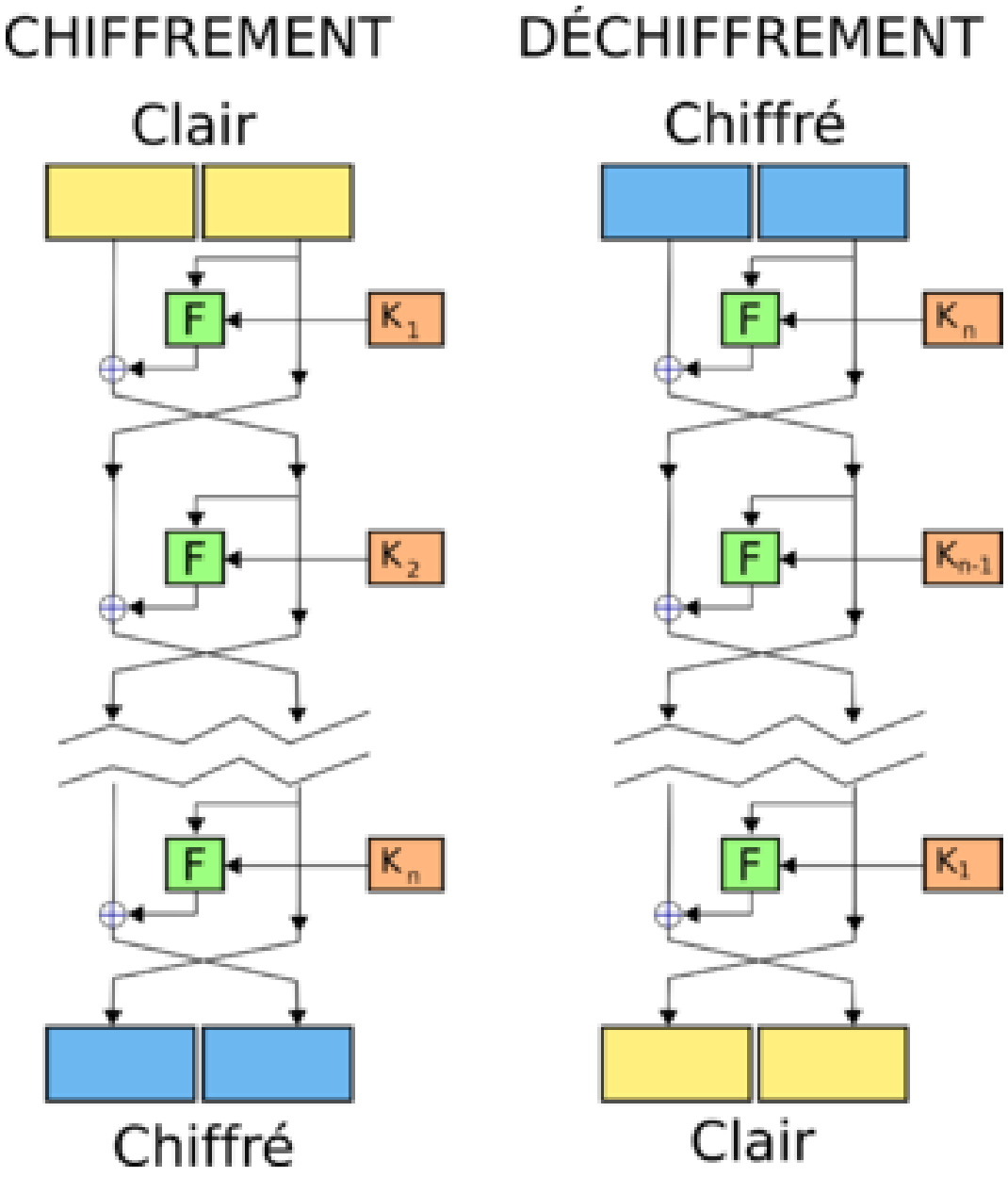

Figure 2.2 – Réseau de Feistel [70]

Par conséquent, le message chiffré, Y, correspond à la concaténation de R_{n+1} et de L_{n+1}, communément notée (R_{n+1}, L_{n+1}). Autrement dit un réseau de Feistel est basé sur la fonction de *round* suivante :

Définition 4. $g : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$

tel que $g(L,R,K) = (R, f(R,K) \oplus L)$
où f peut être n'importe quelle fonction prenant n et m bits respectivement et produisant n bits en sortie. "$\oplus$" symbolise l'opération ou-exclusif. Alors g décrit la *fonction de round d'un réseau de Feistel* . [35]

Il est important de noter que, dans ce système, la fonction f de la fonction de *round* n'a pas à être réversible. La structure de cette section est inspirée de [69] et les informations proviennent de [33].

2.3 Data Encryption Standard

DES est un cryptosystème qui fut développé par IBM dans le début des années 1970. Il a par la suite été soumis au National Bureau of Standards en 1976. Après une collaboration avec la NSA, celle-ci a publié une version modifiée de DES en tant que standard national de chiffrement (FIPS : Federal Information Processing Standard). Cette publication a eu pour effet de diffuser massivement DES et de le faire connaître dans le monde entier. Cette diffusion a fait en sorte que ce cryptosystème fut analysé en profondeur par plusieurs spécialistes du monde entier. Le choix d'une clé courte de 56 bits ainsi que les éléments tenus secrets sur les choix effectués lors de sa conception ont alimenté les rumeurs d'existence d'une porte cachée permettant à la NSA de déchiffrer rapidement et efficacement les chiffrements DES sans connaître la clé.

Depuis la fin des années 1990, DES est jugé non sécuritaire étant donné que la puissance de calcul des ordinateurs est maintenant suffisamment grande pour faire une fouille exhaustive de l'ensemble des clés de 56 bits en moins d'une journée. Cette fouille exhaustive a été effectuée en 1999 par deux organismes afin d'illustrer l'état de faiblesse de DES. [33, 67]

DES est un cryptosystème basé sur un réseau de Feistel à 16 *rounds*, comme illustré par la figure 2.3. Les deux seuls éléments à déterminer sont la fonction F qui est utilisée ainsi que la génération des clés intermédiaires. La fonction F est représentée par l'illustration 2.4. Elle est constituée de trois éléments, soit un ou-exclusif entre une clé intermédiaire, K_i, et la section L_i ou R_i augmentée par une fonction d'expansion, E,

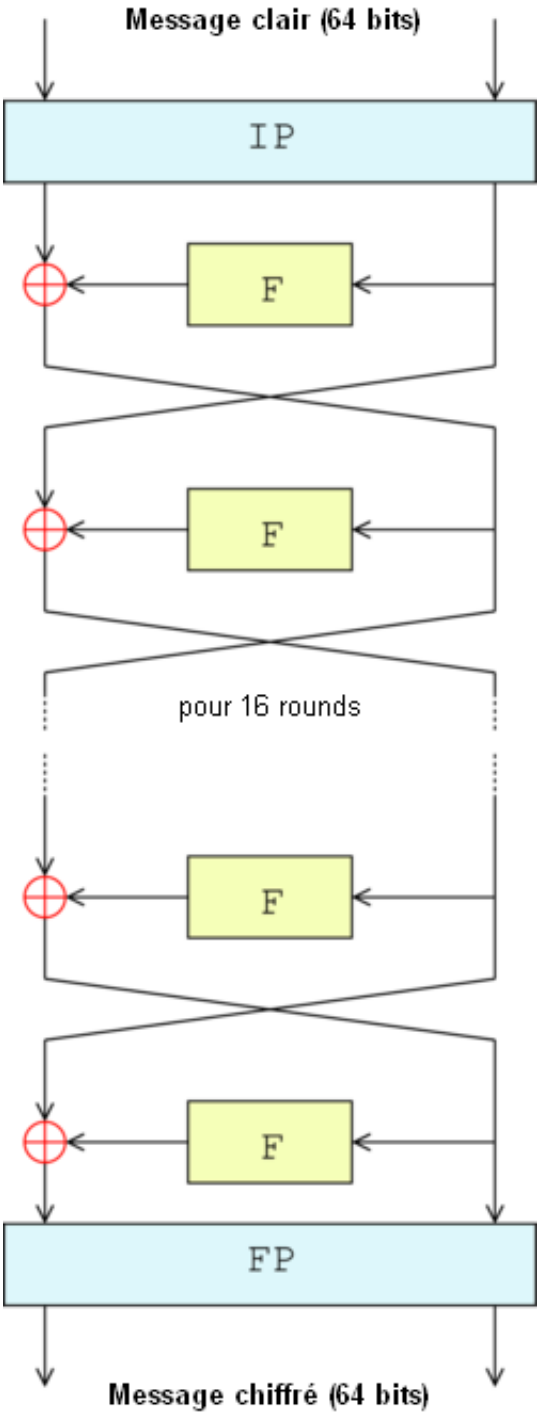

Figure 2.3 – DES [67]

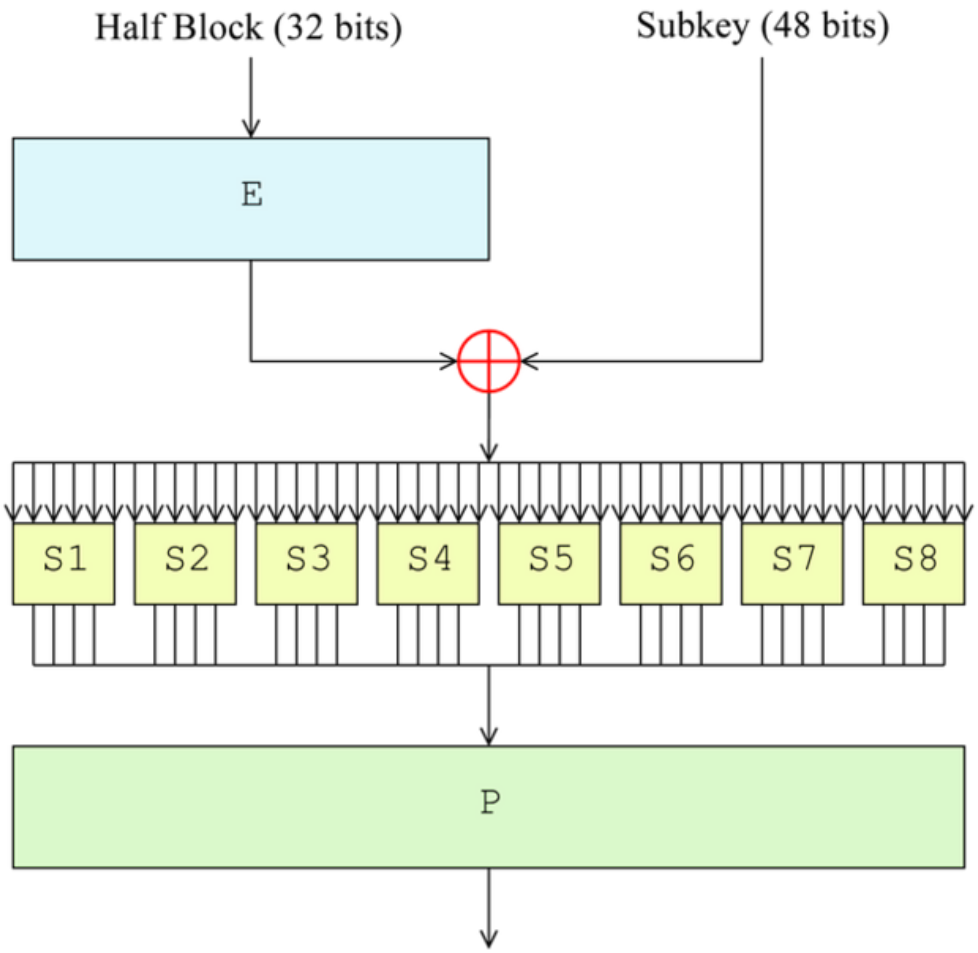

Figure 2.4 – Fonction F de DES [67]

connue. Par la suite, chaque bit résultant du ou-exclusif est distribué dans une série de huit S-Boxes identiques de six bits d'entrée et de quatre bits de sortie. Les S-Boxes de DES sont illustrées en annexe. [33, 67]

Les S-Boxes de DES possèdent trois propriétés importantes :

- Chaque S-Boxe est une fonction 4-vers-1, donc il existe exactement quatre entrées possibles pour chaque sortie de la S-Boxe.
- Chaque ligne de la table représentant la fonction de la S-Boxe contient chacune des chaînes de quatre bits possibles qu'une seule et unique fois, donc chaque ligne est une permutation des 16 chaînes de quatre bits possibles.
- Changer un bit de l'entrée change toujours au moins deux bits de la sortie.

Pour finir, la sortie des S-Boxes est insérée dans la permutation P illustrée dans le tableau 2.I. La sortie de la permutation correspond à la sortie de la fonction F. Afin de générer les clés intermédiaires, il est nécessaire de séparer la clé K en deux parties, soit sa partie de droite, R, et sa partie de gauche, L. Chaque clé intermédiaire K_i est constituée d'une permutation de 24 des 28 bits de R concaténée à une permutation de 24 des 28 bits de L. La permutation exacte utilisée à chaque *round* est évidemment connue publiquement et seule la clé primaire est gardée secrète dans ce protocole. La majorité des informations contenues dans cette section proviennent de [33].

2.3.1 3DES

3DES, ou plus communément appelé Triple DES, est le standard qui a officiellement remplacé DES en 1999. 3DES consiste en l'application de DES trois fois en utilisant deux ou trois clés différentes. Le fait d'utiliser trois fois DES a permis d'augmenter la taille totale de la clé à découvrir et, ainsi, d'améliorer la sécurité du système. Il a permis aux usagers de DES de garder l'infrastructure de DES tout en ayant un cryptosystème jugé beaucoup plus sécuritaire. [33]

2.4 Advance Encryption Standard

2.4.1 Concours Advance Encryption Standard

En septembre 1997, le *NIST*, *National Institute of Standard and Telecommunication*, a lancé le concours AES afin de déterminer un cryptosystème qui pourrait remplacer DES, qui devenait vieux et qui risquait de ne plus être sécuritaire. Le but du concours était de trouver un cryptosystème qui soit au moins aussi sécuritaire que 3DES mais qui serait plus rapide. Dans la demande de soumission, le NIST a posé deux conditions minimales, soit que la taille des blocs puisse être de 128, 192 et 256 bits et que la taille des clés soit également de 128, 192 et 256 bits. Dans une étape subséquente du processus, les restrictions sur la taille des blocs à chiffrer sont tombées à 128 bits seulement. Les deux autres tailles ont été abandonnées. Le concours était composé de quatre étapes : la soumission, l'évaluation des soumissions, la sélection d'un groupe restreint de finalistes

Tableau 2.I – Fonction P de DES [68]

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

et, finalement, le choix du gagnant. La date limite pour effectuer une soumission était le 15 mai 1998. La présentation des projets soumis avait lieu entre le 20 et le 22 août lors d'un *workshop* public. Il y eut 20 soumissions lors de la première phase, dont seulement 15 respectaient les prérequis demandés.

Parmi les 15 cryptosystèmes qui sont passés à la deuxième étape, cinq furent choisis pour passer à la dernière étape. Ces cinq systèmes sont les suivants :

- MARS de IBM ;
- RC6 de RSA ;
- Serpent de Anderson-Biham-Knudsen ;
- Twofish de Counterpane ;
- Rijndael de Daemen-Rijmen.

Les quatre premiers cryptosystèmes seront décrits brièvement dans les sections suivantes. Rijndael sera décrit en détail, car c'est ce cryptosystème qui a gagné le concours AES et qui est par conséquent devenu le nouveau standard le 2 octobre 2000.

Malgré le fait que Rijndael fut déclaré gagnant, aucune des 5 soumissions ayant passé la deuxième étape ne comportait de failles exploitables au moment de la soumission. Dans le domaine de la cryptographie à clé symétrique, il n'est pas possible de déterminer de manière efficace et certaine la sécurité d'un cryptosystème. La seule méthode connue est de montrer la résistance du cryptosystème par rapport à l'ensemble des attaques connues. Des critères tels que la flexibilité d'implémentation, l'élégance de la description et la simplicité du cryptosystème permirent à Rijndael de devenir le nouveau standard. Les informations de cette section ont été recueillies dans [5].

2.4.2 Finalistes du concours

2.4.2.1 MARS

MARS est le cryptosystème proposé par IBM, dont l'un des concepteurs était Don Coppersmith, qui avait conçu DES. C'est un cryptosystème de 32 *rounds* qui chiffre des blocs de 128 bits et utilise des clés de tailles variant entre 128 et 448 bits par bonds de 32 bits. [13, 76]

MARS est composé de trois sections, soit un bloc cryptographique de 16 *rounds* précédé et suivi par huit *rounds* de *forward* et de *backward mixing*. MARS transforme l'entrée de 128 bits en quatre mots de 32 bits, soit A, B, C et D. Le schéma 2.5 représente la structure de MARS.

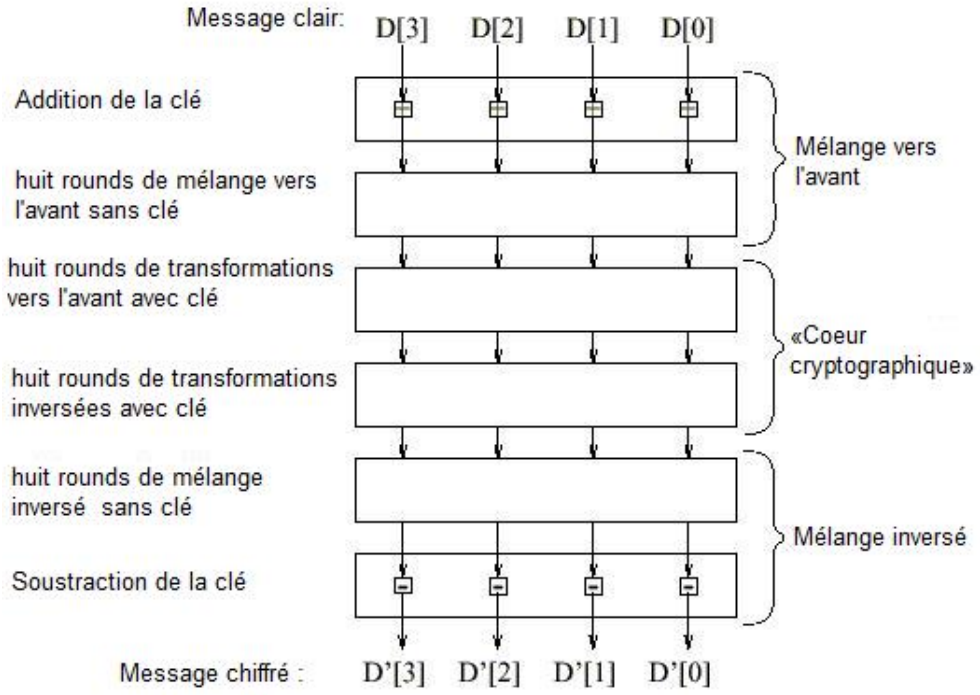

Figure 2.5 – MARS [27]

Le « coeur » cryptographique est basé sur la structure d'un réseau de Feistel. MARS combine des opérations de ou-exclusif (xor), d'addition, de soustraction et de multiplication et des rotations fixes et dépendantes des entrées. Il utilise deux S-Boxes de 8×32

bits nommées $S0$ et $S1$. Le pseudo-code décrivant le fonctionnement global de MARS se trouve dans [27].

En ce qui concerne la sécurité, les concepteurs estimaient que la sécurité de MARS avec n bits de clé est de 2^n bits jusqu'à 256 bits. Avec une clé de plus de 256 bits, les concepteurs n'espéraient pas avoir un niveau de sécurité qui augmente aussi rapidement. Par conséquent, le choix d'avoir des clés de plus de 256 bits aurait plus été une question pratique que sécuritaire. [14]

2.4.2.2 RC6

La compagnie RSA proposa la structure de RC6, qui est basée sur celle de RC5, qu'ils ont également développée. Une des particularités de RC6 est le fait qu'il est paramétrable pour des longueurs de mots, un nombre de *rounds* ainsi qu'une longueur de clé variable. Les versions de RC6 « RC6-128/20/128 », « RC6-128/20/192 » et « RC6-128/20/256 » ont, toutes les trois, été présentées au concours AES afin de respecter le critère des tailles de clé de 128, 192 et 256 bits. La façon d'identifier RC6 est la suivante : RC6-w/r/b, où w correspond à la longueur des messages à chiffrer ; r, au nombre de *rounds* et b, à la taille de la clé utilisée.

La structure de RC6 est basée sur celle d'un réseau de Feistel sans en être vraiment un. Le schéma 2.6 présente la structure de RC6. Le code en annexe I décrit en détail le chiffrement et le déchiffrement utilisés par RC6.

Les concepteurs estiment que sa sécurité est de l'ordre du minimum entre 2^{8b} et 2^{704}, où b correspond au nombre d'octets de la clé. L'attaque la plus rapide serait la recherche exhaustive de l'ensemble des clés et il n'existerait pas de clés faibles selon les concepteurs du système. Les informations concernant RC6 proviennent de [51].

2.4.2.3 Serpent

Serpent est un cryptosystème conçu par Ross Anderson, Eli Biham et Lars Knudsen et basé sur un réseau de substitution-permutation. Ce système possède des clés dont la taille est un multiple de huit, incluant 128, 192 et 256 bits. Serpent comporte 32 *rounds*,

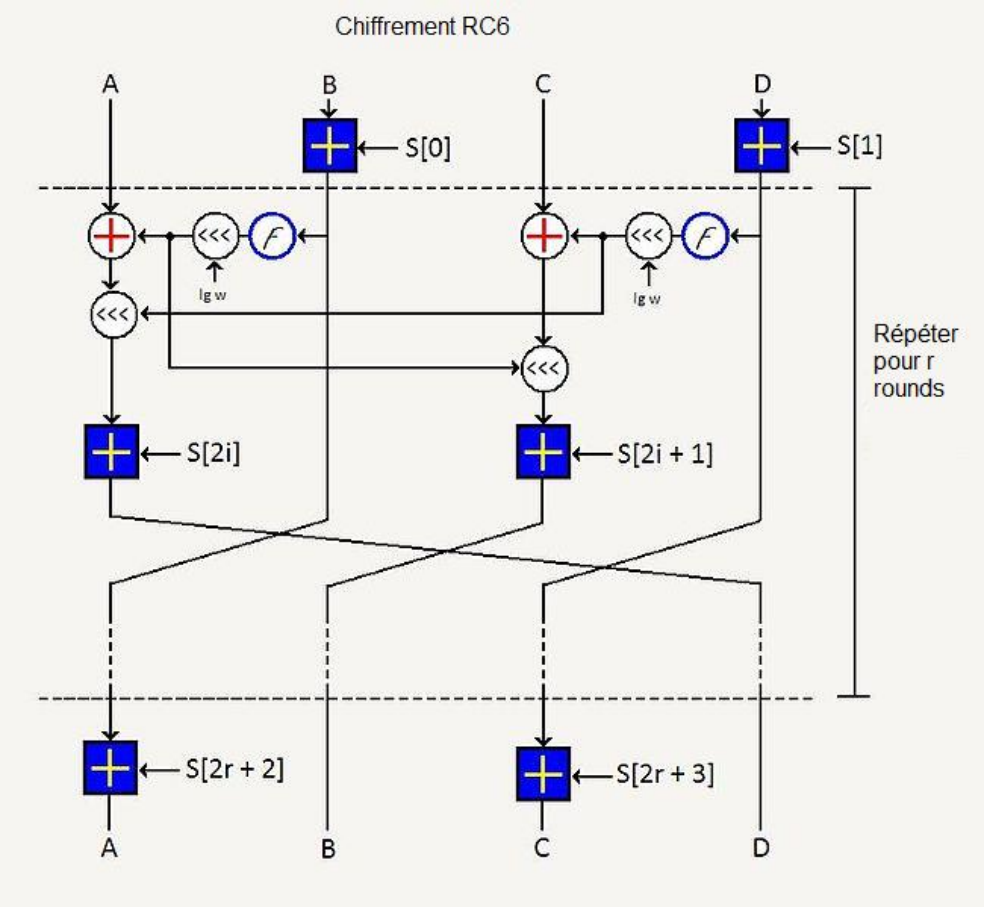

Figure 2.6 – RC6 [71]

et le message est séparé en quatre mots de 32 bits. Chaque *round* utilise une des huit S-Boxes de 4×4 bits. L'application des 32 S-Boxes de chaque *round* peut être effectuée en parallèle. Les concepteurs ont élaboré Serpent de manière à ce que chacune de ses opérations puisse être effectuée en parallèle en utilisant 32 tranches de un bit. L'opération de permutation entre chaque *round* est celle illustrée par la figure 2.7. Ces informations ont été obtenues dans [3, 78].

Les concepteurs ont opté pour une approche plus prudente que celles des autres finalistes du concours AES. Les concepteurs prétendent que Serpent peut résister à tous les types d'attaques connues avec seulement 16 *rounds*. Malgré cela, ils ont spécifié 32 *rounds* dans leur protocole afin de pouvoir pallier à d'éventuelles nouvelles attaques. De plus, Serpent est un cryptosystème public qui n'a pas été breveté et qui peut donc être

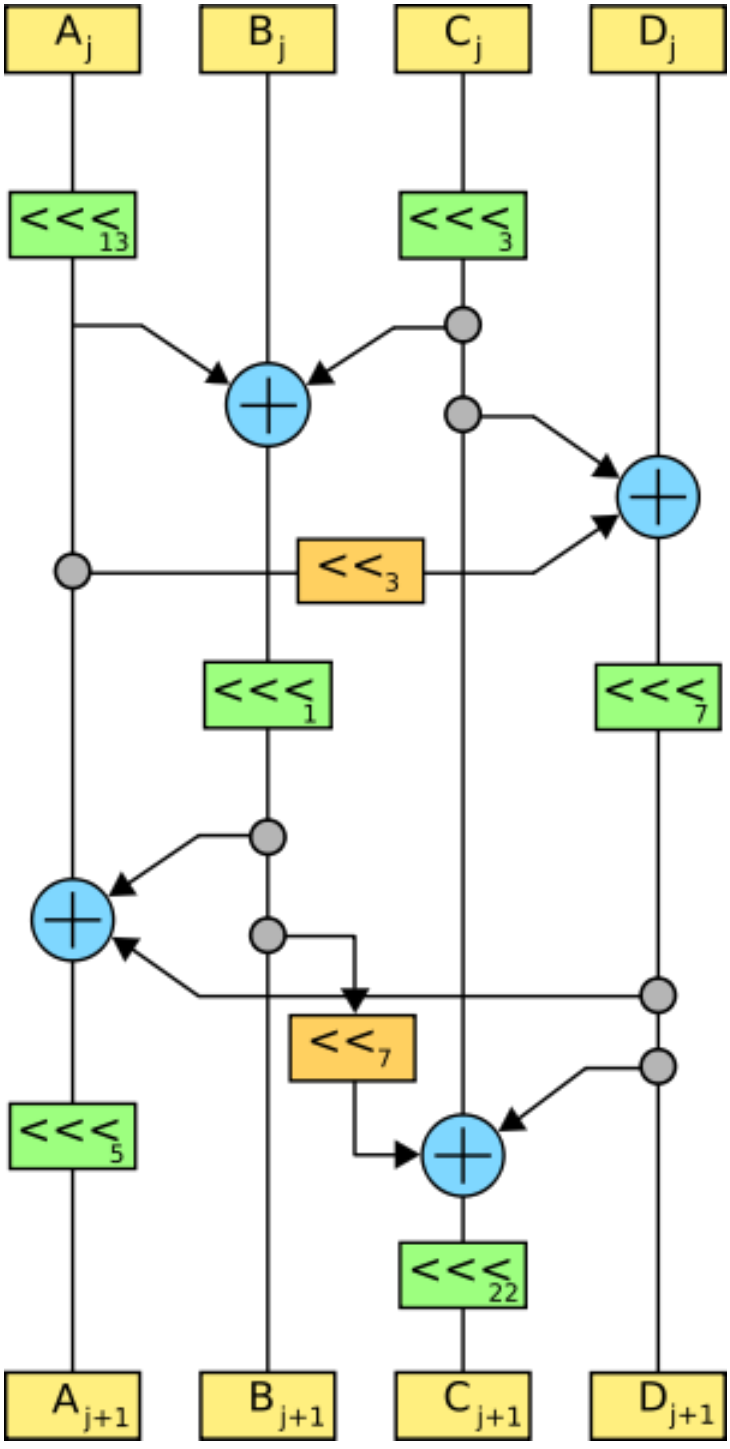

Figure 2.7 – Serpent [78]

implémenté gratuitement par n'importe qui.

En ce qui concerne la sécurité, aucune cryptanalyse de la version complète de Serpent n'a été possible. La meilleure cryptanalyse a été exécutée sur une version réduite de 11 *rounds* et demandait 2^{116} messages connus, un temps de $2^{107.5}$ opérations et un espace mémoire de 2^{104} bits. L'ensemble des informations qui concernent Serpent proviennent de [3, 78].

2.4.2.4 Twofish

Twofish est un algorithme de chiffrement symétrique basé sur le modèle d'un réseau de Feistel avec 16 *rounds*. Ses concepteurs sont Bruce Schneier, Niels Ferguson, John Kelsey, Doug Whiting, David Wagner et Chris Hall. Twofish est une variante de Blowfish qui est décrit dans [53] et qui a également été inventé par Bruce Schneier. Les éléments de base de ce système sont :

- des S-Boxes dépendantes de la clé choisie
- un système de génération de clés très complexe
- l'utilisation de la pseudo-transformation d'Hadamard dans la fonction de diffusion

La figure 2.8 représente l'algorithme de Twofish. Twofish chiffre des messages de 128 bits avec une clé de 128, 192 ou 256 bits. Il a été conçu afin de pouvoir être implémenté dans des systèmes embarqués tels que les cartes à puce. Au niveau de la performance, il est plus rapide que toutes les autres soumissions du concours AES à l'exception de Rijndael. Tout comme Serpent, Twofish n'a pas été breveté, ce qui en fait un cryptosystème public et gratuit. Il est inclus dans le standard *OpenPGP*, mais est par contre moins diffusé et utilisé que son prédécesseur, Blowfish.

En ce qui concerne la sécurité, il n'existe pas de cryptanalyses connues de la version complète de Twofish. Les concepteurs ont publié plusieurs articles démontrant des attaques de cryptanalyse sur des versions réduites de six ou sept *rounds*. La complexité d'une attaque sur la version à cinq *rounds* est de 2^{51} opérations. La structure de cette section est inspirée de [80, 81] et les informations ont été vérifiées dans [54].

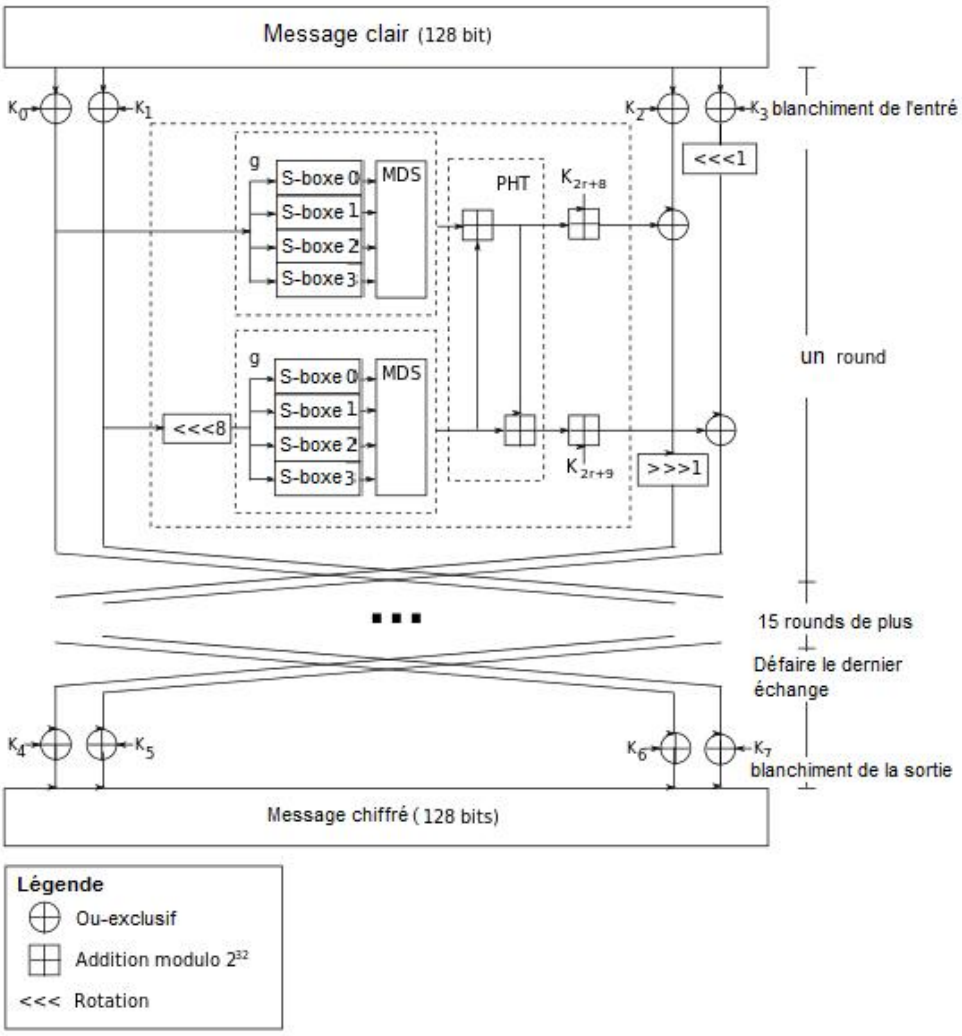

Figure 2.8 – Twofish [80]

2.4.3 Le gagnant : Rijndael

Rijndael est le cryptosystème gagnant du concours AES. Il a été conçu par Joan Daemen et Vincent Rijmen, qui ont nommé leur système en mélangeant une partie de leur nom. Rijndael possède la structure d'un réseau de substitutions et de permutations et est composé de quatre étapes distinctes à chaque *round*.

Une particularité de l'algorithme par rapport au réseau de substitutions et de permutations de base est que les 128 bits d'entrée sont traités comme un tableau de quatre

octets par quatre octets. L'ensemble des opérations est appliqué sur chacun des octets et non sur les bits. Le fait de traiter les bits d'entrée comme des octets permet d'avoir un algorithme plus efficace et rapide.

Les étapes de Rijndael :

- *SubBytes* (Figure 2.9) ;
- *ShiftRows* (Figure 2.10) ;
- *MixColumns* (Figure 2.11) ;
- *AddRoundKey* (Figure 2.12).

Voici l'algorithme général de Rijndael :

- *round* d'expansion de la clé : les clés intermédiaires sont déterminées à partir de la clé primaire et de la routine de génération de clé de Rijndael ;
- *round* initiale ;
  - exécution de l'opération *AddRoundKey* ;
- *round i* ;
  - exécution de l'opération *SubBytes*,
  - exécution de l'opération *ShiftRows*,
  - exécution de l'opération *Mixcolumns*,
  - exécution de l'opération *AddRoundKey*.
- dernier *round* ;
  - exécution de l'opération *SubBytes*,
  - exécution de l'opération *ShiftRows*,
  - exécution de l'opération *AddRoundKey*.

Comme illustrée à la figure 2.12, l'opération *AddRound Key* consiste à simplement effectuer un ou-exclusif entre les bits du message et une clé intermédiaire. L'opération *SubBytes* consiste à l'application d'une S-Boxe de un octet par un octet. La S-Boxe utilisée est connue et publique et elle est disponible en annexe II. L'opération *ShiftRows* consiste à décaler un certain nombre de fois vers la gauche chaque ligne de la matrice représentant l'état du système comme illustré à la figure 2.10. L'opération *MixColumns* consiste à effectuer un produit matriciel entre chaque colonne de l'état du système et la matrice 2.II. Par conséquent, les opérations *ShiftRows* et *MixColumns* servent à aug-

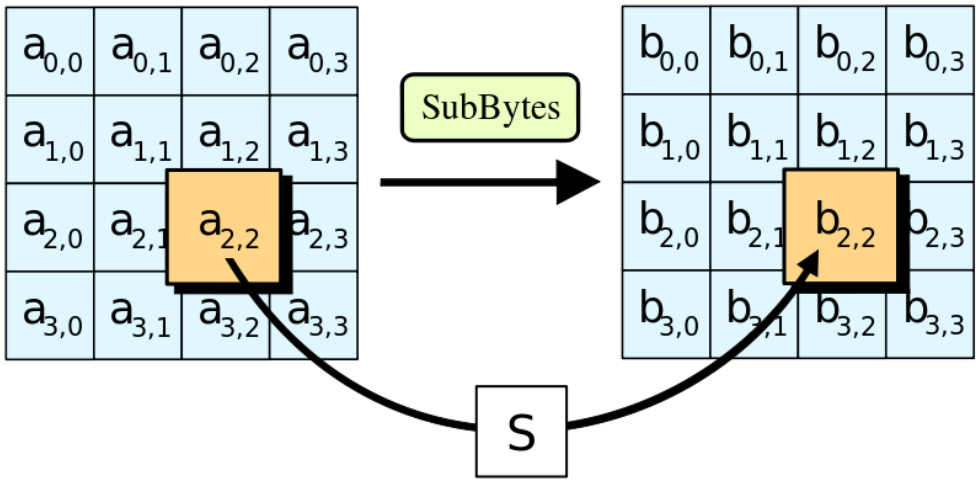

Figure 2.9 – *SubBytes* [65]

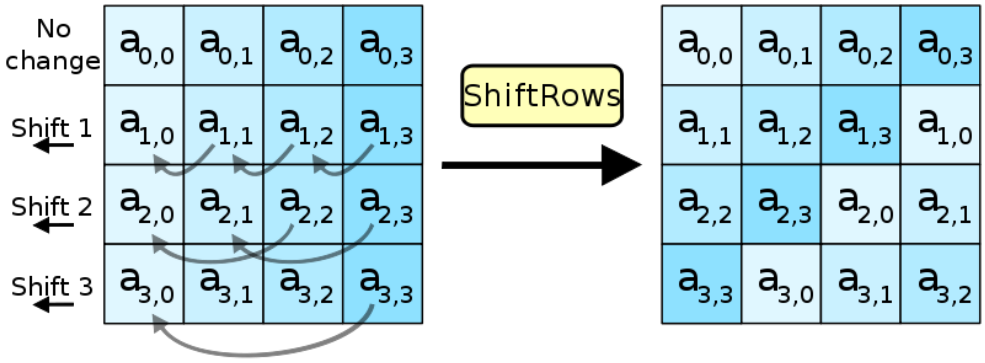

Figure 2.10 – *ShiftRows* [65]

Tableau 2.II – Matrice de Mixcolumns [65]

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

menter au maximum la diffusion dans le système tandis que l'opération *SubBytes* sert à rendre le tout non-linéaire. L'ensemble des informations de cette section provient de [22, 65].

Pour ce qui est de la sécurité, Rijndael a subi des analyses approfondies et continuelles depuis 2000 sous la version AES. [82] Une attaque plus efficace que la fouille

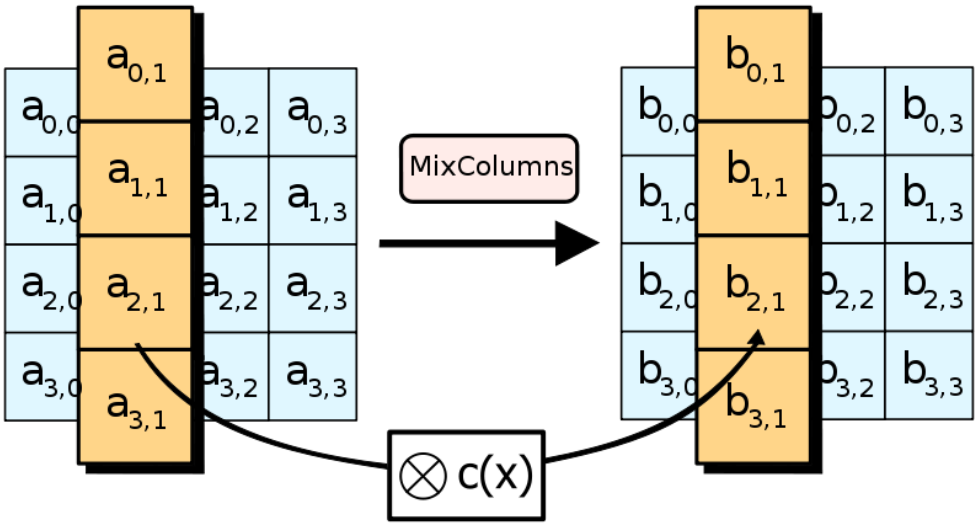

Figure 2.11 – *MixColumns* [65]

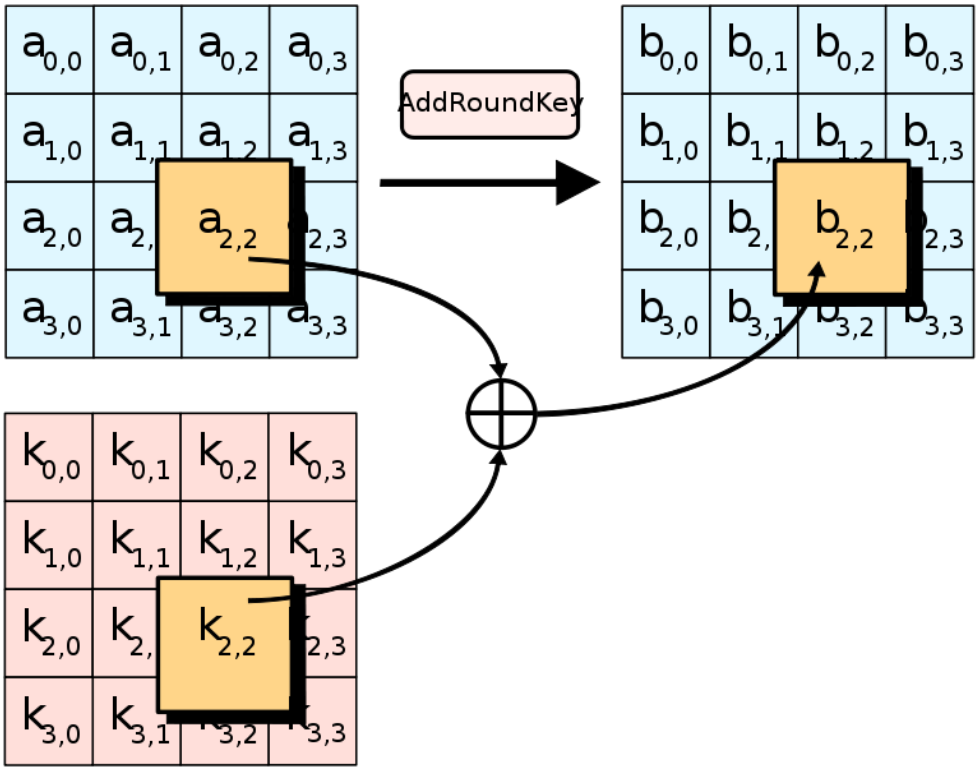

Figure 2.12 – *AddRoundKey* [65]

exhaustive sur la version complète d'AES fut développé, pour la première fois en 2011, par Andrey Bogdanov , Dmitry Khovratovich, et Christian Rechberger. Ils avaient une

stratégie comprenant une complexité de $2^{126.1}$, $2^{189.7}$ et $2^{254.4}$ opérations respectivement pour des clés de 128, 192 et 256 bits. L'attaque était basée sur des bicliques et n'affectait pas la sécurité en pratique de l'algorithme, car la différence avec la force brute était trop petite. [8] Une version d'AES2 a également été présenté dans [9] en 2012.

2.5 Nessie

À la suite du concours AES, deux autres initiatives similaires ont été lancées à travers le monde. La première est le projet NESSIE (*New European Scheme for Signatures, Integrity and Encryption*), qui a été initié en Europe au début de l'an 2000. L'objectif du projet était d'émettre un portfolio comprenant plusieurs primitives cryptographiques qui auraient été obtenues à travers un processus d'appel public avec une analyse transparente. Le projet visait des primitives de chiffrement par bloc (*bloc cipher*), de chiffrement par flots (stream cipher), de fonctions de hachage, d'algorithme de CAM (*MAC*), de signature digitale et de chiffrement à clé publique. [17]

L'objectif à long terme du projet était de maintenir la bonne position de l'Europe dans le domaine de la recherche tout en améliorant la position de l'industrie cryptographique européenne.

Le calendrier du projet était le suivant [17] :

- janvier 2000 : début de la première phase ;
- septembre 2000 : date limite pour les soumissions ;
- novembre 2000 : premier *workshop* de NESSIE ;
- juin 2001 : fin de la première phase ;
- juillet 2001 : début de la deuxième phase ;
- septembre 2001 : deuxième *workshop* de NESSIE ;
- février 2002 : sélection préliminaire des soumissions ;
- novembre 2002 : troisième *workshop* de NESSIE ;
- février 2003 : troisième *workshop* de NESSIE ;
- mars 2003 : sélection finale des soumissions, émissions du rapport final du projet, fin de la deuxième phase.

Le portfolio émis à la fin du projet comptait 12 des 42 algorithmes soumis en plus de cinq standards bien établis. Par contre, aucun des algorithmes de chiffrement par flots soumis ne respectait les critères minimums fixés par le projet. Voici les soumissions retenues en ce qui concerne les protocoles de chiffrement par bloc. La liste complète des recommandations de NESSIE se trouve en annexe III.

- MISTY1 : Mitsubishi Electric ;
- Camellia : Nippon Telegraph and Telephone and Mitsubishi Electric ;
- SHACAL-2 : Gemplus ;
- AES [1] : (Advanced Encryption Standard) (NIST, FIPS Pub 197).

Un autre fait intéressant est que les deux algorithmes basés sur Rijndael, soit Grand Cru et Anubis, n'ont pas passé le premier tour de ce concours. Les informations concernant Nessie ont été obtenues dans [17].

2.6 Cryptrec

La deuxième initiative similaire à AES, initiée en 2000, est CRYPTREC (*Cryptography Research and Evaluation Committees*). Ce projet a été lancé par le gouvernement japonais, qui souhaitait établir une liste des cryptosystèmes et des protocoles cryptographiques qui pourraient être utilisés dans l'élaboration d'un e-gouvernement japonais d'ici 2003. CRYPTREC impliquait les acteurs suivants :

- le ministère des échanges économiques et de l'industrie ;
- le ministère de la gestion publique, des affaires intérieures et de la poste et des télécommunications ;
- l'organisation du progrès en télécommunication (Telecommunications Advancement Organization) ;
- l'agence de promotion des technologies d'information(Information-Technology Promotion Agency).

Contrairement à NESSIE, CRYPTREC était obligé, par ces spécifications, de recommander des systèmes dans chacun des types de protocole. Le projet s'est également

[1] seul algorithme déjà connu

intéressé aux différentes versions possibles des algorithmes afin de recommander correctement leur utilisation.

Le calendrier du projet était le suivant :

- mai 2000 : création du comité d'évaluation des techniques de cryptographie pour l'édition 2000 de CRYPTREC ;
- juin à juillet 2000 : invitation publique pour des soumissions de techniques cryptographiques ;
- août à octobre 2000 : présentation et évaluation des techniques de cryptographie soumises ;
- octobre 2000 : symposium sur les techniques de cryptographie (introduction au but de CRYPTREC) ;
- octobre 2000 à mars 2001 : évaluation détaillée des soumissions ;
- mars 2001 : publication du rapport CRYPTREC 2000 ;
- août à septembre 2001 : invitation publique pour des techniques de cryptographie pour l'édition 2001 de CRYPTREC ;
- octobre 2001 à mars 2002 : évaluation détaillée des techniques soumises ;
- janvier 2002 : "workshop" sur l'évaluation des soumissions ;
- mars 2002 : publication du rapport CRYPTREC 2001 ;
- avril 2002 : publication du rapport annuel de 2001 du comité consultatif de CRYPTREC ;
- avril 2002 à mars 2003 : évaluation détaillée des soumissions ;
- octobre 2002 : publication du rapport CRYPTREC 2001 en version anglaise ;
- février 2003 : introduction des activités de CRYPTREC à NESSIE ;
- 20 février 2003 : publication de la liste des cryptosystèmes recommandés pour un e-Government (MIC, METI) ;
- mars 2003 : publication rapport annuel de 2002 du comité consultatif de CRYPTREC.

En février 2003, Cryptrec a émis l'ensemble de ses recommandations. La liste complète des recommandations de CRYPTREC se trouve en annexe IV. Voici ses recommandations concernant les protocoles de chiffrement par bloc :

- chiffrement par bloc de 64-bit (les chiffrements par bloc de 128 bits sont préférables) ;
  - CIPHERUNICORN-E,
  - Hierocrypt-L1,
  - MISTY1,
  - 3-clés Triple DES (Permis *pour le moment* si utilisé tel que spécifié dans FIPS Pub 46-3, et s'il est spécifié en tant que standard).
- chiffrement par bloc de 128-bits, seulement les chiffrement par bloc de 128-bits sont recommandés ;
  - AES,
  - Camellia,
  - CIPHERUNICORN-A,
  - Hierocrypt-3,
  - SC2000.

À la suite de la publication du rapport listant les algorithmes recommandés, le comité a décidé de continuer ses travaux afin de bien superviser l'implémentation et l'évolution des différents systèmes. CRYPTREC est toujours actif à ce jour. Les informations sur CRYPTREC ont été obtenues dans [21, 66].

CHAPITRE 3

PROPRIÉTÉS MATHÉMATIQUES DÉSIRÉES DES S-BOXES

L'élément le plus important dans un réseau de substitution-permutation est le choix des S-Boxes utilisées. Étant donné que c'est le seul élément non-linéaire de l'ensemble de l'algorithme, la force et le degré de sécurité d'un algorithme de cryptographie à clé privée basé sur un réseau de permutations et de substitutions reposent sur la qualité des S-Boxes. Il existe plus d'un type de S-Boxes :

- S-Boxe fixe [22] : S-Boxes dont le contenu est fixe et connu de tous.
- S-Boxe dépendant d'une clé [31, 34, 47] : S-Boxes dont celle qui est utilisée est définie par la clé choisie par l'utilisateur. L'ensemble des S-Boxes est connu de tous, mais puisque le choix de la clé est confidentiel, la S-Boxe qui est utilisée demande un effort supplémentaire pour être déterminée.
- S-Boxe générée à partir de relations chaotiques(*Chaotic Map*) : plusieurs chercheurs suggèrent d'utiliser des *chaotic maps* dans les systèmes cryptographiques, [85], dont dans la génération des S-Boxes comme présenté dans l'article [58].
- S-Boxe générée à partir de fonctions d'Hadamard : en 1993, Jennifer Seberry, Xian-Mo Zhang et Yuliang Zheng ont proposé des S-Boxes qui furent générées à partir de fonctions d'Hadamard. [55]
- S-Boxe aléatoire [28, 29, 83, 84] : S-Boxes dont les permutations sont choisies aléatoirement. C'est ce type de S-Boxes que nous avons choisi pour notre système. Nous avons choisi ce type de S-Boxes car ses propriétés sont les plus pertinentes et les plus intéressantes pour le protocole que nous proposons.

Au fil des années, quatre caractéristiques principales ont été déterminées comme étant celles qui sont essentielles à une bonne S-Boxe, peu importe son type [1, 2, 23, 43, 52, 64] :

- l'effet d'avalanche(*Avalanche Effect*) ;
- le critère d'indépendance des bits(*Bit Independence Criterion*) ;
- la non-linéarité ;

- la bijection.

Dans les quatre sous-sections qui suivent, nous allons présenter ces caractérisques en détail.

3.1 Effet d'avalanche

L'effet d'avalanche est défini comme suit :

Définition 5. Une fonction $f : \{0,1\}^t \rightarrow \{0,1\}^t$ satisfait le critère d'*effet d'avalanche* si, quand un bit d'entrée est modifié, en moyenne, la moitié des bits de sortie sont modifiés.

C'est un critère très important, car si une S-Boxe ne le satisfait pas, alors le cryptosystème aura un faible taux de diffusion. Or, plus le degré de diffusion d'une S-Boxe est faible, plus il faudra de *rounds* à l'algorithme pour être jugé sécuritaire. Webster et Tavares ont défini, en 1985, dans [64], le critère d'avalanche strict suivant :

Définition 6. Une fonction $f : \{0,1\}^t \rightarrow \{0,1\}^t$ satisfait le *critère d'avalanche strict* si, pour tout i, j appartenant à $\{0,1,...,t-1\}$, modifier le bit d'entrée i modifie le bit de sortie j avec une probabilité exacte de .5.

Ce critère est une généralisation du critère d'avalanche énoncé précédemment qui prend également en compte certains aspects de complétude. [2, 23, 26, 43]

3.2 Critère d'indépendance des bits

Le critère d'indépendance des bits (*Bit Independance Criterion, BIC*), comme le critère d'avalanche, est un autre critère permettant l'analyse de la diffusion d'une S-Boxe. Le *BIC* permet de déterminer si le fait de modifier certains bits d'entrée d'un message crée une tendance prévisible sur les bits de sortie. Dans un contexte idéal, le fait de modifier spécifiquement certains bits du message d'entrée ne devrait pas influencer d'une quelconque façon la probabilité que des bits de sorties spécifiques soient modifiés. On peut définir cela de la manière suivante [2, 23, 43] :

Définition 7. Une fonction $f : \{0,1\}^t \rightarrow \{0,1\}^t$ satisfait le *critère d'indépendance des bits* si, pour tout i, j, k appartenant à $\{0,1,...,t-1\}$ avec j différent de k, inverser le bit d'entrée i a pour effet de modifier indépendamment la valeur des bits de sortie j et k.

Isil Vergili et Melek D. Yücel ont étudié dans [61, 62] ce qui se passerait si nous incluions un pourcentage d'erreur au critère d'indépendance des bits ainsi qu'à celui d'avalanche. Ils ont montré que, plus la taille des S-Boxes choisies aléatoirement est grande, plus il est probable qu'une S-Boxe possède de très bonnes caractéristiques.

3.3 Non-linéarité

La non-linéarité d'une S-Boxe est cruciale, car c'est le seul élément d'un réseau de substitutions et de permutations ou d'un réseau de Feistel qui ne soit pas linéaire. Si la S-Boxe n'a pas un haut niveau de non-linéarité, alors elle sera vulnérable à plusieurs types d'attaques, dont la cryptanalyse linéaire.

Afin de définir la non-linéarité d'une S-Boxe, nous devons d'abord définir celle d'une fonction. Pour ce faire, nous avons besoin des définitions suivantes :

Définition 8. Soit g une fonction, alors g est une *fonction affine* si g est de la forme $(x_1,...,x_t) \rightarrow \sum_i a_i x_i + b$ où $a_i, b \in \{0,1\}$.

Définition 9. Soit A^t l'ensemble de toutes les fonctions affines $g : \{0,1\}^t \rightarrow \{0,1\}$. Soit $f : \{0,1\}^t \rightarrow \{0,1\}$. Alors si nous considérons f et g en tant que chaînes de 2^t bits, nous pouvons définir *la non-linéarité de* f, $nl(f)$, par : $nl(f) = min_{(g \in A^t)}(wt(f \oplus g))$, où wt représente le poids de Hamming et $\oplus$ l'opération ou-exclusif.

La non-linéarité des fonctions qui composent la S-Boxe détermine sa non-linéarité de la manière suivante [2, 23, 43] :

Définition 10. Soit L, l'ensemble de toutes les combinaisons linéaires des colonnes d'une S-Boxe $t \times t$ S, ainsi tout $l \in L$ peut être exprimé en tant qu'une chaîne de 2^t bits représentant le résultat de l'application de l sur les colonnes de S. Alors *la non-linéarité de* S est : $nl(S) = min_{(l \in L \setminus 0)} nl(l)$

Plusieurs approches algébriques ont été développées afin de tenter de linéariser les S-Boxes des systèmes existants pour les briser [18, 30]. Ces approches ont permis de briser certains cryptosystèmes dont *SHARK*, en plus d'affaiblir d'autres procotoles qui étaient jugés auparavant sécuritaires. Il est recommandé de ne pas utiliser des S-Boxes avec un niveau de non-linéarité peu élevé.

3.4 Bijection

Finalement, le dernier critère est essentiel seulement dans le cas des S-Boxes utilisées dans des réseaux de substitutions et de permutations. Dans un réseau de substitutions et de permutations, le fait que la S-Boxe soit bijective, autrement dit qu'elle implémente une bijection, est trivial et essentiel, car il faut que la S-Boxe soit réversible. La réversibilité d'une S-Boxe permet d'effectuer le déchiffrement des messages encodés. Les S-Boxes utilisées dans les réseaux de Feistel n'ont pas à être réversibles, car le déchiffrement ne fonctionne pas de la même manière.

CHAPITRE 4

TECHNIQUES DE CRYPTANALYSE

Les deux principaux types d'attaques contre les réseaux de substitutions et de permutations sont la cryptanalyse linéaire et la cryptanalyse différentielle. Chacune de ces attaques tente de tirer profit d'une faiblesse des S-Boxes, puisque c'est le seul élément non-linéaire du protocole. D'autres attaques basées sur ces deux types de cryptanalyse ont été développées par la suite. Cette section présentera les principaux types d'attaques connus, soit :

- la cryptanalyse linéaire ;
- la cryptanalyse différentielle ;
- la cryptanalyse différentielle tronquée ;
- la cryptanalyse d'ordre supérieur ;
- la cryptanalyse Boomerang ;
- la cryptanalyse XSL.

Il existe également d'autres types d'attaques tels que l'attaque statistique présentée dans [60], qui visent spécifiquement certains cryptosystèmes ou des structures particulières comme celles présentées dans [6, 11].

Il existe cinq niveaux d'attaques lorsque nous analysons la sécurité d'un cryptosystème :

- l'attaque à message chiffré seulement : dans ce mode, l'attaquant ne possède qu'un certain nombre de messages chiffrés sans connaître les messages clairs qui leur sont associés. Typiquement, ce mode sert à simuler la situation où un adversaire réussit à écouter les communications entre les deux parties et qu'il ne peut qu'avoir accès aux messages chiffrés entre eux ;
- l'attaque à message clair connu : dans ce mode, l'attaquant possède des messages chiffrés dont il connaît le message clair initial correspondant ;
- l'attaque à message clair choisi : dans ce mode, l'attaquant peut demander des messages chiffrés correspondant à des messages clairs qu'il a choisis ;

- l'attaque à message chiffré choisi : dans ce mode, l'attaquant obtient également un ensemble de paires de messages chiffrés et de messages clairs, mais cette fois, il choisit les messages chiffrés dont il souhaite obtenir le message clair ;
- l'attaque adaptative : dans ce mode, l'attaquant peut adapter ses demandes de paires de messages clairs ou chiffrés basées sur les paires de messages obtenues précédemment. Chacun des modes d'attaques précédents possède son mode d'attaque adaptative.

Un système est jugé sécuritaire s'il résiste au dernier mode d'attaque énoncé. La plupart des types d'attaques utilisés pour briser des protocoles sont soit des attaques à messages clairs choisis ou des attaques à messages clairs connus. Ces informations sont tirées de [30, 33, 57].

4.1 Cryptanalyse linéaire

La cryptanalyse linéaire a été développée par M. Matsui en 1993. Cette technique a été créée lors de l'analyse de DES. La cryptanalyse linéaire est une attaque à messages clairs connus qui cible, comme mentionné précédemment, la structure des S-Boxe.

Afin d'expliquer plus simplement le fonctionnement de la cryptanalyse linéaire, elle sera décrite dans le contexte d'un réseau de substitutions et de permutations conventionnel tel qu'illustré à la figure 2.1. Il est possible de remarquer qu'entre chaque passage dans les S-Boxes, les bits sont permutés à l'aide d'une permutation fixe qui envoie chaque bit de sortie d'une S-Boxe dans une autre différente. Il y a également l'application d'un ou-exclusif entre les bits sortant des S-Boxes et une clé lors de chaque *round*.

La cryptanalyse linéaire se base sur le fait qu'une S-Boxe ayant le même nombre de bits d'entrée que de bits de sortie admet plusieurs relations linéaires entre des bits d'entrée et des bits de sortie qui sont valides plus de 50% du temps. Le but de la cryptanalyse linéaire est de trouver une équation dans le genre suivant : $M_{i_1} \oplus M_{i_2} \ldots \oplus M_{i_k} \oplus C_{i_1} \oplus C_{i_2} \ldots \oplus C_{i_j} = K_{i_1} \ldots \oplus K_{i_l}$, où les M_{i_k} correspondent au bit k du message i à chiffrer ; les C_{i_j}, au bit j du message i chiffré et les K_{i_l}, au bit l de la $i$ième clé utilisée tout au long du cryptosystème. Une telle équation devrait être véridique 50% du temps dans un cryp-

tosystème idéal. Or, puisqu'il existe des relations linéaires biaisées dans les S-Boxes, il est possible de déterminer et d'exploiter une telle équation.

La stratégie employée pour appliquer la cryptanalyse linéaire est de déterminer une suite de relations linéaires biaisées applicables aux S-Boxes. Lorsque nous les joignons les unes à la suite des autres, nous pouvons attaquer des bits spécifiques du cryptosystème pour l'ensemble de ses *rounds*. Afin de déterminer le biais d'une relation linéaire sur l'ensemble du système à partir du biais des relations linéaires sur les S-Boxes, nous utilisons le lemme du *Piling-up* suivant, présenté par M. Matsui dans [39] :

Lemme 1. *Soit une équation linéaire sous la forme d'un ou-exclusif de variables binaires :* $X_1 \oplus X_2 \oplus ... \oplus X_N = 0$. *Soit N variables aléatoires, indépendantes et binaires (le résultat de l'événement est soit 0, soit 1),*X_1 ,X_2, ..., X_N , *la probabilité que cette équation soit correcte est de* $P(X_1 \oplus X_2 \oplus ... \oplus X_N = 0) = 1/2 + 2^{N-1} \prod_{i=1}^{N} \varepsilon_i$ *avec* ε_i , *le biais linéaire de la variable aléatoire* X_i . *Ce biais peut être positif ou négatif et quantifie l'écart par rapport à une distribution uniforme où l'espérance d'une variable aléatoire binaire est 1/2. [15, 77]*

Par la suite, il suffit de tester sur un grand nombre de messages l'ensemble des valeurs des bits de la dernière clé impliqués dans la relation linéaire pour déterminer leur valeur. Une fois la valeur de ces bits déterminée, nous pouvons recommencer le même processus en ciblant d'autres bits de clé afin de briser totalement le système. Le nombre de messages nécessaires à utiliser pour chaque étape est défini par la relation suivante : $1/(\varepsilon_S)^2$ où ε_S correspond au biais de la relation linéaire sur l'ensemble du système.

Par exemple, prenons la S-Boxe S 4×4 décrite par l'illustration 4.1. Il est possible de remarquer que les relations $\alpha = x_1 \oplus x_3 \oplus x_4 = y_2$ et $\beta = x_2 = y_2 \oplus y_4$ sont vraies 12 fois sur 16. Par conséquent, ces relations créent un biais de 25% chacune sur la S-Boxe. Ce type de biais est inévitable pour des S-Boxes dont le nombre de bits fournis en entrée est égal au nombre de bits de sortie comme présenté dans [25, 41, 48, 56]. Par contre, plus la taille des S-Boxes est grande, plus ce biais diminue .

À partir de la structure d'un réseau de substitutions et de permutations, α devient $x_{1,5} \oplus k_{1,5} \oplus x_{1,7} \oplus k_{1,7} \oplus x_{1,8} \oplus k_{1,8} = y_{1,6}$, où les $x_{i,j}$ correspondent au bit d'entrée j de

Figure 4.1 – S-Boxe

x	y
0000	0110
0001	0100
0010	0111
0011	0001
0100	0010
0101	1111
0110	1001
0111	1010
1000	0011
1001	1000
1010	1110
1011	1100
1100	0101
1101	1011
1110	0000
1111	1101

l'étape de S-Boxes i, les $y_{i,j}$ correspondent au bit j de sortie de l'étape de S-Boxes i, et les $k_{i,j}$ correspondent au bit j de la clé intermédiaire i. La permutation utilisée envoie $y_{1,6}$ en tant que bit d'entrée $x_{2,6}$ après un ou-exclusif avec une deuxième clé. Nous pouvons utiliser β afin d'obtenir $x_{2,6} \oplus k_{2,6} = y_{2,6} \oplus y_{2,8}$. Par la suite, il faut suivre $y_{2,6}$ et $y_{2,8}$, qui deviennent les bits d'entrée de $x_{3,6}$ et $x_{3,14}$ après un ou-exclusif avec la clé du troisième *round*. Pour le *round* suivant, nous obtenons les équations suivantes :

$$x_{3,6} \oplus k_{3,6} = y_{3,6} \oplus y_{3,8}$$

$$x_{3,14} \oplus k_{3,14} = y_{3,14} \oplus y_{3,16}$$

Nous obtenons le système décrit par la figure 4.2 et les relations linéaires suivantes :

$$x_{1,5} \oplus x_{1,7} \oplus x_{1,8} = M_{1,5} \oplus k_{1,5} \oplus M_{1,7} \oplus k_{1,7} \oplus M_{1,8} \oplus k_{1,8} = y_{1,6}$$

$$x_{2,6} = y_{1,6} \oplus k_{2,6} = y_{2,6} \oplus y_{2,8}$$

$$x_{3,6} = y_{2,6} \oplus k_{3,6} = y_{3,6} \oplus y_{3,8}$$

$$x_{3,14} = y_{2,14} \oplus k_{3,14} = y_{3,14} \oplus y_{3,16}$$

En combinant l'ensemble des relations précédentes, nous obtenons la relation sui-

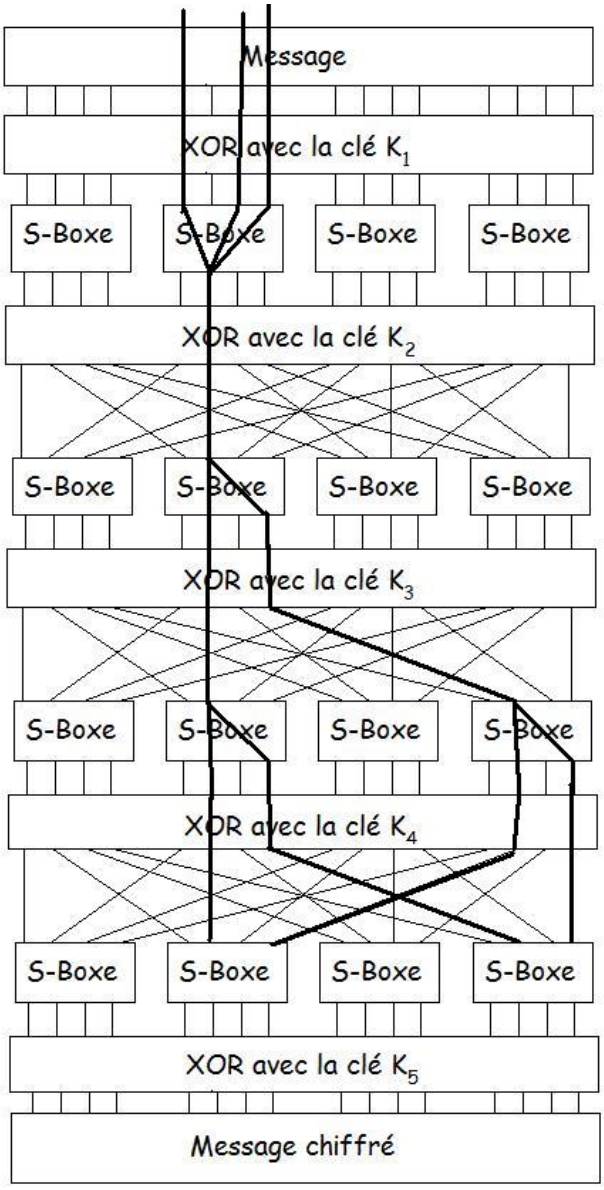

Figure 4.2 – Attaque sur le réseau de substitutions et de permutations

vante : $x_{4,6} \oplus x_{4,8} \oplus x_{4,14} \oplus x_{4,16} \oplus M_{1,5} \oplus M_{1,7} \oplus M_{1,8} \oplus k_{1,5} \oplus k_{1,7} \oplus k_{1,8} \oplus k_{2,6} \oplus k_{3,6} \oplus k_{3,8} \oplus k_{4,6} \oplus k_{4,8} \oplus k_{4,14} \oplus k_{4,16} = 0$, qui peut être modifiée pour obtenir la relation suivante en rassemblant les opérations sur les bits des clés :

$M_{1,5} \oplus M_{1,7} \oplus M_{1,8} \oplus x_{4,6} \oplus x_{4,8} \oplus x_{4,14} \oplus x_{4,16} = k_{1,5} \oplus k_{1,7} \oplus k_{1,8} \oplus k_{2,6} \oplus k_{3,6} \oplus k_{3,8} \oplus k_{4,6} \oplus k_{4,8} \oplus k_{4,14} \oplus k_{4,16} = 0$.

Par le principe de *piling up*, la relation précédente sera vraie $\varepsilon_S = 2*(1/4)^3$ du temps, puisque les relations linéaires utilisées ont la même probabilité et qu'elles sont utilisées sur trois *rounds*.

Afin de déterminer les bits $k_{5,6}, k_{5,8}, k_{5,14}, k_{5,16}$, il faut tester sur un grand nombre de messages l'ensemble des valeurs possibles de ceux-ci. La combinaison dont le taux sera le plus proche de $1/32$ correspondra aux bonnes valeurs des bits $k_{5,6}, k_{5,8}, k_{5,14}$ et $k_{5,16}$.

Ainsi, il est possible de découvrir quatre bits de la quatrième clé intermédiaire en testant les seize clés différentes et en utilisant $1/(((2*1/4)^3)^2) = 1024$ messages différents. Cette attaque est plus efficace que de tenter de briser le réseau par la force brute. Une fois ces quatre bits de clés déterminés, il suffit de recommencer pour les autres bits de cette clé et, ensuite, de remonter pour briser les autres clés en suivant le même principe. Cette section est basée sur les informations contenues dans [15].

4.2 Cryptanalyse différentielle

La cryptanalyse différentielle a été développée par Eli Biham et Adi Shamir en 1991 dans [4]. Lors de leur analyse de la résistance de DES, Biham et Shamir se sont rendus compte que le cryptosystème semblait particulièrement résistant à leur attaque par rapport à d'autres cryptosystèmes basés sur le même modèle. Ce constat a amené des questionnements sur le fait que ce type d'attaques était déjà connu par IBM ou la NSA lors de la création de DES. Cette information a été finalement confirmée par les concepteurs de DES. La NSA a confirmé que l'un des objectifs de conception de DES était qu'il résiste à des attaques dans le style de la cryptanalyse différentielle. À l'interne, cette attaque était nommée la *T-attack* ou *Tickling attack*. Ces informations proviennent de [16, 75].

La cryptanalyse différentielle est une attaque qui se base sur les relations linéaires qui existent dans la *différence* entre les sorties de deux entrées différentes d'une S-Boxe. Le principe de base diffère de la cryptanalyse linéaire, car il consiste à modifier les bits d'entrée du cryptosystème afin de générer une différence prévisible dans les bits du message chiffré et non à analyser directement une relation des bits de sortie du système par

rapport au bit d'entrée. C'est donc une attaque à message clair choisi et non à message connu. L'analyse est principalement centrée sur les S-Boxes, car c'est le seul élément non-linéaire de ce type de protocole. Formellement, la cryptanalyse différentielle tente une *différentielle* qui est définie comme suit :

Définition 11. Soit deux messages M et M' ainsi que leur message chiffré correspondant C et C', alors la paire composée des différences d'entrée($\delta M = M \oplus M'$) et des différences de sortie($\delta C = C \oplus C'$) est appelée *une différentielle* pour le système.

Étant donné que le seul élément non linéaire d'un réseau de substitutions et de permutations est la S-Boxe, cette attaque tente de combiner une séquence de différentielles chez les S-Boxes de différents niveaux qui permettraient de créer une différentielle pour l'ensemble du cryptosystème. Une S-Boxe serait idéale si, pour chaque différence δx de ses bits d'entrée, toutes les valeurs possibles de différence δy étaient équiprobables. Or, si la S-Boxe possède le même nombre de bits d'entrée que de bits de sortie, comme mentionné dans [41, 48, 56], une telle S-Boxe est impossible. Les S-Boxes qui sont les plus près de cet idéal sont des S-Boxes qui implémentent des fonctions courbées(*Bent Function*). Or, les fonctions courbées impliquent qu'il y ait plus de bits d'entrée que de bits de sortie.

Afin de déterminer le biais de la différentielle à partir du biais des différentielles des S-Boxes utilisées, nous pouvons utiliser le lemme du *piling-up* de la même manière que pour la cryptanalyse linéaire. [4, 77] En testant avec plusieurs messages l'ensemble des valeurs possibles pour le bit de la dernière clé impliqué dans la différentielle, il est possible de déterminer les valeurs de ces bits de clés. Le nombre de messages à utiliser, N_m, peut être déterminé par l'équation suivante, qui diffère légèrement de l'équation précédente : $N_m = c/(\varepsilon_S)$, où $c > 0$ est une constance.

Par la suite, tout comme pour la cryptanalyse linéaire, il suffit de répéter l'expérience pour les autres bits de clé pour déterminer les clés utilisées. Nous pouvons utiliser l'exemple suivant en utilisant le réseau de substitutions et de permutations présenté précédemment afin d'illustrer concrètement le fonctionnement de cette attaque. Comme mentionné précédemment, la cryptanalyse différentielle se base sur les différences entre

les bits d'entrée et les bits de sortie des S-Boxes. Pour appliquer la cryptanalyse différentielle, il faut tenter de trouver des relations linéaires entre une variation des bits d'entrée et une variation des bits de sortie. Dans la S-Boxe 4×4 fournie dans l'illustration 4.1, la différence entre les sorties de la S-Boxe lorsque deux entrées diffèrent de 1010 est 0010 ainsi que 0101 lorsque la différence est de 0010. La première différence se produit trois fois sur seize alors que la deuxième se produit quatre fois sur seize. Le tableau de la figure 4.I représente certaines de ses relations linéaires. Les deux relations mentionnées sont également indiquées en surbrillance.

Tableau 4.I – Tableau caractéristique différentielle

X	Y	Variation de Y			
		$\Delta X = 0001$	$\Delta X = 0010$	$\Delta X = 0011$	$\Delta X = 1010$
0000	0110	0010	0001	0111	1000
0001	0100	0011	0101	0110	1000
0010	0111	0110	0101	1000	0010
0011	0001	0011	1110	1000	1010
0100	0010	1101	1011	1000	0010
0101	1111	0110	0101	1100	0010
0110	1001	0011	1010	0001	1111
0111	1010	1001	0010	0100	1110
1000	0011	1011	1101	1111	0100
1001	1000	0110	0100	1101	1001
1010	1110	0010	1011	0101	1100
1011	1100	1001	0111	1100	0011
1100	0101	1110	0101	1000	1100
1101	1011	1011	0110	1101	0001
1110	0000	1101	0110	0100	0011
1111	1101	1011	1001	1010	0101

La procédure pour profiter de cette relation linéaire est similaire à ce qui est fait pour la cryptanalyse linéaire sauf qu'au lieu de parcourir le protocole en suivant les bits impliqués dans la relation linéaire, il faut suivre les bits altérés par la modification des bits d'entrée. Si nous prenons l'exemple illustré précédemment et que nous choisissons de prendre des paires de messages dont la différence se situe entre les neuvièmes et douzièmes bits, nous obtiendrons le schéma d'attaque différentielle de la figure 4.3.

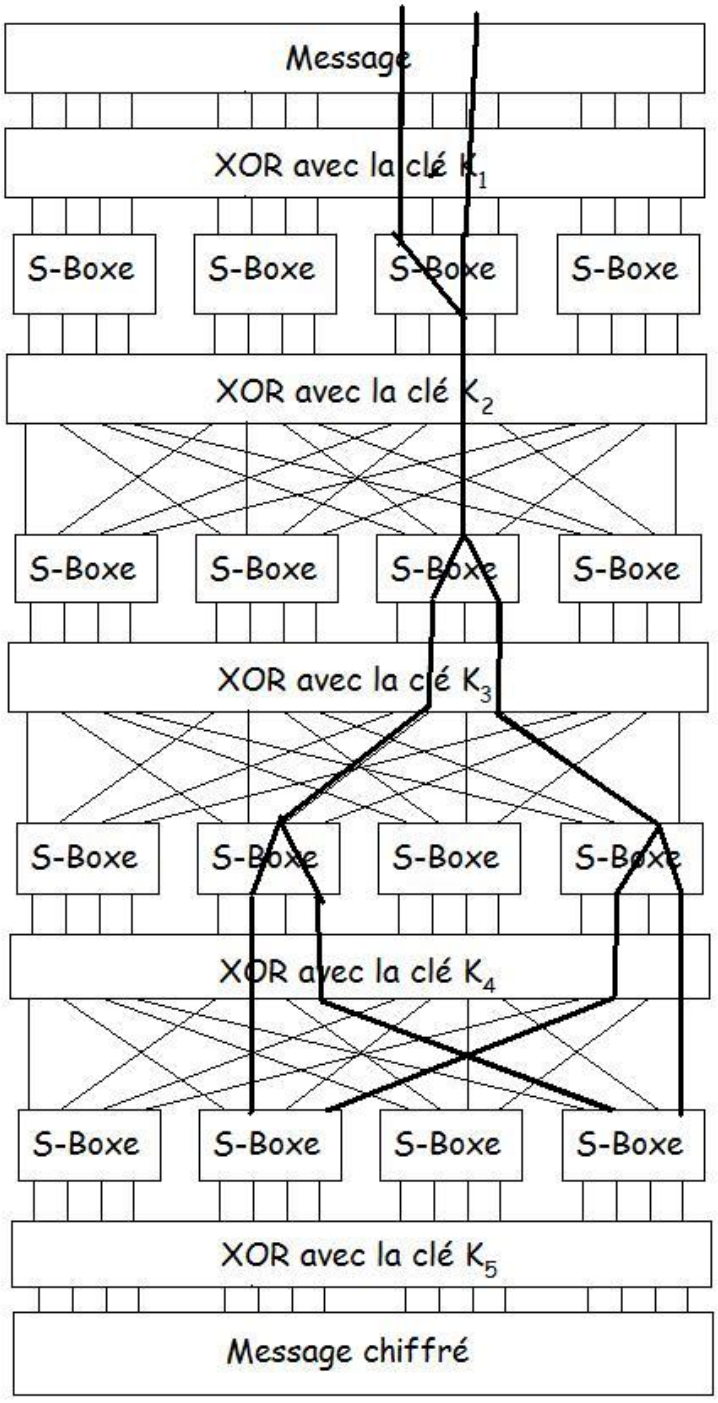

Figure 4.3 – Attaque différentielle

Dans ce schéma, il est possible de voir que, lors d'une première attaque de cryptanalyse différentielle, nous aurions une probabilité de $(3/16)*(4/16)*(4/16)*(4/16) = 3/1024$ de déterminer quatre bits de la dernière clé. Cette probabilité est calculée à partir du lemme du *Piling-up* de la même manière que pour la cryptanalyse linéaire. Par conséquent, il faudrait au plus $(1/((3/16)*(1/4)^3)) = 342$ messages si $c = 1$ afin de briser le protocole. Tout comme pour la cryptanalyse linéaire, ce type d'attaques est plus efficace que la force brute et, une fois un certain nombre de bits de clé brisés, il suffit de recommencer le même procédé pour l'ensemble des bits des différentes clés utilisées.

Il fut montré dans [7] qu'il est possible de diminuer le nombre de messages nécessaires en utilisant plusieurs caractéristiques différentielles qui ont les mêmes entrées mais pas les mêmes sorties.

4.2.1 Cryptanalyse différentielle tronquée

La cryptanalyse différentielle tronquée (Truncated differential cryptanalysis) est une généralisation de la cryptanalyse différentielle développée par Lars R. Knudsen en 1994. Il a développé une technique de cryptanalyse qui exploite la possibilité de prédire certains bits seulement du message chiffré. Il a prouvé dans [35] le résultat suivant :

Théorème 1. *Soit* $f(L,R,K) : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n \times \{0,1\}^n$, *la fonction de round (2) d'un réseau de Feistel de cinq rounds avec des blocs L et R de n bits et utilisant cinq clés de round indépendantes de n bits. Soit a, la différence d'entrée non-nulle pour laquelle seulement une portion W de l'ensemble des différences de sorties est possible, alors il est possible d'effectuer une attaque différentielle tronquée d'une complexité de 2L messages choisis et avec un temps d'exécution de* $Lx2^{2n}$, *où L est le plus petit entier tel que* $(W)^L < 2^{-2n}$. *La valeur de L est au plus de* 2^{n+1}.

Cette attaque peut être généralisée pour fonctionner sur des cryptosystèmes avec un nombre arbitraire de *rounds* en incluant toutes les clés de *round* sauf les trois premières [35]. Cette nouvelle technique a été testée en premier sur DES avec six *rounds* avant d'être appliquée sur plusieurs autres versions réduites d'autres cryptosystèmes. Il a été possible de trouver la clé secrète de la version de DES comprenant six *rounds* avec 46

messages choisis dans un temps équivalent à l'exécution de 3,500 chiffrements. Cette attaque ne fonctionnait pas sur la version complète de DES. [35]

Les autres cryptosystèmes, [35, 79], sur lesquels cette technique a été appliquée sont

- SAFER : attaque réussie sur une version réduite de 5 *rounds* [36] ;
- IDEA : attaque réussie sur une version réduite de 3.5 *rounds* [10] ;
- Skipjack : attaque réussie sur une version réduite de 16 *rounds* [37] ;
- E2 : attaque réussie sur une version réduite de 8 *rounds* [40] ;
- Twofish : attaque réussie sur *Twofish* avec 16 *rounds* [44] ;
- Camellia : attaque réussie sur une version réduite de 8 *rounds* [38] ;
- Salsa20 : attaque réussie sur une version réduite de 5 *rounds* [20] .

4.2.2 Cryptanalyse différentielle d'ordre supérieur

La cryptanalyse différentielle d'ordre supérieur a également été développée par Lars R. Knudsen et fut présentée en même temps que la cryptanalyse différentielle tronquée en 1994. Cette technique a été développée à la suite de la publication des travaux de Xuejia Lai sur les « dérivations d'ordre supérieur » (*Higher-Order derivative*) également en 1994.

Le principe de l'attaque est identique à celui de la cryptanalyse différentielle sauf que cette technique de cryptanalyse exploite des caractéristiques différentielles d'ordre supérieur à 1. La cryptanalyse différentielle exploitait des caractéristiques différentielles de premier ordre. Lai a démontré qu'il était possible de déterminer des caractéristiques différentielles d'ordre supérieur. Une caractéristique d'ordre supérieur pour un *round* peut être définie de la manière suivante [35] :

Définition 12. Une caractéristique différentielle d'ordre i pour un *round* est un $(i+1)-tuple(\alpha_1,...,\alpha_i,\beta)$, tel que $\Delta^{(i)}_{\alpha_1,...,\alpha_i} f(x) = \beta$, où $\Delta^{(i)}_{\alpha_1,...,\alpha_i} f(x)$ représente la $i$ème dérivée de f au point $\alpha_1,...,\alpha_i$.

La cryptanalyse différentielle d'ordre supérieur se sert de ces caractéristiques pour briser des cryptosystèmes. Il a également obtenu le résultat suivant :

Résultat 1. *Soit* $f(L,R,K) : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n \times \{0,1\}^n$, *la fonction de round(2) d'un réseau de Feistel de cinq rounds avec des blocs L et R de n bits et utilisant cinq clés de round indépendantes de n bits. Assumons que l'ordre non-linéaire de* f *est r. Alors, une attaque de cryptanalyse différentielle d'ordre r a besoin de* 2^{r+1} *messages choisis et d'un temps d'exécution de* 2^{2n+r}.

Cette technique ne fonctionne malheureusement pas sur tous les réseaux de Feistel de plus de cinq *rounds*. Elle a par contre permis entre autres de briser CAST et les cryptosystèmes KN. [35]

4.3 Cryptanalyse Boomerang

La cryptanalyse Boomerang est une technique inspirée de la cryptanalyse différentielle développée par David Wagner en 1999. Elle a été utilisée, la première fois, pour briser le cryptosystème COCONUT98. Cette technique a ensuite été jumelée à la cryptanalyse différentielle tronquée pour attaquer les cryptosystèmes Khufu et CAST-256.[42]

La cryptanalyse Boomerang nécessite quatre messages, P, P', Q, Q', ainsi que leur chiffrement respectif, C, C', D, D'. Soit $E()$, la fonction de chiffrement du cryptosystème que nous décomposons de la manière suivante : $E() = E1 * E0$, où $E0$ représente la première moitié de l'algorithme de chiffrement et $E1$, la dernière moitié. Il faudra également deux caractéristiques différentielles, soit $\alpha \rightarrow \alpha*$, associé à $E0$ et $\delta \rightarrow \delta*$, associé à $E1^{-1}$. La première caractéristique sera associée à la paire de messages P, P' et la deuxième, aux paires de messages P, Q et P', Q'. Avec ces caractéristiques, il est possible d'affirmer que la paire Q, Q' aura la caractéristique $\alpha* \rightarrow \alpha$ sur l'opération $E0^{-1}$. Ces relations sont illustrées dans la figure 4.4

Il est suggéré de définir les messages P, P', Q et Q' de la manière suivante :

- $P' = P + \alpha$;
- $C = E(P)$;
- $C' = E(P')$;
- $D = C + \delta$;
- $D' = C' + \delta*$;

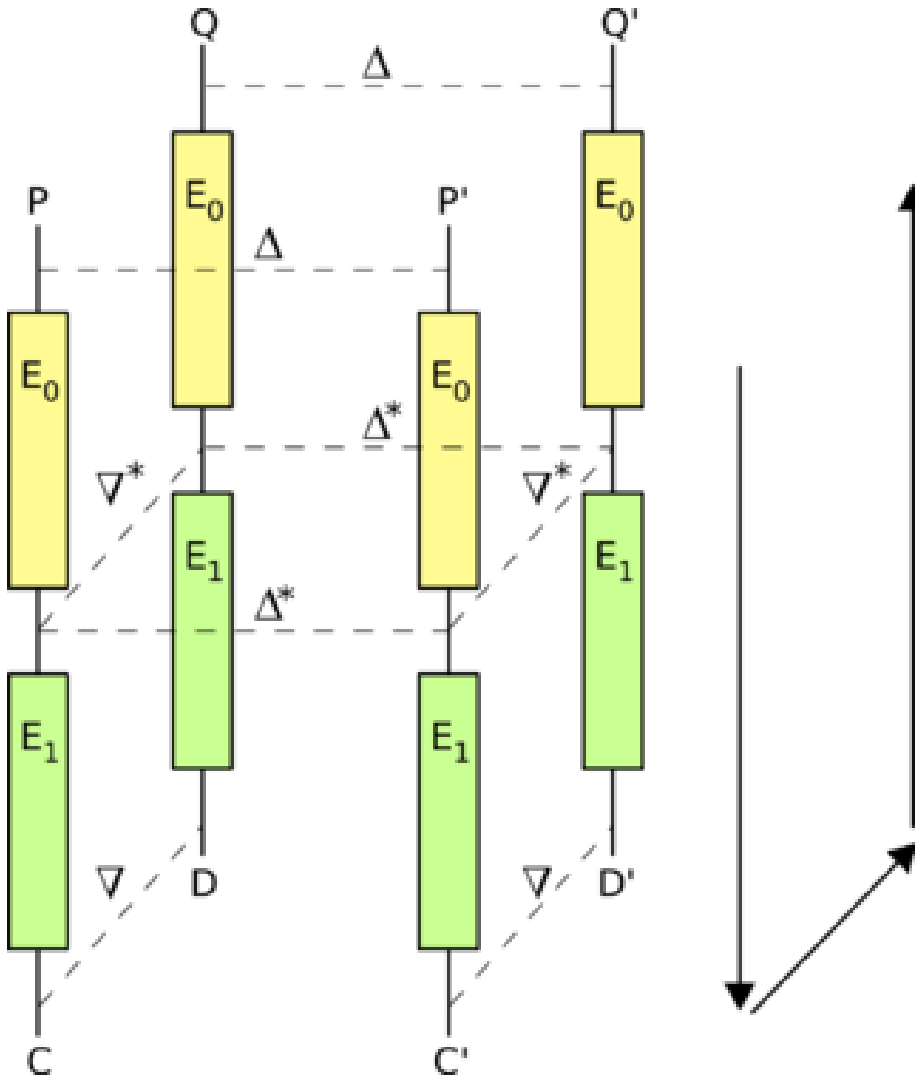

Figure 4.4 – Attaque Boomerang [74]

- $Q = E^{(}-1)(D)$;
- $Q' = E^{(}-1)(D')$.

Le tableau 4.II représente le niveau de complexité des attaques de cryptanalyse Boomerang contre COCONUT98, Khufu, CAST-256 et FEAL.

L'attaque de cryptanalyse Boomerang a, par la suite, été améliorée par John Kelsey, Tadayoshi Kohno et Bruce Schneider en 2000 pour être appliquée aux cryptosystèmes MARS et Serpent. Ils ont nommé leur attaque « l'attaque Boomerang amplifiée (Amplified Boomerang Attack) ».

Tableau 4.II – Complexité de l'attaque Boomerang [63]

Cryptosystème	Nombre de *rounds*	Cryptanalyse Boomerang	
		Complexité au niveau des données	Complexité en temps
COCONUT98	8	2^{16}	2^{38}
Khufu	16	2^{18}	2^{18}
CAST-256	16	$2^{49,9*}$	$2^{49.3*}$
FEAL	6	4	-

* cette attaque a été effectuée en mode message clair connu au lieu du mode message chiffré choisi adaptatif

En 2001, Eli Biham, Orr Dunkelman et Nathan Keller ont développé une nouvelle attaque basée sur l'attaque de cryptanalyse Boomerang qu'ils ont nommée « l'attaque Rectangle(*Rectangle Attack*) » pour briser le cryptosystème Serpent. [63, 74]

4.4 Cryptanalyse par linéarisation éparse étendue

En 2002, Nicolas T. Courtois et Josef Pieprzyk ont développé une nouvelle technique de cryptanalyse basée sur la résolution d'équations quadratiques complexes et simultanées qu'ils ont publiée dans Nature et qu'ils ont nommée *eXtended Sparse Linearization(XSL) attack*. Ils prétendaient que leur attaque pouvait briser AES plus rapidement que la force brute. Il a par la suite été démontré que cette nouvelle attaque n'était pas plus rapide que la recherche exhaustive, même si elle nécessitait que très peu de messages pour être effectuée. [19, 73]

Courtois et Pieprzyk ont testé leur attaque sur les cryptosystèmes Serpent et AES, car ceux-ci avaient respectivement de petites S-Boxes ou des propriétés algébriques spécifiques. Le but de la cryptanalyse XSL est d'exprimer un cryptosystème dans une série d'équations quadratiques avec de multiples variables. Par la suite, il suffit de résoudre le système d'équations pour retrouver les clés du système et ainsi le briser.

Le problème de résoudre des équations quadratiques à variables multiples (Multivariate Quadratic problem(MQ-problem)) est connu depuis longtemps dans les domaines mathématique et cryptographique. Le *MQ-problem* est défini comme un problème NP-

dur, et la sécurité de quelques cryptosystèmes est basée sur cette hypothèse. Il a par contre été démontré, à AsiaCrypt, en 2000, par Adi Shamir, Jacques Patarin, Nicolas Courtois et Alexander Klimov que ce niveau de complexité est substantiellement réduit lorsque le système d'équations est surdéfini (overdefined), c'est-à-dire qu'il y a plus d'équations que d'inconnus. L'article de 2002 de Courtois et Pieprzyk indique que cette complexité diminue davantage si le système est *sparse* et s'il possède une structure régulière. [19]

Les auteurs ont défini une classe de cryptosystèmes, les XSL-cryptosystèmes, qui sont définis de la manière suivante :

- X : le premier *round* $i = 1$ commence avec un ou-exclusif entre le message et la clé de session K_{i-1} ;
- S : ensuite, une étape de B S-Boxes bijectives est appliquée en parralèle ;
- L : finalement, une diffusion linéaire est appliquée.

Un XSL-cryptosystème répète ces trois étapes pendant R *rounds* et termine par l'étape X. Les cryptosystèmes Serpent et AES possèdent justement ces caractéristiques, d'où l'intérêt de tester ces méthodes sur ces cryptosystèmes. La complexité de l'attaque XSL sur AES avec 128 bits de clé est de l'ordre de 2^{230} et de l'ordre de 2^{143} sur Serpent. Par conséquent, cette attaque n'est pas plus efficace que la recherche exhaustive sur AES-128, mais elle permet de briser Serpent avec 192 et 256 bits de clé. [19]

CHAPITRE 5

PROTOCOLE *SAND*

Notre protocole, *SAND*, est un algorithme de chiffrement à clé symétrique basé sur un réseau de substitutions et de permutations. Les deux principales composantes de notre système sont l'utilisation d'une même grande S-Boxe aléatoire ainsi qu'un ensemble de transformations linéaires également aléatoires.

L'ensemble de transformations linéaires aléatoires correspond au groupe général linéaire $GL(n,\mathbb{F}_2)$, où $\mathbb{F}_2$ est le groupe à deux éléments avec l'opération ou-exclusif. Les éléments de $GL(n,\mathbb{F}_2)$ peuvent être représentés par des matrices inversibles $n\times n$ sur $\mathbb{F}_2$. *SAND* utilise une transformation $T \in GL(n,\mathbb{F}_2)$ choisie aléatoirement parmi tous les éléments de $GL(n,\mathbb{F}_2)$ où n correspond à la taille du message à chiffrer. *SAND* est illustré à la figure 5.1.

Les variables suivantes sont associées au protocole et seront utilisées dans le reste du document :

- N : longueur des blocs des messages à chiffrer ;
- S : la S-Boxe utilisée dans le cryptosystème ;
- m : le bloc ou message à chiffrer ;
- m_i : le ième bit du bloc à chiffrer ;
- c : le message chiffré ;
- c_i : le ième bit du bloc de chiffrement ;
- k : la clé utilisée dans le cryptosystème ;
- k_i : le ième bit de la clé utilisée ;
- T : l'élément choisi aléatoirement de $GL(N,\mathbb{F}_2)$ utilisé par le cryptosystème ;
- R : le nombre de *rounds* du cryptosystème ;
- L : le nombre de S-Boxes utilisées ;
- n : taille des S-Boxes (nombre de bits d'entrée).

L'algorithme est composé de trois étapes principales.

1. Xor entre le message et la clé : cette étape consiste à effectuer un ou-exclusif entre

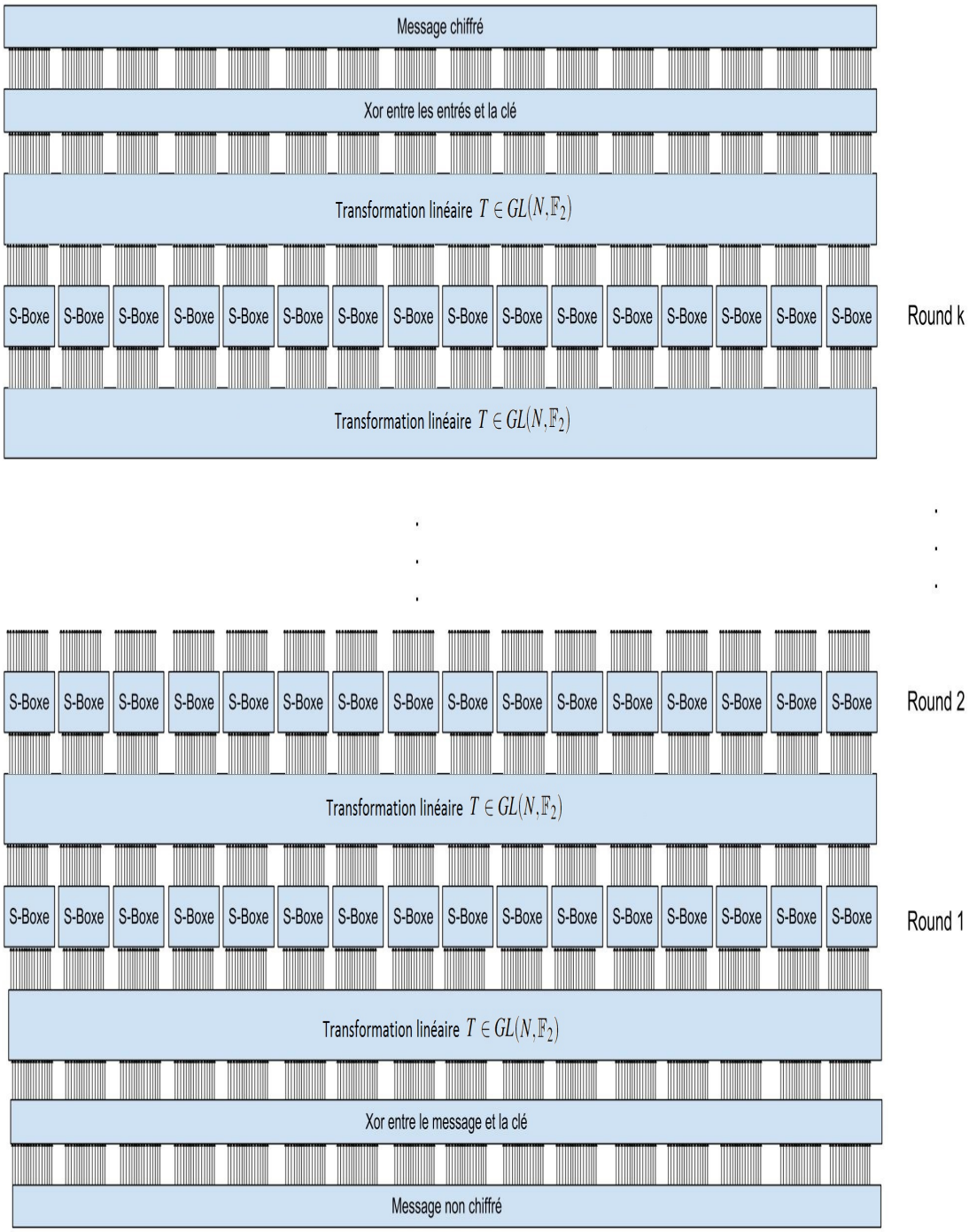

Figure 5.1 – *SAND*

les bits d'entrée (soit le message au début ou le résultat des différentes étapes par la suite) et la clé du système. Cette étape n'est effectué que deux fois soit au début et à la fin du protocole.

2. L'ensemble de fonctions linéaires : cette deuxième étape consiste à appliquer $T \in GL(N, \mathbb{F}_2)$ sur le vecteur de bits fourni à cette étape. Le résultat nous donne le vecteur de sortie correspondant où chaque bit du vecteur correspond au résultat d'une transformation linéaire des bits d'entrée.
3. S-Boxe : la troisième étape consiste à transmettre chaque bit du vecteur de sortie de l'étape précédente à l'entrée d'une série de L S-Boxes identiques dont le fonctionnement interne a été décidé aléatoirement et uniformément parmi toutes les permutations possibles.

Un *round* correspond à l'application des étapes 2 et 3. L'étape 1 n'est effectuée qu'avant le premier *round* et à la fin du dernier *round*. *SAND* peut donc être défini formellement de la manière suivante :

Définition 13. *SAND*, en tant que protocole de chiffrement, est défini de la manière suivante : $E_K(M) = K \oplus (T \circ ((\underset{L}{\times} S) \circ T)^R)(K \oplus M)$ où $(\underset{L}{\times} S) = S \times ... \times S$ correspond à l'application des L S-Boxes.

La taille de la matrice de l'étape 2 est égale à la taille de la clé utilisée à l'étape 1. Pour une clé de 256 bits, 65 536 bits (64Kb) aléatoires sont nécessaires pour exprimer la matrice. La longueur des clés doit être un multiple de la largeur des S-Boxes choisies. Au coeur de l'algorithme se trouve la S-Boxe aléatoire. Celle-ci est choisie parmi toutes les fonctions réversibles possibles et sa description nécessitera une grande quantité de mémoire. Notre algorithme possède au moins trois versions définies par la taille des S-Boxes utilisées. Nous proposons des S-Boxes de 16, 24 ou 32 bits. De telles S-Boxes nécessitent respectivement 256Kb, 96Mb et 32 Gb de mémoire pour être décrites dans le processus de chiffrement. Nous devrons garder en mémoire ou sur un disque dédié la S-Boxe de chiffrement, la transformation linéaire, l'inverse de la S-Boxe ainsi que l'inverse de la transformation linéaire pour le déchiffrement. Afin de déterminer l'inverse de la S-Boxe, il suffit d'effectuer un tri sur les sorties de la S-Boxe de chiffrement pour obtenir la

S-Boxe de déchiffrement ordonnée. L'effort pour obtenir la S-Boxe de déchiffrement est donc de $O(nlog(n))$. En ce qui concerne l'inversion de l'ensemble des transformations linéaires, nous pouvons l'obtenir par la technique d'élimination de Gauss, puisque l'ensemble peut être représenté par une matrice et que, par définition, celle-ci est inversible. Cette technique possède une complexité de $O(N^3)$.

Avant de considérer l'utilisation d'une élément de $GL(N, \mathbb{F}_2)$, nous avons étudié les propriétés d'un protocole utilisant des transformations de type *butterfly*, qui correspond à la transformation représentée à la figure 2.1. Or, après analyse, nous avons constaté que ce type de transformations serait très vulnérable à la cryptanalyse linéaire et différentielle. Nous avons donc jugé important de la modifier afin d'avoir un système plus résistant. Ce constat a également été fait par d'autres spécialistes qui ont étudié et suggéré d'autres types de transformations pour cette étape dans les réseaux de substitutions et de permutations. C'est d'ailleurs pour cette raison que AES n'utilise pas le *butterfly* mais une transformation linéaire spécifique conçue pour atteindre un très haut niveau de diffusion à travers tout le système.

5.1 Utilisation en tant que cryptosystème

En tant que cryptosystème, les détails de l'algorithme gagne à être rendus publics comme c'est toujours le cas en cryptographie, mais les utilisateurs peuvent choisir de conserver les choix aléatoires confidentiels sans nuire à la sécurité. L'ensemble de l'algorithme est conçu de manière à être public et connu de tous à l'exception des clés utilisées. Il sera analysé dans le contexte où le contenu des transformations linéaires et des S-Boxes est connu de l'adversaire. Un utilisateur peut par contre garder secrets les choix aléatoires qu'il a effectués pour la transformation linéaire et les S-Boxes sans que cela ne contredise le principe de Kerckhoffs. Le principe de Kerckhoffs stipule que la sécurité d'un cryptosystème ne doit pas reposer sur le fait que des éléments autres que sa clé sont tenus secrets. *SAND* est le seul que nous connaissons qui comporte la possibilité de garder d'autres éléments secrets sans qu'ils ne contredisent le principe de Kerckhoffs. La sécurité du système dépend du fait que les choix sont aléatoires et elle ne dépend pas

du choix spécifique lui-même.

La particularité de notre algorithme consiste à utiliser une structure particulière et à introduire l'utilisation de fonctions aléatoirement choisies. La composante aléatoire de l'algorithme étant incompressible, nous obtenons un algorithme simple à décrire mais dont la représentation informatique nécessite une grande quantité de mémoire. Il est important de noter que l'efficacité de l'algorithme reste tout à fait acceptable étant donné que l'étape la plus utilisée est un accès au tableau, aussi appelé *table lookup*. La structure simple de l'algorithme permet à un utilisateur ayant choisi ses propres fonctions aléatoires d'être assuré qu'aucune porte cachée ou secrète n'a été insérée dans le protocole.

Le nombre de *rounds* à effectuer est laissé à la discrétion de l'utilisateur. Il est toutefois nécessaire qu'il y ait au moins cinq *rounds*, car sinon, il est possible d'attaquer le système et de le briser avec une attaque plus efficace que la force brute. Ces attaques seront expliquées dans le chapitre suivant.

5.2 Utilisation en tant que fonction de hachage

SAND peut également être utilisé en tant que fonction de hachage cryptographique. Dans une fonction de hachage, toutes les parties de l'algorithme doivent être connues de tous. Par conséquent, la clé, les S-Boxes ainsi que l'ensemble des fonctions aléatoires sont connus. Nous pouvons donc utiliser 0 en tant que clé puisqu'une fois qu'elle est connue, il est possible de factoriser la clé de l'algorithme. Sa valeur n'a donc plus d'effets ni d'importance pour le système. Afin de définir formellement une fonction de hachage cryptographique, nous devons d'abord définir une fonction de hachage [33] :

Définition 14. Une *fonction de hachage* est une paire d'algorithmes en temps polynomial probabiliste (Gen, H) qui satisfait les conditions suivantes :

- *Gen* est un algorithme probabiliste qui prend en entrée un paramètre de sécurité 1^b et émet en sortie une clé s. Nous assumons que 1^b est implicite dans s.
- Il existe un polynôme l telle que H prend en entrée une clé s et une chaîne de caractère $x \in \{0,1\}^*$ et émet en sortie une chaîne de caractère $H^s(x) \in \{0,1\}^{l(b)}$, où b est la valeur du paramètre de sécurité implicite dans s.

Ensuite, afin de définir une fonction de hachage cryptographhque, nous devons définir l'expérience suivante [33] :

Définition 15. L'expérience de détection de collisions : $Hash-coll_{A,\Pi}(b)$:

1. Une clé s est générée en exécutant $Gen(1^b)$.
2. L'adversaire A reçoit s et produit les sorties x,x'.
3. La sortie de l'expérience est 1 si, et seulement si, $x \neq x'$ et que $H(x) = H(x')$. Dans ce cas, nous disons que l'adversaire A a trouvé une collision.

Nous pouvons maintenant définir une fonction de hachage cryptographique [33] :

Définition 16. Une fonction de hachage $\Pi = (Gen, H)$ est une *fonction de hachage cryptographique* si elle est résistante aux collisions. Autrement dit, si $\forall$ adversaire en temps polynomial probabiliste A, il existe une fonction *negl* négligeable tel que $Pr[Hash-coll_{A,\Pi}(b) = 1] \leq negl(b)$, où $Pr[x]$ correspond à la probabilité que x soit vrai.

Nous pouvons utiliser notre système afin de définir la fonction de hachage suivante :

Définition 17. *Fonction de hachage :* Soit *Gen* un algorithme probabiliste générant une clé et soit $h(x) := \{0,1\}^l \rightarrow \{0,1\}^k = [f(x,0)]^k$, où $k < N$, $l < N$ et que f représente *SAND*. Alors $(Gen, h(x))$ est une fonction de hachage utilisant *SAND*.

Par conséquent, afin d'utiliser notre cryptosystème en tant que fonction de hachage, il suffit de rembourrer (*padder*) l'entrée avec $N-l0$ et de ne pas divulguer $N-k$ bits de sortie du système. Ainsi, il devient impossible de déterminer l'ensemble des bits d'entrée avec seulement une partie des bits de sortie.

En ce qui concerne les collisions, il nous semble très difficile de déterminer deux entrées différentes de h qui auront la même sortie. Pour obtenir une collision, il faut déterminer deux sorties de f dont les k premiers bits sont identiques et dont les $N-l$ derniers bits d'entrée sont 0. Avec les propriétés de diffusion de notre algorithme, qui seront décrites en détail à la section 6.1, déterminer une telle paire de messages nécessite de tester l'ensemble des 2^{N-k} possibilités par force brute. Le niveau de complexité des collisions est donc déterminé par le nombre de bits de sortie de notre cryptosystème qui ne sont pas divulgués par la fonction de hachage.

5.3 Implémentation

Au niveau de l'efficacité en pratique, nous n'avons pas effectué d'implémentations optimisées et efficaces de *SAND*. Par contre, nous pouvons quand même analyser la complexité de l'algorithme. Comme il a été mentionné précédemment, il est nécessaire de garder en mémoire ou sur un disque la S-Boxe de chiffrement, la transformation linéaire, l'inverse de la S-Boxe ainsi que l'inverse de la transformation linéaire pour le déchiffrement. Si nous optons pour des blocs de 256 bits avec des S-Boxes de 16 bits ainsi que des blocs de 512 bits avec des S-Boxes de 32 bits, alors il faut respectivement 128Kb et 512Kb pour conserver les tableaux des transformations linéaires ainsi que leurs inverses. Les S-Boxes demanderont 2Mb et 256Gb respectivement. Ensuite, à chaque *round*, il y aura L appels au tableau des S-Boxes pour obtenir la bonne substitution. Le vecteur de sortie des L S-Boxes subit, par la suite, un produit matriciel avec la matrice des transformations linéaires. Ce produit matriciel est dans $O(N^2)$ opérations. Par conséquent, un *round* nécessite L appels de S-Boxes et $O(N^2)$ opérations.

CHAPITRE 6

PROPRIÉTÉ DU PROTOCOLE

Notre algorithme possède plusieurs propriétés qui en font un cryptosystème simple et intéressant à utiliser. Tout d'abord, comme indiqué précédemment, le protocole possède une structure simple, mais nécessite une grande quantité de mémoire. De plus, la description des S-Boxes est incompressible au sens de Kolmogorov, car la permutation qu'elles réalisent est choisie aléatoirement.

Il possède les propriétés suivantes :

- S-Boxe bijective ;
- S-Boxe respectant l'effet d'avalanche ;
- S-Boxe hautement non-linéaire ;
- S-Boxe respectant le critère d'indépendance des bits ;
- propriété de diffusion optimale du système atteinte dès le premier *round* (6.1) ;
- résistance à la cryptanalyse linéaire (6.2) ;
- résistance à la cryptanalyse différentielle (6.3) ;
- résistance aux attaques de biais (6.5).

Ensuite, au niveau des S-Boxes, il est facile de remarquer qu'elles sont bijectives, car nos S-Boxes sont $n \times n$ et réversibles. Par conséquent, il n'existe qu'une seule et unique entrée pour chaque sortie possible des S-Boxes.

En ce qui concerne l'effet d'avalanche, puisque les permutations des S-Boxes sont déterminées aléatoirement, l'espérance du nombre de bits de sortie, modifié par la modification d'un bit, est de $n/2$, ce qui correspond à la moitié des bits.

Nos S-Boxes sont également non linéaires, car elles sont choisies aléatoirement. Si la distance de Hamming de nos S-Boxes par rapport à n'importe quelle fonction affine était faible, alors cela impliquerait qu'il serait possible de compresser la description des S-Boxes. En général, dans l'analyse des propriétés des S-Boxes, le principe simple qui sera utilisé sera celui qui dicte qu'une chaîne aléatoire peut être compressée de D bits avec probabilité au plus de $1/2^D$. Ce principe représente le coeur de plusieurs résultats

présents dans ce mémoire. Le niveau d'incompressibilité de la description des S-Boxes est expliqué dans la section 6.5. Afin que le poids de Hamming des S-Boxes soit proche d'une fonction affine, il faudrait pouvoir compresser le tableau de $O(2^{31})$ bits. La probabilité que nous puissions effectuer ce niveau de compression est de $O(1/(2^{2^{31}}))$.

Le *BIC*, 3.2, est également respecté par nos S-Boxes. Encore une fois, puisque les permutations sont choisies aléatoirement, chaque bit de sortie d'une S-Boxe agit indépendamment de l'ensemble des autres bits de sortie. Si ce n'était pas le cas, alors il serait possible d'utiliser des techniques de cryptanalyse comme la cryptanalyse linéaire pour briser notre système.

Il a été montré dans [28, 29, 84] que plus la taille des S-Boxes choisies aléatoirement augmente, plus il est probable et facile de tomber sur de bonnes S-Boxes, c'est-à-dire des S-Boxes qui ont de bonnes propriétés de diffusion et qui respectent le critère d'avalanche strict et le *BIC*. L'utilisation des S-Boxes $n \times n$ a pour conséquence d'assurer la présence de fonctions linéaires biaisées dans la description du protocole. Par contre, étant donné la taille des S-Boxes, l'effet de ce biais sera diminué de manière importante comme indiqué dans [28].

L'analyse et les différents résultats de la résistance du cryptosystème aux différents types d'attaques sont présentés dans les sections 6.2 à 6.5. *SAND* résiste très bien à la cryptanalyse linéaire ainsi qu'à la cryptanalyse différentielle et ses variantes. Nous allons également définir une généralisation des différents types d'attaques afin de démontrer que *SAND* possède une bonne résistance contre de nouvelles techniques qui pourraient être développées dans le futur.

6.1 Propriété d'avalanche

Comme mentionné précédemment, la diffusion dans les réseaux de substitutions et de permutations est l'un des éléments les plus importants pour leur sécurité. Certains travaux ont suggéré différentes méthodes de diffusion afin d'accroître celle-ci au maximum. [12, 49, 50] Pour notre système, nous avons opté pour un palier de transformations linéaires inversibles. Il est essentiel que cette transformation soit inversible pour garantir

la réversibilité du protocole. Afin d'analyser le niveau de diffusion, nous devons d'abord définir ce que nous considérons comme des S-Boxes actives :

Définition 18. Une S-Boxe est active si la valeur d'au moins un de ses bits d'entrée est modifiée lorsque nous modifions un message M afin d'obtenir un message M_1.

Définition 19. Le niveau de diffusion est le nombre de S-Boxes qui deviennent actives lorsque nous modifions un bit m_i au hasard du message M ainsi que le nombre de bits de C qui sont modifiés par ce changement.

Fait 1. *SAND atteint son niveau de diffusion maximum après un round. Son niveau de diffusion maximum correspond à rendre la quasi-totalité des S-Boxes actives et à ce que la modification d'un bit engendre la modification, en moyenne, de la moitié des bits de sortie du système.*

Pour analyser la diffusion dans le système, supposons que nous modifions un bit d'entrée du système, x_i. Si nous suivons son parcours dans le cryptosystème, x_i subit d'abord un ou-exclusif avec le bit de la clé k_i. Puisque x_i correspond à une modification par rapport au message précédent, le résultat de l'addition est également différent. La modification se propage donc jusqu'au palier suivant, qui est le palier des transformations linéaires, où chaque bit de sortie de la transformation linéaire a une chance sur deux de dépendre de x_i. Par conséquent, en moyenne, la valeur de x_i affectera la moitié des bits de sortie de la transformation linéaire. La modification d'un bit à l'entrée du palier de transformations linéaires aura donc pour effet de propager et diffuser la modification en moyenne dans la moitié des bits de sortie du palier de transformations linéaires.

Ensuite, ces modifications seront transmises au palier des S-Boxes. Puisque la moitié des bits de sortie du palier précédent a été modifiée, au moins la moitié des S-Boxes aura des bits d'entrée qui auront été modifiés. Il est même hautement probable que presque la totalité des S-Boxes reçoive au moins un bit d'entrée modifié, car la probabilité qu'aucun bit d'entrée d'une S-Boxe n'appartienne à un des bits modifiés est de $1/2^n$, où n est le nombre de bits d'entrée de la S-Boxe. Si $n = 16$, alors la probabilité qu'il existe une S-Boxe qui ne soit pas modifiée est de $1/2^{16} = 0.000015 = 0.0015\%$. Par conséquent, la

quasi-totalité des S-Boxes produira des sorties qui auront au moins un bit modifié, même si, en moyenne, la moitié des bits de sortie d'une S-Boxe sera modifiée. Par conséquent, dès le premier *round*, la quasi-totalité des S-Boxes devient active et, en moyenne, la moitié des bits de sortie est modifiée, ce qui correspond au niveau de diffusion maximum de notre cryptosystème.

6.2 Résistance à la cryptanalyse linéaire

La cryptanalyse linéaire est, quant à elle, basée sur des caractéristiques linéaires qui permettent de linéariser les S-Boxes pour simplifier le système et le briser par traitement statistique. Il n'existe pas de S-Boxes $n \times n$ qui ne possèdent pas de caractéristiques linéaires. Plusieurs études ont été effectuées afin de montrer que la meilleure façon d'éviter d'être vulnérable à la cryptanalyse linéaire était d'utiliser des *Bent functions* qui nécessitent un nombre de bits d'entrée des S-Boxes supérieur au nombre de bits de sortie. [25, 41, 48, 56] Puisque nous n'utilisons pas de telles S-Boxes dans notre système, il est inévitable qu'il y ait des caractéristiques linéaires qui soient théoriquement exploitables pour des attaques de cryptanalyses linéaires. Nous allons expliquer que, malgré cela, la cryptanalyse linéaire est totalement inefficace contre notre système.

Fait 2. *SAND résiste à la cryptanalyse linéaire.*

Afin d'expliquer le comportement de notre système contre la cryptanalyse linéaire, nous prendrons les paramètres suivants : $N := 16$, $R := 3$, $L := 4$, $n := 4$.

Ces paramètres correspondent à une version très réduite de *SAND*. Cette version n'est utilisée que pour simplifier les équations et les explications de cette section. Nous considérerons par exemple que la S-Boxe S possède la relation $e_1 \neq s_3$, où e_1 et s_3 correspondent respectivement au premier bit d'entrée et au troisième bit de sortie de la S-Boxe. Nous supposerons également que cette relation est véridique trois fois sur quatre, ce qui correspond à un très gros biais. Il sera expliqué à la section 6.5 pourquoi un aussi gros biais est virtuellement impossible. Dans le cas de notre système, les biais seront au plus de l'ordre de 10% et, plus probablement, de l'ordre de 1% ou moins.

Sans perte de généralité, nous pouvons analyser ce qui se passe lorsque nous tentons d'exploiter cette faiblesse au premier niveau de S-Boxes. Afin de pouvoir déterminer la valeur de e_1, il faut que nous considérions l'ensemble des variables impliquées dans l'équation du premier bit de sortie des transformations linéaires. Par exemple, supposons que la valeur de e_1 est déterminée par l'équation suivante, qui correspond à une transformation linéaire :

$e_1 = k_2 + k_3 + k_4 + k_5 + k_7 + k_8 + k_9 + k_{11} + k_{13} + k_{14} + m_2 + m_3 + m_4 + m_5 + m_7 + m_8 + m_9 + m_{11} + m_{13} + m_{14}$, où les k_i correspondent aux ièmes bits de la clé et les m_i, aux ièmes bits du message à chiffrer.

Afin de pouvoir utiliser la cryptanalyse linéaire, il faudrait être en mesure, en évaluant pour toutes les combinaisons possibles, de remonter à partir des bits du chiffrement jusqu'au bit de sortie d'au moins une S- Boxe. Or, le problème entre cette méthode et notre système est que les bits de sortie des transformations linéaires correspondent seulement à la parité d'un ensemble de termes correspondant aux sorties des S-Boxes mais que, pour obtenir la valeur réelle des termes, il faut réussir à inverser complètement l'ensemble des transformations linéaires. Cela impliquerait de tester l'ensemble de toutes les clés possibles. Par conséquent, l'application de la cryptanalyse linéaire, dans le cas présent, serait aussi coûteuse que d'utiliser la force brute pour briser le protocole.

Ce qui rend inefficace la cryptanalyse linéaire est le fait que connaître seulement quelques bits d'entrée et de sortie des transformations linéaires ne permet pas d'inverser les transformations et de pouvoir accéder aux S- Boxes contrairement à la structure traditionnelle des réseaux de permutations et de substitutions. Afin de pouvoir inverser les transformations linéaires, il faut absolument obtenir soit l'ensemble des bits d'entrée, soit l'ensemble des bits de sortie, ce qui implique, dans le cas de notre système, de tester l'ensemble des bits de clés et donc d'effectuer une attaque de force brute.

6.3 Résistance à la cryptanalyse différentielle

La cryptanalyse différentielle est basée sur la présence de caractéristiques différentielles qui se retrouvent systématiquement sur toutes les S-Boxes $n \times n$. Il est possible de

concevoir les S-Boxes afin que ces caractéristiques ne puissent pas rendre les systèmes totalement vulnérables. La façon la plus simple est de faire en sorte que la longueur de ces caractéristiques soit inférieure au nombre total de *rounds* du système. Par exemple, certains systèmes sont conçus de façon à ce qu'ils ne possèdent pas de caractéristiques différentielles qui dépassent trois ou quatre *rounds* dans un système en comportant plus d'une douzaine [22, 32]. Un programme analysant la résistance des S-Boxes face à la cryptanalyse linéaire et différentielle fut présenté dans [46] en 2012. Nous expliquerons que la présence de ces caractéristiques dans notre système ne le rend pas vulnérable, car il n'est pas possible, lorsque nous atteignons aussi peu que quatre *rounds*, d'exploiter les caractéristiques en question.

Fait 3. *SAND, avec plus de quatre rounds, est résistant à la cryptanalyse différentielle.*

Premièrement, nos S-Boxes ne possèdent pas de biais significatifs qui pourraient être exploitables. Cette caractéristique est due au fait qu'elles sont générées aléatoirement et qu'il est hautement improbable qu'une telle S- Boxe possède un biais suffisant pour que la cryptanalyse différentielle puisse l'exploiter. La généralisation des attaques de biais ainsi que la probabilité qu'un tel biais existe sont présentées dans la section 6.5. Malgré cela, même dans le cas hautement improbable où nos S-Boxes auraient un biais exploitable par la cryptanalyse différentielle, celle-ci ne pourrait pas s'appliquer. Comme il sera expliqué dans la section 6.5, la probabilité qu'il existe un biais qui est vrai 3 fois sur 8 est de $1/2^{80.73}$ si $n = 4$. Afin de justifier cette affirmation, nous reprendrons les paramètres utilisés dans la section précédente 6.2, soit $N = 16$, $R = 3$, $L = 4$, $n = 4$, ainsi que la transformation linéaire représentée par la matrice de la figure 6.I.

Nous supposerons également que notre S-Boxe possède deux caractéristiques différentielles qui apparaissent 3 fois sur 8, soit 37,5% du temps :

- $0011 \Rightarrow 0110$;
- $0101 \Rightarrow 0110$.

Dans le contexte d'un réseau traditionnel de permutations et de substitutions, ces caractéristiques seraient suffisantes pour briser le cryptosystème et déterminer la clé utilisée grâce à la cryptanalyse différentielle. Pour appliquer la cryptanalyse différentielle,

Tableau 6.I – Matrice de la transformation linéaire

$$\left[\begin{array}{cccccccccccccccc}
0&1&0&1&0&1&0&1&0&1&0&0&1&1&0&1\\
0&1&1&0&0&0&1&0&0&1&1&1&0&0&0&0\\
1&0&1&0&1&0&1&0&1&0&1&0&0&1&0&0\\
1&1&0&0&0&1&1&0&0&1&0&0&0&1&1&1\\
1&0&1&0&1&0&0&1&0&0&0&0&0&0&1&1\\
1&0&0&1&0&0&1&1&1&0&1&1&0&0&0&0\\
0&1&1&1&1&0&1&0&0&0&1&1&0&1&0&1\\
1&0&0&0&1&0&0&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&0&0&1&1&1&1&0&1&0\\
1&1&1&0&1&0&1&0&0&0&0&1&1&0&1&0\\
0&1&1&0&0&0&1&0&0&0&1&1&1&0&0&1\\
1&1&0&1&1&1&0&0&0&1&0&1&0&0&0&1\\
0&1&1&0&0&1&1&1&0&0&1&0&0&1&1&1\\
1&0&0&0&1&1&0&0&0&1&0&0&1&0&0&0\\
1&1&1&0&0&1&1&0&1&1&0&1&0&1&0&0\\
0&0&0&1&1&0&0&1&1&1&0&0&1&0&0&1
\end{array}\right]$$

il faudrait déterminer les bits du message à changer afin de créer la différence désirée au niveau de l'entrée des S-Boxes. Sans pertes de généralité, nous pouvons prendre la première caractéristique et créer la différence au niveau de la deuxième S-Boxe. Afin de créer la différence désirée, nous pourrions modifier uniquement le cinquième bit du message, car ce bit est présent dans l'équation des deux derniers bits de l'entrée de la S-Boxe mais pas des premiers bits, comme indiqué dans la figure 6.I. Ainsi, en modifiant uniquement la valeur de ce bit, nous créons la différence désirée. Malheureusement, le fait de modifier la valeur du cinquième bit ne modifie pas seulement la valeur des deux derniers bits de la deuxième S-Boxe. Ce bit se retrouve également dans les équations de sept autres bits d'entrée réparties ailleurs dans le système. Par conséquent, la modification d'un seul bit du message original permet effectivement d'obtenir la différence voulue au niveau de la deuxième S-Boxe afin de tenter d'exploiter la faiblesse de celle-ci. Par contre, cette modification crée également des effets de bord en modifiant également les entrées d'autres S-Boxes. Cette situation n'est pas un cas exceptionnel dû au choix de modifier spécifiquement ce bit. Tout choix de modifications du message peut entrainer des modifications sur plus d'une S- Boxe. Pour avoir une modification qui ne s'appli-

querait qu'à une seule S-Boxe, il faudrait trouver une combinaison linéaire des entrées voulues de la S-Boxe dont seulement un nombre pair des termes de la combinaison se retrouverait parmi les équations des autres entrées des S-Boxes. Cela est possible lorsque nous appliquons l'inverse de la matrice représentant les transformations linéaires sur la différence voulue. Il n'y a qu'une seule combinaison possible.

Par conséquent, même s'il était possible qu'il n'y ait pas d'effets de bord pour les bits d'entrée de la première étape de S-Boxes, ces effets de bord seraient présents pour le *round* suivant. Après un *round*, nous nous retrouverions dans le meilleur cas avec la moitié des S-Boxes dont les entrées ont été modifiées par la modification faite au premier *round*. Avec les propriétés de diffusion de notre système, le nombre de S-Boxes actives n'augmentera pas avec le nombre de *rounds*, mais plus il y aura de *rounds* et plus il sera difficile de suivre et de prédire le comportement du système. Lorsqu'il n'y a qu'un nombre réduit de bits qui influence l'ensemble des bits d'entrée des S-Boxes suivantes, il est possible de prédire le comportement et la diffusion, mais lorsque le nombre de bits ne cesse de grossir et que plusieurs combinaisons de bits modifiés peuvent occasionner le même résultat, il devient très ardu de pouvoir prédire le comportement du système.

De plus, avec le lemme du *piling up*, s'il y a beaucoup de S-Boxes impliquées, il est nécessaire que le biais soit très important pour que ça reste profitable d'utiliser la cryptanalyse différentielle, par exemple, avec les paramètres suivants : $N := 256$; $R := 3$; $L := 16$; $n := 16$. Supposons que nos S-Boxes possèdent une caractéristique différentielle qui est biaisée à 45% du temps et que nous considérons que la moitié des S-Boxes seront actives à partir du deuxième *round* en raison des propriétés de diffusion. Le biais sur l'ensemble du système sera alors de $.45^{1+8+8} = 0.0000012737$ après trois *rounds*. Dans ce cas-là, il faudrait donc $1/(0.0000012737)^2 = 6.17 * 10^{11}$ messages pour pouvoir utiliser la cryptanalyse différentielle avec des caractéristiques ayant un biais de 45% .

Malgré le fait que la cryptanalyse différentielle soit totalement inefficace contre notre système, nous allons étudier, dans la prochaine section, une variante qui obtient un certain succès lorsque le nombre de *rounds* est très petit.

6.4 Cryptanalyse du système

Afin de bien comprendre les caractéristiques de *SAND*, nous avons analysé sa sécurité lorsqu'il ne possédait qu'un nombre réduit de *rounds*. Les prochaines sections expliqueront la meilleure méthode de cryptanalyse que nous connaissons qui permet de briser notre système lorsqu'il a moins de cinq *rounds*. Nous avons donné à l'adversaire le maximum de pouvoir en considérant une attaque à messages clairs choisis. Tout comme AES et DES, notre cryptosystème devient significativement plus difficile à briser si une attaque de type messages clairs connus ou messages chiffrés seulement est utilisée.

6.4.1 Un *round*

Afin de briser *SAND* avec un seul *round*, il faut trouver une combinaison linéaire des bits de sortie de la transformation linéaire ayant des caractéristiques précises. Cette combinaison doit correspondre à une caractéristique différentielle voulue des bits d'entrée de la S-Boxe ciblée. Il faut également que seulement un nombre pair des termes de la combinaison se retrouve parmi les équations des autres entrées des S-Boxes. Cela permet d'éviter des effets de bord indésirables lors de l'attaque du système. C'est possible lorsque nous appliquons l'inverse de la matrice représentant les transformations linéaires sur la différence voulue : il n'y a qu'une seule combinaison possible, car normalement, à l'étape de la transformation linéaire, nous avons $aM_t = b$, où a est le vecteur d'entrée des transformations linéaires ; M_t, la matrice de transformations linéaires et, par conséquent, b, le vecteur de sortie du palier de transformations linéaires. Dans notre cas, nous souhaitons trouver a, qui est obtenu par l'opération suivante : $bM_t^{-1} = a$, en connaissant b et M_t^{-1} . Avec un seul *round*, le système est facilement brisable une fois que cette combinaison est connue.

Lemme 2. *Nous pouvons briser SAND à un round à l'aide d'une attaque à messages clairs choisis dont la complexité correspond à utiliser* $L+1$ *messages et à tester* $L2^n$ *clés possibles.*

D'un point de vue pratique, supposons que nous disposons d'une puissance de calcul permettant d'effectuer un giga de chiffrement ou de déchiffrement par seconde, soit un

gigahertz de puissance. Cette puissance de calcul peut sembler énorme, mais elle est tout à fait réalisable avec la puissance actuelle des processeurs et la parallélisation. Dans ce cas, il faudrait un peu plus d'une minute, soit 69 secondes, pour briser notre système à un *round* lorsque $n = 32$ et $L = 16$, soit une clé de 512 bits.

Démonstration. Afin de poursuivre avec l'explication de l'attaque, nous utilisons la combinaison a mentionnée précédemment afin de déterminer les bits à modifier d'un premier message M. Nous demandons ensuite le chiffrement du message M_1, qui correspond au message M dont nous avons modifié les bits appropriés pour obtenir la caractéristique différentielle désirée. Par la suite, nous calculons la différence entre le chiffrement de M et de M_1. Nous obtenons ainsi la différence d_1 produite par notre modification.

En appliquant l'inverse de la matrice des transformations linéaires sur d_1, nous obtenons exactement les bits de sortie de la S-Boxe visée qui ont été modifiés par a. Dans le pire cas, cette combinaison de bits de sortie de la S-Boxe qui ont été modifiés correspond à l'ensemble des valeurs possibles de bits d'entrée de la S- Boxe. Dans ce cas, cela implique que la caractéristique était valide 100% du temps. Dans une attaque de cryptanalyse différentielle traditionnelle, cette caractéristique serait fortement bénéfique et utilisée, mais dans notre cas, nous recherchons, au contraire, des caractéristiques avec les plus faibles pourcentages. La raison pour laquelle nous préférons des caractéristiques avec les plus faibles pourcentages est qu'avec un pourcentage de 100%, cela implique que nous n'obtiendrons aucune information sur les valeurs possibles des bits d'entrée de la S-Boxe ciblée par le message M. Or, si nous avons une caractéristique dont le pourcentage correspond à $((2^n) - 1)/(2^n)$, où n correspond au nombre de bits d'entrée de la S-Boxe, alors nous pouvons éliminer avec certitude une possibilité parmi les valeurs possibles des bits d'entrée de la S-Boxe ciblée par le message M. La figure 6.1 représente cette attaque.

Par la suite, nous pouvons choisir une autre S-Boxe et déterminer la différence c, qui créerait la caractéristique différentielle voulue. Nous pouvons ensuite procéder de manière similaire qu'avec le message M_1 afin d'obtenir d_2. Nous pourrons ainsi éliminer au moins une valeur possible de plus des bits d'entrée de la S-Boxe pour le message

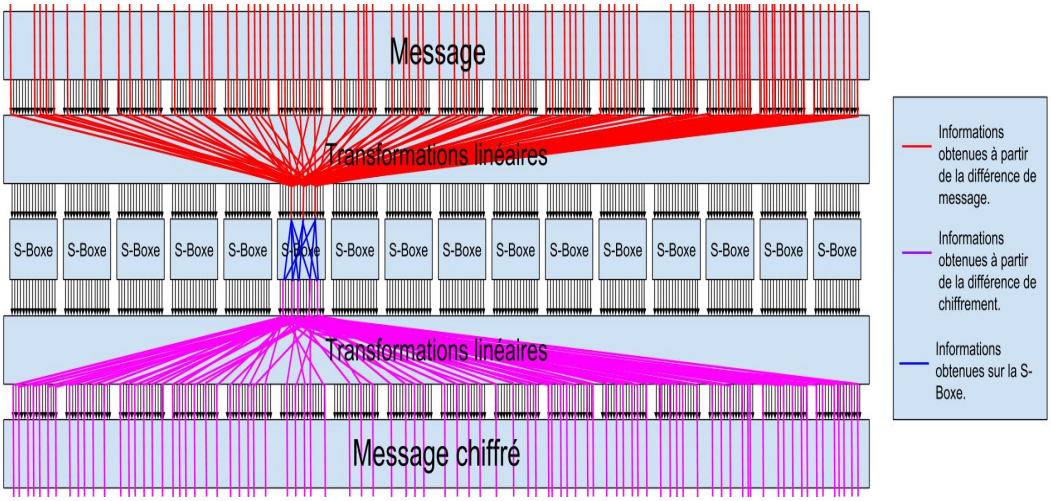

Figure 6.1 – Schéma d'attaque à un *round*

M. Il suffit d'appliquer ce processus sur l'ensemble des k S-Boxes afin de réduire le nombre de possibilités d'au moins $((2^n)^k) - (((2^n) - 1)^k)$ valeurs possibles. Ensuite, nous appliquons la matrice inverse des transformations linéaires sur les $(((2^n) - 1)^k)$ valeurs possibles et appliquons un ou-exclusif sur le résultat pour chaque valeur avec le message M. Nous obtenons ainsi les $(((2^n) - 1)^k)$ valeurs de clés possibles.

Finalement, il suffit de tester chacune de ces clés sur un des messages pour obtenir la clé du système. Dans le cas où la clé utilisée à la fin du système n'est pas la même que celle utilisée au début, tester les différentes premières clés possibles permet de déterminer, pour chacune d'elles, une seule seconde clé possible. Une fois ces paires de clés obtenues, il suffit d'utiliser un deuxième message et de tester l'ensemble des paires pour obtenir uniquement celle qui correspondrait aux bonnes clés. Cela correspond au pire cas où nous ne pourrions pas réduire le nombre de possibilités pour les valeurs d'entrée des S-Boxes de plus d'un à chaque fois. Or, même dans ce cas-ci, il n'est pas impossible que, pour une des S-Boxes, la valeur des bits modifiés corresponde à une seule valeur pour les bits d'entrée de la S-Boxe. Dans ce cas, nous obtenons un bien meilleur gain en terme de clés éliminées. □

Si nous prenons l'exemple proposé précédemment ainsi que le tableau de caractéristiques différentielles de la figure 4.I, nous pouvons remarquer que la différence de sortie

produite par une différence des bits d'entrée de 0001 correspond à, au plus, trois valeurs possibles des bits d'entrée, ce qui implique que nous n'aurons que $2^4 = 16$ clés possibles à tester sur les $2^{16} = 65536$ clés existantes, ce qui est un gain remarquable.

Cette attaque ne correspond pas à une approche traditionnelle de la cryptanalyse différentielle, mais nous considérons que c'est la variante la plus efficace contre notre système. La complexité de cette attaque est d'ailleurs linéaire par rapport à la taille des S-Boxes et au nombre de celles-ci. Effectivement, afin qu'elle fonctionne peu importe la taille des S-Boxes, il faudra uniquement un message choisi de plus que le nombre de S-Boxes. De plus, le coût pour réduire le nombre de clés possibles correspond à chiffrer et déchiffrer chacun des messages choisis. Ce coût reste donc négligeable. Ce qui est le plus coûteux est de tester l'ensemble des clés possibles.

En résumé, nous créons une différence entre deux messages afin de pouvoir déterminer la valeur des bits d'entrée d'une S-Boxe. Par la suite, nous répétons l'opération pour chaque S-Boxe afin de pouvoir déterminer la valeur des bits de la première clé utilisée. Finalement, à partir de cette information, il est possible de déterminer la valeur des bits de la dernière clé et, ainsi, de briser complètement le cryptosystème.

6.4.2 Deux *rounds*

Lemme 3. *Nous pouvons briser SAND à deux rounds avec une attaque à messages clairs choisis. La complexité de l'attaque correspond à utiliser $L+1$ messages et à tester, en moyenne, $L2^{3n/2}$ clés possibles.*

En comparaison avec le système à un *round*, supposons de nouveau que nous possédions une puissance d'un GHz de chiffrement et que $n = 32$. Il faudrait, dans ce cas, environ $3.25L$ jours afin de tester toutes les clés nécessaires. Par exemple, tester l'ensemble des clés possibles de cette attaque prendrait respectivement 26 jours si $L = 8$ et 104 jours si $L = 32$.

Démonstration. Il est possible d'adapter l'attaque utilisée pour un seul *round* afin de réduire le nombre total de clés pour le système avec deux *rounds*. Comme précédemment,

nous attaquerons chaque S-Boxe séparément avec un message M_1 différent correspondant à une différence d_i voulue. Notre approche à deux *rounds* est légèrement différente, car lorsque nous calculons les bits qui ont été modifiés à l'entrée du dernier bloc de transformation linéaire, nous n'arrivons pas directement à la S-Boxe ciblée mais plutôt au bloc de S-Boxes suivant. Par contre, en connaissant les bits de sortie des S-Boxes qui ont été modifiés, il est possible de dresser une liste des entrées possibles pour chaque S-Boxe en fonction de la modification obtenue. La figure 6.2 représente cette attaque.

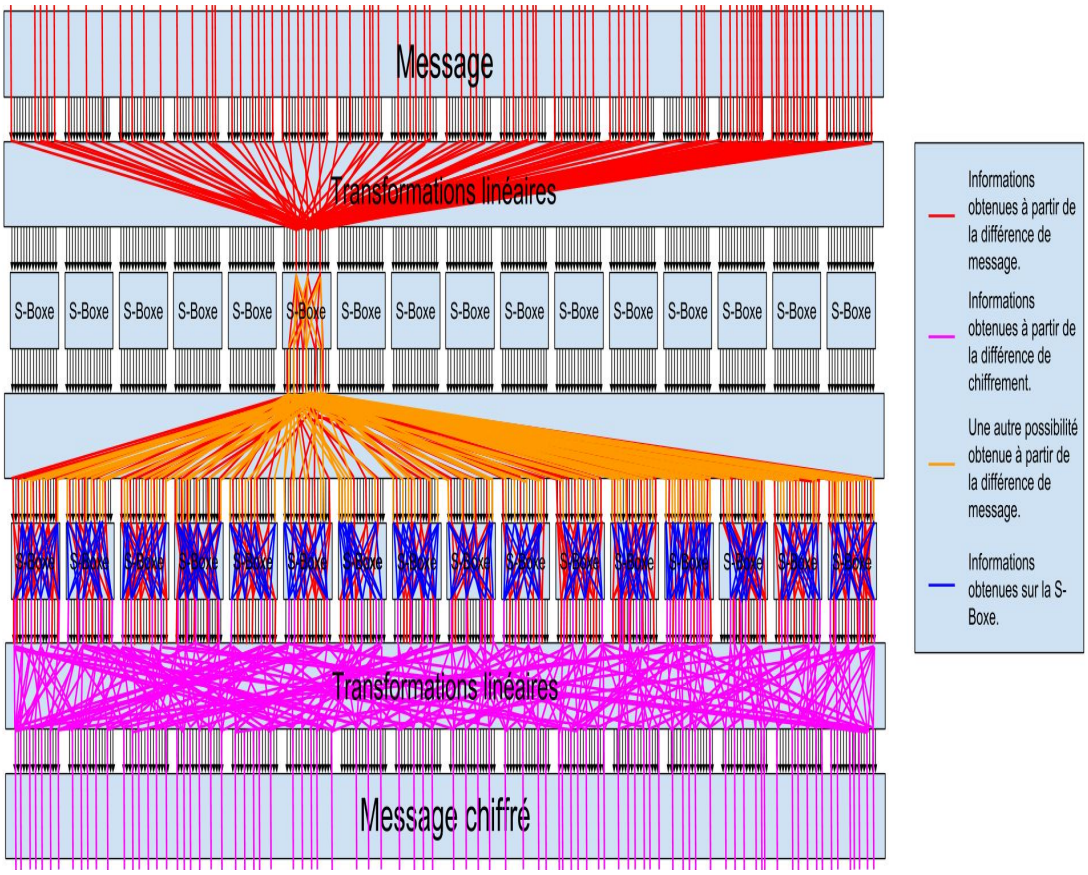

Figure 6.2 – Schéma d'attaque à deux *rounds*

Ensuite, ce qu'il nous reste à tester est le bloc des transformations linéaires sur toutes les différences de sorties possibles en fonction de la différence choisie entre M et M_1. En appliquant ces transformations linéaires, nous obtenons une deuxième liste. En effectuant l'intersection entre les deux listes, nous obtenons une liste d'éléments qui déter-

mine exactement les différences qui résultent de la différence choisie sur l'entrée de la S-Boxe. Il est même probable que l'intersection ne contienne qu'un seul élément. Dans ce cas, nous avons déterminé la différence qui a résulté de la différence occasionnée par la modification des bits d'entrée de la S- Boxe. Par la suite, il suffit d'appliquer le même processus que celui utilisé pour un *round* afin de déterminer la clé en répétant les étapes précédentes pour l'ensemble des S-Boxes. Dans le cas où l'intersection contiendrait plus d'un élément, il suffit d'ajouter ces autres éléments parmi les entrées possibles, ce qui a pour effet d'augmenter le nombre de clés possibles. Nous gardons quand même un gain remarquable comparé à une attaque de force brute. La complexité de cette attaque reste dans le même ordre que l'attaque sur un *round*, car la seule opération supplémentaire effectuée est la constitution des deux listes ainsi que leur comparaison pour déterminer leur intersection.

La complexité reste donc liée à la taille et au nombre de S-Boxes et non à la taille de la clé. La première liste possèdera, au plus, 2^n éléments, où n correspond à la taille des S-Boxes. La deuxième liste sera donc plus volumineuse avec, dans le pire cas, $N_A(2^n)$ éléments, où N_A correspond au nombre de S-Boxes dont au moins un bit a été modifié par la modification entre M et M_1 que nous appellons des S-Boxes actives. En moyenne, les listes auront respectivement $2^{n/2}$ et $N_A(2^{n/2})$ éléments. Dans le cas où une opération de ou-exclusif avec une autre clé est effectuée entre les deux niveaux, cette action n'annulera pas ni n'influencera la complexité globale du problème. Le fait d'ajouter cette étape supplémentaire n'empêchera pas notre attaque d'obtenir les bits de clés de la première clé utilisée. Par contre, si des clés intermédiaires sont ajoutées entre chaque *round* du système, alors pour déterminer ces clés et briser totalement le protocole, il faudra effectuer une nouvelle attaque pour chacune d'entre elles. Par contre, ces nouvelles attaques viseront, pour chacune, un système qui sera réduit d'un *round* par rapport à l'attaque précédente et donc la complexité en sera aussi réduite. Dans ce cas, si une clé est utilisée entre les deux *rounds*, une fois que les bits de la première clé seront déterminés, il faudra effectuer une nouvelle attaque afin de déterminer les bits de cette nouvelle clé. Par contre, puisque la première clé aura été déterminée, alors le système en sera réduit qu'à un système d'un seul niveau. □

6.4.3 Trois *rounds*

Avec trois *rounds*, le système devient plus résistant et la complexité de bris du système augmente énormément par rapport à l'augmentation du nombre de *rounds*.

Lemme 4. *SAND à trois rounds peut être brisé par une attaque à messages clairs choisis. La complexité de l'attaque correspond à utiliser $L+1$ messages et à tester $L2^{rn+n-r}$ clés possibles, où r correspond au nombre de S-Boxes actives.*

Supposons de nouveau que nous disposions d'une puissance d'un GHz de chiffrement et que $n = 32$. Dans le cas où $r = 8$, il faudrait environ $(6x10^{67})L$ années afin de tester toutes les clés nécessaires. Même dans le cas où $n = 16$ et $r = 8$, il faudrait $(2.75x10^{24})L$ années. Cette attaque devient donc irréalisable en pratique si nous avons seulement une puissance équivalente à un GHz de chiffrement. Afin de pouvoir effectuer cette attaque en $2.75L$ années avec les derniers paramètres, nous devrions avoir une puissance équivalente à 10^{33} Hertz. Par contre, si nous comparons cela avec une attaque de force brute, cette attaque est plus efficace que la force brute si $r < L$. Il est par contre *important* de noter, qu'avec les propriétés d'avalanche de *SAND*, il est peu probable que $r < L$. Comme indiqué à la section 6.1, la probabilité qu'une S-Boxe ne soit pas active est de .0015%.

Démonstration. En ce qui concerne l'attaque en soi, si nous appliquons le même type de raisonnement d'attaques que dans les versions avec un ou deux *rounds*, nous nous retrouvons à faire des comparaisons entre deux listes dont le nombre d'éléments est de l'ordre de $((2^{n-1})^{r+1})$ et $((2^{n-1})^{r})$. Nous obtenons de telles listes, car en ajoutant un *round* au système, nous ajoutons une opération de transformation linéaire qui n'influence pas le nombre de possibilités, car c'est un processus linéaire qui ne permet qu'une seule sortie pour chaque changement d'entrée. En plus de cette opération de transformation linéaire, un bloc de S-Boxes est également ajouté et c'est cet ajout qui crée l'augmentation de la complexité. Chaque changement au niveau des bits d'entrée entraine 2^{n-1} changements possibles pour la sortie d'une S-Boxe. La raison de l'explosion de la taille des listes est qu'avec ce niveau supplémentaire, il n'est pas possible de s'assurer qu'une seule S-Boxe

sera touchée par les modifications effectuées par les niveaux précédents. La modification des bits d'entrée d'une seule S-Boxe entraine, à travers le bloc de transformations linéaires qui suit, une modification de l'ensemble des entrées des S- Boxes du niveau suivant. Il n'est pas possible de contrôler les modifications du premier bloc de S-Boxes afin de ne modifier qu'un nombre précis ou exactement certaines S-Boxes du niveau suivant, car il n'y a aucune garantie qu'en modifiant les entrées d'une S-Boxe, nous obtenions les sorties désirées et que nous puissions donc modifier, comme nous le voudrions, les S-Boxes du niveau suivant.

Tout ce que nous pouvons faire est d'essayer toutes les combinaisons possibles afin d'obtenir des résultats aussi fiables qu'avec un système comportant moins de *rounds*. Chacune de ces 2^{n-1} combinaisons va entrainer 2^{n-1} nouvelles possibilités pour chaque S-Boxe qui sera modifiée. Nous estimons qu'avec le système de transformation linéaire, la moitié des bits sera modifiée et donc que l'ensemble des S-Boxes sera touché par une modification. Le nombre de combinaisons à la sortie de ces S-Boxes monte à $(2^{n-1})r_i$, où r_i correspond au nombre de S-Boxes touchées par la modification. Ce nombre est valide pour chacune des 2^{n-1} possibilités de modifications occasionnées par le niveau précédent, d'où le fait que nous ayons une liste de $(2^{n-1})^{r+1}$ éléments. Pour obtenir la deuxième liste de comparaison, il faut partir de la fin du système comme avec les versions à deux (ou un) *rounds*. En partant de la fin, nous sommes toujours en mesure de remonter jusqu'aux entrées du dernier bloc de S-Boxes. À ce stade, nous avons k listes d'en moyenne 2^{n-1} éléments. Si nous prenons l'ensemble de ces possibilités afin d'inverser le bloc de transformations linéaires, nous avons $(2^{n-1})^k$ entrées possibles au bloc de transformations linéaires. Nous sommes également au même emplacement dans le système que notre liste de $(2^{n-1})^{r+1}$, que nous avions déterminée en descendant dans le système. Si nous comparons ces deux listes, nous obtiendrons une liste nettement plus petite contenant l'ensemble des solutions possibles. Pour chaque élément de cette dernière liste, nous calculons, comme précédemment, les entrées possibles du message M et déterminons l'ensemble des clés possibles. La figure 6.3 représente cette attaque.

□

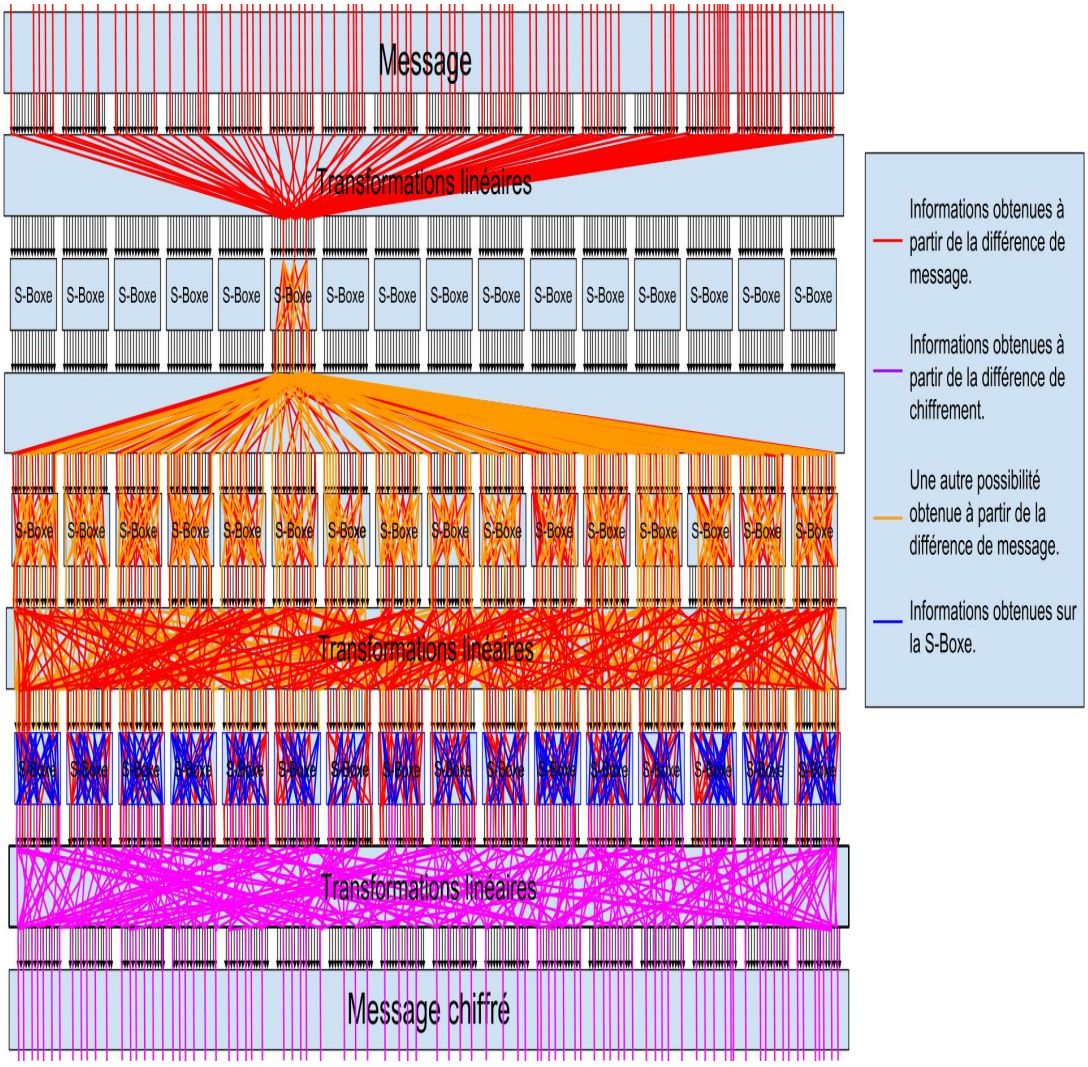

Figure 6.3 – Schéma d'attaque à trois *rounds*

6.4.4 Quatre *rounds*

Lemme 5. *SAND à quatre rounds peut être brisé par une attaque à messages clairs choisis. La complexité de l'attaque est d'utiliser* $L+1$ *messages et de tester* $L2^{rn+n-r}$ *clés possibles, où r correspond au nombre de S-Boxes actives.*

En ce qui concerne la puissance nécessaire pour effectuer cette attaque, nous obtenons les mêmes conclusions que dans la section précédente 6.4.3, puisque la complexité de l'attaque est la même.

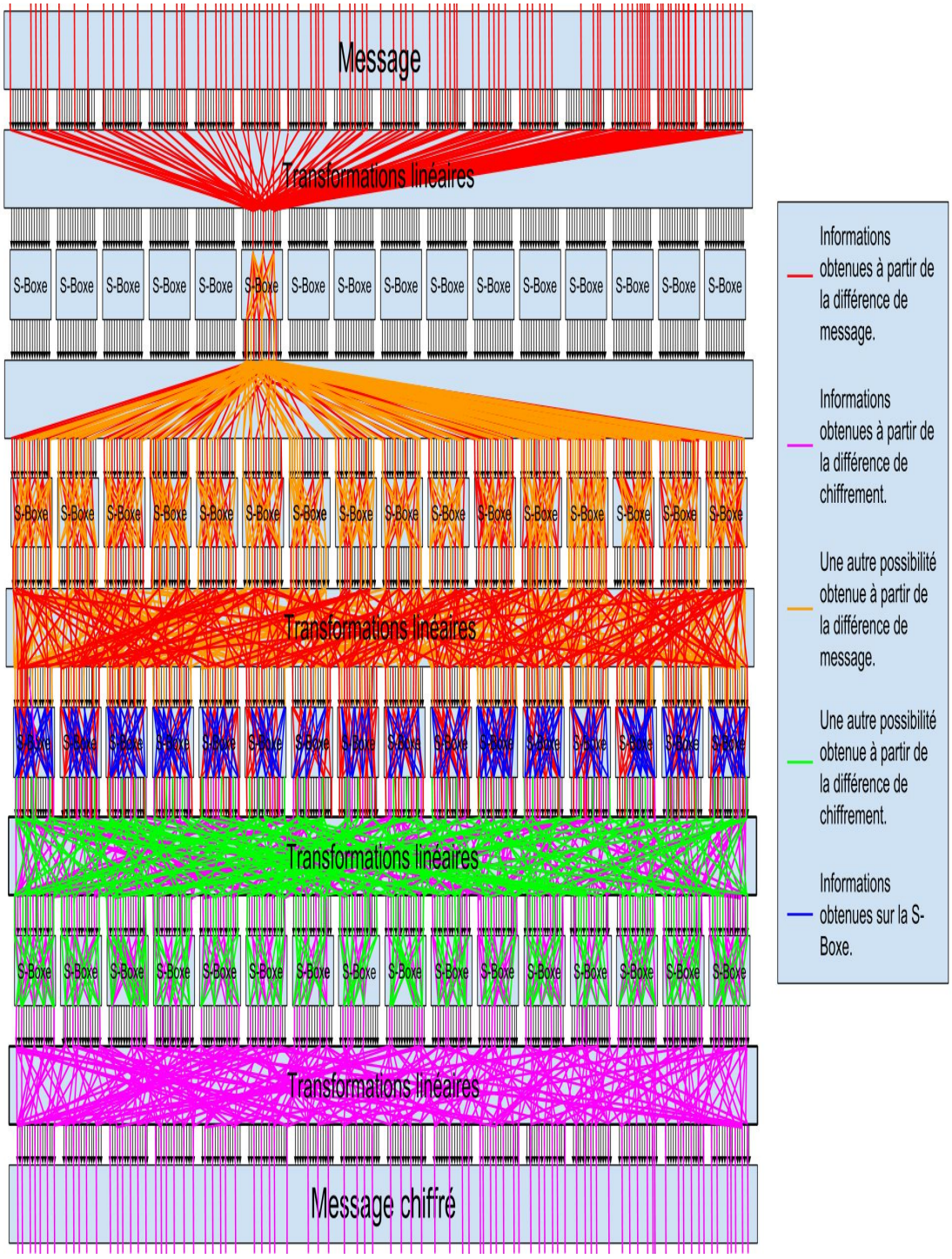

Figure 6.4 – Schéma d'attaque à quatre *rounds*

Démonstration. La principale différence, dans ce contexte, est que nous devons ajouter un bloc de transformations linéaires suivi d'un bloc de S-Boxes au point de rencontre des deux listes que nous avions obtenues précédemment. Cela a pour effet de séparer les deux listes. Par contre, il suffit d'appliquer le bloc de transformations linéaires à nos $(2^{n-1})^{r+1}$ possibilités pour déterminer les entrées possibles du bloc de S-Boxes sans avoir à augmenter le nombre de possibilités. De plus, nous avons les $(2^{n-1})^k$ possibilités de sortie des S-Boxes que nous avons pu déterminer. Afin de briser le système et de se retrouver dans les mêmes conditions que les fois précédentes, il suffit d'effectuer $(2^{n-1})^k$ fois l'inverse du bloc des S-Boxes afin de déterminer la liste des combinaisons possibles qui permettra de déterminer la clé du système. La figure 6.4 représente cette attaque. □

6.4.5 Cinq *rounds*

Avec cinq *rounds*, notre protocole devient résistant à notre attaque.

Lemme 6. *Lorsque SAND a cinq rounds, l'attaque à messages clairs choisis utilisée précédemment possède une complexité plus grande qu'une attaque de force brute.*

Démonstration. Nous obtenons un système qui possède plus d'un emplacement possible où nous pourrions comparer les changements d_i occasionnés par les changements d'entrée. Si nous comparons les changements au même emplacement que dans la version à quatre *rounds*, soit à l'avant-dernière série de S-Boxes, nous devons comparer deux listes comprenant $(2^{n-1})^{2r+1}$ et $(2^{n-1})^k$ possibilités respectivement. Le nombre de possibilités pour la première liste a augmenté, car nous devons effectuer une transformation linéaire ainsi qu'une série de S-Boxes supplémentaire par rapport au cas précédent. La deuxième liste n'a pas changé, car elle se situe, comme indiqué, exactement au même emplacement que dans le cas précédent.

Il est également possible de vouloir comparer les informations plus tôt dans le cryptosystème, soit au même emplacement, à partir du début, que dans le cas à quatre *rounds*. Dans ce cas, il y aura respectivement $(2^{n-1})^{r+1}$ possibilités provenant du chiffrement et $(2^{n-1})^{k_1+k_2}$ possibilités provenant du déchiffrement, où les k_i correspondent aux nombres de S-Boxes actives lors du déchiffrement.

Avec ces listes, il est toujours possible de briser le système, mais la quantité de travail est nettement augmentée. Par exemple, si nous considérons que seulement la moitié des S-Boxes seront actives autant dans le déchiffrement que dans le chiffrement, nous obtenons des listes de $(2^{n-1})^{2*(8)+1} = (2^{(n-1)*17})$ et $(2^{n-1})^{(8)}$ éléments, ce qui implique que nous avons plus de $(2^{((16*16)-1)*8})$ comparaisons à effectuer. Or, au niveau de la force brute, nous devrions tester (2^{16*16}) combinaisons possibles. Par conséquent, à cinq *rounds*, notre attaque est plus coûteuse qu'une attaque de force brute. □

6.5 Résistance aux attaques de biais

Nous pouvons noter que toutes les attaques présentées précédemment sont basées sur l'hypothèse que les S-Boxes ont un biais en faveur d'une fonction spécifique. Par exemple, dans le cas de la cryptanalyse linéaire, le biais concerne les relations linéaires. Nous allons donc généraliser au maximum ce concept afin d'expliquer que notre système est résistant aux attaques basées sur des biais simples des S-Boxes. Il existe plusieurs types de biais possibles. Notre système ne peut pas être résistant à tous les biais possibles, car les S-Boxes seront toujours biaisées en faveur de la fonction les décrivant. Par contre, dans notre cas, cette fonction possède une description complexe qui demande énormément de bits. Nous allons montrer que notre système est résistant aux biais en faveur de fonctions dont la description est succincte par rapport à la taille de la description des S-Boxes. Ceci inclut les biais envers des fonctions linéaires et les fonctions utilisées par les techniques cryptanalyses présentées au chapitre 4. En effet, ces fonctions demandent très peu de bits pour les décrire.

Définition 20. Une S-Boxe S est *biaisée en faveur d'une fonction* $F : \{0,1\}^t \rightarrow \{0,1\}$, où $t \leq n$ s'il existe un ε non négligeable tel que $Pr[F(x) = 1] \geq N_F/2^t + \varepsilon$, où N_F correspond au nombre d'entrées possibles e de F telles que $F(e) = 1$ et où x correspond à une sortie choisie au hasard de S.

Nous allons montrer qu'il est hautement improbable que nos S-Boxes soient biaisées en faveur d'une fonction qui peut s'exprimer de manière succincte. Notre but est d'expliquer qu'il est pratiquement impossible que nos S-Boxes choisies aléatoirement soient

biaisées envers une fonction possédant une courte description. Pour ce faire, nous allons expliquer les effets et la probabilité d'une compression du tableau décrivant la fonction représentant le fonctionnement des S-Boxes. Il est important de noter que nous avons conçu nos S-Boxes de manière à ce qu'elles soient incompressibles au sens de Kolmogorov et de Shannon, c'est-à-dire que nous considérons que le tableau représentant nos S-Boxes constitue la représentation la plus courte de celles-ci. Nous allons également expliquer comment déterminer si les S-Boxes sont biaisées par un test statistique.

Tout d'abord, s'il existait une fonction pour laquelle la S-Boxe est biaisée, alors il serait possible de compresser la description de celle-ci. Or, selon un argument de comptage, il est possible de déterminer la probabilité qu'une telle compression puisse être effectuée.

Théorème 2. Probabilité de compression d'une fonction *F : Soit une fonction de compression Comp déterminée et soit une fonction F choisie au hasard dont la description prend B bits. Alors la probabilité que Comp puisse compresser F de k bits est de* $1/2^k$.

Démonstration. Soit une fonction de compression *Comp*, et une fonction F choisie au hasard. Alors, pour que *Comp* puisse compresser F de k bits, il faut que F appartienne aux fonctions que *Comp* est en mesure d'exprimer avec $B-k$ bits. Il existe exactement 2^{B-k} fonctions qui peuvent être exprimées avec $B-k$ bits. Par conséquent, la probabilité qu'une fonction F choisie au hasard appartienne à ces fonctions correspond à la probabilité que F soit l'une de ces 2^{B-k} fonctions. Puisque nous avons choisi F au hasard parmi toutes les fonctions se décrivant avec B bits, alors la probabilité que la fonction choisie fasse partie des fonctions pouvant être compressées de k bits est $2^{B-k}/2^B = 1/2^k$. □

Il est important de noter que nous supposons, comme prémisse, qu'une telle fonction F existe et que nous la connaissons. Or, il n'y a aucune certitude sur le fait qu'il soit possible de trouver efficacement une telle fonction. Le nombre de fonctions possibles est dans $O(2^{2^n})$.

Il est donc possible de déterminer, à partir du niveau de compression, la probabilité que notre système soit biaisé en faveur d'une fonction F. Afin de définir le niveau de compression d'une S-Boxe, nous devons définir les paramètres suivants :

- P_F correspond à la probabilité que $F(Y) = 1$;
- P_S correspond à la proportion de bits de sortie de S dont $F(Y) = 1$;
- n correspond à la taille des S-Boxes.

Lemme 7. *Le niveau de compression d'une S-Boxe biaisée en faveur d'une fonction* $F : \{0,1\}^t \rightarrow \{0,1\}$ *est défini par l'équation suivante :*

$C = (2^n)(H(P_F) + (P_F Log_2(P_S 2^n)) + ((1 - P_F)Log_2((1 - P_S)2^n)))$

Résultat 2. *Le nombre de bits nécessaires pour décrire une fonction F dont* $P_F = .5$ *pour que nos S-Boxes aient un biais de 10% avec une probabilité de* $1/2^4$ *est de 126 473 230 bits lorsque* $n = 32$ *et de 1 925 bits lorsque* $n = 16$.

Afin d'expliquer la résistance globale de notre système contre l'ensemble des types d'attaques de biais possibles, nous vous proposons la mise en situation suivante : soit une fonction $F : \{0,1\}^t \rightarrow \{0,1\}$, où $t \leq n$ tel que la proportion des valeurs d'entrée telles que la sortie vaut 1 est P_F. Nous sommes maintenant intéressés à savoir ce qui se passerait si nous appliquions cette fonction sur un sous-ensemble des bits de sortie de la S-Boxe $n \times n$, S, dont les permutations qui la composent ont été choisies aléatoirement.

En appliquant la fonction F sur les bits de sortie de notre système, il sera possible de constater si la S-Boxe était biaisée en faveur ou non de cette fonction. Si un sous-ensemble de bits $\{y_1, ..., y_t\}$ de S était biaisé en faveur de F, alors il serait possible de compresser le tableau en transmettant des informations sur les bits pour lesquelles S est biaisée. Tout d'abord, il suffirait de transmettre une chaîne de 2^n bits où chaque bit indique si les valeurs $F(y_1, ..., y_t) = 1$ ou non. Avec cette information, il est possible de séparer l'ensemble des sorties possibles de chaque ligne en deux sous-ensembles :

- un sous-ensemble ordonné comprenant uniquement les sorties tel que $F(Y) = 1$;
- un sous-ensemble ordonné comprenant uniquement les sorties tel que $F(Y) = 0$.

La chaîne de caractères transmise permet donc de savoir à quel sous-ensemble la sortie appartient. Par conséquent, dans le pire des cas, chaque ensemble possède $2^{n/2}$ éléments. Par la suite, pour chaque ligne, il suffit d'indiquer à quel rang dans un ordre prédéfini se trouve la bonne sortie. Ce processus permet de compresser le tableau seulement si la S-Boxe est biaisée en faveur ou non de F. Le nombre de bits transmis respecte

l'équation suivante :

$C = (2^n)(H(P_F) + (P_F Log_2(P_S 2^n)) + ((1-P_F)Log_2((1-P_S)2^n)))$, où P_S correspond à la proportion des valeurs des bits de sortie de S dont $F(Y) = 1$

Or, si $P_F = P_S$, alors $C = n2^n$, ce qui correspond à la taille du tableau lorsqu'il est transmis de manière traditionnelle. Par contre, si $P_F \neq P_S$, alors il est possible de compresser le tableau. La compression augmente de manière considérable plus P_S est éloigné de P_F. Il est important de noter que, dans cette mise en situation, nous considérons que la description de la fonction F choisie est connue et qu'il n'est pas nécessaire de la transmettre. Maintenant que nous avons une manière de possiblement compresser le tableau de données, nous pouvons nous demander comment déterminer si S est biaisé par rapport à F. Pour ce faire, nous pouvons effectuer l'expérience suivante :

1. analyser un échantillon des lignes, L_i, de S, choisies aléatoirement, sur lesquelles nous déterminons si $F(L_i) = 1$ ou non ;
2. calculer P_S pour l'échantillon ;
   - si $|P_S - P_F| \leq \varepsilon$ avec un ε déterminé préalablement, alors nous considérons que le tableau n'est pas biaisé ;
   - sinon, nous pouvons considérer que S est biaisée par rapport à F et nous pouvons exploiter cette faiblesse pour attaquer S et le cryptosystème.

Il serait intéressant de savoir avec quelle probabilité S peut être biaisé par rapport à une fonction F. Pour ce faire, nous pouvons utiliser le théorème présenté précédemment, qui stipule que la probabilité que la description d'une fonction choisie aléatoirement soit compressible est de $1/2^z$, où z correspond au nombre de bits dont la description de la fonction peut être compressée. Comme indiqué précédemment, plus S sera biaisée par rapport à F, plus il sera possible de compresser le tableau. Par conséquent, il est peu probable que S soit énormément biaisée par rapport à F si S a été déterminée aléatoirement.

Avec la formule de compression, nous pouvons remarquer qu'avec des S-Boxes de 16 bits ainsi qu'un biais de 3% sur un $P_F = .5$, il est possible de compresser le tableau de 170 bits. La compression passe à plus de 580 bits si $P_F = .9$. Sur une S-Boxe de 32 bits, le nombre de bits qu'il est possible de compresser dépasse les 100 000 bits avec aussi peu

que 1% de biais et $P_F = .5$. Le tableau 6.II résume le nombre de bits qu'il est possible de compresser sur des S- Boxes de 16 et 32 bits ainsi que la probabilité de détecter qu'une S-Boxe soit biaisée avec différentes tailles d'échantillons de lignes L_i testés. La première colonne indique la taille de l'échantillon choisi aléatoirement de paires d'entrée-sortie de la S-Boxe. La deuxième colonne indique le P_F utilisé. La troisième colonne représente le taux d'erreur maximal que nous tolérons sur P_S par rapport à l'échantillon avant de considérer que la S-Boxe est biaisée. Ainsi, si nous obtenons un $P_S \geq P_F + \varepsilon$, alors nous considérons que la S-Boxe est biaisée. La quatrième colonne indique la probabilité que la S-Boxe soit biaisée si nous obtenons $P_S \leq P_F + \varepsilon$ à partir de l'échantillon testé. Les deux dernières colonnes du tableau indiquent le nombre de bits gagnés dans la compression du tableau lorsque les S-Boxes ont respectivement 16 et 32 bits.

Tableau 6.II – Niveau de compression en fonction du biais

Échantillon testé	P_F	ε	Probabilité biais	Compression 16 bits	Compression 32 bits
1000	.5	.002	.525	.76	49570.8
1000	.75	.002	.519	1.01	66212.2
1000	.9	.002	.514	2.11	138521
1000	.5	.01	.624	19	123935
1000	.75	.01	.634	25	166739
1000	.9	.01	.682	54	354990
1000	.5	.025	.794	118.23	774864
1000	.75	.025	.824	161.3	1057090
1000	.9	.025	.91	356.25	2334710
10000	.5	.002	.579	2.11	49570.8
10000	.5	.0002	.508	.0076	495.71

Tous ces éléments nous permettent de conclure que, pour qu'une S-Boxe ait une bonne probabilité d'être biaisée par rapport à une possible fonction F, il faut que le nombre de bits dont nous sommes en mesure de compresser la S-Boxe soit le plus faible possible. Le seul moyen de diminuer la compression consiste à prendre en compte la taille de la description de la fonction envers laquelle les S-Boxes sont biaisées. Le but de déterminer la quantité de compression possible était de pouvoir déterminer la taille des descriptions des fonctions envers lesquelles nos S-Boxes pouvaient être biaisées.

Ainsi, nous pouvons déterminer s'il existe un biais fort envers une fonction simple ou si les biais forts ne sont possibles qu'envers des fonctions très complexes. L'analyse que nous avons effectuée permet de constater que, pour avoir un biais de plus de 10% avec une probabilité de $1/16$, il faut que la description de la fonction F prenne plus de 1900 bits et plus de 126 473 230 bits avec des S-Boxes de 16 et 32 bits respectivement et $P_F = .5$. Afin d'analyser l'ordre de grandeur des probabilités, nous rappelons que la probabilité de gagner à la loterie *649* est d'une chance sur 14 millions, ce qui est plus grand qu'une chance sur 2^{27}. Par conséquent, si la description de la fonction pour laquelle nous voulons que les S-Boxes soient biaisées prend plus de 26 bits de moins que le gain de compression, alors il est plus probable de gagner à la loterie que nos S-Boxes soient biaisées envers cette fonction. Or, il est possible de constater que nous avons analysé la résistance des S-Boxes en allant jusque dans les quadrillions de bits de compression. À ce niveau, il n'est plus question de chance ou de malchance concernant le fait d'obtenir une S-Boxe biaisée mais plutôt d'impossibilité. Le principe important à noter est que ce qui détermine la probabilité d'existence d'une fonction correspond à la différence de taille entre la description de la fonction et le gain obtenu par la compression du tableau. Si nous avons une différence de 100 bits, alors la probabilité est de $1/2^{100}$, ce qui est impossible.

Un procédé efficace afin de représenter le comportement d'une fonction est d'élaborer un circuit logique l'implémentant. La taille et la complexité des circuits sont proportionnelles à la complexité de Kolmogorov des fonctions. Autrement dit, plus la description d'une fonction sera complexe, plus le circuit la représentant sera complexe également. À titre d'exemple, une caractéristique linéaire s'exprime avec au plus $2n+1$ bits, où n est la taille des S-Boxes. Nous pouvons donc conclure qu'il est improbable que notre système soit vulnérable à des attaques qui peuvent être décrites par un circuit petit ou simple, principalement avec des S-Boxes de 32 bits.

CHAPITRE 7

CONCLUSION

En conclusion, l'ensemble de ce mémoire explique pourquoi nous croyons que *SAND* est au moins aussi sécuritaire que AES ou tout autre cryptosystème à clé symétrique actuellement utilisé sur le marché. De plus, à cause de sa simplicité de structure, nous pouvons garantir l'absence de portes cachées ou d'autres types de moyens détournés pour déchiffrer les messages encodés par notre protocole.

L'ensemble des résultats présentés dans ce mémoire a été réalisé en collaboration avec Alain Tapp. Voici une liste de mes contributions à ce projet :

- analyse de la littérature afin d'analyser le potentiel du cryptosystème ;
- découverte des faiblesses reliées à l'utilisation d'un *butterfly* dans le schéma ;
- analyse des propriétés des S-Boxes ;
- analyse des propriétés d'avalanche du système ;
- analyse de la résistance aux techniques de cryptanalyses ;
- cryptanalyse du système jusqu'à cinq *rounds* ;
- analyse de la résistance aux biais en suivant les recommandations d'Alain Tapp.

Il reste par contre les quelques éléments suivants à analyser avant que le protocole puisse être convenablement déployé :

- l'implémentation physique du protocole : malgré l'ensemble de nos tests, nous n'avons pas implémenté notre système dans un environnement dédié à celui-ci.
- les propriétés d'efficacité : avec une implémentation physique du protocole, il sera possible d'obtenir des propriétés d'efficacité mesurable de notre protocole telles que le temps de chiffrement et le temps de génération des diverses sections du protocole.
- la résistance aux attaques d'effets de bord (*Side-effect attack*) : il existe plusieurs types d'attaques basées sur l'implémentation physique d'un protocole que nous n'avons pas analysé dans ce mémoire. Pour contrer ces attaques, il faudra déterminer si l'utilisation de certaines clés peut être détectable par le temps de chif-

frement ou l'énergie utilisée par le protocole ainsi que d'autres propriétés propres aux attaques d'effets de bord.

Ces trois éléments concernent des aspects plus techniques et pratiques du cryptosystème.

BIBLIOGRAPHIE

[1] Carlisle Adams and Stafford Tavares. Good s-boxes are easy to find. In *Advances in Cryptology-CRYPTO'89 Proceedings*, pages 612–615. Springer, 1990.

[2] Carlisle Adams and Stafford Tavares. The structured design of cryptographically good s-boxes. *journal of Cryptology*, 3(1) :27–41, 1990.

[3] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent : A proposal for the advanced encryption standard. *NIST AES Proposal*, 174, 1998.

[4] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1) :3–72, 1991.

[5] Alex Biryukov. Block ciphers and stream ciphers : The state of the art. *IACR Cryptology ePrint Archive*, 2004 :94, 2004.

[6] Alex Biryukov and Adi Shamir. Structural cryptanalysis of sasas. In *Advances in Cryptology-EUROCRYPT 2001*, pages 395–405. Springer, 2001.

[7] Céline Blondeau, Benoît Gérard, and Kaisa Nyberg. Multiple differential cryptanalysis using llr and χ 2 statistics. In *Security and Cryptography for Networks*, pages 343–360. Springer, 2012.

[8] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full aes. In *Advances in Cryptology–ASIACRYPT 2011*, pages 344–371. Springer, 2011.

[9] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Francois-Xavier Standaert, John Steinberger, and Elmar Tischhauser. Key-alternating ciphers in a provable setting : encryption using a small number of public permutations. In *Advances in Cryptology–EUROCRYPT 2012*, pages 45–62. Springer, 2012.

[10] Johan Borst, Lars R Knudsen, and Vincent Rijmen. Two attacks on reduced idea. In *Advances in CryptologyŮEUROCRYPTŠ97*, pages 1–13. Springer, 1997.

[11] Joseph Alexander Brown, Sheridan Houghten, and Beatrice Ombuki-Berman. Genetic algorithm cryptanalysis of a substitution permutation network. In *Computational Intelligence in Cyber Security, 2009. CICS'09. IEEE Symposium on*, pages 115–121. IEEE, 2009.

[12] Lawrence Brown and Jennifer Seberry. On the design of permutation p in des type cryptosystems. In *Advances in Cryptology-EUROCRYPT'89*, pages 696–705. Springer, 1990.

[13] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr, Luke O'Connor, Mohammad Peyravian, Jr. Luke, O'connor Mohammad Peyravian, David Stafford, and Nevenko Zunic. Mars - a candidate cipher for aes. *NIST AES Proposal*, 1999.

[14] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M Matyas, Luke O'Connor, Mohammad Peyravian, David Safford, et al. The mars encryption algorithm. *IBM, August*, 27, 1999.

[15] Baudoin Collard and François-Xavier Standaert. Experimenting linear cryptanalysis. *Advanced Linear Cryptanalysis of Block and Stream Ciphers. Cryptology and Information Security Series*, 7, 2011.

[16] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM journal of research and development*, 38(3) :243–250, 1994.

[17] COSIC. Nessie. Disponible à l'adresse `http://www.cosic.esat.kuleuven.be/nessie/`. Dernier accès le 08 mai 2014.

[18] Nicolas T Courtois. The inverse s-box, non-linear polynomial relations and cryptanalysis of block ciphers. In *Advanced Encryption Standard–AES*, pages 170–188. Springer, 2005.

[19] Nicolas T Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology-ASIACRYPT 2002*, pages 267–287. Springer, 2002.

[20] Paul Crowley. Truncated differential cryptanalysis of five rounds of salsa20. *IACR Cryptology ePrint Archive*, 2005 :375, 2005.

[21] CRYPTREC. Cryptrec. Disponible à l'adresse `http://www.cryptrec.go.jp/english/`. Dernier accès le 08 mai 2014.

[22] Joan Daemen, Joan Daemen, Joan Daemen, Vincent Rijmen, and Vincent Rijmen. Aes proposal : Rijndael, 1998.

[23] MH Dawson and Stafford E Tavares. An expanded set of s-box design criteria based on information theory and its relation to differential-like attacks. In *Advances in Cryptology-EUROCRYPT'91*, pages 352–367. Springer, 1991.

[24] Jean-Paul Delahaye. Information noyée, information cachée. Disponible à l'adresse `http://apprendre-en-ligne.net/crypto/stegano/229_142_146.pdf`. Dernier accès le 2 septembre 2014.

[25] John Detombe and Stafford Tavares. Constructing large cryptographically strong s-boxes. In *Advances in Cryptology-AUSCRYPT'92*, pages 165–181. Springer, 1993.

[26] Réjane Forré. Methods and instruments for designing s-boxes. *Journal of Cryptology*, 2(3) :115–130, 1990.

[27] Reto Gali. Rapport sur mars. Disponible à l'adresse `http://reto.orgfree.com/us/projectlinks/MARSReport.html`. Dernier accès le 10 février 2014.

[28] JA Gordon and H Retkin. Are big s-boxes best ? In *Cryptography*, pages 257–262. Springer, 1983.

[29] Howard M Heys and Stafford E Tavares. Substitution-permutation networks resistant to differential and linear cryptanalysis. *Journal of cryptology*, 9(1) :1–19, 1996.

[30] Thomas Jakobsen. Higher-order cryptanalysis of block ciphers. 1999.

[31] John B. Kam and George I Davida. Structured design of substitution-permutation encryption networks. *Computers, IEEE Transactions on*, 100(10) :747–753, 1979.

[32] Ju-Sung Kang, Seokhie Hong, Sangjin Lee, Okyeon Yi, Choonsik Park, and Jongin Lim. Practical and provable security against differential and linear cryptanalysis for substitution-permutation networks. *ETRI journal*, 23(4) :158–167, 2001.

[33] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography : principles and protocols*. CRC Press, 2007.

[34] Liam Keliher. *Substitution-permutation network cryptosystems using key-dependent s-boxes.* Citeseer, 1998.

[35] Lars R Knudsen. Truncated and higher order differentials. In *Fast Software Encryption*, pages 196–211. Springer, 1995.

[36] Lars R Knudsen and Thomas A Berson. Truncated differentials of safer. In *Fast Software Encryption*, pages 15–26. Springer, 1996.

[37] Lars R Knudsen, Matthew JB Robshaw, and David Wagner. Truncated differentials and skipjack. In *Advances in CryptologyŮCRYPTOŠ99*, pages 165–180. Springer, 1999.

[38] Seonhee Lee, Seokhie Hong, Sangjin Lee, Jongin Lim, and Seonhee Yoon. Truncated differential cryptanalysis of camellia. In *Information Security and CryptologyŮICISC 2001*, pages 32–38. Springer, 2002.

[39] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Advances in Cryptology-EUROCRYPT'93*, pages 386–397. Springer, 1994.

[40] Mitsuru Matsui and Toshio Tokita. Cryptanalysis of a reduced version of the block cipher e2. In *Fast Software Encryption*, pages 71–80. Springer, 1999.

[41] Willi Meier and Othmar Staffelbach. Nonlinearity criteria for cryptographic functions. In *Advances in Cryptology-EUROCRYPT'89*, pages 549–562. Springer, 1990.

[42] Ralph C Merkle. Fast software encryption functions. In *Advances in Cryptology-CRYPT0'90*, pages 477–501. Springer, 1991.

[43] Chris Mitchell. Enumerating boolean functions of cryptographic significance. *Journal of Cryptology*, 2(3) :155–170, 1990.

[44] Shiho Moriai and Yiqun Lisa Yin. Cryptanalysis of two sh (ii).

[45] Nicky Mouha. Automated techniques for hash function and block cipher cryptanalysis (automatische technieken voor hashfunctie-en blokcijfercryptanalyse). 2012.

[46] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Information Security and Cryptology*, pages 57–76. Springer, 2012.

[47] Ghulam Murtaza, Azhar Ali Khan, Syed Wasi Alam, and Aqeel Farooqi. Fortification of aes with dynamic mix-column transformation. *IACR Cryptology ePrint Archive*, 2011 :184, 2011.

[48] Kaisa Nyberg. Perfect nonlinear s-boxes. In *Advances in Cryptology-EUROCRYPT'91*, pages 378–386. Springer, 1991.

[49] Luke O'Connor. Enumerating nondegenerate permutations. In *Advances in Cryptology-EUROCRYPT'91*, pages 368–377. Springer, 1991.

[50] Luke O'Connor. An analysis of a class of algorithms for s-box construction. *Journal of Cryptology*, 7(3) :133–151, 1994.

[51] Ronald L Rivest, MJB Robshaw, Ray Sidney, and Yiqun Lisa Yin. The rc6 block cipher. In *First Advanced Encryption Standard (AES) Conference*, 1998.

[52] Przemysław Rodwald and Piotr Mroczkowski. How to create "good" s-boxes ? In *1st International Conference for Young Researchers in Computer Science, Control, Electrical Engineering and Telecommunications–ICYR, Zielona Góra, Poland*, 2006.

[53] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption*, pages 191–204. Springer, 1994.

[54] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish : A 128-bit block cipher. *NIST AES Proposal*, 15, 1998.

[55] Jennifer Seberry, Xian-Mo Zhang, and Yuliang Zheng. Systematic generation of cryptographically robust s-boxes. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 171–182. ACM, 1993.

[56] Jennifer Seberry, Xian-Mo Zhang, and Yuliang Zheng. Relationships among nonlinearity criteria. In *Advances in Cryptology-EUROCRYPT'94*, pages 376–388. Springer, 1995.

[57] Douglas Stinson. Cryptographie-théorie et pratique. Technical report, International Thomson Publishing France, 1996.

[58] Guoping Tang, Xiaofeng Liao, and Yong Chen. A novel method for designing s-boxes based on chaotic maps. *Chaos, Solitons & Fractals*, 23(2) :413–419, 2005.

[59] Justin Troutman. Alice tutors bob on mature and minimalist cryptography. 2009.

[60] Serge Vaudenay. An experiment on des statistical cryptanalysis. In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 139–147. ACM, 1996.

[61] Isil Vergili and Melek D Yücel. On satisfaction of some security criteria for randomly chosen s-boxes. In *Proc. 20th Biennial Symp. on Communications, Kingston (May 2000)*, pages 64–68.

[62] Isil Vergili and Melek D Yücel. Avalanche and bit independence properties for the ensembles of randomly chosen$\times$ s-boxes. *Turk J Elec Engin*, 9(2) :137–145, 2001.

[63] David Wagner. The boomerang attack. In *Fast Software Encryption*, pages 156–170. Springer, 1999.

[64] AF Webster and Stafford E Tavares. On the design of s-boxes. In *Advances in Cryptology-CRYPTO'85 Proceedings*, pages 523–534. Springer, 1986.

[65] Wikipedia. Aes, . Disponible à l'adresse http://en.wikipedia.org/wiki/Advanced_Encryption_Standard. Dernier accès le 08 mai 2014.

[66] Wikipedia. Cryprec, . Disponible à l'adresse http://en.wikipedia.org/wiki/CRYPTREC. Dernier accès le 08 mai 2014.

[67] Wikipedia. Des, . Disponible à l'adresse http://en.wikipedia.org/wiki/Data_Encryption_Standard. Dernier accès le 08 mai 2014.

[68] Wikipedia. Documentation supplémentaire sur des, . Disponible à l'adresse http://en.wikipedia.org/wiki/DES_supplementary_material#Substitution_boxes_.28S-boxes.29. Dernier accès le 08 mai 2014.

[69] Wikipedia. Réseau de feistel, . Disponible à l'adresse http://en.wikipedia.org/wiki/Feistel_cipher. Dernier accès le 08 mai 2014.

[70] Wikipedia. Réseau de feistel, . Disponible à l'adresse http://fr.wikipedia.org/wiki/R%C3%A9seau_de_Feistel. Dernier accès le 08 mai 2014.

[71] Wikipedia. Rc6, . Disponible à l'adresse http://en.wikipedia.org/wiki/RC6. Dernier accès le 08 mai 2014.

[72] Wikipedia. Réseau de substitution et permutation, . Disponible à l'adresse http://http://en.wikipedia.org/wiki/Substitution-permutation_network. Dernier accès le 08 mai 2014.

[73] Wikipedia. Attaque xsl, . Disponible à l'adresse http://en.wikipedia.org/wiki/XSL_attack. Dernier accès le 08 mai 2014.

[74] Wikipedia. Cryptanalyse boomerang, . Disponible à l'adresse http://en.wikipedia.org/wiki/Boomerang_attack. Dernier accès le 15 janvier 2014.

[75] Wikipedia. Cryptanalyse différentielle, . Disponible à l'adresse http://en.wikipedia.org/wiki/Differential_cryptanalysis. Dernier accès le 05 mars 2014.

[76] Wikipedia. Mars, . Disponible à l'adresse http://en.wikipedia.org/wiki/MARS_(cryptography). Dernier accès le 02 juillet 2013.

[77] Wikipedia. Lemme du piling-up, . Disponible à l'adresse http://fr.wikipedia.org/wiki/Lemme_Piling-Up. Dernier accès le 08 mai 2014.

[78] Wikipedia. Serpent, . Disponible à l'adresse http://en.wikipedia.org/wiki/Serpent_(cipher). Dernier accès le 08 mai 2014.

[79] Wikipedia. Cryptanalyse différentielle tronqué, . Disponible à l'adresse http://en.wikipedia.org/wiki/Truncated_differential_cryptanalysis. Dernier accès le 13 janvier 2014.

[80] Wikipedia. Twofish, . Disponible à l'adresse http://en.wikipedia.org/wiki/Twofish. Dernier accès le 08 mai 2014.

[81] Wikipedia. Twofish, . Disponible à l'adresse http://fr.wikipedia.org/wiki/Twofish. Dernier accès le 08 mai 2014.

[82] Paul D Yacoumis. On the security of the advanced encryption standard, 2005.

[83] AM Youssef, S Mister, and SE Tavares. On the design of linear transformations for substitution permutation encryption networks. Citeseer.

[84] Amr M Youssef and Stafford E Tavares. Resistance of balanced s-boxes to linear and differential cryptanalysis. *Information Processing Letters*, 56(5) :249–252, 1995.

[85] Liang Zhao, Di Xiao, and Kouichi Sakurai. Image encryption design based on multi-dimensional matrix map and partitioning substitution and diffusion-integration substitution network structure. In *Information Science and Applications (ICISA), 2010 International Conference on*, pages 1–8. IEEE, 2010.

Annexe I

Pseudocode RC6

Le pseudocode provient de [71].

```
Encryption/Decryption with RC6-w/r/b

Input:
Plaintext stored in four w-bit input registers A, B, C & D
r is the number of rounds
w-bit round keys S[0, ... , 2r + 3]

Output: Ciphertext stored in A, B, C, D

Encryption Procedure:

B = B + S[0]
D = D + S[1]
for i = 1 to r do
{
    t = (B*(2B + 1)) <<< lg w
    u = (D*(2D + 1)) <<< lg w
    A = ((A ? t) <<< u) + S[2i]
    C = ((C ? u) <<< t) + S[2i + 1]
    (A, B, C, D)  =  (B, C, D, A)
}
A = A + S[2r + 2]
C = C + S[2r + 3]

Decryption Procedure:
```

```
C = C - S[2r + 3]
A = A - S[2r + 2]

for i = r downto 1 do
{
   (A, B, C, D) = (D, A, B, C)
    u = (D*(2D + 1)) <<< lg w
    t = (B*(2B + 1)) <<< lg w
    A = ((A - S[2i]) >>> u) ? t
}
D = D - S[1]
B = B - S[0]
```

Annexe II

S-Boxe de DES

Les tableaux suivants proviennent de [68]

S-boxes

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
S_1																
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2																
	x0000x	**x0001x**	**x0010x**	**x0011x**	**x0100x**	**x0101x**	**x0110x**	**x0111x**	**x1000x**	**x1001x**	**x1010x**	**x1011x**	**x1100x**	**x1101x**	**x1110x**	**x1111x**
0yyyy0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
0yyyy1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
1yyyy0	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
1yyyy1	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
	x0000x	**x0001x**	**x0010x**	**x0011x**	**x0100x**	**x0101x**	**x0110x**	**x0111x**	**x1000x**	**x1001x**	**x1010x**	**x1011x**	**x1100x**	**x1101x**	**x1110x**	**x1111x**
0yyyy0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
0yyyy1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
1yyyy0	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1yyyy1	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
	x0000x	**x0001x**	**x0010x**	**x0011x**	**x0100x**	**x0101x**	**x0110x**	**x0111x**	**x1000x**	**x1001x**	**x1010x**	**x1011x**	**x1100x**	**x1101x**	**x1110x**	**x1111x**
0yyyy0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
0yyyy1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
1yyyy0	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
1yyyy1	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Figure II.1 – S-Boxe 1 à 4 de DES

S_5																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
0yyyy1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
1yyyy0	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
1yyyy1	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
0yyyy1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
1yyyy0	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
1yyyy1	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
0yyyy1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1yyyy0	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
1yyyy1	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
0yyyy1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
1yyyy0	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
1yyyy1	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Figure II.2 – S-Boxe 5 à 8 de DES

Annexe III

Recommandations NESSIE

- Block ciphers
  - MISTY1 : Mitsubishi Electric
  - Camellia : Nippon Telegraph and Telephone and Mitsubishi Electric
  - SHACAL-2 : Gemplus
  - AES* : (Advanced Encryption Standard) (NIST, FIPS Pub 197) (aka Rijndael)
- Public-key encryption
  - ACE Encrypt# : IBM Zurich Research Laboratory
  - PSEC-KEM : Nippon Telegraph and Telephone Corp
  - RSA-KEM* : RSA key exchange mechanism (draft of ISO/IEC 18033-2)
- MAC algorithms and cryptographic hash functions
  - Two-Track-MAC : Katholieke Universiteit Leuven and debis AG
  - UMAC : Intel Corp, Univ. of Nevada at Reno, IBM Research Laboratory, Technion Institute, and Univ. of California at Davis
  - CBC-MAC* : (ISO/IEC 9797-1) ;
  - HMAC* : (ISO/IEC 9797-1) ;
  - WHIRLPOOL : Scopus Tecnologia S.A. and K.U.Leuven
  - SHA-256*, SHA-384* and SHA-512* : NSA, (US FIPS 180-2)
- Digital signature algorithms
  - ECDSA# : Certicom Corp
  - RSA-PSS : RSA Laboratories
  - SFLASH : Schlumberger Corp (SFLASH was broken in 2007[1] and should not be used anymore).
- Identification schemes
  - GPS-auth : Ecole Normale Supérieure, France Télécom, and La Poste

(* représente les algorithmes déjà connus, # représente les algorithmes qui sont sous une license payante)

Annexe IV

Recommandation CRYPTREC

NB : italics denote contingent recommendation condition(s)

- Public Key Algorithms (aka asymmetric key algorithms w/ public/private key property)
  - authentication – none recommended
  - signature
    - DSA
    - ECDSA (ANSI X9.62, SEC 1) (empirically secure)
    - ESIGN (forgeable factor discovered does not have provable security)
    - TSH-ESIGN (does not have provable security)
    - RSA-PSS (provably secure)
    - RSASSA-PKCS1 v1.5 (empirically secure)
  - confidentiality
    - ECIES (vulnerable to chosen plaintext attacks)
    - HIME(R) (no provable security, specification errors)
    - RSA-OAEP (provably secure)
    - RSAES-PKCS1 v1.5 (Permitted 'for the time being' (as empirically secure), due to use in SSL3.0/TSL1.0 – use only with maximum caution)
  - key agreement
    - DH (empirically secure)
    - ECDH (empirically secure)
    - PSEC-KEM (recommended only in Data Encapsulation Mechanism construction w/ elliptic curve parameters as defined by SEC 1)
- Symmetric Key Cipher Algorithms
  - 64-bit block ciphers (128 bit block ciphers are preferable if possible)
    - CIPHERUNICORN-E
    - Hierocrypt-L1

    - MISTY1
    - 3-key Triple DES (Permitted 'for the time being' if used as specified in FIPS Pub 46-3, and if specified as a de facto standard)
  - 128-bit block ciphers – only 128-bit block ciphers are recommended
    - AES
    - Camellia
    - CIPHERUNICORN-A
    - Hierocrypt-3
    - SC2000
  - stream ciphers
    - MUG1
    - MULTI-S01
    - RC4 (128-bit keys only)
- Cryptographic Hash Algorithms (256-bit or larger digests are to be preferred in new designs. The two 160-bit digest algorithms listed are acceptable if already included in a current public key specification)
  - RIPEMD-160 (160 bit digest)
  - SHA-1 (160 bit digest)
  - SHA-256
  - SHA-384
  - SHA-512
- Cryptographic Pseudo-Random Number Generators - those listed are examples only, none is recommended
  - PRNG based on SHA-1 in ANSI X9.42-2001 Annex C.1
  - PRNG based on SHA-1 for general purposes in FIPS Pub 186-2 (inc change notice 1) Appendix 3.1
  - PRNG based on SHA-1 for general purposes in FIPS Pub 186-2 (inc change notice 1) revised Appendix 3.1