Conception, développement et analyse de systèmes de fonction booléennes décrivant les algorithmes de chiffrement et de déchiffrement de l'Advanced Encryption Standard

Michel Dubois

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

ARTS ET MÉTIERS ParisTech

2017-ENAM-0024

École doctorale n°432 : Sciences des Métiers de l'ingénieur

Doctorat ParisTech

THÈSE

pour obtenir le grade de docteur délivré par

l'**École Nationale Supérieure d'Arts et Métiers**
Spécialité Mathématiques appliquées

présentée et soutenue publiquement par

Michel Dubois

le 24 juillet 2017

Conception, développement et analyse de systèmes de fonctions booléennes décrivant les algorithmes de chiffrement et de déchiffrement de l'Advanced Encryption Standard

Directeur de thèse : **Éric Filiol**

Jury

M. Jean-Marc Steyaert	Professeur	École Polytechnique	Président
M. Maroun Chamoun	Professeur	Université Saint Joseph de Beyrouth	Rapporteur
M. Radu State	Professeur	Université du Luxembourg	Rapporteur
M. Robert Erra	Docteur	EPITA	Examinateur
M. Éric Filiol	Professeur	ESIEA	Examinateur
M. Éric Jaeger	Docteur	Ministère de la Défense	Examinateur

À Delphine, mon épouse

À Clotilde, Stanislas,
Astrid, Marie-Liesse,
et Félicité, nos enfants

Remerciements

« Labor omnia vincit improbus »

Virgile – Géorgiques

Lorsque, quelques années en arrière, je me suis lancé dans cette thèse, je savais que ce serait difficile et je n'imaginais pas que serait aussi éprouvant, mais, au bilan, j'ai vécu une expérience aussi passionnante qu'exaltante. Réalisée en grande partie sur mon temps libre, alors que j'ai une vie professionnelle, associative et familiale déjà bien dense, j'ai avancé pas à pas n'osant croire que j'y arriverai un jour. Maintenant que l'exercice arrive à son terme, et à la vue des obstacles franchis, je prends conscience du soutien important dont ma famille, mes amis et mes collègues m'ont gratifié. À tous je veux dire ma gratitude.

Je voudrais remercier en premier lieu mon directeur de thèse, Éric Filiol. Devant mon insistance et ma ténacité à entrer dans cette démarche, et malgré l'éloignement géographique et les multiples difficultés en perspective, il a décidé de m'accorder sa confiance. S'en sont suivis un soutien et un accompagnement sans faille, me faisant entrer de plein pied dans le monde fascinant de la recherche scientifique. Je le remercie pour son amitié, sa confiance et sa disponibilité au travers desquelles j'ai retrouvé le chef militaire et l'officier qu'il est dans l'âme. Enfin, Éric a su me communiquer sa passion pour les mathématiques et la cryptographie, ainsi que sa force de travail et je fais mienne sa philosophie, résumée dans l'introduction de son premier livre sur les virus informatiques : *« Il ne faut pas se complaire du peu que l'on a pu apprendre mais toujours regarder la masse impressionnante et insolente de ce que l'on ignore »*. Merci Éric.

Mes remerciements vont ensuite aux autres membres du jury pour avoir bien voulu lire et valider mes travaux. En premier lieu, je remercie très chaleureusement Jean-Marc Steyaert pour avoir bien voulu présider le jury. À Maroun Chamoun et Radu State je veux dire ma gratitude pour m'avoir fait l'honneur d'être les rapporteurs de cette thèse. Je remercie enfin Robert Erra et Éric Jaeger pour leur examen attentif du manuscrit.

Ce travail de recherche m'a permis d'intégrer un grand laboratoire : le laboratoire de virologie et de cryptologie opérationnelles. C'est un grand laboratoire par la richesse des thèmes de recherche qui y sont étudiés, par la haute technicité, l'ouverture et le dynamisme de ses chercheurs, par l'humilité et la bonne humeur qui les animent. Les moments passés avec eux ont été riches, merci Richard, Nicolas, Arnaud, Baptiste, Jonathan, Olivier, Laurence et Vincent.

Je tiens à remercier mon frère et mathématicien Loïc Dubois, mon ami Philippe Loudenot, mon fils Stanislas et ma femme Delphine pour avoir pris le temps de relire avec patience mes publications et mon mémoire de thèse. Leurs remarques m'ont permis d'en améliorer la qualité et la lisibilité quelle que soit la langue utilisée.
Je remercie particulièrement mes chefs et notamment le médecin chef des services Éric Multon, le médecin général Jean-Jacques Mascart et le médecin général Alain Codaccioni. Ils ont non seulement permis que cette thèse puisse se dérouler dans de bonnes conditions mais ils m'ont également apporté leur soutien et leurs encouragements. Qu'ils reçoivent ici toute ma gratitude.
Une fois n'est pas coutume, je souhaite profiter de ces remerciements pour mettre à l'honneur un grand compositeur : Jean Sébastien Bach et deux de ses plus grands interprètes : les pianistes Glenn Gould et Alexandre Tharaud. Se plonger dans une séance de réflexion intense sur un problème mathématique ou dans la programmation d'une primitive cryptographique, accompagné des préludes et fugues du clavier bien tempéré, des partitas ou des variations Goldberg interprétés par Glenn Gould ou Alexandre Tharaud, relève du pur bonheur.
Je voudrais également remercier mes beaux-parents. Durant mes multiples séjours à Laval, ils m'ont toujours ouvert en grand leur maison, me donnant ainsi un toit où dormir, mais surtout, m'offrant un espace de détente et de réconfort chaleureux et familial. Merci pour votre fidélité et votre patience.
Pour terminer, je remercie affectueusement ma famille. Delphine, ma femme, pendant ces années et pour que je puisse mener à bien ce projet, tu as accepté de nombreux sacrifices et malgré cela tu m'as toujours soutenu et encouragé quoiqu'il t'en as coûté. Sans ton amour, ta confiance, ton soutien et ta patience je n'y serai pas arrivé. Clotilde, Stanislas, Astrid, Marie-Liesse et Félicité, mes enfants, tout au long de ces années, vous avez accepté d'avoir un papa en pointillé et pourtant vous m'avez soutenu et encouragé, ne serait-ce que par l'admiration et la fierté que je sentais dans vos regards. Ma femme et mes enfants, ce travail vous est dédié.

Table des matières

Table des figures

Liste des codes source

Résumé

La cryptologie est une des disciplines des mathématiques, elle est composée de deux sous-ensembles : la cryptographie et la cryptanalyse. Tandis que la cryptographie s'intéresse aux algorithmes permettant de modifier une information afin de la rendre inintelligible sans la connaissance d'un secret, la seconde s'intéresse aux méthodes mathématiques permettant de recouvrer l'information originale à partir de la seule connaissance de l'élément chiffré [1].
La cryptographie se subdivise elle-même en deux sous-ensembles : la cryptographie symétrique et la cryptographie asymétrique. La première utilise une clef identique pour les opérations de chiffrement et de déchiffrement, tandis que la deuxième utilise une clef pour le chiffrement et une autre clef, différente de la précédente, pour le déchiffrement. Enfin, la cryptographie symétrique travaille soit sur des blocs d'information soit sur des flux continus d'information. Ce sont les algorithmes de chiffrement par blocs qui nous intéressent ici.
L'objectif de la cryptanalyse est de retrouver l'information initiale sans connaissance de la clef de chiffrement et ceci dans un temps plus court que l'attaque par force brute. Il existe de nombreuses méthodes de cryptanalyse comme la cryptanalyse fréquentielle, la cryptanalyse différentielle, la cryptanalyse intégrale, la cryptanalyse linéaire...
Beaucoup de ces méthodes sont maintenues en échec par les algorithmes de chiffrement modernes. En effet, dans un jeu de la lance et du bouclier, les cryptographes développent des algorithmes de chiffrement de plus en plus efficaces pour protéger l'information chiffrée d'une attaque par cryptanalyse. C'est le cas notamment de l'Advanced Encryption Standard (AES). Cet algorithme de chiffrement par blocs a été conçu par Joan Daemen et Vincent Rijmen et transformé en standard par le National Institute of Standards and Technology (NIST) en 2001. Afin de contrer les méthodes de cryptanalyse usuelles les concepteurs de l'AES lui ont donné une forte structure algébrique.
Ce choix élimine brillamment toute possibilité d'attaque statistique, cependant, de récents travaux tendent à montrer, que ce qui est censé faire la robustesse de l'AES, pourrait se révéler être son point faible. En effet, selon ces études, cryptanalyser l'AES se "résume" à résoudre un système d'équations quadratiques symbolisant la structure du chiffrement de l'AES [49]. Malheureusement, la taille du système d'équations obtenu et le manque d'algorithmes de résolution

1. Dans certains cas la cryptanalyse utilise également des éléments d'information non chiffrés, nous parlons alors de cryptanalyse à clair connu ou à clair choisi.

efficaces font qu'il est impossible, à l'heure actuelle, de résoudre de tels systèmes dans un temps raisonnable.
L'enjeu de cette thèse est, à partir de la structure algébrique de l'AES, de décrire son algorithme de chiffrement et de déchiffrement sous la forme d'un nouveau système d'équations booléennes. Puis, en s'appuyant sur une représentation spécifique de ces équations, d'en réaliser une analyse combinatoire afin d'y détecter d'éventuels biais statistiques.

Mots clés

<Cryptographie>, <Cryptanalyse>, <AES>, <Combinatoire>, <Structure algébrique>, <Fonction booléenne>

Abstract

Cryptology is one of the mathematical fields, it is composed of two subsets : cryptography and cryptanalysis. While cryptography focuses on algorithms to modify an information by making it unintelligible without knowledge of a secret, the second focuses on mathematical methods to recover the original information from the only knowledge of the encrypted element [2].
Cryptography itself is subdivided into two subsets : symmetric cryptography and asymmetric cryptography. The first uses the same key for encryption and decryption operations, while the second uses one key for encryption and another key, different from the previous one, for decryption. Finally, symmetric cryptography is working either on blocks of information either on continuous flow of information. It is the block encryption algorithms that interest us here
The aim of cryptanalysis is to recover the original information without knowing the encryption key and this, into a shorter time than the brute-force attack. There are many methods of cryptanalysis as frequency cryptanalysis, differential cryptanalysis, integral cryptanalysis, linear cryptanalysis...
Many of these methods are defeated by modern encryption algorithms. Indeed, in a game of spear and shield, cryptographers develop encryption algorithms more efficient to protect the encrypted information from an attack by cryptanalysis. This is the case of the Advanced Encryption Standard (AES). This block cipher algorithm was designed by Joan Daemen and Vincent Rijmen and transformed into standard by the National Institute of Standards and Technology (NIST) in 2001. To counter the usual methods of cryptanalysis of AES designers have given it a strong algebraic structure.
This choice eliminates brilliantly any possibility of statistical attack, however, recent work suggests that what is supposed to be the strength of the AES, could prove to be his weak point. According to these studies, the AES cryptanalysis comes down to "solve" a quadratic equations symbolizing the structure of the AES encryption [49]. Unfortunately, the size of the system of equations obtained and the lack of efficient resolution algorithms make it impossible, at this time, to solve such systems in a reasonable time.
The challenge of this thesis is, from the algebraic structure of the AES, to describe its encryption and decryption processes in the form of a new Boolean equations system. Then, based on a specific representation of these equations,

2. In some cases cryptanalysis also uses elements of unencrypted information, then we are talking about known plaintext or chosen plaintext cryptanalysis.

to achieve a combinatorial analysis to detect potential statistical biases.

Keywords

<Cryptography>, <Cryptanalysis>, <AES>, <Combinatorics>, <Algebraic structure>, <Boolean function>

Chapitre 1

Introduction

Durant ces années de recherche, lorsque mon entourage m'interrogeait sur le sujet de ma thèse, une fois passés les premiers instants d'étonnement devant l'énoncé, la question qui revenait systématiquement portait sur la cryptographie. L'étonnement se muait alors en incrédulité lorsque j'expliquais que nous l'utilisons régulièrement dans nos activités quotidiennes. En effet, que ce soit dans les communications téléphoniques, lors d'un paiement par carte bancaire ou la consultation d'un site Web sécurisé, la cryptographie, autrefois réservée aux militaires, est aujourd'hui utilisée par chacun d'entre nous.
La cryptographie est la science qui assure la sécurité de nos informations. Durant les vingt dernières décennies, elle a investi le marché des produits de masse et chaque citoyen en fait un usage quotidien. Elle est notamment utilisée pour l'authentification et le chiffrement dans les cartes bancaires, la téléphonie sans fil, le commerce électronique, le paiement des impôts en ligne ou la télévision à péage. Elle est également utilisée dans le domaine du contrôle d'accès pour ouvrir sa voiture à distance par exemple, dans la protection des œuvres numérique avec les techniques de marquages ou enfin, au cœur de nos démocraties avec le vote électronique.
La cryptographie est vieille comme le monde. Dès que l'homme a su communiquer, il s'est toujours trouvé des situations où deux êtres ont voulu échanger des confidences en se mettant à l'abri des oreilles indiscrètes. Dans le même temps, la curiosité ou l'intérêt ont poussé d'autres individus à chercher à entendre et à comprendre ces confidences. Dès lors, la lutte s'est engagée entre ceux qui voulaient protéger leur communication et ceux qui voulaient l'écouter.
La plus ancienne preuve de l'utilisation de la cryptographie [117] a été trouvée, en Égypte, dans une inscription, gravée vers 1900 avant Jésus-Christ, dans la chambre principale de la tombe du noble Khnumhotep II. Le scribe a utilisé des symboles hiéroglyphiques inhabituels à la place de ceux utilisés couramment. Le but n'étant pas tant de cacher le message, mais plutôt de changer sa forme d'une manière à enjoliver l'épitaphe de son maître.
D'autres civilisations ont utilisé la cryptographie durant l'antiquité, ainsi, certains scribes assyriens signaient leurs tablettes d'une suite de nombres. Ces nombres représentant leur nom sous forme chiffrée. De même, les hébreux, dans

le livre de Jérémie [1], ont utilisé un mécanisme de substitution appelé "ATBASH". Le principe consistait à remplacer la première lettre de l'alphabet *Aleph* par la dernière *Tav*, la seconde *Beth* par l'avant-dernière *Shin* et ainsi de suite. Le nom de la ville de "Babylone" devient alors "Sheshakh". Avec la Scytale, les lacédémoniens, ont inventé le chiffrement par permutation. Pour cacher leur message, ils enroulaient une ceinture sur un bâton, puis écrivaient le message sur la ceinture en suivant le sens longitudinal du bâton. Une fois la ceinture enlevée du bâton, le message devenait illisible. Enfin, les romains utilisaient le chiffre de César [117] consistant à substituer les caractères de l'alphabet en les décalant de trois rangs.

En Europe, ce n'est qu'à partir de la Renaissance que la cryptographie devient un art véritable et qu'elle acquiert une certaine importance dans les correspondances diplomatiques. En 1553, l'italien Giovan Batista Belaso, secrétaire du cardinal Rodolfo Pio di Carpi, décrit, dans son livre "*La cifra*", le premier algorithme de chiffrement poly-alphabétique. C'est un peu plus tard, en 1585, que le diplomate français Blaise Vigenère, secrétaire du roi Charles IX, publie son "*Traicté des Chiffres*" dans lequel il généralise le chiffrement de Belaso. Son algorithme restera inviolable jusqu'au milieu du XIX$^{\text{ème}}$ siècle.

La Renaissance marque également l'essor de la cryptanalyse. Ainsi, Leon Battista Alberti rédige, en 1466, le premier essai de cryptanalyse en Europe [2] [73] : "*De Componendis Cifris*". Il y décrit comment décrypter un message à partir de l'analyse de la fréquence des lettres dans les langues latines et italiennes. L'utilisation de la cryptanalyse se généralisant, les rois s'adjoignent des secrétaires chiffreurs particuliers. Ainsi, François 1$^{\text{er}}$ emploie Philibert Babou pour décrypter les dépêches interceptées. Henri IV s'assure des bons offices du cryptanalyste François Viète. Ce dernier parvient à casser les codes des lettres secrètes espagnoles. C'est ainsi, qu'en 1590, Henri IV l'autorise à rendre publique des lettres révélant que le chef de la Ligue en France, le duc de Mayenne, projette de devenir roi à la place d'Henri IV. La cryptographie et la cryptanalyse sont au cœur des conspirations, ainsi, Marie Stuart, emprisonnée par sa cousine, la reine Elisabeth 1$^{\text{ère}}$ d'Angleterre, complote contre celle-ci pour s'emparer du trône. Les messages qu'elle envoie à ses complices sont chiffrés par simple substitution. Ils seront interceptés et décryptés par les services de la reine et serviront de prétexte à sa condamnation à mort pour trahison.

Après une période d'apogée avec le roi Louis XIV et la dynastie Rossignol, la cryptographie va perdre de l'intérêt en France au profit d'actions d'espionnage. Ainsi, Auguste Kerckhoffs raconte [120] que, pendant les guerres du premier Empire, les généraux disposaient du grand chiffre et du petit chiffre pour correspondre entre eux et avec l'état-major général. Ces chiffres sont mal utilisés et les quelques correspondances chiffrées sont systématiquement interceptées et décryptées. Le Tsar Alexandre I$^{\text{er}}$ en fait état et décrit l'avantage stratégique que cela lui donne durant la campagne de Russie. En 1870, durant la guerre contre la Prusse, le général Bazaine et Napoléon III éprouvent des difficultés pour communiquer parce que leurs états-majors ne disposent pas des mêmes documents du chiffre.

1. Le livre de Jérémie est un livre de la Bible écrit par le prophète Jérémie. Sa rédaction commence en 585 avant Jésus-Christ, avant la destruction de Jérusalemen par les Babyloniens.

2. Nous précisons en Europe, car les premiers écrits de cryptanalyse sont arabes et remontent au IX$^{\text{ème}}$ siècle après Jésus-Christ. Ainsi, c'est al-Kindi [1] qui est le premier à expliquer comment utiliser l'analyse des fréquences des lettres pour décrypter un message.

De la fin du XIX^{ème} siècle jusqu'à la Première Guerre Mondiale, la cryptographie va connaître un nouvel essor. Durant cette période, elle quitte le monde diplomatique pour être pris en main par les militaires. C'est aussi à cet époque, en 1883, que le linguiste Auguste Kerckhoffs publie son essai sur la "*cryptographie militaire*" [120] dans lequel il énonce ses fameux principes :

1. le système doit être matériellement, sinon mathématiquement, indéchiffrable ;
2. il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;
3. la clef doit pouvoir en être communiquée et retenue sans le secours de notes écrites, et être changée ou modifiée au gré des correspondants ;
4. il faut qu'il soit applicable à la correspondance télégraphique ;
5. il faut qu'il soit portatif, et que son maniement ou son fonctionnement n'exige pas le concours de plusieurs personnes ;
6. enfin, il est nécessaire, vu les circonstances qui en commandent l'application, que le système soit d'un usage facile, ne demandant ni tension d'esprit, ni la connaissance d'une longue série de règles à observer.

Ces principes servent de base à toute la cryptographie moderne. Ils posent deux points clefs qui en font toute la pertinence : le secret ne réside pas dans le système, le secret réside dans la clef. Ainsi, tout algorithme de chiffrement doit pouvoir être rendu public sans risque de compromission, par contre la clef de chiffrement est la base sur laquelle repose la protection du système.
La Première Guerre Mondiale va être marquée par une succession d'échecs et de succès cryptographiques. Ainsi, La bataille de Tannenberg entre la Russie et la Prusse, perdue par le tsar Nicolas II, alors que son armée est supérieure en nombre, du fait de manque de moyens cryptographiques lui imposant de transmettre ses plans d'invasion en clair. De même, il faudra attendre le 9 novembre 1917 et le désastre de la bataille de Caporetto pour que les italiens prennent conscience de la faiblesse de leur chiffre systématiquement décrypté, depuis le début de la guerre, par les cryptanalystes autrichiens. Le 17 janvier 1917, c'est un message du ministre allemand des affaires étrangères Arthur Zimmermann, intercepté et décrypté par le service de renseignement britannique qui va précipiter les États-Unis dans la guerre. Ce message secret, à destination de l'ambassade d'Allemagne à Mexico, proposait une alliance avec le Mexique avec la promesse de reconquérir ses territoires perdus au Texas, en Arizona et au Nouveau Mexique.
À partir de mars 1918, l'armée allemande utilise un nouveau système de chiffrement appelé ADFGVX. Il faudra trois mois au capitaine Georges Painvin pour décrypter ce chiffre. Le premier message, ainsi décrypté, permettra au général Mangin de prendre connaissance d'une attaque allemande sur Compiègne le 9 juin 1918. Il déclenchera une contre-offensive qui sera victorieuse. Cette avance sur le chiffre allemand donnera ensuite l'initiative aux alliés et enclenchera la dynamique de la victoire. À la fin de la guerre, Clémenceau déclarera qu'à elle seule l'équipe du chiffre valait tout un corps d'armée.
La Seconde Guerre Mondiale va apporter une amélioration significative aux techniques cryptographiques : l'utilisation de calculateurs électroniques. Le premier d'entre eux, la machine Enigma, va marquer l'histoire de ce conflit [3].

3. Pendant la Seconde Guerre Mondiale les États-Unis vont utiliser un autre calculateur

Inventée en 1918 par l'ingénieur en électricité allemand Arthur Scherbius elle est adoptée par l'armée allemande dès 1926. Cette machine utilise des rotors et un circuit électrique pour réaliser un chiffrement par substitution. Très rapidement l'équipe polonaise du chiffre va s'atteler au décryptage de l'Enigma. Son chef, Maksymilian Ciezki, comprend que, pour y arriver il va devoir faire appel aux mathématiciens et recrute trois d'entre eux : Marian Rejewski, Jerzy Różycki et Henryk Zygalski. Dès 1933, les Polonais arrivent à mettre au point la bombe cryptographique permettant de décrypter Enigma. En juillet 1939, les Polonais remettent aux Britanniques et aux Français deux répliques d'Enigma. Les Anglais acheminent la leur vers le domaine de Bletchley Park où siège le *Government Code and Cypher School* qui était le bureau anglais responsable de l'interception et du décryptage des communications étrangères. C'est là que le mathématicien Alan Turing met au point le premier calculateur dédié à la cryptanalyse : la bombe électromécanique. Cette bombe calcule toutes les combinaisons possibles des rotors de l'Enigma, et procède par élimination, à partir d'un test mis au point par Turing, pour trouver le bon positionnement des rotors et décrypter les messages. Selon les historiens, les travaux de Bletchley Park ont permis d'écourter la guerre d'un an dans le Pacifique et de deux ans en Europe.
L'histoire de la cryptographie va prendre un nouveau tournant à l'issue de la Seconde Guerre Mondiale, avec un transfert de la maîtrise de la cryptographie du monde militaire au monde civil. Parallèlement, et ce pendant toute la guerre froide, les états vont exercer un contrôle strict sur la diffusion et l'utilisation de la cryptographie et vont occulter tout ce qui traite de la cryptanalyse tout en y engageant des moyens considérables.
C'est Claude Elwood Shannon qui va être le précurseur de ce tournant. Cet ingénieur en génie électrique et mathématicien américain est l'un des pères fondateurs de la théorie de l'information. Pendant la Seconde Guerre Mondiale, Shannon travaille comme cryptographe pour les services secrets américains. À l'issue de la guerre il publie le fruit de ses recherches sous la forme de deux articles : "*A Mathematical Theory of Communication*" en 1948 [190] et "*Communication Theory of Secrecy Systems*" en 1949 [189]. Dans le premier article il décrit le diagramme général d'un système de communication (Cf. figure 1.1 page 5) qui servira de base à la théorie de l'information. Dans le deuxième article, il parle de la cryptographie sous l'angle de la théorie de l'information, ce document deviendra l'un des traités fondamentaux de la cryptographie moderne. Il y démontre notamment, que le seul algorithme sûr est une extension du chiffre de Vernam appelée "*One-time pad*".
L'algorithme cryptographique de Vernam consiste à chiffrer un texte en le combinant, par addition modulaire, avec une clef secrète aléatoire de même longueur que le texte. Le "*One-time pad*" utilise le même principe que Vernam en imposant en plus que la clef secrète ne soit jamais réutilisée. Dans son article, Shannon a montré que si la clef est réellement aléatoire, qu'elle est au moins aussi longue que le texte à chiffrer et qu'elle ne sera jamais complètement ou partiellement réutilisée, alors il est impossible de décrypter le message chiffré. Pour l'anecdote, le "*téléphone rouge*" déployé, entre Washington et Moscou, en 1963 après la crise des missiles de Cuba, utilisait des téléscripteurs protégés par un système de one-time pad. Dans le même temps, la France déployait le

électronique, la M-209, dérivée d'un calculateur inventé à la fin des années trente par le belge Boris Hagelin.

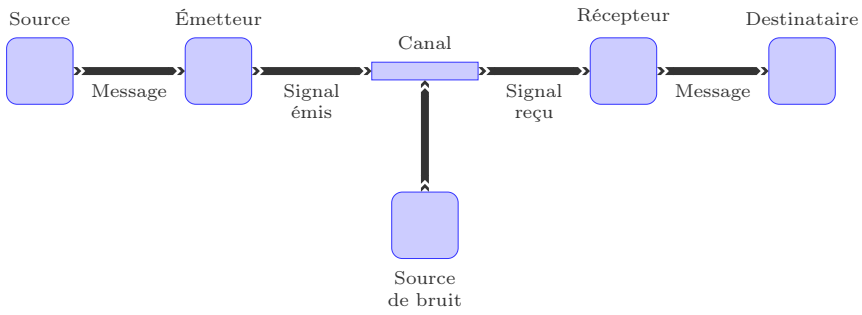


FIGURE 1.1 – Diagramme général d'un système de communication de Shannon

système TAREC, acronyme pour "*Translation Automatique Régénératrice Et Chiffrante*". Ce système, étudié par la Marine et développé par la société SAGEM, chiffrait un message en utilisant également le principe du one-time pad [4]. Il a été utilisé par l'armée française et plusieurs pays de l'OTAN jusque dans les années 1970.

L'évolution de la cryptographie va profiter de l'évolution de l'informatique et de l'explosion de l'internet. Ainsi, la deuxième moitié du XX$^{\text{ème}}$ siècle va connaître la révolution de la cryptographie asymétrique, l'apparition de standards cryptographiques comme le DES et l'AES pour les américains et l'ensemble des suites cryptographiques GOST pour les russes, l'invention de la carte à puce, la cryptographie quantique,...

Sous la pression des milieux économiques qui ont également besoin de protéger la confidentialité de leurs échanges, la cryptographie va peu à peu se libérer du contrôle voulu par les états et leurs services de renseignements. Ainsi en France, un décret-loi de 1939 avait classé les moyens de chiffrement dans la catégorie des armes de guerre. Ce n'est qu'avec la loi du 26 juillet 1996 et les décrets et arrêtés de 1998, que le chiffrement a été libéralisé pour l'utilisation de clefs d'une longueur inférieure ou égale à 40 bits. Il faudra l'article 30 de la loi pour la confiance dans l'économie numérique du 21 juin 2004 [180] pour que l'usage de moyens de cryptologie soit totalement libéré sur le territoire français.

Ce rapide historique de la cryptographie nous a permis d'entrevoir le rôle important qu'elle a joué et qu'elle joue encore dans nos sociétés. Nous l'avons vu, et l'Histoire en est une cruelle illustration, l'enjeu se situe dans la bataille que se vouent les cryptographes et les cryptanalystes. Les premiers cherchent à trouver l'ultime algorithme de chiffrement tandis que les seconds n'ont qu'un objectif : trouver le moyen de décrypter tous les messages chiffrés.

La cryptanalyse est le point de départ et la motivation de cette thèse.

À la fin des années 1990, le National Institute of Standards and Technology (NIST) américain lance un concours pour définir le standard d'un nouvel algorithme de chiffrement par blocs. C'est l'algorithme Rijndael, inventé par deux cryptographes belges Joan Daemen et Vincent Rijmen, qui remporte le concours et devient l'Advanced Encryption Standard (AES) en 2001.

Novateur dans sa conception, cet algorithme résiste depuis à toutes les attaques connues. Dix ans après sa publication, de nouvelles approches de cryptanalyse voient le jour, elles se basent sur l'écriture des mécanismes internes de l'AES

sous la forme de systèmes d'équations. Dans notre description algébrique de l'Advanced Encryption Standard [80], nous avons décrit l'une d'entres elles. Présentée par Carlos Cid, Sean Murphy et Matthew Robshaw [49], cette mise en équations algébriques de l'AES s'appuie sur une fonction de bijection entre les deux espaces vectoriels $A = GF(2^8)^{16}$ et $B = GF(2^8)^{128}$. Cette approche leur permet de décrire les mécanismes de chiffrement de l'AES sous la forme d'un système de 8576 polynômes et de 4288 variables. Nous avons montré qu'à partir des algorithmes de résolution de grands systèmes d'équations existant, le temps nécessaire pour recouvrer une clef de chiffrement avec ces systèmes d'équations était plus important que le temps nécessaire pour recouvrer cette clef par la force brute.
En nous appuyant sur l'idée d'utiliser l'algèbre pour cryptanalyser l'AES, nous avons alors cherché un autre moyen de décrire l'AES, non pas sous la forme d'un système d'équations quadratiques mais sous la forme d'un système d'équations booléennes. Ainsi, en nous appuyant sur la forme algébrique normale des fonctions booléennes, nous avons décrit dans [82], le mécanisme de génération de ce système d'équations booléennes que nous avons ensuite appliqué à l'AES ([83] et [84]).
Le premier objectif de notre démarche n'est pas d'arriver à obtenir un système d'équations plus facilement résolvable mais plutôt, à partir du résultat obtenu, et en nous appuyant sur une présentation spécifique de ces équations, de revenir à une cryptanalyse statistique plus classique. En effet, les équations booléennes aléatoires présentent des propriétés statistiques auxquelles les équations que nous avons obtenues doivent répondre à défaut de présenter un biais statistique alors plus facilement identifiable. Les résultats obtenus montrent que les fonctions mises en œuvre dans un tour de chiffrement de l'AES présentent un écart par rapport à la répartition attendue dans une fonction booléenne aléatoire. Cependant, cet écart est compensé d'une part par le mécanisme de dérivation de la clef et, d'autre part, par le nombre de tours mis en œuvre dans le processus de chiffrement de l'AES.
Notre démarche achoppe sur la difficulté à décrire la fonction de dérivation de la clef sous forme d'équations booléennes. Ce contretemps peut être compensé par le rajout d'une inconnue décrivant, pour chaque tour, la variable de la clef. L'analyse combinatoire des équations booléennes décrivant le mécanisme de dérivation de la clef révèle une disparité importante dans l'utilisation des bits de clefs. En effet, là où chaque bit de clef devrait intervenir avec la même probabilité, nous avons découvert que les 96 premiers bits de clefs sont utilisés entre deux et quatre fois chacun alors que les 32 derniers bits de clefs sont utilisés plus de 1500 fois chacun.
Il reste alors la question de la résolution de notre système d'équations booléennes. Cette dernière reste ouverte et pourrait faire l'objet de travaux complémentaires dans la continuité de cette thèse.

Première partie

Description algébrique de l'AES

Chapitre 2

l'Advanced Encryption Standard

« The design and strength of all key lengths of the AES algorithm are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use. »

National Policy on the Use of the AES to Protect National Security Systems and National Security Information [52]

2.1 Introduction à la cryptographie

2.1.1 Définitions

Imaginons le scénario suivant : Alice souhaite envoyer une lettre à Bob. Elle ne fait pas confiance au transporteur et souhaite que Bob reçoive la lettre sans que cette dernière ait pu ni être modifiée ni être ouverte pendant le transport.
Il existe deux solutions au problème d'Alice.
La première solution (voir fig. 2.1 p. 10) s'appuie sur une information gardée secrète par Alice et Bob, nous l'appellerons la solution symétrique. Dans ce cas, Alice fait l'acquisition d'un cadenas, elle envoie la clef du cadenas à Bob en utilisant un canal de transport sécurisé. Puis elle enferme la lettre dans un coffre qu'elle ferme avec le cadenas et envoie le tout à Bob. En recevant le coffre Bob, peut l'ouvrir avec la clef du cadenas et lire sa lettre.
Avec cette solution, Alice a la certitude que personne n'a pu ouvrir ou modifier la lettre en dehors de Bob. En revanche, si Alice souhaite envoyer une lettre à chaque membre de sa famille, elle devra acheter un cadenas par correspondant ce qui sera plus onéreux et plus difficile à mettre en œuvre. En cryptographie symétrique, le cadenas correspond à l'algorithme de chiffrement et la clef du cadenas à la clef de chiffrement du message.
La deuxième solution (voir fig. 2.2 p. 10) à notre énigme s'appuie aussi sur une information dont une partie est disponible à tous et l'autre reste privée. Nous

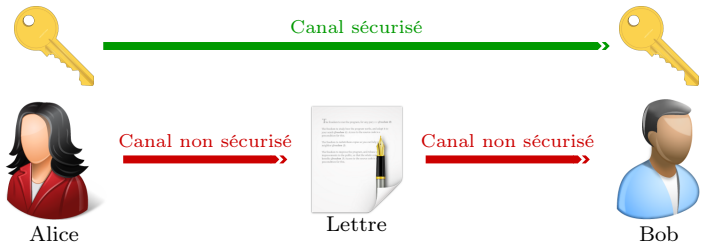


FIGURE 2.1 – La solution symétrique

l'appellerons la solution asymétrique. Dans ce cas, Bob choisit un cadenas et le dépose, ouvert, dans un lieu public tout en gardant la clef par-devers lui. Lorsque Alice souhaite envoyer une lettre à Bob, il lui suffit de récupérer le cadenas de Bob, de placer sa lettre dans une boîte, de la fermer avec le cadenas et de l'envoyer à Bob. À la réception, Bob utilise la clef en sa possession pour ouvrir la boîte et découvrir sa lettre.
Dans ce cas également, Alice a la certitude que personne n'a pu ouvrir ou modifier sa lettre pour Bob. Cette solution présente également l'avantage de gérer efficacement la montée en charge, si Alice souhaite envoyer une lettre à toute sa famille : il suffit que chacun de ses correspondants possèdent un cadenas. En cryptographie asymétrique [77], le cadenas correspond à la clef publique de Bob tandis que la clef du cadenas correspond à sa clef privée.

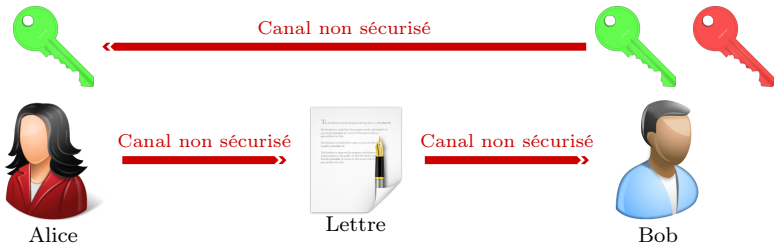


FIGURE 2.2 – La solution symétrique

L'objectif fondamental de la cryptographie (voir fig. 2.3 p. 11) est donc de permettre à deux personnes de communiquer via un canal non sécurisé sans qu'un étranger puisse accéder au contenu de la communication.
Depuis son apparition il y a plus de 4000 ans, la cryptographie n'a cessé d'évoluer. Initialement dévolue à la protection des messages échangés, le panel de ses applications s'est élargi pour couvrir aujourd'hui un seul enjeu : la sécurité des systèmes d'information [1]. Ainsi, la cryptographie moderne intervient dans tous les grands domaines de la sécurité de l'information que ce soit au travers de son enjeu premier : la confidentialité mais aussi au travers des processus d'authentification, de non-répudiation, de signature, d'horodatage, de contrôle d'accès,

1. Un système d'information désigne l'ensemble des moyens informatiques ayant pour finalité d'élaborer, de traiter, de stocker, d'acheminer, de présenter ou de détruire l'information [108].

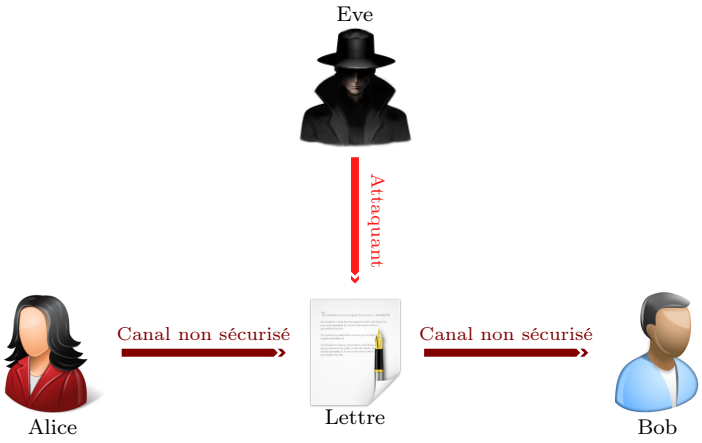


FIGURE 2.3 – l'objectif de la cryptographie

...
En parallèle, les techniques d'attaque contre les algorithmes de chiffrement se sont améliorées profitant de l'évolution des connaissances mathématiques et de l'explosion de la puissance de calcul des ordinateurs. Ainsi, à l'instar, de la cryptographie, la cryptanalyse s'est développée dans un jeu de lutte de l'épée contre le bouclier où elle a eu tantôt le dessus, tantôt le dessous.
La cryptographie et la cryptanalyse sont regroupées dans une science du secret : la cryptologie. Cette dernière, élevée au rang de discipline des mathématiques, fait appel à d'autres disciplines comme l'arithmétique modulaire, l'algèbre, la complexité, la théorie de l'information, ou encore les codes correcteurs d'erreurs.
Avant de donner une définition formelle de la cryptographie fixons quelques éléments de notation :

- **A** est l'ensemble fini appelé alphabet ;
- **P** est l'ensemble fini appelé espace des textes en clair ;
- **C** est l'ensemble fini appelé espace des textes chiffrés ;
- **K** est l'ensemble fini appelé espace des clés ;
- **E** et **D** définissent respectivement les ensembles finis des fonctions de chiffrement et de déchiffrement ;

Définition 2.1 (Fonction de chiffrement)**.** Si $\forall k \in \mathbf{K}$, il y a une fonction $E_k \in \mathbf{E} : \mathbf{P} \mapsto \mathbf{C}$ alors E_k est appelée fonction de chiffrement et k est la clé de chiffrement.

Définition 2.2 (Fonction de déchiffrement)**.** Si $\forall k' \in \mathbf{K}$, il y a une fonction $D_{k'} \in \mathbf{D} : \mathbf{C} \mapsto \mathbf{P}$ alors $D_{k'}$ est appelée fonction de déchiffrement et k' est la clé de déchiffrement.

Définition 2.3 (Système cryptographique)**.** Un système cryptographique est le quintuplet **S** tel que $\mathbf{S} = \{\mathbf{P}, \mathbf{C}, \mathbf{K}, \mathbf{E}, \mathbf{D}\}$.

Définition 2.4 (Algorithme cryptographique)**.** Un algorithme cryptographique est l'ensemble des fonctions de chiffrement $\{E_k : k \in \mathbf{K}\}$ et de l'ensemble des fonctions de déchiffrement associées $\{D_{k'} : k' \in \mathbf{K}\}$ tels que $\forall k \in \mathbf{K}$ il y a une

clef unique $k' \in \mathbf{K}$ telle que $D_{k'} = E_k^{-1}$. Autrement dit $D_{k'}(E_k(p)) = p, \quad \forall p \in \mathbf{P}$ avec (k, k') la paire de clefs.

Définition 2.5 (Algorithme cryptographique symétrique)**.** Un algorithme cryptographique symétrique est un algorithme cryptographique pour lequel $\forall k \in \mathbf{K}$ et $\forall k' \in \mathbf{K}$ la paire de clefs (k, k') est telle que $k = k'$.

Définition 2.6 (Algorithme cryptographique asymétrique)**.** Un algorithme cryptographique asymétrique est un algorithme cryptographique pour lequel $\forall k \in \mathbf{K}$ et $\forall k' \in \mathbf{K}$ la paire de clefs (k, k') est telle que $k \neq k'$.

Un algorithme cryptographique est dit cryptanalysable si une tierce partie, sans connaissance préalable de la paire de clef (k, k'), peut systématiquement recouvrer le message en clair, à partir du message chiffré, dans un temps raisonnable.

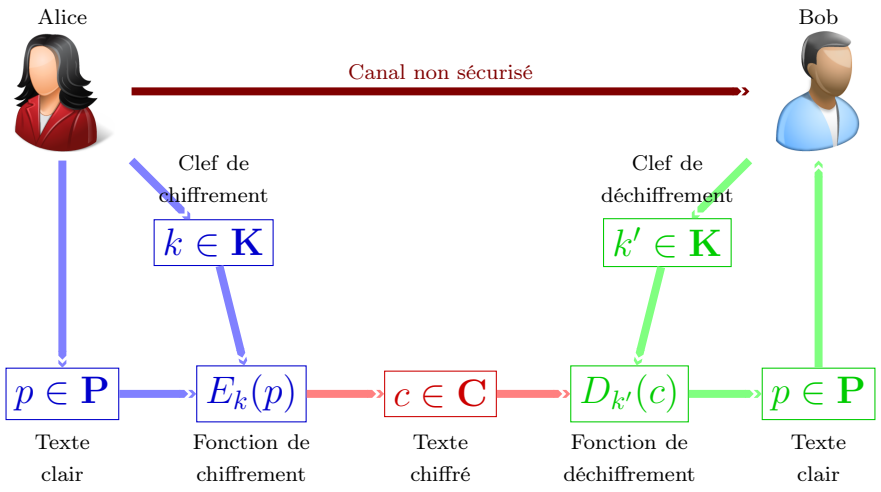


FIGURE 2.4 – Le processus de chiffrement

2.1.2 Substitution et Permutation

Il existe principalement deux transformations de base pour chiffrer un message : la substitution et la permutation.
Afin, de mieux appréhender l'action de ces deux transformations, nous allons présenter, tout au long de ce paragraphe, le résultat de l'action de chiffrement sur une image. Le principe retenu s'appuie sur l'image d'un chat. Cette image a une définition de 200 pixels de haut et 200 pixels de large ce qui fait que la longueur du message à chiffrer est de $200 \times 200 = 40000$. De plus, l'image est au format Truevision Targa [114]. Ce format présente le double avantage de stocker les données des pixels sous la forme d'un tableau à une entrée et de permettre le codage de la couleur de chaque pixel sur un octet. L'image est donc en échelle de gris avec des niveaux de gris allant de 0 pour le noir à 255 pour le blanc. Ce qui nous donne, en terme cryptographique, un message de 40 000 caractères construit à partir d'un alphabet de 256 éléments. Nous avons développé en python le programme nécessaire à la réalisation de ces images [81, VisualCrypto].

2.1.2.1 Substitution

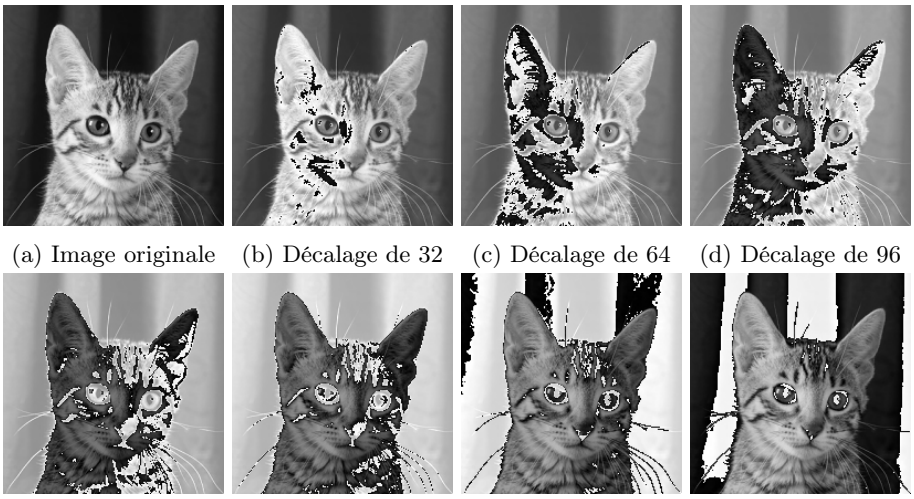
La substitution a été utilisée dès l'antiquité, notamment, avec le carré de Polybe décrit vers 150 avant Jésus-Christ. Son implémentation la plus célèbre, le chiffre de César, consiste en un décalage des lettres de l'alphabet de 3 crans. Dans ce cas, il s'agit d'un algorithme de chiffrement par substitution mono-alphabétique. La fonction de chiffrement est la suivante :

$$E_k(x) = x + k \mod 26$$

tandis que la fonction de déchiffrement est :

$$D_k(y) = y - k \mod 26$$

La valeur k correspond à la clef de chiffrement et de déchiffrement et vaut $k = 3$ pour le chiffre de César et $k = 13$ pour l'algorithme ROT13.
L'inconvénient de l'opération de substitution est qu'elle ne modifie pas l'entropie du message initial (voir fig. 2.5 p. 13). Une analyse statistique simple, en utilisant la fréquence d'apparition des caractères, permet de retrouver systématiquement le message d'origine à partir du message chiffré.



(a) Image originale (b) Décalage de 32 (c) Décalage de 64 (d) Décalage de 96

(e) Décalage de 128 (f) Décalage de 160 (g) Décalage de 192 (h) Décalage de 224

FIGURE 2.5 – Exemples de chiffrement par substitution mono-alphabétique

Au XVIe siècle, le diplomate français Blaise Vigenère, reprenant une idée de Giovan Batista Belaso, imagine une extension du chiffre de César en améliorant le concept de clef. La clef utilisée par Vigenère est un mot k de longueur n. Chaque lettre du mot définit une variable de décalage pour chaque lettre du message. Le chiffre de Vigenère est un algorithme de chiffrement par substitution poly-alphabétique. Sa fonction de chiffrement est la suivante :

$$E_k(x_1, \ldots, x_n) = (x_1 + k_1 \mod 26), \ldots, (x_n + k_n \mod 26)$$

et sa fonction de déchiffrement est :

$$D_k(y_1, \ldots, y_n) = (y_1 - k_1 \mod 26), \ldots, (y_n - k_n \mod 26)$$

Le fonctionnement du chiffre de Vigenère est le suivant. Pour chiffrer la phrase "LABOR OMNIA VINCIT IMPROBUS" en utilisant la clef "VIGENERE", il suffit, à partir de la table de Vigenère (voir fig. 2.6 p. 14), de trouver la lettre située au croisement de la colonne correspondant à la première lettre du texte clair avec la ligne correspondant à la première lettre de la clef et de renouveler ce processus pour chaque lettre. Dans notre cas, le chiffré obtenu est "GIHSE SDRDI BMAGZX DUVVBFLW".

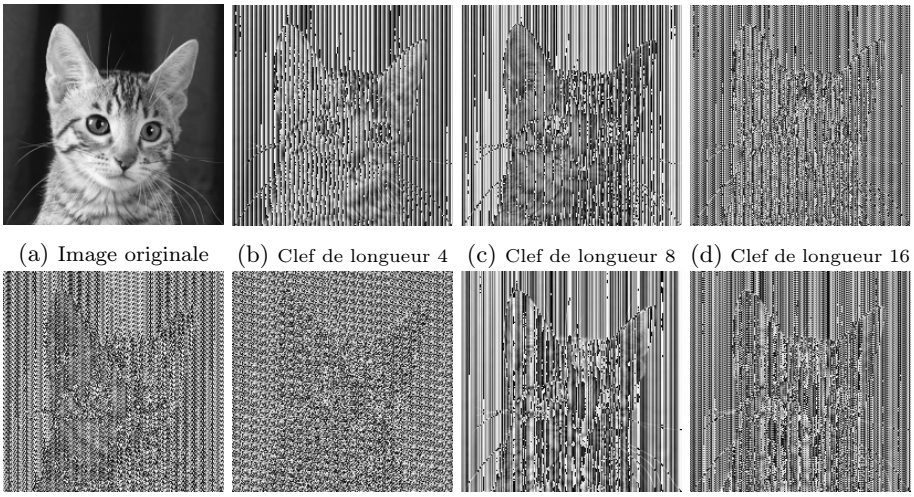
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T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

FIGURE 2.6 – La table de Vigenère

La cryptanalyse du chiffre de Vigenère se résume, à partir de la connaissance de la taille de la clef, à une analyse statistique basée sur la fréquence d'apparition des caractères (voir fig. 2.7 p. 15).

2.1.2.2 Permutation

Une implémentation célèbre de l'opération de permutation est la scytale, ou bâton de Plutarque, mise en œuvre par les Spartiates. Son principe consiste, après avoir enroulé une bandelette de cuir sur un bâton de diamètre fixé, à écrire le message transversalement sur la bandelette en plaçant une lettre par circonvolution. Une fois celle-ci déroulée le message chiffré peut être transporté par un messager. Pour déchiffrer le message, il est nécessaire de disposer d'un



(a) Image originale (b) Clef de longueur 4 (c) Clef de longueur 8 (d) Clef de longueur 16

(e) Clef de longueur 32 (f) Clef de longueur 64 (g) Clef de longueur 100 (h) Clef de longueur 400

FIGURE 2.7 – Exemples de chiffrement par substitution poly-alphabétique

bâton de même diamètre que celui utilisé par l'émetteur. Le diamètre du bâton utilisé correspond à la clef de chiffrement.
Le principe de fonctionnement de la scytale est un algorithme de chiffrement par permutation mono-alphabétique : le message chiffré est un anagramme du message original. La fonction de chiffrement est la suivante :

$$E_k(x_1, \ldots, x_m) = (x_{k(1)}, x_{k(2)}, \ldots, x_{k(m)})$$

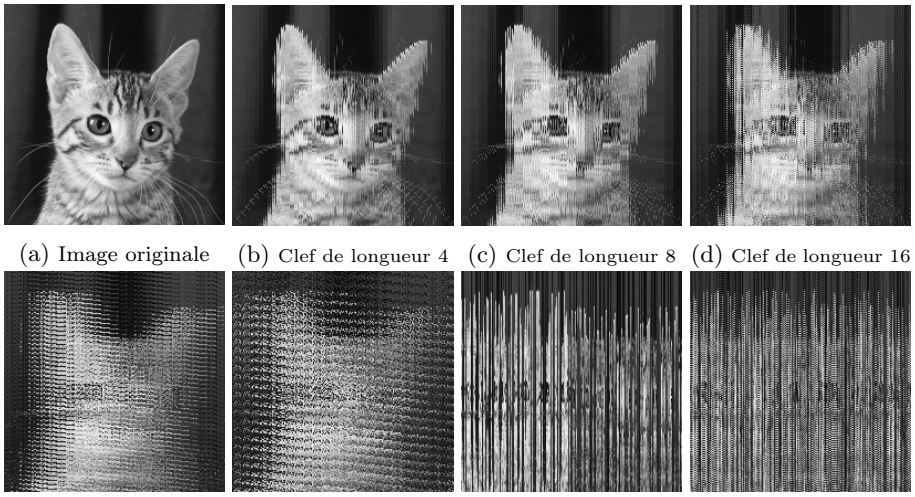
où $k(x)$ réalise une permutation. La fonction de déchiffrement est :

$$D_k(y_1, \ldots, y_m) = (y_{k(1)}^{-1}, y_{k(2)}^{-1}, \ldots, y_{k(m)}^{-1})$$

L'opération de permutation mono-alphabétique introduit la notion de chiffrement par blocs. En effet, le texte clair est d'abord découpé en blocs de tailles identiques puis, la permutation est appliquée sur chacun des blocs. L'opération de permutation conserve la distribution des caractères du texte clair (voir fig. 2.8 p. 16). Une analyse statistique de la fréquence des caractères permet de retrouver le message clair initial. Cependant, cette analyse est complexifiée par la nécessité de découvrir, au préalable, la longueur de la clef.

2.1.2.3 SP-Network

En 1949, Claude Shannon [189] définit deux opérations indispensables pour un algorithme de chiffrement sécurisé : la confusion et la diffusion.
La confusion est la propriété qui permet de rendre plus complexe la relation statistique entre le texte chiffré et la clef de chiffrement. La confusion est réalisée par une primitive de substitution. En cryptographie moderne, la substitution remplace des bits par d'autres bits selon des règles établies dans l'algorithme.
La deuxième propriété introduite par Shannon est la diffusion. Cette dernière permet de rendre plus complexe la relation statistique entre le texte clair et



(a) Image originale (b) Clef de longueur 4 (c) Clef de longueur 8 (d) Clef de longueur 16

(e) Clef de longueur 32 (f) Clef de longueur 64 (g) Clef de longueur 100 (h) Clef de longueur 400

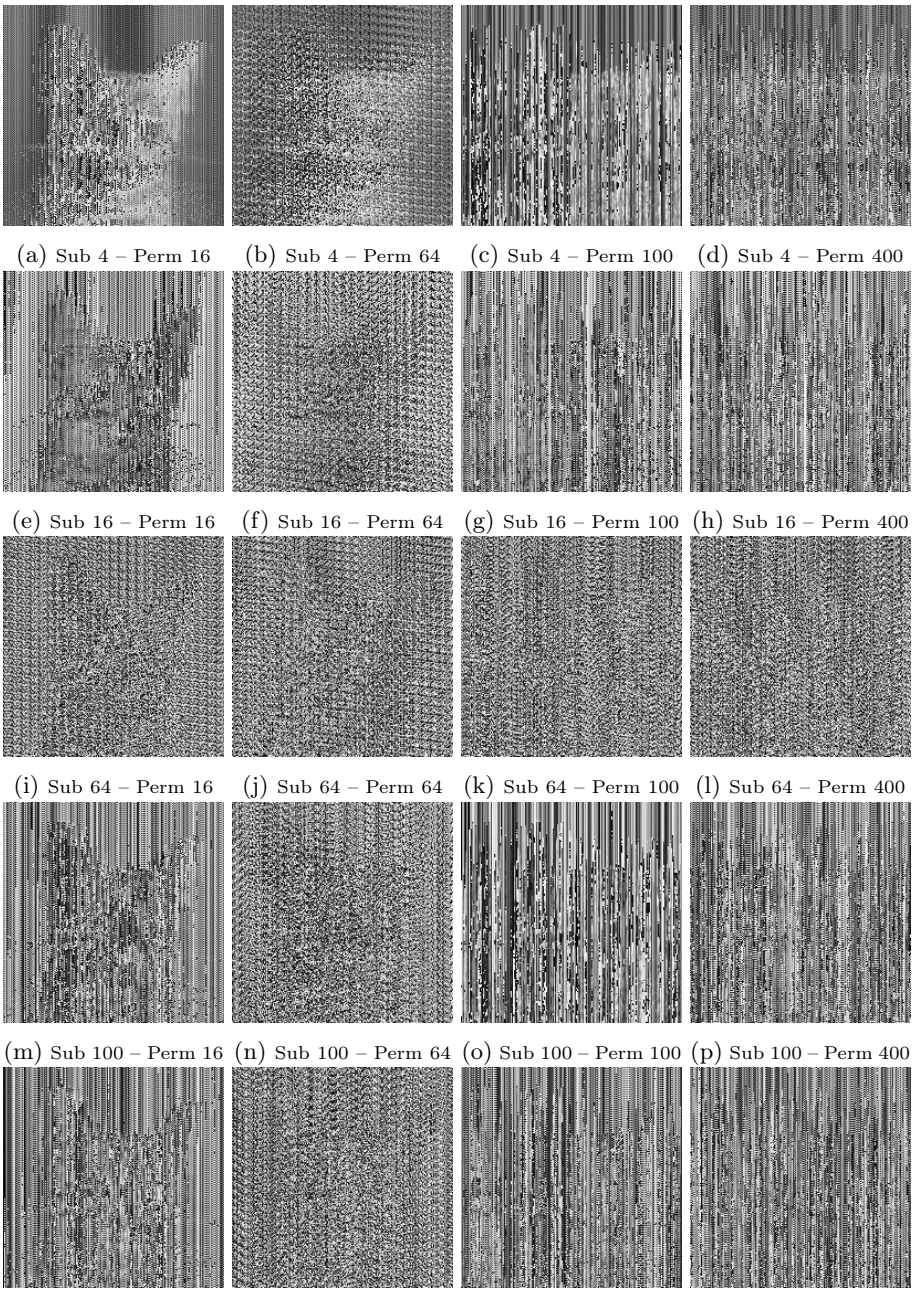
FIGURE 2.8 – Exemples de chiffrement par permutation

le texte chiffré. La diffusion est réalisée par une primitive de permutation. En cryptographie moderne, la permutation change l'ordre des bits.
Afin de rendre plus performantes la confusion et la diffusion dans l'algorithme cryptographique, Shannon propose de répéter et de chaîner les opérations de substitution et de permutation (voir fig. 2.9 p. 17).
Il existe deux grandes structures pour réaliser le chaînage des fonctions de substitution et de permutation : les réseaux de Feistel et les réseaux de substitution-permutation ou SP Networks (voir fig. 2.10 p. 18). Les réseaux de Feistel sont utilisés dans les algorithmes de chiffrement comme le DES, Blowfish, GOST 28147-89, Twofish, tandis que les SP-networks sont utilisés par l'AES, SAFER, SHARK and Square.
Dans ces réseaux, composés de boîtes de substitution ou S-Box et de boîtes de permutation ou P-Box, le chiffrement se déroule en faisant intervenir la clef de chiffrement et un produit de chiffrement reposant sur une combinaison de S-Box et de P-Box. Enfin, toujours dans un but d'optimisation des propriétés de confusion et de diffusion, chaque séquence de combinaison clef de chiffrement, S-Box et P-Box est répété plusieurs fois formant ainsi une succession de tours ou *rounds* en anglais (voir fig. 2.11 p. 19).

2.2 Présentation de l'AES

Depuis le 26 novembre 2001, l'algorithme de chiffrement par bloc "Rijndael", dans sa version 128 bits, est devenu le successeur du DES sous le nom d'*Advanced Encryption Standard* (AES).
Issu d'un concours lancé par le *National Institute of Standards and Technology* (NIST) en 1997, Rijndael [69] a franchi toutes les étapes de sélection et est maintenant un standard fédéral américain enregistré sous le numéro FIPS 197 [155]. Inscrit sur la suite B de la National Security Agency (NSA) [2], l'AES a vocation,

2. http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml



(a) Sub 4 – Perm 16 (b) Sub 4 – Perm 64 (c) Sub 4 – Perm 100 (d) Sub 4 – Perm 400

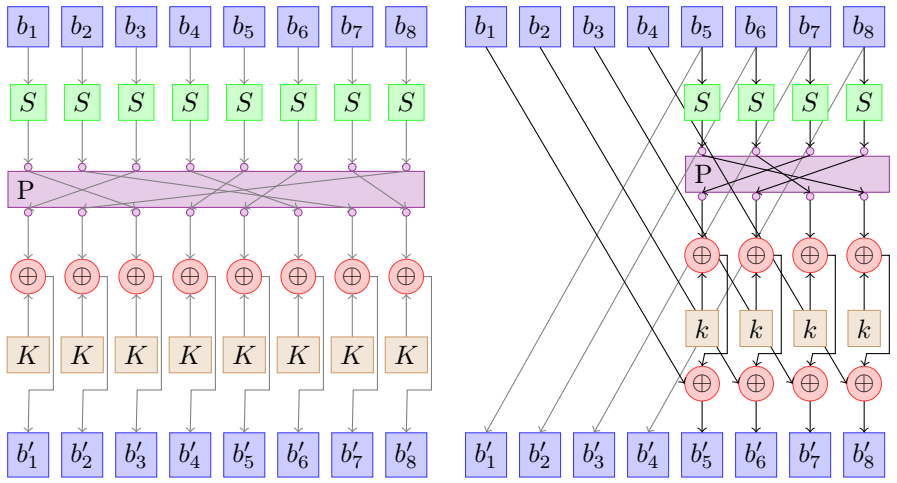
(e) Sub 16 – Perm 16 (f) Sub 16 – Perm 64 (g) Sub 16 – Perm 100 (h) Sub 16 – Perm 400

(i) Sub 64 – Perm 16 (j) Sub 64 – Perm 64 (k) Sub 64 – Perm 100 (l) Sub 64 – Perm 400

(m) Sub 100 – Perm 16 (n) Sub 100 – Perm 64 (o) Sub 100 – Perm 100 (p) Sub 100 – Perm 400

(q) Sub 400 – Perm 16 (r) Sub 400 – Perm 64 (s) Sub 400 – Perm 100 (t) Sub 400 – Perm 400

FIGURE 2.9 – Exemples de chiffrement par chaînage de substitution et de permutation. Un seul tour est mis en œuvre. Il comprend une substitution et une permutation. La taille des clefs utilisées est indiquée dans la légende. Ainsi, *Sub 400 – Perm 16* indique une clef de longueur 400 pour la fonction de substitution et de longueur 16 pour la fonction de permutation.



(a) SP-network (b) Réseau de Feistel

FIGURE 2.10 – Les deux grandes familles de chaînage de substitution et permutation. Application sur un octet en entrée.

promu par le gouvernement américain, à devenir un standard pour l'échange sécurisé des informations classifiées, aux États-Unis et entre les États-Unis et leurs partenaires [53]. En effet, initialement réservé au chiffrement d'informations sensibles, non classifiées (article 6 de [155]), le champ d'application de l'AES a évolué et devient, à compter du premier octobre 2015, l'algorithme de chiffrement pour les informations classifiées jusqu'au niveau TOP SECRET aux États-Unis (Annexe B de [53]). De même, il est, aujourd'hui, l'algorithme symétrique de chiffrement par bloc le plus couramment utilisé en occident [3].
L'AES est un algorithme de chiffrement symétrique par bloc. Il chiffre et déchiffre des blocs de données à partir d'une seule clef.
Contrairement au DES, basé sur un réseau de Feistel, l'AES s'appuie sur un réseau de substitutions et de permutations (SP-network). Ce dernier est constitué de fonctions de substitution non linéaires contenues dans une S-Box et de fonctions de permutation linéaire que l'on peut regrouper dans une P-Box. Chaque boîte prend un bloc de texte et la clé en entrée et fournit un bloc de texte chiffré en sortie. Le cheminement de l'information dans une suite définie de plusieurs P-Box et S-Box formant un tour.
Historiquement, l'AES a deux prédécesseurs. Le premier est l'algorithme de chiffrement par bloc Shark [171] publié en 1996 par Vincent Rijmen, Joan Daemen, Bart Preneel, Anton Bosselaers et Erik de Win. Shark utilise des blocs de 64 bits et une clé de 128 bits. À l'instar de l'AES, il utilise un SP-network avec six tours. Le deuxième algorithme s'appelle Square, il a été publié en 1997 par Joan Daemen et Vincent Rijmen. Il utilise un SP-network avec huit tours et travaille sur des blocs de 128 bits et une clé de 128 bits également.

3. La Russie, par exemple, utilise les algorithmes de chiffrement définis par les standards GOST 28147-89, GOST R 34.10-2001. De même la Chine utilise les algorithmes de chiffrement définis par les standards SM2, SM4 et ZUC.

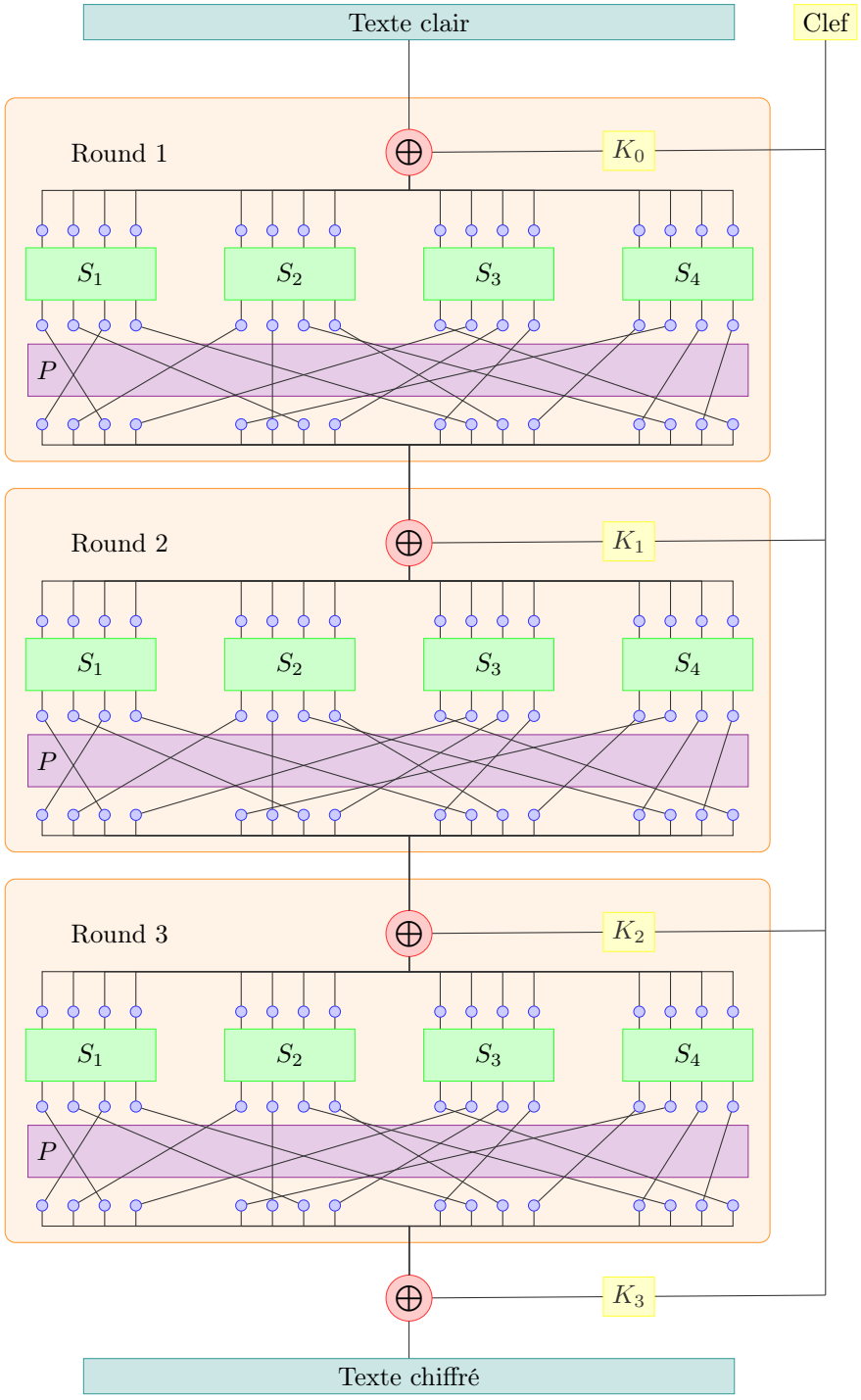


FIGURE 2.11 – Exemple de SP Network avec 3 tours. Message en entrée de 16 bits et S-Box travaillant sur 4 bits.

Les entrées et sorties de l'AES sont des blocs de 128 bits et la longueur de la clé peut être de 128, 192 ou 256 bits. Chaque bit d'un bloc d'entrée ou de clé est numéroté de 0 à $i-1$ avec i pouvant prendre 128, 192 ou 256 comme valeur. Ainsi, i, aussi appelé index du bit de bloc, est compris dans les intervalles suivants : $0 \leqslant i < 128$, $0 \leqslant i < 192$ ou $0 \leqslant i < 256$.
L'unité de base de l'algorithme est l'octet, c'est-à-dire une séquence de huit bits traités comme une seule entité. Chaque octet est représenté comme la concaténation de la valeur de chacun des bits (0 ou 1) le composant.
Pour son fonctionnement interne, chaque bloc de données fourni en entrée de l'algorithme est organisé en tableau de quatre colonnes et quatre lignes, chaque case contenant un octet, soit $4 \times 4 \times 8 = 128$ bits par tableau.
Au début des opérations de chiffrement et de déchiffrement, le bloc d'octets en entrée est copié dans le tableau d'états [4]. Les opérations de chiffrement et de déchiffrement sont effectuées sur ce tableau, puis, le résultat est copié dans un tableau de sortie (voir fig. 2.12 p. 20).

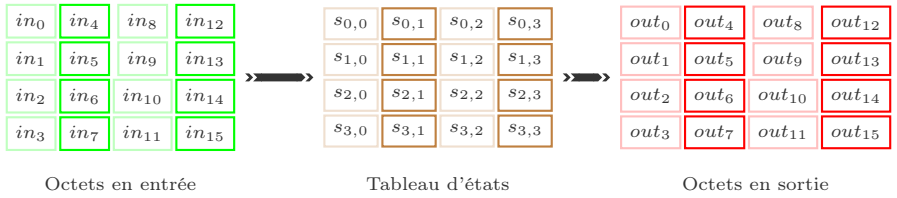


FIGURE 2.12 – Organisation des tableaux d'entrée, d'états et de sortie

2.3 Fonctionnement détaillé de l'AES

Les entrées et sorties de l'AES sont des blocs de 128 bits et la longueur de la clé peut être de 128, 192 ou 256 bits. L'unité de base de l'algorithme est l'octet. Les blocs de données fournis en entrée sont transformés en tableaux de quatre colonnes et quatre lignes, chaque case contenant un octet, soit $4 * 4 * 8 = 128$ bits par tableau.
Pour les opérations de chiffrement et de déchiffrement, l'algorithme de l'AES utilise une fonction de tour composée de quatre fonctions différentes :

1. la première exécute une substitution d'octets en utilisant une table de substitution ou S-box ;
2. la deuxième exécute un glissement des rangées du tableau d'états à partir d'offsets différents ;
3. la troisième réalise un mélange des colonnes du tableau d'états ;
4. la quatrième ajoute la clef de tour au tableau d'états

La deuxième et la troisième fonction forment la P-box du tour.

4. Le tableau d'états est un tableau d'octets à deux dimensions comprenant n lignes et m colonnes. Pour l'AES, $n = m = 4$.

2.3.1 Opérations de chiffrement

Les opérations de chiffrement (voir fig. 2.13 p. 21) s'appuient sur quatre fonctions prédéfinies : `AddRoundKey`, `SubBytes`, `ShiftRows` et `MixColumns`. Chacune de ces fonctions est exécutée sur le tableau d'états. Le cycle de chiffrement comprend une transformation initiale, des tours intermédiaires et un tour final.

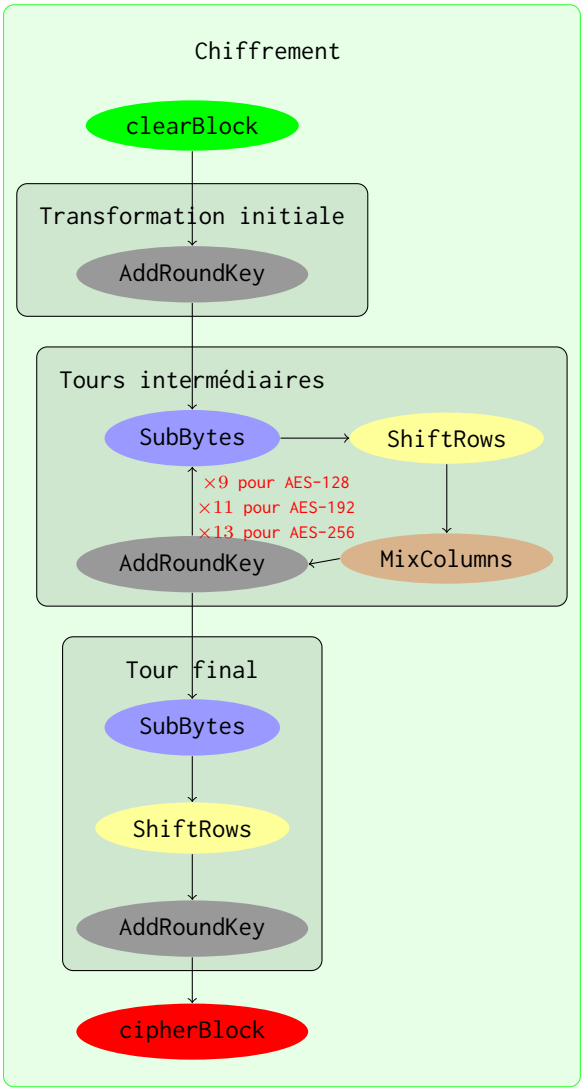


FIGURE 2.13 – Le processus de chiffrement de l'AES

La transformation initiale consiste à appliquer la fonction `AddRoundKey` au tableau d'états. Les tours intermédiaires exécutent, dans l'ordre, les fonctions `SubBytes`, `ShiftRows`, `MixColumns` et `AddRoundKey` sur le tableau d'états. Le tour final diffère des tours intermédiaires, par la suppression de la fonction `MixColumns` dans le cycle des transformations.
Le nombre de tour dans l'AES est dépendant de la taille de la clé. Ainsi, pour

une clé de 128 bits, le nombre de tours est 10. De même, nous avons 12 tours pour une clé de 192 bits et 14 tours pour une clé de 256 bits.
Finalement, le pseudo code de la fonction de chiffrement de l'AES peut s'écrire de la façon suivante (voir fig. 2.14 p. 22). Dans ce cas, `Nb` correspond au nombre de mots de 32 bits et par conséquent au nombre de colonnes du tableau d'états et `Nr` correspond au nombre de tours utilisés dans l'algorithme.

```
1:  function CIPHER(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
2:      byte state[4,Nb]
3:      state ← in
4:      AddRounkey(state, w[0, Nb-1])
5:      for round=1 step 1 to Nr-1 do
6:          SubBytes(state)
7:          ShiftRows(state)
8:          MixColumns(state)
9:          AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
10:     end for
11:     SubBytes(state)
12:     ShiftRows(state)
13:     AddRounkey(state, w[Nr*Nb, (Nr+1)*Nb-1])
14:     return state
15: end function
```

FIGURE 2.14 – Pseudo-code pour le chiffrement

Afin de mieux appréhender le fonctionnement interne de l'algorithme AES, nous allons l'implémenter en utilisant le langage Python. La fonction de chiffrement décrite précédemment s'écrit alors selon le code du listing 1 page 22.

```python
def cipher(block, key, Nb=4, Nr=10):
    state = createState(block, Nb)
    roundKey = "".join(key[0:Nb])
    state = addRoundKey(state, createState(roundKey, Nb), Nb)

    for round in xrange(1,Nr):
        state = subBytes(state, Nb)
        state = shiftRows(state, Nb)
        state = mixColumns(state, Nb)
        roundKey = "".join(key[round*Nb:(round*Nb)+Nb])
        state = addRoundKey(state, createState(roundKey, Nb), Nb)

    state = subBytes(state, Nb)
    state = shiftRows(state, Nb)
    roundKey = "".join(key[Nr*Nb:(Nr*Nb)+Nb])
    state = addRoundKey(state, createState(roundKey, Nb), Nb)

    return createBlock(state, Nb)
```

Listing 1 – Fonction de chiffrement de l'AES en python

La fonction `cipher` prend un block de texte clair en entrée ainsi que la clef de

chiffrement. Le texte clair est saisi sous sa forme hexadécimale, par exemple : `3243f6a8885a308d313198a2e0370734`, tandis que la clef de chiffrement est un tableau contenant les différentes clefs de tours. La fonction `cipher` commence par créer une variable `state` contenant le tableau d'états (ligne 2) et une variable `roundKey` contenant la clef du tour (ligne 3). Ensuite, elle déroule l'algorithme de l'AES en faisant appel aux quatre fonctions de tour : `addRoundKey`, `subBytes`, `shiftRows` et `mixColumns`. Finalement, la fonction retourne les 128 bits de texte chiffré en hexadécimal.
La fonction `createState` sert à convertir un bloc de 128 bits, en héxadécimal, en un tableau reflétant la structure du tableau d'états. La fonction `createBlock` est la fonction inverse de la précédente : à partir d'un tableau d'états, elle construit un bloc de 128 bits en héxadécimal.

2.3.1.1 SubBytes

La transformation `SubBytes` effectue une substitution d'octets non linéaire en utilisant une fonction de substitution représentée sous forme de table (*S-Box*). Ainsi, chaque élément $\mathsf{S}_{r,c}$, $0 \leq r, c \leq 3$ du tableau d'états est substitué selon la fonction suivante :

$$\mathsf{S}_{r,c} \mapsto \text{S-Box}[\mathsf{S}_{r,c}]$$

La `S-Box`, est un tableau construit par la composition de deux fonctions :

1. pour chaque octet, prendre son inverse dans $GF(2^8)$, l'octet `0x00` est, par convention, son propre inverse ;
2. lui appliquer la transformation suivante :

$$b'_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

avec $0 \leq i < 8$, b_i le i^{eme} bit de l'octet et c_i le i^{eme} bit de l'octet `0x63` ou encore $(01100011)_2$;

La forme matricielle de cette transformation est la suivante :

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

L'utilisation de la `S-Box`, telle qu'elle est présentée dans la figure 2.15 page 24, est la suivante : en choisissant, par exemple, l'octet `0x53`, la valeur de substitution est obtenue en prenant la valeur de l'octet à l'intersection de la rangée d'indice 5 et de la colonne d'indice 3. Ainsi, `SubBytes(0x53) = 0xED`.
Le code de l'implémentation en python de la fonction `SubBytes` est donné dans le listing 2 page 24.
La fonction python `SubBytes` prend en valeur d'entrée une structure reflétant le tableau d'états et retourne le même tableau d'états substitué par la `S-Box`. La fonction commence par déclarer une structure de tableau d'états temporaire

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

FIGURE 2.15 – Table de transformation de la `S-Box`

```
1 def subBytes(state, Nb):
2     tmp = [[0 for i in xrange(Nb)] for i in xrange(Nb)]
3     for i in xrange(Nb):
4         for j in xrange(Nb):
5             tmp[i][j] = S[int(state[i][j][0], 16)]
6                         [int(state[i][j][1], 16)]
7     return tmp
```

Listing 2 – Fonction de subBytes de l'AES en python

(ligne 2), puis, dans un processus itératif, effectue la substitution de chaque octet du tableau d'états par l'octet de la `S-Box` correspondant (lignes 3 à 6).

2.3.1.2 ShiftRows

La transformation `ShiftRows` effectue une permutation circulaire des octets dans les trois dernières lignes du tableau d'états. Ainsi, chaque élément $\mathsf{S}_{r,c}$ du tableau d'états est remplacé selon la fonction (avec Nb = 4 pour l'AES) :

$$\mathsf{S}_{r,c} \mapsto \mathsf{S}_{r,((c+r) \bmod \text{Nb})}$$

Le code de l'implémentation en python de la fonction `ShiftRows` est donné dans le listing 3 page 25.
La fonction python `ShiftRows` prend en valeur d'entrée une structure reflétant le tableau d'états et retourne le tableau d'états résultant après la transformation. La fonction commence par déclarer une structure de tableau d'états temporaire (ligne 2), puis, dans un processus itératif, effectue la permutation circulaire des octets (lignes 3 à 5).

2.3.1.3 MixColumns

La transformation `MixColumns` agit sur les colonnes du tableau d'états. Chaque colonne est considérée comme un polynôme sur le corps fini à 256 éléments

```python
def shiftRows(state, Nb):
    tmp = [[0 for i in xrange(Nb)] for i in xrange(Nb)]
    for i in xrange(Nb):
        for j in xrange(Nb):
            tmp[i][j] = state[i][(j + i) % Nb]
    return tmp
```

Listing 3 – Fonction ShiftRows de l'AES en python

$GF(2^8)$ et est multipliée, modulo $x^4 + 1$, par le polynôme :

$$a(x) = \{0x03\}x^3 + \{0x01\}x^2 + \{0x01\}x + \{0x02\}$$

Cette opération revient à multiplier chaque colonne du tableau d'états par une matrice carrée d'ordre 4 selon l'équation suivante :

$$\begin{pmatrix} \mathsf{S}_{0,c} \\ \mathsf{S}_{1,c} \\ \mathsf{S}_{2,c} \\ \mathsf{S}_{3,c} \end{pmatrix} \mapsto \begin{pmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{pmatrix} \begin{pmatrix} \mathsf{S}_{0,c} \\ \mathsf{S}_{1,c} \\ \mathsf{S}_{2,c} \\ \mathsf{S}_{3,c} \end{pmatrix}$$

Ainsi, les quatre octets d'une colonne du tableau d'états sont remplacés par les octets suivants :

$$\begin{aligned} s'_{0,c} &= (0x02 \cdot s_{0,c}) \oplus (0x03 \cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (0x02 \cdot s_{1,c}) \oplus (0x03 \cdot s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (0x02 \cdot s_{2,c}) \oplus (0x03 \cdot s_{3,c}) \\ s'_{3,c} &= (0x03 \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (0x02 \cdot s_{3,c}) \end{aligned}$$

Le code de l'implémentation en python de la fonction `MixColumns` est donné dans le listing 4 page 26.
La fonction python `mixColumns` prend en valeur d'entrée une structure reflétant le tableau d'états et retourne le tableau d'états résultant après la transformation. La fonction commence par déclarer une structure de tableau d'états temporaire (ligne 2), puis, dans un processus itératif, effectue la multiplication matricielle sur chacune des colonnes (lignes 3 à 19).
Pour réaliser la multiplication dans $GF(2^8)$, nous utilisons une variante de l'algorithme de multiplication dite "du paysan russe". Le code de l'implémentation en python de la fonction de multiplication dans $GF(2^8)$ est donné dans le listing 5 page 26.

2.3.1.4 AddRoundKey

Dans la transformation `AddRoundKey`, l'AES ajoute, par un XOR, la clef du tour au tableau d'états. Cette opération s'effectue colonne par colonne selon la fonction équivalente :

$$\mathsf{S}_{r,c} \mapsto \mathsf{S}_{r,c} + \mathsf{K}_{i,4c+r}$$

```python
def mixColumns(state, Nb):
    tmp = [[0 for i in xrange(Nb)] for i in xrange(Nb)]
    for i in xrange(Nb):
        tmp[0][i] = int2hex(gMult(state[0][i], '02') ^
                gMult(state[1][i], '03') ^
                gMult(state[2][i], '01') ^
                gMult(state[3][i], '01'))
        tmp[1][i] = int2hex(gMult(state[0][i], '01') ^
                gMult(state[1][i], '02') ^
                gMult(state[2][i], '03') ^
                gMult(state[3][i], '01'))
        tmp[2][i] = int2hex(gMult(state[0][i], '01') ^
                gMult(state[1][i], '01') ^
                gMult(state[2][i], '02') ^
                gMult(state[3][i], '03'))
        tmp[3][i] = int2hex(gMult(state[0][i], '03') ^
                gMult(state[1][i], '01') ^
                gMult(state[2][i], '01') ^
                gMult(state[3][i], '02'))
    return tmp
```

Listing 4 – Fonction MixColumns de l'AES en python

```python
def gMult(a, b):
    product = '00000000'
    a = hex2bin(a)
    b = hex2bin(b)
    for i in xrange(8):
        if int(b[7], 2) & 1:
            product = bin2byte(bin(int(product, 2) ^
                    int(a, 2)).lstrip('0b'))
        aHighBit = int(a[0], 2)
        a = bin2byte(bin(int(a, 2) << 1).lstrip('0b'))
        if aHighBit & 1:
            num = hex2bin('1b')
            a = bin2byte(bin(int(a, 2) ^ int(num, 2)).lstrip('0b'))
        b = bin2byte(bin(int(b, 2) >> 1).lstrip('0b'))
    return int(product, 2)
```

Listing 5 – Fonction de multiplication dans $GF(2^8)$ en python

avec $\mathsf{S}_{r,c}$ un élément du tableau d'états, $0 \leq i \leq \mathrm{Nr}$ le numéro du tour et K_i la clé du tour obtenue par la fonction d'expansion de l'AES.
Le code de l'implémentation en python de la fonction `AddRoundKey` est donné dans le listing 6 page 27.

```
1 def addRoundKey(state, roundkey, Nb):
2     tmp = [[] for i in xrange(Nb)]
3     for i in xrange(Nb):
4         for j in xrange(Nb):
5             tmp[i].append(int2hex(int(state[i][j], 16) ^
6                 int(roundkey[i][j], 16)))
7     return tmp
```

Listing 6 – Fonction addRoundKey de l'AES en python

La fonction python `addRoundKey` prend en valeur d'entrée une structure reflétant le tableau d'états et un tableau contenant les clefs de tours issues de la fonction d'expansion de la clef. Elle retourne le tableau d'états résultant après l'ajout de la clef de tour. La fonction commence par déclarer une structure de tableau d'états temporaire (ligne 2), puis, dans un processus itératif, effectue l'ajout de la clef de tour à chacun des octets (lignes 3 à 6).

2.3.2 Expansion de la clef

Pour rappel, dans l'algorithme de l'AES, $\mathrm{Nb} = 4$ mots et $\mathrm{Nr} = (10, 12, 14)$ mots avec 1 mot = 4 octets = 32 bits.
L'algorithme de l'AES, à partir d'une clef de chiffrement K de 128, 192 ou 256 bits, exécute une procédure d'expansion de la clef pour générer les clefs de tours. La procédure d'expansion de la clef génère un total de $\mathrm{Nb}(\mathrm{Nr}+1)$ mots. En effet, l'algorithme a besoin d'un ensemble initial de Nb mots de clef et chacun des Nr tours nécessite également Nb mots de clef pour chiffrer les données.
Le résultat de la procédure d'expansion de la clef est un tableau contenant des mots de 4 octets appelés $[w_i]$ avec $0 \leq i < \mathrm{Nb}(\mathrm{Nr}+1)$.
L'algorithme d'expansion de la clef est détaillé dans le pseudo-code de la figure 2.16 page 28. Cet algorithme fait intervenir deux fonctions `SubWord` et `RotWord` ainsi qu'une constante de tour `Rcon`.
Nous pouvons déjà constater que les `Nk` premiers mots du processus d'expansion de la clef correspondent à la clef de chiffrement elle-même.
La fonction `SubWord` prend un mot de quatre octets en entrée, applique la S-Box sur chacun des octets et retourne le mot ainsi obtenu.
En langage python, la fonction `SubWord` s'écrit selon le code du listing 7 page 28.
La fonction `RotWord` prend un mot de quatre octets en entrée $w = [a_0, a_1, a_2, a_3]$, effectue une permutation sur les octets et retourne le mot ainsi obtenu $w' = [a_1, a_2, a_3, a_0]$.
En langage python, la fonction `RotWord` s'écrit selon le code du listing 8 page 28.
La constante de tour `Rcon` est un tableau de mots de quatre octets contenant les valeurs suivantes : $[x^{i-1}, 0\mathrm{x}00, 0\mathrm{x}00, 0\mathrm{x}00]$ avec $x = 0\mathrm{x}02$.
En langage python, la fonction délivrant la constante de tour `Rcon` s'écrit selon le code du listing 9 page 28.

```
1: function KEYEXPANSION(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
2:     word temp
3:     i ← 0
4:     while (i < Nk) do
5:         w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
6:         i = i + 1
7:     end while
8:     i ← Nk
9:     while (i < Nb*(Nr+1)) do
10:        temp ← w[i-1]
11:        if (i mod Nk = 0) then
12:            temp ← SubWord(RotWord(temp)) ⊕ Rcon[i/Nk]
13:        else if (Nk > 6 and i mod Nk = 4) then
14:            temp ← SubWord(temp)
15:        end if
16:        w[i] = w[i-Nk] ⊕ temp
17:        i = i + 1
18:    end while
19: end function
```

FIGURE 2.16 – Pseudo-code pour la fonction d'expansion de la clef

```python
def subWord(w):
    (a0, a1, a2, a3) = wortToBytes(w)
    return S[int(a0[0], 16)][int(a0[1], 16)] +
        S[int(a1[0], 16)][int(a1[1], 16)] +
        S[int(a2[0], 16)][int(a2[1], 16)] +
        S[int(a3[0], 16)][int(a3[1], 16)]
```

Listing 7 – Fonction SubWord en python

```python
def rotWord(w):
    (a0, a1, a2, a3) = wortToBytes(w)
    return a1 + a2 + a3 + a0
```

Listing 8 – Fonction rotWord en python

```python
def rcon(n):
    RCON = ['01000000', '02000000', '04000000', '08000000',
            '10000000', '20000000', '40000000', '80000000',
            '1b000000', '36000000']
    return RCON[n]
```

Listing 9 – Fonction RCON en python

Finalement, la fonction d'expansion de la clef telle que nous l'avons décrite dans la figure 2.16 page 28 s'écrit en langage python selon le code du listing 10 page 29.

```
1  def keyExpansion(key, Nk=4, Nb=4, Nr=10):
2      w = [0]*Nb*(Nr + 1)
3      kw = keyToWords(key, Nk)
4
5      for i in xrange(Nk):
6          w[i] = kw[i]
7
8      for i in xrange(Nk, Nb*(Nr + 1)):
9          tmp = w[i-1]
10         if (i % Nk == 0):
11             tmp = int2hex(int(subWord(rotWord(tmp)), 16) ^
12                     int(rcon((i/Nk)-1), 16))
13         elif ((Nk > 6) and (i % Nk == 4)):
14             tmp = subWord(tmp)
15         w[i] = int2hex(int(w[i - Nk], 16) ^ int(tmp, 16))
16     return w
```

Listing 10 – Fonction keyExpansion de l'AES en python

La fonction python `keyExpansion` prend en entrée la clef de chiffrement, sous sa forme hexadécimale, et les variables `Nk`, `Nb`, et `Nr` décrivant respectivement la taille de la clef en nombre de mots de 32 bits, la taille d'un bloc en nombre de mots de 32 bits et le nombre de tours.
Ensuite, la fonction crée un tableau `w` de taille $Nb(Nr+1)$ initialisé avec des 0 et un tableau `kw` contenant la clef de chiffrement sous forme de mots. Le tableau `w` est appelé à contenir les clefs de tours. Dans une première série d'itérations, la fonction initialise les premières clefs de tours par simple copie des mots de la clef de chiffrement. Puis, dans une deuxième série d'itérations, la fonction exécute l'algorithme d'expansion de la clef et intègre les mots obtenus dans le tableau `w` des clefs de tours.
En fin d'exécution, la fonction python `keyExpansion` retourne un tableau contenant les $Nb(Nr+1)$ clefs de tours.
La fonction python `keyExpansion` utilise la fonction python `keyToWords` pour transformer une clef de `Nk` mots en `Nk` blocs 32 bits (voir listing 11 p. 30).

2.3.3 Les vecteurs de test

Nous venons de détailler les mécanismes de chiffrement de l'*Advanced Encryption Standard* et, en parallèle, nous avons présenté la traduction de chacune de ces étapes en langage python. Le programme ainsi obtenu est disponible sur Internet [81, PythonAES].
Afin de valider une implémentation de l'algorithme AES, le *Federal Information Processing Standards* (FIPS) donne, dans le document normatif décrivant l'AES : le FIPS 197 [155], un certain nombre de vecteurs de tests et la description détaillée des opérations de chiffrement et de déchiffrement à partir de ces vecteurs.

```python
def keyToWords(key, Nk):
    keyWords = [[] for i in xrange(Nk)]
    word = 0
    tmp = ""
    for cpt in xrange(len(key)):
        if (cpt % 8 == 0) and (cpt <> 0):
            keyWords[word] = tmp
            word += 1
            tmp = key[cpt]
        else:
            tmp += key[cpt]
    keyWords[word] = tmp
    return keyWords
```

Listing 11 – Fonction keyToWords en python

Nous allons maintenant utiliser ces derniers pour valider notre implémentation de l'AES en langage python.
Commençons par l'algorithme d'expansion de la clef.
L'annexe A.1 du FIPS 197 donne le tableau des clefs de tour que l'on doit obtenir à partir de la clef de 128 bits suivante :

Cipher Key = `2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c`

Utilisons comme fonction principale de notre programme python la fonction suivante :

```python
if __name__ == "__main__":
    key = '2b7e151628aed2a6abf7158809cf4f3c'
    keyExp = keyExpansion(key)
```

Nous obtenons le résultat suivant :

```
key round 0 --> 2b7e1516    key round 24 --> 6d88a37a
key round 1 --> 28aed2a6    key round 25 --> 110b3efd
key round 2 --> abf71588    key round 26 --> dbf98641
key round 3 --> 09cf4f3c    key round 27 --> ca0093fd
key round 4 --> a0fafe17    key round 28 --> 4e54f70e
key round 5 --> 88542cb1    key round 29 --> 5f5fc9f3
key round 6 --> 23a33939    key round 30 --> 84a64fb2
key round 7 --> 2a6c7605    key round 31 --> 4ea6dc4f
key round 8 --> f2c295f2    key round 32 --> ead27321
key round 9 --> 7a96b943    key round 33 --> b58dbad2
key round 10 --> 5935807a    key round 34 --> 312bf560
key round 11 --> 7359f67f    key round 35 --> 7f8d292f
key round 12 --> 3d80477d    key round 36 --> ac7766f3
key round 13 --> 4716fe3e    key round 37 --> 19fadc21
key round 14 --> 1e237e44    key round 38 --> 28d12941
key round 15 --> 6d7a883b    key round 39 --> 575c006e
key round 16 --> ef44a541    key round 40 --> d014f9a8
key round 17 --> a8525b7f    key round 41 --> c9ee2589
key round 18 --> b671253b    key round 42 --> e13f0cc8
key round 19 --> db0bad00    key round 43 --> b6630ca6
```

```
key round 20 --> d4d1c6f8
key round 21 --> 7c839d87
key round 22 --> caf2b8bc
key round 23 --> 11f915bc
```

Ce résultat est conforme aux données du standard AES décrit dans le FIPS 197. En utilisant les vecteurs des clefs en 192 bits et en 256 bits nos résultats correspondent également aux données fournies par le FIPS 197. Notre implémentation de l'algorithme d'expansion de la clef est donc conforme et pourra servir de base de travail pour la suite de la thèse.
Une fois le programme décrivant l'expansion de la clef validé, nous pouvons maintenant contrôler la validité de notre implémentation de la fonction de chiffrement.
Pour le chiffrement avec une clef de 128 bits, le FIPS 197 propose les vecteurs de tests suivants :

Clear text = `00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`
Cipher Key = `00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f`

Utilisons comme fonction principale de notre programme python la fonction suivante :

```
if __name__ == "__main__":
    clearBlock = '00112233445566778899aabbccddeeff'
    key128 = '000102030405060708090a0b0c0d0e0f'
    keyExp = keyExpansion(key128, Nk=4, Nb=4, Nr=10)
    cipherBlock = cipher(clearBlock, keyExp, Nb=4, Nr=10)
```

Nous obtenons le résultat suivant :

```
round[ 0 ].input 00112233445566778899aabbccddeeff
round[ 0 ].k_sch 000102030405060708090a0b0c0d0e0f
round[ 1 ].start 00102030405060708090a0b0c0d0e0f0
round[ 1 ].s_box 63cab7040953d051cd60e0e7ba70e18c
round[ 1 ].s_row 6353e08c0960e104cd70b751bacad0e7
round[ 1 ].m_col 5f72641557f5bc92f7be3b291db9f91a
round[ 1 ].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe
...
round[ 9 ].start fde3bad205e5d0d73547964ef1fe37f1
round[ 9 ].s_box 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 9 ].s_row 54d990a16ba09ab596bbf40ea111702f
round[ 9 ].m_col e9f74eec023020f61bf2ccf2353c21c7
round[ 9 ].k_sch 549932d1f08557681093ed9cbe2c974e
round[ 10 ].start bd6e7c3df2b5779e0b61216e8b10b689
round[ 10 ].s_box 7a9f102789d5f50b2beffd9f3dca4ea7
round[ 10 ].s_row 7ad5fda789ef4e272bca100b3d9ff59f
round[ 10 ].k_sch 13111d7fe3944a17f307a78b4d2b30c5
round[ 10 ].output 69c4e0d86a7b0430d8cdb78070b4c55a
```

Ce résultat est conforme aux résultats donnés dans le FIPS 197 pour cet ensemble de vecteurs. Nous obtenons des résultats équivalents pour les vecteurs de tests du chiffrement avec une clef de 192 et de 256 bits. L'ensemble des résultats obtenus est détaillé dans l'annexe A page 163.

Notre implémentation de l'algorithme de chiffrement de l'AES en langage Python est donc conforme.

2.3.4 Opérations de déchiffrement

Le sujet de notre travail concernant essentiellement l'analyse des processus de chiffrement, nous allons décrire succinctement l'algorithme de déchiffrement mis en œuvre dans l'AES.
Le déchiffrement (voir fig. 2.17 p. 32) est réalisé en effectuant les opérations inverses des quatre fonctions de chiffrement, dans l'ordre inverse.

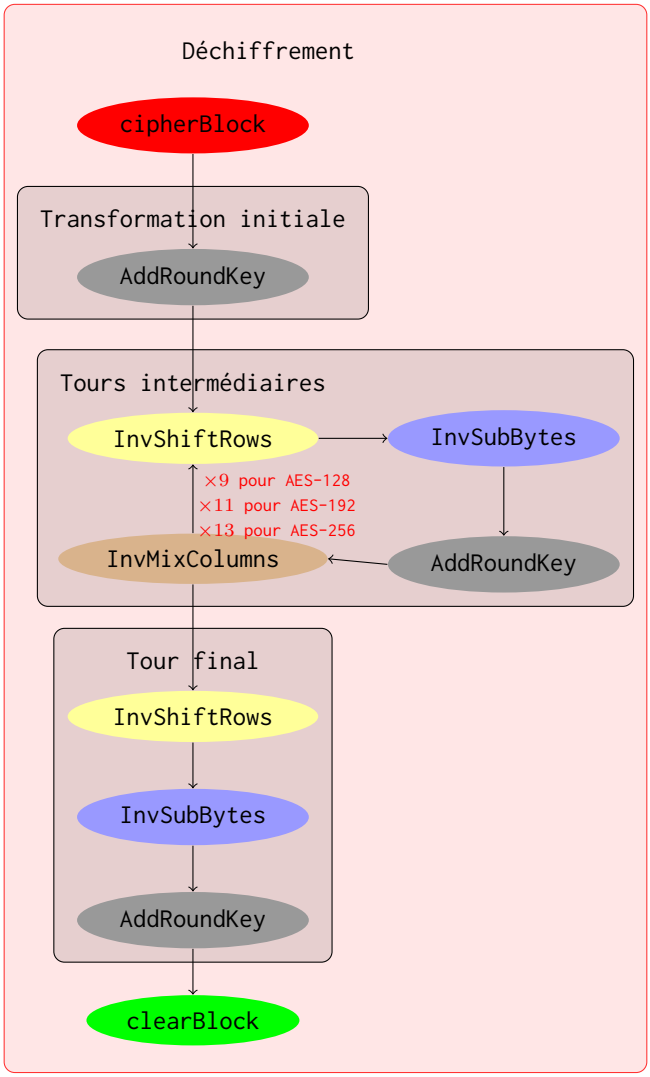


FIGURE 2.17 – Le processus de déchiffrement de l'AES

Ainsi, chaque fonction utilisée dans les opérations de chiffrement dispose de sa fonction inverse, utilisée pour le déchiffrement : `InvShiftRows`, `InvSubBytes` et

`InvMixColumns`. La fonction `AddRoundKey` reste inchangée. À l'instar du chiffrement, le processus de déchiffrement comprend une transformation initiale, des tours intermédiaires et un tour final.
La transformation initiale consiste à appliquer la fonction `AddRoundKey` au tableau d'états. Les tours intermédiaires exécutent, dans l'ordre, les fonctions `InvShiftRows`, `InvSubBytes`, `AddRoundKey` et `InvMixColumns` sur le tableau d'états. Le tour final diffère des tours intermédiaires, par la suppression de la fonction `InvMixColumns` dans le cycle des transformations.

—=oOo=—

Après une rapide introduction à la cryptographie et à ses principales primitives, nous avons détaillé le fonctionnement des algorithmes de chiffrement et de déchiffrement de l'Advanced Encryption Standard.
Parce que c'est un moyen très efficace de bien comprendre comment fonctionne un algorithme, nous avons développé, en parallèle, une implémentation de l'AES en langage python [5]. Enfin, nous avons montré que notre implémentation est conforme en l'utilisant avec les vecteurs de tests fournis par le NIST. Cette implémentation nous servira dans la suite de notre travail.
Nous pouvons maintenant nous plonger dans l'algèbre et les travaux récents sur l'écriture de l'AES sous la forme d'un système d'équations.

5. Le code source est disponible sur Internet : https://github.com/archoad.

Chapitre 3

Représentation algébrique de l'AES

« Breaking a good cipher should require as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type. »

Claude Elwood Shannon

Afin de contrer les méthodes de cryptanalyse usuelles, reposant sur des méthodes statistiques, Joan Daemen et Vincent Rijmen, les concepteurs de l'*Advanced Encryption Standard* (AES) lui ont donné une forte architecture algébrique. Ce choix complexifie grandement toute tentative d'attaque statistique, mais il pourrait également être sa faiblesse, en le rendant susceptible d'être compromis par ces mêmes structures algébriques [94, 19, 173].
En effet, de récents travaux [49, 63, 58] tendent à montrer que cryptanalyser l'AES pourrait se "résumer" à résoudre un système d'équations quadratiques symbolisant la structure du chiffrement de l'AES.
Mais avant de détailler ces structures, nous allons présenter les fondements mathématiques sur lesquels elles s'appuient.

3.1 Fondements mathématiques

La plupart des opérations dans Rijndael s'effectuent au niveau de l'octet ou sur un mot de quatre octets. L'octet est donc le plus petit élément d'information traité dans l'algorithme. Mathématiquement, ce dernier peut être considéré à la fois comme une chaîne binaire, un vecteur à coefficients dans $GF(2)$ de dimension 8 et un élément de $GF(2^8)$ c'est-à-dire, un polynôme de degré inférieur ou égal à 7 et dont les coefficients sont les bits (voir fig. 3.1 p. 36).
La conception de l'AES s'articule autour des corps de Galois [155]. Un corps de Galois, est un corps contenant un nombre fini d'éléments. Toutes les opérations utilisées dans l'AES sont donc décrites par des opérations algébriques sur des corps finis de même caractéristique.

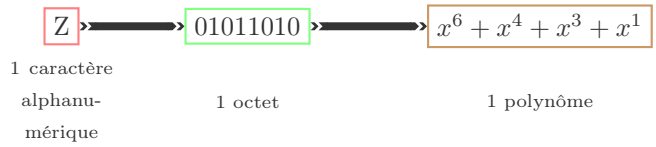


FIGURE 3.1 – Représentation d'un caractère dans l'AES

Avant d'appréhender les opérations arithmétiques de base dans les corps de Galois, décrivons les structures algébriques nécessaires à la compréhension des propriétés des corps finis.

3.1.1 Les groupes

Les groupes constituent la structure algébrique de base des mathématiques, puisque à partir de ceux-ci sont créés les anneaux, les corps, les espaces vectoriels... [129]
Soit G un ensemble non vide doté d'une loi de composition interne :

$$\circ : G \times G \mapsto G$$

Alors, G est un groupe noté $(G, \circ)$, si :
- sa loi de composition interne est associative ;
- il existe un élément neutre e tel que $e \circ g = g \circ e = g, \forall g \in G$. L'élément e est unique et est aussi nommé *élément identité* ;
- $\forall g \in G$ il existe un unique $g^{-1} \in G$ tel que $g \circ g^{-1} = g^{-1} \circ g = e$.

L'ordre de $(G, \circ)$ est la cardinalité de l'ensemble G. Si l'ordre de $(G, \circ)$ est fini, alors $(G, \circ)$ est un groupe fini. Enfin, un groupe, dont la loi de composition interne est commutative, est un groupe commutatif ou encore appelé groupe abélien.
À titre d'exemple, l'ensemble des entiers relatifs $\mathbb{Z}$ muni de l'opération d'addition forme un groupe abélien noté $(\mathbb{Z}, +)$.
Une permutation sur un ensemble non vide E est une bijection de $E \mapsto E$. L'ensemble des permutations de E, muni de l'opération de composition, forme un groupe nommé *groupe symétrique* de E et noté $\mathfrak{S}_E$. Si E est fini et de cardinalité $n > 0, n \in \mathbb{N}$, le groupe symétrique de cet ensemble E est nommé *groupe symétrique d'indice* n, il est noté $\mathfrak{S}_n$. L'orde de $\mathfrak{S}_n$ est $n!$, les éléments de $\mathfrak{S}_n$ sont des permutations. Un élément de $\mathfrak{S}_n$ qui permute deux éléments de E et laisse les autres éléments inchangés est appelé une transposition.
Soit deux groupes $(G, \bullet)$ et $(H, \star)$, la fonction $f : G \mapsto H$ est un *homomorphisme* de groupe si $\forall g, g' \in G$ alors $f(g \bullet g') = f(g) \star f(g')$. Un homomorphisme de groupe bijectif est appelé *isomorphisme*. Dans ce cas, G et H sont dit *isomorphes*, ce qui se note par $G \cong H$. Deux groupes isomorphes ont la même structure algébrique et peuvent être considérés comme représentant le même objet algébrique.

3.1.2 Les anneaux

Soit A un ensemble non vide associé à deux opérations binaires internes $+$ et $\bullet$ de $A \times A \mapsto A$. Alors $(A, +, \bullet)$ est un anneau (voir fig. 3.2 p. 37) si :

— $(A, +)$ est un groupe abélien ;
— l'opération $\bullet$ est associative ;
— l'opération $\bullet$ est distributive sur $+$;
— il existe un élément $1 \in A$ tel que $1 \bullet a = a \bullet 1 = a, \forall a \in A$.

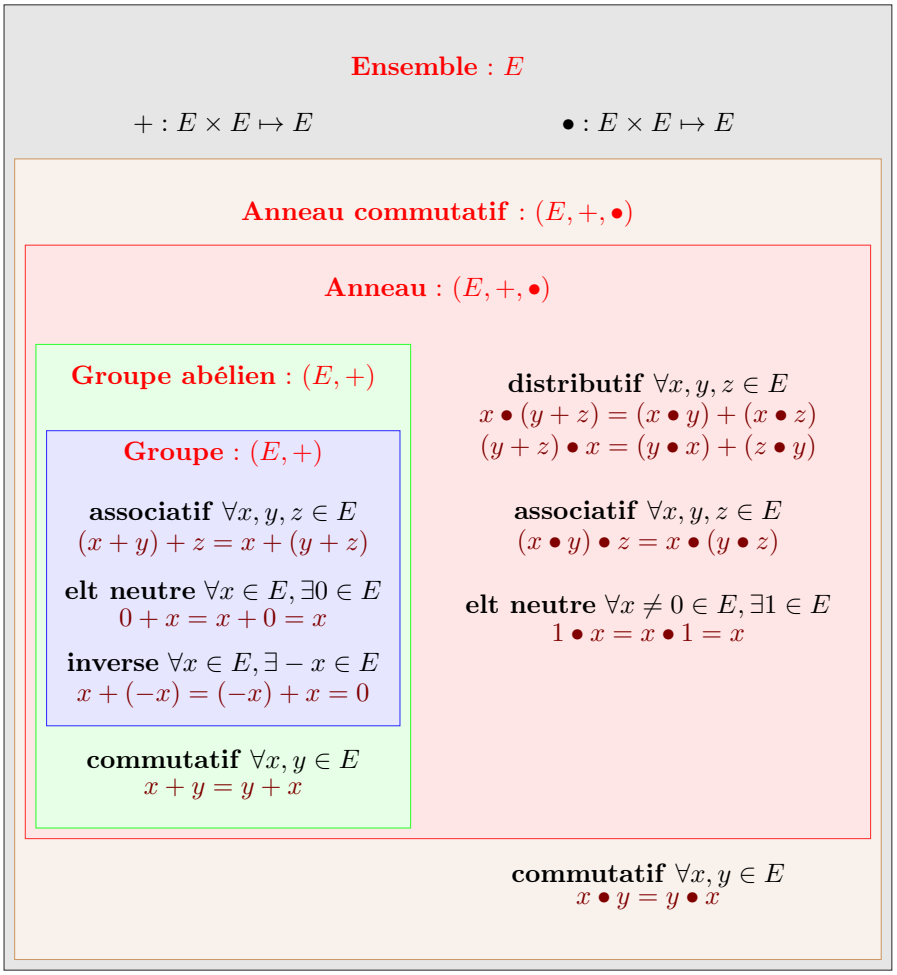


FIGURE 3.2 – Architecture d'un anneau

L'élément identité de $(A, +)$ est 0 et constitue le zéro de l'anneau $(A, +, \bullet)$. L'élément 1 est l'élément identité de l'anneau $(A, +, \bullet)$. Enfin, un anneau est commutatif si sa seconde loi est commutative.
L'ensemble des entiers relatifs $\mathbb{Z}$ muni des opérations d'addition et de multiplication forme un anneau commutatif noté $(\mathbb{Z}, +, \bullet)$.
Un anneau commutatif $(A, +, \bullet)$ est un *anneau intègre* s'il ne contient aucun diviseur de zéro, c'est-à-dire si $a \bullet a' \neq 0 \quad \forall a, a' \in A \backslash \{0\}$.
Un élément non nul $a \in A$ est dit inversible s'il existe $a^{-1} \in A$ tel que $a \bullet a^{-1} = a^{-1} \bullet a = 1$.
Soit l'anneau $(A, +, \bullet)$ et I un sous-ensemble non vide de A, I est un *idéal* de A, noté $I \triangleleft A$, si $(I, +)$ est un sous groupe de $(A, +)$ et si $\forall i \in I$ et $\forall a \in A$ alors

$i \bullet a \in I$ et $a \bullet i \in I$. Ainsi, quel que soit $k \in \mathbb{Z}$, $k\mathbb{Z}$ est un idéal de $(\mathbb{Z}, +, \bullet)$.
Si S est un sous-ensemble non-vide de l'anneau A, alors l'idéal engendré par S est noté $\langle S \rangle$ et consiste en toutes les sommes finies de la forme $\sum a_i \bullet s_i$ où $a_i \in A$ et $s_i \in S$.
L'idéal I de l'anneau $(A, +, \bullet)$ est dit *principal* s'il existe un élément $a \in A$ tel que $I = a.A$. Autrement dit, I est principal s'il peut être engendré par un élément $a \in A$.
Un anneau commutatif est dit *principal* si tous ses idéaux sont principaux. Un anneau intègre dans lequel chaque idéal est un idéal principal est appelé *anneau principal*.
Un anneau A est *factoriel* s'il est intègre, si tout élément non nul admet une décomposition en produit de facteurs irréductibles et si cette décomposition est unique à permutation près. C'est à dire $\forall a \in A \backslash \{0\}$ il existe $u \in A^*$ et $p_1, \cdots, p_r$ irréductibles tels que $a = up_1 \cdots p_r$ et que cette décomposition est unique.

3.1.3 Les corps

Un corps est un anneau dans lequel tout élément non nul est inversible. Autrement dit, le corps K est l'anneau $(K, +, \bullet)$ tel que $(K, +)$ et $(K \setminus \{0\}, \bullet)$ sont des groupes abéliens. L'*ordre* d'un corps est le nombre d'éléments qu'il contient. L'ensemble des nombres relatifs $\mathbb{Q}$, l'ensemble des nombres réels $\mathbb{R}$ et l'ensemble des nombres complexes $\mathbb{C}$ sont des corps munis des opérations d'addition et de multiplication.

3.1.4 Les anneaux polynomiaux

L'analyse algébrique de l'AES fait un usage intensif des anneaux polynomiaux. Un anneau polynomial est un cas particulier d'anneau commutatif.

3.1.4.1 Les anneaux polynomiaux univariés

Un polynôme à une variable x, ou *polynôme univarié* en x, dans un corps K est une combinaison linéaire finie sur K de monômes en x dont l'expression formelle est de la forme :

$$c_d x^d + c_{d-1} x^{d-1} + \cdots + c_2 x^2 + c_1 x + c_0 \tag{3.1}$$

avec d un entier positif, $c_d, \cdots, c_0 \in K$ et si $d > 0$ alors $c_d \neq 0$.
L'ensemble de tous les polynômes à une variable x dans un corps K forme un anneau, noté $K[x]$, muni des opérations standards d'addition et de multiplication polynomiales. Cet anneau est un anneau principal nommé *anneau des polynômes univariés* sur K. Comme l'anneau des polynômes univariés sur le corps K est principal il est également factoriel.
Soit $f(x) \in K[x]$ un polynôme univarié tel que :

$$f(x) = c_d x^d + c_{d-1} x^{d-1} + \cdots + c_i x^i + \cdots + c_2 x^2 + c_1 x + c_0 \tag{3.2}$$

le degré de $f(x)$, noté $deg(f(x))$, est l'entier maximum d tel que $c_d \neq 0$, les $c_i x^i$ sont les *termes* de $f(x)$ et c_i est le *coefficient* du monôme x^i.

Soient $f(x), g(x) \in K[x]$, il existe $q(x), r(x) \in K[x]$ avec $deg(r(x)) < deg(g(x))$ tel que $f(x) = q(x) \bullet g(x) + r(x)$. Un polynôme $f(x) \in K[x]$ avec $deg(f(x)) > 0$ est dit irréductible s'il n'y a pas de factorisation de la forme $f(x) = q(x) \bullet g(x)$ avec $q(x), g(x) \in K[x]$ et $deg(g(x)) > 0$ et $deg(q(x)) > 0$.

3.1.4.2 Les anneaux polynomiaux multivariés

Un polynôme à n variables $x_1, \cdots, x_n$, ou *polynôme multivarié*, dans un corps K est une combinaison linéaire finie sur K de monômes en $x_1, \cdots, x_n$ dont l'expression formelle est de la forme :

$$\sum_{\alpha} c_\alpha x^\alpha \tag{3.3}$$

avec $c_\alpha \in K$ et pour tout $\alpha = (\alpha_1, \cdots, \alpha_n)$ et $\alpha_i \in \mathbb{N} \setminus \{0\}$:

$$x^\alpha = \prod_{i=1}^{n} x_i^{\alpha_i} = x_1^{\alpha_1} \cdots x_n^{\alpha_n} \tag{3.4}$$

L'ensemble de tous les polynômes à n variables dans un corps K forme un anneau, noté $K[x_1, \cdots, x_n]$, muni des opérations standards d'addition et de multiplication polynomiales. Cet anneau, qui n'est pas principal, est nommé *anneau des polynômes multivariés* sur K. Enfin, l'anneau $K[x_1, \cdots, x_n]$ est factoriel car K est factoriel.
L'anneau des polynômes $K[x_1, \cdots, x_n]$ est un espace vectoriel sur K.
Un *ordre monomial* est un ordre total sur tous les monômes d'un anneau polynomial tel que pour tout triplet de monômes m_1, m_2, m_3, si $m_1 < m_2$ alors $m_1 m_3 < m_2 m_3$. À titre d'exemple, l'ordre monomial *lexicographique* est défini par $x^\alpha <_{lex} x^\beta$ si la première entrée non nulle du vecteur $\beta - \alpha$ est positive. De même, l'ordre monomial *degré-lexicographique* est défini par $x^\alpha <_{grlex} x^\beta$ si $|\alpha| < |\beta|$ puis si $|\alpha| = |\beta|$ et $x^\alpha <_{lex} x^\beta$.
Soit $<$ un ordre monomial sur $K[x_1, \cdots, x_n]$ et $f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha x^\alpha \in K[x_1, \cdots, x_n]$, alors le *monôme de tête* de f est $LM(f) = \max(x^\alpha)$ avec $c_\alpha \neq 0$, le *coefficient de tête* de f est $LC(f) = c_\alpha \in K$ avec α tel que $LM(f) = x^\alpha$ et le *terme de tête* de f est $LT(f) = LC(f) \cdot LM(f)$. Enfin, pour un idéal I, $LT(I) = \{LT(f) : f \in I\}$.

3.1.5 Les corps finis ou corps de Galois $GF(2^8)$

L'ensemble $\mathbb{Z}_p = \{0, \cdots, p-1\}$, noté $\mathbb{Z}/p$, muni des opérations d'addition et de multiplication modulo p forme un corps fini si et seulement si p est premier. En l'honneur du mathématicien français Évariste Galois, fondateur de la théorie des corps finis, ce corps est nommé corps de Galois. Il est d'ordre p et est noté $GF(p)$.
Comme un corps de Galois est en fait l'anneau des entiers modulo p, les opérations d'addition, de soustraction et de multiplication sont réalisées comme sur un entier standard suivi de la réduction modulo p. Ainsi, dans $GF(5)$, $4+3=7$ est réduit à 2 modulo 5. Enfin, la division dans un corps de Galois est la multiplication par l'inverse modulo p.
Le plus petit corps de Galois : $GF(2)$, contient les éléments 0 et 1 et nous avons $GF(2) = \{0, 1\}$. Dans $GF(2)$ l'addition est réalisée par un ou exclusif XOR et

la multiplication par un AND. Enfin, comme le seul élément inversible est 1, la division est la fonction identité.

3.1.5.1 Construction des corps de Galois

Soit $K[x]$ un anneau polynomial formé de l'ensemble de tous les polynômes à une variable x sur le corps K et muni des opérations d'addition et de multiplication polynomiales. L'anneau quotient $\frac{K[x]}{\langle f(x)\rangle}$ est un corps si et seulement si $f(x)$ est irréductible dans $K[x]$.
Soient K un corps fini d'ordre $q = p^n$ et $f(x) \in K[x]$ un polynôme irréductible de degré d. L'anneau quotient $A = \frac{K[x]}{\langle f(x)\rangle}$ est un corps d'ordre $q^d = p^{nd}$ qui est une extension de degré d de K. Ces éléments peuvent être présentés sous la forme :

$$c_{d-1}x^{d-1} + \cdots + c_2x^2 + c_1x + c_0 \tag{3.5}$$

avec $c_i \in K$. Tout corps fini d'ordre p^{nd} est isomorphe à A.
Il existe d'autres façons de construire et de représenter les éléments d'un corps fini [131]. Nous retiendrons que, à toute puissance première correspond un corps fini et que, par conséquent, toutes les représentations de $GF(2^8)$ sont isomorphes. Cependant, malgré cette équivalence, la représentation choisie a un impact sur la complexité de l'implémentation. Le standard de l'AES a choisi d'utiliser la représentation polynomiale.
Ainsi, un octet contenant les bits $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ est considéré comme un polynôme avec des coefficient dans $\{0, 1\}$:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \tag{3.6}$$

Par exemple, le caractère alphanumérique `"a"` a la valeur hexadécimale `61`, soit `01100001` en binaire, dans la table de correspondance ASCII. Sa traduction polynomiale est donc la suivante :

$$x^6 + x^5 + 1 \tag{3.7}$$

3.1.6 Opérations dans $GF(2^8)$

C'est dans le corps fini $GF(2^8)$ que l'AES effectue ses opérations. Concrètement, cela signifie que les calculs sont effectués sur des polynômes de degré 7 à coefficients dans $\{0, 1\}$, modulo un polynôme irréductible $m(x)$ de degré 8 valant $x^8 + x^4 + x^3 + x + 1$ pour l'AES.

3.1.6.1 L'addition

L'addition de deux éléments dans $GF(2^8)$ est réalisée en ajoutant les coefficients des puissances correspondantes des polynômes représentant ces éléments. Cet ajout s'effectuant modulo 2, cela revient à appliquer un XOR sur les coefficients.

3.1.6.2 La multiplication

La multiplication dans $GF(2^8)$ s'effectue modulo un polynôme irréductible. Nous avons vu que pour l'AES, ce polynôme est $m(x) = x^8 + x^4 + x^3 + x + 1$. La réduction modulaire permet de garantir que le résultat de la multiplication sera un polynôme de degré inférieur à 8 et donc, que ce dernier correspondra bien à la représentation d'un octet.
La multiplication de deux polynômes s'effectue en deux étapes. Dans un premier temps, la multiplication est effectuée classiquement et dans un deuxième temps, la réduction polynomiale est appliquée au résultat.

3.2 Description algébrique de l'AES

La plupart des attaques connues sur les algorithmes de chiffrement par blocs comme la cryptanalyse différentielle [18] et sa version améliorée : l'attaque boomerang [205], la cryptanalyse linéaire [13], la cryptanalyse intégrale et son application sur l'algorithme Square [68], sont des attaques probabilistes et s'appuient sur le fait que l'attaquant dispose d'une grande quantité de paires de textes clairs/textes chiffrés.
Comme nous l'avons vu dans l'introduction de ce chapitre, les concepteurs de l'AES ont utilisé des outils algébriques pour fournir à leur algorithme, un niveau de garantie inégalé [50] contre les techniques de cryptanalyse statistique standards. Pourtant dès 2001, une nouvelle voie de cryptanalyse de l'AES fait son apparition, ironiquement cette voie utilise l'algèbre pour attaquer Rijndael. En effet, fin 2001, Ferguson, Schroeppel et Whiting montrent comment écrire Rijndael en une équation algébrique fermée [1] sur $GF(2^8)$ [95]. En s'appuyant sur le fait que, dans un corps de Galois, l'élévation au carré est une opération linéaire, ils proposent une écriture de l'AES sous la forme d'une équation algébrique contenant 2^{50} termes pour une taille de clef de 128 bits et 2^{70} termes pour une clef de 256 bits. La cryptanalyse de l'AES dépend donc maintenant de la difficulté de résolution d'une telle équation dans $GF(2^8)$.
Quelques mois plus tard, début 2002, Courtois et Pieprzyk publient un article dans lequel ils décrivent une nouvelle attaque contre Rijndael et Serpent [63]. Le document décrit une attaque que les auteurs qualifient « *de plus efficace que la force brute* »contre Serpent et peut-être aussi contre Rijndael. Leur attaque repose, dans un premier temps, sur la description de l'AES sous la forme d'un système d'équations polynomiales multivariées sur $GF(2)$ et, dans un deuxième temps, sur un algorithme de résolution de ce système d'équations baptisé `Extended Sparse Linearization` (XSL). Selon les auteurs, les points clefs de leur attaque sont que les équations sont quadratiques, que le système est surdéterminé et que leur représentation matricielle est creuse.
Toujours en 2002, dans le but de diminuer la complexité de la cryptanalyse de l'AES liée au fait que les opérations y sont basées dans deux corps de Galois différents, $GF(2)$ et $GF(2^8)$, Murphy et Robshaw créent le `Big Encryption System` (BES)[147].
Dans l'AES les blocs de textes clairs sont intégrés dans un tableau d'états qui

1. Une équation est dite a solution fermée si elle résout un problème donné en termes de fonctions et d'opérations mathématiques à partir d'un ensemble donné généralement accepté [208].

est transformé par les différentes fonctions de chiffrement des tours. Tandis que l'AES a un tableau d'états de 16 octets, le BES travaille sur un tableau d'états de 128 octets. Ainsi, si nous notons **A** l'espace du tableau d'états de l'AES et **B** l'espace du tableau d'états du BES, nous avons les équivalences décrites dans le tableau suivant (voir fig. 3.3 p. 42).

A	Tableau d'états de l'AES	Espace vectoriel F^{16}
B	Tableau d'états du BES	Espace vectoriel F^{128}
$\mathbf{B_A}$	Sous-ensemble de **B** correspondant à **A**	Sous-ensemble de F^{128}

FIGURE 3.3 – Définition et équivalences entre les tableaux d'états de l'AES et du BES

Ainsi, en partant de l'AES comme un sous-ensemble du BES, Murphy et Robshaw offrent une plateforme simplifiée pour l'analyse de l'AES. À partir de cette plateforme, ils décrivent les opérations de chiffrement de l'AES sous la forme d'un systèmes d'équations quadratiques multivariées comprenant 5248 équations, 3968 variables. Il suffit donc de résoudre ce systèmes d'équations pour retrouver la clef de chiffrement utilisée.
En septembre 2002, dans une note transmise au projet New European Schemes for Signature, Integrity, and Encryption (NESSIE) [51], Murphy et Robshaw précisent le lien entre le BES et l'attaque XSL. Selon eux, l'effort nécessaire – environ 2^{230} étapes – pour recouvrer la clef à partir de l'attaque XSL appliqué au système d'équations proposé par Courtois et Pieprzyk est beaucoup plus conséquent que l'effort à fournir – environ 2^{100} étapes – pour obtenir le même résultat sur leur système d'équations.
Comme nous venons de voir, il existe plusieurs représentations algébriques de l'AES, nous allons maintenant nous intéresser à celle exposée dans [95].

3.2.1 Équation pour la S-Box

La *S-Box* de l'AES est la composition de trois opérations algébriques simples :

1. une inversion dans $GF(2^8)$ avec $0^{-1} \mapsto 0$;
2. une fonction de permutation linéaire : chaque bit en sortie est la somme de certains bits en entrée ;
3. l'addition de la constante `63`

La fonction d'inversion est donnée par

$$x \mapsto x^{(-1)} = x^{254} \tag{3.8}$$

et la fonction de permutation est donnée par la fonction polynomiale

$$x \mapsto \texttt{05}x^{254} + \texttt{09}x^{253} + \texttt{f9}x^{251} + \texttt{25}x^{247} + \texttt{f4}x^{239} + \texttt{01}x^{223} + \texttt{b5}x^{191} + \texttt{8f}x^{127} \tag{3.9}$$

Ainsi, la *S-Box* est définie par la fonction polynomiale sur F :

$$S(x) = \texttt{63} + \sum_{d=0}^{7} w_d x^{255-2^d} \tag{3.10}$$

où les w_d sont définis dans la fonction polynomiale décrite par l'équation 3.9.
L'équation 3.10 peut être simplifiée en effectuant les approximations suivantes :

— suppression de la constante `63` et remplacement par l'ajout d'une constante spécifique à la clef de chiffrement ;
— suppression de la puissance 255 en partant du principe que $x^{255} = 1$ pour tous les x sauf quand $x = 0$.

L'équation 3.10 devient alors :

$$S(x) = \sum_{d=0}^{7} w_d x^{-2^d} = \sum_{d=0}^{7} \frac{w_d}{x^{2^d}} \tag{3.11}$$

3.2.2 Équation pour un tour de l'AES

En utilisant la notation détaillée dans l'annexe A de [94], $a_{i,j}^{(r)}$ correspond à l'octet situé dans la ligne i, colonne j du tableau d'états au début du tour r. De même, $s_{i,j}^{(r)}$ représente l'octet situé à la position (i, j) après l'exécution de la fonction `subBytes` du tour r.
Si l'on détaille le fonctionnement d'un tour, nous obtenons tout d'abord, à partir de l'équation 3.11, après application de la *S-Box* à chaque octet du tableau d'états :

$$s_{i,j}^{(r)} = S(a_{i,j}^{(r)}) = \sum_{d_r=0}^{7} w_{d_r} (a_{i,j}^{(r)})^{-2^{d_r}} \tag{3.12}$$

Ensuite, en prenant $t_{i,j}^{(r)}$ comme étant l'octet à la position (i, j) après application de la fonction `ShiftRows`, le tableau d'états peut s'écrire sous la forme :

$$t_{i,j}^{(r)} = s_{i,j+i}^{(r)} = \sum_{d_r=0}^{7} w_{d_r} (a_{i,j+i}^{(r)})^{-2^{d_r}} \tag{3.13}$$

Définissons maintenant $m_{i,j}^{(r)}$ comme étant l'octet à la position (i, j) après application de la fonction `MixColumn`. Cette dernière peut s'écrire de la façon suivante :

$$m_{i,j}^{(r)} = \sum_{e_r=0}^{3} v_{i,e_r} t_{e_r,j}^{(r)} \tag{3.14}$$

où les $v_{i,j}$ sont les coefficients de la matrice de diffusion

$$M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \tag{3.15}$$

À partir de là, après l'application de la fonction `MixColumns`, le tableau d'états s'écrit sous la forme :

$$
\begin{aligned}
m_{i,j}^{(r)} &= \sum_{e_r=0}^{3} v_{i,e_r} \sum_{d_r=0}^{7} w_{d_r} (a_{e_r,j-e_r}^{(r)})^{-2^{d_r}} \\
&= \sum_{e_r=0}^{3} \sum_{d_r=0}^{7} w_{i,e_r,d_r} (a_{e_r,e_r+j}^{(r)})^{-2^{d_r}}
\end{aligned} \tag{3.16}
$$

Enfin, la dernière étape est réalisée par la fonction `AddRoundKey` qui ajoute la clef :

$$
\begin{aligned}
a_{i,j}^{(r+1)} =& m_{i,j}^{(r)} + k_{i,j}^{(r)} \\
=& k_{i,j}^{(r)} + \sum_{e_r=0}^{3} \sum_{d_r=0}^{7} w_{i,e_r,d_r} (a_{e_r,e_r+j}^{(r)})^{-2^{d_r}} \\
=& k_{i,j}^{(r)} + \sum_{e_r=0}^{3} \sum_{d_r=0}^{7} \frac{w_{i,e_r,d_r}}{(a_{e_r,e_r+j}^{(r)})^{2^{d_r}}} \\
=& k_{i,j}^{(r)} + \sum_{\substack{e_r=0\ldots3 \\ d_r=0\ldots7}} \frac{w_{i,e_r,d_r}}{(a_{e_r,e_r+j}^{(r)})^{2^{d_r}}}
\end{aligned} \tag{3.17}
$$

où $k_{i,j}^{(r)}$ est la clef du tour r à la position (i,j).
L'équation 3.17 est donc une expression algébrique décrivant les transformations du tableau d'états durant un tour.

3.2.3 Généralisation

Si nous appliquons ce résultat pour calculer l'équation des trois premiers tours, nous obtenons.

$$
a_{i,j}^{(4)} = k_{i,j}^{(3)} + \sum_{e_3,d_3} \frac{w_{i,e_3,d_3}}{\left(k_{e_3,e_r+j}^{(2)} + \sum_{e_2,d_2} \frac{w_{e_3,e_2,d_2}}{\left(k_{e_2,e_2+e_3+j}^{(1)} + \sum_{e_1,d_1} \frac{w_{e_2,e_1,d_1}}{\left(a_{e_1,e_1+e_2+e_3+j}^{(1)} \right)^{2^{d_1}}} \right)^{2^{d_2}}} \right)^{2^{d_3}}} \tag{3.18}
$$

Comme nous travaillons dans $GF(2)$, nous pouvons utiliser une de ses propriété : $(a+b)^2 = a^2 + b^2$. Ainsi, l'équation 3.18 devient :

$$
a_{i,j}^{(4)} = k_{i,j}^{(3)} + \sum_{e_3,d_3} \frac{w_{i,e_3,d_3}}{(k_{e_3,e_r+j}^{(2)})^{2^{d_3}} + \sum_{e_2,d_2} \frac{w_{e_3,e_2,d_2}^{2^{d_3}}}{(k_{e_2,e_2+e_3+j}^{(1)})^{2^{d_2+d_3}} + \sum_{e_1,d_1} \frac{w_{e_2,e_1,d_1}^{2^{d_2+d_3}}}{\left(a_{e_1,e_1+e_2+e_3+j}^{(1)} \right)^{2^{d_1+d_2+d_3}}}}} \tag{3.19}
$$

Cette formule semble compliquée, mais nous pouvons la simplifier. En effet, chaque numérateur est une constante que l'on va noter C, de même nous allons noter K chaque octet d'expansion de la clef et enfin, nous remplaçons chaque puissance par un $*$. Nous pouvons réaliser ces simplifications car chacune de ces données est connue et indépendante de la clef et du texte clair. Enfin, nous utilisons le fait que les octets en entrée du tableau d'états, avant le premier tour, peuvent s'écrire : $a_{i,j}^{(1)} = p_{4j+i} + k_{i,j}^{(0)}$ où les $p_{i,j}$ sont les octets de texte clair.
En prenant en compte ces simplifications, l'équation 3.19 devient :

$$a_{i,j}^{(4)} = K + \sum_{e_3,d_3} \frac{C}{K^* + \sum_{e_2,d_2} \frac{C}{K^* + \sum_{e_1,d_1} \frac{C}{K^* + p_*^*}}} \tag{3.20}$$

avec chaque K correspondant à un octet issu de la procédure d'expansion de la clef, chaque C correspondant à une constante connue et chaque $*$ correspondant à un exposant ou un indice connu. L'équation 3.20 nous donne donc la valeur intermédiaire du processus de chiffrement, après trois tours, sous la forme d'une fonction dont les variables sont les octets de texte clair et les octets du processus d'expansion de la clef.
La version la plus simple de l'AES compte 10 tours, ce qui, en généralisant l'équation 3.20 et en supprimant les symboles de sommations, correspond finalement à une expression algébrique comptant environs 2^{50} termes [2]. Si nous souhaitions stocker une telle expression, en partant du principe q'un terme correspond à un octet, nous aurions besoin de 1024 To d'espace de stockage. Cette technique représente malgré tout un avantage. En effet, le DES pouvait également être décrit sous la forme d'une expression algébrique comptant 2^{64} [95] termes alors qu'il n'était basé que sur des blocs de 64 bits et sur une clef de 56 bits.
Nous venons de décrire une représentation algébrique possible de l'AES. Malheureusement, comme le précisent les inventeurs de cette équation [95, page 8], il est impossible pour un ordinateur de résoudre une équation algébrique de cette forme. Afin de présenter des résultats pratiques, nous allons nous intéresser maintenant à une autre représentation algébrique de l'AES.

3.3 Application pratique

3.3.1 L'AES sous forme de système d'équations

« Breaking a good cipher should require as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type. »[189]
Cette phrase [3] célèbre de Claude Elwood Shannon est au cœur de ce que nous allons décrire à présent. En effet, le système d'équations quadratiques multivariées que nous allons présenter est implémenté sur ordinateur et il est donc possible de le mettre pratiquement en œuvre. Ce système d'équations est décrit par Cid, Murphy et Robshaw dans [49].
Ils y définissent deux variantes réduites de l'AES notées SR et SR^*. Ces deux variantes ont les mêmes paramètres à savoir : n le nombre de tours pour le

2. Pour l'AES 256, qui compte 14 tours, le nombre de termes s'élève à 2^{70}.
3. Briser un bon algorithme de chiffrement doit exiger autant de travail que la résolution d'un système d'équations avec un grand nombre d'inconnues de type complexe.

chiffrement, r et c définissent respectivement le nombre de lignes et de colonnes de la table d'entrée et e la taille d'un mot en bits. SR et SR^* diffèrent par la forme du dernier tour. Ces deux environnements travaillent sur des blocs de taille $r \times c \times e$ et le tableau d'états est un tableau $r \times c$ contenant des mots de e bits. Enfin, la version complète de l'AES-128 est décrite par : $SR^*(10, 4, 4, 8)$.
L'existence d'un système d'équations quadratiques multivariées a été montrée [147] en définissant le *Big Encryption System (BES)*. Le BES est un système de chiffrement par bloc utilisant des blocs de 128 octets et une clef de 16 octets. L'AES et le BES utilisent tous les deux un tableau d'états contenant des octets, tableau qui est transformé durant les différentes étapes des tours.
Les espaces d'états de l'AES et du BES sont respectivement les espaces vectoriels $A = GF(2^8)^{16}$ et $B = GF(2^8)^{128}$. Tous les textes clairs, textes chiffrés et clefs possibles sont des éléments de A pour l'AES et de B pour le BES. Ainsi, l'AES est un sous-ensemble du BES et la relation définissant l'image de l'AES dans le BES est donnée par :

$$B_A = \phi(A) \subset B$$

où $\phi() : GF(2^8) \mapsto GF(2^8)^8$ est le vecteur conjugué de l'application entre A et B défini par :

$$\phi(a) = \left(a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7}\right)$$

Ainsi, chaque élément $a \in GF(2^8)$ peut être inclus comme un élément

$$\left(a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7}\right) \in GF(2^8)^8$$

avec $\phi()$.
L'espace d'états de BES est $B = GF(2^8)^{128}$ et l'équation d'un tour de chiffrement dans BES est donnée par :

$$b \mapsto M_B(b^{(-1)}) + k_{B_i}$$

où M_B est la matrice de diffusion linéaire définie dans [50] et k_{B_i} est la clef BES pour le tour i.
En prenant la version à 10 tours de l'AES, le chiffrement du texte clair $p \in B$ vers le texte chiffré $c \in B$ par le BES est décrit par le système d'équation suivant :

$$w_0 = p + k_0 \tag{3.21a}$$
$$x_i = w_i^{(-1)} \qquad [i = 0, \cdots, 9] \tag{3.21b}$$
$$w_i = M_B x_{i-1} + k_i \qquad [i = 0, \cdots, 9] \tag{3.21c}$$
$$c = M_B^* x_9 + k_{10} \tag{3.21d}$$

où $w \in B$ et $x \in B$ sont les vecteurs d'états respectivement avant et après l'opération d'inversion et M_B^* est la version modifiée de la matrice M_B pour le tour final.

3.3.2 Implémentation de SR

Afin d'étudier les systèmes d'équations précédemment décrits, nous allons utiliser le logiciel `Sage` [192]. `Sage` est l'acronyme de *Software for Algebra and Geometry Experimentation*, c'est un logiciel open source, basé sur le langage python.

Il a pour objectif de fournir une alternative viable à Magma [28], Maple [57], Mathematica [169] et Matlab [134].
Sage est fourni par défaut avec un certain nombre de modules comprenant notamment des outils pour travailler sur des constructions algébriques. Ainsi, le code suivant montre comment créer un anneau polynomial univarié sur l'anneau $\mathbb{Z}$:

```
sage: Z.<x> = ZZ[]
sage: Z
Univariate Polynomial Ring in x over Integer Ring
sage:
```

Une fois cet anneau créé, plusieurs opérations sont possibles comme la génération d'un membre aléatoire de l'anneau, l'affichage de la caractéristique de l'anneau ou encore de sa finitude :

```
sage: Z.random_element()
-x^2 - x + 3
sage: Z.characteristic()
0
sage: Z.is_finite()
False
```

De la même façon, il est possible de travailler avec des anneaux polynomiaux sur $GF(2)$:

```
sage: A.<x> = GF(2)[]
sage: A
Univariate Polynomial Ring in x over Finite Field of size 2
sage: (x^2 + 1).is_irreducible()
False
sage: (x^3 + x + 1).is_irreducible()
True
sage: x.degree()
1
sage: x.list()
[0, 1]
sage:
```

En 2007, Martin Albrecht [2] a commencé à développer un module pour le logiciel Sage implémentant l'environnement SR détaillé ci-dessus.
Commençons par travailler avec $SR(1, 1, 1, 4)$, c'est-à-dire avec un chiffrement AES sur 1 tour avec un tableau d'états de 1 colonne et de 1 ligne et un mot de 4 bits.
Le code suivant permet de construire $SR(1, 1, 1, 4)$ et d'afficher le résultat :

```
sage: sr = mq.SR(1, 1, 1, 4)
sage: sr
SR(1,1,1,4)
sage:
```

À partir de là, nous pouvons afficher les variables :

```
sage: print sr.R.repr_long()
Polynomial Ring
  Base Ring : Finite Field in a of size 2^4
       Size : 20 Variables
   Block  0 : Ordering : degrevlex
              Names    : k100, k101, k102, k103, x100, ...
sage:
```

Puis, calculer et afficher le polynôme irréductible :

```
sage: sr.base_ring().polynomial()
a^4 + a + 1
sage:
```

Ou encore, afficher la *S-Box* :

```
sage: sr.sbox()
(6, 11, 5, 4, 2, 14, 7, 10, 9, 13, 15, 12, 3, 1, 0, 8)
sage:
```

Et enfin, calculer le système d'équations polynomiales multivariées :

```
sage: sr.polynomial_system()
(Polynomial System with 40 Polynomials in 20 Variables,
 {k003: a, k002: a^2 + 1, k001: a + 1, k000: a^2})
sage:
```

Comme nous l'avons vu plus haut, dans notre description de l'environnement SR, l'AES-128 est donné par : $SR^*(10, 4, 4, 8)$. En utilisant le module SR de Sage, nous obtenons les données suivantes :
Construction de $SR^*(10, 4, 4, 8)$ et affichage du résultat :

```
sage: sr = mq.SR(10, 4, 4, 8, star=True)
sage: sr
SR*(10,4,4,8)
sage: sr.base_ring()
Finite Field in a of size 2^8
sage:
```

Calcul et affichage du système polynomial correspondant :

```
sage: sr.polynomial_system()

(Polynomial System with 8576 Polynomials in 4288 Variables,
 {k001507: a^7 + a^6 + a,
  k001506: a^6 + a^5 + a + 1,
  k001505: a^5 + a^4 + a^2 + a,
  k001504: a^5 + a^4 + a^3 + 1,
```

```
  k001503: a^5 + a^3 + a^2 + a + 1,
  k001502: a^7 + a^6 + a^4 + a + 1,
  k001501: a^6 + a^5 + a^2 + a,
  k001500: a^5 + a^4 + a^2 + 1,
...
sage:
```

Afin de valider l'implémentation de l'environnement SR^* dans sage, nous allons utiliser le script dont le code est détaillé dans le listing 12.

```
1  def aes_128(round):
2      """ AES: FIPS 197 implementation """
3      aes = mq.SR(round, 4, 4, 8, star=True, allow_zero_inversions=True)
4      print aes
5      print aes.base_ring()
6      print aes.base_ring().polynomial()
7      print aes.sbox_constant()
8      print
9      print '#### key schedule ####'
10     key = '2b7e151628aed2a6abf7158809cf4f3c'
11     print 'key:', key
12     temp = [aes.base_ring().fetch_int(ZZ(key[i:i+2], 16))\\
13         for i in range(0, len(key), 2)]
14     key = aes.state_array(temp)
15     print key
16     print aes.hex_str(key)
17     for r in range(aes.n):
18         key = aes.key_schedule(key, r+1)
19         print aes.hex_str(key)
20     print
21     print '#### AES ciphering process ####'
22     plain = '00112233445566778899aabbccddeeff'
23     key = '000102030405060708090a0b0c0d0e0f'
24     set_verbose(2)
25     cipher = aes(plain, key)
26     set_verbose(0)
```

Listing 12 – Sage : implémentation de l'AES à partir de l'environnement SR^*

La fonction aes_128 prend en paramètre le nombre de tours à effectuer. Pour exécuter un chiffrement AES 128 complet, il faut réaliser 10 tours. La ligne 1 crée l'environnement SR^* et l'affecte à l'objet aes. Les lignes 4 à 7 affichent des informations sur l'objet aes et notamment, son polynôme irréductible et sa S-BOX. Les lignes 9 à 19 affichent les clefs des différents tours issues de la fonction d'expansion de la clef de l'AES. Enfin, les lignes 21 à 26 réalisent le chiffrement d'un vecteur de texte clair avec une clef, tous deux fournis dans [155].

3.3.3 Méthodes de résolution de systèmes d'équations polynomiales

Avant de présenter un cas concret de recouvrement de clefs, nous allons décrire les différentes méthodes de résolution de systèmes d'équations multivariées.

Soient un corps K et un anneau polynomial multivarié $K[x_1, \cdots, x_n]$ à n variables dans K. Soit l'ensemble m des polynômes $f_1, \cdots, f_m$ dans $K[x_1, \cdots, x_n]$. Chercher à trouver les solutions du système d'équations $f_i = 0 \quad (1 \leq i \leq m)$ revient à trouver $(a_1, \cdots, a_n) \in K^n$ tel que

$$f_1(a_1, \cdots, a_n) = \cdots = f_m(a_1, \cdots, a_n) = 0$$

Ce problème revient à trouver la variété algébrique associée avec l'idéal

$$I = \langle f_1, \cdots, f_m \rangle \lhd K[x_1, \cdots, x_n]$$

générée par les polynômes $f_1, \cdots, f_m$. Cette variété $V(I)$ est définie par :

$$V(I) = \{(a_1, \cdots, a_n) \in K^n \quad | \quad f(a_1, \cdots, a_n) = 0 \quad \forall f \in I\}$$

3.3.3.1 La méthode des bases de Gröbner

Les bases de Gröbner sont utilisées entres autres pour résoudre des systèmes d'équa-tions polynomiales multivariées [30].
Soient I un idéal de $k[x_1, \cdots, x_n]$ et $<$ un ordre admissible. Soient $LT(I)$ l'ensemble des termes à coefficient principal de I et $\langle LT(I) \rangle$ l'idéal généré par les éléments de $LT(I)$. Soit $G = \{g_1, \cdots, g_s\}$ alors G est une base de Gröbner de I si et seulement si

$$\langle LT(g_1), \cdots, LT(g_s) \rangle = \langle LT(I) \rangle$$

Tout idéal I de $k[x_1, \cdots, x_n]$ différent de 0 a une base de Gröbner [65].

Algorithme de Buchberger Développé en 1965, l'algorithme de Buchberger est l'un des plus connu pour calculer la base de Gröbner d'un idéal polynomial [31].
Ce dernier s'appuie sur le S-polynôme, ou polynôme de *syzygie*, obtenu à partir d'une paire de polynômes. Ainsi, le S-polynôme des polynômes $f, g \in K[x_1, \cdots, x_n]$ est :

$$S(f, g) = \left(\frac{lcm(LM(f), LM(g))}{LT(f)} \right) \cdot f - \left(\frac{lcm(LM(f), LM(g))}{LT(g)} \right) \cdot g$$

avec *lcm* correspondant au plus petit commun multiple.
Soit $I = \langle f_1, \cdots, f_s \rangle \neq \{0\}$ un idéal polynomial. Alors une base de Gröbner pour I peut être construite en un nombre fini d'étapes par l'algorithme de la figure 3.4 page 51 [65].
L'algorithme de Buchberger utilise donc la réduction polynomiale ce qui est très consommateur en temps et en ressource mémoire. De plus, il implique le calcul de nombreux S-polynômes dont la plupart sont inutiles [203].
La complexité de l'algorithme de Buchberger est liée au degré total des polynômes intermédiaires générés par l'algorithme. Plus précisément, le calcul d'une base de Gröbner pour l'idéal d'un système de polynômes de degrés au plus d génère des polynômes de degré proportionnel à 2^{2^d} [65]. En fait, l'algorithme de Buchberger peut avoir une complexité exponentielle double [50].
Cependant, pour les systèmes d'équations présentés ci-dessus, le degré des polynômes intermédiaires générés par l'algorithme de Buchberger est au plus le nombre total de variables du système d'équations. Ainsi, la complexité de l'algorithme de Buchberger appliqué au système d'équation de l'AES est exponentielle simple [50].

```
function BUCHBERGER(F = (f_1, ··· , f_s))
    G ← F
    repeat
        G' ← G
        for chaque paire de polynômes distincts p et q de G' do
            Calculer le S-polynôme S(p, q)
            Calculer le reste r de la division de S(p, q) par les polynômes de G'
            if r ≠ 0 then
                G ← G ∪ {r}
            end if
        end for
    until G = G'
    return G
end function
```

FIGURE 3.4 – Algorithme de Buchberger

Algorithmes F4 et F5 Les algorithmes F4 [91], présenté en 1999, et F5 [92], présenté en 2002, sont une alternative à l'algorithme de Buchberger pour le calcul de bases de Gröbner.
L'algorithme F4 est une amélioration de l'algorithme de Buchberger dans le sens où il remplace la réduction polynomiale par la réduction matricielle. Ainsi, on ne traite plus une seule paire critique à la fois mais un sous-ensemble de paires. Rendu célèbre parce qu'il a permis la cryptanalyse du challenge HFE [93], l'algorithme F5 fonctionne de manière similaire à l'algorithme F4 en utilisant la réduction matricielle. Il rajoute, en plus, un critère de sélection qui permet d'éliminer les réductions inutiles [203].
La complexité de l'algorithme F5 pour des systèmes d'équations polynomiales multivariées sur $GF(2)$ a été présentée dans [14]. Pour un système de m équations quadratiques à n variables sur $GF(2)$ la complexité de F5 est la suivante :

Condition	**Complexité**
m croît linéairement avec n	Exponentielle en n
$n \ll m \ll n^2$	sous-exponentielle en n
m croît linéairement avec n^2	Polynomiale en n

Pour résoudre le système d'équations de l'AES décrit dans [50], la matrice générée par l'algorithme F5 aurait une taille de 2^{341}. Ce qui signifie que la complexité de l'algorithme F5 pour résoudre ce système d'équations est de l'ordre de $2^{341\omega}$ opérations, avec ω correspondant à l'exposant de la matrice de réduction. L'algorithme F5 n'est donc pas envisageable dans notre cas.

3.3.3.2 La méthode de linéarisation

La linéarisation est une autre technique permettant de résoudre certains systèmes d'équations polynomiales multivariées.
Soit un tel système

$$f_1(x_1, \cdots, x_n) = 0, \cdots, f_m(x_1, \cdots, x_n) = 0$$

sur un corps K et soit $\{f_1, \cdots, f_m\}$ l'ensemble des polynômes dans l'anneau polynomiale $K[x_1, \cdots, x_n]$. Alors, chaque polynôme est une combinaison linéaire de monômes $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} = X^\alpha$ sur K. Nous avons alors :

$$f_i = \sum_{\alpha \in N'} c_\alpha^i X^\alpha$$

avec $c_\alpha^i \in K$ et N' le sous-ensemble fini de l'ensemble des indices multiples K^n. On peut linéariser ce système en considérant les monômes X^α comme de nouvelles variables indépendantes X_α. Nous obtenons alors, un système linéaire qui peut être résolu en construisant la matrice A_L correspondante

$$\begin{array}{c} \\ f_1 \\ \vdots \\ f_m \end{array} \begin{array}{c} \begin{array}{ccc} X_\alpha & \cdots & X'_\alpha \end{array} \\ \begin{pmatrix} c_\alpha^1 & \cdots & c_\alpha'^1 \\ \vdots & \ddots & \vdots \\ c_\alpha^m & \cdots & c_\alpha'^m \end{pmatrix} \end{array}$$

puis en la réduisant à sa forme échelonnée.

L'algorithme XL La méthode de linéarisation ne fonctionne pas s'il n'y a pas suffisamment de polynômes linéairement indépendants. Une solution consiste à étendre le système original afin de générer suffisamment d'équations linéairement indépendantes pour permettre l'application de la méthode de linéarisation.
C'est l'objectif de l'algorithme *extended linearization* ou algorithme XL [186]. L'algorithme XL multiplie les polynômes du système original par tous les monômes d'un degré fixé. Ensuite la méthode de linéarisation, précédemment décrite, peut être appliquée.
Le système d'équations de l'AES sur $GF(2)$ proposé dans [50] compte 8000 équations à 1600 variables. La complexité estimée de l'algorithme XL, donnée dans [186], pour résoudre ce système d'équations est de l'ordre de 2^{330} opérations. Une autre estimation [46] donne un nombres d'opérations pour résoudre ce système d'équations de l'ordre de 2^{681}. L'algorithme XL n'est donc clairement pas envisageable dans notre cas.

L'algorithme XSL Basé sur l'algorithme XL, l'algorithme *extended sparse linearization*, ou algorithme XSL [63, 64], se différencie de son prédécesseur en ne multipliant les polynômes du système que par certains monômes choisis avec soin. L'algorithme XSL ajoute également une étape appelée "méthode T'" par laquelle de nouvelles équations linéairement indépendantes sont créées sans ajouter de nouveaux monômes.
La complexité de l'algorithme XSL pour résoudre le système d'équations décrivant l'AES est également estimée dans [64]. Bien qu'elle s'appuie sur une analyse n'intégrant pas le processus d'extension de la clef de l'AES, la complexité est de l'ordre de 2^{298} opérations. Cependant, cette estimation est trop optimiste. Une évaluation plus complète de l'algorithme XSL est donnée dans [46]. Elle montre que l'algorithme XSL ne peut pas résoudre le système d'équations résultant de l'AES.

3.3.4 Recouvrement de clefs avec SR

L'objectif recherché, en écrivant l'algorithme de l'AES sous la forme d'un système d'équations, est d'arriver à résoudre ce système dans un temps inférieur à la cryptanalyse par force brute. Hors, les systèmes d'équations obtenus par ces différentes approches sont très importants et, nous l'avons vu, les méthodes de résolution de systèmes d'équations polynomiales existant à ce jour, ne permettent pas d'atteindre cet objectif. Même en utilisant des bases de Gröbner comme proposé par Cid, Murphy et Robshaw, les temps de calculs ne permettent pas de recouvrer la clef dans un temps raisonnable.
Afin d'avoir une meilleure idée des temps de calcul nécessaires, nous allons nous appuyer sur l'environnement SR de l'application `sage`. Pour cela, nous avons développé un ensemble de programmes dont celui du listing 13. Celui-ci définit un objet représentant l'AES à l'aide de SR^* et permet de recouvrer la clef utilisée pour chiffrer un message. Nous avons développé et publié sur Internet une généralisation de ce programme [81, SageAES].

```
1  #!/usr/bin/env sage -python
2  # -*- coding: utf-8 -*-
3
4  import time
5  from sage.all import *
6
7  def recoverKey():
8      aes = mq.SR(2, 1, 1, 8, gf2=True, star=True, allow_zero_inversions=True)
9      plain = aes.vector([0, 1, 1, 0, 0, 1, 0, 1])
10     print "plain:", plain._list()
11     key = aes.vector([1, 1, 0, 0, 0, 1, 1, 1])
12     print "key:", key._list()
13     set_verbose(2)
14     cipher = aes(plain, key)
15     set_verbose(0)
16     print "cipher:", cipher._list()
17     t0 = time.time()
18     F, s = aes.polynomial_system(P=plain, C=cipher)
19     print "number of solutions:", len(F.ideal().variety())
20     print F.groebner_basis()
21     for V in F.ideal().variety():
22         tmp = []
23         for key, value in sorted(V.items()):
24             if str(key)[0:2] == "k0":
25                 tmp.append(int(value))
26         result = []
27         for i in range(len(tmp)):
28             result.append(tmp[len(tmp)-1-i])
29         print result
30     print time.time() - t0
31
32 if __name__ == "__main__":
33     recoverKey()
```

Listing 13 – `Sage` : programme de recouvrement de la clef en utilisant l'environnement SR^*

Dans le code 13, la ligne 8 définit un objet représentant une version simplifiée de

l'AES en utilisant l'environnement SR^*. Dans notre cas, nous définissons l'AES dans $GF(2^8)$ avec deux tours et un tableau d'états comprenant une colonne et une rangée. Les lignes 9 à 12 définissent le message en clair et la clef de chiffrement, tous deux sur un octet. La ligne 14 réalise le chiffrement du message. La ligne 17 définit le temps t_0 nécessaire pour déterminer la durée d'exécution du recouvrement de la clef. Les lignes 18 à 30 calculent le système d'équations et cherchent une solution en utilisant une base de Gröbner. Enfin, la ligne 30 affiche le temps mis par l'ordinateur pour recouvrer la clef.
En lançant l'exécution de ce programme, nous obtenons le résultat détaillé dans le listing 14.

```
./sage_aes_recover.py
plain: [0, 1, 1, 0, 0, 1, 0, 1]
key: [1, 1, 0, 0, 0, 1, 1, 1]
R[01].start   A2
R[01].s_box   3A
R[01].s_row   3A
R[01].m_col   3A
R[01].k_sch   C7
R[02].s_box   54
R[02].s_row   54
R[02].k_sch   C4
R[02].output  90
cipher: [1, 0, 0, 1, 0, 0, 0, 0]
number of solutions: 2
Polynomial Sequence with 71 Polynomials in 72 Variables
[1, 0, 1, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 0, 1, 1, 1]
16.6776130199
```

Listing 14 – Sage : recouvrement de la clef en utilisant l'environnement SR^*

Ainsi, en prenant l'algorithme de l'AES réduit à deux tours et à un tableau d'états de seulement une colonne et une rangée, nous obtenons un système de 71 polynômes et 72 variables. Ce système d'équations compte deux solutions dont l'une correspond à la clef utilisée pour chiffrer le message. Le temps de recouvrement de la clef est de 16.6776130199 secondes.
Dans les tableaux 3.5 et 3.6 page 55, nous donnons les résultats obtenus en faisant évoluer le nombre de tours, la taille du tableau d'états et la taille du mot [4].
En analysant les résultats obtenus (voir fig. 3.7 p. 55) nous constatons une corrélation forte entre le nombre de polynômes / variables du système d'équations et le temps de recouvrement de la clef. De même, pour un même nombre de tours et une taille du tableau d'états identique, le temps de recouvrement évolue en fonction de la taille du mot.
Finalement, ce graphe montre la progression exponentielle du temps nécessaire pour résoudre le système d'équations et recouvrer la clef de chiffrement. Pour 150 polynômes, le temps de recouvrement dépasse les 60 heures de calcul. On peut aisément imaginer que, même avec un ordinateur plus puissant, le temps néces-

4. Tous les calculs ont été réalisés sur un macbook pro avec un processeur Intel Core i5 cadencé à 2.5 GHz et 16 Go de RAM.

Tour	Col.	Rangée	$GF(2^n)$	Polynômes	Variables	Temps (s)
1	1	1	$n = 4$	20	20	0.946846961975
2	1	1	$n = 4$	35	36	2.79799604416
3	1	1	$n = 4$	52	52	4.71912503242
4	1	1	$n = 4$	67	68	11.7606241703
1	2	1	$n = 4$	39	40	3.28968000412
2	2	1	$n = 4$	72	72	10.1509840488
3	2	1	$n = 4$	103	104	39.367041111
4	2	1	$n = 4$	135	136	88.8811769485
1	2	2	$n = 4$	72	72	9.33853793144
2	2	2	$n = 4$	127	128	1417.09561396

FIGURE 3.5 – Tableau de résultats des tests de recouvrement pour $n = 4$

Tour	Colonne	Rangée	$GF(2^n)$	Polynômes	Variables	Temps (s)
1	1	1	$n = 8$	39	40	2.93503212929
2	1	1	$n = 8$	71	72	17.8794009686
3	1	1	$n = 8$	103	104	103.00883007
4	1	1	$n = 8$	136	136	464.058065891
1	2	1	$n = 8$	79	80	15.7268610001
2	2	1	$n = 8$	143	144	609.276160955
1	2	2	$n = 8$	144	144	992.330612183

FIGURE 3.6 – Tableau de résultats des tests de recouvrement pour $n = 8$

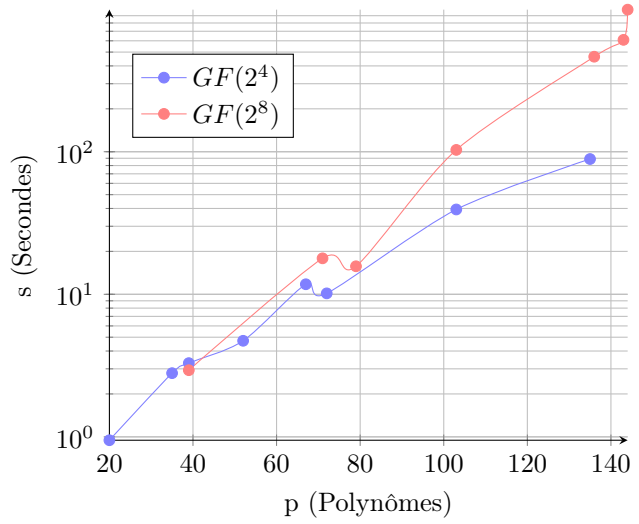


FIGURE 3.7 – Temps de recouvrement en fonction du nombre de polynômes

saire pour résoudre le système d'équations à 8576 polynômes et 4288 variables de l'AES 128 [50] n'est pas raisonnable pour une cryptanalyse opérationnelle. Ces résultats confirment également ce que nous avons détaillé plus haut : l'utilisation des bases de Gröbner est de complexité exponentielle et elle croît en fonction du nombre de polynômes.

—=oOo=—

Décrire l'AES comme un système d'équations quadratiques multivariées est une étape importante pour sa cryptanalyse en utilisant des méthodes algébriques. Cependant, tant que des algorithmes de résolution de grands systèmes d'équations plus efficients n'auront pas été trouvés, cette branche de la cryptanalyse de l'AES semble être une impasse, même si des résultats encourageants existent, notamment avec la cryptanalyse réussie de HFE [93].
En 2002, Nicolas Courtois et Joseph Pieprzyk ont annoncé [64] que l'algorithme XSL appliqué au système BES, décrit ci-dessus, permet de cryptanalyser l'AES avec une complexité estimée à 2^{100}. Par la suite, plusieurs études [46, 44] ont confirmé que cette démarche était une impasse dans le sens où elle n'était pas plus efficace que la cryptanalyse par force brute : "Our conclusion is that if XSL works on BES, then it is worse than brute force" [44].
Malgré tout, une autre voie permettant d'avoir un système plus simple, toujours basée sur ses propriétés algébriques, pourrait consister à écrire l'AES sous la forme d'un système d'équations booléennes construit à partir des tables de vérité de ses fonctions internes.
C'est cette voie que nous allons aborder maintenant.

Deuxième partie

Nouveau jeu d'équations et analyse

Chapitre 4

Génération d'équations booléennes pour l'AES

« Let it further be agreed, that by the combination xy shall be represented that class of things to which the names or descriptions represented by x and y are simultaneously applicable. Thus, if x alone stands for "white things", and y for "sheep", let xy stand for "white sheep" ; and in like manner, if z stand for "horned things", and x and y retain their previous interpretations, let zxy represent "horned white sheep", i.e. that collection of things to which the name "sheep", and the descriptions "white" and "horned" are together applicable. »

George Boole [27]

4.1 Les fonctions booléennes

4.1.1 Définition

Soient l'ensemble $B = \{0, 1\}$ et $\mathcal{B}_2 = \{B, \wedge, \vee, \neg\}$ une algèbre booléenne, alors $\mathcal{B}_2^n = (x_1, x_2, \cdots, x_n)$ tel que $x_i \in \mathcal{B}_2$ et $1 \leqq i \leqq n$, est un sous-ensemble de $\mathcal{B}_2$ contenant tous les n-tuples de 0 et 1. La variable x_i est appelée variable booléenne si elle n'accepte que des valeurs appartenant à B, c'est-à-dire, si et seulement si $x_i = 0$ ou $x_i = 1$ quel que soit $1 \leqq i \leqq n$.

Définition 4.1. Une fonction booléenne de degré n avec $n > 1$ est une fonction f définie de $\mathcal{B}_2^n \to \mathcal{B}_2$, c'est-à-dire construite à partir de variables booléennes et n'acceptant de valeurs de retour que dans l'ensemble $B = \{0, 1\}$.

Par exemple, la fonction $f(x_1, x_2) = x_1 \wedge \neg x_2$ définie de $\mathcal{B}_2^2 \to \mathcal{B}_2$ est une fonction booléenne de degré deux avec :

$$f(0,0) = 0 \tag{4.1}$$
$$f(0,1) = 0 \tag{4.2}$$
$$f(1,0) = 1 \tag{4.3}$$
$$f(1,1) = 0 \tag{4.4}$$

Définition 4.2. Soient n et m deux entiers positifs. Une fonction booléenne vectorielle est une fonction booléenne f définie de $\mathcal{B}_2^n \to \mathcal{B}_2^m$.

À titre d'exemple, la S-box de l'AES est une fonction booléenne vectorielle avec $n = m = 8$.
Enfin, nous pouvons définir une fonction booléenne aléatoire comme étant une fonction booléenne f dont les valeurs sont des variables aléatoires indépendantes et identiquement distribuées, c'est-à-dire :

$$\forall (x_1, x_2, \cdots, x_n) \in \mathcal{B}_2^n, \quad P[f(x_1, x_2, \cdots, x_n) = 0] = \frac{1}{2}$$

Le nombre de fonctions booléennes est limité et dépendant de n. Ainsi, il existe 2^{2^n} fonctions booléennes. De même, le nombre de fonctions booléennes vectorielles est limité et dépendant de n et m. Ainsi, il existe $(2^m)^{2^n}$ fonctions booléennes vectorielles.
Si nous prenons, par exemple, $n = 2$, il existe alors $\left(2^2\right)^2 = 16$ fonctions booléennes de degré deux. Ces 16 fonctions booléennes sont présentées dans le tableau de la figure 4.1 page 61. Parmi les fonctions booléennes de degré 2, les plus connues sont les fonctions OR, AND et XOR (voir fig. 4.3 p. 62), (voir fig. 4.4 p. 62) et (voir fig. 4.2 p. 61). Le programme que nous avons développé pour réaliser ces visualisations est disponible sur Internet [81, VisBool3d]. Il a pour objectif de représenter les fonctions booléennes de degré deux en trois dimensions. La hauteur des colonnes correspond à la valeur obtenue en appliquant la fonction booléenne choisie à l'abscisse et à l'ordonnée correspondante.
Le support $supp(f)$ d'une fonction booléenne est l'ensemble des éléments x tel que $f(x) \neq 0$, le poids de Hamming $wt(f)$ d'une fonction booléenne est le cardinal de son support et nous avons donc :

$$wt(f) = |\{x \in \mathcal{B}_2^n \quad | \quad f(x) = 1\}|$$

Une fonction booléenne est dite équilibrée si $wt(f) = 2^{n-1}$. De même, une fonction booléenne vectorielle $\mathcal{B}_2^n \to \mathcal{B}_2^m$ est dite équilibrée si $wt(f) = 2^{n-m}$ [38].
Par exemple, le support de la fonction $f(x_1, x_2) = x_1 \vee x_2$, correspondant au `OU` logique, est $supp(f) = \{(0,1),(1,0),(1,1)\}$ et son poids est $wt(\mathrm{f}) = 3$.

4.1.2 Représentations

Il existe de multiples représentations des fonctions booléennes [66]. Nous allons nous intéresser à la plus simple – la table de vérité – et à celle que nous utiliserons par la suite – une représentation dans $GF(2)$.

4.1.2.1 La table de vérité

Les différentes valeurs prises par une fonction booléenne peuvent être présentées sous la forme d'un tableau appelé table de vérité. La table de vérité caractérise une fonction booléenne.

f_0	0
f_1	$x_1 \wedge x_2$
f_2	$x_1 \wedge \neg x_2$
f_3	x_1
f_4	$\neg x_1 \wedge x_2$
f_5	x_2
f_6	$x_1 \veebar x_2$
f_7	$x_1 \vee x_2$
f_8	$\neg(x_1 \vee x_2)$
f_9	$\neg(x_1 \veebar x_2)$
f_{10}	$\neg x_2$
f_{11}	$x_1 \vee \neg x_2$
f_{12}	$\neg x_1$
f_{13}	$\neg x_1 \vee x_2$
f_{14}	$\neg(x_1 \wedge x_2)$
f_{15}	1

FIGURE 4.1 – Les 16 fonctions booléennes de degré 2

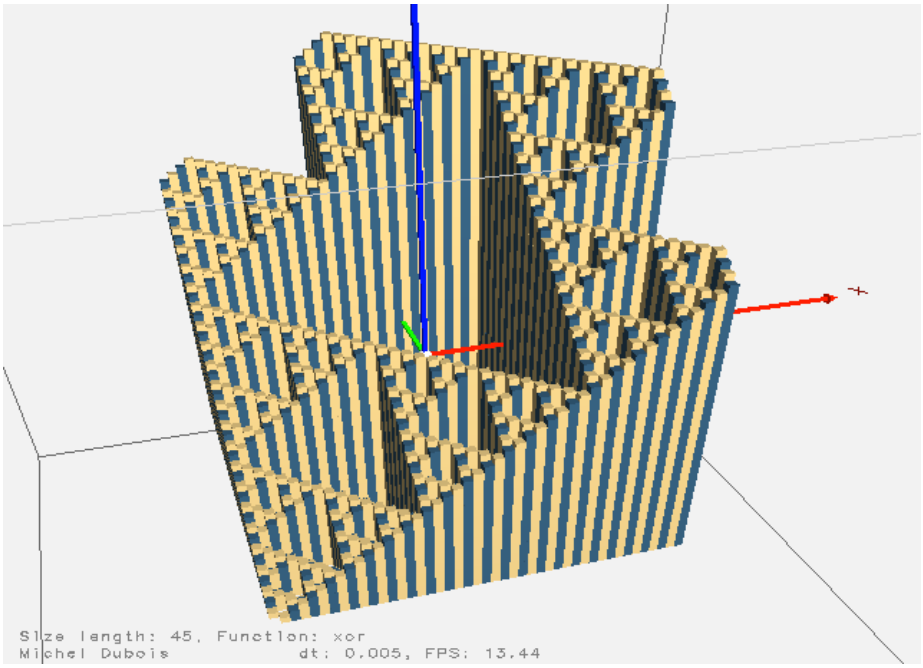


FIGURE 4.2 – La fonction XOR

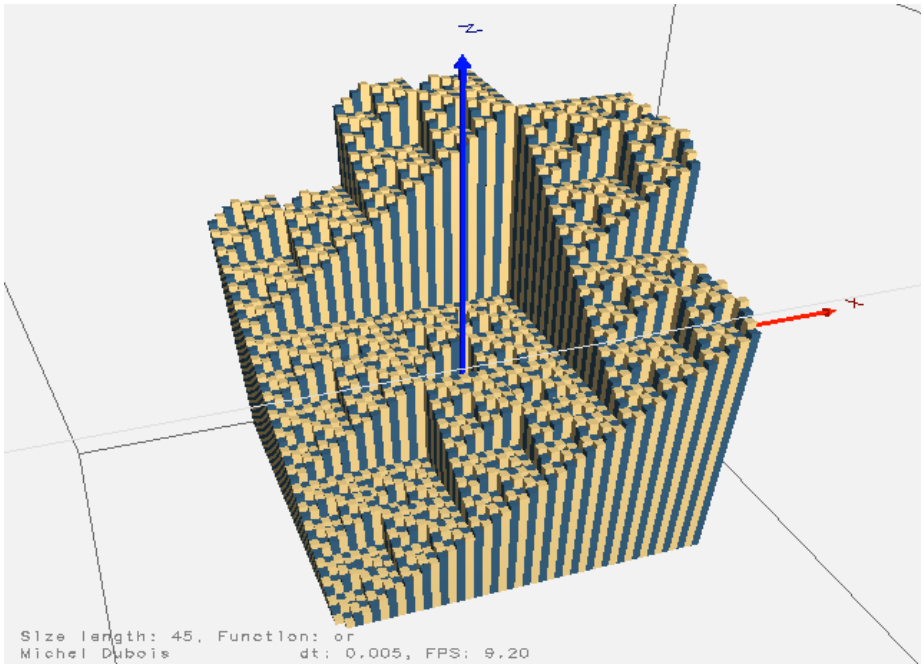


FIGURE 4.3 – La fonction `OR`

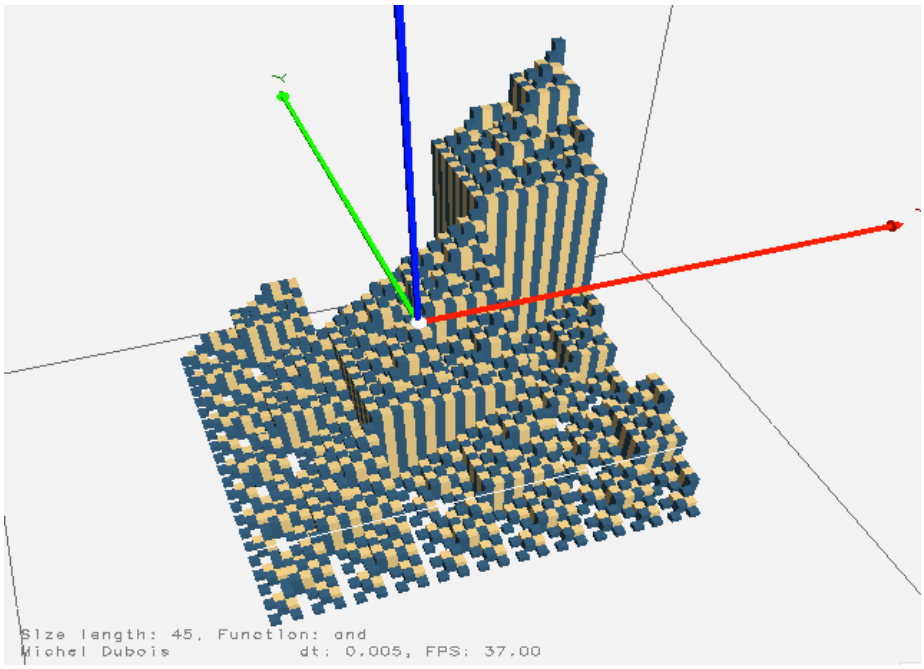


FIGURE 4.4 – La fonction `AND`

À titre d'exemple, le tableau de la figure 4.5 page 63 détaille les tables de vérité des 16 fonctions booléennes de degré deux.

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

x_1	x_2	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

FIGURE 4.5 – Les tables de vérité des 16 fonctions booléennes de degré 2

4.1.2.2 La représentation dans GF(2)

Une fonction booléenne peut également être présentée sous la forme d'une suite de conjonctions comprenant des disjonctions, des négations et/ou des variables. Il s'agit alors de la forme normale conjonctive (CNF). Ainsi, la séquence $f = (a \vee b) \wedge (\neg a \vee b)$ est la forme normale conjonctive de la fonction f. À l'inverse, une fonction booléenne peut être présentée sous la forme d'une suite de disjonctions comprenant des conjonctions, des négations et/ou des variables. Il s'agit alors de la forme normale disjonctive (DNF). Ainsi, la séquence $g = (a \wedge b) \vee (\neg a \wedge b)$ est la forme normale disjonctive de la fonction g. Les CNF et DNF existent et ne sont pas uniques [66].
Intéressons nous maintenant à la représentation des fonctions booléennes dans $GF(2)$.
L'ensemble $B = \{0, 1\}$ associé aux opérations $\wedge$, $\vee$ et $\neg$ est l'algèbre booléenne $\mathcal{B}_2 = \{B, \wedge, \vee, \neg\}$ avec les tables de vérité des opérations décrites dans la figure 4.6 page 63. Si nous introduisons les deux opérations binaires $\oplus$ et $\bullet$ définies par les tables de vérité de la figure 4.7 page 64, alors $\mathcal{B}_2$ et le corps de Galois $GF(2)$ sont assimilables. Plus précisément, l'algèbre de Boole $(B, \wedge, \vee, \neg)$ et le corps $(GF(2), \bullet, \oplus)$ sont liées par les formules de transformation suivantes :

$a \wedge b = a \bullet b$ $\quad$ $a \bullet b = a \wedge b$
$a \vee b = a \oplus b \oplus (a \bullet b)$ $\quad$ $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$
$\neg a = a \oplus 1$

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

a	0	1
$\neg a$	1	0

FIGURE 4.6 – Règles pour l'algèbre de Boole à deux éléments

Nous pouvons maintenant définir une fonction booléenne comme étant une fonction $f : \mathbb{F}_2^n \to \mathbb{F}_2$ avec $\mathbb{F}_2^n$ l'ensemble des vecteurs binaires de longueur $n > 1$.

$\bullet$	0	1
0	0	0
1	0	1

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

FIGURE 4.7 – Tables de vérité de $\bullet$ et $\oplus$

Le poids de Hamming $wH(x)$ du vecteur binaire $x \in \mathbb{F}_2^n$ est le nombre de ses coordonnées non nulles c'est-à-dire la taille de l'ensemble $\{i \in \mathbb{N} \quad | \quad x_i \neq 0\}$. Le poids de Hamming d'une fonction booléenne $f : \mathbb{F}_2^n \to \mathbb{F}_2$ est la taille de son support. Enfin, la distance de Hamming entre deux fonctions booléennes f et g est la taille de l'ensemble $\{x \in \mathbb{F}_2^n \quad | \quad f(x) \neq g(x)\}$.
Parmi les représentation classiques des fonctions booléennes, celle la plus fréquemment utilisée en cryptographie est la représentation polynomiale à n-variables sur $GF(2)$. Cette représentation est de la forme [37] :

$$f(x) = \bigoplus_{I \in P(N)} a_I \left(\prod_{i \in I} x_i \right) = \bigoplus_{I \in P(N)} a_i x^I$$

$P(N)$ désigne l'ensemble des puissances de $N = \{1, \cdots, n\}$. Chaque coordonnée x_i apparaît dans ce polynôme avec un exposant au moins égal à un parce que, dans $\mathbb{F}_2$, nous avons $x^2 = x$. Cette représentation est décrite dans $\mathbb{F}_2[x_1, \cdots, x_n]/(x_1^2 \oplus x_1, \cdots, x_n^2 \oplus x_n)$.
Cette représentation des fonctions booléennes dans $GF(2)$ est appelée expansion de Reed-Muller ou polynômes de Zhegalkin ([152] page 169) ou, plus couramment, *forme normale algébrique* ou *Algebraic Normal Form* (ANF) en anglais. Le degré de $ANF(f)$ correspond au plus haut degré des monômes de $ANF(f)$ à coefficients non nuls. Enfin, la forme normale algébrique d'une fonction booléenne existe et est unique.
En résumé, toute fonction booléenne peut être représentée, de façon unique, par sa forme normale algébrique sous la forme de l'équation :

$$\begin{aligned} f(x_1, \cdots, x_n) = a_0 & \\ & \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n \\ & \oplus a_{1,2} x_1 x_2 \oplus \cdots \oplus a_{n-1,n} x_{n-1} x_n \\ & \oplus \cdots \oplus \\ & \oplus a_{1,2,\ldots,n} x_1 x_2 \ldots x_n \end{aligned}$$

Prenons un exemple. Soit la fonction $f(x_1, x_2, x_3)$ décrite par la table de vérité suivante :

x_1	x_2	x_3	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

La forme algébrique normale de $f(x_1, x_2, x_3)$ est :

$$\begin{aligned} f(x_1, x_2, x_3) = a_0 & \\ & \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_3 \\ & \oplus a_{1,2} x_1 x_2 \oplus a_{1,3} x_1 x_3 \oplus a_{2,3} x_2 x_3 \\ & \oplus a_{1,2,3} x_1 x_2 x_3 \end{aligned}$$

et nous devons déterminer les valeurs des variables a.
Le poids de la fonction f est $wt(f) = 3$. Nous pouvons donc réduire f à la somme de 3 fonctions atomiques f_1, f_2 et f_3. La fonction $f_1 = 1$ si seulement si $1 \oplus x_1 = 1$, $1 \oplus x_2 = 1$ et $x_3 = 1$. De là nous pouvons déduire que l'ANF de la fonction f_1 peut être obtenue par expansion du produit $(1 \oplus x_1)(1 \oplus x_2)x_3$. En appliquant ce raisonnement aux fonctions f_2 et f_3 nous obtenons l'équation suivante :

$$\begin{aligned} ANF(f) &= (1 \oplus x_1)(1 \oplus x_2)x_3 \oplus x_1(1 \oplus x_2)x_3 \oplus x_1 x_2 x_3 \\ &= x_1 x_2 x_3 \oplus x_2 x_3 \oplus x_3 \end{aligned} \tag{4.5}$$

4.2 Mécanisme des équations

Après cette présentation des fonctions booléennes, nous disposons des outils nécessaires à l'élaboration de systèmes d'équations booléennes décrivant l'*Advanced Encryption standard*. Afin d'avancer progressivement nous allons commencer par appliquer notre méthode sur le mini-aes puis nous l'étendrons à l'AES.

4.2.1 Transformée de Möbius

Nous venons de voir comment générer simplement la forme algébrique normale (ANF) d'une fonction booléenne. La méthode présentée n'est pas facilement automatisable dans un programme informatique. Nous allons donc lui préférer l'utilisation de la transformée de Möbius.
La transformée de Möbius de la fonction booléenne f est définie par [141] :

$$\begin{aligned} TM(f) &: \mathbb{F}_2^n \to \mathbb{F}_2 \\ u &= \bigoplus_{v \leqslant u} f(v) \mathrm{mod} 2 \end{aligned}$$

avec $v \leqslant u$ si et seulement si $\forall i, v_i = 1 \Rightarrow u_i = 1$.
De là, nous pouvons définir la forme algébrique normale d'une fonction booléenne f à n variables par :

$$\bigoplus_{u=(u_1,\cdots,u_n)\in\mathbb{F}_2^n} TM(u)x_1^{u_1}\cdots x_n^{u_n}$$

Afin de mieux appréhender les mécanismes mis en œuvre, dans l'utilisation de la transformée de Möbius, prenons un exemple avec la fonction `MajParmi3`. Cette fonction de $\mathbb{F}_2^3 \to \mathbb{F}_2$ est caractérisée par la table de vérité présentée dans la figure 4.8 page 66.

x_1	x_2	x_3	`MajParmi3`
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

FIGURE 4.8 – La table de vérité de la fonction `MajParmi3`

En calculant la transformée de Möbius de la fonction nous obtenons le résultat de la figure 4.9 page 66.

x_1	x_2	x_3	`MajParmi3`	$\to$	calcul de $TM(f)$				$TM(f)$
0	0	0	0	$\to$	0	0	0	0	0
0	0	1	0	$\to$	0	0	0	0	0
0	1	0	0	$\to$	0	0	0	0	0
0	1	1	1	$\to$	1	1	1	1	1
1	0	0	0	$\to$	0	0	0	0	0
1	0	1	1	$\to$	1	1	1	1	1
1	1	0	1	$\to$	1	1	1	1	1
1	1	1	1	$\to$	1	0	1	0	0

FIGURE 4.9 – Calcul de la transformée de Möbius pour `MajParmi3`

Une fois la transformée de Möbius de la fonction obtenue, nous prenons les éléments de $\mathbb{F}_2^3$ pour lesquels $TM(\text{MajParmi3}) \neq 0$. Dans notre cas nous avons les triplets $(0,1,1), (1,0,1), (1,1,0)$ d'où nous pouvons déduire l'équation :

$$\text{MajParmi3}(x_1, x_2, x_3) = x_2x_3 \oplus x_1x_3 \oplus x_1x_2$$

Avec l'addition correspondant à un `XOR` et la multiplication à un `AND`.
La mise en œuvre de la transformée de Möbius en langage Python est réalisée par les deux fonctions décrites dans le listing 15 page 67. Pour calculer la transformée de Möbius, nous utilisons l'algorithme de Cooley-Tukey [56].

```python
def xorTab(t1, t2):
    """Takes two tabs t1 and t2 of same lengths and returns t1 XOR t2."""
    result = ''
    for i in xrange(len(t1)):
        result += str(int(t1[i]) ^ int(t2[i]))
    return result

def moebiusTransform(tab):
    """Takes a tab and return tab[0 : len(tab)/2],
    tab[0 : len(tab)/2] ^ tab[len(tab)/2 : len(tab)].
    usage: moebiusTransform(1010011101010100) --> [1100101110001010]"""
    if len(tab) == 1:
        return tab
    else:
        t1 = tab[0 : len(tab)/2]
        t2 = tab[len(tab)/2 : len(tab)]
        t2 = xorTab(t1, t2)
        t1 = moebiusTransform(t1)
        t2 = moebiusTransform(t2)
        t1 += t2
        return t1
```

Listing 15 – Calcul de la transformée de Möbius en python

4.3 Application au mini-AES

Nous allons maintenant élaborer le système d'équations booléennes décrivant l'AES. En raison de la complexité de ses fonctions internes, nous commençons par appliquer le processus précédemment décrit sur une version simplifiée de l'AES : le mini-AES.

4.3.1 Le mini-AES

C'est avec l'objectif d'aider les étudiants en cryptographie et les cryptanalystes à mieux comprendre les mécanismes internes de l'AES que Raphael Chung-Wei Phan a présenté, en 2002, sa mini version de l'AES [162]. Cette version utilise des paramètres restreints par rapport à l'AES tout en préservant sa structure interne et ses propriétés algébriques.
Le mini-AES est un algorithme de chiffrement par bloc reposant sur les mêmes primitives mathématiques que son grand frère l'AES. Les éléments atomiques avec lequel le mini-AES travaille sont des éléments du corps fini $GF(2^4)$ appelés *nibbles*. Comme l'AES, le mini-AES utilise un tableau d'états contenant quatre nibbles ce qui en fait un algorithme de chiffrement par bloc de 16 bits.
Le processus de chiffrement du mini-AES consiste en deux tours faisant intervenir les fonctions *NibbleSub* appliquant la SBOX au tableau d'états, *ShiftRow* exécutant une rotation des cases du tableau d'états et *MixColumn* multipliant chaque colonne du tableau d'états par une matrice constante. L'architecture des tours est présentée dans la figure 4.10 page 68 et l'implémentation du mini-AES en langage python que nous avons développé est disponible sur Internet [81, BooleanAES].

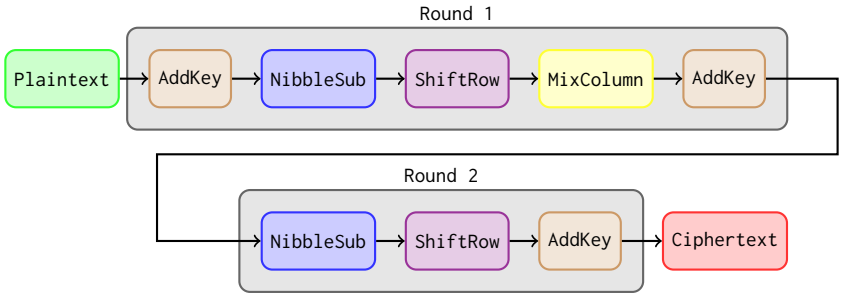


FIGURE 4.10 – Architecture des tours du mini-aes

4.3.2 Les équations pour le mini-AES

En reprenant le principe de génération d'équations énoncé plus haut, nous allons définir les équations pour le mini-AES.
Afin de simplifier le processus nous réduisons le mini-AES à cinq fonctions booléennes de $\mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$, une fonction pour chaque tour et trois fonctions pour la dérivation de la clef.
Les fonctions pour les tours X_1 et X_2 résultent de la conjonction des fonctions `NibbleSub` $NS()$, `ShiftRow` $SR()$ et `MixColumn` $MC()$ telle que, en prenant le bloc de 16 bits $B = (b_1, \cdots, b_{16})$, nous ayons

$$X_1(B) = MC \circ SR \circ NS(B)$$

et

$$X_2(B) = SR \circ NS(B)$$

Les trois fonctions K_1, K_2 et K_3 décrivent le processus de dérivation de la clef tel que, à partir du bloc de clef de 16 bits $K = (k_1, \cdots, k_{16})$, nous ayons les clefs utilisées dans les rounds $k_i = (k_{i,1}, \cdots, k_{i,16})$ avec $i \in (1, 2, 3)$.
Finalement, le mini-AES peut s'écrire sous la forme de deux équations R_1 et R_2, décrivant chacune un tour, telles que :

$$\begin{aligned} B' = R_1(B) =& X_1(B \oplus K_1(K)) \oplus K_2(K) \\ =& (x_{1,1}, \cdots, x_{1,16}) \oplus (k_{2,1}, \cdots, k_{2,16}) \\ =& (b'_1, \cdots, b'_{16}) \\ B'' = R_2(B') =& X_2(B') \oplus K_3(K) \\ =& (x_{2,1}, \cdots, x_{2,16}) \oplus (k_{3,1}, \cdots, k_{3,16}) \\ =& (b''_1, \cdots, b''_{16}) \end{aligned}$$

Avec $x_{1,i} = b_i \oplus k_{1,i} \quad \forall i \in (1, \cdots, 16)$ et B, B', B'' désignant respectivement le bloc de 16 bits en entrée, le bloc de 16 bits à la fin du premier tour et le bloc de 16 bits à la fin du deuxième tour et K le bloc de 16 bits de clef.
Nous pouvons alors calculer les tables de vérité des fonctions booléennes K_1, K_2, K_3, X_1 et X_2 (Cf. figure 4.11 page 69). Puis en utilisant la méthodologie retenue pour la fonction `MajParmi3`, nous obtenons un ensemble de 16 équations pour chaque fonction booléenne, soit une équation pour chaque bit de bloc.

Cependant, du fait que, contrairement à la fonction `MajParmi3`, nous avons deux variables différentes et que les fonctions de chaque tour sont composées, ces dernières seront présentées et traitées séparément dans chaque fichier.

$$
\begin{aligned}
X_1(1026) &= 1000101101011001\\
X_1(1027) &= 0011110001011001\\
X_1(1028) &= 0101100101011001\\
X_1(1029) &= 1100110101011001\\
&\dots\\
X_1(32000) &= 0100001000000111\\
X_1(32001) &= 0011111100000111\\
X_1(32002) &= 0010011100000111\\
X_1(32003) &= 1001000000000111\\
&\dots\\
X_1(65000) &= 1111101100011000\\
X_1(65001) &= 1110001100011000\\
X_1(65002) &= 0101010000011000\\
X_1(65003) &= 0010100100011000
\end{aligned}
$$

FIGURE 4.11 – Quelques équations extraites de la table de vérité de la fonction $X_1()$

Les équations ainsi obtenues pour les fonctions X_1 et X_2 sont détaillées dans l'annexe B page 165.

4.3.3 Mise en forme des équations

Afin d'en faciliter l'analyse et notamment d'y tenter une étude combinatoire nous allons mettre en œuvre une présentation spécifique pour les équations ainsi obtenues.
Le principe retenu consiste à générer un fichier par bit pour les fonctions R_1 et R_2, nous aurons donc finalement 32 fichiers. Le contenu de chacun de ces fichiers est constitué de lignes contenant des suites de 0 et de 1. Chaque ligne décrit un monôme de l'équation et le passage d'une ligne à l'autre signifie l'application d'un `XOR` (cf. figure 4.12 page 70).
Dans le but de faciliter la compréhension du mécanisme retenu nous détaillons la réalisation du fichier correspondant au bit b'_1 de la fonction R_1 dans la figure 4.13 page 71.

4.4 Application à l'AES

Nous allons maintenant appliquer à l'AES le mécanisme décrit ci-dessus pour le mini-AES. La difficulté de ce passage à l'échelle réside dans le fait que, avec le mini-AES, nous avons des fonctions booléennes de $F_2^{16} \to F_2^{16}$ et qu'il est facile

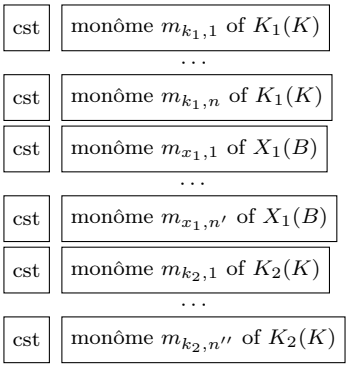


FIGURE 4.12 – Structure du fichier pour la fonction $R_1(b)$

de calculer leurs tables de vérité. Les fonctions de chiffrement et de déchiffrement de l'algorithme de l'AES prennent 128 bits en entrée et fournissent 128 bits en sortie. Nous aurons donc des fonctions booléennes de $F_2^{128} \rightarrow F_2^{128}$ et il est impossible de calculer leurs tables de vérité. En effet, dans ce cas, nous avons $2^{128} = 3,402823 \times 10^{38}$ combinaisons possibles de blocs de 128 bits et l'espace de stockage nécessaire pour archiver ces blocs est de $3,868562 \times 10^{25}$ téraoctets. Nous devons donc trouver une solution pour décrire les fonctions de chiffrement et de déchiffrement de l'AES sous la forme de fonctions booléennes et obtenir ainsi le même résultat que pour l'algorithme du mini-AES.
À l'issue, comme pour le mini-AES, nous devons obtenir 128 fichiers, chacun décrivant la transformation d'un bit de bloc. Le contenu de chacun de ces fichiers consiste en lignes contenant des séquences de 0 et de 1. Chaque ligne décrivant un monôme de la forme normale algébrique des équations booléennes et la transition d'une ligne à une autre correspondant à l'application d'un `XOR`. Comme pour le mini-AES, les variables étant de natures différentes et les fonctions étant combinées, les fonctions de tours seront présentées et traitées par blocs indépendants.

4.4.1 Les équations pour les fonctions de chiffrement

Nous allons maintenant détailler la solution mise en œuvre pour chacune des sous-fonctions de l'algorithme de chiffrement de l'AES.

4.4.1.1 Solution pour la fonction SubBytes

La fonction `SubBytes` est une substitution non-linéaire qui travaille sur chaque octet du tableau d'états en utilisant une table de substitution (S-Box).
Cette fonction est appliquée indépendamment sur chaque octet du bloc d'entrée. Hors, la S-Box de l'AES est une fonction prenant 8 bits en entrée et fournissant 8 bits en sortie. Nous pouvons donc la décrire sous la forme d'une fonction booléenne de $F_2^8 \rightarrow F_2^8$. À partir de là, nous pouvons calculer la table de vérité de la S-Box et utiliser la transformée de Möbius pour obtenir la forme algébrique normale de la S-Box. En appliquant ensuite ces résultats sur les 16 octets du bloc d'entrée, nous obtenons 128 équations, chacune décrivant un bit de bloc.

$b'_1 \rightarrow K_1(K) \rightarrow k_1,$

$$\begin{aligned} X_1(B) \rightarrow & 1 \oplus x_{1,15}x_{1,16} \oplus x_{1,14} \oplus x_{1,14}x_{1,16} \oplus x_{1,13} \oplus x_{1,13}x_{1,15} \\ & \oplus x_{1,13}x_{1,15}x_{1,16} \oplus x_{1,4} \oplus x_{1,3}x_{1,4} \oplus x_{1,2}x_{1,4} \oplus x_{1,2}x_{1,3} \\ & \oplus x_{1,2}x_{1,3}x_{1,4} \oplus x_{1,1}x_{1,3} \oplus x_{1,1}x_{1,3}x_{1,4} \oplus x_{1,1}x_{1,2} \\ & \oplus x_{1,1}x_{1,2}x_{1,3} \end{aligned}$$

$$K_2(K) \rightarrow 1 \oplus k_{16} \oplus k_{14} \oplus k_{14}k_{15} \oplus k_{14}k_{15}k_{16} \oplus k_{13} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{15} \oplus k_1$$

k_1	0	1000000000000000
1	1	0000000000000000
$x_{1,15}x_{1,16}$	0	0000000000000011
$x_{1,14}$	0	0000000000000100
$x_{1,14}x_{1,16}$	0	0000000000000101
$x_{1,13}$	0	0000000000001000
$x_{1,13}x_{1,15}$	0	0000000000001010
$x_{1,13}x_{1,15}x_{1,16}$	0	0000000000001011
$x_{1,4}$	0	0001000000000000
$x_{1,3}x_{1,4}$	0	0011000000000000
$x_{1,2}x_{1,4}$	0	0101000000000000
$x_{1,2}x_{1,3}$	0	0110000000000000
$x_{1,2}x_{1,3}x_{1,4}$	0	0111000000000000
$x_{1,1}x_{1,3}$	0	1010000000000000
$x_{1,1}x_{1,3}x_{1,4}$	0	1011000000000000
$x_{1,1}x_{1,2}$	0	1100000000000000
$x_{1,1}x_{1,2}x_{1,3}$	0	1110000000000000
1	1	0000000000000000
k_{16}	0	0000000000000001
k_{14}	0	0000000000000100
$k_{14}k_{15}$	0	0000000000000110
$k_{14}k_{15}k_{16}$	0	0000000000000111
k_{13}	0	0000000000001000
$k_{13}k_{14}$	0	0000000000001100
$k_{13}k_{14}k_{15}$	0	0000000000001110
k_1	0	1000000000000000

FIGURE 4.13 – Fichier correspondant au bit b'_1

À titre d'exemple, l'équation du traitement du bit b_{127} par la fonction `SubByte` est donnée dans la figure 4.14 page 72.

$$\begin{aligned}\text{SubByte}(b_{127}) = {} & 1 \oplus b_{127} \oplus b_{126}b_{127} \oplus b_{125} \oplus b_{125}b_{126} \oplus b_{124} \oplus b_{124}b_{126} \oplus b_{124}b_{125} \oplus b_{124}b_{125}b_{126} \oplus \\ & b_{124}b_{125}b_{126}b_{127} \oplus b_{123} \oplus b_{123}b_{127} \oplus b_{123}b_{126} \oplus b_{123}b_{126}b_{127} \oplus b_{123}b_{125} \oplus b_{123}b_{125}b_{127} \oplus \\ & b_{123}b_{125}b_{126} \oplus b_{123}b_{124}b_{127} \oplus b_{123}b_{124}b_{126} \oplus b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{122}b_{127} \oplus b_{122}b_{125}b_{127} \oplus \\ & b_{122}b_{125}b_{126}b_{127} \oplus b_{122}b_{124}b_{127} \oplus b_{122}b_{124}b_{125} \oplus b_{122}b_{124}b_{125}b_{127} \oplus b_{122}b_{124}b_{125}b_{126} \oplus \\ & b_{122}b_{123}b_{126}b_{127} \oplus b_{122}b_{123}b_{125}b_{127} \oplus b_{122}b_{123}b_{125}b_{126}b_{127} \oplus b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{127} \oplus \\ & b_{121}b_{126} \oplus b_{121}b_{126}b_{127} \oplus b_{121}b_{125} \oplus b_{121}b_{125}b_{127} \oplus b_{121}b_{125}b_{126} \oplus b_{121}b_{125}b_{126}b_{127} \oplus \\ & b_{121}b_{124}b_{127} \oplus b_{121}b_{124}b_{125}b_{126} \oplus b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{121}b_{123} \oplus b_{121}b_{123}b_{127} \oplus \\ & b_{121}b_{123}b_{126} \oplus b_{121}b_{123}b_{125}b_{126} \oplus b_{121}b_{123}b_{124}b_{127} \oplus b_{121}b_{123}b_{124}b_{126} \oplus b_{121}b_{123}b_{124}b_{126}b_{127} \oplus \\ & b_{121}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{121}b_{122} \oplus b_{121}b_{122}b_{126} \oplus b_{121}b_{122}b_{125} \oplus \\ & b_{121}b_{122}b_{125}b_{127} \oplus b_{121}b_{122}b_{124}b_{127} \oplus b_{121}b_{122}b_{124}b_{126}b_{127} \oplus b_{121}b_{122}b_{124}b_{125}b_{126} \oplus \\ & b_{121}b_{122}b_{123} \oplus b_{121}b_{122}b_{123}b_{127} \oplus b_{121}b_{122}b_{123}b_{126} \oplus b_{121}b_{122}b_{123}b_{126}b_{127} \oplus b_{121}b_{122}b_{123}b_{125} \oplus \\ & b_{121}b_{122}b_{123}b_{125}b_{126} \oplus b_{121}b_{122}b_{123}b_{124}b_{127} \oplus b_{121}b_{122}b_{123}b_{124}b_{125} \oplus b_{121}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus \\ & b_{120}b_{126}b_{127} \oplus b_{120}b_{125} \oplus b_{120}b_{125}b_{127} \oplus b_{120}b_{125}b_{126}b_{127} \oplus b_{120}b_{124}b_{126} \oplus b_{120}b_{124}b_{125} \oplus \\ & b_{120}b_{124}b_{125}b_{127} \oplus b_{120}b_{124}b_{125}b_{126} \oplus b_{120}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{123}b_{127} \oplus b_{120}b_{123}b_{126}b_{127} \oplus \\ & b_{120}b_{123}b_{125} \oplus b_{120}b_{123}b_{125}b_{127} \oplus b_{120}b_{123}b_{125}b_{126} \oplus b_{120}b_{123}b_{125}b_{126}b_{127} \oplus b_{120}b_{123}b_{124} \oplus \\ & b_{120}b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{122} \oplus b_{120}b_{122}b_{125} \oplus b_{120}b_{122}b_{125}b_{127} \oplus \\ & b_{120}b_{122}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{124} \oplus b_{120}b_{122}b_{124}b_{126}b_{127} \oplus b_{120}b_{122}b_{124}b_{125} \oplus \\ & b_{120}b_{122}b_{124}b_{125}b_{126} \oplus b_{120}b_{122}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{127} \oplus b_{120}b_{122}b_{123}b_{126} \oplus \\ & b_{120}b_{122}b_{123}b_{125} \oplus b_{120}b_{122}b_{123}b_{125}b_{127} \oplus b_{120}b_{122}b_{123}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{124}b_{127} \oplus \\ & b_{120}b_{122}b_{123}b_{124}b_{126} \oplus b_{120}b_{122}b_{123}b_{124}b_{125} \oplus b_{120}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{120}b_{121} \oplus \\ & b_{120}b_{121}b_{125} \oplus b_{120}b_{121}b_{125}b_{126} \oplus b_{120}b_{121}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{124} \oplus b_{120}b_{121}b_{124}b_{126} \oplus \\ & b_{120}b_{121}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{124}b_{125} \oplus b_{120}b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{123}b_{127} \oplus \\ & b_{120}b_{121}b_{123}b_{125}b_{127} \oplus b_{120}b_{121}b_{123}b_{125}b_{126} \oplus b_{120}b_{121}b_{123}b_{124}b_{127} \oplus b_{120}b_{121}b_{123}b_{124}b_{126}b_{127} \oplus \\ & b_{120}b_{121}b_{123}b_{124}b_{125}b_{126} \oplus b_{120}b_{121}b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{122} \oplus b_{120}b_{121}b_{122}b_{126} \oplus \\ & b_{120}b_{121}b_{122}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{125}b_{126} \oplus b_{120}b_{121}b_{122}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{124} \oplus \\ & b_{120}b_{121}b_{122}b_{124}b_{127} \oplus b_{120}b_{121}b_{122}b_{124}b_{125}b_{127} \oplus b_{120}b_{121}b_{122}b_{123}b_{126}b_{127} \oplus \\ & b_{120}b_{121}b_{122}b_{123}b_{125} \oplus b_{120}b_{121}b_{122}b_{123}b_{125}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{125}b_{126}b_{127} \oplus \\ & b_{120}b_{121}b_{122}b_{123}b_{124}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{125} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{125}b_{127}\end{aligned}$$

FIGURE 4.14 – Équation du bit b_{127} de la fonction `SubByte`

4.4.1.2 Solution pour la fonction ShiftRows

Dans la fonction `ShiftRows`, les octets de la troisième colonne du tableau d'états sont décalés cycliquement selon un offset dont la taille est dépendante du numéro de la ligne. Les octets de la première ligne ne subissent pas ce décalage.
Pour cette fonction, nous n'avons pas besoin de calculer de fonction booléenne spécifique. En effet, la seule transformation réalisée consiste à décaler des octets dans le tableau d'états. Dans nos fichiers, cette transformation peut aisément être résolue par l'utilisation d'un `XOR`.
Ainsi, à titre d'exemple, le deuxième octet du tableau d'états devient le sixième octet après l'application de `ShiftRows`. Ce qui se traduit par les lignes suivantes :

```
00000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

À titre d'exemple, les équations de la fonction `ShiftRows` pour les bits b_0 à b_{63} sont données dans la figure 4.15 page 73.

$\text{ShiftRows}(b_0) = b_0$	$\text{ShiftRows}(b_{16}) = b_{80}$	$\text{ShiftRows}(b_{32}) = b_{32}$	$\text{ShiftRows}(b_{48}) = b_{112}$
$\text{ShiftRows}(b_1) = b_1$	$\text{ShiftRows}(b_{17}) = b_{81}$	$\text{ShiftRows}(b_{33}) = b_{33}$	$\text{ShiftRows}(b_{49}) = b_{113}$
$\text{ShiftRows}(b_2) = b_2$	$\text{ShiftRows}(b_{18}) = b_{82}$	$\text{ShiftRows}(b_{34}) = b_{34}$	$\text{ShiftRows}(b_{50}) = b_{114}$
$\text{ShiftRows}(b_3) = b_3$	$\text{ShiftRows}(b_{19}) = b_{83}$	$\text{ShiftRows}(b_{35}) = b_{35}$	$\text{ShiftRows}(b_{51}) = b_{115}$
$\text{ShiftRows}(b_4) = b_4$	$\text{ShiftRows}(b_{20}) = b_{84}$	$\text{ShiftRows}(b_{36}) = b_{36}$	$\text{ShiftRows}(b_{52}) = b_{116}$
$\text{ShiftRows}(b_5) = b_5$	$\text{ShiftRows}(b_{21}) = b_{85}$	$\text{ShiftRows}(b_{37}) = b_{37}$	$\text{ShiftRows}(b_{53}) = b_{117}$
$\text{ShiftRows}(b_6) = b_6$	$\text{ShiftRows}(b_{22}) = b_{86}$	$\text{ShiftRows}(b_{38}) = b_{38}$	$\text{ShiftRows}(b_{54}) = b_{118}$
$\text{ShiftRows}(b_7) = b_7$	$\text{ShiftRows}(b_{23}) = b_{87}$	$\text{ShiftRows}(b_{39}) = b_{39}$	$\text{ShiftRows}(b_{55}) = b_{119}$
$\text{ShiftRows}(b_8) = b_{40}$	$\text{ShiftRows}(b_{24}) = b_{120}$	$\text{ShiftRows}(b_{40}) = b_{72}$	$\text{ShiftRows}(b_{56}) = b_{24}$
$\text{ShiftRows}(b_9) = b_{41}$	$\text{ShiftRows}(b_{25}) = b_{121}$	$\text{ShiftRows}(b_{41}) = b_{73}$	$\text{ShiftRows}(b_{57}) = b_{25}$
$\text{ShiftRows}(b_{10}) = b_{42}$	$\text{ShiftRows}(b_{26}) = b_{122}$	$\text{ShiftRows}(b_{42}) = b_{74}$	$\text{ShiftRows}(b_{58}) = b_{26}$
$\text{ShiftRows}(b_{11}) = b_{43}$	$\text{ShiftRows}(b_{27}) = b_{123}$	$\text{ShiftRows}(b_{43}) = b_{75}$	$\text{ShiftRows}(b_{59}) = b_{27}$
$\text{ShiftRows}(b_{12}) = b_{44}$	$\text{ShiftRows}(b_{28}) = b_{124}$	$\text{ShiftRows}(b_{44}) = b_{76}$	$\text{ShiftRows}(b_{60}) = b_{28}$
$\text{ShiftRows}(b_{13}) = b_{45}$	$\text{ShiftRows}(b_{29}) = b_{125}$	$\text{ShiftRows}(b_{45}) = b_{77}$	$\text{ShiftRows}(b_{61}) = b_{29}$
$\text{ShiftRows}(b_{14}) = b_{46}$	$\text{ShiftRows}(b_{30}) = b_{126}$	$\text{ShiftRows}(b_{46}) = b_{78}$	$\text{ShiftRows}(b_{62}) = b_{30}$
$\text{ShiftRows}(b_{15}) = b_{47}$	$\text{ShiftRows}(b_{31}) = b_{127}$	$\text{ShiftRows}(b_{47}) = b_{79}$	$\text{ShiftRows}(b_{63}) = b_{31}$

FIGURE 4.15 – Équations pour les bits b_0 à b_{63} de la fonction `ShiftRows`

4.4.1.3 Solution pour la fonction MixColumns

La fonction `MixColumns` agit sur le tableau d'états, colonne par colonne, traitant chaque colonne comme un polynôme à quatre termes. Chacune de ces colonnes est multipliée par une matrice carrée. Pour chaque colonne nous avons donc :

$$\begin{pmatrix} b'_i \\ b'_{i+1} \\ b'_{i+2} \\ b'_{i+3} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \bullet \begin{pmatrix} b_i \\ b_{i+1} \\ b_{i+2} \\ b_{i+3} \end{pmatrix}$$

Ainsi, pour le premier octet de la colonne nous avons cette équation :

$$b'_i = 02 \bullet b_i \oplus 03 \bullet b_{i+1} \oplus 01 \bullet b_{i+2} \oplus 01 \bullet b_{i+3}$$

Comme dans GF_2^8, 01 est l'identité pour la multiplication, cette équation devient :

$$b'_i = 02 \bullet b_i \oplus 03 \bullet b_{i+1} \oplus b_{i+2} \oplus b_{i+3}$$

Nous avons la même simplification pour toutes les équations décrivant la multiplication de la colonne du tableau d'états par la matrice carrée. Par conséquent nous devons seulement calculer les tables de vérité pour la multiplication par 02 et 03 dans GF_2^8.
À titre d'exemple, les équations des bits b_{120} à b_{127} sont données dans la figure 4.16 page 74.

4.4.1.4 Solution pour la fonction d'expansion de la clef

Pour rappel, dans l'algorithme de l'AES-128, Nb = 4 mots et Nr = 10 mots, avec 1 mot = 4 octets = 32 bits.

$$\begin{aligned}
\text{MixColumns}(b_{120}) &= b_{97} \oplus b_{96} \oplus b_{104} \oplus b_{112} \oplus b_{121} \\
\text{MixColumns}(b_{121}) &= b_{98} \oplus b_{97} \oplus b_{105} \oplus b_{113} \oplus b_{122} \\
\text{MixColumns}(b_{122}) &= b_{99} \oplus b_{98} \oplus b_{106} \oplus b_{114} \oplus b_{123} \\
\text{MixColumns}(b_{123}) &= b_{100} \oplus b_{99} \oplus b_{96} \oplus b_{107} \oplus b_{115} \oplus b_{124} \oplus b_{120} \\
\text{MixColumns}(b_{124}) &= b_{101} \oplus b_{100} \oplus b_{96} \oplus b_{108} \oplus b_{116} \oplus b_{125} \oplus b_{120} \\
\text{MixColumns}(b_{125}) &= b_{102} \oplus b_{101} \oplus b_{109} \oplus b_{117} \oplus b_{126} \\
\text{MixColumns}(b_{126}) &= b_{103} \oplus b_{102} \oplus b_{96} \oplus b_{110} \oplus b_{118} \oplus b_{127} \oplus b_{120} \\
\text{MixColumns}(b_{127}) &= b_{103} \oplus b_{96} \oplus b_{111} \oplus b_{119} \oplus b_{120}
\end{aligned}$$

FIGURE 4.16 – Équations pour les bits b_{120} à b_{127} de la fonction `MixColumns`

La fonction `AddRoundKey` ajoute une clef de tour au tableau d'état par une simple opération de `XOR` bit à bit. Ces clefs de tour sont calculées par une fonction d'expansion de la clef. Cette dernière génère un ensemble de $Nb(Nr + 1) = 44$ mots de 32 bits soit 11 clefs de 128 bits dérivées de la première clef. L'algorithme utilisé pour l'expansion de la clef fait intervenir deux fonctions `SubWord` et `RotWord` ainsi qu'une constante de tour `Rcon`.
La génération d'une fonction booléenne globale d'expansion de la clef est impossible du fait que la génération de la clef du tour n fait intervenir la clef du tour $n-1$. Cette imbrication des clefs de tour ne permet pas de générer une fonction booléenne globale. Par contre il est possible de générer une fonction booléenne correspondant au calcul d'une clef de tour.
Le premier mot w_{i_0} de la clef de tour i est calculé selon l'équation suivant :

$$w_{i_0} = (SW \circ RW(w_{(i-1)_3})) \oplus Rcon_i \oplus w_{(i-1)_0}$$

avec $SW()$ et $RW()$ correspondant respectivement aux fonctions `SubWord` et `RotWord`.
Les mots suivants w_{i_1}, w_{i_2} et w_{i_3} sont calculés selon l'équation suivante :

$$w_{i_n} = w_{i_{n-1}} \oplus w_{(i-1)_n}$$

avec $1 \leq n \leq 3$.
Les fonctions `SubWord` et `RotWord` sont construites sur le même principe que les fonctions `SubBytes` et `ShiftRows`, nous pouvons donc réutiliser la méthodologie précédente.
En langage python, la fonction de génération d'un mot s'écrit selon le code suivant (voir listing 16 p. 75).
Dans ce code, plusieurs cas de figure sont pris en compte.
La fonction `generateWord` prend en paramètre le numéro du mot à générer, nous savons que ce numéro est compris entre 0 et 43. Si le numéro est inférieur à 4, la fonction retourne la fonction booléenne identité puisque la première clef utilisée par l'AES est la clef de chiffrement. Si le numéro modulo 4 est nul, la fonction retourne une fonctionne booléenne décrivant la composition des fonctions `SubWord` et `RotWord` et l'application du `XOR` avec la constante `Rcon`. Enfin si le numéro modulo 4 est non nul, la fonction retourne la fonction booléenne décrivant le `XOR` avec le mot correspondant au tour précédent.

```python
def generateWord(num):
    if (num < 4):
        w = generateGenericWord(wordSize*num, 'x')
    if (num >= 4):
        if ((num % 4) == 0):
            w = generateWord(3)
            w = rotWord(w)
            w = subWord(w, rconList[(num/4)-1])
            w = xorWords(w, generateWord(0))
        else:
            w = generateWord(num-1)
            w = xorWords(w, generateWord(num%4))
    return w
```

Listing 16 – Fonction de génération d'un mot de clef en python

Nous avons maintenant une fonction booléenne décrivant un tour d'expansion de la clef. Comme nous l'avons vu plus haut, l'algorithme d'expansion de la clef fait intervenir au tour n les clefs du tour $n-1$. Pour pouvoir intégrer notre fonction booléenne dans le processus de chiffrement de l'AES, il faut, à chaque tour, ajouter une variable temporaire correspondant à la clef du tour précédent. À titre d'exemple, l'équation booléenne du bit b_0 du quatrième mot sur les 44 mots générés par le processus d'expansion de la clef, est donnée dans la figure 4.17 page 75.

$$w_4(k_0) = k_{109} \oplus k_{109}k_{111} \oplus k_{109}k_{110} \oplus k_{108}k_{109}k_{111} \oplus k_{108}k_{109}k_{110} \oplus k_{108}k_{109}k_{110}k_{111} \oplus k_{107} \oplus k_{107}k_{110}k_{111} \oplus k_{107}k_{109} \oplus k_{107}k_{109}k_{110}k_{111} \oplus k_{107}k_{108}k_{110}k_{111} \oplus k_{107}k_{108}k_{109}k_{110} \oplus k_{107}k_{108}k_{109}k_{110}k_{111} \oplus k_{106} \oplus k_{106}k_{110}k_{111} \oplus k_{106}k_{109}k_{111} \oplus k_{106}k_{109}k_{110}k_{111} \oplus k_{106}k_{108} \oplus k_{106}k_{108}k_{111} \oplus k_{106}k_{108}k_{110} \oplus k_{106}k_{108}k_{109} \oplus k_{106}k_{108}k_{109}k_{111} \oplus k_{106}k_{108}k_{109}k_{110} \oplus k_{106}k_{107}k_{111} \oplus k_{106}k_{107}k_{109}k_{110} \oplus k_{106}k_{107}k_{108} \oplus k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{111} \oplus k_{105}k_{110}k_{111} \oplus k_{105}k_{109} \oplus k_{105}k_{109}k_{110} \oplus k_{105}k_{108}k_{111} \oplus k_{105}k_{108}k_{110} \oplus k_{105}k_{108}k_{110}k_{111} \oplus k_{105}k_{108}k_{109}k_{111} \oplus k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{105}k_{107} \oplus k_{105}k_{107}k_{109} \oplus k_{105}k_{107}k_{109}k_{111} \oplus k_{105}k_{107}k_{109}k_{110} \oplus k_{105}k_{107}k_{109}k_{110}k_{111} \oplus k_{105}k_{107}k_{108}k_{111} \oplus k_{105}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{106}k_{111} \oplus k_{105}k_{106}k_{109} \oplus k_{105}k_{106}k_{108}k_{111} \oplus k_{105}k_{106}k_{108}k_{109}k_{110} \oplus k_{105}k_{106}k_{107} \oplus k_{105}k_{106}k_{107}k_{110}k_{111} \oplus k_{105}k_{106}k_{107}k_{109}k_{110} \oplus k_{105}k_{106}k_{107}k_{108} \oplus k_{105}k_{106}k_{107}k_{108}k_{111} \oplus k_{105}k_{106}k_{107}k_{108}k_{109} \oplus k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104} \oplus k_{104}k_{111} \oplus k_{104}k_{110} \oplus k_{104}k_{109}k_{111} \oplus k_{104}k_{109}k_{110}k_{111} \oplus k_{104}k_{108}k_{111} \oplus k_{104}k_{108}k_{109}k_{111} \oplus k_{104}k_{108}k_{109}k_{110} \oplus k_{104}k_{107}k_{110} \oplus k_{104}k_{107}k_{110}k_{111} \oplus k_{104}k_{107}k_{109}k_{111} \oplus k_{104}k_{107}k_{108}k_{111} \oplus k_{104}k_{107}k_{108}k_{110} \oplus k_{104}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{107}k_{108}k_{109} \oplus k_{104}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{106} \oplus k_{104}k_{106}k_{109}k_{110}k_{111} \oplus k_{104}k_{106}k_{108} \oplus k_{104}k_{106}k_{108}k_{111} \oplus k_{104}k_{106}k_{107} \oplus k_{104}k_{106}k_{107}k_{110} \oplus k_{104}k_{106}k_{107}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{109}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{111} \oplus k_{104}k_{105}k_{109} \oplus k_{104}k_{105}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{108}k_{111} \oplus k_{104}k_{105}k_{108}k_{110} \oplus k_{104}k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{107} \oplus k_{104}k_{105}k_{107}k_{111} \oplus k_{104}k_{105}k_{107}k_{110} \oplus k_{104}k_{105}k_{107}k_{109} \oplus k_{104}k_{105}k_{107}k_{109}k_{110} \oplus k_{104}k_{105}k_{107}k_{108}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{110} \oplus k_{104}k_{105}k_{106}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{109} \oplus k_{104}k_{105}k_{106}k_{109}k_{110} \oplus k_{104}k_{105}k_{106}k_{108}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{107} \oplus k_{104}k_{105}k_{106}k_{107}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_0$$

FIGURE 4.17 – Équation du bit de clef b_0 du $4^{\text{ème}}$ mot

4.4.1.5 Solution globale

Nous avons maintenant une fonction booléenne pour chacune des fonctions `SubBytes` $SB()$, `ShiftRows` $SR()$ et `MixColumns` $MC()$. Hors, dans l'agencement d'un tour, ces fonctions sont combinées. Ainsi, pour un bloc de 128 bits $B = (b_1, \cdots, b_{128})$ en sortie de la fonction `AddRoundKey`, le bloc $B' = (b'_1, \cdots, b'_{128})$ en sortie de la combinaison de ces trois fonctions est tel que :

$$B' = MC \circ SR \circ SB(B)$$

Pour pouvoir réaliser le même type de fichiers que ceux mis en œuvre pour le mini-AES, il est nécessaire de réduire la composition de ces trois fonctions en une seule équation booléenne. Pour y parvenir, il suffit de remplacer chaque variable en entrée d'une fonction par sa valeur en sortie de la fonction précédente selon l'équation suivante :

$$b'_i = MC(SR(SB(b_i))) \quad \forall i \in (1, \cdots, 128)$$

En langage python, la fonction de génération d'un tour s'écrit selon le code suivant (voir listing 17 p. 76).

```
1  def writeRoundEnc(numRound, equaSB, equaSR, equaMC):
2      printColor('## Round%s' % numRound, GREEN)
3      resultSR = []
4      resultMC = []
5      for i in xrange(blockSize):
6          equaSR[i] = equaSR[i].split('_')
7          resultSR.append(equaSB[int(equaSR[i][1])])
8
9      for i in xrange(blockSize):
10         tmp = ''
11         for monomial in equaMC[i].split('+'):
12             tmp += resultSR[int(monomial.split('_')[1])]
13             tmp += '+'
14         resultMC.append(tmp.rstrip('+'))
15     binMon = generateBinaryMonomes(resultMC)
16     return resultMC
```

Listing 17 – Calcul de l'équation pour une fonction de tour chiffrement

L'équation booléenne d'un tour de l'AES pour le bit b_0 est donnée dans la figure 4.18 page 77.
Finalement, nous pouvons maintenant décrire le processus intégral de chiffrement de l'AES sous la forme d'équations booléennes. La fonction en langage python calculant ce processus est donnée dans le listing 18 page 78.

4.4.2 Les équations pour les fonctions de déchiffrement

Nous allons maintenant détailler la solution mise en œuvre pour chacune des sous-fonctions de l'algorithme de déchiffrement de l'AES.

$$
\begin{aligned}
b'_0 = {} & 1 \oplus b_4 \oplus b_4b_6 \oplus b_4b_6b_7 \oplus b_4b_5 \oplus b_3b_7 \oplus b_3b_6b_7 \oplus b_3b_5b_7 \oplus b_3b_5b_6b_7 \oplus b_3b_4b_6 \oplus b_3b_4b_6b_7 \oplus b_3b_4b_5 \oplus \\
& b_3b_4b_5b_6 \oplus b_2 \oplus b_2b_7 \oplus b_2b_6b_7 \oplus b_2b_5b_7 \oplus b_2b_5b_6 \oplus b_2b_5b_6b_7 \oplus b_2b_4 \oplus b_2b_4b_7 \oplus b_2b_4b_5b_7 \oplus b_2b_4b_5b_6 \oplus \\
& b_2b_4b_5b_6b_7 \oplus b_2b_3b_7 \oplus b_2b_3b_5b_7 \oplus b_2b_3b_5b_6 \oplus b_2b_3b_4b_6b_7 \oplus b_2b_3b_4b_5 \oplus b_2b_3b_4b_5b_7 \oplus b_2b_3b_4b_5b_6 \oplus b_1 \oplus \\
& b_1b_5b_7 \oplus b_1b_5b_6 \oplus b_1b_4b_7 \oplus b_1b_4b_6 \oplus b_1b_4b_6b_7 \oplus b_1b_4b_5b_7 \oplus b_1b_4b_5b_6 \oplus b_1b_4b_5b_6b_7 \oplus b_1b_3 \oplus b_1b_3b_7 \oplus \\
& b_1b_3b_6 \oplus b_1b_3b_5 \oplus b_1b_3b_5b_7 \oplus b_1b_3b_4 \oplus b_1b_3b_4b_7 \oplus b_1b_3b_4b_6 \oplus b_1b_3b_4b_5 \oplus b_1b_2b_7 \oplus b_1b_2b_6 \oplus b_1b_2b_4b_5 \oplus \\
& b_1b_2b_4b_5b_7 \oplus b_1b_2b_4b_5b_6b_7 \oplus b_1b_2b_3 \oplus b_1b_2b_3b_5b_6 \oplus b_1b_2b_3b_5b_6b_7 \oplus b_1b_2b_3b_4b_6 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_0b_7 \oplus \\
& b_0b_6 \oplus b_0b_5b_6 \oplus b_0b_4 \oplus b_0b_4b_6b_7 \oplus b_0b_4b_5 \oplus b_0b_4b_5b_7 \oplus b_0b_4b_5b_6b_7 \oplus b_0b_3b_7 \oplus b_0b_3b_6 \oplus b_0b_3b_5 \oplus b_0b_3b_5b_6 \oplus \\
& b_0b_3b_4b_7 \oplus b_0b_3b_4b_6 \oplus b_0b_3b_4b_5b_6 \oplus b_0b_3b_4b_5b_6b_7 \oplus b_0b_2 \oplus b_0b_2b_7 \oplus b_0b_2b_5b_7 \oplus b_0b_2b_5b_6b_7 \oplus b_0b_2b_4 \oplus b_0b_2b_4b_6 \oplus \\
& b_0b_2b_4b_5 \oplus b_0b_2b_4b_5b_6 \oplus b_0b_2b_3b_6 \oplus b_0b_2b_3b_6b_7 \oplus b_0b_2b_3b_5b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_2b_3b_4b_6 \oplus b_0b_2b_3b_4b_6b_7 \oplus \\
& b_0b_1b_6 \oplus b_0b_1b_6b_7 \oplus b_0b_1b_4 \oplus b_0b_1b_4b_6b_7 \oplus b_0b_1b_3b_7 \oplus b_0b_1b_3b_6 \oplus b_0b_1b_3b_5b_7 \oplus b_0b_1b_3b_5b_6b_7 \oplus b_0b_1b_3b_4b_7 \oplus \\
& b_0b_1b_3b_4b_6b_7 \oplus b_0b_1b_3b_4b_5 \oplus b_0b_1b_2 \oplus b_0b_1b_2b_5b_7 \oplus b_0b_1b_2b_5b_6 \oplus b_0b_1b_2b_5b_6b_7 \oplus b_0b_1b_2b_4b_6b_7 \oplus b_0b_1b_2b_4b_5 \oplus \\
& b_0b_1b_2b_3 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_4 \oplus b_0b_1b_2b_3b_4b_6 \oplus b_0b_1b_2b_3b_4b_6b_7 \oplus 1 \oplus \\
& b_{44} \oplus b_{44}b_{46} \oplus b_{44}b_{46}b_{47} \oplus b_{44}b_{45} \oplus b_{43}b_{47} \oplus b_{43}b_{46}b_{47} \oplus b_{43}b_{45}b_{47} \oplus b_{43}b_{45}b_{46}b_{47} \oplus b_{43}b_{44}b_{46} \oplus \\
& b_{43}b_{44}b_{46}b_{47} \oplus b_{43}b_{44}b_{45} \oplus b_{43}b_{44}b_{45}b_{46} \oplus b_{42} \oplus b_{42}b_{47} \oplus b_{42}b_{46}b_{47} \oplus b_{42}b_{45}b_{47} \oplus b_{42}b_{45}b_{46} \oplus b_{42}b_{45}b_{46}b_{47} \oplus \\
& b_{42}b_{44} \oplus b_{42}b_{44}b_{47} \oplus b_{42}b_{44}b_{45}b_{47} \oplus b_{42}b_{44}b_{45}b_{46} \oplus b_{42}b_{44}b_{45}b_{46}b_{47} \oplus b_{42}b_{43}b_{47} \oplus b_{42}b_{43}b_{45}b_{47} \oplus \\
& b_{42}b_{43}b_{45}b_{46} \oplus b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{42}b_{43}b_{44}b_{45} \oplus b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{42}b_{43}b_{44}b_{45}b_{46} \oplus b_{41} \oplus b_{41}b_{45}b_{47} \oplus \\
& b_{41}b_{45}b_{46} \oplus b_{41}b_{44}b_{47} \oplus b_{41}b_{44}b_{46} \oplus b_{41}b_{44}b_{46}b_{47} \oplus b_{41}b_{44}b_{45}b_{47} \oplus b_{41}b_{44}b_{45}b_{46} \oplus b_{41}b_{44}b_{45}b_{46}b_{47} \oplus \\
& b_{41}b_{43} \oplus b_{41}b_{43}b_{47} \oplus b_{41}b_{43}b_{46} \oplus b_{41}b_{43}b_{45} \oplus b_{41}b_{43}b_{45}b_{47} \oplus b_{41}b_{43}b_{44} \oplus b_{41}b_{43}b_{44}b_{47} \oplus b_{41}b_{43}b_{44}b_{46} \oplus \\
& b_{41}b_{43}b_{44}b_{45} \oplus b_{41}b_{42}b_{47} \oplus b_{41}b_{42}b_{46} \oplus b_{41}b_{42}b_{44}b_{45} \oplus b_{41}b_{42}b_{44}b_{45}b_{47} \oplus b_{41}b_{42}b_{44}b_{46}b_{47} \oplus b_{41}b_{42}b_{43} \oplus \\
& b_{41}b_{42}b_{43}b_{45}b_{46} \oplus b_{41}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{45} \oplus \\
& b_{41}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{45}b_{46} \oplus b_{41}b_{42}b_{43}b_{44}b_{45}b_{46}b_{47} \oplus b_{40} \oplus b_{40}b_{47} \oplus b_{40}b_{46} \oplus b_{40}b_{46}b_{47} \oplus \\
& b_{40}b_{45} \oplus b_{40}b_{45}b_{47} \oplus b_{40}b_{45}b_{46} \oplus b_{40}b_{45}b_{46}b_{47} \oplus b_{40}b_{44} \oplus b_{40}b_{44}b_{47} \oplus b_{40}b_{44}b_{46} \oplus b_{40}b_{44}b_{46}b_{47} \oplus b_{40}b_{44}b_{45} \oplus \\
& b_{40}b_{44}b_{45}b_{47} \oplus b_{40}b_{44}b_{45}b_{46} \oplus b_{40}b_{44}b_{45}b_{46}b_{47} \oplus b_{40}b_{43} \oplus b_{40}b_{43}b_{47} \oplus b_{40}b_{43}b_{46} \oplus b_{40}b_{43}b_{46}b_{47} \oplus \\
& b_{40}b_{43}b_{45} \oplus b_{40}b_{43}b_{45}b_{46} \oplus b_{40}b_{43}b_{44}b_{47} \oplus b_{40}b_{43}b_{44}b_{46} \oplus b_{40}b_{43}b_{44}b_{45}b_{46} \oplus b_{40}b_{43}b_{44}b_{45}b_{46}b_{47} \oplus b_{40}b_{42} \oplus \\
& b_{40}b_{42}b_{47} \oplus b_{40}b_{42}b_{45}b_{47} \oplus b_{40}b_{42}b_{44} \oplus b_{40}b_{42}b_{44}b_{46} \oplus b_{40}b_{42}b_{44}b_{45} \oplus b_{40}b_{42}b_{44}b_{45}b_{46} \oplus b_{40}b_{42}b_{43}b_{46} \oplus \\
& b_{40}b_{42}b_{43}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{45}b_{47} \oplus b_{40}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{44}b_{46} \oplus b_{40}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus \\
& b_{40}b_{41}b_{46} \oplus b_{40}b_{41}b_{46}b_{47} \oplus b_{40}b_{41}b_{44} \oplus b_{40}b_{41}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{47} \oplus b_{40}b_{41}b_{43}b_{46} \oplus b_{40}b_{41}b_{43}b_{45}b_{47} \oplus \\
& b_{40}b_{41}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{45} \oplus b_{40}b_{41}b_{42} \oplus \\
& b_{40}b_{41}b_{42}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{45}b_{46} \oplus b_{40}b_{41}b_{42}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{44}b_{45} \oplus \\
& b_{40}b_{41}b_{42}b_{43} \oplus b_{40}b_{41}b_{42}b_{43}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{44} \oplus \\
& b_{40}b_{41}b_{42}b_{43}b_{44}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{45} \oplus b_{45}b_{47} \oplus b_{45}b_{46} \oplus b_{44}b_{45}b_{47} \oplus b_{44}b_{45}b_{46} \oplus b_{44}b_{45}b_{46}b_{47} \oplus \\
& b_{43} \oplus b_{43}b_{46}b_{47} \oplus b_{43}b_{45} \oplus b_{43}b_{45}b_{46}b_{47} \oplus b_{43}b_{44}b_{46}b_{47} \oplus b_{43}b_{44}b_{45}b_{46} \oplus b_{43}b_{44}b_{45}b_{46}b_{47} \oplus b_{42} \oplus b_{42}b_{46}b_{47} \oplus \\
& b_{42}b_{45}b_{47} \oplus b_{42}b_{45}b_{46}b_{47} \oplus b_{42}b_{44} \oplus b_{42}b_{44}b_{47} \oplus b_{42}b_{44}b_{46} \oplus b_{42}b_{44}b_{45} \oplus b_{42}b_{44}b_{45}b_{47} \oplus b_{42}b_{44}b_{45}b_{46} \oplus \\
& b_{42}b_{43}b_{47} \oplus b_{42}b_{43}b_{45}b_{46} \oplus b_{42}b_{43}b_{44} \oplus b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{41} \oplus b_{41}b_{46}b_{47} \oplus b_{41}b_{45} \oplus \\
& b_{41}b_{45}b_{46} \oplus b_{41}b_{44}b_{47} \oplus b_{41}b_{44}b_{46} \oplus b_{41}b_{44}b_{46}b_{47} \oplus b_{41}b_{44}b_{45}b_{47} \oplus b_{41}b_{44}b_{45}b_{46}b_{47} \oplus b_{41}b_{43} \oplus b_{41}b_{43}b_{45} \oplus \\
& b_{41}b_{43}b_{45}b_{47} \oplus b_{41}b_{43}b_{45}b_{46} \oplus b_{41}b_{43}b_{45}b_{46}b_{47} \oplus b_{41}b_{43}b_{44}b_{47} \oplus b_{41}b_{43}b_{44}b_{45}b_{47} \oplus b_{41}b_{42}b_{47} \oplus b_{41}b_{42}b_{45} \oplus \\
& b_{41}b_{42}b_{44}b_{47} \oplus b_{41}b_{42}b_{44}b_{45}b_{46} \oplus b_{41}b_{42}b_{43} \oplus b_{41}b_{42}b_{43}b_{46}b_{47} \oplus b_{41}b_{42}b_{43}b_{45}b_{46} \oplus b_{41}b_{42}b_{43}b_{44} \oplus \\
& b_{41}b_{42}b_{43}b_{44}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{45} \oplus b_{41}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{40} \oplus b_{40}b_{47} \oplus b_{40}b_{46} \oplus b_{40}b_{45}b_{47} \oplus b_{40}b_{45}b_{46}b_{47} \oplus \\
& b_{40}b_{44}b_{47} \oplus b_{40}b_{44}b_{45}b_{47} \oplus b_{40}b_{44}b_{45}b_{46} \oplus b_{40}b_{43}b_{46} \oplus b_{40}b_{43}b_{46}b_{47} \oplus b_{40}b_{43}b_{45}b_{47} \oplus b_{40}b_{43}b_{44}b_{47} \oplus \\
& b_{40}b_{43}b_{44}b_{46} \oplus b_{40}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{43}b_{44}b_{45} \oplus b_{40}b_{43}b_{44}b_{45}b_{47} \oplus b_{40}b_{42} \oplus b_{40}b_{42}b_{45}b_{46}b_{47} \oplus b_{40}b_{42}b_{44} \oplus \\
& b_{40}b_{42}b_{44}b_{47} \oplus b_{40}b_{42}b_{43} \oplus b_{40}b_{42}b_{43}b_{46} \oplus b_{40}b_{42}b_{43}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus \\
& b_{40}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{40}b_{41}b_{47} \oplus b_{40}b_{41}b_{45} \oplus b_{40}b_{41}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{44}b_{47} \oplus b_{40}b_{41}b_{44}b_{46} \oplus \\
& b_{40}b_{41}b_{44}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{43} \oplus b_{40}b_{41}b_{43}b_{47} \oplus b_{40}b_{41}b_{43}b_{46} \oplus b_{40}b_{41}b_{43}b_{45} \oplus b_{40}b_{41}b_{43}b_{45}b_{46} \oplus \\
& b_{40}b_{41}b_{43}b_{44}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{46} \oplus b_{40}b_{41}b_{42}b_{46}b_{47} \oplus \\
& b_{40}b_{41}b_{42}b_{45} \oplus b_{40}b_{41}b_{42}b_{45}b_{46} \oplus b_{40}b_{41}b_{42}b_{44}b_{47} \oplus b_{40}b_{41}b_{42}b_{44}b_{46} \oplus b_{40}b_{41}b_{42}b_{44}b_{46}b_{47} \oplus \\
& b_{40}b_{41}b_{42}b_{44}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{43} \oplus b_{40}b_{41}b_{42}b_{43}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{44} \oplus \\
& b_{40}b_{41}b_{42}b_{43}b_{44}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{85} \oplus b_{85}b_{87} \oplus b_{85}b_{86} \oplus b_{84}b_{85}b_{87} \oplus \\
& b_{84}b_{85}b_{86} \oplus b_{84}b_{85}b_{86}b_{87} \oplus b_{83} \oplus b_{83}b_{86}b_{87} \oplus b_{83}b_{85} \oplus b_{83}b_{85}b_{86}b_{87} \oplus b_{83}b_{84}b_{86}b_{87} \oplus b_{83}b_{84}b_{85}b_{86} \oplus \\
& b_{83}b_{84}b_{85}b_{86}b_{87} \oplus b_{82} \oplus b_{82}b_{86}b_{87} \oplus b_{82}b_{85}b_{87} \oplus b_{82}b_{85}b_{86}b_{87} \oplus b_{82}b_{84} \oplus b_{82}b_{84}b_{87} \oplus b_{82}b_{84}b_{86} \oplus b_{82}b_{84}b_{85} \oplus \\
& b_{82}b_{84}b_{85}b_{87} \oplus b_{82}b_{84}b_{85}b_{86} \oplus b_{82}b_{83}b_{87} \oplus b_{82}b_{83}b_{85}b_{86} \oplus b_{82}b_{83}b_{84} \oplus b_{82}b_{83}b_{84}b_{86}b_{87} \oplus b_{82}b_{83}b_{84}b_{85}b_{87} \oplus \\
& b_{81}b_{87} \oplus b_{81}b_{86}b_{87} \oplus b_{81}b_{85} \oplus b_{81}b_{85}b_{86} \oplus b_{81}b_{84}b_{87} \oplus b_{81}b_{84}b_{86} \oplus b_{81}b_{84}b_{86}b_{87} \oplus b_{81}b_{84}b_{85}b_{87} \oplus \\
& b_{81}b_{84}b_{85}b_{86}b_{87} \oplus b_{81}b_{83} \oplus b_{81}b_{83}b_{85} \oplus b_{81}b_{83}b_{85}b_{87} \oplus b_{81}b_{83}b_{85}b_{86} \oplus b_{81}b_{83}b_{85}b_{86}b_{87} \oplus b_{81}b_{83}b_{84}b_{87} \oplus \\
& b_{81}b_{83}b_{84}b_{85}b_{87} \oplus b_{81}b_{82}b_{87} \oplus b_{81}b_{82}b_{85} \oplus b_{81}b_{82}b_{84}b_{87} \oplus b_{81}b_{82}b_{84}b_{85}b_{86} \oplus b_{81}b_{82}b_{83} \oplus b_{81}b_{82}b_{83}b_{86}b_{87} \oplus \\
& b_{81}b_{82}b_{83}b_{85}b_{86} \oplus b_{81}b_{82}b_{83}b_{84} \oplus b_{81}b_{82}b_{83}b_{84}b_{87} \oplus b_{81}b_{82}b_{83}b_{84}b_{85} \oplus b_{81}b_{82}b_{83}b_{84}b_{85}b_{87} \oplus b_{80} \oplus b_{80}b_{87} \oplus \\
& b_{80}b_{86} \oplus b_{80}b_{85}b_{87} \oplus b_{80}b_{85}b_{86}b_{87} \oplus b_{80}b_{84}b_{87} \oplus b_{80}b_{84}b_{85}b_{87} \oplus b_{80}b_{84}b_{85}b_{86} \oplus b_{80}b_{83}b_{86} \oplus b_{80}b_{83}b_{86}b_{87} \oplus \\
& b_{80}b_{83}b_{85}b_{87} \oplus b_{80}b_{83}b_{84}b_{87} \oplus b_{80}b_{83}b_{84}b_{86} \oplus b_{80}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{83}b_{84}b_{85} \oplus b_{80}b_{83}b_{84}b_{85}b_{87} \oplus \\
& b_{80}b_{82} \oplus b_{80}b_{82}b_{85}b_{86}b_{87} \oplus b_{80}b_{82}b_{84} \oplus b_{80}b_{82}b_{84}b_{87} \oplus b_{80}b_{82}b_{83} \oplus b_{80}b_{82}b_{83}b_{86} \oplus b_{80}b_{82}b_{83}b_{86}b_{87} \oplus \\
& b_{80}b_{82}b_{83}b_{85}b_{86}b_{87} \oplus b_{80}b_{82}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{82}b_{83}b_{84}b_{85}b_{87} \oplus b_{80}b_{81}b_{87} \oplus b_{80}b_{81}b_{85} \oplus b_{80}b_{81}b_{85}b_{86}b_{87} \oplus \\
& b_{80}b_{81}b_{84}b_{87} \oplus b_{80}b_{81}b_{84}b_{86} \oplus b_{80}b_{81}b_{84}b_{85}b_{86}b_{87} \oplus b_{80}b_{81}b_{83} \oplus b_{80}b_{81}b_{83}b_{87} \oplus b_{80}b_{81}b_{83}b_{86} \oplus \\
& b_{80}b_{81}b_{83}b_{85} \oplus b_{80}b_{81}b_{83}b_{85}b_{86} \oplus b_{80}b_{81}b_{83}b_{84}b_{87} \oplus b_{80}b_{81}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{81}b_{83}b_{84}b_{85}b_{87} \oplus \\
& b_{80}b_{81}b_{82}b_{86} \oplus b_{80}b_{81}b_{82}b_{86}b_{87} \oplus b_{80}b_{81}b_{82}b_{85} \oplus b_{80}b_{81}b_{82}b_{85}b_{86} \oplus b_{80}b_{81}b_{82}b_{84}b_{87} \oplus b_{80}b_{81}b_{82}b_{84}b_{86} \oplus \\
& b_{80}b_{81}b_{82}b_{84}b_{86}b_{87} \oplus b_{80}b_{81}b_{82}b_{84}b_{85}b_{87} \oplus b_{80}b_{81}b_{82}b_{83} \oplus b_{80}b_{81}b_{82}b_{83}b_{86} \oplus b_{80}b_{81}b_{82}b_{83}b_{85}b_{87} \oplus \\
& b_{80}b_{81}b_{82}b_{83}b_{84} \oplus b_{80}b_{81}b_{82}b_{83}b_{84}b_{86} \oplus b_{80}b_{81}b_{82}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{81}b_{82}b_{83}b_{84}b_{85}b_{87} \oplus b_{125} \oplus \\
& b_{125}b_{127} \oplus b_{125}b_{126} \oplus b_{124}b_{125}b_{127} \oplus b_{124}b_{125}b_{126} \oplus b_{124}b_{125}b_{126}b_{127} \oplus b_{123} \oplus b_{123}b_{126}b_{127} \oplus b_{123}b_{125} \oplus \\
& b_{123}b_{125}b_{126}b_{127} \oplus b_{123}b_{124}b_{126}b_{127} \oplus b_{123}b_{124}b_{125}b_{126} \oplus b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{122} \oplus b_{122}b_{126}b_{127} \oplus \\
& b_{122}b_{125}b_{127} \oplus b_{122}b_{125}b_{126}b_{127} \oplus b_{122}b_{124} \oplus b_{122}b_{124}b_{127} \oplus b_{122}b_{124}b_{126} \oplus b_{122}b_{124}b_{125} \oplus b_{122}b_{124}b_{125}b_{127} \oplus \\
& b_{122}b_{124}b_{125}b_{126} \oplus b_{122}b_{123}b_{127} \oplus b_{122}b_{123}b_{125}b_{126} \oplus b_{122}b_{123}b_{124} \oplus b_{122}b_{123}b_{124}b_{126}b_{127} \oplus \\
& b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{127} \oplus b_{121}b_{126}b_{127} \oplus b_{121}b_{125} \oplus b_{121}b_{125}b_{126} \oplus b_{121}b_{124}b_{127} \oplus b_{121}b_{124}b_{126} \oplus \\
& b_{121}b_{124}b_{126}b_{127} \oplus b_{121}b_{124}b_{125}b_{127} \oplus b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{121}b_{123} \oplus b_{121}b_{123}b_{125} \oplus b_{121}b_{123}b_{125}b_{127} \oplus \\
& b_{121}b_{123}b_{125}b_{126} \oplus b_{121}b_{123}b_{125}b_{126}b_{127} \oplus b_{121}b_{123}b_{124}b_{127} \oplus b_{121}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{122}b_{127} \oplus \\
& b_{121}b_{122}b_{125} \oplus b_{121}b_{122}b_{124}b_{127} \oplus b_{121}b_{122}b_{124}b_{125}b_{126} \oplus b_{121}b_{122}b_{123} \oplus b_{121}b_{122}b_{123}b_{126}b_{127} \oplus \\
& b_{121}b_{122}b_{123}b_{125}b_{126} \oplus b_{121}b_{122}b_{123}b_{124} \oplus b_{121}b_{122}b_{123}b_{124}b_{127} \oplus b_{121}b_{122}b_{123}b_{124}b_{125} \oplus \\
& b_{121}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{120} \oplus b_{120}b_{127} \oplus b_{120}b_{126} \oplus b_{120}b_{125}b_{127} \oplus b_{120}b_{125}b_{126}b_{127} \oplus b_{120}b_{124}b_{127} \oplus \\
& b_{120}b_{124}b_{125}b_{127} \oplus b_{120}b_{124}b_{125}b_{126} \oplus b_{120}b_{123}b_{126} \oplus b_{120}b_{123}b_{126}b_{127} \oplus b_{120}b_{123}b_{125}b_{127} \oplus \\
& b_{120}b_{123}b_{124}b_{127} \oplus b_{120}b_{123}b_{124}b_{126} \oplus b_{120}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{123}b_{124}b_{125} \oplus b_{120}b_{123}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{122} \oplus b_{120}b_{122}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{124} \oplus b_{120}b_{122}b_{124}b_{127} \oplus b_{120}b_{122}b_{123} \oplus b_{120}b_{122}b_{123}b_{126} \oplus \\
& b_{120}b_{122}b_{123}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{121}b_{127} \oplus b_{120}b_{121}b_{125} \oplus b_{120}b_{121}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{124}b_{127} \oplus b_{120}b_{121}b_{124}b_{126} \oplus \\
& b_{120}b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{123} \oplus b_{120}b_{121}b_{123}b_{127} \oplus b_{120}b_{121}b_{123}b_{126} \oplus b_{120}b_{121}b_{123}b_{125} \oplus \\
& b_{120}b_{121}b_{123}b_{125}b_{126} \oplus b_{120}b_{121}b_{123}b_{124}b_{127} \oplus b_{120}b_{121}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{123}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{121}b_{122}b_{126} \oplus b_{120}b_{121}b_{122}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{125} \oplus b_{120}b_{121}b_{122}b_{125}b_{126} \oplus \\
& b_{120}b_{121}b_{122}b_{124}b_{127} \oplus b_{120}b_{121}b_{122}b_{124}b_{126} \oplus b_{120}b_{121}b_{122}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{121}b_{122}b_{123} \oplus b_{120}b_{121}b_{122}b_{123}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{125}b_{127} \oplus b_{120}b_{121}b_{122}b_{123}b_{124} \oplus \\
& b_{120}b_{121}b_{122}b_{123}b_{124}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus k_{109} \oplus \\
& k_{109}k_{111} \oplus k_{109}k_{110} \oplus k_{108}k_{109}k_{111} \oplus k_{108}k_{109}k_{110} \oplus k_{108}k_{109}k_{110}k_{111} \oplus k_{107} \oplus k_{107}k_{110}k_{111} \oplus \\
& k_{107}k_{109} \oplus k_{107}k_{109}k_{110}k_{111} \oplus k_{107}k_{108}k_{110}k_{111} \oplus k_{107}k_{108}k_{109}k_{110} \oplus k_{107}k_{108}k_{109}k_{110}k_{111} \oplus k_{106} \oplus \\
& k_{106}k_{110}k_{111} \oplus k_{106}k_{109}k_{111} \oplus k_{106}k_{109}k_{110}k_{111} \oplus k_{106}k_{108} \oplus k_{106}k_{108}k_{111} \oplus k_{106}k_{108}k_{110} \oplus \\
& k_{106}k_{108}k_{109} \oplus k_{106}k_{108}k_{109}k_{111} \oplus k_{106}k_{108}k_{109}k_{110} \oplus k_{106}k_{107}k_{111} \oplus k_{106}k_{107}k_{109}k_{110} \oplus k_{106}k_{107}k_{108} \oplus \\
& k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{111} \oplus k_{105}k_{110}k_{111} \oplus k_{105}k_{109} \oplus k_{105}k_{109}k_{110} \oplus \\
& k_{105}k_{108}k_{111} \oplus k_{105}k_{108}k_{110} \oplus k_{105}k_{108}k_{110}k_{111} \oplus k_{105}k_{108}k_{109}k_{111} \oplus k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{105}k_{107} \oplus \\
& k_{105}k_{107}k_{109} \oplus k_{105}k_{107}k_{109}k_{111} \oplus k_{105}k_{107}k_{109}k_{110} \oplus k_{105}k_{107}k_{109}k_{110}k_{111} \oplus k_{105}k_{107}k_{108}k_{111} \oplus \\
& k_{105}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{106}k_{111} \oplus k_{105}k_{106}k_{109} \oplus k_{105}k_{106}k_{108}k_{111} \oplus k_{105}k_{106}k_{108}k_{109}k_{110} \oplus \\
& k_{105}k_{106}k_{107} \oplus k_{105}k_{106}k_{107}k_{110}k_{111} \oplus k_{105}k_{106}k_{107}k_{109}k_{110} \oplus k_{105}k_{106}k_{107}k_{108} \oplus \\
& k_{105}k_{106}k_{107}k_{108}k_{111} \oplus k_{105}k_{106}k_{107}k_{108}k_{109} \oplus k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104} \oplus k_{104}k_{111} \oplus \\
& k_{104}k_{110} \oplus k_{104}k_{109}k_{111} \oplus k_{104}k_{109}k_{110}k_{111} \oplus k_{104}k_{108}k_{111} \oplus k_{104}k_{108}k_{109}k_{111} \oplus k_{104}k_{108}k_{109}k_{110} \oplus \\
& k_{104}k_{107}k_{110} \oplus k_{104}k_{107}k_{110}k_{111} \oplus k_{104}k_{107}k_{109}k_{111} \oplus k_{104}k_{107}k_{108}k_{111} \oplus k_{104}k_{107}k_{108}k_{110} \oplus \\
& k_{104}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{107}k_{108}k_{109} \oplus k_{104}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{106} \oplus k_{104}k_{106}k_{109}k_{110}k_{111} \oplus \\
& k_{104}k_{106}k_{108} \oplus k_{104}k_{106}k_{108}k_{111} \oplus k_{104}k_{106}k_{107} \oplus k_{104}k_{106}k_{107}k_{110} \oplus k_{104}k_{106}k_{107}k_{110}k_{111} \oplus \\
& k_{104}k_{106}k_{107}k_{109}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{111} \oplus \\
& k_{104}k_{105}k_{109} \oplus k_{104}k_{105}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{108}k_{111} \oplus k_{104}k_{105}k_{108}k_{110} \oplus \\
& k_{104}k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{107} \oplus k_{104}k_{105}k_{107}k_{111} \oplus k_{104}k_{105}k_{107}k_{110} \oplus k_{104}k_{105}k_{107}k_{109} \oplus \\
& k_{104}k_{105}k_{107}k_{109}k_{110} \oplus k_{104}k_{105}k_{107}k_{108}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{109}k_{111} \oplus \\
& k_{104}k_{105}k_{106}k_{110} \oplus k_{104}k_{105}k_{106}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{109} \oplus k_{104}k_{105}k_{106}k_{109}k_{110} \oplus \\
& k_{104}k_{105}k_{106}k_{108}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{109}k_{111} \oplus \\
& k_{104}k_{105}k_{106}k_{107} \oplus k_{104}k_{105}k_{106}k_{107}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108} \oplus \\
& k_{104}k_{105}k_{106}k_{107}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_0
\end{aligned}
$$

FIGURE 4.18 – Équation du bit b_0 pour un tour

```
def generateEncFullFiles():
    printColor('## Ciphering process', YELLOW)
    createAESFiles('enc')
    addRoundKey(0, 'enc')
    writeRoundEnc(0, subBytes(), shiftRows(), mixColumns())
    addRoundKey(1, 'enc')
    writeRoundEnc(1, subBytes(), shiftRows(), mixColumns())
    addRoundKey(2, 'enc')
    writeRoundEnc(2, subBytes(), shiftRows(), mixColumns())
    addRoundKey(3, 'enc')
    writeRoundEnc(3, subBytes(), shiftRows(), mixColumns())
    addRoundKey(4, 'enc')
    writeRoundEnc(4, subBytes(), shiftRows(), mixColumns())
    addRoundKey(5, 'enc')
    writeRoundEnc(5, subBytes(), shiftRows(), mixColumns())
    addRoundKey(6, 'enc')
    writeRoundEnc(6, subBytes(), shiftRows(), mixColumns())
    addRoundKey(7, 'enc')
    writeRoundEnc(7, subBytes(), shiftRows(), mixColumns())
    addRoundKey(8, 'enc')
    writeRoundEnc(8, subBytes(), shiftRows(), mixColumns())
    addRoundKey(9, 'enc')
    writeFinalRoundEnc(9, subBytes(), shiftRows())
    addRoundKey(10, 'enc')
    writeEndFlag('enc')
    printColor('## Files generated', YELLOW)
```

Listing 18 – Calcul des fonctions booléennes du processus de chiffrement de l'AES

4.4.2.1 Solution pour la fonction de tour

L'algorithme de déchiffrement de l'AES utilise les fonctions `InvShiftRows`, `InvSubBytes` et `InvMixColumns`. Ces fonctions sont respectivement les fonctions inverses des fonctions `ShiftRows`, `SubBytes` et `MixColumns` utilisées dans le processus de chiffrement. Le pseudo code de la fonction de déchiffrement peut s'écrire de la façon suivante (voir fig. 4.19 p. 79), `Nb` correspondant au nombre de mots de 32 bits et `Nr` au nombre de tours utilisés dans l'algorithme.
Les mécanismes internes aux trois fonctions utilisées dans le tour lors du déchiffrement sont similaires à ceux des fonctions de chiffrement. Nous utilisons donc le même raisonnement que celui mis en œuvre plus haut pour générer les équations booléennes correspondantes.
À titre d'exemple, les équations booléennes des trois transformations utilisées dans le processus de déchiffrement pour le bit b_0 sont données dans la figure 4.20 page 79.

4.4.2.2 Solution pour la fonction d'expansion de la clef

La fonction d'expansion de la clef est la même pour les processus de chiffrement et de déchiffrement. Les équations booléennes que nous avons construites précédemment sont donc réutilisables.

```
1: function InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
2:     byte state[4,Nb]
3:     state ← in
4:     AddRounkey(state, w[Nr*Nb, (Nr+1)*Nb-1])
5:     for round=Nr-1 step -1 downto 1 do
6:         InvShiftRows(state)
7:         InvSubBytes(state)
8:         AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
9:         InvMixColumns(state)
10:    end for
11:    InvShiftRows(state)
12:    InvSubBytes(state)
13:    AddRounkey(state, w[0, Nb-1])
14:    return state
15: end function
```

FIGURE 4.19 – Pseudo-code pour le déchiffrement

$$\text{invSubBytes}(b_0) = b_6b_7 \oplus b_5b_6 \oplus b_4 \oplus b_4b_7 \oplus b_4b_5b_7 \oplus b_4b_5b_6 \oplus b_4b_5b_6b_7 \oplus b_3b_7 \oplus b_3b_6b_7 \oplus b_3b_5 \oplus b_3b_5b_6 \oplus b_3b_5b_6b_7 \oplus b_3b_4 \oplus b_3b_4b_7 \oplus b_3b_4b_6b_7 \oplus b_3b_4b_5b_6 \oplus b_3b_4b_5b_6b_7 \oplus b_2b_6 \oplus b_2b_5 \oplus b_2b_5b_6 \oplus b_2b_5b_6b_7 \oplus b_2b_4b_6 \oplus b_2b_4b_6b_7 \oplus b_2b_4b_5b_7 \oplus b_2b_3b_7 \oplus b_2b_3b_6b_7 \oplus b_2b_3b_5b_7 \oplus b_2b_3b_5b_6b_7 \oplus b_2b_3b_4b_6 \oplus b_2b_3b_4b_5b_6 \oplus b_1b_7 \oplus b_1b_6 \oplus b_1b_6b_7 \oplus b_1b_5 \oplus b_1b_4b_6b_7 \oplus b_1b_4b_5b_7 \oplus b_1b_3b_6 \oplus b_1b_3b_6b_7 \oplus b_1b_3b_5 \oplus b_1b_3b_5b_6b_7 \oplus b_1b_2 \oplus b_1b_2b_7 \oplus b_1b_2b_6b_7 \oplus b_1b_2b_5b_6b_7 \oplus b_1b_2b_4 \oplus b_1b_2b_4b_7 \oplus b_1b_2b_4b_6b_7 \oplus b_1b_2b_4b_5b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_1b_2b_4b_5b_6b_7 \oplus b_1b_2b_3b_7 \oplus b_1b_2b_3b_5b_7 \oplus b_1b_2b_3b_5b_6 \oplus b_1b_2b_3b_4 \oplus b_1b_2b_3b_4b_6 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_1b_2b_3b_4b_5 \oplus b_1b_2b_3b_4b_5b_6 \oplus b_0b_7 \oplus b_0b_5b_7 \oplus b_0b_5b_6b_7 \oplus b_0b_4b_6b_7 \oplus b_0b_4b_5b_6b_7 \oplus b_0b_3 \oplus b_0b_3b_6 \oplus b_0b_3b_5 \oplus b_0b_3b_5b_7 \oplus b_0b_3b_5b_6b_7 \oplus b_0b_3b_4b_6b_7 \oplus b_0b_3b_4b_5 \oplus b_0b_3b_4b_5b_7 \oplus b_0b_2b_6 \oplus b_0b_2b_5 \oplus b_0b_2b_5b_6 \oplus b_0b_2b_5b_6b_7 \oplus b_0b_2b_4 \oplus b_0b_2b_4b_7 \oplus b_0b_2b_4b_6 \oplus b_0b_2b_4b_5 \oplus b_0b_2b_4b_5b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_2b_3b_4 \oplus b_0b_2b_3b_4b_6b_7 \oplus b_0b_2b_3b_4b_5 \oplus b_0b_2b_3b_4b_5b_6 \oplus b_0b_1b_7 \oplus b_0b_1b_5b_7 \oplus b_0b_1b_5b_6b_7 \oplus b_0b_1b_4b_7 \oplus b_0b_1b_4b_6b_7 \oplus b_0b_1b_4b_5b_6b_7 \oplus b_0b_1b_3 \oplus b_0b_1b_3b_6 \oplus b_0b_1b_3b_6b_7 \oplus b_0b_1b_3b_5b_6b_7 \oplus b_0b_1b_3b_4b_5b_7 \oplus b_0b_1b_2b_6b_7 \oplus b_0b_1b_2b_5 \oplus b_0b_1b_2b_5b_7 \oplus b_0b_1b_2b_4 \oplus b_0b_1b_2b_4b_6 \oplus b_0b_1b_2b_4b_5 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_0b_1b_2b_4b_5b_6 \oplus b_0b_1b_2b_3 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_1b_2b_3b_5b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_0b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_5$$

$$\text{invShiftRows}(b_0) = b_0$$

$$\text{invMixColumns}(b_0) = b_3 \oplus b_2 \oplus b_1 \oplus b_{11} \oplus b_9 \oplus b_8 \oplus b_{19} \oplus b_{18} \oplus b_{16} \oplus b_{27} \oplus b_{24}$$

FIGURE 4.20 – Équations booléennes des trois fonctions de déchiffrement pour le bit b_0

4.4.2.3 Solution globale

Nous avons maintenant une fonction booléenne pour chacune des fonctions `InvSubBytes` $ISB()$, `InvShiftRows` $ISR()$ et `InvMixColumns` $IMC()$. Cependant, contrairement à l'agencement des tours intermédiaires du processus de chiffrement, ces trois fonctions ne sont pas combinées entres elles. En effet, la fonction `AddRoundKey` intervient non plus en fin de tour mais s'intercale entre les fonctions `InvSubBytes` et `InvMixColumns`.
Ainsi, pour un bloc $B = (b_1, \cdots, b_{128})$ et une clef $K = (k_1, \cdots, k_{128})$ en entrée du tour, le bloc $B' = (b'_1, \cdots, b'_{128})$ en sortie est tel que :

$$B' = IMC(ISB \circ ISR(B) \oplus AD(K))$$

Afin de réduire les équations booléennes, nous n'allons donc pouvoir combiner que les équations de `InvSubBytes` et `InvShiftRows`. Comme précédemment, pour

y parvenir il suffit de remplacer chaque variable en entrée d'une fonction par sa valeur en sortie de la fonction précédente selon l'équation suivante :

$$b'_i = ISB(ISR(b_i)) \quad \forall i \in (1, \cdots, 128)$$

En langage python, la fonction de génération d'un tour s'écrit selon le code suivant (voir listing 19 p. 80).

```
1 def writeRoundDec(numRound, equaSB, equaSR):
2     printColor('## Round %s' % numRound, GREEN)
3     resultSR = []
4     for i in xrange(blockSize):
5         equaSR[i] = equaSR[i].split('_')
6         resultSR.append(equaSB[int(equaSR[i][1])])
7     binMon = generateBinaryMonomes(resultSR)
8     return resultSR
```

Listing 19 – Calcul de l'équation pour une fonction de tour de déchiffrement

Comme pour le processus de chiffrement, nous pouvons maintenant décrire le processus intégral de déchiffrement de l'AES sous la forme d'équations booléennes. La fonction en langage python calculant ce processus est donnée dans le listing 20 page 81.

4.4.3 Implémentation et preuve

Nous disposons maintenant de deux systèmes d'équations booléennes correspondants aux processus de chiffrement et de déchiffrement de l'AES. Ces deux systèmes comptent chacun :

- 128 équations, une par bit de bloc ;
- 1280 variables pour le bloc en entrée ;
- 1280 variables pour la clef.

Concernant les variables de clefs, le fait que nous ayons une équation booléenne par clef de tour implique que nous ayons un jeu de 128 nouvelles variables à chaque tour soit 1280 variables pour l'AES 128. Chacune des variables de clef du tour n étant décrite en fonction des variables de clef du tour $n-1$. En conséquence et du fait de l'opération de `XOR` entre la clef de tour et les bits issus de la fonction de tour, nous sommes obligés d'insérer un nouveau jeu de 128 variables pour décrire l'évolution du bloc de données à chaque tour.

Finalement nous avons décrit les processus de chiffrement et de déchiffrement de l'AES sous la forme de deux systèmes d'équations booléennes à 128 équations et 2560 variables. À titre de comparaison nous rappelons, dans la figure 4.21 page 81, les résultats obtenus avec les systèmes d'équations présentés au début de cette thèse.

Ce mécanisme nous permet alors de décrire l'ensemble du processus de chiffrement et de déchiffrement de l'AES sous la forme de fichiers en utilisant la même représentation que pour le mini-AES. Pour chaque processus de chiffrement et de déchiffrement, nous avons donc 128 fichiers, un par bit de bloc. Dans ces fichiers, chaque ligne décrit un monôme et le passage d'une ligne à la suivante

```
def generateDecFullFiles():
    printColor('## Deciphering process', YELLOW)
    createAESFiles('dec')
    addRoundKey(10, 'dec')
    writeRoundDec(9, invSubBytes(), invShiftRows())
    addRoundKey(9, 'dec')
    writeInvMixColumns(9)
    writeRoundDec(8, invSubBytes(), invShiftRows())
    addRoundKey(8, 'dec')
    writeInvMixColumns(8)
    writeRoundDec(7, invSubBytes(), invShiftRows())
    addRoundKey(7, 'dec')
    writeInvMixColumns(7)
    writeRoundDec(6, invSubBytes(), invShiftRows())
    addRoundKey(6, 'dec')
    writeInvMixColumns(6)
    writeRoundDec(5, invSubBytes(), invShiftRows())
    addRoundKey(5, 'dec')
    writeInvMixColumns(5)
    writeRoundDec(4, invSubBytes(), invShiftRows())
    addRoundKey(4, 'dec')
    writeInvMixColumns(4)
    writeRoundDec(3, invSubBytes(), invShiftRows())
    addRoundKey(3, 'dec')
    writeInvMixColumns(3)
    writeRoundDec(2, invSubBytes(), invShiftRows())
    addRoundKey(2, 'dec')
    writeInvMixColumns(2)
    writeRoundDec(1, invSubBytes(), invShiftRows())
    addRoundKey(1, 'dec')
    writeInvMixColumns(1)
    writeRoundDec(0, invSubBytes(), invShiftRows())
    addRoundKey(0, 'dec')
    writeEndFlag('dec')
    printColor('## Files generated', YELLOW)
```

Listing 20 – Calcul des fonctions booléennes du processus de déchiffrement de l'AES

Source	Nombre d'équations	Nombre de variables	Nombre de termes
Nicolas Courtois et Joseph Pieprzyk [63]	8000	1600	2^{20}
Sean Murphy et Matthew Robshaw [147]	2688	3968	5248
Michel Dubois et Eric Filiol [87]	128	2560	7881

FIGURE 4.21 – Tableau comparatif des systèmes d'équations décrivant le processus de chiffrement de l'AES

correspond à l'opération `XOR`. Un fichier d'exemple ainsi obtenu est donné dans l'annexe C page 169.
Pour implémenter ce mécanisme de description de l'algorithme de chiffrement de l'AES et générer les 128 fichiers, nous avons développé et utilisé un script python s'appuyant sur celui décrit plus haut dans notre présentation de l'AES. Les fichiers source de ce programme sont disponibles sur Internet [81, BooleanAES]. Le programme principal, `aes_equa.py`, offre la possibilité d'une part, de générer les fichiers pour les fonctions de chiffrement et de déchiffrement de l'AES avec les fonctions `generateEncFullFiles()` et `generateDecFullFiles()` et, d'autre part, de contrôler que le chiffrement et le déchiffrement à partir des fichiers obtenus est bien conforme.
Ainsi, les fonctions `controlEncFullFiles()` et `controlDecFullFiles()` réalisent respectivement le chiffrement et le déchiffrement à partir des fichiers générés précédemment. La fonction `controlEncFullFiles()` prend en entrée un bloc de 128 bits de texte clair et un bloc de 128 bits de clef tandis que la fonction `controlDecFullFiles()` prend en entrée un bloc de 128 bits de chiffré et un bloc de 128 bits de clef. Les blocs choisis sont ceux fournis comme vecteurs de test dans l'annexe B du FIPS 197 [155]. Les résultats obtenus correspondent à ceux fournis dans le FIPS : les fichiers que nous avons générés décrivent bien l'algorithme de chiffrement et de déchiffrement de l'AES.

4.4.3.1 Résultats obtenus pour le processus de chiffrement

Le résultat obtenu par la fonction `generateEncFullFiles()` est montré dans le listing 21 page 83 et le résultat obtenu par la fonction `controlEncFullFiles()` est montré dans le listing 22 page 84. La fonction de contrôle `controlEncFullFiles()`, dont le code est donné dans le listing 23 page 85, injecte les 128 variables initiales correspondant au bloc de texte clair et les 1280 variables correspondant aux blocs de clefs de chaque tour, dans les fonctions booléennes .

4.4.3.2 Résultats obtenus pour le processus de déchiffrement

Selon le même principe que pour les fonctions booléennes du chiffrement, le résultat obtenu par la fonction `generateDecFullFiles()` est montré dans le listing 24 page 86 et le résultat obtenu par la fonction `controlDecFullFiles()` est montré dans le listing 25 page 87.
Que ce soit pour le processus de chiffrement que pour celui du déchiffrement, les résultats que nous obtenons en utilisant les fichiers pour chiffrer ou déchiffrer un bloc sont conformes à ceux donnés dans le FIPS 197. Nos systèmes d'équations booléennes sont donc justes et décrivent bien l'algorithme de l'AES.

—=oOo=—

Après avoir présenté succinctement l'algèbre de Boole, les fonctions booléennes et deux de leurs représentations, nous avons élaboré un processus nous permettant de traduire l'algorithme de chiffrement du mini-AES en fonctions booléennes. Nous avons ensuite appliqué ce processus à l'algorithme de chiffrement et de déchiffrement de l'AES. Puis nous avons défini un mode de représentation

```
./aes_equa.py
## Ciphering process
## Create directory AES_files
## AddRoundKey0
## Round0
## AddRoundKey1
## Round1
## AddRoundKey2
## Round2
## AddRoundKey3
## Round3
## AddRoundKey4
## Round4
## AddRoundKey5
## Round5
## AddRoundKey6
## Round6
## AddRoundKey7
## Round7
## AddRoundKey8
## Round8
## AddRoundKey9
## Round9
## AddRoundKey10
## Files generated
```

Listing 21 – Résultat du programme de création des fichiers pour le chiffrement

de ces fonctions booléennes sous la forme de fichiers informatiques. Enfin, nous avons mis au point un programme permettant d'implémenter ce processus et de contrôler que les résultats attendus sont conformes à ceux fournis dans le FIPS. Finalement, nous avons obtenu deux nouveaux systèmes d'équations booléennes, le premier décrivant le processus de chiffrement tandis que le deuxième décrit le processus de déchiffrement de l'*Advanced Encryption Standard* et comprenant chacun 128 équations et $(128 \times 10) + (128 \times 10) = 2560$ variables.
Ce travail a fait l'objet de plusieurs publications en France [83] et à l'étranger [82], [84], [88] et [87].

```
 ./aes_equa.py
## Clear block 00112233445566778899aabbccddeeff
## Key block 000102030405060708090a0b0c0d0e0f
## addRoundKey0
00102030405060708090a0b0c0d0e0f0 32
## Round0
5f72641557f5bc92f7be3b291db9f91a 32
## addRoundKey1
89d810e8855ace682d1843d8cb128fe4 32
## Round1
ff87968431d86a51645151fa773ad009 32
## addRoundKey2
4915598f55e5d7a0daca94fa1f0a63f7 32
## Round2
4c9c1e66f771f0762c3f868e534df256 32
## addRoundKey3
fa636a2825b339c940668a3157244d17 32
## Round3
6385b79ffc538df997be478e7547d691 32
## addRoundKey4
247240236966b3fa6ed2753288425b6c 32
## Round4
f4bcd45432e554d075f1d6c51dd03b3c 32
## addRoundKey5
c81677bc9b7ac93b25027992b0261996 32
## Round5
9816ee7400f87f556b2c049c8e5ad036 32
## addRoundKey6
c62fe109f75eedc3cc79395d84f9cf5d 32
## Round6
c57e1c159a9bd286f05f4be098c63439 32
## addRoundKey7
d1876c0f79c4300ab45594add66ff41f 32
## Round7
baa03de7a1f9b56ed5512cba5f414d23 32
## addRoundKey8
fde3bad205e5d0d73547964ef1fe37f1 32
## Round8
e9f74eec023020f61bf2ccf2353c21c7 32
## addRoundKey9
bd6e7c3df2b5779e0b61216e8b10b689 32
## Round9
7ad5fda789ef4e272bca100b3d9ff59f 32
## addRoundKey10
69c4e0d86a7b0430d8cdb78070b4c55a 32
69c4e0d86a7b0430d8cdb78070b4c55a (FIPS result)
```

Listing 22 – Résultat du programme de contrôle des fichiers pour le chiffrement

```
def controlEncFullFiles():
    clearBlock = '00112233445566778899aabbccddeeff'
    key = '000102030405060708090a0b0c0d0e0f'
    cipherBlock = '69c4e0d86a7b0430d8cdb78070b4c55a'

    printColor('## Clear block %s' % (clearBlock), BLUE)
    print largeHex2Bin(clearBlock), len(largeHex2Bin(clearBlock))
    printColor('## Key block %s' % (key), BLUE)
    print largeHex2Bin(key), len(largeHex2Bin(key))

    key = largeHex2Bin(key)
    clearBlock = largeHex2Bin(clearBlock)

    block = controlBlock('enc', '## addRoundKey0', '## Round0', clearBlock, key)
    block = controlBlock('enc', '## Round0', '## addRoundKey1', block)
    block = controlBlock('enc', '## addRoundKey1', '## Round1', block, key)
    block = controlBlock('enc', '## Round1', '## addRoundKey2', block)
    block = controlBlock('enc', '## addRoundKey2', '## Round2', block,\
        largeHex2Bin('d6aa74fdd2af72fadaa678f1d6ab76fe'))
    block = controlBlock('enc', '## Round2', '## addRoundKey3', block)
    block = controlBlock('enc', '## addRoundKey3', '## Round3', block,\
        largeHex2Bin('b692cf0b643dbdf1be9bc5006830b3fe'))
    block = controlBlock('enc', '## Round3', '## addRoundKey4', block)
    block = controlBlock('enc', '## addRoundKey4', '## Round4', block,\
        largeHex2Bin('b6ff744ed2c2c9bf6c590cbf0469bf41'))
    block = controlBlock('enc', '## Round4', '## addRoundKey5', block)
    block = controlBlock('enc', '## addRoundKey5', '## Round5', block,\
        largeHex2Bin('47f7f7bc95353e03f96c32bcfd058dfd'))
    block = controlBlock('enc', '## Round5', '## addRoundKey6', block)
    block = controlBlock('enc', '## addRoundKey6', '## Round6', block,\
        largeHex2Bin('3caaa3e8a99f9deb50f3af57adf622aa'))
    block = controlBlock('enc', '## Round6', '## addRoundKey7', block)
    block = controlBlock('enc', '## addRoundKey7', '## Round7', block,\
        largeHex2Bin('5e390f7df7a69296a7553dc10aa31f6b'))
    block = controlBlock('enc', '## Round7', '## addRoundKey8', block)
    block = controlBlock('enc', '## addRoundKey8', '## Round8', block,\
        largeHex2Bin('14f9701ae35fe28c440adf4d4ea9c026'))
    block = controlBlock('enc', '## Round8', '## addRoundKey9', block)
    block = controlBlock('enc', '## addRoundKey9', '## Round9', block,\
        largeHex2Bin('47438735a41c65b9e016baf4aebf7ad2'))
    block = controlBlock('enc', '## Round9', '## addRoundKey10', block)
    block = controlBlock('enc', '## addRoundKey10', '## end', block,\
        largeHex2Bin('549932d1f08557681093ed9cbe2c974e'))
    print('%s (FIPS result)' % (cipherBlock))
```

Listing 23 – Code de la fonction `controlEncFullFiles()`

```
./aes_equa.py
## Deciphering process
## Create directory AES_files
## AddRoundKey10
## Round 9
## AddRoundKey9
## InvMixColumns 9
## Round 8
## AddRoundKey8
## InvMixColumns 8
## Round 7
## AddRoundKey7
## InvMixColumns 7
## Round 6
## AddRoundKey6
## InvMixColumns 6
## Round 5
## AddRoundKey5
## InvMixColumns 5
## Round 4
## AddRoundKey4
## InvMixColumns 4
## Round 3
## AddRoundKey3
## InvMixColumns 3
## Round 2
## AddRoundKey2
## InvMixColumns 2
## Round 1
## AddRoundKey1
## InvMixColumns 1
## Round 0
## AddRoundKey0
## Files generated
```

Listing 24 – Résultat du programme de création des fichiers pour le déchiffrement

```
./aes_equa.py
## Cipher block 69c4e0d86a7b0430d8cdb78070b4c55a
## Key block 000102030405060708090a0b0c0d0e0f
## addRoundKey10
7ad5fda789ef4e272bca100b3d9ff59f 32
## Round9
bd6e7c3df2b5779e0b61216e8b10b689 32
## addRoundKey9
e9f74eec023020f61bf2ccf2353c21c7 32
## invMixColumns9
54d990a16ba09ab596bbf40ea111702f 32
## Round8
fde3bad205e5d0d73547964ef1fe37f1 32
## addRoundKey8
baa03de7a1f9b56ed5512cba5f414d23 32
## invMixColumns8
3e1c22c0b6fcbf768da85067f6170495 32
## Round7
...
## Round3
fa636a2825b339c940668a3157244d17 32
## addRoundKey3
4c9c1e66f771f0762c3f868e534df256 32
## invMixColumns3
3bd92268fc74fb735767cbe0c0590e2d 32
## Round2
4915598f55e5d7a0daca94fa1f0a63f7 32
## addRoundKey2
ff87968431d86a51645151fa773ad009 32
## invMixColumns2
a7be1a6997ad739bd8c9ca451f618b61 32
## Round1
89d810e8855ace682d1843d8cb128fe4 32
## addRoundKey1
5f72641557f5bc92f7be3b291db9f91a 32
## invMixColumns1
6353e08c0960e104cd70b751bacad0e7 32
## Round0
00102030405060708090a0b0c0d0e0f0 32
## addRoundKey0
00112233445566778899aabbccddeeff 32
00112233445566778899aabbccddeeff (FIPS result)
```

Listing 25 – Résultat du programme de contrôle des fichiers pour le déchiffrement

Chapitre 5

Analyse statistique visuelle

> *« The field of cryptography will perhaps be the most rewarding. There is a remarkably close parallel between the problems of the physicist and those of the cryptographer. The system on which a message is enciphered corresponds to the laws of the universe, the intercepted messages to the evidence available, the keys for a day or a message to important constants which have to be determined. The correspondence is very close, but the subject matter of cryptography is very easily dealt with by discrete machinery, physics not so easily. »*
>
> Alan Turing – Intelligent Machinery

Dans le domaine de la sécurité des systèmes d'information, la visualisation est couramment utilisée pour différentes tâches comme l'analyse de logs [197], la détection d'attaques [149], l'analyse de binaires [79] et l'ingénierie inverse [55, 54], mais aujourd'hui, il n'existe pas de façon simple d'analyser et de différencier des données aléatoires. Cependant, les systèmes d'exploitation ou les protocoles cryptographiques utilisent constamment la génération d'aléa, par exemple, pour générer un numéro de séquence TCP ou pour générer une clef de chiffrement aléatoire pour le wi-fi ou le Web.

5.1 Algorithme cryptographique et aléa

Le Graal de tout algorithme cryptographique est d'obtenir, à chaque étape interne et à l'issue du processus de chiffrement, une séquence d'apparence la plus proche possible de l'aléa parfait. En effet, la sécurité d'un algorithme cryptographique dépend de sa capacité à générer des quantités imprévisibles.
En partant du principe que l'aléa parfait n'est qu'une vision philosophique et que, dans les faits, la perfection de l'aléa est tributaire des tests statistiques qui lui ont été appliqués [97], nous pouvons dire que l'aléa cryptographique doit être aléatoire dans le sens où la probabilité d'une valeur particulière choisie doit être suffisamment faible pour empêcher un adversaire de gagner l'avantage grâce à l'optimisation d'une stratégie de recherche basée sur cette probabilité [142].

Le moyen pour arriver à réaliser ce paradigme est de concevoir les algorithmes cryptographiques comme des générateurs de bits pseudo-aléatoires (**Pseudo-random Bit Generator** ou PRBG en anglais).

5.1.1 Générateur de bits pseudo-aléatoire

Définition 5.1 (*Générateur de bits aléatoire*)**.** Un générateur de bits aléatoire est un équipement ou un algorithme qui produit une séquence de bits statistiquement indépendants et non biaisés [142].

Certains équipements matériels génèrent de l'aléa à partir du temps écoulé entre l'émission de particules durant la phase de décroissance radioactive ou encore à partir du bruit thermique émis par une résistance ou une diode à semi-conducteurs. De même, pour générer de l'aléa, certains logiciels utilisent des algorithmes associant diverses sources comme le temps écoulé entre deux frappes au clavier, le mouvement de la souris ou le contenu des buffers d'entrée/sortie.

Définition 5.2 (*Générateur de bits pseudo-aléatoire*)**.** Un générateur de bits pseudo-aléatoire (PRBG) est un algorithme déterministe qui, à partir d'une séquence de bits réellement aléatoire de longueur k, appelée graine, produit une séquence de bits de longueur $l \gg k$ qui semble aléatoire. La séquence initiale est appelée la graine tandis que la séquence produite par le PRBG est appelée séquence de bits pseudo-aléatoire.

Générer des nombres aléatoires est un problème très difficile pour les ordinateurs parce qu'ils sont déterministes et, comme le dit John von Neumann *« Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin »* [204]. Ainsi, la séquence produite par un PRBG n'est pas réellement aléatoire. Plus précisément, le nombre de séquences possibles produites par le PRBG est, au plus, une petite fraction, à savoir $\frac{2^k}{2^l}$ [142], de toutes les séquences binaires possibles de longueur l. L'objectif, est de prendre une petite séquence réellement aléatoire et de l'étendre à une séquence de longueur beaucoup plus grande, de telle sorte que l'attaquant ne puisse pas facilement faire la distinction entre les séquences de sortie du PRBG et des séquences réellement aléatoires de même longueur.
Pour qu'un algorithme de PRBG soit cryptographiquement sûr, trois principales règles doivent être respectées [142] :

1. la longueur k de la graine doit être de taille suffisante. En fait, k doit être telle, que la recherche exhaustive sur l'ensemble des 2^k éléments de l'espace des graines soit calculatoirement difficile pour l'attaquant ;
2. la séquence produite par un PRBG doit être statistiquement proche d'une séquence réellement aléatoire, ou, plus précisément, approximée par une séquence de variables binaires, indépendantes et identiquement distribuées. Nous disons alors qu'un PRBG passe tous les tests statistiques en temps polynomial si aucun algorithme polynomial ne peut correctement faire la distinction entre une séquence produite par le PRBG et une séquence réellement aléatoire de même longueur avec une probabilité $p \gg \frac{1}{2}$;
3. les bits produits ne doivent pas être prédictibles, à partir d'une séquence partielle déjà connue, pour un attaquant ayant des ressources de calcul

limitées. Un PRBG respecte cette règle, dite du "bit suivant", si, à partir des premiers l bits d'une séquence s produite par le PRBG, aucun algorithme polynomial n'est capable de prédire le bit $(l+1)$ de s avec une probabilité $p \gg \frac{1}{2}$.

5.1.2 Représentation de PRBG

Afin de mieux appréhender le résultat obtenu à partir de différents algorithmes de PRBG, nous allons représenter les séquences obtenues dans un environnement en deux et trois dimensions simultanément.

5.1.2.1 Représentation en 2 dimensions

Un PRBG est assimilable à un système non-linéaire générant une série chronologique de données. Si nous voulons représenter une telle série dans un environnement en deux dimensions, une première approche pourrait consister à parcourir linéairement tous les points du plan en assignant à chaque point une couleur correspondant à une entrée de la série. Cette idée semble bonne mais elle a l'inconvénient de ne pas représenter la réalité de la série.
En effet, si nous prenons un plan délimité par un rectangle de largeur x et de hauteur y avec $x \times y = |n|$, $|n|$ représentant le cardinal des éléments de la série n, alors le point de coordonnées (i, j), représentant l'élément $n(t), t < |n|$ de la série, a comme voisins les points $(i-1, j)$ et $(i+1, j)$ représentant respectivement les éléments $n(t-1)$ et $n(t+1)$ de la série, mais a également comme voisins les points $(i, j-1)$ et $(i, j+1)$ représentant les éléments $n(t-x)$ et $n(t+x)$ de la série. Ainsi, les points sur une même ligne correspondent à des éléments qui se suivent dans la série mais il n'y a pas de lien exploitable entre différentes lignes. À titre d'exemple, nous pouvons voir que sur la figure 5.1 page 91, le point d'index 15 a comme voisins les points d'index 14 et 16 mais également les points d'index 8 et 20 qui ne lui sont pas voisins dans la séquence.

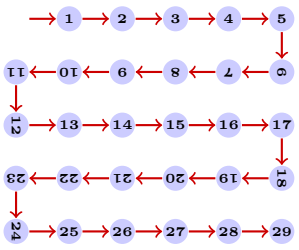


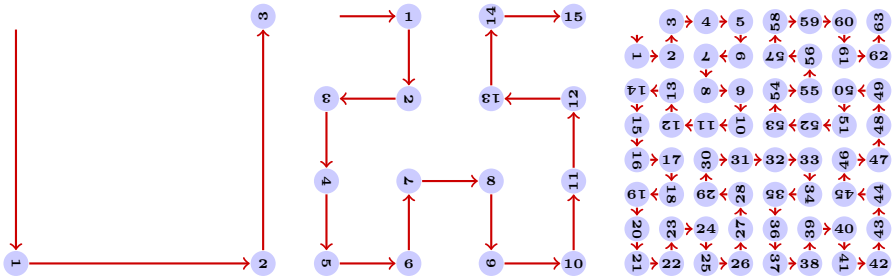
FIGURE 5.1 – Courbe standard

Donc, pour obtenir une meilleure représentation de notre série dans un espace à deux dimensions nous avons besoin d'un algorithme plus pertinent. Une courbe de remplissage est une fonction injective continue qui fait correspondre un intervalle compact à un hypercube n-dimensionnel [110]. Les courbes de remplissage ont été découvertes en 1890 par le mathématicien Giuseppe Peano [160]. Pour réaliser notre objectif nous utilisons une courbe de remplissage spécifique proposée par David Hilbert [113] peu de temps après la découverte de Peano.
Une courbe de Hilbert est une courbe de remplissage qui réalise la projection d'un intervalle à une dimension dans un espace à deux dimensions. Sa construc-

tion repose sur la répétition d'un schéma simple : les trois premiers côtés d'un carré. À chaque étape le carré est tourné, réduit et répété jusqu'à obtenir une courbe qui remplit le plan. En fait, une courbe de Hilbert peut être vue comme un système de Lindenmayer [210], connu également sous le nom de L-system. Un L-system est un système de réécriture de séquence de caractères qui peut être utilisé pour générer des fractales de dimensions comprises entre un et deux. Pour la courbe de Hilbert les règles du L-system sont :

$$L = +RF - LFL - FR+$$
$$R = -LF + RFR + FL-$$

avec l'alphabet constitué des lettres L et R, F signifiant dessiner vers l'avant, $-$ signifiant tourner à gauche de 90 degrés et $+$ signifiant tourner à droite de 90 degrés.
Les premières itérations d'une courbe de Hilbert sont présentées dans la figure 5.2. Comme nous pouvons le voir sur les figures 5.2a, 5.2b et 5.2c, chaque point a un index correspondant à l'index de chaque entrée de la série. Nous constatons alors que, contrairement à notre première approche décrite ci-dessus, l'utilisation d'une courbe de Hilbert nous permet de préserver la localité des données. Nous entendons par là, que le voisinage des points dans la série est conservé dans l'espace à deux dimensions.



(a) Ordre 1 (b) Ordre 2 (c) Ordre 3

FIGURE 5.2 – Premières itérations de la courbe de Hilbert

Ainsi, au travers de ce mode de présentation en deux dimensions, nous avons une première approche pour représenter un algorithme de PRBG.

5.1.2.2 Représentation en 3 dimensions

Après l'approche en deux dimensions nous allons essayer de représenter notre séquence dans un environnement en trois dimensions.
Un des moyens les plus couramment utilisé pour analyser une série de ce type est de reconstruire son espace des phases en utilisant la méthode des délais [119]. L'espace des phases est un espace à n dimensions qui décrit complètement l'état d'un système à n variables. À titre d'exemple, l'espace des phases décrivant l'atterrissage d'une fusée est un espace à deux dimensions. La première dimension est la vitesse de la fusée et la deuxième dimension est sa distance au sol. L'espace des phases est alors un graphe représentant en abscisse la vitesse et en ordonnée la distance au sol. Ainsi, pour que la fusée atterrisse sans encombre, il

faut que la courbe décrivant sa progression, dans son espace des phases, tende vers zéro (voir figure 5.3 page 93).

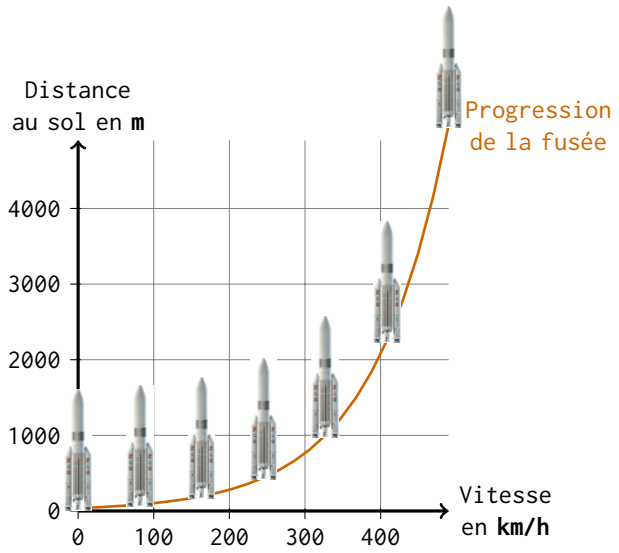


FIGURE 5.3 – Espace des phases de la trajectoire d'une fusée

Ici, notre objectif est donc de représenter en trois dimensions une séquence à une dimension. La méthode des délais [159] permet de reconstruire les dimensions manquantes en utilisant les valeurs précédentes comme coordonnées supplémentaires. Pour cela, au lieu d'utiliser les valeurs brutes retournées par la fonction, nous calculons, pour chaque coordonnée, la différence de deux valeurs successives. Cela nous permet de générer un résultat plus utile pour montrer la dynamique de la fonction. Ainsi, si $s[t]$ est la séquence fournie par un PRBG en fonction du temps t, alors les coordonnées x, y, z d'un point dans notre environnement sont calculées à partir des équations suivantes :

$$x[t] = s[t-2] - s[t-3]$$
$$y[t] = s[t-1] - s[t-2]$$
$$z[t] = s[t] - s[t-1]$$

Ensuite, en représentant la séquence de points ainsi obtenue dans un environnement en trois dimensions nous obtenons une forme spécifique à la fonction de PRBG donnée. Cette forme, appelée *attracteur*, révèle la nature complexe des dépendances entre les différents éléments de la séquence générés par l'algorithme étudié [216, 217].
Prenons un exemple concret : la suite de nombres suivante est issue de l'algorithme de PRBG qui génère les numéros de séquences de session TCP du système d'exploitation GNU/Linux RedHat dans sa version 7.3.

```
3281499104, 3271545868, 3287443610, 3238749981, 3274168813, 3302234066,
3229771300, 3287970591, 3295595222, 3298841199, 3292774952, 3294591612,
3294540537, 3294046036, 3296969037, 3293746299, 3300112100, 3292483752,
3235813772, 3298333679, 3273849495, 3293225350, 3295916141, 3299559674,
3295896492, 3303667282, 3301180722, 3290619488, 3301904507, 3286172964
```

Au premier abord, il est difficile de déterminer s'il existe un lien entre les différents éléments de cette suite de nombres. Par contre, la figure 5.4 page 94 permet de formaliser ce lien en dévoilant une forme qui est caractéristique du PRBG utilisé par le noyau Linux 2.4.18 utilisé par cette distribution de Linux.

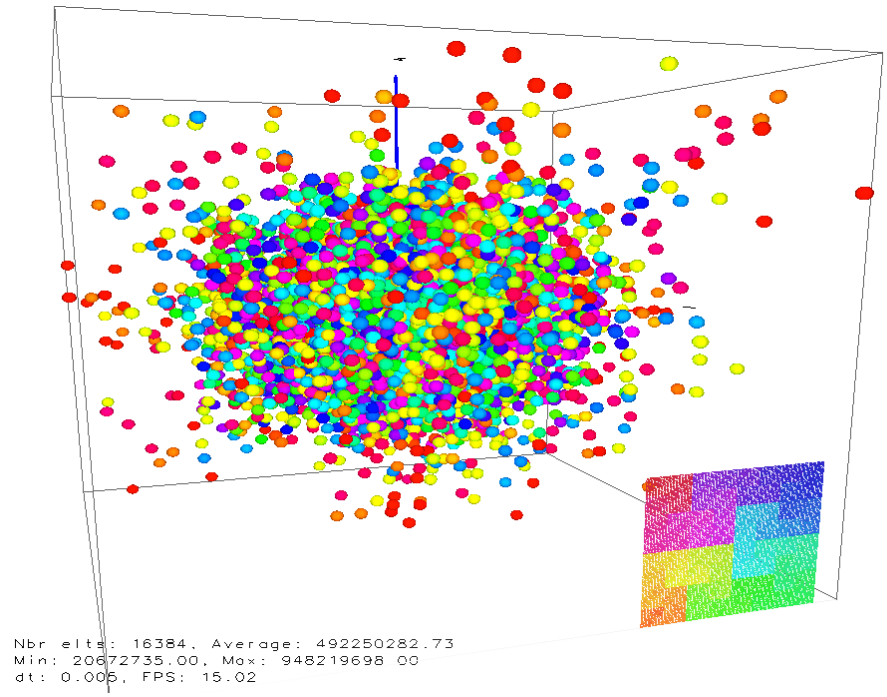


FIGURE 5.4 – Attracteur du PRBG de Redhat 7.3 - vue 45°

Grâce à ce mode de présentation, il nous est maintenant plus facile d'identifier un algorithme de PRBG, étant donné qu'un même algorithme donnera toujours le même attracteur.
Pour la suite de ce chapitre, nous avons développé un ensemble de programmes de génération et de visualisation en trois dimensions de différents types de PRBG. Ces programmes sont disponibles sur Internet [81, PRBG-3D].

5.1.3 Exemples de PRBG

Nous avons vu plus haut qu'un algorithme cryptographique doit être conçu comme un générateur de bits pseudo-aléatoire. Nous allons maintenant étudier quelques algorithmes de PRBG. Nous finirons par la présentation de l'attracteur du RC4 et de l'*Advanced Encryption Standard*.

5.1.3.1 Véritable aléa

Avant de débuter notre comparaison de PRBG, il nous faut un référentiel, c'est-à-dire la représentation de l'attracteur d'un véritable aléa. Comme un ordinateur est une machine déterministe, il ne lui est pas possible de produire un véritable aléa. Seuls des équipements matériels s'appuyant sur des éléments aléatoires physiques peuvent fournir ce type d'aléa.
Le site Internet www.random.org fournit la possibilité de générer des séquences de véritable aléa. Il utilise le bruit atmosphérique pour produire cet aléa. Ce site est issu d'un projet scientifique du docteur Mads Haahr de la "*School of Computer Science and Statistics*" du Trinity College de Dublin [109]. Il est utilisé pour des jeux en lignes, pour générer l'aléa de jeux de loterie, pour des projets scientifiques... Nous avons généré une séquence de 10 000 entiers aléatoires à partir de ce site. Les premières entrées de cette séquence sont les suivantes :

```
72329, 95447, 11130, 52803, 25986, 58390, 84305, 98618, 54545, 64850,
27412, 15977, 13214, 30421, 91625, 48878, 35783, 58844, 16061, 74799,
99777, 87273, 61979, 77926, 66628, 56546, 19300, 34809, 11633, 86476
```

Et la représentation en 3 dimensions de l'attracteur correspondant est montrée dans la figure 5.5 page 95.

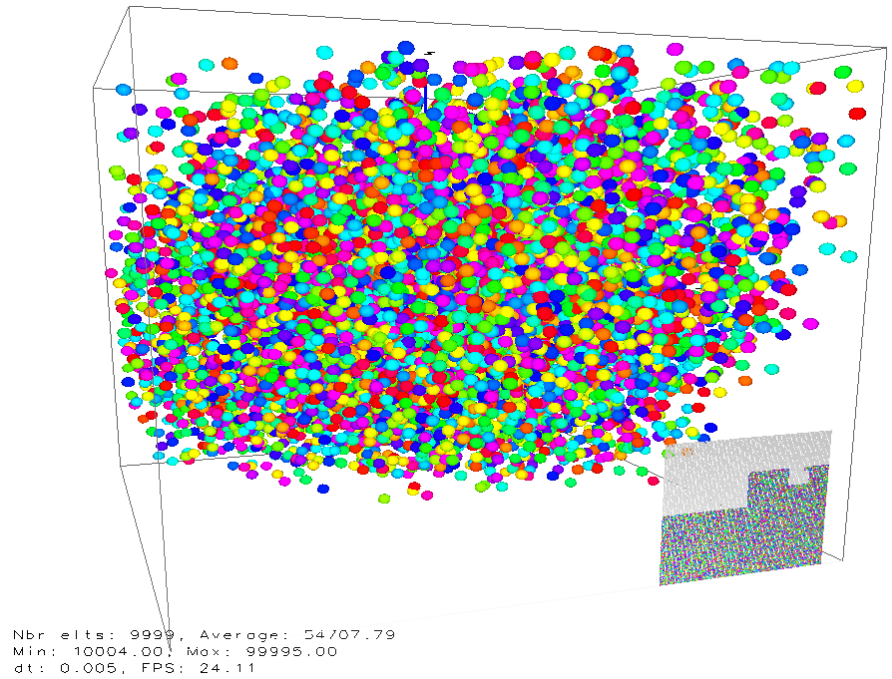


FIGURE 5.5 – Attracteur d'un aléa véritable - vue 45°

La figure 5.5 page 95 montre un environnement en trois dimensions, matérialisé par un cube et les trois axes x en rouge, y en vert et z en bleu. La répartition des points dans cet espace forme un nuage homogène et couvre uniformément un volume sphérique selon les trois axes. Aucun schéma spécifique ne se dégage. Les valeurs fournies sont comprises entre 0 et 99995 avec une valeur moyenne

```
1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3,
2, 3, 8, 4, 6, 2, 6, 4, 3, 3, 8, 3, 2, 7, 9,
5, 0, 2, 8, 8, 4, 1, 9, 7, 1, 6, 9, 3, 9, 9,
3, 7, 5, 1, 0, 5, 8, 2, 0, 9, 7, 4, 9, 4, 4,
5, 9, 2, 3, 0, 7, 8, 1, 6, 4, 0, 6, 2, 8, 6,
2, 0, 8, 9, 9, 8, 6, 2, 8, 0, 3, 4, 8, 2, 5,
        3, 4, 2, 1, 1, 7, 0, 6, 7, 9
```

FIGURE 5.6 – Les 100 premières décimales de Pi

de 54707.79. Dans le coin droit, nous avons la représentation de l'ensemble des données à l'aide de la courbe de Hilbert.
De plus, le programme que nous utilisons, attribue une couleur spécifique à chaque point : l'ensemble des couleurs de la palette graphique est réparti sur l'ensemble des points de façon chronologique. Ainsi, le premier point sur la liste reçoit la première couleur de la palette, le deuxième point reçoit la deuxième couleur et ainsi de suite jusqu'au dernier point. Ce principe nous permet de rajouter une quatrième dimension à notre graphe : le temps.
Sur notre courbe la répartition des couleurs semble complètement aléatoire.

5.1.3.2 Les décimales de Pi

Le nombre Pi est une constante mathématique définie par le ratio de la circonférence d'un cercle et de son diamètre. Pi est communément approximé par 3.14159265. Étant un nombre irrationnel, Pi ne peut pas être exprimé exactement sous la forme d'une fraction. Enfin, les décimales de Pi semblent être distribuées de façon aléatoire, cependant aucune preuve de cet état de fait n'a été découverte [212].
Si nous prenons les 100 premières décimales de Pi, nous obtenons une séquence d'entiers décrite dans la figure 5.6 page 96. La représentation dans un environnement en trois dimensions de cette séquence donne l'attracteur de la figure 5.7 page 97.
Dans la figure 5.7, nous pouvons voir que la courbe de Hilbert montre une distribution des couleurs qui semble aléatoire. En revanche, la représentation en trois dimensions montre un nuage de points proche de l'aléa véritable mais avec des alignements ordonnés. Il semble que les décimales de Pi ne sont pas véritablement aléatoire.
En fait, cet effet d'alignement ordonné est dû à la limite des décimales de Pi. En effet ces dernières sont comprises entre 0 et 9 et les différentes combinaisons possibles de ces décimales dans les soustractions que nous faisons pour calculer les cordonnées x, y et z sont limitées. En les regroupant par 4, nous obtenons une répartitions des résultats qui s'étage entre 0 et 9999. Le résultat obtenu est donc plus pertinent. La figure 5.8 montre alors un nuage parfaitement aléatoire. On a donc une première approche de la confirmation de l'hypothèse de l'aléa des décimales du nombre Pi.

5.1.3.3 Générateur de congruence linéaire

Le générateur de congruence linéaire produit une séquence pseudo-aléatoire de nombres entiers x_1, x_2, x_3... en fonction de la récurrence linéaire suivante [142, 124] :

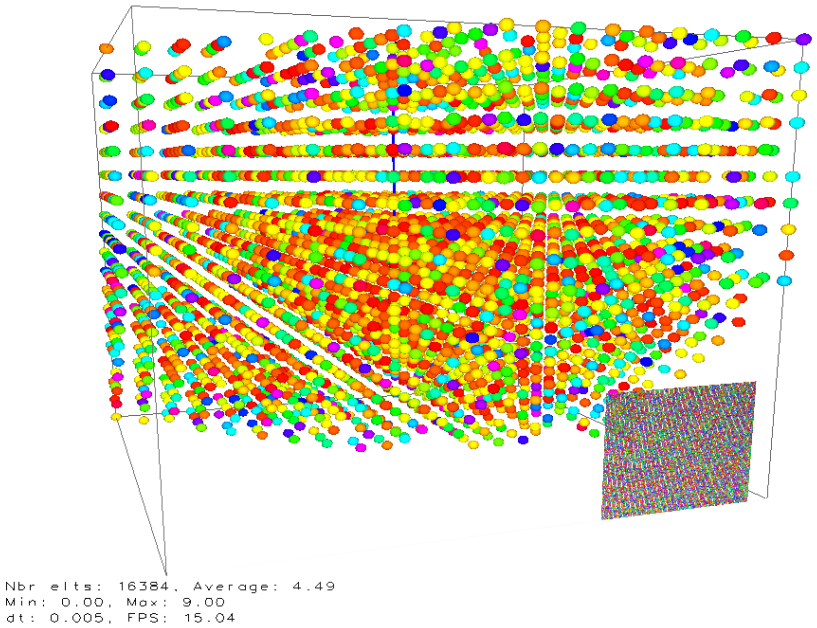


FIGURE 5.7 – Attracteur des décimales de Pi - vue 45°

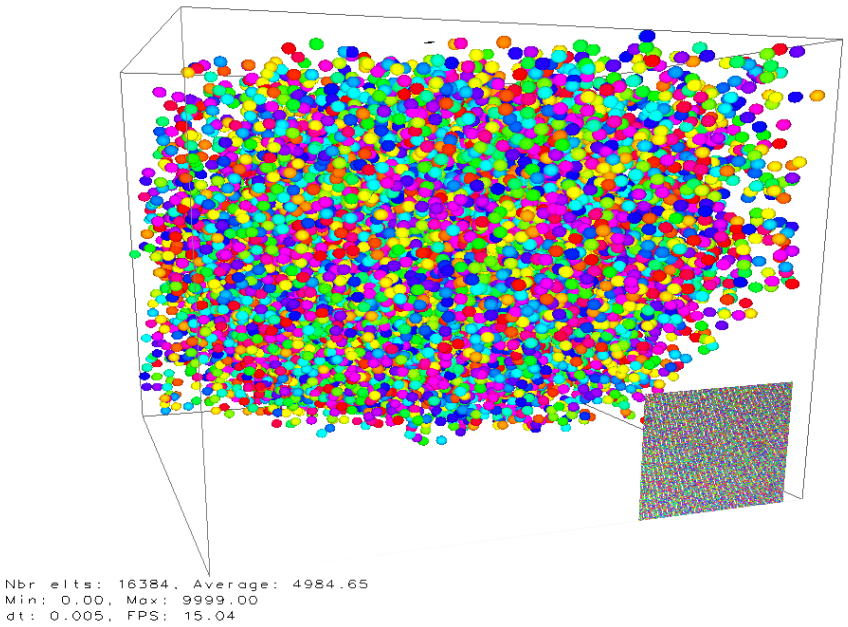


FIGURE 5.8 – Attracteur des décimales de Pi (regroupement par 4) - vue 45°

$$x_{n+1} = ax_n + b \pmod{m} \quad | \quad n \geq 0$$

Les entiers a, b et m sont les paramètres qui caractérisent le générateur et x_0 est la graine. Un exemple de séquence produite avec $a = 5$, $b = 3$ et $m = 4096$ est donné ci-dessous :

```
772, 3863, 2934, 2385, 3736, 2299, 3306, 149, 748, 3743, 2334, 3481,
1024, 1027, 1042, 1117, 1492, 3367, 454, 2273, 3176, 3595, 1594, 3877,
3004, 2735, 1390, 2857, 2000, 1811, 866, 237, 1188, 1847, 1046, 1137,
1592, 3867, 2954
```

Le code en langage C permettant de coder cette fonction est donné dans la figure 5.9 page 98 et l'attracteur correspondant est présenté dans les figures 5.10 page 98 et 5.11 page 99.

```c
double generateLinearCongruence(void) {
    static double xn = 1;
    double value = xn;
    xn = fmod((5 * xn + 3), 4096); //equivalent to (5 * xn + 3) % 4096
    return value;
}
```

FIGURE 5.9 – Code C pour le générateur de congruence linéaire

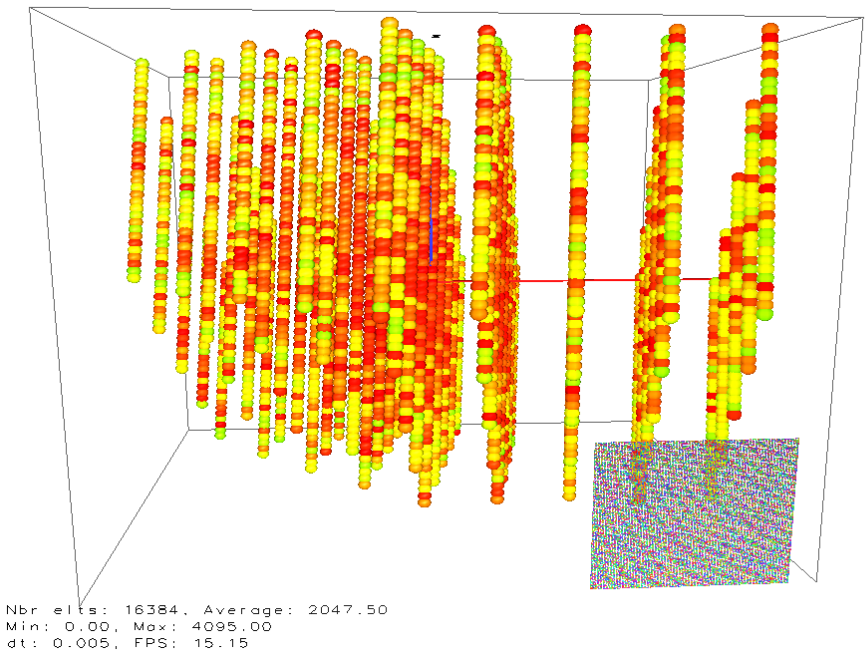


FIGURE 5.10 – Attracteur du générateur de congruence linéaire - axe $y \rightarrow y'$

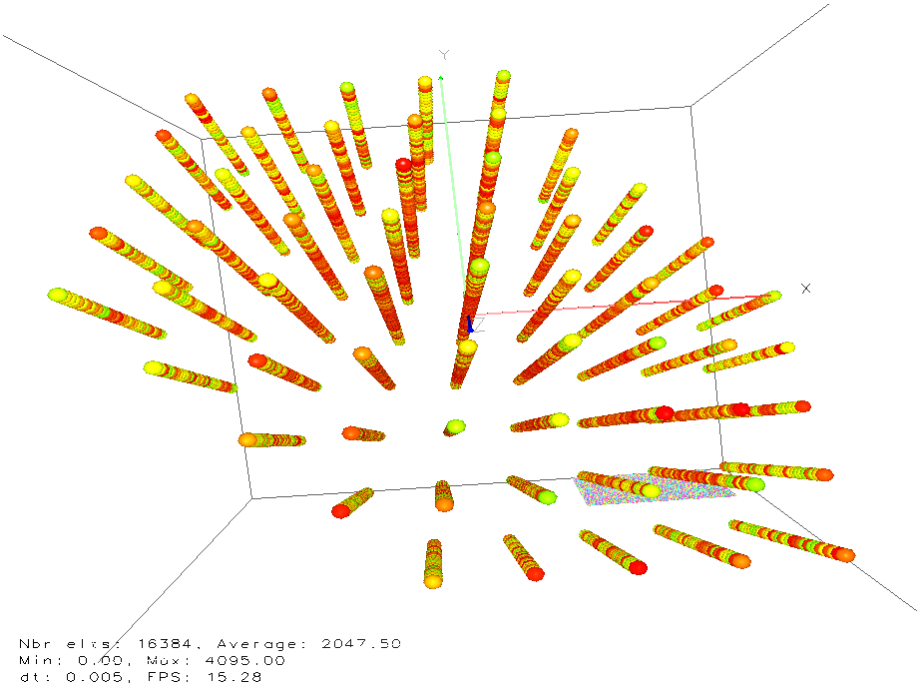


FIGURE 5.11 – Attracteur du générateur de congruence linéaire - axe $z \rightarrow z'$

La taille de l'échantillon est de 16384, la valeur minimale est 0 et la valeur maximale 4095, la valeur moyenne est 2047.50.
La distribution des couleurs dans la courbe de Hilbert semble aléatoire.
Mais, contrairement au générateur de véritable aléa que nous avons pris comme référence, la répartition des points dans les figures 5.10 et 5.11 présente un schéma spécifique sous forme de lignes alignées sur plusieurs plans. Ces lignes correspondent à la période de l'algorithme.
Nous constatons enfin, que seules les couleurs à dominante rouge apparaissent, cela signifie que seule la fin du spectre est visible. Les premiers points sont recouverts par les derniers, l'algorithme fournit donc plusieurs fois les mêmes valeurs.
Ce type de générateur est prédictible et ne convient pas pour un usage cryptographique. En effet, à partir d'une séquence partielle, et sans connaissance des paramètres a, b et m, il est possible de reconstruire le reste de la séquence. De plus, dans ce cas, les deux représentations – en deux et trois dimensions – nous permettent d'affiner la propriété de l'aléa de cet algorithme.

5.1.3.4 Générateur à récurrence non linéaire

Pour notre exemple suivant, nous utilisons un générateur à récurrence non linéaire. Ce dernier produit une séquence pseudo-aléatoire de nombres entiers x_1, x_2, x_3... en fonction de la relation de récurrence suivante [211] :

$$x_{n+1} = \lambda x_n(1 - x_n) \quad | \quad n \geq 0$$

Cette suite dite "logistique" conduit, si $\lambda > 3,56995$, à une suite chaotique. La

suite logistique est utilisée pour modéliser la taille d'une population biologique au fil des générations [140]. Un exemple de séquence produite est donné ci-dessous :

```
451098855224, 451068728783, 451038599037, 451008465983, 450978329622,
450948189954, 450918046977, 450887900692, 450857751097, 450827598193,
450797441977, 450767282451, 450737119614, 450706953464, 450676784002,
450646611227, 450616435137, 450586255734, 450556073016, 450525886982,
450495697633, 450465504967, 450435308985, 450405109684, 450374907066,
450344701129, 450314491873, 450284279297, 450254063401, 450223844184,
450193621646, 450163395785, 450133166603, 450102934097, 450072698268
```

Le code en langage C permettant de coder cette fonction est donné dans la figure 5.12 page 100 et l'attracteur correspondant est présenté dans la figure 5.13 page 101.

```c
double generateLogisticMap(void) {
    static double xn = 0.7364738523; // [0 .. 1]
    static double lambda = 3.8; // > 3,56995
    double value = xn;
    xn = (lambda * xn) * (1 - xn);
    return value;
}
```

FIGURE 5.12 – Code C pour le générateur à récurrence non linéaire

La figure 5.13 page 101 montre également un schéma spécifique ce qui en fait un PRBG non sûr pour une application cryptographique. Cependant, contrairement au générateur de congruence linéaire, nous n'avons plus d'apparition de schéma périodique. Les données générées ne sont pas aléatoires durant les premières itérations (couleur verte) puis semblent le devenir davantage avec l'augmentation du nombre d'itérations.
Dans ce cas, l'étude de la courbe de Hilbert est intéressante parce que la distribution des couleurs semble aléatoire au début mais devient rapidement uniforme à la fin de liste (couleur identique).
Dans notre exemple, la taille de l'échantillon est de 16384, la valeur minimale est 184264892666.96 et la valeur maximale 948850856936.47, la moyenne de l'échantillon est de 631015962902.33.

5.1.3.5 Générateur Blum-Blum-Shub

Le générateur de bits pseudo-aléatoire Blum-Blum-Shub [23] est un PRBG calculatoirement sûr tant que la factorisation de grands nombres composés reste un problème calculatoirement difficile. Ce générateur produit une séquence de bits pseudo-aléatoire selon l'algorithme suivant :

- générer deux grands nombres premiers de Blum p et q et calculer $n = pq$. Un nombre premier de Blum est un nombre premier congruent à 3 modulo 4 ;
- choisir une graine s dans l'intervalle $[1, n-1]$ tel que $\text{pgcd}(s, n) = 1$;
- calculer $x_0 = s^2 \pmod{n}$;
- la séquence est définie comme $x_{i+1} = x_i^2 \pmod{n}$;

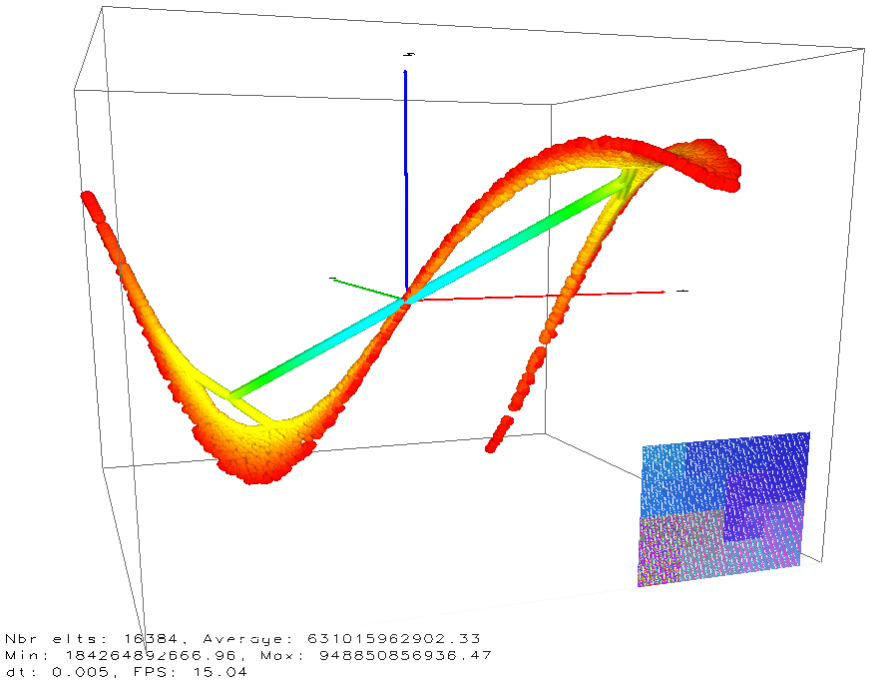


FIGURE 5.13 – Attracteur du générateur à récurrence non linéaire - axe $y \rightarrow y'$

— si x_i est pair alors $z_i = 0$ et si x_i est impair alors $z_i = 1$;
— la sortie de la séquence est z_1, z_2, z_3,...

Le code en langage C permettant de coder cette fonction est donné dans la figure 5.14 page 102 et l'attracteur correspondant est présenté dans la figure 5.15 page 102. Pour faciliter la mise en œuvre de notre présentation, nous n'avons pas appliqué la dernière étape consistant à récupérer le bit de poids faible. Nous constatons que l'attracteur obtenu se répartit sur les trois axes sous la forme d'un nuage homogène. Nous retrouvons la même forme que dans le cas de l'aléa véritable. En fait, le générateur Blum-Blum-Shub est cryptographiquement sûr en supposant l'insolubilité de la résiduosité quadratique [22]. Dans notre exemple, la taille de l'échantillon est de 16384, la valeur minimale est 56782881.0 et la valeur maximale est 512462853845.0, la moyenne de l'échantillon est de 257668663192.64.

5.1.4 Algorithmes cryptographiques

5.1.4.1 Attracteur de RC4

Pour calculer l'attracteur de l'algorithme de RC4, nous utilisons une séquence d'entiers $n = 2^{105} + i$ avec $i = [1 \cdots 60\ 000]$ que nous convertissons en blocs de 128 bits. Ensuite, nous chiffrons chacun de ces blocs avec la même clef de chiffrement de 64 bits :

$$k = 0101010101010101 \tag{5.1}$$

```c
double generateBlumBlumShub(void) {
    // https://en.wikipedia.org/wiki/Blum_Blum_Shub
    // x0 = (s*s) mod(M) avec M=p.q et p, q nombre de Blum
    // p = 869393, q = 589497, n = pq = 512504565321
    // s in [1, n-1] -> s= 412504565321
    static double xn = (412504565321 * 412504565321) % 512504565321;
    double value = xn;
    xn = fmod((xn * xn), 512504565321); // equivalent to (xn * xn) % M
    return value;
}
```

FIGURE 5.14 – Code C pour le générateur Blum-Blum-Shub

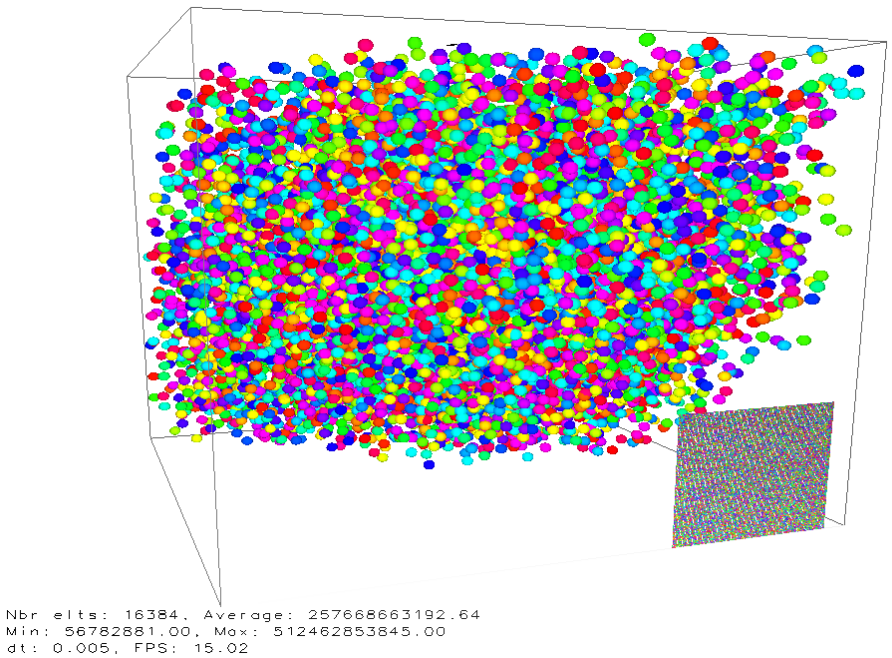


FIGURE 5.15 – Attracteur du générateur Blum-Blum-Shub - vue 45°

Finalement, nous convertissons le résultat en entiers. Ainsi, nous partons d'une séquence linéaire d'entiers pour obtenir une séquence pseudo aléatoire d'entiers. La liste suivante montre le résultat obtenu pour les huit premières entrées de la séquence :

$$
\begin{aligned}
RC4_k(00000200000000000000000000000001) &=_{16} (06080c0e18202929393349576676878 2)\\
&=_{10} (80171508551538131176008731931814480 66)\\
RC4_k(00000200000000000000000000000002) &=_{16} (06080c0e18202929393349576676878 1)\\
&=_{10} (80171508551538131176008731931814480 65)\\
RC4_k(00000200000000000000000000000003) &=_{16} (06080c0e18202929393349576676878 0)\\
&=_{10} (80171508551538131176008731931814480 64)\\
RC4_k(00000200000000000000000000000004) &=_{16} (06080c0e18202929393349576676878 7)\\
&=_{10} (80171508551538131176008731931814480 71)\\
RC4_k(00000200000000000000000000000005) &=_{16} (06080c0e18202929393349576676878 6)\\
&=_{10} (80171508551538131176008731931814480 70)\\
RC4_k(00000200000000000000000000000006) &=_{16} (06080c0e18202929393349576676878 5)\\
&=_{10} (80171508551538131176008731931814480 69)\\
RC4_k(00000200000000000000000000000007) &=_{16} (06080c0e18202929393349576676878 4)\\
&=_{10} (80171508551538131176008731931814480 68)\\
RC4_k(00000200000000000000000000000008) &=_{16} (06080c0e18202929393349576676878b)\\
&=_{10} (80171508551538131176008731931814480 75)
\end{aligned}
$$

Nous avons développé une suite d'outils spécifiques [81, VisCipher3d] pour générer la liste et l'afficher ensuite dans un environnement en trois dimensions à l'instar de la démarche présentée ci-dessus pour les PRBG. Nous obtenons la figure 5.16.

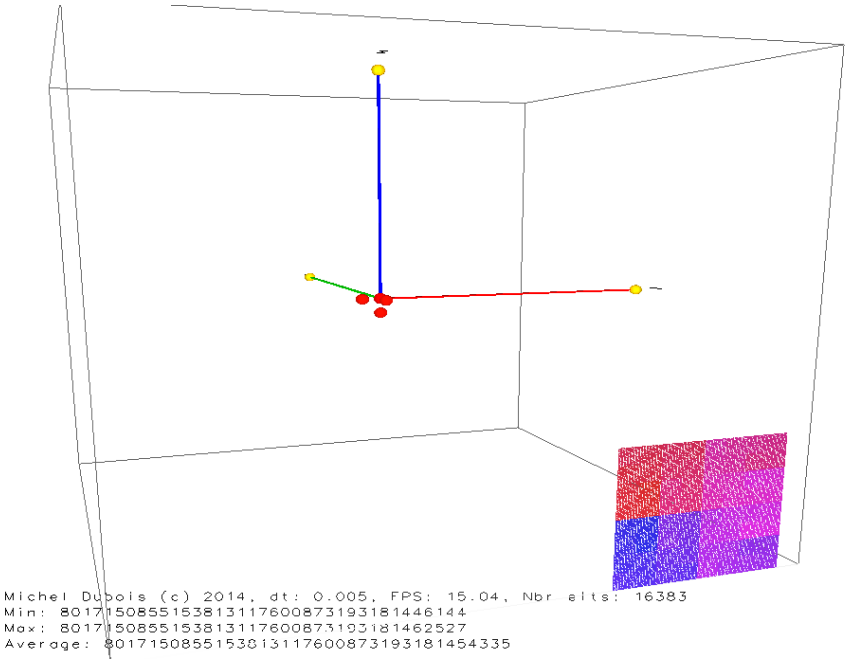


FIGURE 5.16 – Attracteur de RC4 - axe $x \rightarrow x'$

L'attracteur de RC4 est très clairsemé et se résume à quatre boules rouges. La couleur est importante, c'est la couleur de la fin du panel de couleurs. Nous

pouvons en déduire que tous les points de l'attracteur ont les mêmes coordonnées. Nous sommes loin d'une distribution aléatoire dans l'espace comme pour le générateur Blum-Blum-Shub.
Si nous changeons la clef de chiffrement, nous obtenons différents types d'attracteurs : figure 5.17 et figure 5.18. Nous avons donc un lien étroit entre l'attracteur obtenu et la clef de chiffrement utilisée. La représentation de RC4 à l'aide de la courbe de Hilbert montre le même lien entre l'attracteur et la clef utilisée. Plus important encore, la courbe de Hilbert montre que cet algorithme semble loin de l'aléa parfait.

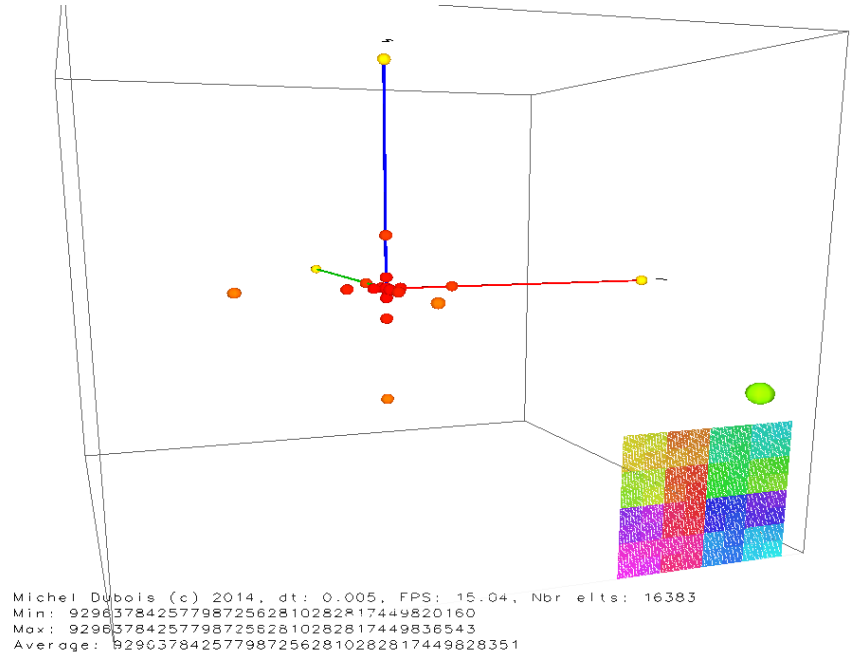


FIGURE 5.17 – Attracteur de RC4 - $k = 5555555555555555$ - axe $x \rightarrow x'$

5.1.4.2 Attracteur de la machine Enigma

La machine Enigma a été inventée par l'ingénieur allemand Arthur Scherbius en 1927 [117]. Le modèle A a été vite abandonné au profit du modèle B, de la taille d'une machine à écrire, puis par une version portable équipée d'indicateurs à lampes avec le modèle C. La machine Enigma et l'entreprise de Scherbius, fondée pour sa commercialisation, vont végéter durant l'entre-deux guerre. La Wehrmacht en achète quelques exemplaires pour évaluation. Ce n'est que lorsque Hitler commence à réarmer l'Allemagne que les experts en cryptologie de la Wehrmacht décident de l'adopter et d'en équiper l'armée allemande.
La machine Enigma est une machine de chiffrement polyalphabétique. Sa première version dispose de 3 rotors comportant chacun 26 positions [142]. Le rotor R_1 tourne chaque fois qu'une touche est pressée, le rotor R_2 tourne selon le mouvement du rotor R_3 qui agit comme un odomètre. Le mouvement des rotors permet de générer une combinaison différente à chaque pression sur une touche du clavier. La clef de chiffrement est définie par la position initiale des 3

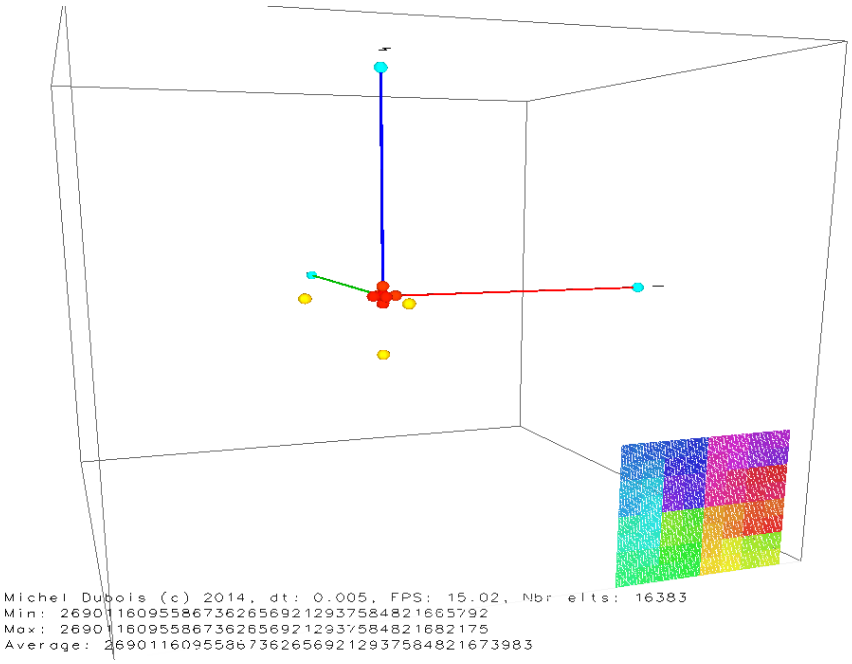


FIGURE 5.18 – Attracteur de RC4 - $k =$ `0f0f0f0f0f0f0f0f` - axe $x \rightarrow x'$

rotors.
Pour calculer l'attracteur de la machine Enigma, nous utilisons une séquence comprenant toutes les combinaisons possibles de 5 caractères de `AAAAA` à `EZZZZ`. Soit $5 \times 26^4 = 2284880$ entrées. Chacune d'entre elles est ensuite chiffrée à l'aide d'un programme reproduisant le fonctionnement d'une machine Engima à trois rotors. La clef utilisée est `AAA`. Le résultat obtenu est ensuite converti en hexadécimal. Ainsi la séquence `AAAAA` devient `FTZMG` $\Rightarrow$ `(0x46545a4d47)` et `EZZZZ` devient `DXILI` $\Rightarrow$ `(0x4458494c49)`.
L'attracteur de la machine Enigma dans un environnement en trois dimensions est présenté dans la figure 5.19 page 106.
L'analyse de l'attracteur de l'Enigma montre un nuage de points aléatoires. Cependant, nous distinguons des alignements et un nuage de points assez épars. Étant donné qu'il y a plus de deux millions de points affichés nous devrions avoir un nuage plus dense.
En zoomant dans le graphique (figure 5.20 page 106), nous voyons que les points sont regroupés en grappes et que leurs couleurs sont similaires. En comptant les points à l'intérieur des grappes, nous trouvons qu'ils contiennent environ 26 points, soit un point pour chaque caractère de l'alphabet. Nous n'avons aucune explication pour ce fait mais une investigation plus approfondie pourrait être intéressante.

5.1.4.3 Attracteur de l'AES

Nous terminons notre comparaison avec l'AES. Pour calculer l'attracteur de l'AES, nous partons de la séquence d'entiers $n = 2^{105}+i$ avec $i = [1 \cdots 1\,000\,000]$ que nous convertissons en blocs de 128 bits. Nous chiffrons ensuite chacun de ces

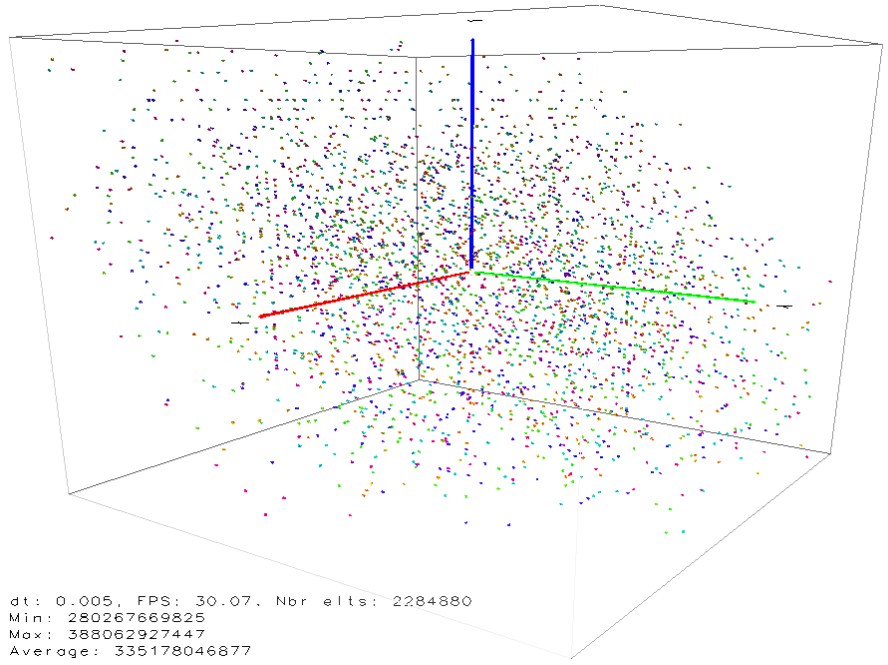


FIGURE 5.19 – Attracteur de la machine Enigma - vue 45°

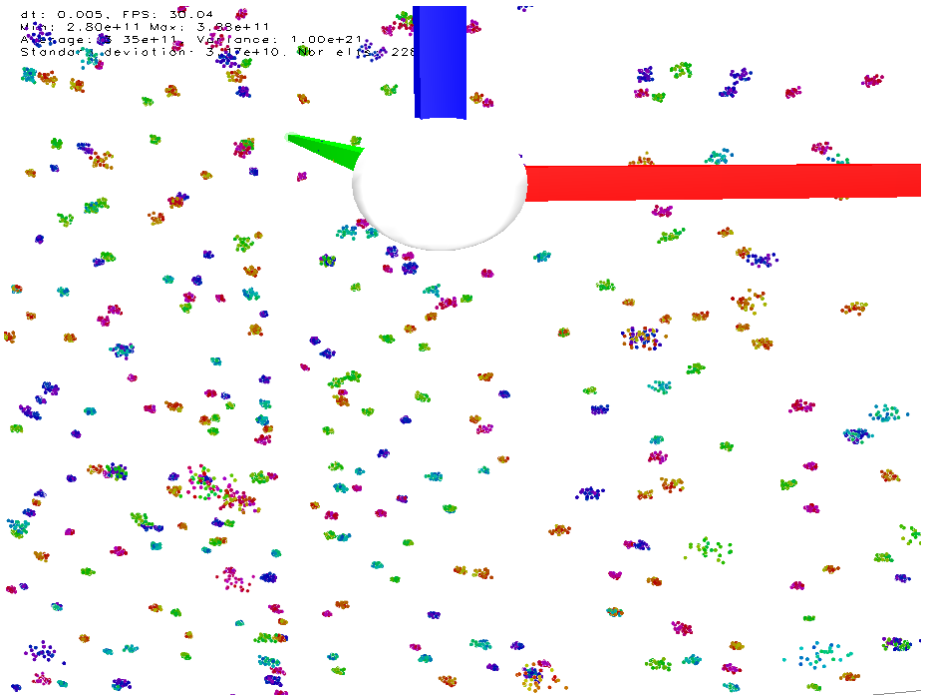


FIGURE 5.20 – Enigma machine attractor - zoomed view

blocs avec la même clef de chiffrement : $k =$ `80000000000000000000000000000000`
Finalement, nous convertissons le résultat obtenu en nombres entiers. Nous partons donc d'une séquence linéaire d'entiers pour obtenir une séquence pseudo-aléatoire d'entiers. La liste de la figure 5.21 page 107 montre le résultat obtenu pour les 16 premières entrées de la séquence.

$$
\begin{aligned}
AES_k(00000200000000000000000000000001) &= (\text{d557cc9f49fe887227b4b9ddece84715})_{16} \\
&= (283581443160616712863761136161128990485)_{10} \\
AES_k(00000200000000000000000000000002) &= (\text{09c7c5a6ebf426a5f19e03bd9040e79d})_{16} \\
&= (13000327896503708020216929648547784605)_{10} \\
AES_k(00000200000000000000000000000003) &= (\text{5f8fc8bf51275ac26522097cb3035f0b})_{16} \\
&= (127023229689953121336747928769328799499)_{10} \\
AES_k(00000200000000000000000000000004) &= (\text{c966ecf33d34f92dc353498769996462})_{16} \\
&= (267709247352391222087383818398701544546)_{10} \\
AES_k(00000200000000000000000000000005) &= (\text{09d800a81f3caa9871aa6ee1ee32ed4e})_{16} \\
&= (130846014035064444883418091784767358 22)_{10} \\
AES_k(00000200000000000000000000000006) &= (\text{723e5071689e7a932405c6d351a87eed})_{16} \\
&= (151855545502638255213403946355100516077)_{10} \\
AES_k(00000200000000000000000000000007) &= (\text{56cfe7cf8cc63cbc4a527ae09957a949})_{16} \\
&= (115393114767635113880023375849033541961)_{10} \\
AES_k(00000200000000000000000000000008) &= (\text{f7ca0f9ad9dfc972e5890cab472d3476})_{16} \\
&= (329368475429008126664105441219366958198)_{10} \\
AES_k(00000200000000000000000000000009) &= (\text{719e9057c4acf3249ee8d48cbeefabb1})_{16} \\
&= (151026074048045206361522544693195287473)_{10} \\
AES_k(0000020000000000000000000000000a) &= (\text{ff6e305fdcebebc447f0dc37c25a86e0})_{16} \\
&= (339525272730300705870076126914789672672)_{10} \\
AES_k(0000020000000000000000000000000b) &= (\text{834edd9eddbf15c1a77c1a99e909ede3})_{16} \\
&= (174538361601988974922403086931709259235)_{10} \\
AES_k(0000020000000000000000000000000c) &= (\text{5344aa331536075f86295762f218bff4})_{16} \\
&= (110682451893361796760390205723301691380)_{10} \\
AES_k(0000020000000000000000000000000d) &= (\text{1b7e76767b1447b707228becfdf25754})_{16} \\
&= (365457880017156978186058074066519550 28)_{10} \\
AES_k(0000020000000000000000000000000e) &= (\text{6907210e0726b85b1c81fe6c0335577b})_{16} \\
&= (139605956066350310999159704129570690939)_{10} \\
AES_k(0000020000000000000000000000000f) &= (\text{e62909d098b6985395aa59b0ae3635a9})_{16} \\
&= (305935522270137279866245407329373009321)_{10}
\end{aligned}
$$

FIGURE 5.21 – Premières entrées de la séquence pour l'AES

En utilisant les mêmes outils que pour l'algorithme de RC4, nous obtenons les figures 5.22 page 108 et 5.23 page 108.
À la différence du générateur Blum-Blum-Shub nous constatons que l'attracteur de l'AES prend la forme d'un nuage uniformément réparti sur les trois axes sous la forme de cubes. Il diffère, en cela, de l'attracteur de l'aléa véritable. Néanmoins, nous pouvons raisonnablement dire que l'AES est un algorithme de chiffrement conçu comme un générateur de bits pseudo-aléatoire, proche de l'aléa véritable.
Après avoir analysé graphiquement le comportement de l'AES, nous allons chercher à voir comment l'algorithme se comporte avec des clefs de chiffrement différentes. Pour cela, nous avons reproduit le processus décrit ci-dessus en utilisant la même séquence initiale et en la chiffrant avec les deux clefs :

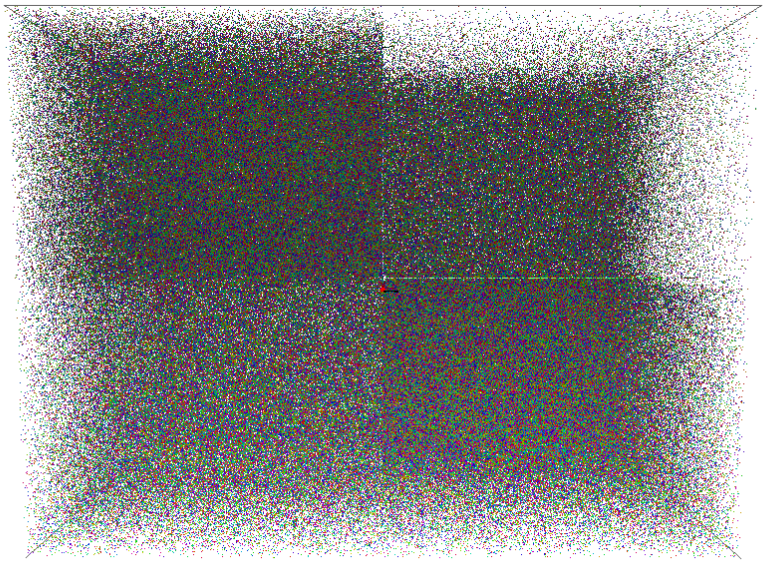


FIGURE 5.22 – Attracteur de l'AES - axe $x \rightarrow x'$

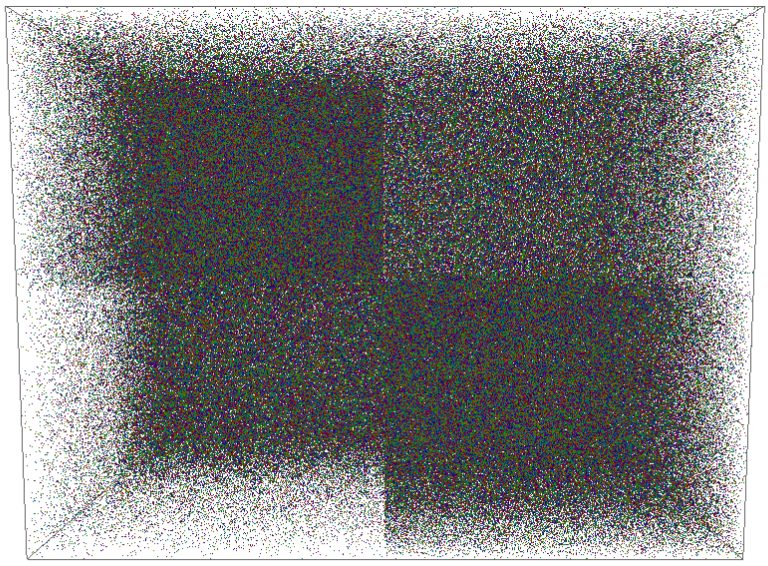


FIGURE 5.23 – Attracteur de l'AES - vue 45°

$$k_1 = \texttt{80000000000000000000000000000000} \tag{5.2}$$

$$k_2 = \texttt{40000000000000000000000000000000} \tag{5.3}$$

Ces deux clefs diffèrent de seulement 1 bit sur les 128 bits les composant.
En affectant ensuite une couleur différente à chaque attracteur, nous pouvons les comparer deux à deux. Le résultat obtenu est présenté dans les figures 5.24 page 109 et 5.25 page 110. Pour générer ces figures nous avons calculé le chiffrement d'une séquence de 450000 blocs identiques.

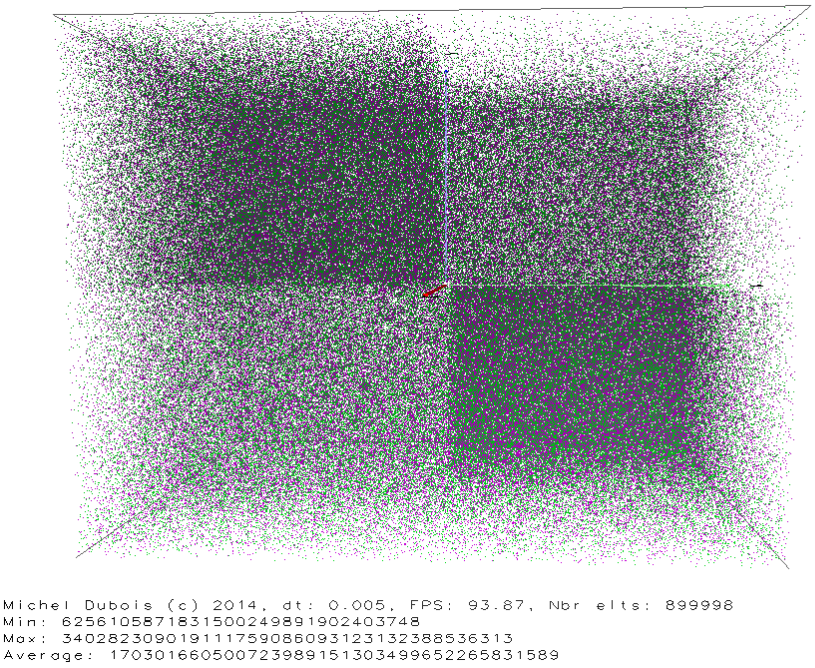


FIGURE 5.24 – Attracteurs de deux chiffrements par l'AES avec des clefs différentes - axe $z \rightarrow z'$

En effectuant la comparaison de ces deux attracteurs, nous constatons qu'ils s'imbriquent étroitement en couvrant uniformément les trois dimensions, toujours en gardant cette répartition en cubes. Cependant, il n'est pas possible de distinguer une possible corrélation entre ces deux ensembles d'éléments. Afin de mieux visualiser une éventuelle corrélation entre les deux ensembles de blocs chiffrés, nous allons représenter le même graphe mais en joignant deux à deux les points correspondants à la même entrée. Ainsi, le point représentant le bloc chiffré avec la clef k_1 du premier bloc de la séquence sera relié au point correspondant au bloc chiffré avec la clef k_2 du même bloc en clair.
Le résultat obtenu est présenté dans les figures 5.26 page 110 et 5.27 page 111. Afin que les graphes soient lisibles, nous avons réduit la taille de l'échantillon à 250 entrées. Le tracé des droites reliant les points est désordonné et ne révèle aucun schéma identifiable.

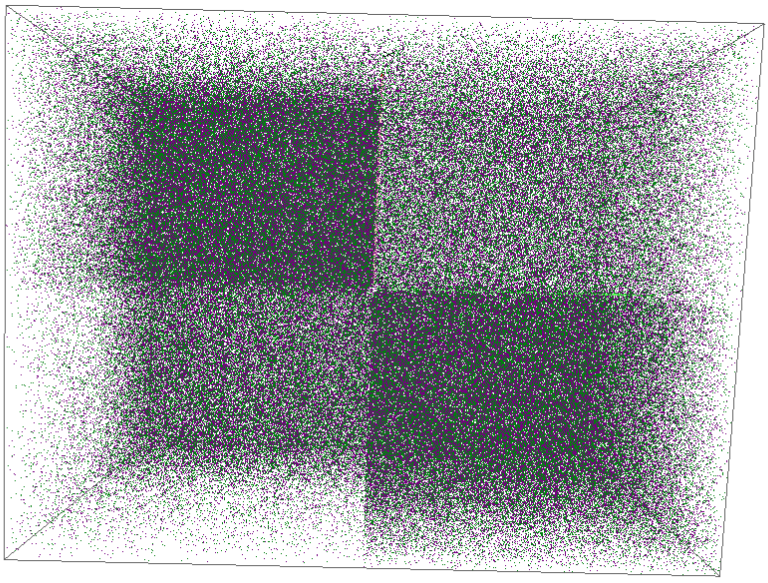


FIGURE 5.25 – Attracteurs de deux chiffrements par l'AES avec des clefs différentes - vue 45°

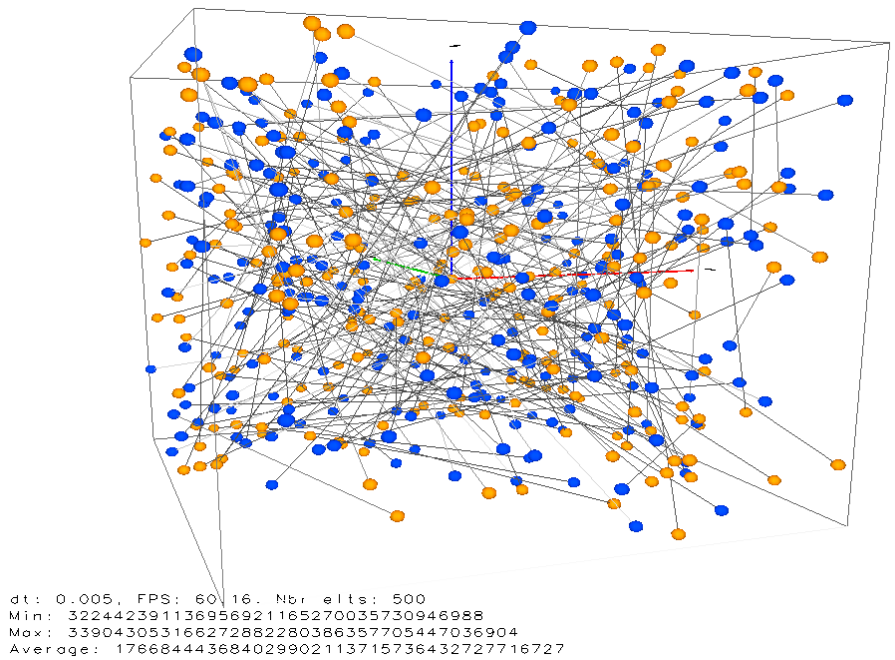


FIGURE 5.26 – Corrélation entre deux attracteurs de l'AES - vue 45°

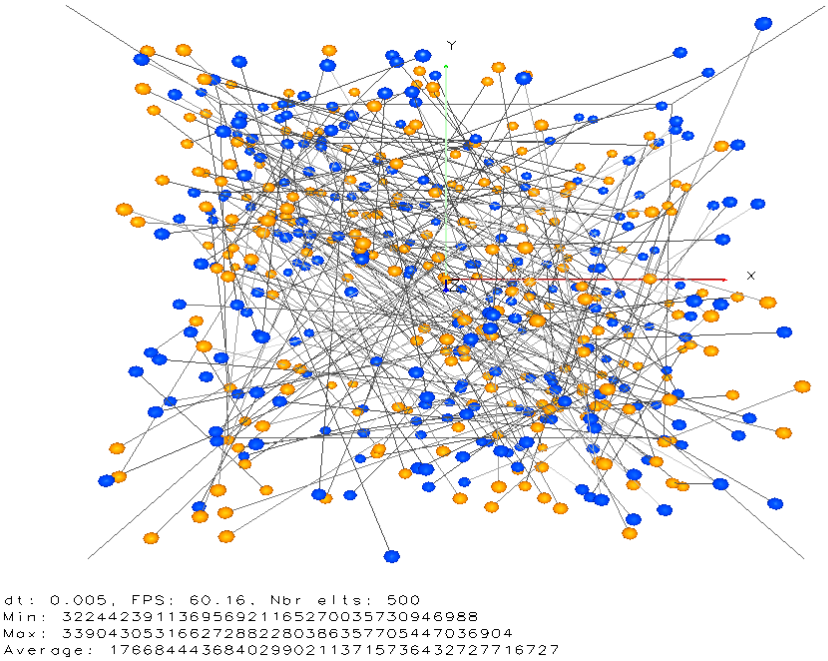


FIGURE 5.27 – Corrélation entre deux attracteurs de l'AES - axe $z \rightarrow z'$

Nous pouvons en déduire raisonnablement qu'il n'y a pas de corrélation entre le chiffrement d'un même bloc de clair avec des clefs de chiffrement différentes. Le calcul des distances de Minkowski [209], avec $p = 2$, pour chaque point du graphe confirme ce résultat. En effet, la liste des distances, dont les premières entrées sont présentées ci-dessous, se comportent comme une suite aléatoire. L'attracteur résultant de la représentation de ces distances en trois dimensions (voir figure 5.28 page 112) a la forme d'un nuage dont la forme spécifique est proche de celle d'un losange. Il serait intéressant de chercher à comprendre ce qui justifie cette forme, mais ce n'est pas l'objet de ce travail.

```
12026847709294981120, 12026869209288865792, 19971070890283577344,
26113153353646714880, 25972774069680922624, 14954846745435789312,
7391670208430327808, 16736310912763425 28, 2444035869612603904,
2802878496097698304, 24268436007518049 28, 17531236510957637 12,
9810861268161230848, 10326031878050832384, 16430236778313869312,
14585735298828013568, 14873569053264308224, 13174990125624895488
```

—=oOo=—

Notre analyse de l'aléa produit par les PRBG et par les algorithmes cryptographiques, bien que visuellement parlante, n'est pas suffisante pour prouver que les séquences produites par ces algorithmes sont statistiquement aléatoires. Cependant, la visualisation de l'aléa, tel que nous l'avons exposé, présente deux

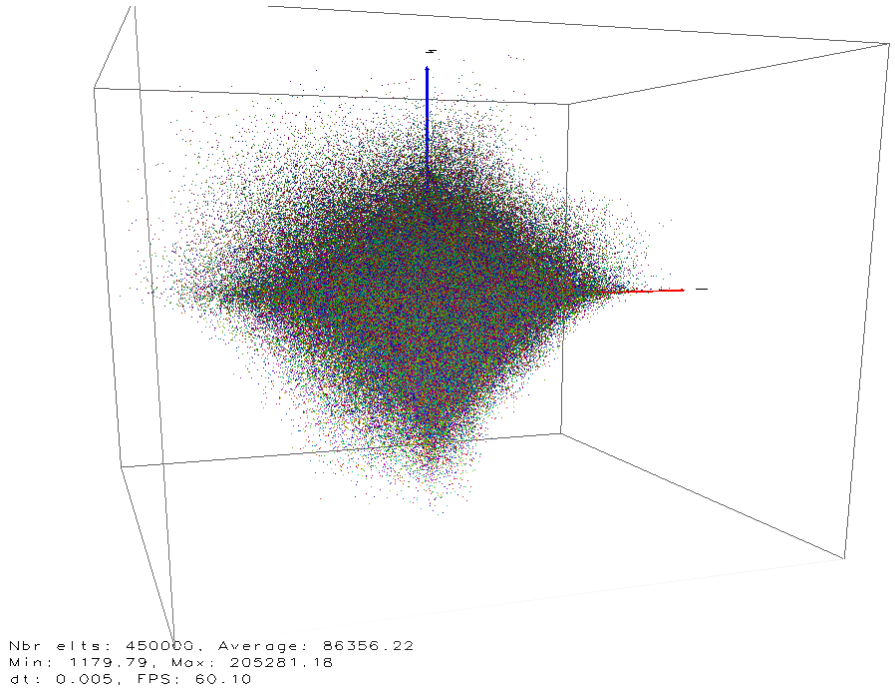


FIGURE 5.28 – Attracteur des distances entre deux processus de chiffrement avec l'AES - vue 45°

avantages intéressants : le premier est qu'elle permet de détecter des patterns et donc ici des problèmes puisque l'on cherche de l'aléatoire, et la deuxième réside dans le fait que notre cerveau ayant plus de facilité pour distinguer rapidement deux images différentes, notre approche nous permet plus facilement de visualiser et différencier un PRBG d'un autre. Ainsi, juste en visualisant dans des environnements en deux et trois dimensions la sortie d'une séquence aléatoire, nous avons la possibilité de détecter d'éventuels patterns et d'identifier le PRBG utilisé pour générer cette séquence. Cette approche nous permet également de réaliser une première analyse rapide d'un potentiel biais sur des algorithmes cryptographiques inconnus.
Ce travail a fait l'objet de publications en France [89] et à l'international [85]. Ce dernier article a reçu le prix du meilleur article et de la meilleure présentation. De nombreux tests statistiques existent pour définir ce qui peut être considéré comme aléatoire. Parmi ces tests, nous pouvons citer le test de fréquence, le test de série, le test d'auto-corrélation, le test statistique universel de Maurer [139],le test de Kolmogorov-Smirnov [135] ou encore la suite de tests du NIST [175]. Aujourd'hui, nous pouvons considérer que les algorithmes de chiffrement symétriques sont statistiquement fiables, dans le sens où ils ont passé avec succès les tests statistiques connus. Le travail du cryptanalyste consiste donc maintenant a trouver un biais statistique exploitable, lié à un défaut de conception inconnu. Dans ce contexte, son objectif est, à partir de nouvelles hypothèses, de développer de nouveaux tests statistiques.

5.2 Test statistique de Möbius

Le test statistique de Möbius est un nouveau test statistique présenté dans [97, 100] et s'appuyant sur le nombre de monômes de degré exactement égal à d dans la forme algébrique normale de toutes les fonctions booléennes modélisant chaque bit produit par un algorithme de chiffrement. Ce test permet de déterminer si ces fonctions booléennes sont réellement aléatoires.

5.2.1 Présentation du test

Le théorème de base sur lequel s'appuie le test statistique de Möbius est le suivant :

Théorème 5.1. Soit la forme algébrique normale d'une fonction booléenne $f : \mathbb{B}_2^n \to \mathbb{B}_2$. Le nombre n de monômes de degré d de f a une distribution normale avec une valeur moyenne E et une variance V donnée par :

$$E_n = \frac{1}{2}C_d^n \quad \text{et} \quad V_n = \frac{1}{4}C_d^n$$

Ainsi, pour des fonctions booléennes de $\mathbb{B}_2^n \to \mathbb{B}_2$ avec $n = (2, 4, 8, 16)$, nous obtenons le tableau de la figure 5.29 page 114.
À titre d'exemple, nous avons généré 16 fonctions booléennes aléatoires de $\mathbb{B}_2^{16} \to \mathbb{B}_2^{16}$. Ensuite, pour chacune de ces fonctions, nous avons calculé leur transformée de Möbius et obtenu ainsi 256 fonctions booléennes de $\mathbb{B}_2^{16} \to \mathbb{B}_2$. Enfin, dans chacune de ces fonctions, nous avons compté le nombre de monômes pour chaque degré $d \in (1, 2, \cdots, 16)$. Le résultat obtenu est formalisé, sous forme de graphe, dans les figures 5.30 page 114 et 5.31 page 115. Nous pouvons constater que ces fonctions suivent bien la répartition décrite dans le théorème 5.1 et formalisée dans le tableau 5.29 page 114.
En partant de l'hypothèse que le nombre de monômes de degré d est distribué selon le théorème 5.1, le test statistique de Möbius affirme que si un algorithme cryptographique passe ce test, alors, il ne présente pas de biais statistique structurel.

5.2.2 Application au mini-AES

Le mini-AES est un algorithme de chiffrement reprenant les mêmes algorithmes que l'AES mais sur 16 bits seulement (Cf. figure 4.10 page 68). Cette taille de bloc et de clef représente un avantage, dans notre cas, puisqu'il est aisé de construire les tables de vérité de l'ensemble d'un tour de chiffrement. En effet, l'espace des clefs et des blocs sont de taille suffisamment réduite pour être traité par un ordinateur conventionnel.

5.2.2.1 Premier tour du mini-AES

Pour rappel, nous avons défini plus haut, que la fonction X_1, décrivant le premier tour du mini-AES, résulte de la conjonction des fonctions `NibbleSub` $NS()$, `ShiftRow` $SR()$ et `MixColumn` $MC()$ de telle façon que nous ayons :

$$X_1(B) = MC \circ SR \circ NS(B)$$

Degré	Taille de bloc							
	2		4		8		16	
	E_2	V_2	E_4	V_4	E_8	V_8	E_{16}	V_{16}
0	0.5	0.25	0.5	0.25	0.5	0.25	0.5	0.25
1	1.0	0.5	2.0	1.0	4.0	2.0	8.0	4.0
2	0.5	0.25	3.0	1.5	14.0	7.0	60.0	30.0
3			2.0	1.0	28.0	14.0	280.0	140.0
4			0.5	0.25	35.0	17.5	910.0	455.0
5					28.0	14.0	2184.0	1092.0
6					14.0	7.0	4004.0	2002.0
7					4.0	2.0	5720.0	2860.0
8					0.5	0.25	6435.0	3217.5
9							5720.0	2860.0
10							4004.0	2002.0
11							2184.0	1092.0
12							910.0	455.0
13							280.0	140.0
14							60.0	30.0
15							8.0	4.0
16							0.5	0.25

FIGURE 5.29 – Répartition du nombre de monômes en fonction de la taille de bloc

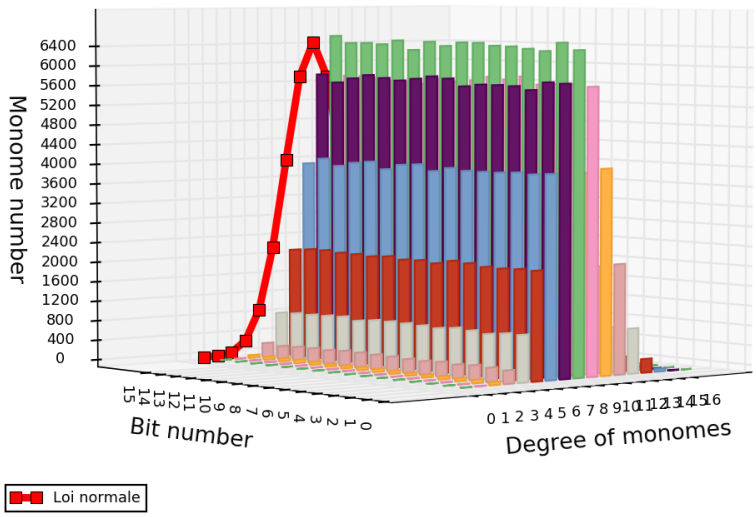


FIGURE 5.30 – Distribution des monômes de degré d (vue 1)

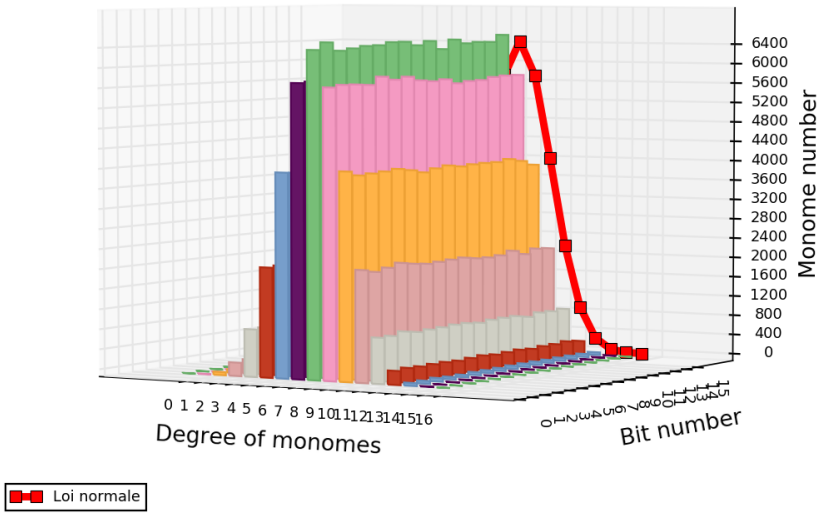


FIGURE 5.31 – Distribution des monômes de degré d (vue 2)

Après avoir généré la table de vérité des fonctions booléennes décrivant chacun des bits de sortie de X_1 et calculé les transformées de Möbius correspondantes, nous obtenons les 16 formes algébriques normales décrivant X_1.
À partir de là, nous pouvons établir le tableau 5.32 page 116 décrivant la répartition du nombre de monômes en fonction du degré d. Le degré maximal des monômes est $d = 3$.
La représentation graphique de la distribution des degrés des monômes de X_1 est donnée dans les figures 5.33 page 116 et 5.34 page 117.
L'analyse du tableau et des figures montrent que le premier round du mini-AES ne passe pas le test statistique de Möbius. En effet, la répartition des monômes n'est pas conforme à celle de notre cas de référence (voir fig. 5.30 p. 114) aussi bien en regardant bit par bit mais également en suivant l'évolution par degré de monômes sur les 16 bits. Par ailleurs, sur ce dernier point, nous constatons une évolution périodique du nombre de monôme de même degré sur les 16 bits.

5.2.2.2 Procédure d'expansion de la clef

Comme pour le premier tour du mini-AES appliquons le test statistique de Möbius à sa procédure d'expansion de la clef. Nous avons précédemment défini les trois fonctions K_1, K_2 et K_3 décrivant le processus de dérivation de la clef tel que, à partir du bloc de clef de 16 bits $K = (k_1, \cdots, k_{16})$, nous ayons les clefs utilisées dans les rounds $k_i = (k_{i,1}, \cdots, k_{i,16})$ avec $i \in (1, 2, 3)$. Nous allons nous intéresser aux fonctions booléennes de la fonction K_3.
En appliquant la même démarche que pour le premier tour, nous obtenons 16 équations à partir desquelles nous pouvons établir le tableau 5.35 page 118 décrivant la répartition du nombre de monômes en fonction du degré d. Le degré

Bit	Nombre de monômes de degrés :			
	0 ($E = 0, 5$)	1 ($E = 8, 0$)	2 ($E = 60, 0$)	3 ($E = 280, 0$)
0	1	3	8	4
1	1	6	9	5
2	1	4	6	6
3	0	6	6	5
4	1	3	8	4
5	1	6	9	5
6	1	4	6	6
7	0	6	6	5
8	1	3	8	4
9	1	6	9	5
10	1	4	6	6
11	0	6	6	5
12	1	3	8	4
13	1	6	9	5
14	1	4	6	6
15	0	6	6	5

FIGURE 5.32 – Tableau de distribution des degrés des monômes pour la fonction X_1 du mini-AES

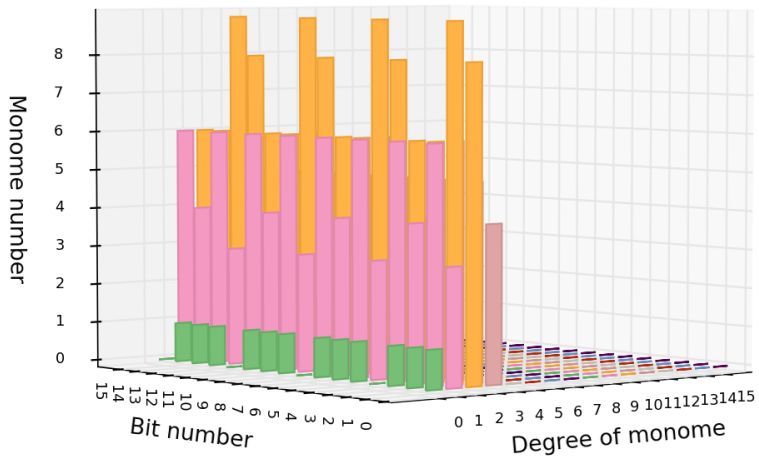


FIGURE 5.33 – Distribution des monômes de degré d pour la fonction X_1 du mini-AES (vue 1)

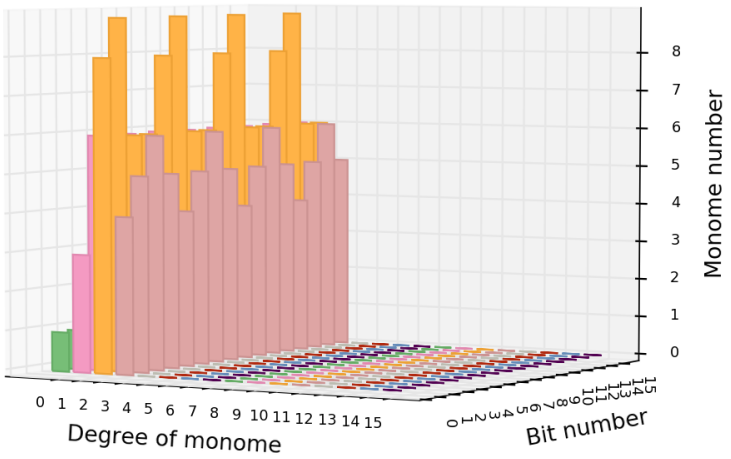


FIGURE 5.34 – Distribution des monômes de degré d pour la fonction X_1 du mini-AES (vue 2)

maximal des monômes est $d = 5$.
La représentation graphique de la distribution des degrés des monômes de K_3 est donnée dans les figures 5.36 page 118 et 5.37 page 119.
Comme pour la fonction X_1, la distribution des degrés des monômes montre de nombreuses disparités par rapport à la distribution normales des monômes de notre cas de référence (voir fig. 5.30 p. 114). De même, comme pour la fonction X_1, nous constatons la présence d'une évolution périodique du nombre de monômes de même degré sur les 16 bits. La période constatée est de 4, ce qui correspond à la taille d'un nibble, brique de base du mini-AES. Nous sommes donc en présence d'un schéma d'évolution plus proche d'une suite périodique que d'un véritable aléa.
En conclusion, nous pourrions penser que le mini-AES ne passe pas le test statistique de Möbius. En effet, chaque fonction interne de l'algorithme, prise séparément, présente des biais statistiques. En fait, faute de puissance de calcul, il est impossible de modéliser l'intégralité du processus de chiffrement du mini-AES. Nous pensons que ces biais statistiques s'estompent par l'imbrication des fonctions internes de l'algorithme et aussi par le jeu des tours. Cependant, cet algorithme ne répond pas complètement au paradigme exposé au début de ce chapitre [1]. En effet, même si le résultat obtenu à l'issue du processus du chiffrement semble proche de l'aléa véritable, il n'en va pas de même des étapes internes.

1. Le Graal de tout algorithme cryptographique est d'obtenir, à chaque étape interne et à l'issue du processus de chiffrement, une séquence semblant la plus proche possible de l'aléa parfait.

Bit	Nombre de monômes de degrés :					
	0 ($E = 0, 5$)	1 ($E = 8$)	2 ($E = 60$)	3 ($E = 280$)	4 ($E = 910$)	5 ($E = 2184$)
0	1	10	62	153	217	87
1	0	13	45	116	103	51
2	1	13	39	163	153	87
3	0	10	52	116	115	51
4	0	9	64	151	217	87
5	1	13	44	117	103	51
6	0	13	39	163	153	87
7	1	12	52	117	115	51
8	1	9	62	153	217	87
9	0	14	45	116	103	51
10	1	12	39	163	153	87
11	0	9	52	116	115	51
12	0	10	64	151	217	87
13	1	12	44	117	103	51
14	0	12	39	163	153	87
15	1	11	52	117	115	51

FIGURE 5.35 – Tableau de distribution des degrés des monômes pour la fonction K_3 du mini-AES

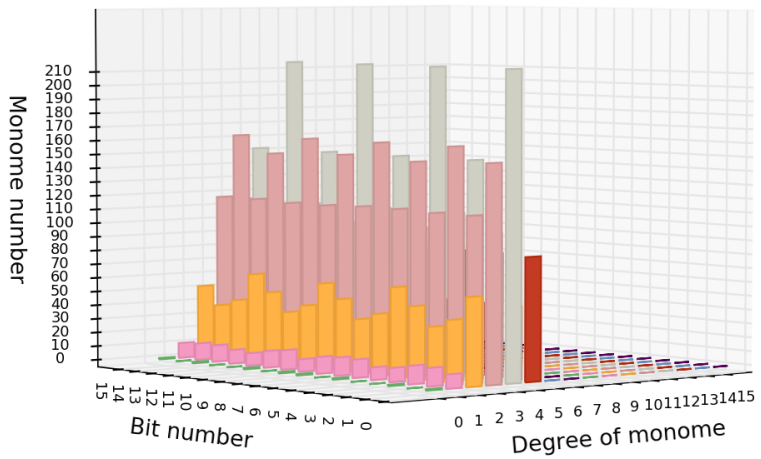


FIGURE 5.36 – Distribution des monômes de degré d pour la fonction K_3 du mini-AES (vue 1)

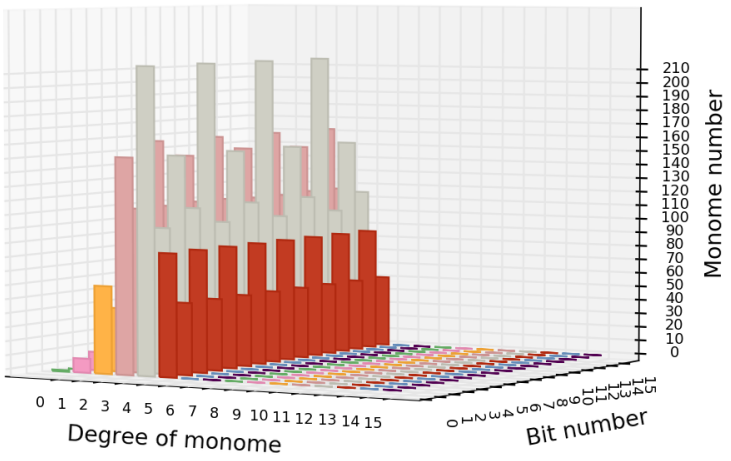


FIGURE 5.37 – Distribution des monômes de degré d pour la fonction K_3 du mini-AES (vue 2)

5.2.3 Application à l'AES

Un algorithme de chiffrement par bloc comme l'AES, utilise une clef K de n bits pour chiffrer un bloc B de m bits. La clef K est elle-même étendue en t clefs de tour de n bits chacune. Cet algorithme peut être représenté par un ensemble de fonctions booléennes tel que le message chiffré C est :

$$C = (c_0, c_1, \cdots, c_{m-1}) = (f_0(K_0, P), f_1(K_1, P), \cdots, f_{n-1}(K_{t-1}, P))$$

Chacune de ces fonctions $(f_0, \cdots, f_{n-1})$ étant des fonctions booléennes $\mathbb{B}_2^{n+m} \rightarrow \mathbb{B}_2$. Dans le cas de l'AES-128 nous avons $n = 128$, $m = 128$ et $t = 10$.
La démarche que nous avons mise en place au début de ce chapitre nous a permis de générer les 128 fonctions booléennes décrivant les bits de sortie de l'*Advanced Encryption Standard*. À partir des fichiers découlant de la forme algébrique normale de ces fonctions, il est maintenant plus aisé d'appliquer le test statistique de Möbius à l'AES.

5.2.3.1 Les fonctions de tour

La fonction `SubBytes` applique une substitution non-linéaire à chaque octet du tableau d'état en utilisant une table de substitution (S-Box).
La représentation de la distribution des degrés des monômes de chacune des 128 formes algébriques normales de la fonction `SubBytes` est détaillée dans les figures 5.38 page 120, 5.39 page 121, 5.40 page 121 et 5.41 page 122. Afin d'obtenir ce résultat, pour chaque équation, nous calculons le nombre de monômes et

leur répartition par degré. Nous obtenons le résultat suivant (présenté sur les 7 derniers bits) :

Bit	Nbr. monômes	Répartition des degrés	Répartition attendue
120	110	$[0, 4, 10, 26, 33, 24, 11, 2, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
121	112	$[1, 3, 10, 31, 30, 24, 11, 2, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
122	114	$[1, 3, 7, 32, 32, 27, 10, 2, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
123	131	$[0, 5, 17, 29, 32, 34, 11, 3, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
124	136	$[0, 4, 11, 29, 39, 33, 16, 4, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
125	145	$[0, 4, 12, 38, 36, 33, 17, 5, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
126	133	$[1, 4, 14, 30, 33, 31, 16, 4, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
127	132	$[1, 4, 16, 30, 33, 30, 15, 3, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$

Nous constatons que sur l'ensemble des monômes, le degré maximal est $d = 7$.

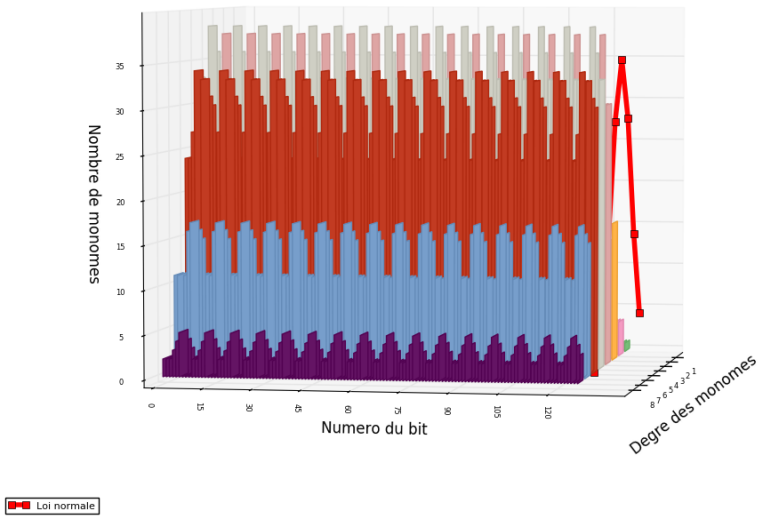


FIGURE 5.38 – Distribution des monômes de degré d pour la fonction SubBytes de l'AES (vue 1)

Afin de déterminer de façon formelle si la fonction `SubBytes` passe le test statistique de Möbius, nous allons tester l'adéquation de la distribution normale des degrés décrite dans le théorème 5.1 et celle observée dans les formes algébriques normales des 128 fonctions booléennes décrivant les bits de sortie de la fonction `subBytes`.
Pour cela, nous calculons, pour chaque degré, la distribution prévue des degrés $d_n = \frac{1}{2}C_d^n$ avec $n = 8$, puis, celle obtenue à partir des 128 formes algébriques normales de la fonction, nous l'appelons $\hat{d}_n$. Ensuite nous calculons la statistique T telle que :

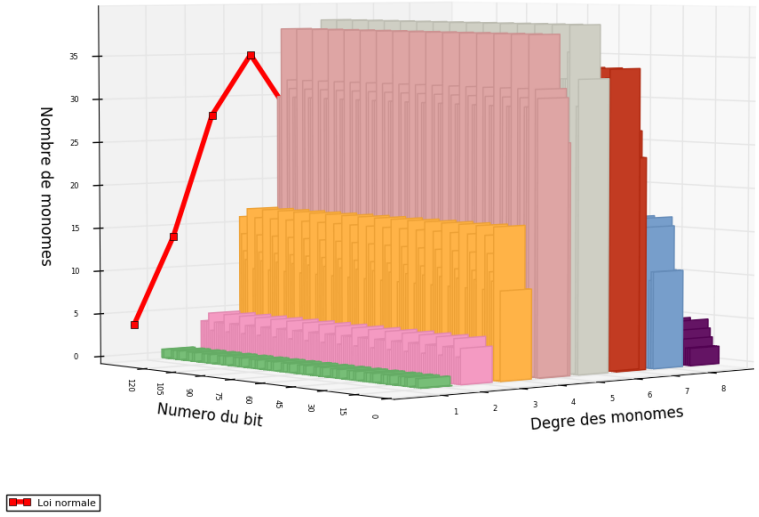


FIGURE 5.39 – Distribution des monômes de degré d pour la fonction SubBytes de l'AES (vue 2)

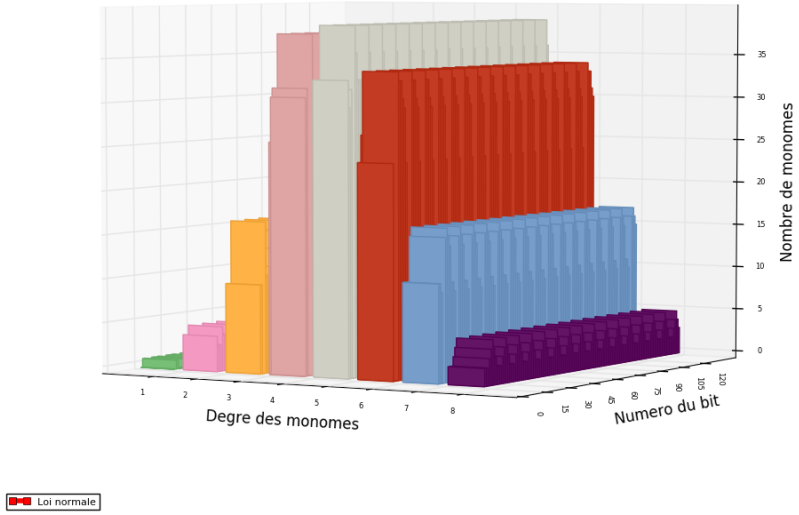


FIGURE 5.40 – Distribution des monômes de degré d pour la fonction SubBytes de l'AES (vue 3)

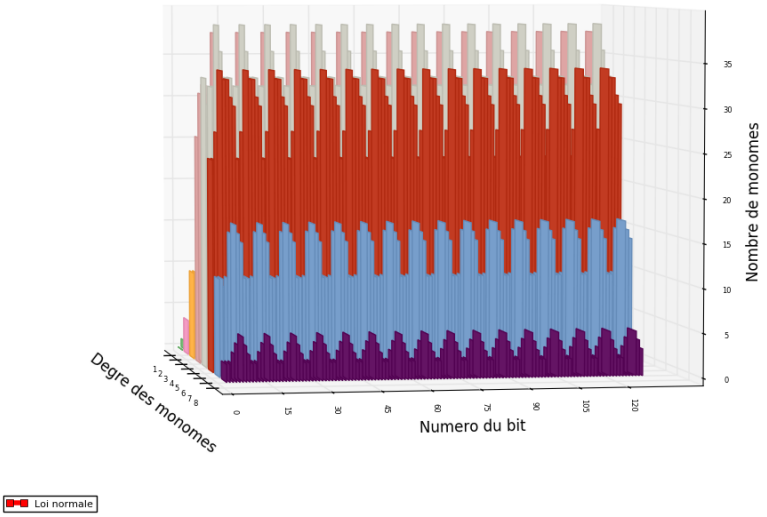


FIGURE 5.41 – Distribution des monômes de degré d pour la fonction SubBytes de l'AES (vue 4)

$$T = \sum_{i=1}^{128} \frac{(\hat{d}_n^i - d_n^i)^2}{d_n^i}$$

Cette statistique suit alors une loi du χ^2 à 7 degrés de liberté dont les valeurs théoriques sont : $\chi^2 = 154,30$ pour $\alpha = 0,050$, $\chi^2 = 166,99$ pour $\alpha = 0,010$, $\chi^2 = 181,99$ pour $\alpha = 0,001$.
Nous obtenons alors le tableau suivant [2] :

Degré	T_i	$\alpha = 0,05$	$\alpha = 0,01$	$\alpha = 0,001$
$d = 0$	64.0	passe	passe	passe
$d = 1$	12.0	passe	passe	passe
$d = 2$	122.29	passe	passe	passe
$d = 3$	79.43	passe	passe	passe
$d = 4$	32.91	passe	passe	passe
$d = 5$	75.43	passe	passe	passe
$d = 6$	69.71	passe	passe	passe
$d = 7$	60.0	passe	passe	passe
$d = 8$	64.0	passe	passe	passe

Si nous appliquons le même raisonnement à la fonction `ShiftRows`, après le calcul du nombre de monômes et leur répartition par degré, nous obtenons le résultat suivant (présenté sur les 7 derniers bits) :

2. Le test est validé si la valeur obtenue pour T est inférieure à la valeur de référence de χ^2.

Bit	Nbr. monômes	Répartition des degrés	Répartition attendue
120	110	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
121	112	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
122	114	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
123	131	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
124	136	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
125	145	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
126	133	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
127	132	$[0, 1, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$

En appliquant ensuite le test statistique de Möbius nous obtenons le tableau suivant :

Degré	T_i	$\alpha = 0,05$	$\alpha = 0,01$	$\alpha = 0,001$
$d = 0$	64.0	passe	passe	passe
$d = 1$	288.0	échec	échec	échec
$d = 2$	1792.0	échec	échec	échec
$d = 3$	3584.0	échec	échec	échec
$d = 4$	4480.0	échec	échec	échec
$d = 5$	3584.0	échec	échec	échec
$d = 6$	1792.0	échec	échec	échec
$d = 7$	512.0	échec	échec	échec
$d = 8$	64.0	passe	passe	passe

De même, pour la fonction `MixColumns`, après le calcul du nombre de monômes et leur répartition par degré, nous obtenons le résultat suivant (présenté sur les 7 derniers bits) :

Bit	Nbr. monômes	Répartition des degrés	Répartition attendue
120	110	$[0, 5, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
121	112	$[0, 5, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
122	114	$[0, 5, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
123	131	$[0, 7, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
124	136	$[0, 7, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
125	145	$[0, 5, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
126	133	$[0, 7, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
127	132	$[0, 5, 0, 0, 0, 0, 0, 0, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$

En appliquant ensuite le test statistique de Möbius nous obtenons le tableau suivant

Degré	T_i	$\alpha = 0,05$	$\alpha = 0,01$	$\alpha = 0,001$
$d = 0$	64.0	passe	passe	passe
$d = 1$	128.0	passe	passe	passe
$d = 2$	1792.0	échec	échec	échec
$d = 3$	3584.0	échec	échec	échec
$d = 4$	4480.0	échec	échec	échec
$d = 5$	3584.0	échec	échec	échec
$d = 6$	1792.0	échec	échec	échec
$d = 7$	512.0	échec	échec	échec
$d = 8$	64.0	passe	passe	passe

En conclusion, nous pouvons dire que la fonction `SubBytes` ne présente pas de biais statistique au regard du test statistique de Möbius. Par contre les fonctions

`ShiftRows` et `MixColumns` semblent en présenter. En réalité, cette différence de résultat s'explique par le fait que dans le tour de l'AES seule la fonction `SubBytes` fait de la permutation sur les bits et assure donc la fonction de diffusion dans l'algorithme. Les deux autres fonctions ne font que de la substitution de bits ce qui explique que le degré maximal des fonctions soit de 1. Pour les fonctions `ShiftRows` et `MixColumns`, il ne s'agit donc pas d'un biais statistique.

5.2.3.2 Tours de chiffrement et de déchiffrement

Nous allons appliquer maintenant la même démarche à un tour de chiffrement de l'AES. À l'instar du mini-AES, nous réduisons le tour de chiffrement à la composition des fonctions `SubBytes`, `ShiftRows` et `MixColumns`. La fonction résultante est donc $T(B) = MC \circ SR \circ NS(B)$.
La représentation de la distribution des degrés des monômes de chacune des 128 formes algébriques normales de la fonction $T(B)$ est détaillée dans l'annexe D page 175 et dans les figures 5.42 page 125, 5.43 page 125, 5.44 page 126 et 5.45 page 126. Comme pour la fonction `SubBytes`, pour chaque équation, nous calculons le nombre de monômes et leur répartition par degré. Nous obtenons le résultat suivant (présenté sur les 7 derniers bits) :

Bit	Nbr. monômes	Répartition des degrés	Répartition attendue
120	554	$[2, 18, 50, 140, 159, 120, 55, 10, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
121	564	$[5, 15, 44, 157, 154, 126, 53, 10, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
122	604	$[3, 19, 55, 154, 160, 149, 52, 12, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
123	885	$[0, 31, 93, 197, 240, 216, 87, 21, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
124	918	$[0, 28, 77, 215, 255, 213, 104, 26, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
125	701	$[2, 20, 64, 174, 174, 161, 83, 23, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
126	883	$[5, 28, 94, 202, 231, 201, 100, 22, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
127	616	$[3, 20, 68, 142, 165, 138, 67, 13, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$

Nous constatons que sur l'ensemble des monômes, le degré maximal est $d = 7$.
En calculant la fonction T décrite plus haut nous obtenons également une statistique qui suit une loi du χ^2 à 7 degrés de liberté dont les valeurs théoriques sont : $\chi^2 = 154,30$ pour $\alpha = 0,05$, $\chi^2 = 166,99$ pour $\alpha = 0,01$, $\chi^2 = 181,99$ pour $\alpha = 0,001$.
Nous pouvons maintenant établir le tableau suivant :

Degré	T_i	$\alpha = 0,05$	$\alpha = 0,01$	$\alpha = 0,001$
$d = 0$	1856.0	échec	échec	échec
$d = 1$	11740.0	échec	échec	échec
$d = 2$	29603.43	échec	échec	échec
$d = 3$	98959.43	échec	échec	échec
$d = 4$	96102.4	échec	échec	échec
$d = 5$	92569.14	échec	échec	échec
$d = 6$	37761.14	échec	échec	échec
$d = 7$	6700.0	échec	échec	échec
$d = 8$	64.0	passe	passe	passe

Comme nous l'avons vu précédemment, le tour de déchiffrement insère le `XOR` avec la clef de tour entre les fonctions `InvShiftRows` et `InvMixColumns`. La fonction de tour ne permet donc de combiner que les équations `InvSubBytes` et `InvShiftRows`. La fonction résultante est donc $T(B) = ISB \circ ISR(B)$.

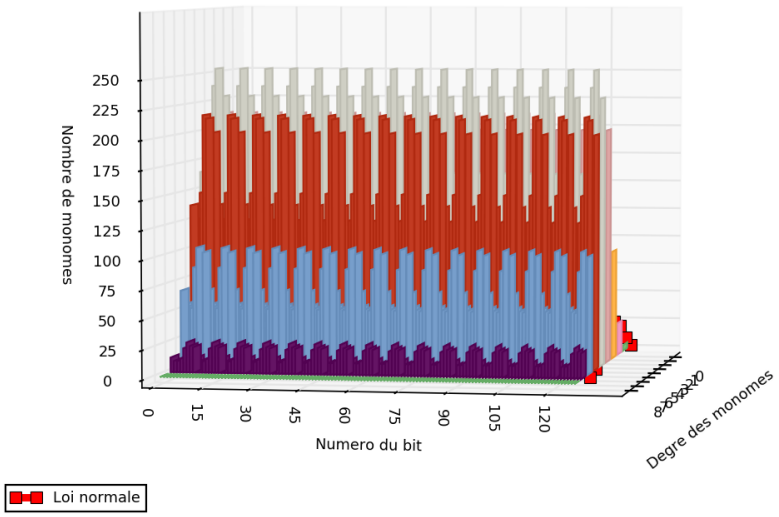


FIGURE 5.42 – Distribution des monômes de degré d pour un tour de chiffrement de l'AES (vue 1)

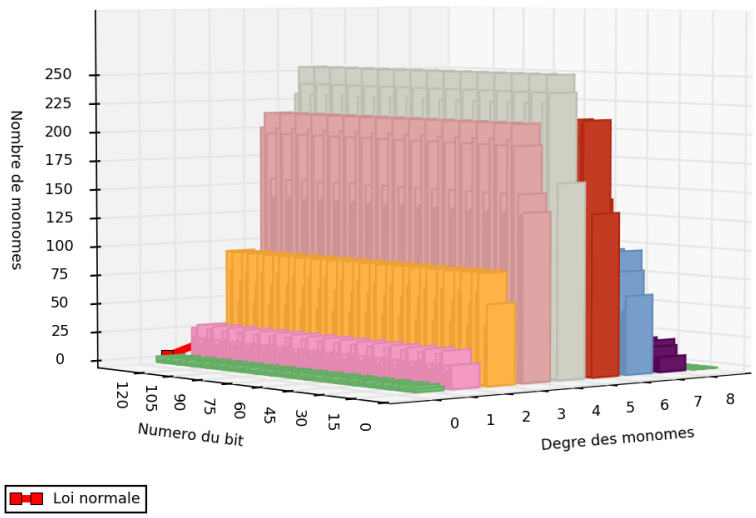


FIGURE 5.43 – Distribution des monômes de degré d pour un tour de chiffrement de l'AES (vue 2)

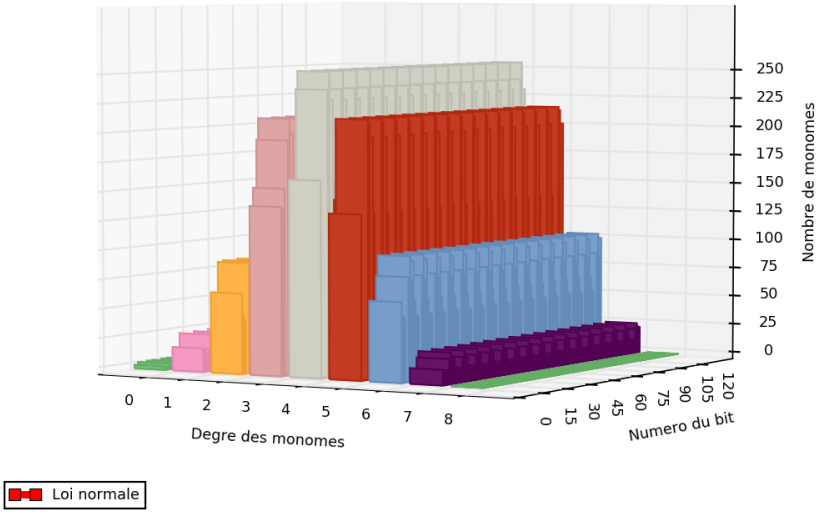


FIGURE 5.44 – Distribution des monômes de degré d pour un tour de chiffrement de l'AES (vue 3)

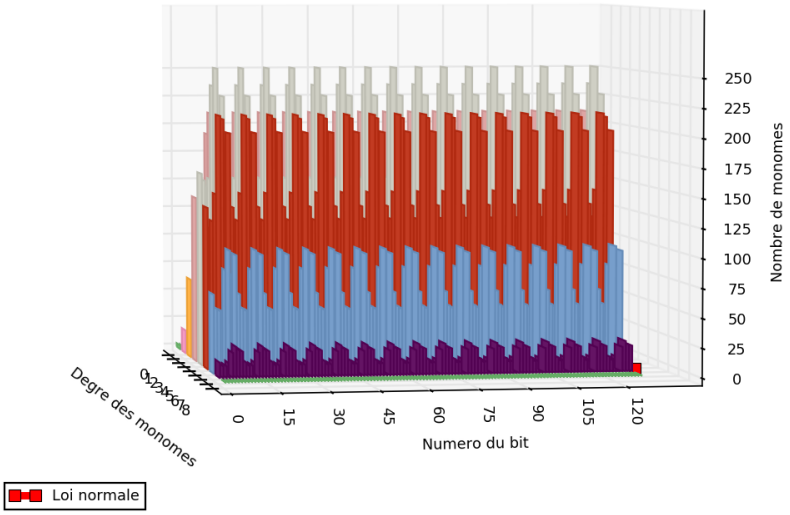


FIGURE 5.45 – Distribution des monômes de degré d pour un tour de chiffrement de l'AES (vue 4)

Appliquons maintenant le raisonnement précédent à cette fonction de tour de déchiffrement. Nous commençons par calculer le nombre de monômes et leur répartition par degré. Nous obtenons le résultat suivant (présenté sur les 7 derniers bits) :

Bit	Nbr. monômes	Répartition des degrés	Répartition attendue
120	110	$[0, 1, 14, 21, 32, 27, 14, 1, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
121	129	$[1, 4, 15, 30, 31, 30, 17, 1, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
122	121	$[0, 6, 14, 29, 27, 30, 11, 4, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
123	137	$[1, 2, 16, 29, 39, 30, 16, 4, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
124	117	$[0, 5, 8, 25, 35, 31, 9, 4, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
125	143	$[0, 3, 17, 33, 41, 31, 13, 5, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
126	132	$[1, 4, 8, 26, 41, 31, 16, 5, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
127	115	$[0, 2, 12, 25, 31, 29, 15, 1, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$

Nous constatons que sur l'ensemble des monômes, le degré maximal est $d = 7$. En appliquant ensuite le test statistique de Möbius nous obtenons le tableau suivant :

Degré	T_i	$\alpha = 0,05$	$\alpha = 0,01$	$\alpha = 0,001$
$d = 0$	64.0	passe	passe	passe
$d = 1$	92.0	passe	passe	passe
$d = 2$	102.86	passe	passe	passe
$d = 3$	58.29	passe	passe	passe
$d = 4$	88.23	passe	passe	passe
$d = 5$	23.43	passe	passe	passe
$d = 6$	60.57	passe	passe	passe
$d = 7$	116.0	passe	passe	passe
$d = 8$	64.0	passe	passe	passe

Nous constatons un comportement différent entre les tours de chiffrement et de déchiffrement. Cette différence est due, d'une part au fait que pour le déchiffrement, nous effectuons notre test sur des fonctions booléennes qui ne décrivent qu'une partie du tour et, d'autre part, au fait que le nombre de monômes des fonctions booléennes ainsi que leur degré maximal influent sur le résultat du test statistique de Möbius. À ce point de notre analyse, nous ne pouvons donc pas nous prononcer sur l'existence d'un éventuel biais statistique sur les fonctions de tour.

5.2.3.3 Fonction d'expansion de la clef

Appliquons maintenant notre démarche à la fonction d'expansion de la clef. Les fonctions booléennes que nous avons décrites correspondent à la construction d'une clef de tour. Ces fonctions sont identiques pour le processus de chiffrement et pour celui de déchiffrement.
La représentation de la distribution des degrés des monômes de chacune des 128 formes algébriques normales de la fonction d'expansion d'un tour de clef est détaillée dans les figures 5.46 page 128, 5.47 page 129, 5.48 page 129 et 5.49 page 130.
Pour chaque équation, le calcul du nombre de monômes et leur répartition par degré donne le résultat suivant (présenté sur les 7 derniers bits) :

Bit	Nbr. monômes	Répartition des degrés	Répartition attendue
120	114	$[0, 8, 10, 26, 33, 24, 11, 2, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
121	116	$[1, 7, 10, 31, 30, 24, 11, 2, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
122	118	$[1, 7, 7, 32, 32, 27, 10, 2, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
123	135	$[0, 9, 17, 29, 32, 34, 11, 3, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
124	140	$[0, 8, 11, 29, 39, 33, 16, 4, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
125	149	$[0, 8, 12, 38, 36, 33, 17, 5, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
126	137	$[1, 8, 14, 30, 33, 31, 16, 4, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$
127	136	$[1, 8, 16, 30, 33, 30, 15, 3, 0]$	$[0.5, 4, 14, 28, 35, 28, 14, 4, 0.5]$

Nous constatons que sur l'ensemble des monômes, le degré maximal est $d = 7$.

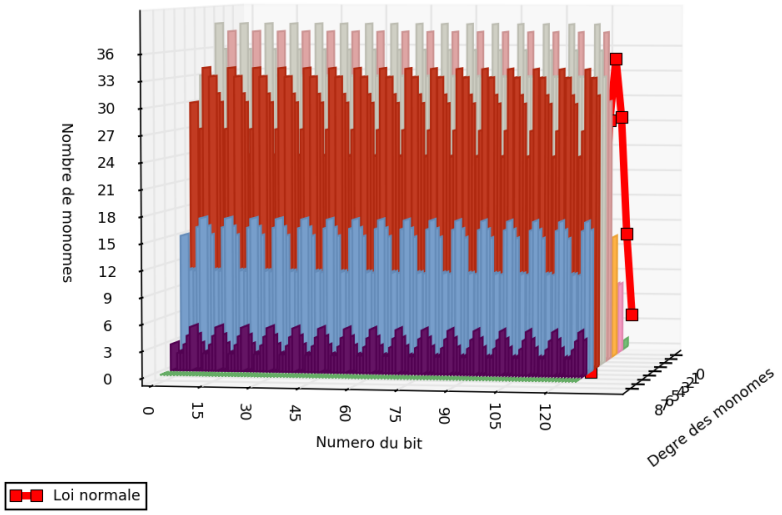


FIGURE 5.46 – Distribution des monômes de degré d pour la fonction d'expansion de la clef de l'AES (vue 1)

Enfin, en calculant la fonction T décrite plus haut nous obtenons également une statistique qui suit une loi du χ^2 à 7 degrés de liberté dont les valeurs théoriques sont : $\chi^2 = 154,30$ pour $\alpha = 0,05$, $\chi^2 = 166,99$ pour $\alpha = 0,01$, $\chi^2 = 181,99$ pour $\alpha = 0,001$.
Nous pouvons maintenant établir le tableau suivant :

Degré	T_i	$\alpha = 0,05$	$\alpha = 0,01$	$\alpha = 0,001$
$d = 0$	64.0	passe	passe	passe
$d = 1$	232.0	échec	échec	échec
$d = 2$	122.29	passe	passe	passe
$d = 3$	79.43	passe	passe	passe
$d = 4$	32.91	passe	passe	passe
$d = 5$	75.43	passe	passe	passe
$d = 6$	69.71	passe	passe	passe
$d = 7$	60.0	passe	passe	passe
$d = 8$	64.0	passe	passe	passe

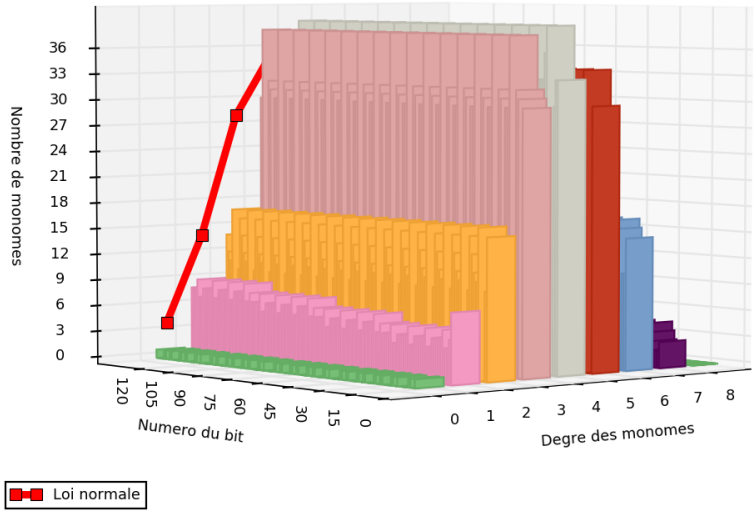


FIGURE 5.47 – Distribution des monômes de degré d pour la fonction d'expansion de la clef de l'AES (vue 2)

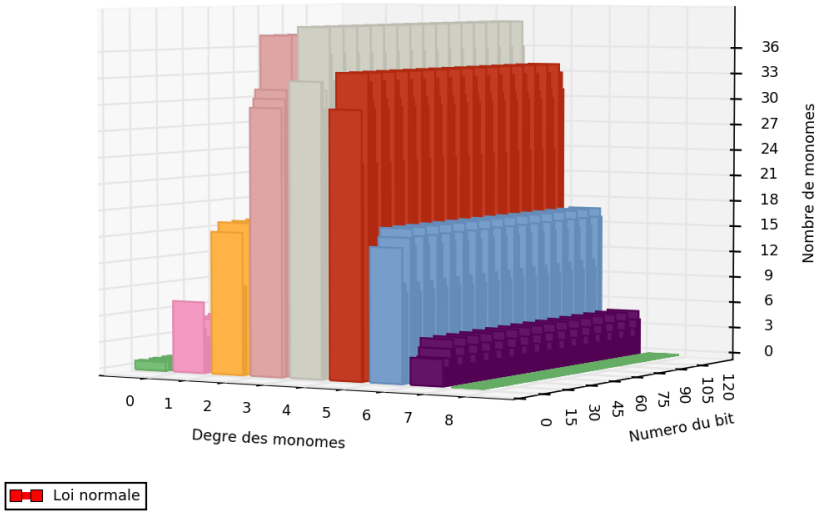


FIGURE 5.48 – Distribution des monômes de degré d pour la fonction d'expansion de la clef de l'AES (vue 3)

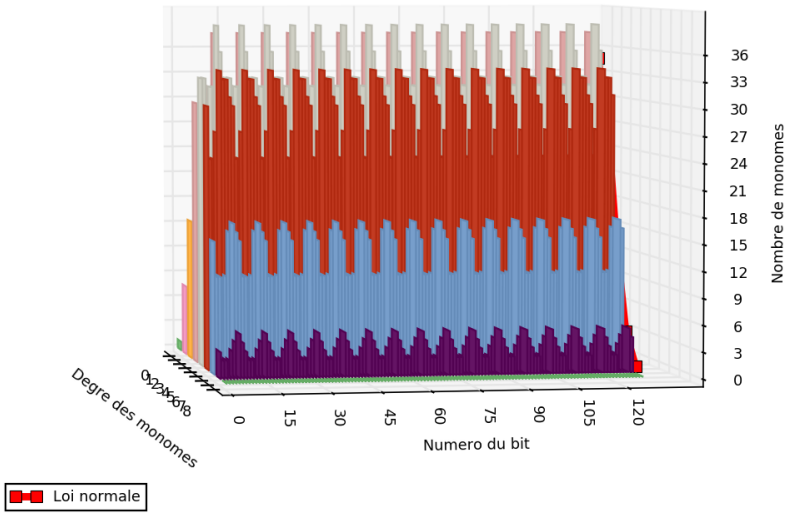


FIGURE 5.49 – Distribution des monômes de degré d pour la fonction d'expansion de la clef de l'AES (vue 4)

En conclusion, nous pouvons dire que la distribution des degrés des monômes de la fonction d'expansion de la clef de tour de l'AES ne présente pas de biais statistique sauf pour le cas ou $d = 1$. Dans ce dernier cas, il existe un biais statistique au regard du test statistique de Möbius. Il peut s'expliquer par le fait qu'à chaque étape de la fonction, l'algorithme utilise un bloc du tour précédent. En effet, la fonction d'expansion de la clef travaille sur un bloc de 4 octets en entrée et fournit un bloc de 4 octets en sortie, hors le bloc obtenu en sortie ne dépend pas uniquement du bloc d'entrée mais également du bloc du tour précédent avec lequel il est `XOR`é. Cette opération est la source du surnombre de monômes de degré 1.

5.3 Analyse structurelle des équations booléennes de l'AES

La combinatoire, est la branche des mathématiques qui étudie le dénombrement, la combinaison et la permutation d'ensembles d'éléments et les relations mathématiques qui caractérisent leurs propriétés.

5.3.1 Dénombrement des monômes

Au travers du test de Möbius nous avons mis en évidence un certain nombre de biais statistiques compensés par le jeu des tours. Attachons nous maintenant à compter et combiner les monômes des équations booléennes que nous avons

obtenues afin d'essayer de confirmer ou d'infirmer ces biais.
Pour cela, commençons par compter le nombre de monômes pour chacun des processus de chiffrement et de déchiffrement. Le graphe obtenu est présenté dans la figure 5.50 page 131. Ce graphe détaille le nombre de monômes de chacune des équations décrivant le processus de chiffrement et de déchiffrement de l'AES. Le nombre moyen de monômes est de 7881 pour le chiffrement et de 2711 pour le déchiffrement.

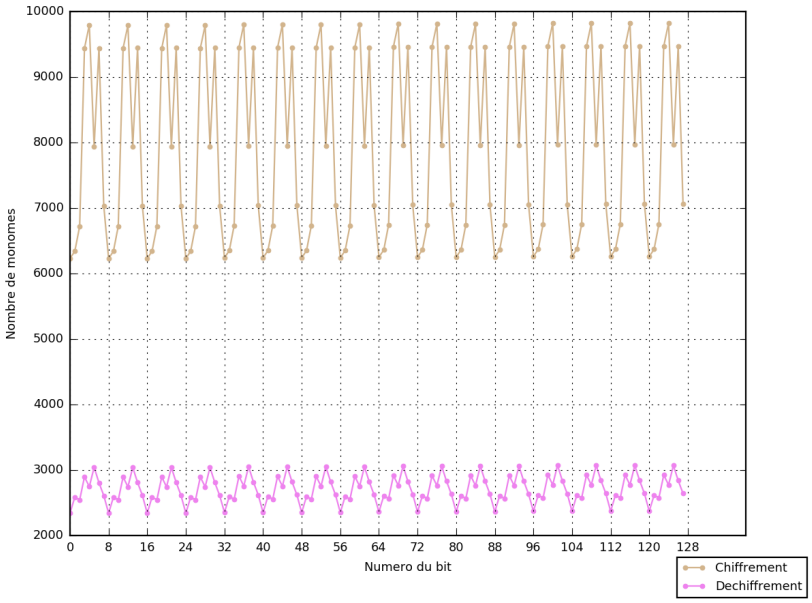
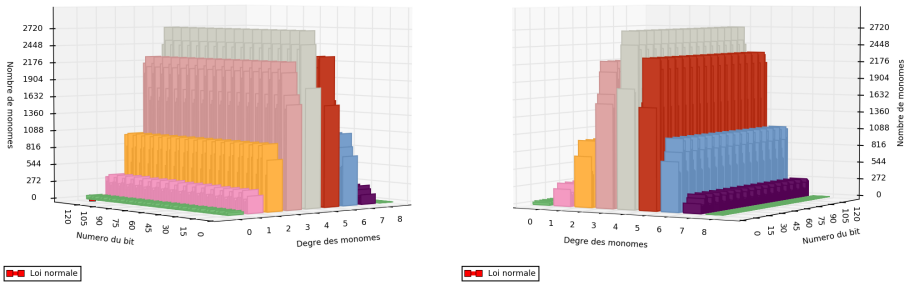


FIGURE 5.50 – Distribution des monômes

Ce graphe met en évidence trois éléments marquants.
Tout d'abord, nous constatons, sur les deux courbes, une évolution cyclique du nombre de monômes. Ce cycle a une période de 8, ce qui correspond au fait que l'élément de base de l'AES est l'octet. Ce premier point est donc normal.
Le deuxième constat, est que l'amplitude des courbes est plus importante pour le chiffrement que pour le déchiffrement. Ce point révèle un nombre de monômes réparti de façon plus resserrée pour le processus de déchiffrement que pour celui de chiffrement. Enfin, le troisième fait notable est la différence importante entre le nombre de monômes des processus de chiffrement et de déchiffrement.
Le processus de déchiffrement étant la fonction inverse de la fonction de chiffrement il vient naturellement à l'esprit que les fonctions décrivant ces deux processus sont similaires et que le nombre et la répartition des monômes de leurs fonctions respectives soient équivalentes. Nous constatons ici qu'il n'en est rien. Sachant que ce sont les équations du processus de déchiffrement qui sont les plus intéressantes pour la cryptanalyse d'un message chiffré cette différence n'est pas anodine.

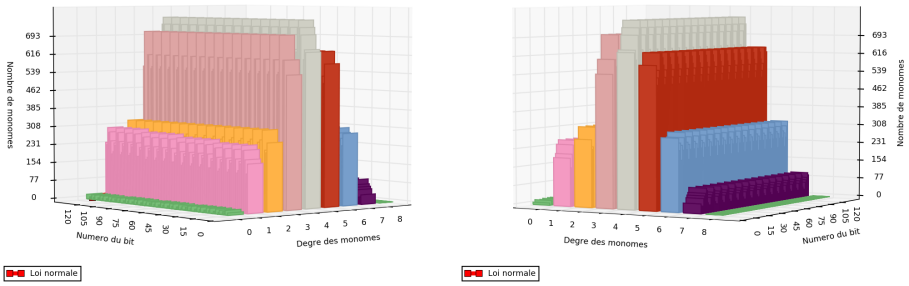
5.3.2 Dénombrement des degrés des monômes

Comme nous l'avons déjà abordé dans la section précédente, nous pouvons compter également le nombre de monômes par degré. Contrairement à l'approche précédente, nous prenons ici tous les monômes - sans distinction des variables qu'ils représentent - nous obtenons alors les graphes des figures 5.51 page 132 et 5.52 page 132.



(a) Vue à 230° (b) Vue à 300°

FIGURE 5.51 – Dénombrement et répartition des degrés des monômes - Processus de chiffrement



(a) Vue à 230° (b) Vue à 300°

FIGURE 5.52 – Dénombrement et répartition des degrés des monômes - Processus de déchiffrement

Nous constatons une différence entre les deux processus. En effet, outre la différence du nombre de monômes déjà évoquée, nous pouvons voir que le nombre de monômes de degré 1, 2, et 6 est quasiment identique. Il en va de même pour les monômes de degré 3 et 5.

5.3.3 Dénombrement des bits de bloc

Chacune des 256 équations booléennes - 128 pour le processus de chiffrement et 128 pour le processus de déchiffrement - que nous avons élaborée peut être

décrite par sa forme algébrique normale. Pour rappel, la forme générale d'une forme algébrique normale est :

$$f(x_0, x_1, \cdots, x_{n-1}) = \sum_{i=(i_0,\cdots,i_{n-1})\in\mathbb{F}_n^2} a_i x_0^{i_0} x_1^{i_1} \cdots x_{n-1}^{i_{n-1}} (mod 2)$$

En fonction de l'équation x_i correspond à un bit de texte clair ou à un bit de chiffré ou à un bit de clef.
Ainsi, à titre d'exemple, l'équation pour le premier bit de la première partie d'un tour de déchiffrement de l'AES - c'est à dire invSubBytes() and invShiftRows() - est la suivante :

$$\begin{aligned}
f(x_0, x_1, \cdots, x_{127}) \;=\; & x_6x_7 + x_5x_6 + x_4 + x_4x_7 + x_4x_5x_7 + x_4x_5x_6 + x_4x_5x_6x_7 + \\
& x_3x_7 + x_3x_6x_7 + x_3x_5 + x_3x_5x_6 + x_3x_5x_6x_7 + x_3x_4 + x_3x_4x_7 + x_3x_4x_6x_7 + x_3x_4x_5x_6 + \\
& x_3x_4x_5x_6x_7 + x_2x_6 + x_2x_5 + x_2x_5x_6 + x_2x_5x_6x_7 + x_2x_4x_6 + x_2x_4x_6x_7 + x_2x_4x_5x_7 + \\
& x_2x_3x_7 + x_2x_3x_6x_7 + x_2x_3x_5x_7 + x_2x_3x_5x_6x_7 + x_2x_3x_4x_6 + x_2x_3x_4x_5x_6 + x_1x_7 + \\
& x_1x_6 + x_1x_6x_7 + x_1x_5 + x_1x_4x_6x_7 + x_1x_4x_5x_7 + x_1x_3x_6 + x_1x_3x_6x_7 + x_1x_3x_5 + \\
& x_1x_3x_5x_6x_7 + x_1x_2 + x_1x_2x_7 + x_1x_2x_6x_7 + x_1x_2x_5x_6x_7 + x_1x_2x_4 + x_1x_2x_4x_7 + \\
& x_1x_2x_4x_6x_7 + x_1x_2x_4x_5x_7 + x_1x_2x_4x_5x_6 + x_1x_2x_4x_5x_6x_7 + x_1x_2x_3x_7 + x_1x_2x_3x_5x_7 + \\
& x_1x_2x_3x_5x_6 + x_1x_2x_3x_4 + x_1x_2x_3x_4x_6 + x_1x_2x_3x_4x_6x_7 + x_1x_2x_3x_4x_5 + x_1x_2x_3x_4x_5x_6 + \\
& x_0x_7 + x_0x_5x_7 + x_0x_5x_6x_7 + x_0x_4x_6x_7 + x_0x_4x_5x_6x_7 + x_0x_3 + x_0x_3x_6 + x_0x_3x_5 + \\
& x_0x_3x_5x_7 + x_0x_3x_5x_6x_7 + x_0x_3x_4x_6x_7 + x_0x_3x_4x_5 + x_0x_3x_4x_5x_7 + x_0x_2x_6 + x_0x_2x_5 + \\
& x_0x_2x_5x_6 + x_0x_2x_5x_6x_7 + x_0x_2x_4 + x_0x_2x_4x_7 + x_0x_2x_4x_6 + x_0x_2x_4x_5 + x_0x_2x_4x_5x_7 + \\
& x_0x_2x_3x_5x_6x_7 + x_0x_2x_3x_4 + x_0x_2x_3x_4x_6x_7 + x_0x_2x_3x_4x_5 + x_0x_2x_3x_4x_5x_6 + x_0x_1x_7 + \\
& x_0x_1x_5x_7 + x_0x_1x_5x_6x_7 + x_0x_1x_4x_7 + x_0x_1x_4x_6x_7 + x_0x_1x_4x_5x_6x_7 + x_0x_1x_3 + x_0x_1x_3x_6 + \\
& x_0x_1x_3x_6x_7 + x_0x_1x_3x_5x_6x_7 + x_0x_1x_3x_4x_5x_7 + x_0x_1x_2x_6x_7 + x_0x_1x_2x_5 + x_0x_1x_2x_5x_7 + \\
& x_0x_1x_2x_4 + x_0x_1x_2x_4x_6 + x_0x_1x_2x_4x_5 + x_0x_1x_2x_4x_5x_7 + x_0x_1x_2x_4x_5x_6 + x_0x_1x_2x_3 + \\
& x_0x_1x_2x_3x_7 + x_0x_1x_2x_3x_5x_7 + x_0x_1x_2x_3x_5x_6 + x_0x_1x_2x_3x_5x_6x_7 + x_0x_1x_2x_3x_4x_5
\end{aligned}$$

Dans cette équation chaque variable x_i correspond à un bit de bloc de clair.
Notre objectif ici est de compter le nombre de fois où chaque bit apparaît dans les équations. Normalement chaque bit devrait apparaître un nombre de fois équivalent. Dans le cas contraire, cela signifierait que certains bits ont moins de poids que d'autres, leur connaissance faciliterait grandement la cryptanalyse.
Afin d'avoir un point de comparaison, nous allons commencer par générer 16 équations booléennes aléatoires à 16 variables. En comptant le nombre de fois où chacune des variables apparaît dans les 16 équations nous obtenons le tableau de la figure 5.53 page 134. Nous y observons une répartition aléatoire des variables sans aucun schéma identifiable.
En appliquant le même mode de comptage aux 128 équations décrivant un tour de chiffrement - `SubBytes()`, `ShiftRows()` et `MixColumns()` -, de déchiffrement - `InvShiftRows()` et `InvSubBytes()` - et d'expansion de la clef de l'AES nous obtenons les graphes des figures 5.54 page 134, 5.55 page 135 et 5.56 page 135. Ces graphes diffèrent d'une utilisation aléatoire de chacune des variables. Dans l'algorithme de l'AES, certains bits de bloc ont plus de poids que d'autres. Ainsi, que ce soit pour le chiffrement que pour le déchiffrement, les bits $x_{7\ (\mathrm{mod}\ 8)}$ sont les plus utilisés et les bits $x_{2\ (\mathrm{mod}\ 8)}$ sont les moins utilisés. Les graphes des figures 5.57 page 136 et 5.58 page 136 présentent l'importance des bits par ordre de poids décroissant.
Le cas de la répartition de l'utilisation des bits dans la fonction d'expansion de la clef est particulièrement intéressant. En effet, nous constatons que seuls les 32 derniers bits sont utilisés. En fait, pour être précis, la répartition détaillée

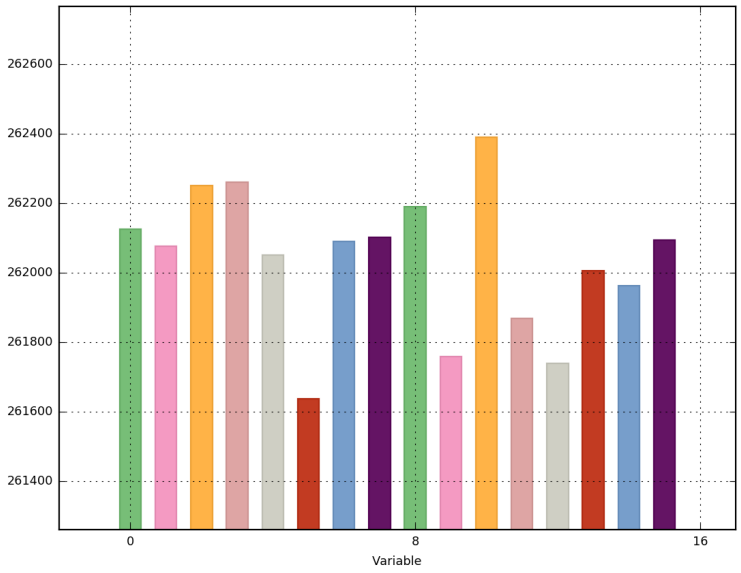


FIGURE 5.53 – Distribution des variables - Fonctions booléennes aléatoires

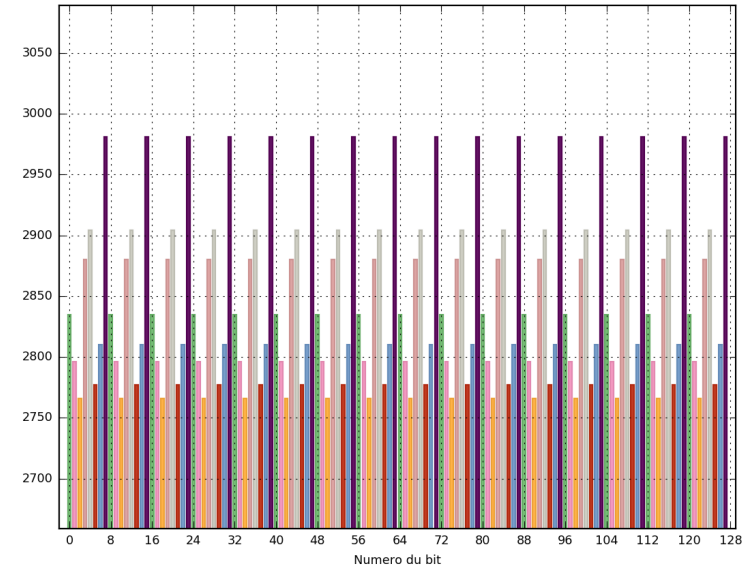


FIGURE 5.54 – Distribution des variables - Fonctions d'un tour de chiffrement

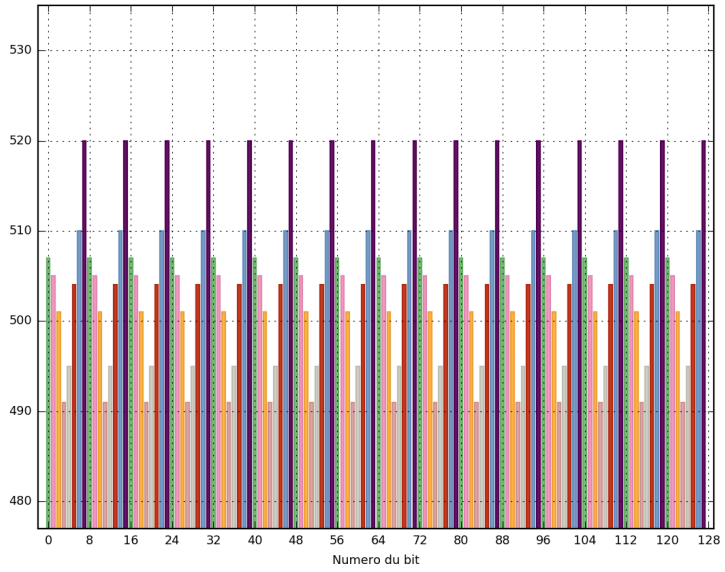


FIGURE 5.55 – Distribution des variables - Fonctions d'un tour de déchiffrement

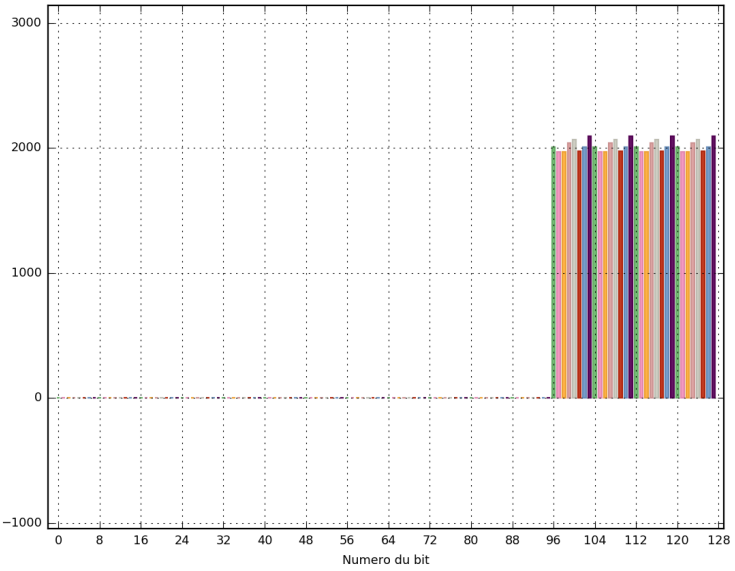


FIGURE 5.56 – Distribution des variables - Fonctions d'expansion de la clef

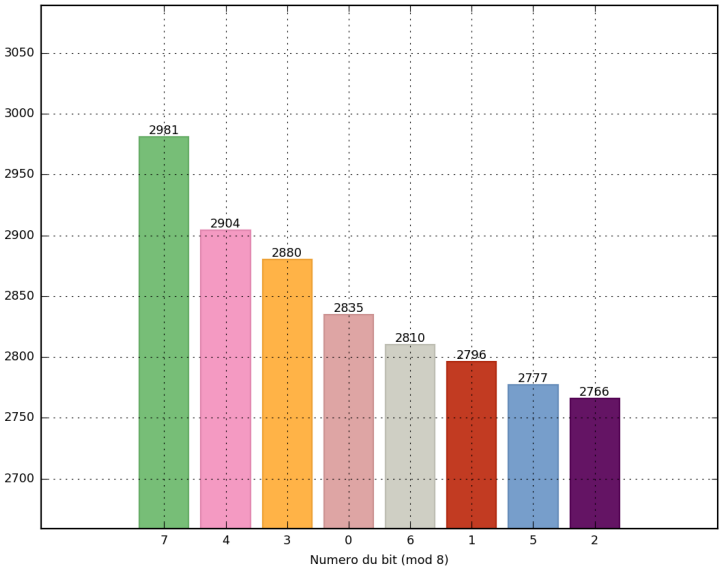


FIGURE 5.57 – Poids des bits - Fonctions d'un tour de chiffrement

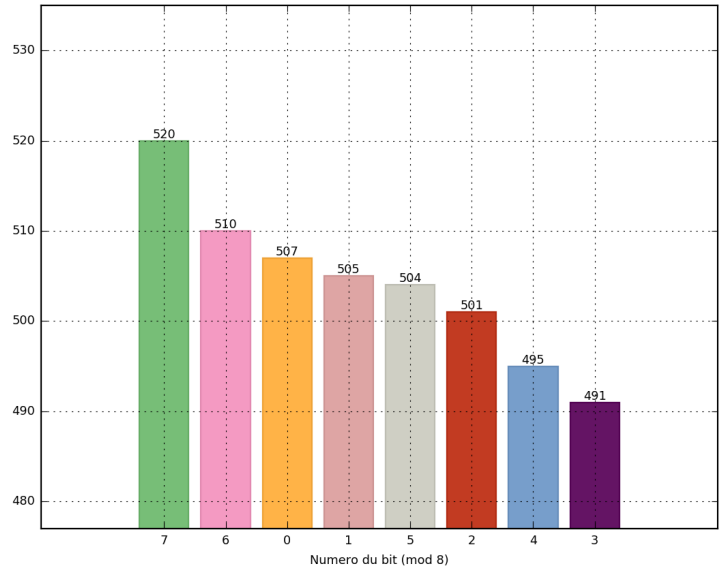


FIGURE 5.58 – Poids des bits - Fonctions d'un tour de déchiffrement

de l'utilisation des bits de clefs est présentée dans le tableau de la figure 5.59 page 137. Nous pouvons voir que, dans le processus d'expansion de la clef, les 32 premiers bits de clefs sont utilisés quatre fois, les 32 bits suivants trois fois, les 32 bits suivants seulement deux fois et que seuls les 32 derniers bits sont réellement utilisés. Cela signifie que sur 128 bits de clefs, les 32 derniers bits ont plus de poids. Cette vulnérabilité est compensée d'une part par l'intrication des bits de clef dans le processus d'expansion de la clef et par le jeu des tours.

	Numéro du bit							
Numéro de l'octet	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	4	4	4	4	4	4	4	4
2	4	4	4	4	4	4	4	4
3	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4
5	3	3	3	3	3	3	3	3
6	3	3	3	3	3	3	3	3
7	3	3	3	3	3	3	3	3
8	3	3	3	3	3	3	3	3
9	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2
11	2	2	2	2	2	2	2	2
12	2	2	2	2	2	2	2	2
13	2005	1969	1969	2041	2065	1973	2009	2093
14	2005	1969	1969	2041	2065	1973	2009	2093
15	2005	1969	1969	2041	2065	1973	2009	2093
16	2005	1969	1969	2041	2065	1973	2009	2093

FIGURE 5.59 – Utilisation des bits de la clef de tour

Afin de mieux visualiser l'impact de ce biais nous allons utiliser la même approche que celle présentée pour les PRBG. Pour cela nous utilisons l'AES pour chiffrer une séquence de blocs en fixant tous les bits à 0 avec une clef de chiffrement dont les 96 premiers bits sont fixés à 0 et dont les 32 derniers bits s'incrémentent. Nous obtenons ainsi une séquence de blocs que nous pouvons ensuite visualiser dans l'environnement 3D présenté plus haut. Le tableau 5.60 montre le résultat obtenu pour les premiers éléments de la séquence en utilisant un tour et le tableau 5.61 le résultat sur deux tours de l'AES.

Bloc de clef	Bloc de chiffré
00000000000000000000000000000000	01000000010000000100000001000000
00000001000000000000000000000000	010000011e1f213f0100000101000001
00000002000000000000000000000000	0100000215143c2a0100000201000002
00000003000000000000000000000000	0100000319182833010000301000003
00000004000000000000000000000000	010000049091a83d0100000401000004
00000005000000000000000000000000	0100000509081815010000501000005
00000006000000000000000000000000	010000060d0c141e0100000601000006
00000007000000000000000000000000	01000007a7a6f150010000701000007

FIGURE 5.60 – Blocs de chiffrés obtenus sur 1 tour de l'AES

Dans notre environement en trois dimensions, sur un échantillon comprenant

Bloc de clef	Bloc de chiffré
00000000000000000000000000000000	c6e4e48ba48787e8c6e4e48ba48787e8
00000001000000000000000000000000	27a63717a796f9c4d0f28fa62521d048
00000002000000000000000000000000	62d95538fab68aaf4062619e3810a448
00000003000000000000000000000000	977b2976f7372c3b664434d3eb662a8f
00000004000000000000000000000000	6aaad0540d0a4df582a0560794ee0870
00000005000000000000000000000000	3b4a67cd56f22a7ffcde62fa62c6d036
00000006000000000000000000000000	76c9db08e5208e16d7f561af313b0c69
00000007000000000000000000000000	a9ccea9c4201eedff6d40cecc07c517d

FIGURE 5.61 – Blocs de chiffrés obtenus sur 2 tours de l'AES

les 2^{20} premières séquences de clefs, nous obtenons la figure 5.62 sur un tour de l'AES et la figure 5.63 sur deux tours de l'AES. En comparant ces deux figures on constate que la différence de poids des bits de clefs dans l'algorithme d'expansion de la clef s'estompe dès le deuxième tour.

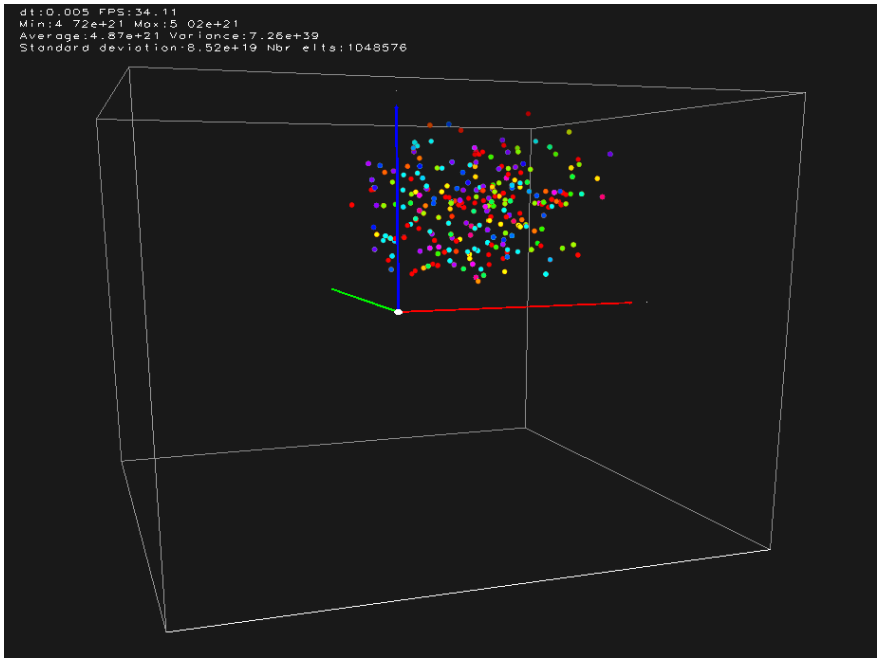


FIGURE 5.62 – Séquence obtenue sur 1 tour de l'AES

5.3.4 Dénombrement des bits de bloc deux par deux

Dans cette dernière étape de notre analyse combinatoire, nous allons dénombrer les paires de bits.
Toujours en partant de la forme algébrique normale des équations booléennes que nous avons générées, nous allons compter l'utilisation des bits deux à deux. Par exemple, si nous prenons le début de l'équation du premier bit d'un tour

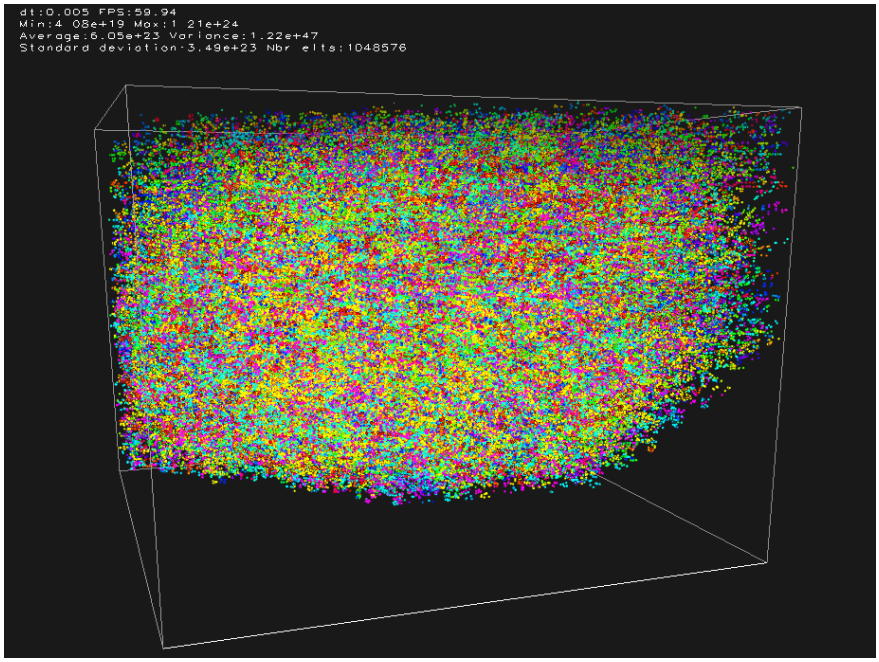


FIGURE 5.63 – Séquence obtenue sur 2 tours de l'AES

de déchiffrement :

$$f(b_0, b_1, \cdots, b_{127}) = b_6b_7 + b_5b_6 + b_4 + b_4b_7 + b_4b_5b_7 + b_4b_5b_6 + b_4b_5b_6b_7$$

nous avons les paires de bits suivantes qui apparaissent : $(6,7)$, $(5,6)$, $(4,7)$, $(4,5)$, $(5,7)$, $(4,5)$, $(5,6)$, $(4,5)$, $(5,6)$, $(6,7)$. Soit deux fois la paire $(6,7)$, trois fois la paire $(5,6)$, une fois la paire $(4,7)$, trois fois la paire $(4,5)$ et une fois la paire $(5,7)$.
Afin d'avoir un point de comparaison, nous générons 16 équations booléennes aléatoires à 16 variables et nous comptons la répartition des paires de variables. Nous obtenons le graphe de la figure 5.64 page 140. Sur ce graphe, chaque paire est représentée par une colonne placée au croisement des indices correspondants. La hauteur de la colonne est donnée par le nombre de fois où chaque paire apparaît dans les équations.
En appliquant le même mode de comptage aux 128 équations décrivant un tour de chiffrement, de déchiffrement et d'expansion de la clef de l'AES nous obtenons les graphes des figures 5.65 page 140, 5.66 page 141 et 5.67 page 141.
Ces trois graphes confirment les constats effectués précédemment. Pour les fonctions de tour ainsi que pour la fonction d'expansion de la clef, il n'y a pas de répartition aléatoire des paires de bits. Toutes les paires sont placées sur la médiane. Enfin, pour la fonction d'expansion de la clef, nous retrouvons l'utilisation des 32 derniers bits essentiellement.

—=oOo=—

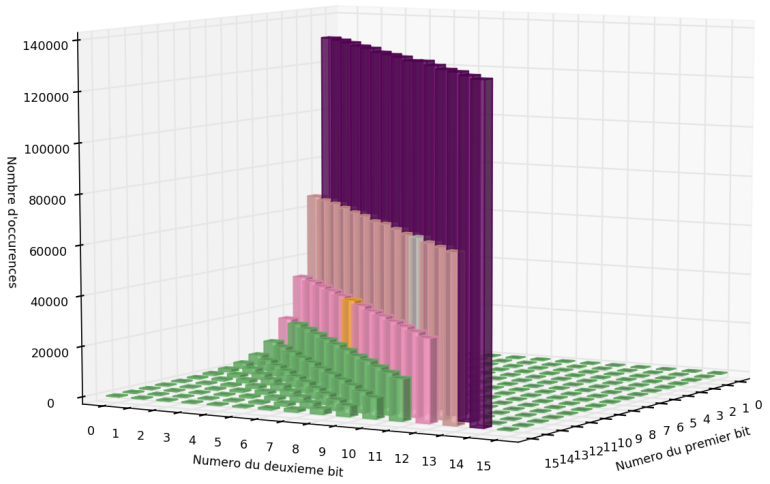


FIGURE 5.64 – Distribution des paires de variables - Fonctions booléennes aléatoires

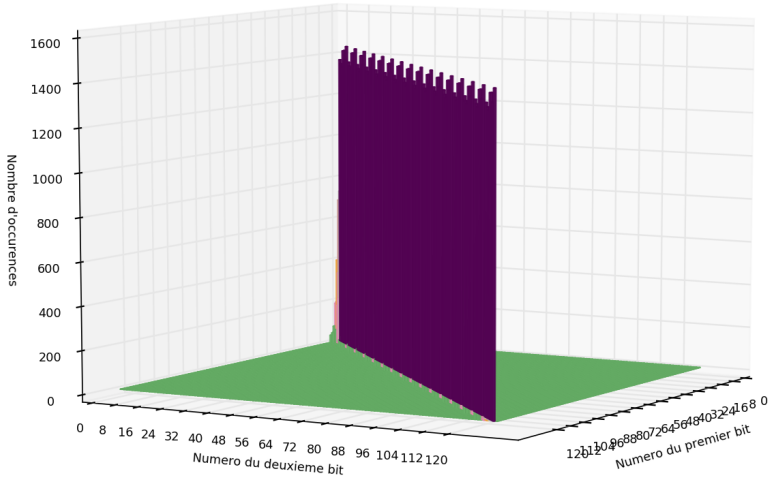


FIGURE 5.65 – Distribution des variables 2 à 2 - Fonctions d'un tour de chiffrement

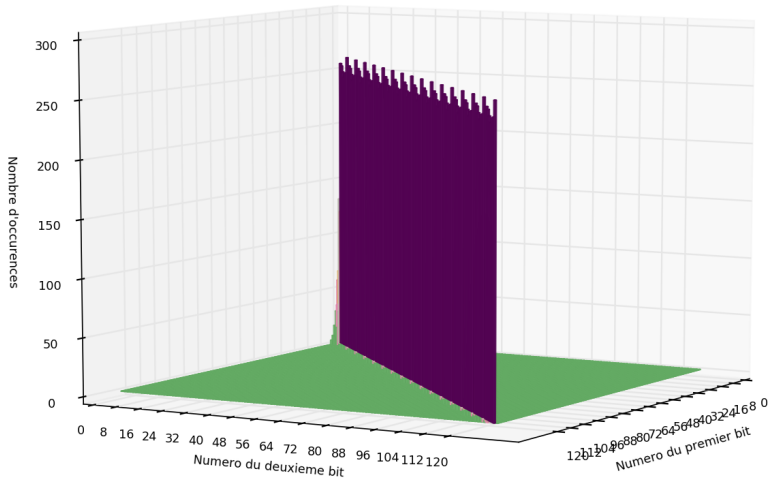


FIGURE 5.66 – Distribution des variables 2 à 2 - Fonctions d'un tour de déchiffrement

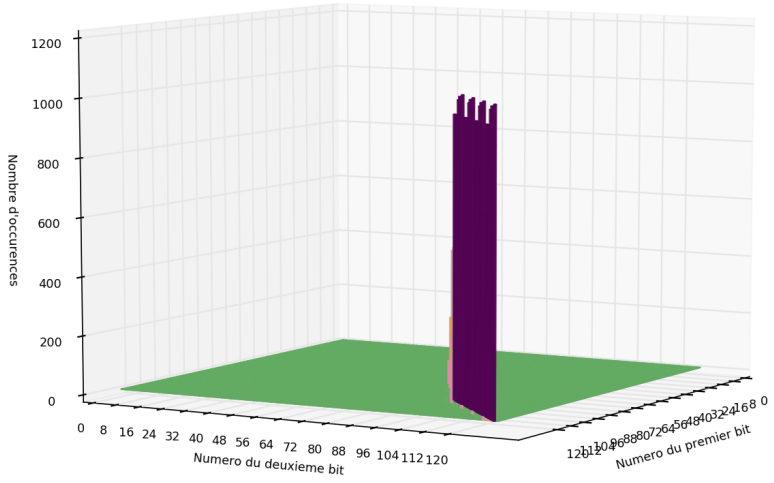


FIGURE 5.67 – Distribution des variables 2 à 2 - Fonctions d'expansion de la clef

Notre analyse des formes algébriques normales des fonctions booléennes décrivant le processus de chiffrement du mini-AES et de l'AES, à l'aide du test statistique de Möbius, nous a permis de trouver des irrégularités dans le mini-AES et surtout nous a permis de montrer formellement que la S-Box de l'AES ne présente pas de biais statistique. En revanche, toujours selon le test statistique de Möbius, nous avons montré que la fonction de tour de chiffrement de l'AES présente des biais statistiques alors que la fonction de tour de déchiffrement est conforme.
Enfin, notre analyse combinatoire montre une forte disparité entre les équations de tour du chiffrement et du déchiffrement. Surtout, cette analyse montre l'importance des 32 derniers bits de clefs par rapport au 96 premiers. Nous avons vu que, sur un petit échantillon, cette disparité ne semble pas avoir de conséquences. Cependant, une telle différence dans le poids des bits de clef n'est pas normale et ouvre une nouvelle perspective de recherche pour l'exploiter en termes de cryptanalyse.

Chapitre 6

Conclusion

Après une rapide introduction à la cryptographie et à ses principales primitives, nous avons détaillé le fonctionnement des algorithmes de chiffrement et de déchiffrement de l'Advanced Encryption Standard. Afin d'appréhender plus facilement son mode de fonctionnement interne, nous avons développé, en parallèle, une implémentation de l'AES en langage python.
Décrire l'AES comme un système d'équations quadratiques multivariées est une étape importante pour sa cryptanalyse. Cependant, nous avons montré que tant que des algorithmes de résolutions de grands systèmes d'équations plus efficients n'auront pas été trouvés, cette démarche est une impasse, dans le sens où elle n'est pas plus efficace que la cryptanalyse par force brute : *« Our conclusion is that if XSL works on BES, then it is worse than brute force »*[44].
Ensuite, après avoir présenté succinctement l'algèbre de Boole, les fonctions booléennes et deux de leurs représentations, nous avons élaboré un processus nous permettant de traduire complètement les algorithmes de chiffrement et de déchiffrement de l'AES en fonctions booléennes. Puis nous avons défini un mode de représentation de ces fonctions sous la forme de fichiers informatiques. Et enfin, nous avons mis au point un programme permettant d'implémenter ce processus et de contrôler que les résultats attendus sont conformes à ceux fournis dans le FIPS.
Finalement, notre analyse des formes algébriques normales des fonctions booléennes décrivant les processus de chiffrement et de déchiffrement de l'AES, à l'aide de l'analyse statistique de Möbius, nous a permis de montrer formellement que la S-Box de l'AES ne présente pas de biais statistique. En revanche, toujours selon le test statistique de Möbius, nous avons montré que la fonction de tour de chiffrement de l'AES présente des biais statistiques alors que la fonction de tour de déchiffrement est conforme. Enfin, notre analyse combinatoire a montré une forte disparité entre les équations de tour du chiffrement et du déchiffrement. Surtout, cette analyse a montré l'importance des 32 derniers bits de clefs par rapport au 96 premiers, même si, sur un petit échantillon, cette disparité ne semble pas avoir de conséquence.
Au final, nous pensons que l'*Advanced Encryption Standard* est un algorithme de chiffrement répondant, en partie, au Graal de tout algorithme cryptographique tel que nous l'avons explicité dans notre étude : *« obtenir, à chaque*

étape interne et à l'issue du processus de chiffrement, une séquence semblant la plus proche possible de l'aléa parfait. ». Il y répond en partie car, comme nous l'avons montré, les étapes internes du processus de chiffrement ne répondent pas à ce paradigme et surtout parce qu'il semble présenter une vulnérabilité importante dans sa fonction d'expansion de la clef.

À partir de ce travail, il reste de nombreuses pistes de recherche à parcourir.

Tout d'abord la recherche d'algorithmes efficaces pour la résolution de grands systèmes d'équations multivariées. Nous l'avons vu, les algorithmes existants ne sont pas plus efficaces que l'attaque par force brute. Pourtant, certains d'entre eux, comme le F5 [92], ont permis, avec succès de cryptanalyser le challenge de l'algorithme asymétrique *Hidden Field Equation* [93]. De plus, la résolution de tels systèmes dépasse le champ de la cryptographie [30] et ouvre des perspectives dans plusieurs autres domaines comme la théorie des codes, la programmation en nombres entiers, la théorie des systèmes,...

Une autre piste de recherche pourrait consister, à partir de ce travail, à utiliser les équations booléennes que nous avons générées pour tenter de recouvrer une clef de chiffrement en s'appuyant sur la différence de poids des différents bits.

Enfin, nous proposons une dernière piste de recherche à partir de ce travail : elle concerne la visualisation. En travaillant sur la représentation en trois dimensions des générateurs de PRBG, nous avons pris conscience de la potentialité de ce type d'animation. Notre cerveau appréhende plus facilement une image qu'une longue suite de chiffres. La généralisation et l'approfondissement de cette technique pourrait faciliter grandement l'identification d'un système d'exploitation, d'un algorithme cryptographique, d'un programme informatique, d'un virus,...

Au début de l'introduction de cette thèse, je faisais part de mon constat de la méconnaissance de mon entourage sur la question de la cryptographie. Ce décalage entre la connaissance que le grand public a de la cryptographie et l'état de l'art de la recherche universitaire et étatique dans ce domaine est préjudiciable.

En effet, les enjeux derrière la cryptanalyse de grands standards cryptographiques comme l'AES sont très importants. Aujourd'hui cet algorithme est universellement utilisé, il est directement intégré dans les microprocesseurs Intel et AMD et il est utilisé couramment dans des protocoles de communication comme TLS, SSH et IPsec. Tous les outils de chiffrement de surfaces et notamment TrueCrypt, LUKS, BitLocker et FileVault l'implémentent et l'utilisent par défaut. La découverte d'une faille sur cet algorithme serait alors une catastrophe : les transactions commerciales ou bancaires ne pourraient plus avoir lieu, les mécanismes de vote électronique seraient remis en cause, la protection des secrets industriels deviendrait inopérante, etc.

De manière plus générale, cet état de dépendance vis à vis de la cryptographie, pose la question de la confiance dans ces algorithmes, dans leur robustesse et surtout dans leur implémentation.

Concernant la robustesse, c'est un des enjeux de notre travail et, plus généralement, l'enjeu de l'une des deux branches de la cryptologie : la cryptanalyse. L'histoire de l'humanité est pleine de rebondissements liés au cassage de codes secrets par l'une des parties adverses [191]. À titre d'exemple, le 16 juin 2016, le sénateur américain Wyden interrogeant John Brennan, directeur de la CIA, à propos de l'impact sur le commerce américain de l'obligation d'utiliser des algorithmes cryptographiques backdoorés s'est vu répondre que les sociétés américaines n'avaient pas le choix : soit elles utilisaient des technologies de

chiffrement backdoorées soit elles étaient "malchanceuses" parce que la notion de solution de chiffrement non-américaine et fiable est théorique [1]. Tout l'enjeu d'une puissance économique, comme les États-Unis, est de faire en sorte que leurs algorithmes de chiffrement deviennent des normes internationales tout en y implantant des vulnérabilités qu'eux seuls sont capables de tourner à leur profit.

Le danger est aussi et notamment dans les implémentations des algorithmes cryptographiques. La faille CVE-2014-0160 dite "*SSL Heartbleed*" en ait un récent exemple. Dans ce cas, une erreur de conception logicielle permet de forcer le serveur SSL à renvoyer 64Ko de mémoire. Les informations collectées ainsi peuvent être notamment des clefs de chiffrement. Dans ce cas ce n'est pas l'implémentation de l'AES qui est en cause mais bien le logiciel qui l'utilise.

Un autre exemple est l'affaire de l'implémentation de SSL dans l'application WhatsApp. Cette application, rachetée en février 2014 par Facebook pour 19 milliards de dollars, permet d'échanger des SMS sans passer par un abonnement téléphonique. Pour chiffrer ces communications, l'application utilisait une ancienne version de SSL présentant une vulnérabilité permettant de forcer l'utilisation de clefs de tailles réduites.

De récents travaux de recherche ont montré que le diable est dans les détails. En effet, les systèmes d'exploitations modernes implémentent tous des primitives cryptographiques. Ces dernières utilisent les librairies de ces systèmes d'exploitation pour générer des clefs de chiffrement. Que se passe t-il si ces librairies ne sont pas fiables et fournissent des clefs statistiquement prédictibles ? Ou encore, que se passe t-il si un processus malveillant modifie, en mémoire, à la volée, l'implémentation de l'algorithme cryptographique du système [101] et en abaisse la sécurité ?

Enfin, l'utilisateur peut penser que ses communications sont correctement chiffrées mais dans les faits l'architecture technique déployée peut annuler ce mécanisme de sécurité. C'est un exemple de ce type qu'a révélé récemment Edward Snowden [105]. La communication entre un client Web et un service de la société Google est chiffrée à l'aide du protocole HTTPS. En fait, la communication chiffrée s'arrête à un proxy placé à l'entrée des réseaux internes de Google, les flux transitant au sein de ses réseaux, accessibles aux services de renseignements américains, circulent en clair.

La cryptographie est omniprésente dans nos vies. Cette omniprésence est telle que beaucoup d'actions que nous effectuons quotidiennement, comme téléphoner ou payer avec sa carte bancaire, ne serait pas possible sans elle. Comme toute technologie, son utilisation présente des risques et il est important que plus de personnes s'intéressent à cette branche passionnante et concrète des mathématiques.

1. http://www.theregister.co.uk/2016/06/17/non_us_encryption_is_theoretical_claims_cia/

Bibliographie

[1] Ibrahim Al-Kadi. Origins of cryptology : The arab contributions. *Cryptologia*, 16 :2 :97–126, 1992. http://dx.doi.org/10.1080/0161-119291866801.

[2] Martin Albrecht. *Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis*. PhD thesis, Royal Holloway, University of London, 2010. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.300.1046.

[3] Martin Albrecht and Carlos Cid. Algebraic techniques in differential cryptanalysis. *Cryptology ePrint Archive, Report 2008/177*, 2008. http://eprint.iacr.org/2008/177.pdf.

[4] Xavier Ameil, Jean-Pierre Vasseur, and Gilles Ruggiu. Histoire de la machine myosotis. *Colloques sur l'Histoire de l'Informatique, des Réseaux et des Télécommunications*, 7 :95–125, 2004. http://www.aconit.org/histoire/colloques/colloque_2004/ruggiu.pdf.

[5] Frederik Armknecht. On the existence of low-degree equations for algebraic attacks. *In ecrypt network of excellence - SASC Workshop Record*, 2004. http://eprint.iacr.org/2004/185.pdf.

[6] Frederik Armknecht and Stefan Lucks. Linearity of the AES key schedule. *Proceedings of the 4th International Conference, AES 2004*, pages 159–169, 2004.

[7] Emil Artin. *Galois Theory - Second Edition*. University of Notre Dame - London, 1964.

[8] Emil Artin. *Algebraic Numbers and Algebraic Functions*. Gordon and Breach, 1967.

[9] E.F. Assmus. On the Reed-Muller codes. *Discrete Math*, 106 :25–33, 1992.

[10] Atanas Atanasov. A short primer on cryptography. *Columbia Science Review*, 5 :10–13, 2008.

[11] Thomas Baignères, Pascal Junod, Yi Lu, Jean Monnerat, and Serge Vaudenay. *A Classical Introduction to Cryptography Applications for Communications Security - Exercise book*. Springer, 2006.

[12] Alan Baker. *A concise introduction to the theory of numbers*. Cambridge University Press, 1984.

[13] Shahram Bakhtiari. *Linear Cryptanalysis of DES Cipher*. PhD thesis, Wollongong University, 1994.

[14] Magali Bardet, Jean-charles Faugère, and Bruno Salvy. Complexity of gröbner basis computation for semi-regular overdetermined sequences over f2 with solutions in f2. *INRIA - Technical report 5049*, 2003. https://hal.inria.fr/inria-00071534/document/.

[15] Elan Barkan and Eli Biham. *The Book of Rijndaels*. Cryptology ePrint Archive, Report 2002/158, 2002. http://eprint.iacr.org/2002/158.

[16] Elan Barkan and Eli Biham. In how many ways can you write rijndael ? *Advances in Cryptology - ASIACRYPT 2002*, 2501 :160–175, 2002.

[17] Singh Bhupendra, Alexander Lexy, and Sanjay Burman. On algebraic relations of serpent s-boxes. *Cryptology ePrint Archive, Report 2009/038*, 2009. https://eprint.iacr.org/2009/038.pdf.

[18] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Concordia University*, 2005.

[19] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on AES variants with up to 10 rounds. *Cryptology ePrint Archive, Report 2009/374*, 2009. http://eprint.iacr.org/2009/374.

[20] Richard Blahut. *Algebraic Codes for Data Transmission*. Cambridge University Press, 2003.

[21] Richard Blahut. *Algebraic Codes on Lines, Planes and Curves*. Cambridge University Press, 2008.

[22] Lenore Blum, Manuel Blum, and Michael Shub. Comparison of two pseudo-random number generators. *Advances in Cryptology*, 82 :61–78, 1983.

[23] Lenore Blum, Manuel Blum, and Michael Shub. A simple unpredictable pseudo-random number generator. *Journal on Computing*, 15 :364–383, 1986.

[24] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. *Cryptology ePrint Archive, Report 2011/449*, 2011. http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf.

[25] Andrey Bogdanov and Andrey Pyshkin. Algebraic side-channel collision attacks on AES. *Cryptology ePrint Archive, Report 2007/477*, 2007. https://eprint.iacr.org/2007/477.pdf.

[26] George Boole. *The Mathematical Analysis of Logic*. Cambridge University Press, 1847. http://www.gutenberg.org/ebooks/36884.

[27] George Boole. *An Investigation of the Laws of Thought*. Courier Dover Publications, 1853. http://www.gutenberg.org/ebooks/15114.

[28] Wieband Bosma et al. The Magma Computational Algebra System (Version 2.22-6), 2016. http://magma.maths.usyd.edu.au/magma/.

[29] Randal Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35 :677–691, 1986. https://www.cs.cmu.edu/~bryant/pubdir/ieeetc86.pdf.

[30] Bruno Buchberger. Gröbner bases : A short introduction for systems theorists. *Computer Aided Systems Theory — EUROCAST 2001*, pages 1–19, 2001. http://people.reed.edu/~davidp/pcmi/buchberger.pdf.

[31] Bruno Buchberger and Franz Winkler. *Gröbner Bases and Applications*. London Mathematical Society Lecture Note Series, 1998.

[32] Stanislav Bulygin and Michael Brickenstein. Obtaining and solving systems of equations in key variables only for the small variants of AES. *Cryptology ePrint Archive, Report 2008/435*, 2008. http://eprint.iacr.org/2008/435.ps.

[33] Stanley Burris. *George Boole*. Edward N. Zalta, 2014. http://plato.stanford.edu/archives/sum2014/entries/boole/.

[34] Anne Canteaut. Cryptographic functions and design criteria for block ciphers. *INDOCRYPT 2001*, pages 1–16, 2001. https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/Canteaut01c.pdf.

[35] Anne Canteaut. Cryptanalyse des chiffrements à clef secrète par blocs. *MISC numéro 2*, 2002.

[36] Anne Canteaut. Open problems related to algebraic attacks on stream ciphers. *In WCC 2005*, pages 120–134, 2005.

[37] Claude Carlet. *Boolean Functions for Cryptography and Error Correcting Codes*. Cambridge University Press, 2010.

[38] Claude Carlet. *Vectorial Boolean Functions for Cryptography*. Cambridge University Press, 2010.

[39] Alexandre Casamayou et al. *Calcul mathématique avec SAGE*. CreateSpace Independent Publishing Platform, 2013. http://sagebook.gforge.inria.fr/.

[40] Pierre-Louis Cayrel. *Construction et optimisation de cryptosystèmes basés sur les codes correcteurs d'erreurs*. PhD thesis, Université de Limoges, 2008.

[41] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. *Advances in Cryptology - EUROCRYPT'94*, pages 356–365, 1995.

[42] Jiun-ming Chen, Nicolas Courtois, and Bo-yin Yang. On asymptotic security estimates in XL and gröbner bases-related algebraic cryptanalysis. *Information Security and Cryptology – ICISC 2004*, pages 401–413, 2004.

[43] Jiun-ming Chen and Bo-yin Yang. All in the XL family : Theory and practice. *Information Security and Cryptology – ICISC 2004*, pages 67–86, 2004.

[44] Lim Chu-Wee and Khoo Khoongming. An analysis of XSL applied to BES. *Lecture Notes in Computer Science*, 4593 :242–253, 2007. http://www1.spms.ntu.edu.sg/~kkhoongm/xsl_bes.pdf.

[45] Carlos Cid. Some algebraic aspects of the AES. *Proceedings of the 4th International Conference, AES 2004*, pages 58–66, 2004.

[46] Carlos Cid and Gaëtan Leurent. An analysis of the XSL algorithm. *Lecture Notes in Computer Science*, 3788 :333–335, 2007.

[47] Carlos Cid, Sean Murphy, and Matthew Robshaw. Computational and algebraic aspects of the advanced encryption standard. *Seventh International Workshop on Computer Algebra in Scientific Computing*, CASC' 2004 :93–103, 2004.

[48] Carlos Cid, Sean Murphy, and Matthew Robshaw. An algebraic framework for cipher embeddings. *10th IMA International Conference*, pages 278–289, 2005.

[49] Carlos Cid, Sean Murphy, and Matthew Robshaw. Small scale variants of the advanced encryption standard. *Fast Software Encryption*, 3557 :145–162, 2005.

[50] Carlos Cid, Sean Murphy, and Matthew Robshaw. *Algebraic Aspects of the advanced encryption standard*. Springer, 2006.

[51] Carlos Cid and Matthew Robshaw. Comments on the security of the AES and the XSL technique. *Electronic Letters*, 39 :26–38, 2003.

[52] CNSS. National policy on the use of the advanced encryption standard (AES) to protect national security systems and national security information, 2003. https://www.cnss.gov.

[53] CNSS. National information assurance policy on the use of public standards for the secure sharing of information among national security systems, 2012. https://www.cnss.gov.

[54] Gregory Conti and Sergey Bratus. Voyage of the reverser : A visual study of binary species. *Black Hat USA*, 2010.

[55] Gregory Conti, Erik Dean, Matthew Sinda, and Benjamin Sangster. Visual reverse engineering of binary and data files. *Workshop on Visualization for Computer Security (VizSEC)*, 2008.

[56] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 1965. https://doi.org/10.1090/S0025-5718-1965-0178586-1.

[57] Jim Cooper et al. Maplesoft, 2016. http://www.maplesoft.com.

[58] Nicolas Courtois. Algebraic attacks over $GF(2^k)$. *7th International Workshop on Theory and Practice in Public Key Cryptography*, pages 201–217, 2004.

[59] Nicolas Courtois. General principles of algebraic attacks and new design criteria for cipher components. *Proceedings of the 4th International Conference, AES 2004*, pages 67–83, 2004.

[60] Nicolas Courtois. How fast can be algebraic attacks on block ciphers ? *Cryptology ePrint Archive, Report 2006/168*, 2006. https://eprint.iacr.org/2006/168.pdf.

[61] Nicolas Courtois and Gregory Bard. Algebraic cryptanalysis of the DES. *Cryptology ePrint Archive, Report 2006/402*, 2006. http://eprint.iacr.org/2006/402.ps.

[62] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. *Advances in Cryptology — EUROCRYPT 2003*, pages 345–359, 2003.

[63] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. *Cryptology ePrint Archive, Report 2002/044*, 2002. https://eprint.iacr.org/2002/044.pdf.

[64] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. *Advances in Cryptology - ASIACRYPT 2002*, 2501 :267–287, 2002.

[65] David Cox, John Little, and Donald 0'Shea. *Ideals, Varieties, and Algorithms*. Undergraduate Texts in Mathematics. Springer, third edition edition, 2000.

[66] Yves Crama and Peter Hammer. *Boolean functions - Theory, Algorithms, and Applications*. Cambridge University Press, 2011.

[67] Dean Crnkovié and Vladimir Tonchev. *Information Security, Coding Theory and Related Combinatorics*. IOS Press, 2010.

[68] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher square. *Fast Software Encryption*, pages 149–165, 1997.

[69] Joan Daemen and Vincent Rijmen. AES proposal : Rijndael, 1999. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[70] Joan Daernen and Vincent Rijmen. *The design of Rijndael : AES - the Advanced Encryption Standard*. Springer, 2002.

[71] Christophe de Cannière and Alex Biryukov. Block ciphers and systems of quadratic equations. *Fast Software Encryption*, 2887 :274–289, 2003.

[72] Christophe de Cannière, Alex Biryukov, and Bart Preneel. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94 :346–356, 2006.

[73] Karl de Leeuw and Jan Bergstra. *The History of Information Security : A Comprehensive Handbook*. Elsevier Science, 2007.

[74] Frédéric Didier. *Codes de Reed-Muller et cryptanalyse du registre filtré*. PhD thesis, École Polytechnique, 2007.

[75] Claus Diem. The xl-algorithm and a conjecture from commutative algebra. *Proceedings of Asiacrypt 2004, LNCS, volume 3329*, pages 323–337, 2004.

[76] Whitfield Diffie and Martin Hellman. Cryptography and computer privacy. *Scientific American*, 228 :55–23, 1973.

[77] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22 :644–654, 1976.

[78] Hans Dobbertin, Lars Knudsen, and Matt Robshaw. The cryptanalysis of the AES - a brief survey. *Proceedings of the 4th International Conference, AES 2004*, pages 1–10, 2004.

[79] Christopher Domas. The future of re : Dynamic binary visualization. *Reverse engineering conference (REcon)*, 2013. https://sites.google.com/site/xxcantorxdustxx/visual-re.

[80] Michel Dubois. Description algébrique de l'advanced encryption standard. *MISC*, 55 :72–82, 2011.

[81] Michel Dubois. Github repositories, 2017. https://github.com/archoad.

[82] Michel Dubois and Éric Filiol. Proposal for a new equation system modelling of block ciphers. *Proceedings of the 2nd IMA Conference on Mathematics in Defence*, 2011. http://www.ima.org.uk/_db/_documents/Dubois.pdf.

[83] Michel Dubois and Éric Filiol. Proposal for a new equation system modelling of block ciphers and application to AES 128. *Proceedings of the 11th European Conference on Information Warfare and Security*, pages 303–312, 2012.

[84] Michel Dubois and Éric Filiol. Proposal for a new equation system modelling of block ciphers and application to AES 128 - long version. *Pioneer Journal of Algebra, Number Theory and its Applications*, 4 :11–40, 2012.

[85] Michel Dubois and Éric Filiol. 3d visualization applied to prbgs and cryptography. *Proceedings of the 11th International Conference on Cyber Warfare and Security*, pages 371–381, 2016.

[86] Michel Dubois and Éric Filiol. 3d visualization applied to prbgs and cryptography - long version. *Computer Science and Information Technology*, 4(5) :171–180, 2016.

[87] Michel Dubois and Éric Filiol. Hacking of the AES with boolean functions. *1st International Workshop on FORmal methods for Security Engineering - ForSE 2017*, pages 599–609, 2016.

[88] Michel Dubois and Éric Filiol. Hacking of the AES with boolean functions long version. *CoRR*, abs/1609.03734, 2016. https://arxiv.org/abs/1609.03734.

[89] Michel Dubois and Éric Filiol. Visualisation 3d appliquée aux prbg et à la cryptographie. *MISC*, 83 :74–82, 2016.

[90] John Eaton, David Bateman, Soren Hauberg, and Rik Wehbring. *GNU Octave version 4.2.0 manual : a high-level interactive language for numerical computations*. Free Software Foundation, 2016. https://www.gnu.org/software/octave/doc/v4.2.0/.

[91] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases (F4). *Journal of Pure and Applied Algebra*, pages 61–88, 1999. http://www-polsys.lip6.fr/~jcf/Papers/F99a.pdf.

[92] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (F5). *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, 2002. https://www.risc.jku.at/research/theorema/Groebner-Bases-Bibliography/gbbib_files/publication_502.pdf.

[93] Jean-charles Faugère and Antoine Joux. Algebraic cryptanalysis of hidden field equation (hfe) cryptosystems using gröbner bases. *In Advances in Cryptology — CRYPTO 2003*, pages 44–60, 2003.

[94] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. *Fast Software Encryption*, 1978 :213–230, 2000.

[95] Niels Ferguson, Richard Schroeppel, and Doug Whiting. A simple algebraic representation of rijndael. *Selected Areas in Cryptography*, 2259 :103–111, 2001.

[96] Éric Filiol. Designs, intersecting families, and weight of boolean functions. *Proceedings of the 7th IMA International Conference*, 1746 :70–80, 1999.

[97] Éric Filiol. A new statistical testing for symmetric ciphers and hash functions. *Proceedings of Information and Communications Security 2002*, 2513 :342–353, 2002.

[98] Éric Filiol. Plaintext-dependant repetition codes cryptanalysis of block ciphers. *In Workshop on Combinatorics, Codes and Cryptology*, 2003.

[99] Éric Filiol. *Reconstruction Techniques in Cryptology and in Coding Theory*. PhD thesis, École Polytechnique, 2005.

[100] Éric Filiol. *Modèles booléens en virologie et en cryptologie*. PhD thesis, Université de Rennes, 2007.

[101] Éric Filiol. Dynamic cryptographic trapdoors. *CanSecWest*, 2011. https://cansecwest.com/csw11/filiol_csw2011.pdf.

[102] Éric Filiol and Caroline Fontaine. Highly nonlinear balanced boolean functions with a good correlation-immunity. *Advances in Cryptology — EUROCRYPT'98*, 1403 :475–488, 1998.

[103] Caroline Fontaine. *Contribution à la recherche de fonctions booléennes hautement non linéaires, et au marquage d'images en vue de la protection des droits d'auteur*. PhD thesis, Université Paris VI, 1998.

[104] Joanne Fuller and William Millan. On linear redundancy in the AES s-box. *Fast Software Encryption*, 2887 :74–86, 2003.

[105] Barton Gellman and Ashkan Soltani. Nsa infiltrates links to yahoo, google data centers worldwide, snowden documents say. *The Washington Post*, 2013. https://ashkansoltani.org/work/washpost/#snowden.

[106] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography, 2008. http://cseweb.ucsd.edu/~mihir/papers/gb.pdf.

[107] Xavier Gourdon. *Algèbre*. Éditions ellipses, 1994.

[108] Secrétaire général de la défense et de la sécurité nationale. Instruction générale interministérielle 1300 sur la protection du secret de la défense nationale, 2011. https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000024892134.

[109] Mads Haahr. Random.org : True random number service, 2016. http://www.random.org.

[110] Chris Hamilton. Compact hilbert indices. *Technical report CS-2006-07*, 2006. https://www.cs.dal.ca/sites/default/files/technical_reports/CS-2006-07.pdf.

[111] Richard Wesley Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29 :147–160, 1950.

[112] Peter Higgins. *Number story : from counting to cryptography*. Copernicus book - Springler, 2008.

[113] David Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, 38 :459–460, 1891. http://dx.doi.org/10.1007/BF01199431.

[114] Truevision Inc. Truevision tga file format specificartion. *www.truevision.com*, 1991. http://www.dca.fee.unicamp.br/~martino/disciplinas/ea978/tgaffs.pdf.

[115] Thomas Jakobsen and Lars Knudsen. Attacks on block ciphers of low algebraic degree. *Journal of Cryptology*, 14 :197–210, 2001.

[116] Pascal Junod. Cryptographic secure pseudo-random bits generation : The blum-blum-shub generator. *Support de cours*, 1999. http://cs.miami.edu/~burt/learning/Csc609.062/docs/bbs.pdf.

[117] David Kahn. *The Codebreakers*. Scribner, 1996.

[118] Michael Kalkbrener. Solving systems of algebraic equations by using gröbner bases. *Proceedings of European Conference on Computer Algebra*, pages 282–292, 1987.

[119] Matthew Kennel, Reggie Brown, and Henry Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, 45 :3403–3411, 1992. http://www.csee.wvu.edu/~xinl/library/papers/physics/embedding_dimension.pdf.

[120] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, 9 :5–38, 1883.

[121] Elizabeth Kleiman. *The XL and XSL attacks on Baby Rijndael*. PhD thesis, Iowa State University, 2005.

[122] Lars Knudsen. Contemporary block ciphers. *Lectures on Data Security*, 1561 :105–126, 1999.

[123] Lars Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Springer, 2011.

[124] Donald Ervin Knuth. *The Art of Computer Programming*, volume Volume 2 - Seminumerical Algorithms. Addison Wesley, 1981.

[125] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994.

[126] Susan Landau. Communications security for the twenty-first century : The advanced encryption standard. *Notices of the American Mathematical Society*, 47 :450–459, 2000.

[127] Susan Landau. Standing the test of time : The data encryption standard. *Notices of the American Mathematical Society*, 47 :341–349, 2000.

[128] Susan Landau. Polynomials in the nation's service : Using algebra to design the advanced encryption standard. *American Mathematical Monthly*, 111 :89–117, 2004.

[129] Serge Lang. *Algebra*. Springer, 2002.

[130] Tri Van Le. Novel cyclic and algebraic properties of AES. *Cryptology ePrint Archive, Report 2003/108*, 2003. https://eprint.iacr.org/2003/108.pdf.

[131] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986.

[132] Rudolf Lidl and Harald Niederreiter. *Finite Fields - Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1997.

[133] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 2000.

[134] Jack Little, Cleve Moler, and Steve Bangert. Matlab, 2016. https://www.mathworks.com.

[135] Frank Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46 :68–78, 1951.

[136] James Massey. Some applications of coding theory in cryptography. *Codes and Ciphers : Cryptography and Coding IV*, pages 33–47, 1995.

[137] James Massey. Applied digital information theory, 2001. http://www.isiweb.ee.ethz.ch/archive/massey_scr/.

[138] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. *Advances in Cryptology - EUROCRYPT 1993*, 765 :386–397, 1993.

[139] Ueli Maurer. A universal statistical test for random bit generators. *Journal of cryptology*, 5 :89–105, 1992.

[140] Robert May. Simple mathematical models with very complicated dynamics. *Nature*, 261 :459–467, 1976. http://abel.harvard.edu/archive/118r_spring_05/docs/may.pdf.

[141] Paul McCarty. *Introduction to Arithmetical Functions*. Springer, 1986.

[142] Alfred Menezes, Paul Oorschot, and Scott Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.

[143] James Milne. Algebraic number theory, 2013. www.jmilne.org/math/.

[144] James Milne. Class field theory, 2013. www.jmilne.org/math/.

[145] James Milne. Fields and galois theory, 2013. www.jmilne.org/math/.

[146] Jean Monnerat and Serge Vaudenay. On some weak extensions of AES and BES. *6th International Conference - Information and Communications Security*, pages 414–426, 2004.

[147] Sean Murphy and Matthew Robshaw. Essential algebraic structure within the AES. *Advances in Cryptology - CRYPTO 2002*, 2442 :1–16, 2002.

[148] Mohammad Musa, Edward Schaefer, and Stephen Wedig. A simplified AES algorithm and its linear and differential cryptanalyses. *Cryptologia*, 27 :148–177, 2003.

[149] Kara Nance, Sebastien Tricaud, and Philippe Saadé. Visualizing network activity using parallel coordinates. *44th Hawaii International Conference on System Sciences*, 2011.

[150] Jones Neil. *Computability and Complexity*. The MIT Press, 1997.

[151] Harris Nover. Algebraic cryptanalysis of AES an overview, 2004. http://www.math.wisc.edu/~boston/nover.pdf.

[152] Ryan O'Donnel. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[153] National Institute of Standards and Technology. Security requirements for cryptographic modules 140-1, 1994. http://csrc.nist.gov/publications/fips/fips140-1/fips1401.pdf.

[154] National Institute of Standards and Technology. Data encryption standard, 1999. http://csrc.nist.gov/publications/fips/archive/fips46-3/fips46-3.pdf.

[155] National Institute of Standards and Technology. Advanced encryption standard, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[156] National Institute of Standards and Technology. Security requirements for cryptographic modules 140-2, 2001. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[157] National Institute of Standards and Technology. Recommendation for the triple data encryption algorithm (tdea) block cipher, 2012. http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf.

[158] Daniel Olejar and Martin Stanek. On cryptographic properties of random boolean functions. *Electronic Journal of Universal Computer Science*, 4 :705–717, 1998.

[159] Norman Packard, James Crutchfield, Doyne Farmer, and Robert Shaw. Geometry from a time series. *Physical Review Letters*, 45 :712–716, 1980. http://tuvalu.santafe.edu/~jdf/papers/geometrytimeseries.pdf.

[160] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36 :157–160, 1890. http://dx.doi.org/10.1007/BF01199438.

[161] Daniel Perrin. *Cours d'algèbre*. Éditions ellipses, 1996.

[162] Raphael Chung-Wei Phan. Mini advanced encryption standard (mini-AES) : A testbed for cryptanalysis students. *Cryptologia*, 26 :283–306, 2002.

[163] Vladimir Platonov and Andrei Rapinchuk. *Algebraic Groups and Number Theory*. Academic Press, INC., 1991.

[164] Bart Preneel, René Govaerts, and Joos Vandewalle. Cryptographic properties of quadratic boolean functions. *1st International Conference on Finite Fields and Applications*, 1991.

[165] Bart Preneel, Werner van Leekwijck, Luc van Linden, René Govaerts, and Joos Vandewalle. Propagation characteristics of boolean functions. *Advances in Cryptology - EUROCRYPT 1990*, 473 :161–173, 1990.

[166] Sagemath project team. Sage's reference manual, 2013. http://www.sagemath.org/doc/reference/.

[167] Irving Reed. A class of multiple-error-correcting codes and the decoding scheme. *Information Theory, IRE Professional Group on*, 4 :31–49, 1954.

[168] Mathieu Renauld and Francois-Xavier Standaert. Algebraic side-channel attacks. *Cryptology ePrint Archive, Report 2009/279*, 2009. https://eprint.iacr.org/2009/279.pdf.

[169] Wolfram Research. Mathematica, 2016. https://www.wolfram.com/mathematica/.

[170] Vincent Rijmen. *Cryptanalysis and Design of Iterated Block Ciphers*. PhD thesis, Katholieke Universiteit Leuven, 1997.

[171] Vincent Rijmen, Joan Daemen, Bart Preneel, and Antoon Bosselaers. The cipher shark, 1996. https://www.cosic.esat.kuleuven.be/publications/article-55.pdf.

[172] Anna Rimoldi, Massimiliano Sala, and Enrico Bertolazzi. Do AES encryptions act randomly ? *CoRR*, abs/1011.2644, 2010. http://arxiv.org/pdf/1011.2644v1.pdf.

[173] Anna Rimoldi, Massimiliano Sala, and Ilia Toli. A possible intrinsic weakness of AES and other cryptosystems. *CoRR*, abs/1006.5894, 2010. http://arxiv.org/abs/1006.5894.

[174] Kenneth Rosen. *Discrete Mathematics and its Applications - Seventh Edition*. Mc Graw Hill, 2012.

[175] Andrew Rukhin et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *National Institute of Standards and Technology*, 2010. http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf.

[176] Bassem Sakkour. *Étude et amélioration du décodage des codes de Reed-Muller d'ordre deux*. PhD thesis, École Polytechnique, 2007.

[177] Gilbert Saporta. *Probabilités et statistiques*. Éditions Technip, 2006.

[178] Palash Sarkar and Subhamoy Maitra. Construction of nonlinear boolean functions with important cryptographic properties. *In Advances in Cryptology - EUROCRYPT 2000*, 1807 :485–506, 2000. http://www.iacr.org/archive/eurocrypt2000/1807/18070491-new.pdf.

[179] Palash Sarkar and Subhamoy Maitra. New directions in design of resilient boolean functions. *IACR Eprint archive*, 2000. http://eprint.iacr.org/2000/009.

[180] Nicolas Sarkozy, Dominique Perben, Renaud Donnedieu de Vabres, Brigitte Girardin, and Patrick Devedjian. Loi 2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique. *Journal officiel de la République française*, 2004. http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000801164.

[181] Bruce Schneier. AES news, 2002. https://www.schneier.com/crypto-gram-0209.html.

[182] Bruce Schneier. *Cryptographie appliquée : algorithmes, protocoles et codes source en C*. Éditions Vuibert, 2002.

[183] Bruce Schneier and Niels Ferguson. *Practical cryptography*. Wiley, 2003.

[184] Ernst Schröder, Jakob Lüroth, and Karl Eugen Müller. *Vorlesungen über die Algebra der Logik*, volume 3. B. G. Teubner, 1895.

[185] Jennifer Seberry and Beomsik Song. Further observations on the structure of the AES algorithm. *Fast Software Encryption*, 2887 :223–233, 2003.

[186] Adi Shamir, Jacques Patarin, Nicolas Courtois, and Alexander Klimov. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. *Advances in Cryptology — EUROCRYPT 2000*, 1807 :392–407, 2000.

[187] Claude Elwood Shannon. *A Symbolic Analysis of Relay and Switching Circuits*. PhD thesis, Massachusets Institute of Technologies - Cambridge, 1938.

[188] Claude Elwood Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 27, 1948.

[189] Claude Elwood Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28, 1949.

[190] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 28, 1949.

[191] Simon Singh. *Histoire des codes secrets*. Éditions Jean-Claude Lattès, 1999.

[192] W.A. Stein et al. Sage Mathematics Software (Version 7.1), 2016. http://www.sagemath.org.

[193] Jacques Stern. *La science du secret*. Éditions Odile Jacob, 2006.

[194] The OpenSSL Project. Openssl library. https://www.openssl.org/, 2017.

[195] Henk Tilborg. *Encyclopedia of Cryptography and Security*. Springer, 2011.

[196] Ilia Toli and Alberto Zanoni. An algebraic interpretation of AES-128. *4th International Conference, AES 2004*, pages 84–97, 2004.

[197] Sebastien Tricaud and Philippe Saadé. Applied parallel coordinates for logs and network traffic attack analysis. *Journal in Computer Virology*, 6 :1–29, 2010.

[198] Tri Van Lee, Rüdiger Sparr, Ralph Wernsdorf, and Yvo Desmedt. Complementation-like and cyclic properties of AES round functions. *Proceedings of the 4th International Conference, AES 2004*, pages 128–141, 2004.

[199] Henk van Tilborg. *Fundamentals of Cryptology*. Kluwer Academic Publishers, 2000.

[200] Henk van Tilborg. Coding theory : a first course, 2013. http://hyperelliptic.org/tanja/teaching/CCI11/CODING.pdf.

[201] Serge Vaudenay. *A Classical Introduction to Cryptography Applications for Communications Security*. Springer, 2006.

[202] Vesselin Velichkov, Vincent Rijmen, and Bart Preneel. Algebraic cryptanalysis of a small-scale version of stream cipher lex. *IET Information Security*, 4 :49–61, 2010.

[203] Vanessa Vitse. *Attaques algébriques du problème du logarithme discret sur courbes elliptiques*. PhD thesis, Université Versailles Saint-Quentin en Yvelines, 2011.

[204] John von Neumann. Various techniques used in connection with random digits. *Monte Carlo Method*, 12 :36–38, 1951.

[205] David Wagner. The boomerang attack. *Fast Software Encryption*, 1636 :156–170, 1999.

[206] F Webster and Stafford Tavares. On the design of s-boxes. *Advances in Cryptology - CRYPTO 85 Proceedings*, 218 :523–534, 1986.

[207] Ralp-Philipp Weinmann. *Evaluating Algebraic Attacks on the AES*. PhD thesis, Technische Universität Darmstadt, 2003.

[208] Eric Weisstein. Closed-form solution, 2016. http://mathworld.wolfram.com/Closed-FormSolution.html.

[209] Eric Weisstein. Distance, 2016. http://mathworld.wolfram.com/Distance.html.

[210] Eric Weisstein. Hilbert curve, 2016. http://mathworld.wolfram.com/HilbertCurve.html.

[211] Eric Weisstein. Logistic map, 2016. http://mathworld.wolfram.com/LogisticMap.html.

[212] Eric Weisstein. Pi digits, 2016. http://mathworld.wolfram.com/PiDigits.html.

[213] Yu Ying. *The use of the big encryption system (BES) for the cryptanalysis of the advanced encryption system (AES)*. PhD thesis, Technische Universität Darmstadt, 2003.

[214] Amr Youssef and Stafford Tavares. On some algebraic structure in the AES round function. *IACR Cryptology ePrint Archive*, 2002. https://eprint.iacr.org/2002/144.pdf.

[215] Amr Youssef and Stafford Tavares. Affine equivalence in the AES round function. *Discrete Applied Mathematics*, 148 :161–170, 2005.

[216] Michal Zalewski. Strange attractors and tcp/ip sequence number analysis. *lcamtuf.coredump.cx*, 2001. http://lcamtuf.coredump.cx/oldtcp/tcpseq.html.

[217] Michal Zalewski. Strange attractors and tcp/ip sequence number analysis - one year later. *lcamtuf.coredump.cx*, 2002. http://lcamtuf.coredump.cx/newtcp/.

[218] Nakao Zensho. Logic functions over galois field GF(4). *Bulletin of the Faculty of Engineering, University of the Ryukyus*, 29 1985/03 :47–59, 1985.

Troisième partie

Annexes

Annexe A

Résultats obtenus avec les vecteurs de tests

A.1 Chiffrement AES avec une clef de 128 bits

A.1.1 Vecteurs de tests

Clear text = 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
Cipher Key = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

A.1.2 Programme principal

```
if __name__ == "__main__":
    clearBlock = '00112233445566778899aabbccddeeff'
    key128 = '000102030405060708090a0b0c0d0e0f'
    keyExp = keyExpansion(key128, Nk=4, Nb=4, Nr=10)
    cipherBlock = cipher(clearBlock, keyExp, Nb=4, Nr=10)
```

A.1.3 Résultats

```
round[ 0 ].input 00112233445566778899aabbccddeeff
round[ 0 ].k_sch 000102030405060708090a0b0c0d0e0f
round[ 1 ].start 00102030405060708090a0b0c0d0e0f0
round[ 1 ].s_box 63cab7040953d051cd60e0e7ba70e18c
round[ 1 ].s_row 6353e08c0960e104cd70b751bacad0e7
round[ 1 ].m_col 5f72641557f5bc92f7be3b291db9f91a
round[ 1 ].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 2 ].start 89d810e8855ace682d1843d8cb128fe4
round[ 2 ].s_box a761ca9b97be8b45d8ad1a611fc97369
round[ 2 ].s_row a7be1a6997ad739bd8c9ca451f618b61
round[ 2 ].m_col ff87968431d86a51645151fa773ad009
round[ 2 ].k_sch b692cf0b643dbdf1be9bc5006830b3fe
round[ 3 ].start 4915598f55e5d7a0daca94fa1f0a63f7
round[ 3 ].s_box 3b59cb73fcd90ee05774222dc067fb68
round[ 3 ].s_row 3bd92268fc74fb735767cbe0c0590e2d
round[ 3 ].m_col 4c9c1e66f771f0762c3f868e534df256
```

```
round[ 3 ].k_sch b6ff744ed2c2c9bf6c590cbf0469bf41
round[ 4 ].start fa636a2825b339c940668a3157244d17
round[ 4 ].s_box 2dfb02343f6d12dd09337ec75b36e3f0
round[ 4 ].s_row 2d6d7ef03f33e334093602dd5bfb12c7
round[ 4 ].m_col 6385b79ffc538df997be478e7547d691
round[ 4 ].k_sch 47f7f7bc95353e03f96c32bcfd058dfd
round[ 5 ].start 247240236966b3fa6ed2753288425b6c
round[ 5 ].s_box 36400926f9336d2d9fb59d23c42c3950
round[ 5 ].s_row 36339d50f9b539269f2c092dc4406d23
round[ 5 ].m_col f4bcd45432e554d075f1d6c51dd03b3c
round[ 5 ].k_sch 3caaa3e8a99f9deb50f3af57adf622aa
round[ 6 ].start c81677bc9b7ac93b25027992b0261996
round[ 6 ].s_box e847f56514dadde23f77b64fe7f7d490
round[ 6 ].s_row e8dab6901477d4653ff7f5e2e747dd4f
round[ 6 ].m_col 9816ee7400f87f556b2c049c8e5ad036
round[ 6 ].k_sch 5e390f7df7a69296a7553dc10aa31f6b
round[ 7 ].start c62fe109f75eedc3cc79395d84f9cf5d
round[ 7 ].s_box b415f8016858552e4bb6124c5f998a4c
round[ 7 ].s_row b458124c68b68a014b99f82e5f15554c
round[ 7 ].m_col c57e1c159a9bd286f05f4be098c63439
round[ 7 ].k_sch 14f9701ae35fe28c440adf4d4ea9c026
round[ 8 ].start d1876c0f79c4300ab45594add66ff41f
round[ 8 ].s_box 3e175076b61c04678dfc2295f6a8bfc0
round[ 8 ].s_row 3e1c22c0b6fcbf768da85067f6170495
round[ 8 ].m_col baa03de7a1f9b56ed5512cba5f414d23
round[ 8 ].k_sch 47438735a41c65b9e016baf4aebf7ad2
round[ 9 ].start fde3bad205e5d0d73547964ef1fe37f1
round[ 9 ].s_box 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 9 ].s_row 54d990a16ba09ab596bbf40ea111702f
round[ 9 ].m_col e9f74eec023020f61bf2ccf2353c21c7
round[ 9 ].k_sch 549932d1f08557681093ed9cbe2c974e
round[ 10 ].start bd6e7c3df2b5779e0b61216e8b10b689
round[ 10 ].s_box 7a9f102789d5f50b2beffd9f3dca4ea7
round[ 10 ].s_row 7ad5fda789ef4e272bca100b3d9ff59f
round[ 10 ].k_sch 13111d7fe3944a17f307a78b4d2b30c5
round[ 10 ].output 69c4e0d86a7b0430d8cdb78070b4c55a
```

Annexe B

Équations du premier tour pour le mini-AES

Les $b_{0,i}$ représentent les bits de clair, les $b_{1,i}$ représentent les bits en sortie du premier tour et les $b_{2,i}$ représentent les bits en sortie de second tour, soit les bits de chiffré. Enfin, les k_i représentent les bits de clefs.

B.1 Équations pour les 16 bits de X1

$b_{1,1} = k_1 \oplus 1 \oplus b_{0,15}b_{0,16} \oplus b_{0,14} \oplus b_{0,14}b_{0,16} \oplus b_{0,13} \oplus b_{0,13}b_{0,15} \oplus b_{0,13}b_{0,15}b_{0,16} \oplus b_{0,4} \oplus b_{0,3}b_{0,4} \oplus b_{0,2}b_{0,4} \oplus b_{0,2}b_{0,3} \oplus b_{0,2}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,3} \oplus b_{0,1}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,2} \oplus b_{0,1}b_{0,2}b_{0,3} \oplus 1 \oplus k_{16} \oplus k_{14} \oplus k_{14}k_{15} \oplus k_{14}k_{15}k_{16} \oplus k_{13} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{15} \oplus k_1$

$b_{1,2} =$
$k_2 \oplus 1 \oplus b_{0,16} \oplus b_{0,15} \oplus b_{0,15}b_{0,16} \oplus b_{0,14}b_{0,16} \oplus b_{0,14}b_{0,15} \oplus b_{0,13}b_{0,16} \oplus b_{0,13}b_{0,15} \oplus b_{0,13}b_{0,14} \oplus b_{0,13}b_{0,14}b_{0,16} \oplus b_{0,13}b_{0,14}b_{0,15} \oplus b_{0,4} \oplus b_{0,3} \oplus b_{0,2} \oplus b_{0,2}b_{0,3} \oplus b_{0,1} \oplus b_{0,1}b_{0,4} \oplus b_{0,1}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,2} \oplus b_{0,1}b_{0,2}b_{0,4} \oplus b_{0,1}b_{0,2}b_{0,3} \oplus 1 \oplus k_{15}k_{16} \oplus k_{14} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{15} \oplus k_{13}k_{15}k_{16} \oplus k_2$

$b_{1,3} =$
$k_3 \oplus 1 \oplus b_{0,16} \oplus b_{0,15} \oplus b_{0,14} \oplus b_{0,14}b_{0,16} \oplus b_{0,14}b_{0,15} \oplus b_{0,14}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,16} \oplus b_{0,13}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,14} \oplus b_{0,13}b_{0,14}b_{0,15} \oplus b_{0,3}b_{0,4} \oplus b_{0,2} \oplus b_{0,2}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,3} \oplus b_{0,1}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,2}b_{0,4} \oplus 1 \oplus k_{16} \oplus k_{15} \oplus k_{15}k_{16} \oplus k_{14}k_{16} \oplus k_{14}k_{15} \oplus k_{13}k_{16} \oplus k_{13}k_{15} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{16} \oplus k_{13}k_{14}k_{15} \oplus k_3$

$b_{1,4} = k_4 \oplus b_{0,16} \oplus b_{0,14} \oplus b_{0,14}b_{0,15} \oplus b_{0,14}b_{0,15}b_{0,16} \oplus b_{0,13} \oplus b_{0,13}b_{0,14} \oplus b_{0,13}b_{0,14}b_{0,15} \oplus b_{0,4} \oplus b_{0,3} \oplus b_{0,2} \oplus b_{0,2}b_{0,4} \oplus b_{0,2}b_{0,3} \oplus b_{0,2}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,4} \oplus b_{0,1}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,2} \oplus b_{0,1}b_{0,2}b_{0,3} \oplus 1 \oplus k_{15} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{16} \oplus k_{13}k_{15}k_{16} \oplus k_4$

$b_{1,5} = k_5 \oplus 1 \oplus b_{0,16} \oplus b_{0,15}b_{0,16} \oplus b_{0,14}b_{0,16} \oplus b_{0,14}b_{0,15} \oplus b_{0,14}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,15} \oplus b_{0,13}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,14} \oplus b_{0,13}b_{0,14}b_{0,15} \oplus b_{0,3}b_{0,4} \oplus b_{0,2} \oplus b_{0,2}b_{0,4} \oplus b_{0,1} \oplus b_{0,1}b_{0,3} \oplus b_{0,1}b_{0,3}b_{0,4} \oplus 1 \oplus k_{16} \oplus k_{14} \oplus k_{14}k_{15} \oplus k_{14}k_{15}k_{16} \oplus k_{13} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{15} \oplus k_5 \oplus k_1$

$b_{1,6} = k_6 \oplus 1 \oplus b_{0,16} \oplus b_{0,15} \oplus b_{0,14} \oplus b_{0,14}b_{0,15} \oplus b_{0,13} \oplus b_{0,13}b_{0,16} \oplus b_{0,13}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,14} \oplus b_{0,13}b_{0,14}b_{0,16} \oplus b_{0,13}b_{0,14}b_{0,15} \oplus b_{0,4} \oplus b_{0,3} \oplus b_{0,3}b_{0,4} \oplus b_{0,2}b_{0,4} \oplus b_{0,2}b_{0,3} \oplus b_{0,1}b_{0,4} \oplus b_{0,1}b_{0,3} \oplus b_{0,1}b_{0,2} \oplus b_{0,1}b_{0,2}b_{0,4} \oplus b_{0,1}b_{0,2}b_{0,3} \oplus 1 \oplus k_{15}k_{16} \oplus k_{14} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{15} \oplus k_{13}k_{15}k_{16} \oplus k_6 \oplus k_2$

$b_{1,7} =$
$k_7 \oplus 1 \oplus b_{0,15}b_{0,16} \oplus b_{0,14} \oplus b_{0,14}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,15} \oplus b_{0,13}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,14}b_{0,16} \oplus b_{0,4} \oplus b_{0,3} \oplus b_{0,2} \oplus b_{0,2}b_{0,4} \oplus b_{0,2}b_{0,3} \oplus b_{0,2}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,4} \oplus b_{0,1}b_{0,3}b_{0,4} \oplus b_{0,1}b_{0,2} \oplus b_{0,1}b_{0,2}b_{0,3} \oplus 1 \oplus k_{16} \oplus k_{15} \oplus k_{15}k_{16} \oplus k_{14}k_{16} \oplus k_{14}k_{15} \oplus k_{13}k_{16} \oplus k_{13}k_{15} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{16} \oplus k_{13}k_{14}k_{15} \oplus k_7 \oplus k_3$

$b_{1,8} = k_8 \oplus b_{0,16} \oplus b_{0,15} \oplus b_{0,14} \oplus b_{0,14}b_{0,16} \oplus b_{0,14}b_{0,15} \oplus b_{0,14}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,16} \oplus b_{0,13}b_{0,15}b_{0,16} \oplus b_{0,13}b_{0,14} \oplus b_{0,13}b_{0,14}b_{0,15} \oplus b_{0,4} \oplus b_{0,2} \oplus b_{0,2}b_{0,3} \oplus b_{0,2}b_{0,3}b_{0,4} \oplus b_{0,1} \oplus b_{0,1}b_{0,2} \oplus b_{0,1}b_{0,2}b_{0,3} \oplus 1 \oplus k_{15} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{16} \oplus k_{13}k_{15}k_{16} \oplus k_8 \oplus k_4$

$b_{1,9} = k_9 \oplus 1 \oplus b_{0,12} \oplus b_{0,11}b_{0,12} \oplus b_{0,10}b_{0,12} \oplus b_{0,10}b_{0,11} \oplus b_{0,10}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,11} \oplus b_{0,9}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,10} \oplus b_{0,9}b_{0,10}b_{0,11} \oplus b_{0,7}b_{0,8} \oplus b_{0,6} \oplus b_{0,6}b_{0,8} \oplus b_{0,5} \oplus b_{0,5}b_{0,7} \oplus b_{0,5}b_{0,7}b_{0,8} \oplus 1 \oplus k_{16} \oplus k_{14} \oplus k_{14}k_{15} \oplus k_{14}k_{15}k_{16} \oplus k_{13} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{15} \oplus k_9 \oplus k_5 \oplus k_1$

$b_{1,10} = k_{10} \oplus 1 \oplus b_{0,12} \oplus b_{0,11} \oplus b_{0,10} \oplus b_{0,10}b_{0,11} \oplus b_{0,9} \oplus b_{0,9}b_{0,12} \oplus b_{0,9}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,10} \oplus b_{0,9}b_{0,10}b_{0,12} \oplus b_{0,9}b_{0,10}b_{0,11} \oplus b_{0,8} \oplus b_{0,7} \oplus b_{0,7}b_{0,8} \oplus b_{0,6}b_{0,8} \oplus b_{0,6}b_{0,7} \oplus b_{0,5}b_{0,8} \oplus b_{0,5}b_{0,7} \oplus b_{0,5}b_{0,6} \oplus b_{0,5}b_{0,6}b_{0,8} \oplus b_{0,5}b_{0,6}b_{0,7} \oplus 1 \oplus k_{15}k_{16} \oplus k_{14} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{15} \oplus k_{13}k_{15}k_{16} \oplus k_{10} \oplus k_6 \oplus k_2$

$b_{1,11} =$
$k_{11} \oplus 1 \oplus b_{0,11}b_{0,12} \oplus b_{0,10} \oplus b_{0,10}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,11} \oplus b_{0,9}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,10}b_{0,12} \oplus b_{0,8} \oplus b_{0,7} \oplus b_{0,6} \oplus b_{0,6}b_{0,8} \oplus b_{0,6}b_{0,7} \oplus b_{0,6}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,8} \oplus b_{0,5}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,6} \oplus b_{0,5}b_{0,6}b_{0,7} \oplus 1 \oplus k_{16} \oplus k_{15} \oplus k_{15}k_{16} \oplus k_{14}k_{16} \oplus k_{14}k_{15} \oplus k_{13}k_{16} \oplus k_{13}k_{15} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{16} \oplus k_{13}k_{14}k_{15} \oplus k_{11} \oplus k_7 \oplus k_3$

$b_{1,12} = k_{12} \oplus b_{0,12} \oplus b_{0,11} \oplus b_{0,10} \oplus b_{0,10}b_{0,12} \oplus b_{0,10}b_{0,11} \oplus b_{0,10}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,12} \oplus b_{0,9}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,10} \oplus b_{0,9}b_{0,10}b_{0,11} \oplus b_{0,8} \oplus b_{0,6} \oplus b_{0,6}b_{0,7} \oplus b_{0,6}b_{0,7}b_{0,8} \oplus b_{0,5} \oplus b_{0,5}b_{0,6} \oplus b_{0,5}b_{0,6}b_{0,7} \oplus 1 \oplus k_{15} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{16} \oplus k_{13}k_{15}k_{16} \oplus k_{12} \oplus k_8 \oplus k_4$

$b_{1,13} = k_{13} \oplus 1 \oplus b_{0,11}b_{0,12} \oplus b_{0,10} \oplus b_{0,10}b_{0,12} \oplus b_{0,9} \oplus b_{0,9}b_{0,11} \oplus b_{0,9}b_{0,11}b_{0,12} \oplus b_{0,8} \oplus b_{0,7}b_{0,8} \oplus b_{0,6}b_{0,8} \oplus b_{0,6}b_{0,7} \oplus b_{0,6}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,7} \oplus b_{0,5}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,6} \oplus b_{0,5}b_{0,6}b_{0,7} \oplus 1 \oplus k_{16} \oplus k_{14} \oplus k_{14}k_{15} \oplus k_{14}k_{15}k_{16} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{15} \oplus k_9 \oplus k_5 \oplus k_1$

$b_{1,14} = k_{14} \oplus 1 \oplus b_{0,12} \oplus b_{0,11} \oplus b_{0,11}b_{0,12} \oplus b_{0,10}b_{0,12} \oplus b_{0,10}b_{0,11} \oplus b_{0,9}b_{0,12} \oplus b_{0,9}b_{0,11} \oplus b_{0,9}b_{0,10} \oplus b_{0,9}b_{0,10}b_{0,12} \oplus b_{0,9}b_{0,10}b_{0,11} \oplus b_{0,8} \oplus b_{0,7} \oplus b_{0,6} \oplus b_{0,6}b_{0,7} \oplus b_{0,5} \oplus b_{0,5}b_{0,8} \oplus b_{0,5}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,6} \oplus b_{0,5}b_{0,6}b_{0,8} \oplus b_{0,5}b_{0,6}b_{0,7} \oplus 1 \oplus k_{15}k_{16} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{15} \oplus k_{13}k_{15}k_{16} \oplus k_{10} \oplus k_6 \oplus k_2$

$b_{1,15} =$
$k_{15} \oplus 1 \oplus b_{0,12} \oplus b_{0,11} \oplus b_{0,10} \oplus b_{0,10}b_{0,12} \oplus b_{0,10}b_{0,11} \oplus b_{0,10}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,12} \oplus b_{0,9}b_{0,11}b_{0,12} \oplus b_{0,9}b_{0,10} \oplus b_{0,9}b_{0,10}b_{0,11} \oplus b_{0,7}b_{0,8} \oplus b_{0,6} \oplus b_{0,6}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,7} \oplus b_{0,5}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,6}b_{0,8} \oplus 1 \oplus k_{16} \oplus k_{15}k_{16} \oplus k_{14}k_{16} \oplus k_{14}k_{15} \oplus k_{13}k_{16} \oplus k_{13}k_{15} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{16} \oplus k_{13}k_{14}k_{15} \oplus k_{11} \oplus k_7 \oplus k_3$

$b_{1,16} = k_{16} \oplus b_{0,12} \oplus b_{0,10} \oplus b_{0,10}b_{0,11} \oplus b_{0,10}b_{0,11}b_{0,12} \oplus b_{0,9} \oplus b_{0,9}b_{0,10} \oplus b_{0,9}b_{0,10}b_{0,11} \oplus b_{0,8} \oplus b_{0,7} \oplus b_{0,6} \oplus b_{0,6}b_{0,8} \oplus b_{0,6}b_{0,7} \oplus b_{0,6}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,8} \oplus b_{0,5}b_{0,7}b_{0,8} \oplus b_{0,5}b_{0,6} \oplus b_{0,5}b_{0,6}b_{0,7} \oplus 1 \oplus k_{16} \oplus k_{15} \oplus k_{14}k_{16} \oplus k_{13} \oplus k_{13}k_{16} \oplus k_{13}k_{15}k_{16} \oplus k_{12} \oplus k_8 \oplus k_4$

B.2 Équations pour les 2 derniers bits de X2

$b_{2,15} = 1 \oplus b_{1,8} \oplus b_{1,7} \oplus b_{1,7}b_{1,8} \oplus b_{1,6}b_{1,8} \oplus b_{1,6}b_{1,7} \oplus b_{1,5}b_{1,8} \oplus b_{1,5}b_{1,7} \oplus b_{1,5}b_{1,6} \oplus b_{1,5}b_{1,6}b_{1,8} \oplus b_{1,5}b_{1,6}b_{1,7} \oplus k_{14} \oplus k_{14}k_{15} \oplus k_{13}k_{16} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{16} \oplus k_{12} \oplus k_{12}k_{16} \oplus k_{12}k_{14}k_{16} \oplus k_{12}k_{14}k_{15} \oplus k_{12}k_{14}k_{15}k_{16} \oplus k_{12}k_{13}k_{15} \oplus k_{12}k_{13}k_{14} \oplus k_{12}k_{13}k_{14}k_{15} \oplus k_{11} \oplus k_{11}k_{16} \oplus k_{11}k_{15} \oplus k_{11}k_{15}k_{16} \oplus k_{11}k_{14}k_{16} \oplus k_{11}k_{14}k_{15}k_{16} \oplus k_{11}k_{13} \oplus k_{11}k_{13}k_{15}k_{16} \oplus k_{11}k_{13}k_{14}k_{16} \oplus k_{11}k_{12} \oplus k_{10} \oplus k_{10}k_{16} \oplus k_{10}k_{14} \oplus k_{10}k_{14}k_{15} \oplus k_{10}k_{14}k_{15}k_{16} \oplus k_{10}k_{13}k_{16} \oplus k_{10}k_{13}k_{14} \oplus k_{10}k_{13}k_{14}k_{15} \oplus k_{10}k_{13}k_{14}k_{15}k_{16} \oplus k_{10}k_{12}k_{16} \oplus k_{10}k_{12}k_{14} \oplus k_{10}k_{12}k_{14}k_{15} \oplus k_{10}k_{12}k_{14}k_{15}k_{16} \oplus k_{10}k_{12}k_{13}k_{14} \oplus k_{10}k_{12}k_{13}k_{14}k_{15} \oplus k_{10}k_{11}k_{16} \oplus k_{10}k_{11}k_{14} \oplus k_{10}k_{11}k_{14}k_{15} \oplus k_{10}k_{11}k_{14}k_{15}k_{16} \oplus k_{10}k_{11}k_{13}k_{14} \oplus k_{10}k_{11}k_{13}k_{14}k_{15} \oplus k_9 \oplus k_9k_{15}k_{16} \oplus k_9k_{14}k_{16} \oplus k_9k_{13}k_{15}k_{16} \oplus k_9k_{13}k_{14} \oplus k_9k_{13}k_{14}k_{15} \oplus k_9k_{13}k_{14}k_{15}k_{16} \oplus k_9k_{12}k_{15}k_{16} \oplus k_9k_{12}k_{14}k_{16} \oplus k_9k_{12}k_{13} \oplus k_9k_{12}k_{13}k_{15} \oplus k_9k_{12}k_{13}k_{15}k_{16} \oplus k_9k_{11}k_{15}k_{16} \oplus k_9k_{11}k_{14}k_{16} \oplus k_9k_{11}k_{13} \oplus k_9k_{11}k_{13}k_{15} \oplus k_9k_{11}k_{13}k_{15}k_{16} \oplus k_9k_{10} \oplus k_9k_{10}k_{15} \oplus k_9k_{10}k_{15}k_{16} \oplus k_9k_{10}k_{14}k_{15} \oplus k_9k_{10}k_{13} \oplus k_9k_{10}k_{13}k_{15} \oplus k_9k_{10}k_{13}k_{15}k_{16} \oplus k_9k_{10}k_{13}k_{14} \oplus k_9k_{10}k_{13}k_{14}k_{16} \oplus k_9k_{10}k_{13}k_{14}k_{15} \oplus k_9k_{10}k_{12} \oplus k_9k_{10}k_{11} \oplus k_8 \oplus k_8k_{16} \oplus k_8k_{14}k_{16} \oplus k_8k_{14}k_{15} \oplus k_8k_{14}k_{15}k_{16} \oplus k_8k_{13}k_{15} \oplus k_8k_{13}k_{14} \oplus k_8k_{13}k_{14}k_{15} \oplus k_8k_{11} \oplus k_8k_{10}k_{16} \oplus k_8k_{10}k_{14} \oplus k_8k_{10}k_{14}k_{15} \oplus k_8k_{10}k_{14}k_{15}k_{16} \oplus k_8k_{10}k_{13}k_{14} \oplus k_8k_{10}k_{13}k_{14}k_{15} \oplus k_8k_9k_{15}k_{16} \oplus k_8k_9k_{14}k_{16} \oplus k_8k_9k_{13} \oplus k_8k_9k_{13}k_{15} \oplus k_8k_9k_{13}k_{15}k_{16} \oplus k_8k_9k_{10} \oplus k_7k_{16} \oplus k_7k_{15} \oplus k_7k_{15}k_{16} \oplus k_7k_{14}k_{16} \oplus k_7k_{14}k_{15}k_{16} \oplus k_7k_{13} \oplus k_7k_{13}k_{15}k_{16} \oplus k_7k_{13}k_{14}k_{16} \oplus k_7k_{12} \oplus k_7k_{10}k_{16} \oplus k_7k_{10}k_{14} \oplus k_7k_{10}k_{14}k_{15} \oplus k_7k_{10}k_{14}k_{15}k_{16} \oplus k_7k_{10}k_{13}k_{14} \oplus k_7k_{10}k_{13}k_{14}k_{15} \oplus k_7k_9k_{15}k_{16} \oplus k_7k_9k_{14}k_{16} \oplus k_7k_9k_{13} \oplus k_7k_9k_{13}k_{15} \oplus k_7k_9k_{13}k_{15}k_{16} \oplus k_7k_9k_{10} \oplus k_7k_8 \oplus k_6 \oplus k_6k_{16} \oplus k_6k_{14} \oplus k_6k_{14}k_{15} \oplus k_6k_{14}k_{15}k_{16} \oplus k_6k_{13}k_{16} \oplus k_6k_{13}k_{14} \oplus k_6k_{13}k_{14}k_{15} \oplus k_6k_{13}k_{14}k_{15}k_{16} \oplus k_6k_{12}k_{16} \oplus k_6k_{12}k_{14} \oplus k_6k_{12}k_{14}k_{15} \oplus k_6k_{12}k_{14}k_{15}k_{16} \oplus k_6k_{12}k_{13}k_{14} \oplus k_6k_{12}k_{13}k_{14}k_{15} \oplus k_6k_{11}k_{16} \oplus k_6k_{11}k_{14} \oplus k_6k_{11}k_{14}k_{15} \oplus k_6k_{11}k_{14}k_{15}k_{16} \oplus k_6k_{11}k_{13}k_{14} \oplus k_6k_{11}k_{13}k_{14}k_{15} \oplus k_6k_9 \oplus k_6k_9k_{15} \oplus k_6k_9k_{15}k_{16} \oplus k_6k_9k_{14}k_{15} \oplus k_6k_9k_{13} \oplus k_6k_9k_{13}k_{15} \oplus k_6k_9k_{13}k_{15}k_{16} \oplus k_6k_9k_{13}k_{14} \oplus k_6k_9k_{13}k_{14}k_{16} \oplus k_6k_9k_{13}k_{14}k_{15} \oplus k_6k_9k_{12} \oplus k_6k_9k_{11} \oplus k_6k_8k_{16} \oplus k_6k_8k_{14} \oplus k_6k_8k_{14}k_{15} \oplus k_6k_8k_{14}k_{15}k_{16} \oplus k_6k_8k_{13}k_{14} \oplus k_6k_8k_{13}k_{14}k_{15} \oplus k_6k_8k_9 \oplus k_6k_7k_{16} \oplus k_6k_7k_{14} \oplus k_6k_7k_{14}k_{15} \oplus k_6k_7k_{14}k_{15}k_{16} \oplus k_6k_7k_{13}k_{14} \oplus k_6k_7k_{13}k_{14}k_{15} \oplus k_6k_7k_9 \oplus k_5 \oplus k_5k_{15}k_{16} \oplus k_5k_{14}k_{16} \oplus k_5k_{13}k_{15}k_{16} \oplus k_5k_{13}k_{14} \oplus k_5k_{13}k_{14}k_{15} \oplus k_5k_{13}k_{14}k_{15}k_{16} \oplus k_5k_{12}k_{15}k_{16} \oplus k_5k_{12}k_{14}k_{16} \oplus k_5k_{12}k_{13} \oplus k_5k_{12}k_{13}k_{15} \oplus k_5k_{12}k_{13}k_{15}k_{16} \oplus$

$k_5k_{11}k_{15}k_{16} \oplus k_5k_{11}k_{14}k_{16} \oplus k_5k_{11}k_{13} \oplus k_5k_{11}k_{13}k_{15} \oplus k_5k_{11}k_{13}k_{15}k_{16} \oplus k_5k_{10} \oplus k_5k_{10}k_{15} \oplus k_5k_{10}k_{15}k_{16} \oplus k_5k_{10}k_{14}k_{15} \oplus k_5k_{10}k_{13} \oplus k_5k_{10}k_{13}k_{15} \oplus k_5k_{10}k_{13}k_{15}k_{16} \oplus k_5k_{10}k_{13}k_{14} \oplus k_5k_{10}k_{13}k_{14}k_{16} \oplus k_5k_{10}k_{13}k_{14}k_{15} \oplus k_5k_{10}k_{12} \oplus k_5k_{10}k_{11} \oplus k_5k_8k_{15}k_{16} \oplus k_5k_8k_{14}k_{16} \oplus k_5k_8k_{13} \oplus k_5k_8k_{13}k_{15} \oplus k_5k_8k_{13}k_{15}k_{16} \oplus k_5k_8k_{10} \oplus k_5k_7k_{15}k_{16} \oplus k_5k_7k_{14}k_{16} \oplus k_5k_7k_{13} \oplus k_5k_7k_{13}k_{15} \oplus k_5k_7k_{13}k_{15}k_{16} \oplus k_5k_7k_{10} \oplus k_5k_6 \oplus k_5k_6k_{15} \oplus k_5k_6k_{15}k_{16} \oplus k_5k_6k_{14}k_{15} \oplus k_5k_6k_{13} \oplus k_5k_6k_{13}k_{15} \oplus k_5k_6k_{13}k_{15}k_{16} \oplus k_5k_6k_{13}k_{14} \oplus k_5k_6k_{13}k_{14}k_{16} \oplus k_5k_6k_{13}k_{14}k_{15} \oplus k_5k_6k_{12} \oplus k_5k_6k_{11} \oplus k_5k_6k_8 \oplus k_5k_6k_7 \oplus k_4 \oplus k_4k_{16} \oplus k_4k_{14}k_{16} \oplus k_4k_{14}k_{15} \oplus k_4k_{14}k_{15}k_{16} \oplus k_4k_{13}k_{15} \oplus k_4k_{13}k_{14} \oplus k_4k_{13}k_{14}k_{15} \oplus k_4k_{11} \oplus k_4k_{10}k_{16} \oplus k_4k_{10}k_{14} \oplus k_4k_{10}k_{14}k_{15} \oplus k_4k_{10}k_{14}k_{15}k_{16} \oplus k_4k_{10}k_{13}k_{14} \oplus k_4k_{10}k_{13}k_{14}k_{15} \oplus k_4k_9k_{15}k_{16} \oplus k_4k_9k_{14}k_{16} \oplus k_4k_9k_{13} \oplus k_4k_9k_{13}k_{15} \oplus k_4k_9k_{13}k_{15}k_{16} \oplus k_4k_9k_{10} \oplus k_4k_7 \oplus k_4k_6k_{16} \oplus k_4k_6k_{14} \oplus k_4k_6k_{14}k_{15} \oplus k_4k_6k_{14}k_{15}k_{16} \oplus k_4k_6k_{13}k_{14} \oplus k_4k_6k_{13}k_{14}k_{15} \oplus k_4k_6k_9 \oplus k_4k_5k_{15}k_{16} \oplus k_4k_5k_{14}k_{16} \oplus k_4k_5k_{13} \oplus k_4k_5k_{13}k_{15} \oplus k_4k_5k_{13}k_{15}k_{16} \oplus k_4k_5k_{10} \oplus k_4k_5k_6 \oplus k_3 \oplus k_3k_{16} \oplus k_3k_{15} \oplus k_3k_{15}k_{16} \oplus k_3k_{14}k_{16} \oplus k_3k_{14}k_{15}k_{16} \oplus k_3k_{13} \oplus k_3k_{13}k_{15}k_{16} \oplus k_3k_{13}k_{14}k_{16} \oplus k_3k_{12} \oplus k_3k_{10}k_{16} \oplus k_3k_{10}k_{14} \oplus k_3k_{10}k_{14}k_{15} \oplus k_3k_{10}k_{14}k_{15}k_{16} \oplus k_3k_{10}k_{13}k_{14} \oplus k_3k_{10}k_{13}k_{14}k_{15} \oplus k_3k_9k_{15}k_{16} \oplus k_3k_9k_{14}k_{16} \oplus k_3k_9k_{13} \oplus k_3k_9k_{13}k_{15} \oplus k_3k_9k_{13}k_{15}k_{16} \oplus k_3k_9k_{10} \oplus k_3k_8 \oplus k_3k_6k_{16} \oplus k_3k_6k_{14} \oplus k_3k_6k_{14}k_{15} \oplus k_3k_6k_{14}k_{15}k_{16} \oplus k_3k_6k_{13}k_{14} \oplus k_3k_6k_{13}k_{14}k_{15} \oplus k_3k_6k_9 \oplus k_3k_5k_{15}k_{16} \oplus k_3k_5k_{14}k_{16} \oplus k_3k_5k_{13} \oplus k_3k_5k_{13}k_{15} \oplus k_3k_5k_{13}k_{15}k_{16} \oplus k_3k_5k_{10} \oplus k_3k_5k_6 \oplus k_3k_4 \oplus k_2 \oplus k_2k_{16} \oplus k_2k_{14} \oplus k_2k_{14}k_{15} \oplus k_2k_{14}k_{15}k_{16} \oplus k_2k_{13}k_{16} \oplus k_2k_{13}k_{14} \oplus k_2k_{13}k_{14}k_{15} \oplus k_2k_{13}k_{14}k_{15}k_{16} \oplus k_2k_{12}k_{16} \oplus k_2k_{12}k_{14} \oplus k_2k_{12}k_{14}k_{15} \oplus k_2k_{12}k_{14}k_{15}k_{16} \oplus k_2k_{12}k_{13}k_{14} \oplus k_2k_{12}k_{13}k_{14}k_{15} \oplus k_2k_{11}k_{16} \oplus k_2k_{11}k_{14} \oplus k_2k_{11}k_{14}k_{15} \oplus k_2k_{11}k_{14}k_{15}k_{16} \oplus k_2k_{11}k_{13}k_{14} \oplus k_2k_{11}k_{13}k_{14}k_{15} \oplus k_2k_9 \oplus k_2k_9k_{15} \oplus k_2k_9k_{15}k_{16} \oplus k_2k_9k_{14}k_{15} \oplus k_2k_9k_{13} \oplus k_2k_9k_{13}k_{15} \oplus k_2k_9k_{13}k_{15}k_{16} \oplus k_2k_9k_{13}k_{14} \oplus k_2k_9k_{13}k_{14}k_{16} \oplus k_2k_9k_{13}k_{14}k_{15} \oplus k_2k_9k_{12} \oplus k_2k_9k_{11} \oplus k_2k_8k_{16} \oplus k_2k_8k_{14} \oplus k_2k_8k_{14}k_{15} \oplus k_2k_8k_{14}k_{15}k_{16} \oplus k_2k_8k_{13}k_{14} \oplus k_2k_8k_{13}k_{14}k_{15} \oplus k_2k_8k_9 \oplus k_2k_7k_{16} \oplus k_2k_7k_{14} \oplus k_2k_7k_{14}k_{15} \oplus k_2k_7k_{14}k_{15}k_{16} \oplus k_2k_7k_{13}k_{14} \oplus k_2k_7k_{13}k_{14}k_{15} \oplus k_2k_7k_9 \oplus k_2k_5 \oplus k_2k_5k_{15} \oplus k_2k_5k_{15}k_{16} \oplus k_2k_5k_{14}k_{15} \oplus k_2k_5k_{13} \oplus k_2k_5k_{13}k_{15} \oplus k_2k_5k_{13}k_{15}k_{16} \oplus k_2k_5k_{13}k_{14} \oplus k_2k_5k_{13}k_{14}k_{16} \oplus k_2k_5k_{13}k_{14}k_{15} \oplus k_2k_5k_{12} \oplus k_2k_5k_{11} \oplus k_2k_5k_8 \oplus k_2k_5k_7 \oplus k_2k_4k_{16} \oplus k_2k_4k_{14} \oplus k_2k_4k_{14}k_{15} \oplus k_2k_4k_{14}k_{15}k_{16} \oplus k_2k_4k_{13}k_{14} \oplus k_2k_4k_{13}k_{14}k_{15} \oplus k_2k_4k_9 \oplus k_2k_4k_5 \oplus k_2k_3k_{16} \oplus k_2k_3k_{14} \oplus k_2k_3k_{14}k_{15} \oplus k_2k_3k_{14}k_{15}k_{16} \oplus k_2k_3k_{13}k_{14} \oplus k_2k_3k_{13}k_{14}k_{15} \oplus k_2k_3k_9 \oplus k_2k_3k_5 \oplus k_1 \oplus k_1k_{15}k_{16} \oplus k_1k_{14}k_{16} \oplus k_1k_{13}k_{15}k_{16} \oplus k_1k_{13}k_{14} \oplus k_1k_{13}k_{14}k_{15} \oplus k_1k_{13}k_{14}k_{15}k_{16} \oplus k_1k_{12}k_{15}k_{16} \oplus k_1k_{12}k_{14}k_{16} \oplus k_1k_{12}k_{13} \oplus k_1k_{12}k_{13}k_{15} \oplus k_1k_{12}k_{13}k_{15}k_{16} \oplus k_1k_{11}k_{15}k_{16} \oplus k_1k_{11}k_{14}k_{16} \oplus k_1k_{11}k_{13} \oplus k_1k_{11}k_{13}k_{15} \oplus k_1k_{11}k_{13}k_{15}k_{16} \oplus k_1k_{10} \oplus k_1k_{10}k_{15} \oplus k_1k_{10}k_{15}k_{16} \oplus k_1k_{10}k_{14}k_{15} \oplus k_1k_{10}k_{13} \oplus k_1k_{10}k_{13}k_{15} \oplus k_1k_{10}k_{13}k_{15}k_{16} \oplus k_1k_{10}k_{13}k_{14} \oplus k_1k_{10}k_{13}k_{14}k_{16} \oplus k_1k_{10}k_{13}k_{14}k_{15} \oplus k_1k_{10}k_{12} \oplus k_1k_{10}k_{11} \oplus k_1k_8k_{15}k_{16} \oplus k_1k_8k_{14}k_{16} \oplus k_1k_8k_{13} \oplus k_1k_8k_{13}k_{15} \oplus k_1k_8k_{13}k_{15}k_{16} \oplus k_1k_8k_{10} \oplus k_1k_7k_{15}k_{16} \oplus k_1k_7k_{14}k_{16} \oplus k_1k_7k_{13} \oplus k_1k_7k_{13}k_{15} \oplus k_1k_7k_{13}k_{15}k_{16} \oplus k_1k_7k_{10} \oplus k_1k_6 \oplus k_1k_6k_{15} \oplus k_1k_6k_{15}k_{16} \oplus k_1k_6k_{14}k_{15} \oplus k_1k_6k_{13} \oplus k_1k_6k_{13}k_{15} \oplus k_1k_6k_{13}k_{15}k_{16} \oplus k_1k_6k_{13}k_{14} \oplus k_1k_6k_{13}k_{14}k_{16} \oplus k_1k_6k_{13}k_{14}k_{15} \oplus k_1k_6k_{12} \oplus k_1k_6k_{11} \oplus k_1k_6k_8 \oplus k_1k_6k_7 \oplus k_1k_4k_{15}k_{16} \oplus k_1k_4k_{14}k_{16} \oplus k_1k_4k_{13} \oplus k_1k_4k_{13}k_{15} \oplus k_1k_4k_{13}k_{15}k_{16} \oplus k_1k_4k_{10} \oplus k_1k_4k_6 \oplus k_1k_3k_{15}k_{16} \oplus k_1k_3k_{14}k_{16} \oplus k_1k_3k_{13} \oplus k_1k_3k_{13}k_{15} \oplus k_1k_3k_{13}k_{15}k_{16} \oplus k_1k_3k_{10} \oplus k_1k_3k_6 \oplus k_1k_2 \oplus k_1k_2k_{15} \oplus k_1k_2k_{15}k_{16} \oplus k_1k_2k_{14}k_{15} \oplus k_1k_2k_{13} \oplus k_1k_2k_{13}k_{15} \oplus k_1k_2k_{13}k_{15}k_{16} \oplus k_1k_2k_{13}k_{14} \oplus k_1k_2k_{13}k_{14}k_{16} \oplus k_1k_2k_{13}k_{14}k_{15} \oplus k_1k_2k_{12} \oplus k_1k_2k_{11} \oplus k_1k_2k_8 \oplus k_1k_2k_7 \oplus k_1k_2k_4 \oplus k_1k_2k_3$

$b_{2,16} = b_{1,7} \oplus b_{1,6}b_{1,8} \oplus b_{1,5} \oplus b_{1,5}b_{1,8} \oplus b_{1,5}b_{1,7}b_{1,8} \oplus 1 \oplus k_{15} \oplus k_{14} \oplus k_{14}k_{16} \oplus k_{14}k_{15}k_{16} \oplus k_{13} \oplus k_{13}k_{15}k_{16} \oplus k_{13}k_{14}k_{15} \oplus k_{13}k_{14}k_{15}k_{16} \oplus k_{12} \oplus k_{12}k_{15}k_{16} \oplus k_{12}k_{14}k_{16} \oplus k_{12}k_{14}k_{15} \oplus k_{12}k_{14}k_{15}k_{16} \oplus k_{12}k_{13} \oplus k_{12}k_{13}k_{14} \oplus k_{12}k_{13}k_{14}k_{16} \oplus k_{12}k_{13}k_{14}k_{15} \oplus k_{11}k_{16} \oplus k_{11}k_{15} \oplus k_{11}k_{15}k_{16} \oplus k_{11}k_{14} \oplus k_{11}k_{14}k_{15} \oplus k_{11}k_{13} \oplus k_{11}k_{13}k_{16} \oplus k_{11}k_{13}k_{14} \oplus k_{11}k_{13}k_{14}k_{15} \oplus k_{11}k_{13}k_{14}k_{15}k_{16} \oplus k_{11}k_{12} \oplus k_{11}k_{12}k_{16} \oplus k_{11}k_{12}k_{14} \oplus k_{11}k_{12}k_{14}k_{15} \oplus k_{11}k_{12}k_{14}k_{15}k_{16} \oplus k_{11}k_{12}k_{13}k_{14} \oplus k_{11}k_{12}k_{13}k_{14}k_{15} \oplus k_{10} \oplus k_{10}k_{16} \oplus k_{10}k_{15} \oplus k_{10}k_{14}k_{16} \oplus k_{10}k_{13} \oplus k_{10}k_{13}k_{16} \oplus k_{10}k_{13}k_{15}k_{16} \oplus k_{10}k_{12} \oplus k_9 \oplus k_9k_{13}k_{15} \oplus k_9k_{13}k_{15}k_{16} \oplus k_9k_{13}k_{14}k_{16} \oplus k_9k_{13}k_{14}k_{15} \oplus k_9k_{13}k_{14}k_{15}k_{16} \oplus k_9k_{12}k_{16} \oplus k_9k_{12}k_{15}k_{16} \oplus k_9k_{12}k_{14}k_{16} \oplus k_9k_{12}k_{14}k_{15} \oplus k_9k_{12}k_{13}k_{16} \oplus k_9k_{12}k_{13}k_{15} \oplus k_9k_{12}k_{13}k_{14} \oplus k_9k_{12}k_{13}k_{14}k_{16} \oplus k_9k_{12}k_{13}k_{14}k_{15} \oplus k_9k_{11} \oplus k_9k_{11}k_{16} \oplus k_9k_{11}k_{15} \oplus k_9k_{11}k_{14}k_{16} \oplus k_9k_{11}k_{13} \oplus k_9k_{11}k_{13}k_{16} \oplus k_9k_{11}k_{13}k_{15}k_{16} \oplus k_9k_{11}k_{12} \oplus k_8k_{15}k_{16} \oplus k_8k_{14}k_{16} \oplus k_8k_{14}k_{15} \oplus k_8k_{14}k_{15}k_{16} \oplus k_8k_{13} \oplus k_8k_{13}k_{14} \oplus k_8k_{13}k_{14}k_{16} \oplus k_8k_{13}k_{14}k_{15} \oplus k_8k_{11} \oplus k_8k_{11}k_{16} \oplus k_8k_{11}k_{14} \oplus k_8k_{11}k_{14}k_{15} \oplus k_8k_{11}k_{14}k_{15}k_{16} \oplus k_8k_{11}k_{13}k_{14} \oplus k_8k_{11}k_{13}k_{14}k_{15} \oplus k_8k_{10} \oplus k_8k_9k_{16} \oplus k_8k_9k_{15}k_{16} \oplus k_8k_9k_{14}k_{16} \oplus k_8k_9k_{14}k_{15} \oplus k_8k_9k_{13}k_{16} \oplus k_8k_9k_{13}k_{15} \oplus k_8k_9k_{13}k_{14} \oplus k_8k_9k_{13}k_{14}k_{16} \oplus k_8k_9k_{13}k_{14}k_{15} \oplus k_8k_9k_{11} \oplus k_7k_{16} \oplus k_7k_{15} \oplus k_7k_{15}k_{16} \oplus k_7k_{14} \oplus k_7k_{14}k_{15} \oplus k_7k_{13} \oplus k_7k_{13}k_{16} \oplus k_7k_{13}k_{14} \oplus k_7k_{13}k_{14}k_{15} \oplus k_7k_{13}k_{14}k_{15}k_{16} \oplus k_7k_{12} \oplus k_7k_{12}k_{16} \oplus k_7k_{12}k_{14} \oplus k_7k_{12}k_{14}k_{15} \oplus k_7k_{12}k_{14}k_{15}k_{16} \oplus k_7k_{12}k_{13}k_{14} \oplus k_7k_{12}k_{13}k_{14}k_{15} \oplus k_7k_9 \oplus k_7k_9k_{16} \oplus k_7k_9k_{15} \oplus k_7k_9k_{14}k_{16} \oplus k_7k_9k_{13} \oplus k_7k_9k_{13}k_{16} \oplus k_7k_9k_{13}k_{15}k_{16} \oplus k_7k_9k_{12} \oplus k_7k_8 \oplus k_7k_8k_{16} \oplus k_7k_8k_{14} \oplus k_7k_8k_{14}k_{15} \oplus k_7k_8k_{14}k_{15}k_{16} \oplus k_7k_8k_{13}k_{14} \oplus k_7k_8k_{13}k_{14}k_{15} \oplus k_7k_8k_9 \oplus k_6 \oplus k_6k_{16} \oplus k_6k_{15} \oplus k_6k_{14}k_{16} \oplus k_6k_{13} \oplus k_6k_{13}k_{16} \oplus k_6k_{13}k_{15}k_{16} \oplus k_6k_{12} \oplus k_6k_8 \oplus k_5 \oplus k_5k_{13}k_{15} \oplus k_5k_{13}k_{15}k_{16} \oplus k_5k_{13}k_{14}k_{16} \oplus k_5k_{13}k_{14}k_{15} \oplus k_5k_{13}k_{14}k_{15}k_{16} \oplus k_5k_{12}k_{16} \oplus k_5k_{12}k_{15}k_{16} \oplus k_5k_{12}k_{14}k_{16} \oplus k_5k_{12}k_{14}k_{15} \oplus k_5k_{12}k_{13}k_{16} \oplus k_5k_{12}k_{13}k_{15} \oplus k_5k_{12}k_{13}k_{14} \oplus k_5k_{12}k_{13}k_{14}k_{16} \oplus k_5k_{12}k_{13}k_{14}k_{15} \oplus k_5k_{11} \oplus k_5k_{11}k_{16} \oplus k_5k_{11}k_{15} \oplus k_5k_{11}k_{14}k_{16} \oplus k_5k_{11}k_{13} \oplus k_5k_{11}k_{13}k_{16} \oplus k_5k_{11}k_{13}k_{15}k_{16} \oplus k_5k_{11}k_{12} \oplus k_5k_8k_{16} \oplus k_5k_8k_{15}k_{16} \oplus k_5k_8k_{14}k_{16} \oplus k_5k_8k_{14}k_{15} \oplus k_5k_8k_{13}k_{16} \oplus k_5k_8k_{13}k_{15} \oplus$

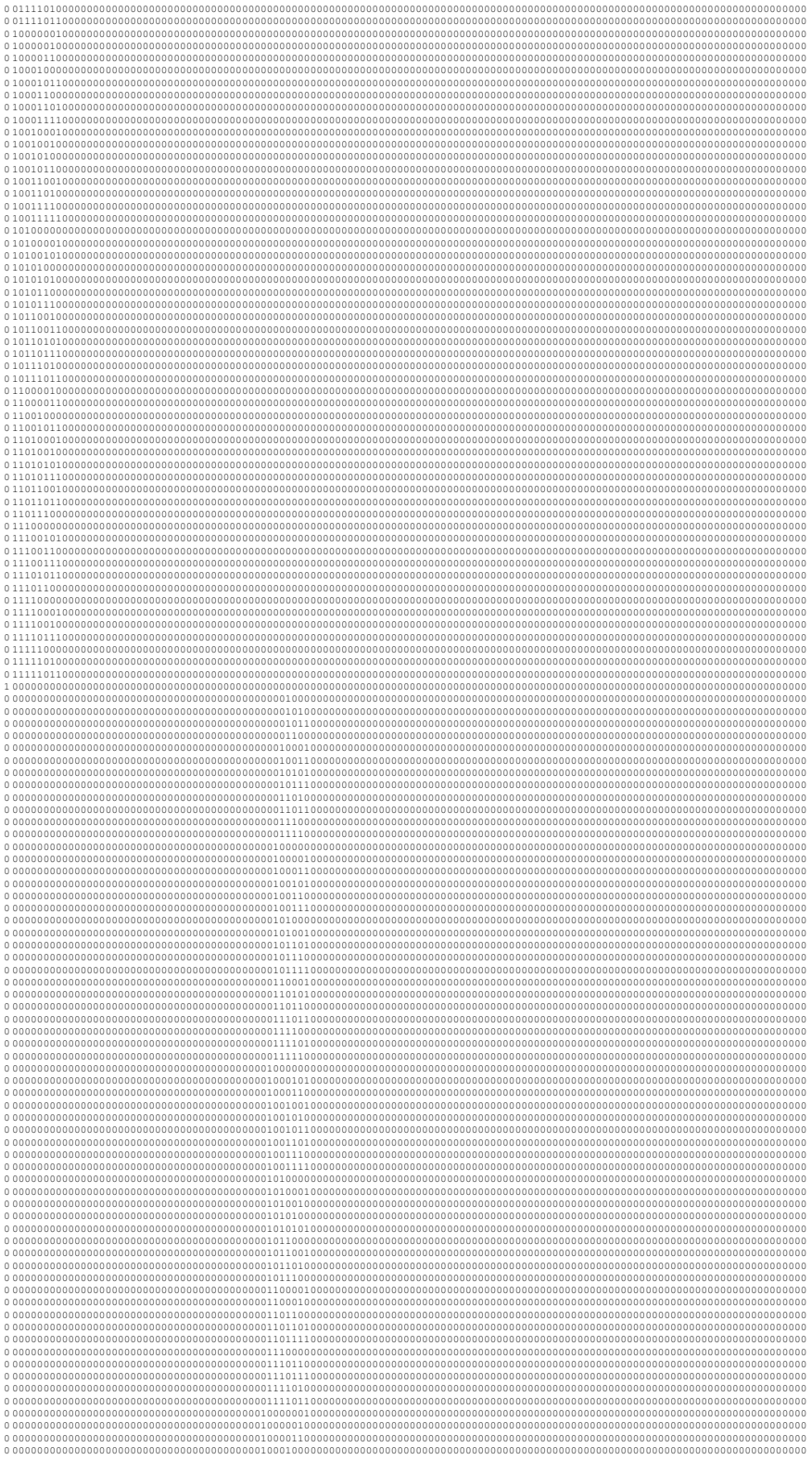
$$k_5k_8k_{13}k_{14} \oplus k_5k_8k_{13}k_{14}k_{16} \oplus k_5k_8k_{13}k_{14}k_{15} \oplus k_5k_8k_{11} \oplus k_5k_7 \oplus k_5k_7k_{16} \oplus k_5k_7k_{15} \oplus k_5k_7k_{14}k_{16} \oplus k_5k_7k_{13} \oplus k_5k_7k_{13}k_{16} \oplus k_5k_7k_{13}k_{15}k_{16} \oplus k_5k_7k_{12} \oplus k_5k_7k_8 \oplus k_4 \oplus k_4k_{15}k_{16} \oplus k_4k_{14}k_{16} \oplus k_4k_{14}k_{15} \oplus k_4k_{14}k_{15}k_{16} \oplus k_4k_{13} \oplus k_4k_{13}k_{14} \oplus k_4k_{13}k_{14}k_{16} \oplus k_4k_{13}k_{14}k_{15} \oplus k_4k_{11} \oplus k_4k_{11}k_{16} \oplus k_4k_{11}k_{14} \oplus k_4k_{11}k_{14}k_{15} \oplus k_4k_{11}k_{14}k_{15}k_{16} \oplus k_4k_{11}k_{13}k_{14} \oplus k_4k_{11}k_{13}k_{14}k_{15} \oplus k_4k_{10} \oplus k_4k_9k_{16} \oplus k_4k_9k_{15}k_{16} \oplus k_4k_9k_{14}k_{16} \oplus k_4k_9k_{14}k_{15} \oplus k_4k_9k_{13}k_{16} \oplus k_4k_9k_{13}k_{15} \oplus k_4k_9k_{13}k_{14} \oplus k_4k_9k_{13}k_{14}k_{16} \oplus k_4k_9k_{13}k_{14}k_{15} \oplus k_4k_9k_{11} \oplus k_4k_7 \oplus k_4k_7k_{16} \oplus k_4k_7k_{14} \oplus k_4k_7k_{14}k_{15} \oplus k_4k_7k_{14}k_{15}k_{16} \oplus k_4k_7k_{13}k_{14} \oplus k_4k_7k_{13}k_{14}k_{15} \oplus k_4k_7k_9 \oplus k_4k_6 \oplus k_4k_5k_{16} \oplus k_4k_5k_{15}k_{16} \oplus k_4k_5k_{14}k_{16} \oplus k_4k_5k_{14}k_{15} \oplus k_4k_5k_{13}k_{16} \oplus k_4k_5k_{13}k_{15} \oplus k_4k_5k_{13}k_{14} \oplus k_4k_5k_{13}k_{14}k_{16} \oplus k_4k_5k_{13}k_{14}k_{15} \oplus k_4k_5k_{11} \oplus k_4k_5k_7 \oplus k_3k_{16} \oplus k_3k_{15} \oplus k_3k_{15}k_{16} \oplus k_3k_{14} \oplus k_3k_{14}k_{15} \oplus k_3k_{13} \oplus k_3k_{13}k_{16} \oplus k_3k_{13}k_{14} \oplus k_3k_{13}k_{14}k_{15} \oplus k_3k_{13}k_{14}k_{15}k_{16} \oplus k_3k_{12} \oplus k_3k_{12}k_{16} \oplus k_3k_{12}k_{14} \oplus k_3k_{12}k_{14}k_{15} \oplus k_3k_{12}k_{14}k_{15}k_{16} \oplus k_3k_{12}k_{13}k_{14} \oplus k_3k_{12}k_{13}k_{14}k_{15} \oplus k_3k_9 \oplus k_3k_9k_{16} \oplus k_3k_9k_{15} \oplus k_3k_9k_{14}k_{16} \oplus k_3k_9k_{13} \oplus k_3k_9k_{13}k_{16} \oplus k_3k_9k_{13}k_{15}k_{16} \oplus k_3k_9k_{12} \oplus k_3k_8 \oplus k_3k_8k_{16} \oplus k_3k_8k_{14} \oplus k_3k_8k_{14}k_{15} \oplus k_3k_8k_{14}k_{15}k_{16} \oplus k_3k_8k_{13}k_{14} \oplus k_3k_8k_{13}k_{14}k_{15} \oplus k_3k_8k_9 \oplus k_3k_5 \oplus k_3k_5k_{16} \oplus k_3k_5k_{15} \oplus k_3k_5k_{14}k_{16} \oplus k_3k_5k_{13} \oplus k_3k_5k_{13}k_{16} \oplus k_3k_5k_{13}k_{15}k_{16} \oplus k_3k_5k_{12} \oplus k_3k_5k_8 \oplus k_3k_4 \oplus k_3k_4k_{16} \oplus k_3k_4k_{14} \oplus k_3k_4k_{14}k_{15} \oplus k_3k_4k_{14}k_{15}k_{16} \oplus k_3k_4k_{13}k_{14} \oplus k_3k_4k_{13}k_{14}k_{15} \oplus k_3k_4k_9 \oplus k_3k_4k_5 \oplus k_2 \oplus k_2k_{16} \oplus k_2k_{15} \oplus k_2k_{14}k_{16} \oplus k_2k_{13} \oplus k_2k_{13}k_{16} \oplus k_2k_{13}k_{15}k_{16} \oplus k_2k_{12} \oplus k_2k_8 \oplus k_2k_4 \oplus k_1 \oplus k_1k_{13}k_{15} \oplus k_1k_{13}k_{15}k_{16} \oplus k_1k_{13}k_{14}k_{16} \oplus k_1k_{13}k_{14}k_{15} \oplus k_1k_{13}k_{14}k_{15}k_{16} \oplus k_1k_{12}k_{16} \oplus k_1k_{12}k_{15}k_{16} \oplus k_1k_{12}k_{14}k_{16} \oplus k_1k_{12}k_{14}k_{15} \oplus k_1k_{12}k_{13}k_{16} \oplus k_1k_{12}k_{13}k_{15} \oplus k_1k_{12}k_{13}k_{14} \oplus k_1k_{12}k_{13}k_{14}k_{16} \oplus k_1k_{12}k_{13}k_{14}k_{15} \oplus k_1k_{11} \oplus k_1k_{11}k_{16} \oplus k_1k_{11}k_{15} \oplus k_1k_{11}k_{14}k_{16} \oplus k_1k_{11}k_{13} \oplus k_1k_{11}k_{13}k_{16} \oplus k_1k_{11}k_{13}k_{15}k_{16} \oplus k_1k_{11}k_{12} \oplus k_1k_8k_{16} \oplus k_1k_8k_{15}k_{16} \oplus k_1k_8k_{14}k_{16} \oplus k_1k_8k_{14}k_{15} \oplus k_1k_8k_{13}k_{16} \oplus k_1k_8k_{13}k_{15} \oplus k_1k_8k_{13}k_{14} \oplus k_1k_8k_{13}k_{14}k_{16} \oplus k_1k_8k_{13}k_{14}k_{15} \oplus k_1k_8k_{11} \oplus k_1k_7 \oplus k_1k_7k_{16} \oplus k_1k_7k_{15} \oplus k_1k_7k_{14}k_{16} \oplus k_1k_7k_{13} \oplus k_1k_7k_{13}k_{16} \oplus k_1k_7k_{13}k_{15}k_{16} \oplus k_1k_7k_{12} \oplus k_1k_7k_8 \oplus k_1k_4k_{16} \oplus k_1k_4k_{15}k_{16} \oplus k_1k_4k_{14}k_{16} \oplus k_1k_4k_{14}k_{15} \oplus k_1k_4k_{13}k_{16} \oplus k_1k_4k_{13}k_{15} \oplus k_1k_4k_{13}k_{14} \oplus k_1k_4k_{13}k_{14}k_{16} \oplus k_1k_4k_{13}k_{14}k_{15} \oplus k_1k_4k_{11} \oplus k_1k_4k_7 \oplus k_1k_3 \oplus k_1k_3k_{16} \oplus k_1k_3k_{15} \oplus k_1k_3k_{14}k_{16} \oplus k_1k_3k_{13} \oplus k_1k_3k_{13}k_{16} \oplus k_1k_3k_{13}k_{15}k_{16} \oplus k_1k_3k_{12} \oplus k_1k_3k_8 \oplus k_1k_3k_4$$

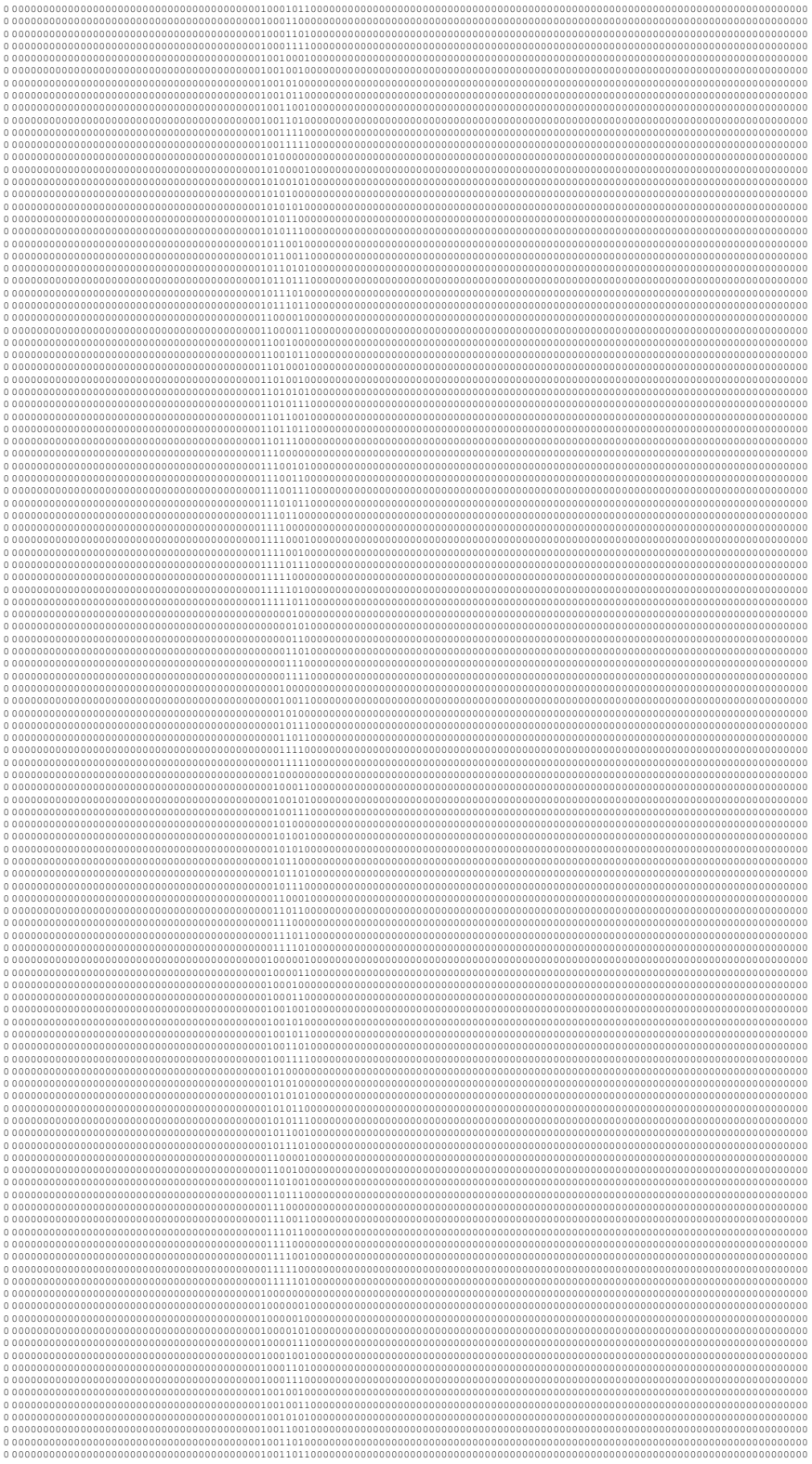
Annexe C

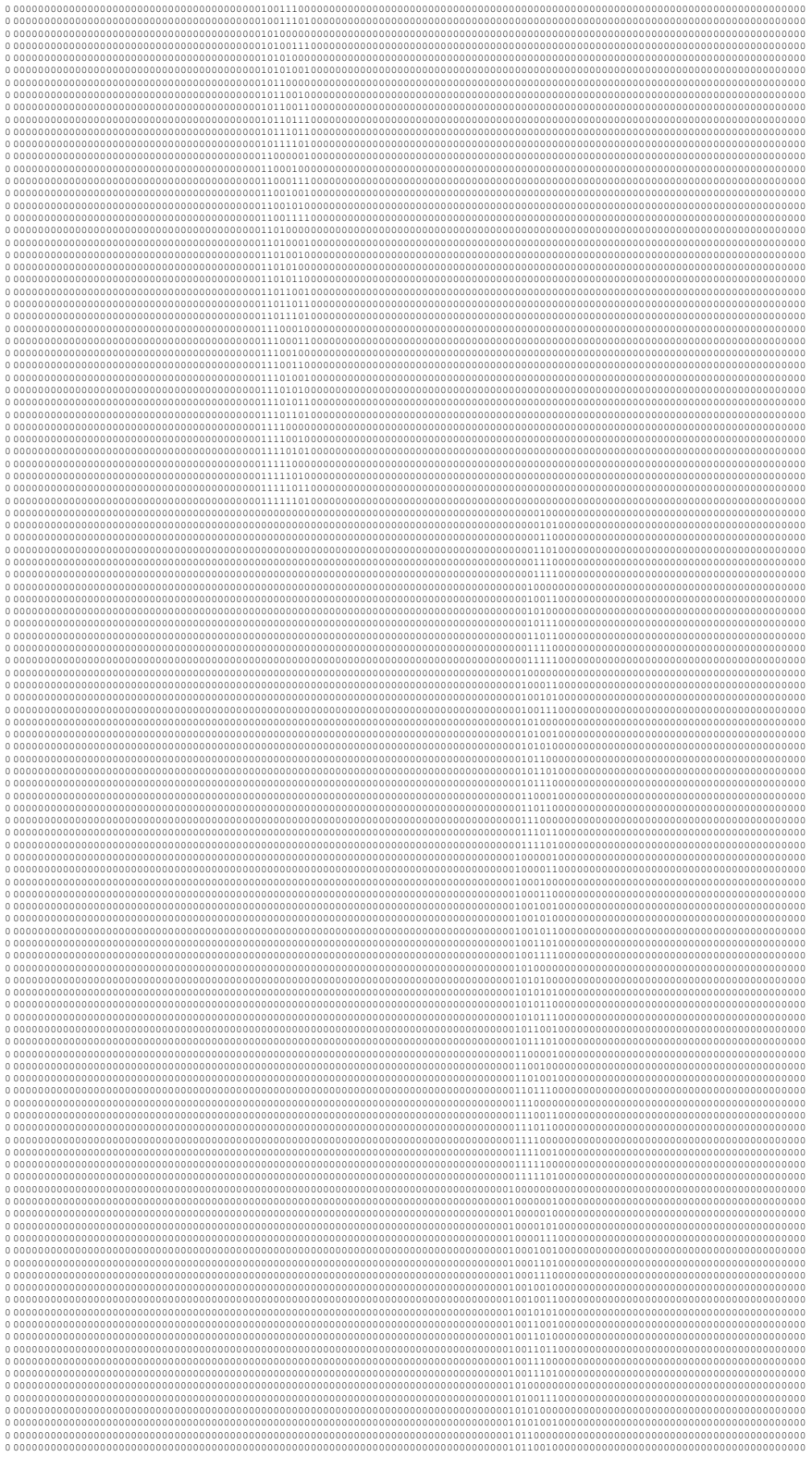
Fichier pour le bit b1 d'un tour de l'AES

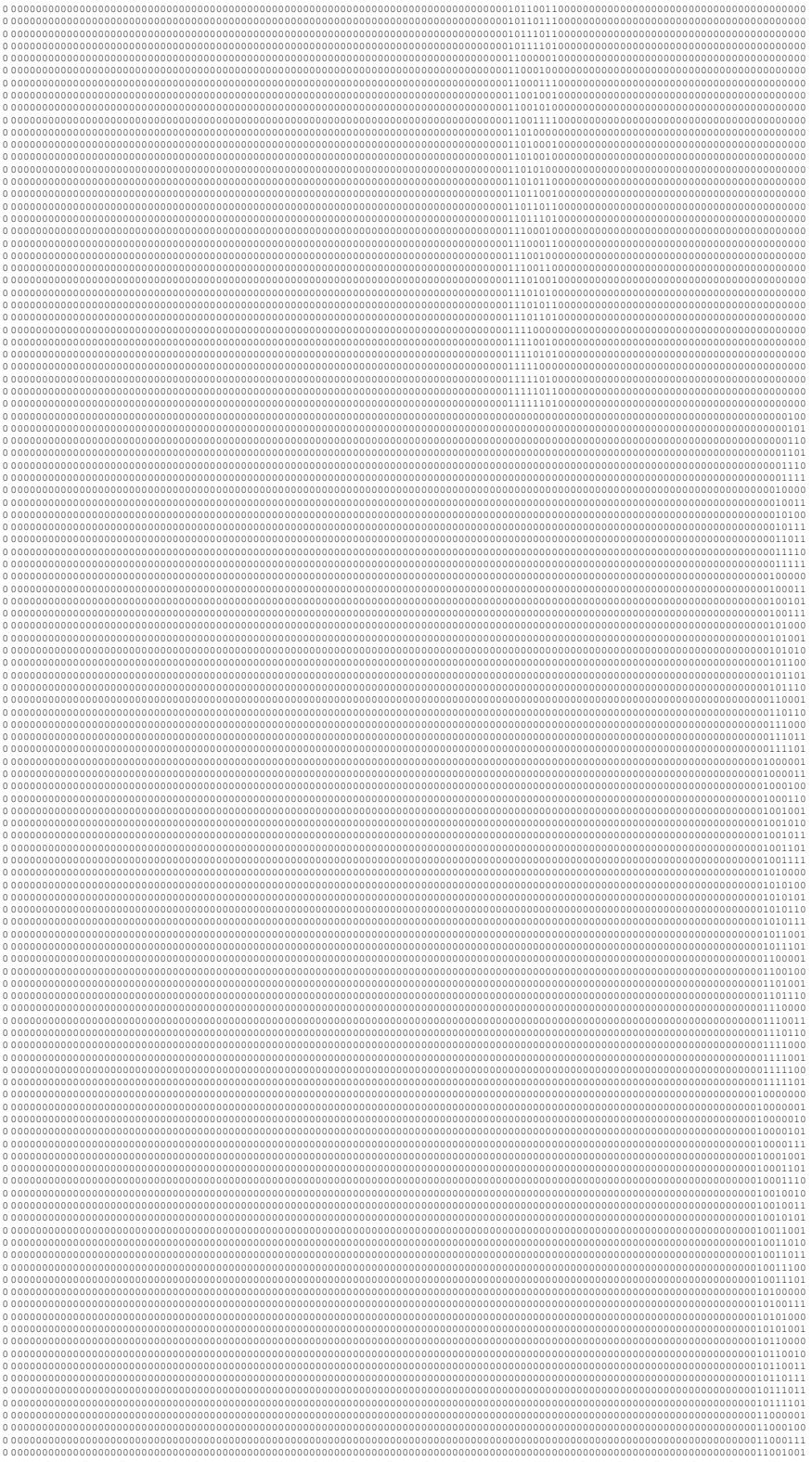
Chaque ligne est composée de 0 et de 1. Chacun d'entre eux correspond à un bit de bloc. Les lignes suivant le mot `addRoundKey` représentent des bits de clef tandis que les lignes suivant le mot `Round` représentent des bits de clair. À chaque ligne correspond un monome de l'équation booléenne et le passage d'une ligne à la suivante correspond à un `XOR`.

```
## addRoundKey
0 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## Round
1 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00001010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00001011000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00001100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00010001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00010011000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00010101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00010111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00011010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00011011000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00011100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00011110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00100001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00100011000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00100101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00100110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00100111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00101001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00101101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00101110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00101111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00110001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00110101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00110110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00111011000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00111100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00111101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 00111110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01000101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01000110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01001001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01001010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01001011000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01001101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01001110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01001111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01010001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01010010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01010100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01010101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01011000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01011001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01011010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01011100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01100001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01100010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01101100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01101101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01101111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01110110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0 01110111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```









```
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011001010
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011001111
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011010000
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011010001
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011010010
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011010100
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011010110
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011011001
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011011011
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011011101
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011100010
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011100011
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011100100
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011100110
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011101001
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011101010
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011101011
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011101101
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011110000
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011110010
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011110101
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011111000
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011111010
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011111011
0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011111101
## end
```

Annexe D

Bilan des 128 fonctions booléennes de chiffrement de l'AES

Ce tableau présente, par bit de clair, le bilan de l'équation booléenne décrivant un tour de chiffrement de l'AES avant application du XOR avec la clef. Ainsi, pour chaque bit nous avons le nombre de monômes et la répartition des monômes par degré.

Bit	Nbr. monômes	Nombre de monômes de degré :								
		0	1	2	3	4	5	6	7	8
0	554	2	18	50	140	159	120	55	10	0
1	564	5	15	44	157	154	126	53	10	0
2	604	3	19	55	154	160	149	52	12	0
3	885	0	31	93	197	240	216	87	21	0
4	918	0	28	77	215	255	213	104	26	0
5	701	2	20	64	174	174	161	83	23	0
6	883	5	28	94	202	231	201	100	22	0
7	616	3	20	68	142	165	138	67	13	0
8	554	2	18	50	140	159	120	55	10	0
9	564	5	15	44	157	154	126	53	10	0
10	604	3	19	55	154	160	149	52	12	0
11	885	0	31	93	197	240	216	87	21	0
12	918	0	28	77	215	255	213	104	26	0
13	701	2	20	64	174	174	161	83	23	0
14	883	5	28	94	202	231	201	100	22	0
15	616	3	20	68	142	165	138	67	13	0
16	554	2	18	50	140	159	120	55	10	0
17	564	5	15	44	157	154	126	53	10	0
18	604	3	19	55	154	160	149	52	12	0
19	885	0	31	93	197	240	216	87	21	0
20	918	0	28	77	215	255	213	104	26	0
21	701	2	20	64	174	174	161	83	23	0
22	883	5	28	94	202	231	201	100	22	0
23	616	3	20	68	142	165	138	67	13	0
24	554	2	18	50	140	159	120	55	10	0

Bit	Nbr. monômes	Nombre de monômes de degré :								
		0	1	2	3	4	5	6	7	8
25	564	5	15	44	157	154	126	53	10	0
26	604	3	19	55	154	160	149	52	12	0
27	885	0	31	93	197	240	216	87	21	0
28	918	0	28	77	215	255	213	104	26	0
29	701	2	20	64	174	174	161	83	23	0
30	883	5	28	94	202	231	201	100	22	0
31	616	3	20	68	142	165	138	67	13	0
32	554	2	18	50	140	159	120	55	10	0
33	564	5	15	44	157	154	126	53	10	0
34	604	3	19	55	154	160	149	52	12	0
35	885	0	31	93	197	240	216	87	21	0
36	918	0	28	77	215	255	213	104	26	0
37	701	2	20	64	174	174	161	83	23	0
38	883	5	28	94	202	231	201	100	22	0
39	616	3	20	68	142	165	138	67	13	0
40	554	2	18	50	140	159	120	55	10	0
41	564	5	15	44	157	154	126	53	10	0
42	604	3	19	55	154	160	149	52	12	0
43	885	0	31	93	197	240	216	87	21	0
44	918	0	28	77	215	255	213	104	26	0
45	701	2	20	64	174	174	161	83	23	0
46	883	5	28	94	202	231	201	100	22	0
47	616	3	20	68	142	165	138	67	13	0
48	554	2	18	50	140	159	120	55	10	0
49	564	5	15	44	157	154	126	53	10	0
50	604	3	19	55	154	160	149	52	12	0
51	885	0	31	93	197	240	216	87	21	0
52	918	0	28	77	215	255	213	104	26	0
53	701	2	20	64	174	174	161	83	23	0
54	883	5	28	94	202	231	201	100	22	0
55	616	3	20	68	142	165	138	67	13	0
56	554	2	18	50	140	159	120	55	10	0
57	564	5	15	44	157	154	126	53	10	0
58	604	3	19	55	154	160	149	52	12	0
59	885	0	31	93	197	240	216	87	21	0
60	918	0	28	77	215	255	213	104	26	0
61	701	2	20	64	174	174	161	83	23	0
62	883	5	28	94	202	231	201	100	22	0
63	616	3	20	68	142	165	138	67	13	0
64	554	2	18	50	140	159	120	55	10	0
65	564	5	15	44	157	154	126	53	10	0
66	604	3	19	55	154	160	149	52	12	0
67	885	0	31	93	197	240	216	87	21	0
68	918	0	28	77	215	255	213	104	26	0
69	701	2	20	64	174	174	161	83	23	0
70	883	5	28	94	202	231	201	100	22	0
71	616	3	20	68	142	165	138	67	13	0
72	554	2	18	50	140	159	120	55	10	0
73	564	5	15	44	157	154	126	53	10	0
74	604	3	19	55	154	160	149	52	12	0

Bit	Nbr. monômes	Nombre de monômes de degré :								
		0	1	2	3	4	5	6	7	8
75	885	0	31	93	197	240	216	87	21	0
76	918	0	28	77	215	255	213	104	26	0
77	701	2	20	64	174	174	161	83	23	0
78	883	5	28	94	202	231	201	100	22	0
79	616	3	20	68	142	165	138	67	13	0
80	554	2	18	50	140	159	120	55	10	0
81	564	5	15	44	157	154	126	53	10	0
82	604	3	19	55	154	160	149	52	12	0
83	885	0	31	93	197	240	216	87	21	0
84	918	0	28	77	215	255	213	104	26	0
85	701	2	20	64	174	174	161	83	23	0
86	883	5	28	94	202	231	201	100	22	0
87	616	3	20	68	142	165	138	67	13	0
88	554	2	18	50	140	159	120	55	10	0
89	564	5	15	44	157	154	126	53	10	0
90	604	3	19	55	154	160	149	52	12	0
91	885	0	31	93	197	240	216	87	21	0
92	918	0	28	77	215	255	213	104	26	0
93	701	2	20	64	174	174	161	83	23	0
94	883	5	28	94	202	231	201	100	22	0
95	616	3	20	68	142	165	138	67	13	0
96	554	2	18	50	140	159	120	55	10	0
97	564	5	15	44	157	154	126	53	10	0
98	604	3	19	55	154	160	149	52	12	0
99	885	0	31	93	197	240	216	87	21	0
100	918	0	28	77	215	255	213	104	26	0
101	701	2	20	64	174	174	161	83	23	0
102	883	5	28	94	202	231	201	100	22	0
103	616	3	20	68	142	165	138	67	13	0
104	554	2	18	50	140	159	120	55	10	0
105	564	5	15	44	157	154	126	53	10	0
106	604	3	19	55	154	160	149	52	12	0
107	885	0	31	93	197	240	216	87	21	0
108	918	0	28	77	215	255	213	104	26	0
109	701	2	20	64	174	174	161	83	23	0
110	883	5	28	94	202	231	201	100	22	0
111	616	3	20	68	142	165	138	67	13	0
112	554	2	18	50	140	159	120	55	10	0
113	564	5	15	44	157	154	126	53	10	0
114	604	3	19	55	154	160	149	52	12	0
115	885	0	31	93	197	240	216	87	21	0
116	918	0	28	77	215	255	213	104	26	0
117	701	2	20	64	174	174	161	83	23	0
118	883	5	28	94	202	231	201	100	22	0
119	616	3	20	68	142	165	138	67	13	0
120	554	2	18	50	140	159	120	55	10	0
121	564	5	15	44	157	154	126	53	10	0
122	604	3	19	55	154	160	149	52	12	0
123	885	0	31	93	197	240	216	87	21	0
124	918	0	28	77	215	255	213	104	26	0

Bit	Nbr. monômes	Nombre de monômes de degré :								
		0	1	2	3	4	5	6	7	8
125	701	2	20	64	174	174	161	83	23	0
126	883	5	28	94	202	231	201	100	22	0
127	616	3	20	68	142	165	138	67	13	0

Conception, développement et analyse de systèmes de fonction booléennes décrivant les algorithmes de chiffrement et de déchiffrement de l'Advanced Encryption Standard

Résumé :

La cryptologie est une des disciplines des mathématiques composée de deux sous-ensembles : la cryptographie et la cryptanalyse. Tandis que la cryptographie s'intéresse aux algorithmes permettant de modifier une information afin de la rendre inintelligible sans la connaissance d'un secret, la seconde s'intéresse aux méthodes permettant de recouvrer l'information originale à partir de la seule connaissance de l'élément chiffré et ceci, dans un temps plus court que l'attaque par force brute. Il existe de nombreuses méthodes de cryptanalyse mais beaucoup de ces méthodes sont maintenues en échec par les algorithmes de chiffrement modernes. C'est le cas notamment de l'Advanced Encryption Standard (AES). Cet algorithme de chiffrement par blocs a été conçu par Joan Daemen et Vincent Rijmen et transformé en standard en 2001. Afin de contrer les méthodes de cryptanalyse usuelles les concepteurs de l'AES lui ont donné une forte structure algébrique. Ce choix élimine brillamment toute possibilité d'attaque statistique, cependant, de récents travaux tendent à montrer, que ce qui est censé faire la robustesse de l'AES, pourrait se révéler être son point faible. En effet, selon ces études, cryptanalyser l'AES se "résume" à résoudre un système d'équations quadratiques symbolisant la structure du chiffrement de l'AES. Malheureusement, la taille du système d'équations obtenu et le manque d'algorithmes de résolution efficaces font qu'il est impossible, à l'heure actuelle, de résoudre de tels systèmes dans un temps raisonnable. L'enjeu de cette thèse est, à partir de la structure algébrique de l'AES, de décrire son algorithme de chiffrement et de déchiffrement sous la forme d'un nouveau système d'équations booléennes. Puis, en s'appuyant sur une représentation spécifique de ces équations, d'en réaliser une analyse combinatoire afin d'y détecter d'éventuels biais statistiques.

Mots clés : Cryptographie, Cryptanalyse, AES, Combinatoire, Structure algébrique, Fonction booléenne

Design, development and analysis of Boolean function systems describing the encryption and decryption algorithms of the Advanced Encryption Standard

Abstract :

Cryptology is one of the mathematical fields composed of two subsets : cryptography and cryptanalysis. While cryptography focuses on algorithms to modify an information by making it unintelligible without knowledge of a secret, the second focuses on methods to recover the original information from the only knowledge of the encrypted element and thus, into a shorter time than the brute-force attack. There are many methods of cryptanalysis but many of these methods are defeated by modern encryption algorithms. This is the case of the Advanced Encryption Standard (AES). This block cipher algorithm was designed by Joan Daemen and Vincent Rijmen and transformed into standard in 2001. To counter the usual methods of cryptanalysis of AES designers have given it a strong algebraic structure. This choice eliminates brilliantly any possibility of statistical attack, however, recent work suggests that what is supposed to be the strength of the AES, could prove to be his weak point. According to these studies, the AES cryptanalysis comes down to "solve" a quadratic equations symbolizing the structure of the AES encryption. Unfortunately, the size of the system of equations obtained and the lack of efficient resolution algorithms make it impossible, at this time, to solve such systems in a reasonable time. The challenge of this thesis is, from the algebraic structure of the AES, to describe its encryption and decryption processes in the form of a new Boolean equations system. Then, based on a specific representation of these equations, to achieve a combinatorial analysis to detect potential statistical biases.

Keywords : Cryptography, Cryptanalysis, AES, Combinatorics, Algebraic structure, Boolean function

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

ARTS ET MÉTIERS
ParisTech