République Algérienne Démocratique et Populaire
Ministére de l'enseignement supérieur et de la Recherche Scientifique
Université Abderrahmane MIRA de Béjaia
Faculté des Sciences Exactes

Département de Recherche Opérationnelle

Mémoire présenté pour l'obtention du diplôme de Master
en Mathématiques Appliquées

Spécialité : Science des Données et Aide à la Décision

Exploration de la cryptanalyse linéaire et son application sur un réseau SPN

Présenté par :

LATOUZ Myrina
MEZEMATE Amel

Sous la direction de : DJABRI Rabah

Défendu le 29/06/2025, devant le jury composé de :

SOUFIT Massinissa	Président du jury	UAMB - Bejaia
ASLI Larbi	Examinateur	UAMB - Bejaia
CHIBANE Zebida	Examinatrice	UAMB - Bejaia

Année Universitaire 2024-2025

Dédicace

À mon père, pilier de ma vie qui ma élevée avec amour et patience, me poussant à devenir une femme forte.
À ma mère, source de tendresse et de courage,dont les douâas et le soutien m'ont toujours portée.

À ma grand-mère, lumière douce de ma vie , dont les douâas constantes m'ont protégée à chaque étape.

À mes 2 sœurs, mes soutiens fidèles et mes compagnes de coeurs.

À mon fiancé, mon appui , une personneunique dans ma vie, qui ma soutenue dans tous les aspects de ma vie avec patience, générosité et force.

À Myrina , avec qui j'ai partagé cette aventure , entre travail, partages et moments de complicité. Merci pour ta présence et ton bon coeur.

À mes copines , pour leur amitié précieuse et leur disponibilité (je vous aime trop).

À ma famille, pour leur patience et leur compréhension durant les moments d'absence, de doute et de fatigue.

Et bien sur à moi , celle qui a traversé les épreuves, qui n'a jamais abandonné, et qui est aujourd'hui fière de sa réussite.

ce mémoire vous est dédié.

Mezemate Amel

Dédicace

À ma mère, DALILA, la femme la plus incroyable et la plus forte que je connaisse.Pour ses prières quotidiennes,pour sa confiance et sa
foi en moi,pour ne jamais se plaindre chaque fois que j'ai besoin de quelque chose,que ce soit un soutien financier ou moral,
pour ses conseils formidables qui fonctionnent toujours,
pour m'avoir appris tout ce qu'elle sait de la vie d'une manière simple, alors qu'elle les a appris de la manière difficile.

À mon père WAQI, qui m'a appris que je peux tout faire par moi-même,qui souffre toujours avec moi,qui court toujours m'apporter
ce dont j'ai besoin sans poser de questions,
qui est toujours à mes côtés pour me réconforter,
et qui essaie toujours, même quand c'est difficile, de m'offrir tout ce qui me manque.

À mon frère TARIK et à sa femme SIHAM,ce n'est jamais un
non quand je leur demande quelque chose dont j'ai besoin dans ma vie, peu importe ce que c'est.
Ils me demandent toujours si j'ai besoin de quelque chose et m'offrent des choses sans attendre,en me donnant toujours des conseils
pour m'aider dans ma vie et ma carrière.

À mon frère du milieu SALEH EDDINE,
qui m'apporte toujours ce qu'il faut sans que je demande, sans hésitation,
et ce sont toujours les bonnes choses.

À mon petit frère RAOUF,
pour essayer de m'aider, même si ce sont de petites choses — elles comptent.

À ma grand-mère, pour ses prières.Et â mon oncle Rachid, toujours présent au bon moment, offrant ses meilleurs conseils,et ne disant jamais non lorsque j'ai besoin de quelque chose.

À ma meilleure amie Amel, avec qui j'ai vécu l'une des plus belles expériences de cette aventure.
Pour son soutien, sa gentillesse et la pureté de son cœur.

Et à moi-même, pour avoir affronté chaque épreuve et réussi malgré tout.

À toutes les personnes qui, de près ou de loin, ont cru en nous :
ce mémoire vous est dédié.

Latouz Myrina

REMERCIEMENTS

À Dieu, pour la force, la patience et les opportunités qu'Il nous a accordées tout au long de ce travail.

À nos parents, pour leur amour inconditionnel, leur patience et leur soutien indéfectible, qui nous ont donné la force d'avancer, de persévérer et de croire en nous à chaque étape de ce parcours.

À notre encadrant de mémoire, **M. DJABRI Rabah**, pour son accompagnement attentif, sa rigueur scientifique et ses conseils précieux tout au long de ce travail.

À tous les étudiants de deuxième année du Master Sciences des Données et Aide à la Décision (promotion 2019–2020), pour les échanges enrichissants, l'entraide et l'ambiance conviviale qui ont accompagné ce parcours.

À l'équipe pédagogique de l'Université Abderrahmane Mira de Béjaïa, pour leur disponibilité et leur expertise tout au long de notre parcours.

« La gratitude est la mémoire du cœur »
- Jean-Baptiste Massieu

Table des matières

Remerciements . . . 3
Liste des figures . . . 7
Liste des algorithmes . . . 8
Liste des tables . . . 9
Liste des abriviations . . . 10

Introduction générale **11**

1 Fondements de la cryptographie **13**
1.1 Introduction . . . 13
1.2 Histoire de la cryptographie . . . 13
1.2.1 L'Antiquité : premières techniques de chiffrement . . . 13
1.2.2 Le Moyen Âge : cryptographie et affaires politiques . . . 13
1.2.3 La renaissance : vers des méthodes plus sophistiquées . . . 13
1.2.4 Le XX^{e} siècle : l'ère des machines . . . 14
1.2.5 La cryptographie asymétrique et l'ère numérique . . . 14
1.2.6 L'avenir de la cryptographie : l'ère quantique . . . 14
1.3 Concepts essentiels en cryptographie . . . 14
1.4 Notions mathématiques de base . . . 16
1.4.1 Le modulo . . . 16
1.4.2 Les ensembles $\mathbb{Z}$ et $\mathbb{Z}_n$ . . . 17
1.4.3 L'inverse modulaire . . . 17
1.4.4 Le PGCD (plus grand commun diviseur) . . . 17
1.4.5 Les nombres premiers . . . 17
1.4.6 L'opération XOR . . . 17
1.5 Les cryptosystèmes . . . 18
1.5.1 Définition . . . 18
1.5.2 Objectif et caractéristique de la cryptographie . . . 19
1.5.2.1 Objectif principal . . . 19
1.5.2.2 caractéristique de la cryptographie . . . 19
1.5.3 Cryptosystèmes symétriques . . . 19
1.5.3.1 Cryptosystèmes symétriques classiques . . . 19
1.5.3.2 Cryptosystèmes symétriques modernes . . . 22
1.5.4 Cryptosystèmes asymétriques . . . 23
1.6 Méthodes de cryptanalyse classiques . . . 27
1.6.1 Attaque exhaustive . . . 28
1.6.1.1 Principe : . . . 28
1.6.1.2 Complexité et faisabilité de l'attaque par force brute : . . . 28
1.6.1.3 Contre-mesures contre l'attaque exhaustive : . . . 28
1.6.2 Analyse de fréquence . . . 29
1.6.2.1 Principe de l'analyse de fréquence . . . 29
1.6.2.2 Limites de l'analyse de fréquence : . . . 30
1.6.2.3 Outils utilisés pour l'analyse de fréquence : . . . 30
1.7 Méthodes de cryptanalyse moderne . . . 32

1.7.1 Cryptanalyse Linéaire . . . 32
1.7.2 Cryptanalyse différentielle . . . 32
1.7.3 Comparaison avec la cryptanalyse différentielle . . . 32
1.8 Conclusion . . . 33

2 Cryptanalyse linéaire 34
2.1 Introduction . . . 34
2.2 Histoire . . . 34
2.3 Principe de la cryptanalyse linéaire . . . 35
2.3.1 Concept de base . . . 35
2.3.2 Rôle des sous-clés . . . 35
2.3.3 Algorithmes de Matsui . . . 35
2.3.3.1 Algorithme 1 de Matsui . . . 35
2.3.3.2 Algorithme 2 de Matsui . . . 36
2.4 Présentation du chiffrement DES . . . 37
2.4.1 Définition . . . 37
2.4.2 Limites de la cryptanalyse linéaire sur DES . . . 37
2.5 Substitutions et permutations . . . 37
2.5.1 S-boîte (boîte de substitution) . . . 38
2.5.2 P-boîte (boîte de permutation) . . . 38
2.6 Le réseau de substitution-permutation (SPN) . . . 39
2.6.1 Définition . . . 39
2.6.2 Algorithme de chiffrement d'un réseau SPN . . . 40
2.7 Analyse d'un réseau SPN . . . 41
2.7.1 Paramètres initiaux d'un SPN . . . 41
2.7.2 Structure générale du chiffrement . . . 41
2.7.3 Étapes de transformation d'un tour . . . 42
2.7.4 Tour final et chiffrement . . . 42
2.7.5 Rôle cryptographique des composantes . . . 42
2.7.6 Avantage pour la cryptanalyse linéaire . . . 43
2.8 Étapes d'application de la cryptanalyse linéaire sur un réseau SPN . . . 43
2.8.1 Définition des composants fondamentaux — S-boîte et P-boîte . . . 43
2.8.2 Construction de la table d' approximation linéaire (LAT) . . . 44
2.8.2.1 Algorithme de construction de la table LAT . . . 44
2.8.2.2 Tableau LAT obtenu . . . 45
2.8.3 Calcul du biais et de la probabilité . . . 45
2.8.4 Approximation linéaire finale et lemme d'empilement . . . 46
2.8.4.1 Approximation finale et chemin linéaire . . . 46
2.8.5 Lemme d'empilement . . . 47
2.8.6 Calcul du nombre de paires clair-chiffré . . . 47
2.8.7 Test et estimation de la clé finale . . . 48
2.8.8 Algorithme d'éstimation de la clé finale . . . 49
2.9 Conclusion . . . 50

3 Implémentation et résultats 51
3.1 Introduction . . . 51
3.2 Implémentation . . . 51
3.2.1 Présentation générale de l'implémentation . . . 51
3.2.2 Outils utilisés . . . 51
3.2.3 Objectif et description du SPN utilisé . . . 52
3.3 Implementation de la méthodologie . . . 53
3.3.1 Construction de la table d'approximation linéaire (LAT) . . . 53
3.3.2 Calcul des biais $\varepsilon(\alpha, \beta)$ . . . 54
3.3.3 Approximations linéaires . . . 55

3.3.4 Application du lemme d'empilement 55
3.3.5 Génération des paires 56
3.3.6 Estimation de la clé 56
3.4 Résultats des attaques 59
3.5 Comparaison des résultats 60
3.5.1 Analyse de la précision 61
3.5.2 Convergence vers la clé réelle 61
3.6 Interprétation et discussion 61
3.7 Conclusion 61

Conclusion générale **62**

Annexes **63**

Bibliographie **63**

Table des figures

1.1 Premières formes de cryptographie dans l'Antiquité [3] 13
1.2 Illustration du chiffrement et du déchiffrement [3] 15
1.3 Exemple de chiffrement et de déchiffrement [3] 15
1.4 Clé publique utilisée pour chiffrer [3] 16
1.5 Clé privée utilisée pour déchiffrer [3] 16
1.6 Principe général d'un cryptosystème [1] 19
1.7 Principe de fonctionnement de l'AES [1] 23
1.8 Vérification de l'intégrité [1] 26
1.9 Fréquence d'apparition des lettres en français 31

2.1 S-boîte de 4 bits. 38
2.2 Exemple d'une P-boîte de 8 bits. 39
2.3 Un SPN de 16 bits(4 tours) 40

3.1 structure de SPN de 2 tours 56
3.2 Schéma de l'attaque linéaire appliquée au SPN implémenté 58

Liste des algorithmes

1 Procédure de cryptanalyse linéaire (estimation du membre droit) 36
2 Recherche du biais maximal d'une approximation linéaire sans clé 36
3 Chiffrement d'un réseau SPN (adapté de [25]) 41
4 Construction de la LAT pour une S-boîte de 4 bits 44
5 Estimation de la sous-clé finale par cryptanalyse linéaire 49

Liste des tables

1.1 Chiffrement de “CRYPTOGRAPHY” avec le chiffre de César 20
1.2 Exemple de chiffre affine 20
1.3 Chiffrement de “HELLO” avec le mot-clé “KEY” 21
1.4 Comparaison entre la cryptanalyse différentielle et la cryptanalyse linéaire 33

2.1 Table de substitution (S-boîte) 38
2.2 Table de permutation (P-boîte) 38
2.3 Exemple de S-boîte pour un bloc de 12 bits 43
2.4 Exemple de P-boîte pour un bloc de 12 bits 43
2.5 Table d’approximation linéaire 45
2.6 Table des biais $\varepsilon(\alpha, \beta)$ 46

3.1 Table d’approximation linéaire (LAT) 53
3.2 Table des biais 54
3.3 Meilleures approximations linéaires 55
3.4 Estimation de la clé avec $N = 28$ 59
3.5 Estimation de la clé avec $N = 114$ 60
3.6 Comparaison des résultats 60
7 Valeurs associées aux lettres de l’alphabet dans $\mathbb{Z}_{26}$ 63

Liste des abréviations

Abréviation	**Signification**
SPN	Réseau de Substitution-Permutation (Substitution-Permutation Network)
LAT	Table d'approximation linéaire (Linear Approximation Table)
DES	Standard de chiffrement des données (Data Encryption Standard)
AES	Standard de chiffrement avancé (Advanced Encryption Standard)
XOR	Opération OU exclusif
GPU	Carte graphique (Graphics Processing Unit)

Introduction générale

La cryptographie, science millénaire dédiée à la protection des communications, occupe aujourd'hui une place centrale dans un monde numérique où les données sont au cœur des échanges. Face à l'explosion des technologies et à l'intensification des cyberattaques, garantir la confidentialité, l'intégrité et l'authenticité des informations est devenu un enjeu crucial. Pourtant, aucun système de chiffrement n'est totalement infaillible : sa robustesse dépend de sa capacité à résister aux tentatives de décryptage non autorisées. C'est dans cette optique que s'inscrit la cryptanalyse, discipline qui étudie et évalue la sécurité des algorithmes de chiffrement en tentant de les contourner ou de les casser.

Parmi les méthodes de cryptanalyse, la cryptanalyse linéaire introduite par Mitsuru Matsui en 1993 [15] se distingue par son approche statistique. Elle s'appuie sur l'identification de biais dans les relations linéaires entre les bits du texte clair, du texte chiffré et de la clé, afin d'estimer cette dernière. Cette méthode s'applique particulièrement bien aux chiffrements par blocs, notamment aux réseaux de substitution-permutation (SPN), structures de chiffrement largement utilisées dans les algorithmes modernes. Le présent mémoire s'intéresse à un problème fondamental : Comprendre comment une attaque peut exploiter des biais statistiques, quel en est le principe, la méthode utilisée, et les conditions nécessaires à sa réussite est essentiel, non seulement pour évaluer la sécurité des algorithmes existants, mais aussi pour concevoir des systèmes cryptographiques plus résistants.

Ce travail, réalisé dans le cadre du Master en science des données et aide à la décision, propose une étude à la fois théorique et expérimentale de la cryptanalyse linéaire appliquée à un réseau de chiffrement que nous avons entièrement conçu et développé . Plus précisément, nous avons construit un réseau SPN , dont tous les paramètres ont été définis de manière personnalisée : nous avons fixé le nombre de tours de chiffrement (deux tours), choisi manuellement trois boîtes de substitution (S-boîtes) identiques, défini une boîte de permutation (P-boîte) assurant la diffusion, établi le format des données en entrée et en sortie, et conçu un schéma de génération des sous-clés à partir d'une clé principale. L'ensemble du processus de chiffrement — de l'application des S-boîtes et de la permutation, jusqu'à l'ajout des sous-clés — a été défini et maîtrisé dans le détail. Ce réseau, spécifiquement construit dans un cadre expérimental contrôlé, constitue la base sur laquelle nous avons appliqué et analysé une attaque par cryptanalyse linéaire.

Concrètement, ce mémoire s'organise autour de plusieurs étapes : dans un premier temps, nous étudierons les concepts de base de la cryptanalyse linéaire, notamment les approximations linéaires, les boîtes de substitution (S-boîtes), les boîtes de permutation (P-boîtes), et le lemme d'empilement, qui permet de combiner les biais issus de plusieurs approximations. Dans un second temps, nous mettrons en œuvre une attaque linéaire sur notre réseau SPN à l'aide du langage Python. Cette attaque reposera sur la construction d'une table d'approximation linéaire (LAT), la sélection de la meilleure approximation, et l'analyse statistique de nombreuses paires clair-chiffré pour tenter de retrouver une sous-clé finale. Enfin, nous comparerons deux formules couramment utilisées pour estimer le nombre minimal de paires nécessaires à la réussite de l'attaque : $N_1 = \frac{4}{\varepsilon_t^2}$ et $N_2 = \frac{16}{\varepsilon_t^2}$. Cette comparaison permettra d'évaluer leur impact sur la précision et la fiabilité de l'estimation de la clé.

Le mémoire est structuré en trois chapitres. Le premier chapitre présente les fondements de la cryptographie, en exposant son évolution historique, ses concepts essentiels, les différents types de

cryptosystèmes ainsi que les principales méthodes de cryptanalyse classiques et modernes.

Le deuxième chapitre est dédié à la cryptanalyse linéaire : il en détaille les principes, les notions théoriques clés comme les approximations linéaires, le biais, les S-boîtes, les P-boîtes, le lemme d'empilement, et il montre comment cette méthode s'applique à des chiffrements réels tels que le DES, puis à un réseau SPN.

On clôture notre travail par le troisième chapitre qui est consacré à l'implémentation pratique de l'attaque sur le réseau SPN que nous avons construit. Ce réseau a été défini avec deux tours, trois S-boîtes identiques, une P-boîte spécifique, 3 clés initialisées aléatoirement et un schéma de l'attaque appliquer au spn implémenté. Nous avons mis en œuvre toutes les étapes nécessaires à l'attaque : génération de la LAT, sélection d'approximations, calcul du biais global, estimation du nombre de paires, génération automatique des paires et test des sous-clés. Les résultats obtenus ont ensuite été analysés et comparés selon deux formules théoriques de calcul du nombre de paires nécessaires.

En combinant la théorie et la pratique sur un réseau SPN que nous avons entièrement construit, ce travail permet de mieux comprendre les faiblesses possibles des systèmes de chiffrement et montre pourquoi il est important de vérifier régulièrement leur sécurité face aux attaques.

Chapitre 1

Fondements de la cryptographie

1.1 Introduction

Dans ce chapitre, nous présentons les fondements théoriques de la cryptographie. Nous allons d'abord définir les concepts de base, puis décrire les principaux types de cryptosystèmes et les fonctions essentielles qui assurent la sécurité des échanges d'informations.

1.2 Histoire de la cryptographie

La cryptographie est une discipline ancienne qui remonte à l'Antiquité et qui n'a cessé d'évoluer au fil des siècles. D'abord utilisée pour des besoins militaires et politiques, elle s'est transformée au gré des avancées technologiques jusqu'à devenir un pilier essentiel de la sécurité numérique moderne.

1.2.1 L'Antiquité : premières techniques de chiffrement

Les premières formes de cryptographie remontent à l'Antiquité. Les Égyptiens utilisaient déjà des hiéroglyphes à des fins de dissimulation. Chez les Grecs et les Romains, des méthodes plus élaborées ont vu le jour. L'exemple le plus emblématique est le **chiffre de César**, qui consiste à décaler chaque lettre du message d'un certain nombre de positions dans l'alphabet pour masquer son contenu (voir[10]).

FIGURE 1.1 – Premières formes de cryptographie dans l'Antiquité [3]

1.2.2 Le Moyen Âge : cryptographie et affaires politiques

Au Moyen Âge, la cryptographie joue un rôle essentiel dans les affaires diplomatiques et militaires. L'une des affaires les plus célèbres est celle de **Marie Stuart**, qui communiquait secrètement par messages chiffrés dans une tentative de renverser la reine Élisabeth I. Ces messages furent interceptés et décryptés, conduisant à son arrestation et à son exécution.

1.2.3 La renaissance : vers des méthodes plus sophistiquées

Durant la Renaissance, les méthodes de chiffrement deviennent plus complexes. Le mathématicien français **Blaise de Vigenère** met au point le **chiffre de Vigenère**, un système de chiffrement poly-

alphabétique qui utilise plusieurs alphabets pour rendre le déchiffrement plus difficile. Cette méthode résiste longtemps à la cryptanalyse jusqu'à ce que **Charles Babbage** et **Friedrich Kasiski** proposent au XIX^{e} siècle des techniques efficaces pour la casser.

1.2.4 Le XX^{e} siècle : l'ère des machines

Le XX^{e} siècle marque une révolution avec l'apparition des machines électromécaniques. La plus célèbre d'entre elles est la machine **Enigma**, utilisée par l'armée allemande pendant la Seconde Guerre mondiale. Grâce au travail de **Alan Turing** et de son équipe à Bletchley Park, les codes d'Enigma sont brisés, jouant un rôle décisif dans l'issue de la guerre [21].

Avec l'essor de l'informatique, la cryptographie entre dans l'ère numérique. **Claude Shannon**, souvent considéré comme le père de la cryptographie moderne, jette les bases théoriques de la sécurité de l'information, introduisant des concepts fondamentaux tels que la confusion et la diffusion. Ces travaux influencent la conception d'algorithmes modernes comme le **Data Encryption Standard (DES)** et plus tard l'**Advanced Encryption Standard (AES)**.

1.2.5 La cryptographie asymétrique et l'ère numérique

Dans les années 1970, un changement majeur survient avec l'apparition de la **cryptographie asymétrique**, notamment grâce à l'algorithme **RSA**, développé par **Rivest**, **Shamir** et **Adleman**. Ce système repose sur une paire de clés — l'une publique pour chiffrer, l'autre privée pour déchiffrer — et rend possible la sécurité des communications sur des réseaux ouverts comme Internet.

1.2.6 L'avenir de la cryptographie : l'ère quantique

Avec les progrès de l'**informatique quantique**, certains algorithmes classiques, notamment RSA et ECC, pourraient devenir vulnérables à des attaques menées par des ordinateurs quantiques. Cela stimule actuellement le développement de nouveaux **algorithmes post-quantiques**, conçus pour résister aux capacités de calcul de ces futures machines, ouvrant ainsi une nouvelle ère pour la cryptographie.

1.3 Concepts essentiels en cryptographie

La cryptographie repose sur un ensemble de concepts fondamentaux qu'il est important de maîtriser pour bien comprendre cette discipline. Cette section présente les principales notions et terminologies utilisées.

- **Cryptographie** : Science qui utilise les mathématiques pour chiffrer et déchiffrer des données. Elle permet de protéger les informations sensibles, en garantissant qu'elles ne puissent être lues que par le destinataire prévu, même lorsqu'elles sont transmises sur des réseaux non sécurisés tels qu'Internet [22].

- **Cryptanalyse** : Ensemble de techniques visant à retrouver un message clair à partir de son texte chiffré sans connaître la clé utilisée. Elle consiste à analyser et attaquer un système cryptographique afin d'en tester ou compromettre la sécurité [22].

- **Cryptologie** : Science englobant à la fois la cryptographie et la cryptanalyse. Elle étudie les méthodes de sécurisation des informations et les techniques pour en évaluer la robustesse [22].

- **Texte en clair (plaintext)** : Message original, non chiffré, lisible sans aucune transformation .

- **Texte chiffré (ciphertext)** : Message transformé par un algorithme de chiffrement. Il est illisible sans le processus inverse, appelé déchiffrement .

- **Chiffre (cipher)** : Algorithme utilisé pour effectuer le chiffrement ou le déchiffrement. Il s'agit d'une suite d'étapes mathématiques bien définies appliquées au message [22].

- **Chiffrement (encryption)** : Processus qui transforme un texte en clair en un texte chiffré afin de le protéger contre toute consultation non autorisée [22].

- **Déchiffrement (decryption)** : Processus inverse du chiffrement. Il permet de retrouver le texte en clair à partir du texte chiffré [22].

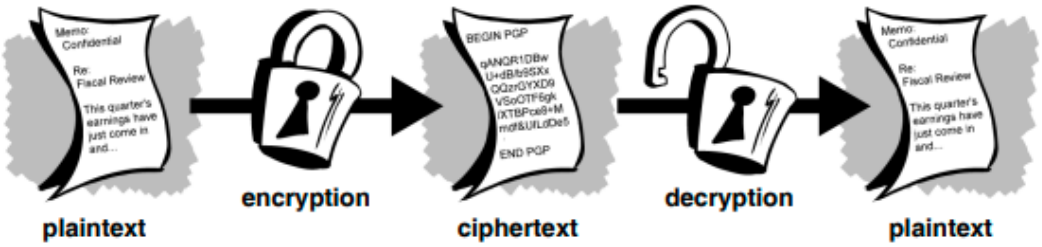

FIGURE 1.2 – Illustration du chiffrement et du déchiffrement [3]

- **Clé de chiffrement** : Information secrète utilisée dans le processus de chiffrement pour sécuriser un message .

- **Clé de déchiffrement** : Information secrète permettant de retrouver le texte en clair à partir du texte chiffré .

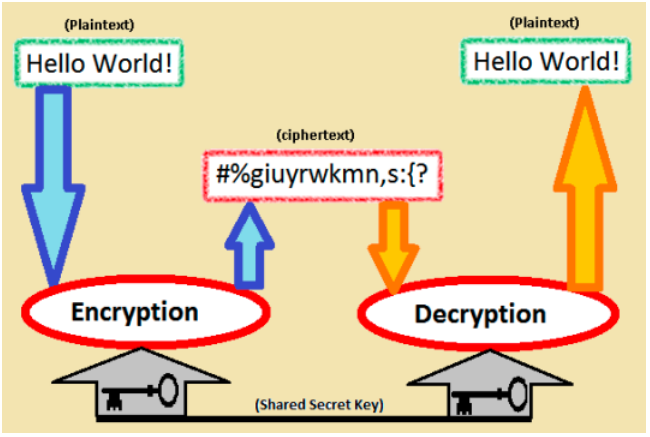

FIGURE 1.3 – Exemple de chiffrement et de déchiffrement [3]

- **Algorithme de chiffrement** : Méthode mathématique qui permet de transformer un texte en clair en un texte chiffré .

- **Algorithme de déchiffrement** : Méthode permettant de retrouver le texte en clair à partir du texte chiffré .

- **Chiffrement symétrique** : Méthode dans laquelle la même clé est utilisée à la fois pour le chiffrement et pour le déchiffrement .

- **Chiffrement asymétrique** : Méthode utilisant deux clés distinctes : une clé publique pour chiffrer et une clé privée pour déchiffrer .

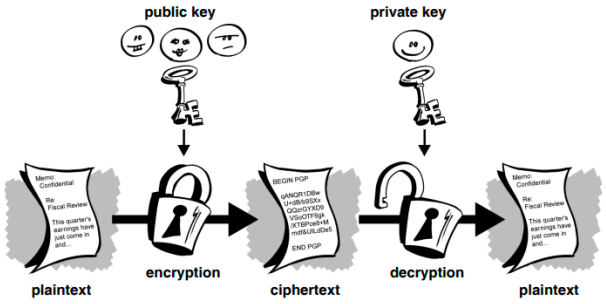

FIGURE 1.4 – Clé publique utilisée pour chiffrer [3]

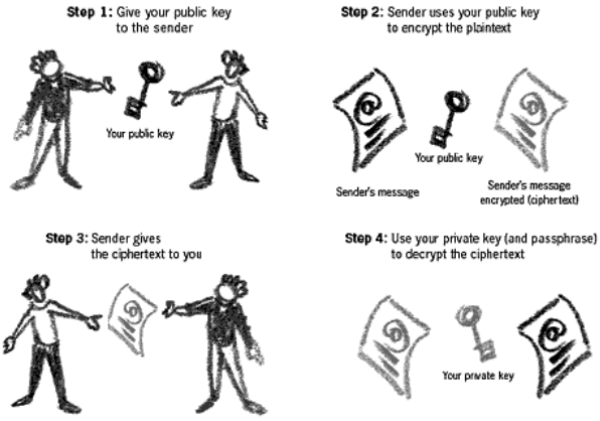

FIGURE 1.5 – Clé privée utilisée pour déchiffrer [3]

- **Chiffrement par substitution** : Technique dans laquelle chaque symbole du texte en clair est remplacé par un autre selon une règle fixe [22].

  On distingue principalement :

  - **Substitution mono-alphabétique** : Chaque lettre du texte clair est toujours remplacée par la même lettre chiffrée (ex. : chiffrement de César) [22].

  - **Substitution poly-alphabétique** : Une même lettre du texte clair peut être remplacée par différentes lettres chiffrées selon la position et une clé (ex. : chiffrement de Vigenère) [22].

- **Chiffrement par permutation** : Méthode où les éléments du message sont réarrangés selon un ordre déterminé, sans modifier leur valeur intrinsèque [22].

- **Chiffrement par blocs** : Le message est découpé en blocs de taille fixe (par exemple, 64 bits pour **DES**, 128 bits pour **AES**), et chaque bloc est chiffré séparément à l'aide d'une clé [22].

- **Réseau de Feistel** : Structure utilisée dans plusieurs algorithmes de chiffrement (comme **DES**) qui divise les données en deux moitiés et applique des opérations successives avec une clé, de manière itérative [22].

1.4 Notions mathématiques de base

Avant d'étudier les systèmes de cryptographie, il est important de comprendre quelques notions mathématiques simples mais essentielles. Ces concepts sont largement utilisés dans les algorithmes de chiffrement et de déchiffrement.

1.4.1 Le modulo

L'opération **modulo** (notée `mod`) donne le **reste** de la division entière d'un nombre par un autre.

- Exemple : $7 \mod 3 = 1$, car $7 = 2 \times 3 + 1$

En cryptographie, on travaille souvent **modulo 26** pour représenter les lettres de l'alphabet (A = 0, B = 1, ..., Z = 25).

1.4.2 Les ensembles $\mathbb{Z}$ et $\mathbb{Z}_n$

- $\mathbb{Z}$ représente l'ensemble des **entiers relatifs**, soit : $\{..., -2, -1, 0, 1, 2, ...\}$.
- $\mathbb{Z}_n$ désigne l'ensemble des entiers **modulo** n : $\{0, 1, 2, ..., n-1\}$.

Dans $\mathbb{Z}_n$, toutes les opérations (addition, soustraction, multiplication) se font modulo n.

- Exemple : dans $\mathbb{Z}_5$, on a $3 + 4 = 7 \equiv 2 \mod 5$

1.4.3 L'inverse modulaire

L'**inverse modulaire** d'un nombre a (modulo n) est un entier x tel que :

$$a \times x \equiv 1 \mod n$$

- Exemple : l'inverse de 3 modulo 7 est 5, car $3 \times 5 = 15 \equiv 1 \mod 7$

Cette notion est très utilisée dans les algorithmes de chiffrement comme RSA.

1.4.4 Le PGCD (plus grand commun diviseur)

Le **PGCD** de deux entiers est le plus grand entier qui divise simultanément les deux.

- Exemple : $\text{PGCD}(12, 8) = 4$

Deux nombres dont le PGCD est égal à 1 sont dits **premiers entre eux**. Cette propriété est fondamentale pour la génération des clés dans certains systèmes cryptographiques.

1.4.5 Les nombres premiers

Un **nombre premier** est un entier naturel strictement supérieur à 1, qui n'a que deux diviseurs : 1 et lui-même.

- Exemples : 2, 3, 5, 7, 11, 13, ...

1.4.6 L'opération XOR

L'opération **XOR** (ou "OU exclusif") est une opération logique fondamentale en cryptographie. Elle est notée $\oplus$ (ou parfois ^ en langage Python).

Règle de base

L'opération XOR compare deux bits. Le résultat est donné par le tableau suivant :

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

- Si les deux bits sont égaux, le résultat est 0.
- Si les deux bits sont différents, le résultat est 1.

Propriétés importantes

- $A \oplus 0 = A$ (XOR avec 0 ne change rien)
- $A \oplus 1 = \overline{A}$ (XOR avec 1 inverse le bit)
- $A \oplus A = 0$ (XOR d'un bit avec lui-même donne toujours 0)
- $(A \oplus B) \oplus C = A \oplus (B \oplus C)$ (Associativité)

Exemple simple

Considérons deux blocs de bits :

Exemple d'utilisation du XOR

Message : 1010
Clé : 1100
Résultat (XOR) : 0110

Chaque bit du message est combiné avec le bit correspondant de la clé.

Utilité en cryptographie

L'opération XOR est très utilisée car :

- Elle est simple et rapide à implémenter.
- Elle est réversible : si $C = P \oplus K$, alors $P = C \oplus K$.
- Elle est utilisée dans de nombreux algorithmes comme DES pour combiner les blocs de texte avec les sous-clés.

1.5 Les cryptosystèmes

1.5.1 Définition

Un **cryptosystème** est un ensemble structuré de techniques cryptographiques combinées à une infrastructure permettant d'assurer la sécurité de l'information. Il est également appelé *système de chiffrement*. Un cryptosystème permet de transformer un message compréhensible (texte en clair) en un message inintelligible (texte chiffré), et inversement, à l'aide de clés et d'algorithmes bien définis. [1]

Les composants essentiels d'un cryptosystème sont les suivants :

- **Texte en clair** : le message d'origine, lisible par tout le monde.
- **Algorithme de chiffrement** : procédé mathématique transformant le texte en clair en texte chiffré.
- **Texte chiffré** : le message illisible obtenu après chiffrement.
- **Algorithme de déchiffrement** : procédé mathématique permettant de retrouver le texte en clair à partir du texte chiffré.
- **Clé de chiffrement** : donnée secrète utilisée pour chiffrer le message.
- **Clé de déchiffrement** : donnée utilisée pour déchiffrer le message (identique ou différente selon le type de cryptosystème).

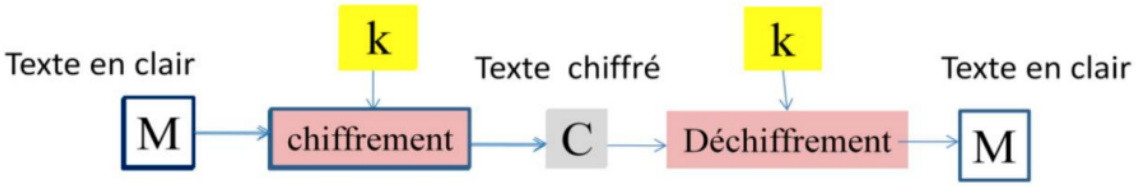

FIGURE 1.6 – Principe général d'un cryptosystème [1]

1.5.2 Objectif et caractéristique de la cryptographie

1.5.2.1 Objectif principal

L'objectif fondamental de la cryptographie est de garantir la sécurité des données sensibles, qu'elles soient stockées localement (disque dur, base de données) ou transmises via des canaux non sécurisés comme Internet ou un réseau sans fil.

1.5.2.2 caractéristique de la cryptographie

La cryptographie vise à assurer plusieurs caractéristiques essentielles à la sécurité de l'information :

Confidentialité

- Assurer que seules les entités autorisées peuvent accéder aux données.
- Même si un message est intercepté, il reste incompréhensible sans la clé appropriée.

Intégrité

- Garantir que les données n'ont pas été modifiées (intentionnellement ou accidentellement) pendant leur transmission ou leur stockage.

Authentification

- Vérifier l'identité de l'expéditeur ou du destinataire d'un message.
- Permet d'établir une confiance dans les échanges d'information.

Non-répudiation

- Empêcher qu'un utilisateur ayant envoyé un message puisse nier l'avoir fait.
- Ce service est particulièrement important dans le cadre de contrats électroniques ou de transactions numériques.

1.5.3 Cryptosystèmes symétriques

Les cryptosystèmes symétriques reposent sur l'utilisation d'une même clé pour le chiffrement et le déchiffrement des données. Cela signifie que l'expéditeur et le destinataire doivent partager cette clé de manière sécurisée. On distingue généralement deux grandes catégories : les cryptosystèmes classiques et les cryptosystèmes modernes.

1.5.3.1 Cryptosystèmes symétriques classiques

Développées avant l'avènement de l'informatique moderne, ces méthodes utilisent principalement des techniques de substitution et de permutation. Voici quelques exemples emblématiques :

- **Le chiffre de César** : il s'agit d'un chiffrement par décalage, où chaque lettre du message est remplacée par une autre située k positions plus loin dans l'alphabet.
  Pour cela, on associe chaque lettre de l'alphabet à un entier de l'ensemble $\mathbb{Z}_d$ (voir Annexe 3.7).
  La fonction de chiffrement $e_k : \mathbb{Z}_d \to \mathbb{Z}_d$ est donnée par :

$$e_k(m) = m + k$$

  La fonction de déchiffrement $d_k : \mathbb{Z}_d \to \mathbb{Z}_d$ associée est :

$$d_k(c) = c - k$$

  Exemple : avec $k = 3$, "CRYPTOGRAPHY" devient "FUBSWRJUDSKB".

Lettre	**Valeur (A=0)**	**Formule**	**Lettre chiffrée**
C	2	$(2 + 3) \bmod 26 = 5$	F
R	17	$(17 + 3) \bmod 26 = 20$	U
Y	24	$(24 + 3) \bmod 26 = 1$	B
P	15	$(15 + 3) \bmod 26 = 18$	S
T	19	$(19 + 3) \bmod 26 = 22$	W
O	14	$(14 + 3) \bmod 26 = 17$	R
G	6	$(6 + 3) \bmod 26 = 9$	J
R	17	$(17 + 3) \bmod 26 = 20$	U
A	0	$(0 + 3) \bmod 26 = 3$	D
P	15	$(15 + 3) \bmod 26 = 18$	S
H	7	$(7 + 3) \bmod 26 = 10$	K
Y	24	$(24 + 3) \bmod 26 = 1$	B

TABLE 1.1 – Chiffrement de "CRYPTOGRAPHY" avec le chiffre de César

- **Le chiffre affine** : Chaque lettre est transformée par une fonction affine définie sur l'ensemble $\mathbb{Z}_d$. La clé est composée de deux entiers k_1 et k_2 tels que $k_1 \in \mathbb{Z}_d^*$ (c'est-à-dire premier avec 26) et $k_2 \in \mathbb{Z}_d$.
  Dans ce contexte, la fonction de chiffrement $e_k : \mathbb{Z}_d \to \mathbb{Z}_d$ est :

$$e_k(m) = k_1 \cdot m + k_2$$

  La fonction de déchiffrement ${}_k : \mathbb{Z}_d \to \mathbb{Z}_d$ est :

$$d_k(c) = k_1^{-1} \cdot (c - k_2)$$

  où k_1^{-1} est l'inverse multiplicatif de k_1 dans $\mathbb{Z}_d$.
  Exemple : pour $k_1 = 5$, $k_2 = 8$, et en utilisant l'alphabet codé dans $\mathbb{Z}_{26}$ (voir Annexe 3.7) :

Lettre	**Valeur (A=0)**	**Formule**	**Lettre chiffrée**
B	1	$(5 \cdot 1 + 8) \bmod 26 = 13$	N
A	0	$(5 \cdot 0 + 8) \bmod 26 = 8$	I
T	19	$(5 \cdot 19 + 8) \bmod 26 = 21$	V

TABLE 1.2 – Exemple de chiffre affine

  Ainsi, le mot "BAT" devient "NIV".

- **Le chiffre de Vigenère** : il s'agit d'un chiffrement polyalphabétique qui repose sur un mot-clé. Chaque lettre du message est chiffrée avec une lettre différente de la clé, ce qui revient à effectuer

un décalage variable dans l'alphabet. La clé est répétée autant de fois que nécessaire pour couvrir toute la longueur du message.
La fonction de chiffrement $e_k : \mathbb{Z}_d^n \to \mathbb{Z}_d^n$ s'écrit :

$$e_k(m) = m + k$$

La fonction de déchiffrement $d_k : \mathbb{Z}_d^n \to \mathbb{Z}_d^n$ est :

$$d_k(c) = c - k$$

Exemple : avec le mot-clé "KEY" (répété sur 5 lettres) et le message "HELLO", on associe chaque lettre à une valeur de $\mathbb{Z}_d$ (voir Annexe 3.7) :

Lettre du message	m_i	**Lettre de la clé**	k_i	**Lettre chiffrée**
H	7	K	10	R
E	4	E	4	I
L	11	Y	24	J
L	11	K	10	V
O	14	E	4	S

TABLE 1.3 – Chiffrement de "HELLO" avec le mot-clé "KEY"

Ainsi, le message "HELLO" est chiffré en "RIJVS".

- **Le chiffre de Hill** : il s'agit d'un chiffrement polygraphique fondé sur l'algèbre linéaire. Le principe repose sur l'utilisation d'une matrice carrée k, appelée matrice-clé, appliquée par multiplication matricielle à des blocs de lettres converties en vecteurs sur $\mathbb{Z}_d$.

  La fonction de chiffrement $e_k : \mathbb{Z}_d^n \to \mathbb{Z}_d^n$ est définie par :

  $$e_k(m) = k \cdot m$$

  où m est un vecteur colonne contenant les valeurs entières des lettres du message clair (voir Annexe 3.7).

  La fonction de déchiffrement $d_k : \mathbb{Z}_d^n \to \mathbb{Z}_d^n$, s'écrit :

  $$d_k(c) = k^{-1} \cdot c$$

  où k^{-1} est l'inverse de la matrice k .

  Exemple : avec la matrice clé

  $$K = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$$

  et le message "HI", on convertit les lettres en entiers : $H = 7$, $I = 8$. On obtient ainsi le vecteur :

  $$m = \begin{bmatrix} 7 \\ 8 \end{bmatrix}$$

  On applique alors le chiffrement :

  $$k \cdot m = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 45 \\ 54 \end{bmatrix} \mod 26 = \begin{bmatrix} 19 \\ 2 \end{bmatrix}$$

  Ainsi, le message "HI" est chiffré en "TC".

- **Le chiffre de Hill affine** : cette variante du chiffre de Hill combine une multiplication matricielle suivie d'une addition vectorielle, de manière analogue au chiffre affine. Il introduit un second paramètre k_2, un vecteur de décalage, ce qui renforce la complexité du chiffrement.

  Dans $\mathbb{Z}_d$, la fonction de chiffrement $e_k : \mathbb{Z}_d^n \to \mathbb{Z}_d^n$, s'écrit :

  $$e_k(m) = k_1 \cdot m + k_2$$

  où :

  - k_1 est une matrice carrée inversible dans $\mathbb{Z}_d$,
  - k_2 est un vecteur colonne de même dimension que m,

  La fonction de déchiffrement $d_k : \mathbb{Z}_d^n \to \mathbb{Z}_d^n$, associée est :

  $$d_k(c) = k_1^{-1} \cdot (c - k_2)$$

  où k_1^{-1} est l'inverse de la matrice k_1 dans $\mathbb{Z}_d$, et c est le vecteur chiffré.
  Exemple : en reprenant le résultat du chiffrement de "HI" avec la matrice

  $$k_1 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$$

  et le vecteur obtenu $k_1 \cdot m = \begin{bmatrix} 19 \\ 2 \end{bmatrix}$, on ajoute un vecteur de décalage :

  $$k_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

  Le résultat final du chiffrement est alors :

  $$\begin{bmatrix} 19 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 20 \\ 4 \end{bmatrix}$$

  Ainsi, le message "HI" devient "UE" après chiffrement affine.

1.5.3.2 Cryptosystèmes symétriques modernes

Contrairement aux méthodes classiques, les algorithmes modernes exploitent des structures mathématiques complexes pour garantir une sécurité accrue. Les plus connus sont :

- **Le Data Encryption Standard (DES)** : développé par IBM dans les années 1970, puis adopté comme standard par le NIST en 1976, DES chiffre les messages par blocs de 64 bits. Si le message n'est pas multiple de 64, un *padding* est ajouté.

- **L'Advanced Encryption Standard (AES)** : successeur du DES, adopté par le NIST en 2001, AES chiffre des blocs de 128 bits avec des clés de 128, 192 ou 256 bits.

  Caractéristiques principales :

  - Blocs de 128 bits
  - 10, 12 ou 14 tours selon la taille de la clé

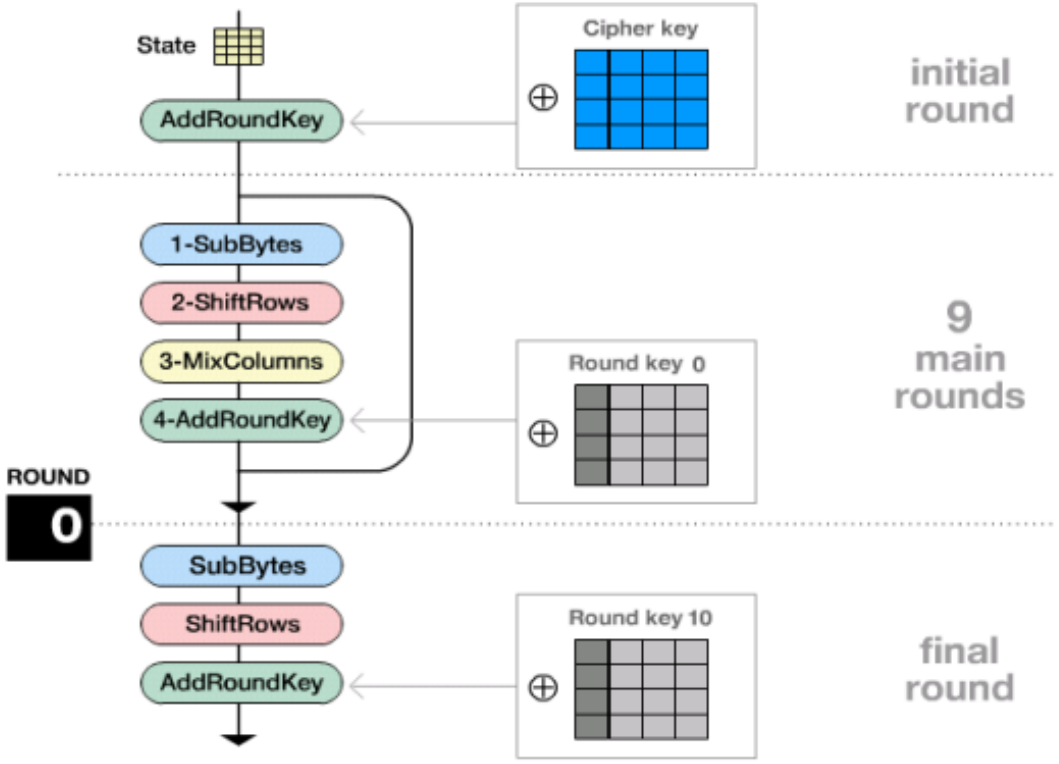

FIGURE 1.7 – Principe de fonctionnement de l'AES [1]

Étapes principales d'un tour AES :

1. **SubBytes** : substitution via une S-boîte
2. **ShiftRows** : permutation des lignes
3. **MixColumns** : transformation linéaire
4. **AddRoundKey** : XOR avec une sous-clé

Exemple : Le message "HELLO", converti en binaire et complété à 128 bits, est soumis aux transformations précédentes pour produire un texte chiffré.

1.5.4 Cryptosystèmes asymétriques

Les cryptosystèmes asymétriques reposent sur une paire de clés : une **clé publique** pour le chiffrement, et une **clé privée** pour le déchiffrement. Contrairement aux systèmes symétriques, la clé de chiffrement peut être librement diffusée, tandis que la clé de déchiffrement reste secrète. Le système le plus célèbre est l'algorithme RSA.

- **RSA (Rivest-Shamir-Adleman)**
  L'algorithme RSA, proposé en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman, repose sur la difficulté de factoriser un grand nombre premier $n = p \times q$, où p et q sont deux grands nombres premiers.

  Principe de fonctionnement :

  - **Génération des clés [6] :**
    1. Choisir deux grands nombres premiers p et q.

2. Calculer $n = p \times q$, appelé le module.

3. Calculer la fonction d'Euler : $\phi(n) = (p-1)(q-1)$.

4. Choisir un entier e tel que $1 < e < \phi(n)$ et $pgcd(e, \phi(n)) = 1$.

5. Calculer l'inverse modulaire d de e modulo $\phi(n)$, tel que :

$$e \times d \equiv 1[\phi(n)]$$

6. La **clé publique** est (n, e), et la **clé privée** est (n, d).

- **Chiffrement :** Pour un message m tel que $m < n$, le chiffrement est donné par :

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

- **Déchiffrement :** Le message est retrouvé avec la clé privée :

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

Exemple :

1. Choisissons deux nombres premiers : $p = 7$ et $q = 13$.
2. Calculons le produit $n = p \times q$:

$$n = 7 \times 13 = 91$$

3. Calculons la fonction d'Euler $\phi(n)$:

$$\phi(n) = (p-1)(q-1) = (7-1)(13-1) = 6 \times 12 = 72$$

4. Choisissons un exposant $e = 5$, qui est premier avec $\phi(n) = 72$.
   Alors la clé publique est :
$$(e, n) = (5, 91)$$
5. Calculons d, l'inverse modulaire de e modulo $\phi(n)$. On obtient : $d = 29$.
   Alors la clé privée est :
$$(d, n) = (29, 91)$$
6. Vérifions que $e \cdot d \equiv 1[\phi(n)]$:

$$5 \times 29 = 145 \quad \Rightarrow \quad 145 \mod 72 = 1$$

Chiffrement du mot SECRET : En convertissant chaque lettre en nombre :

$$S = 19, \quad E = 5, \quad C = 3, \quad R = 18, \quad E = 5, \quad T = 20.$$

Application de la formule de chiffrement $c_i = m_i^e \mod n$:

$$\begin{array}{lll} c_1 = 19^5 \mod 91 = 80 & , c_2 = 5^5 \mod 91 = 31 & , c_3 = 3^5 \mod 91 = 61 \\ , c_4 = 18^5 \mod 91 = 44 & , c_5 = 5^5 \mod 91 = 31 & , c_6 = 20^5 \mod 91 = 76 \end{array}$$

Le message chiffré est : $C = 803161443176$

:

$$S = 19, \quad E = 5, \quad C = 3, \quad R = 18, \quad E = 5, \quad T = 20.$$

Déchiffrement : On applique $m_i = c_i^d \mod n$:

$$\begin{array}{llllll} m_1 = 80^{29} & \mod 91 = 19, & m_2 = 31^{29} & \mod 91 = 5, & m_3 = 61^{29} & \mod 91 = 3, \\ m_4 = 44^{29} & \mod 91 = 18, & m_5 = 31^{29} & \mod 91 = 5 & m_6 = 76^{29} & \mod 91 = 20 \end{array}$$

On retrouve le message : **SECRET**

- **Fonction de Hachage**
  Une fonction de hachage h est une fonction qui prend un message de n'importe quelle taille et le transforme en un message plus court et de longueur fixe. Le résultat obtenu est appelé "haché" ou "condensé". Ce condensé sert comme une empreinte digitale du message original, permettant de l'identifier de façon unique.

  Principe :
  - Elle calcule l'empreinte y d'un message x. Cette fonction doit être à sens unique.
  - Elle doit être très sensible, de sorte qu'une petite modification du message entraîne une grande modification de l'empreinte.
  - En envoyant le message accompagné de son empreinte, le destinataire peut vérifier l'intégrité du message en recalculant l'empreinte à l'arrivée et en la comparant avec celle reçue.
  - Si les deux empreintes sont différentes, cela signifie que le fichier a été modifié ou altéré par une tierce personne.

  Applications des fonctions d' hachage :
  - **Stockage sécurisé des mots de passe** : Les mots de passe sont hachés avant d'être stockés pour éviter qu'ils ne soient facilement récupérés en cas de piratage.
  - **Signature numérique** : Elles permettent de créer une signature unique pour un document, garantissant son authenticité.
  - **Stockage de mots de passe** :Les mots de passe sont hachés avant d'être stockés pour éviter qu'ils ne soient facilement récupérés en cas de piratage.
  - **Vérification de l'intégrité des données** : Elles permettent de s'assurer qu'un fichier ou un message n'a pas été modifié lors de son transfert.

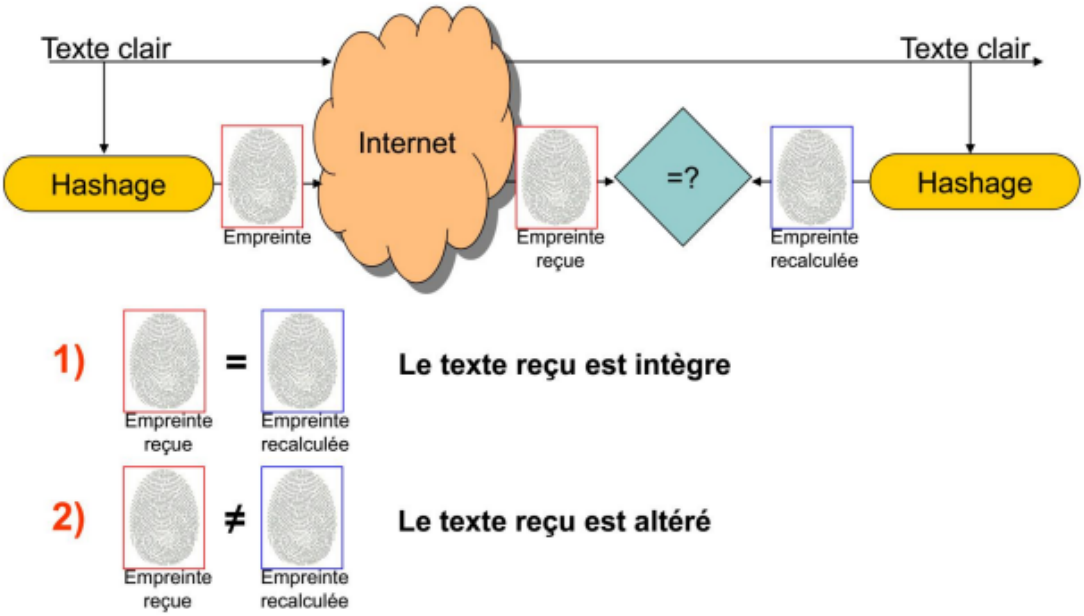

FIGURE 1.8 – Vérification de l'intégrité [1]

Propriétés des fonctions d' hachage Pour être un outil cryptographique efficace, la fonction d'hachage doit avoir les propriétés suivantes :

- **Propriété de compression et calcul rapide** : La fonction de hachage transforme un message de taille variable en une empreinte de taille fixe, et ce processus est rapide.
- **Résistance à la préimage** : Il est difficile de retrouver le message original à partir de empreinte (haché).
- **Résistance à la seconde préimage** : Il est difficile de trouver un autre message ayant la même empreinte que le premier.
- **Résistance aux collisions** : Il est très compliqué de trouver deux messages différents qui produisent exactement la même empreinte.

- **Signature Numérique**
  La signature numérique est un moyen qui permet de vérifier qu'un message a bien été envoyé par une personne spécifique, un peu comme une signature sur un document papier.

  Elle garantit aussi que le message n'a pas été modifié (intégrité) et que l'expéditeur ne peut pas nier l'avoir envoyé (non-répudiation).

  On peut également combiner la signature numérique avec le chiffrement pour assurer à la fois la confidentialité et l'authenticité du message.

  Propriétés de la signature numérique :
  - **Dépendance au signataire et au document** : La signature est unique pour chaque document et pour chaque signataire.
  - **Authenticité** : Il est impossible de découper une signature d'un message et de la recoller sur un autre.

- **Intégrité** : Le document signé ne peut pas être modifié après la signature.
- **Inaltérabilité** : La signature ne peut pas être falsifiée.
- **Non-répudiation** : Le signataire ne peut pas nier avoir signé le document.

Algorithme de la signature numérique avec RSA : La génération des paramètres pour la signature numérique suit le même principe que l'algorithme RSA. L'expéditeur A possède une clé publique (e, n) et une clé privée (d, n).

Voici les étapes pour la signature numérique avec RSA :

1. Alice choisit deux grands nombres premiers p et q.
2. Elle calcule $n = p \times q$, qui sera le module de chiffrement.
3. Calcul de $\phi(n) = (p-1) \times (q-1)$, où ϕ est la fonction d'Euler.
4. Choix d'un entier e tel que $1 < e < \phi(n)$ et $pgcd(e, \phi(n)) = 1$.
5. Calcul de d, l'inverse de e modulo $\phi(n)$, tel que $d \times e \equiv 1[\phi(n)]$.
6. La clé publique est (n, e) et la clé privée est (n, d).
7. Pour signer un message m, Alice calcule $s = m^d \mod n$.
8. Pour vérifier la signature, Bob calcule $s^e \mod n$ et vérifie s'il est égal à m. [6]

Exemple :

- Choisir $p = 3$ et $q = 11$.
- $n = 3 \times 11 = 33$ et $\phi(n) = (3-1) \times (11-1) = 2 \times 10 = 20$.
- Choisir $e = 3$, qui est premier avec 20.
- Calculer d tel que $d \times 3 \equiv 1[20]$. On trouve $d = 7$.
- La clé publique est $(33, 3)$ et la clé privée est $(33, 7)$.
- Pour signer $m = 4$, $s = 4^7 \mod 33 = 16$.
- Vérification : $s^e \mod n = 16^3 \mod 33 = 4$, donc la signature est valide.

1.6 Méthodes de cryptanalyse classiques

Les méthodes classiques de cryptanalyse sont principalement utilisées pour attaquer les anciens systèmes de chiffrement, notamment ceux basés sur des substitutions simples ou des permutations fixes. Deux approches fondamentales sont l'attaque exhaustive et l'analyse de fréquence.

1.6.1 Attaque exhaustive

L'attaque exhaustive, aussi appelée attaque par force brute, est une méthode simple mais très coûteuse en temps et en ressources. Elle est utilisée en cryptanalyse pour trouver un mot de passe ou une clé en testant toutes les combinaisons possibles jusqu'à identifier la bonne, permettant ainsi de déchiffrer le message.

1.6.1.1 Principe :

- L'attaquant génère toutes les clés possibles.
- Il applique chaque clé pour tenter de déchiffrer le message chiffré.
- Une fois que le texte déchiffré a un sens, la clé correcte est trouvée.

1.6.1.2 Complexité et faisabilité de l'attaque par force brute :

L'attaque par force brute devient plus difficile en fonction de plusieurs facteurs :

- **La taille de la clé :** Plus la clé est longue, plus le nombre de combinaisons à tester est élevé, ce qui ralentit l'attaque.

- **La puissance de calcul :** Avec l'évolution des technologies telles que les cartes graphiques (GPU) et le cloud computing, certaines clés trop faibles peuvent être facilement cassées.

- **Les algorithmes utilisés :** Certains systèmes, comme **RSA** et **DES**, sont conçus pour résister aux attaques par force brute. Cependant, DES est vulnérable en raison de sa petite taille de clé, tandis que RSA, avec une clé suffisamment grande, reste sécurisé contre ce type d'attaque.

1.6.1.3 Contre-mesures contre l'attaque exhaustive :

Pour protéger un cryptosystème contre une attaque par force brute, plusieurs stratégies peuvent être adoptées :

- **Utiliser une clé plus longue :** Par exemple, l'AES-256 utilise une clé de 256 bits, ce qui rend le nombre de combinaisons extrêmement élevé et donc l'attaque irréalisable en pratique.
- **Limiter le nombre d'essais :** Implémenter un système de temporisation, comme un blocage de 10 secondes après 5 tentatives incorrectes. Cela ralentit considérablement l'attaquant.
- **Utiliser du salage pour les mots de passe :** Par exemple, des algorithmes comme bcrypt, Argon2 ou PBKDF2 ajoutent un sel unique à chaque mot de passe, rendant l'attaque par force brute inefficace.
- **Adopter la cryptographie post-quantique :** Avec l'arrivée future des ordinateurs quantiques, certains algorithmes traditionnels deviendront vulnérables. Les algorithmes post-quantiques sont conçus pour résister à ces nouvelles menaces.

Exemple : Attaque par force brute sur le chiffre de César : Considérons le texte chiffré suivant :

```
at rdst rthpg, r'thi uprxat p rphhtg
```

Pour décrypter ce message, nous allons appliquer une attaque par force brute en testant les 26 décalages possibles. Cela peut être fait à la main ou à l'aide d'un simple programme.

Résultats des décalages :

- Décalage de 0 : `at rdst rthpg, r'thi uprxat p rphhtg`
- Décalage de 1 : `bu setu suiqh, s'uij vqsybu q sqiiuh`
- Décalage de 2 : `cv tfuv tvjri, t'vjk wrtzcv r trjjvi`
- Décalage de 3 : `dw ugvw uwksj, u'wkl xsuadw s uskkwj`
- Décalage de 4 : `ex vhwx vxltk, v'xlm ytvbex t vtllxk`
- Décalage de 5 : `fy wixy wymul, w'ymn zuwcfy u wummyl`
- Décalage de 6 : `gz xjyz xznvm, x'zno avxdgz v xvnnzm`
- Décalage de 7 : `ha ykza yaown, y'aop bwyeha w ywooan`
- Décalage de 8 : `ib zlab zbpxo, z'bpq cxzfib x zxppbo`
- Décalage de 9 : `jc ambc acqyp, a'cqr dyagjc y ayqqcp`
- Décalage de 10 : `kd bncd bdrzq, b'drs ezbhkd z bzrrdq`
- **Décalage de 11 : `le code cesar, c'est facile a casser`**
- Décalage de 12 : `mf dpef dftbs, d'ftu gbdjmf b dbttfs`
- Décalage de 13 : `ng eqfg eguct, e'guv hcekng c ecuugt`
- Décalage de 14 : `oh frgh fhvdu, f'hvw idfloh d fdvvhu`
- Décalage de 15 : `pi gshi giwev, g'iwx jegmpi e gewwiv`
- Décalage de 16 : `qj htij hjxfw, h'jxy kfhnqj f hfxxjw`
- Décalage de 17 : `rk iujk ikygx, i'kyz lgiork g igyykx`
- Décalage de 18 : `sl jvkl jlzhy, j'lza mhjpsl h jhzzly`
- Décalage de 19 : `tm kwlm kmaiz, k'mab nikqtm i kiaamz`
- Décalage de 20 : `un lxmn lnbja, l'nbc ojlrun j ljbbna`
- Décalage de 21 : `vo myno mockb, m'ocd pkmsvo k mkccob`
- Décalage de 22 : `wp nzop npdlc, n'pde qlntwp l nlddpc`
- Décalage de 23 : `xq oapq oqemd, o'qef rmouxq m omeeqd`
- Décalage de 24 : `yr pbqr prfne, p'rfg snpvyr n pnffre`
- Décalage de 25 : `zs qcrs qsgof, q'sgh toqwzs o qoggsf`

On observe qu'avec un décalage de 11, le message devient clair :

`le code césar, c'est facile à casser`

1.6.2 Analyse de fréquence

L'analyse de fréquence est l'une des méthodes classiques de cryptanalyse utilisée pour attaquer les cryptosystèmes par substitution(comme le chiffrement de César ou Vigenère.). Cette méthode repose sur l'étude statistique de la fréquence d'apparition des lettres dans un texte chiffré, en se basant sur les caractéristiques linguistiques de la langue utilisée.

1.6.2.1 Principe de l'analyse de fréquence

Dans chaque langue, certaines lettres apparaissent plus souvent que d'autres. Par exemple, en français, la lettre 'E' est la plus fréquente, représentant environ 16% du texte. L'idée principale de l'analyse de fréquence est d'identifier les lettres les plus fréquentes dans le texte chiffré et de les associer aux lettres les plus courantes dans la langue française.

L'attaque par analyse de fréquence se déroule en plusieurs étapes :

1. Calcul de la fréquence de chaque lettre dans le texte chiffré.
2. Comparaison avec les fréquences des lettres dans la langue française.
3. Substitution des lettres les plus fréquentes par 'E', 'A', 'S', etc.
4. **A**nalyse progressive pour reconstituer le texte clair.

1.6.2.2 Limites de l'analyse de fréquence :

Bien que cette méthode soit efficace pour casser les chiffrements par substitution simple, elle présente certaines limites :

- Inefficace contre les algorithmes modernes comme **DES** et **AES**.
- Difficulté face aux chiffrements polyalphabétiques comme **Vigenère**.
- Moins efficace sur les textes courts, car la distribution des lettres est moins représentative.

1.6.2.3 Outils utilisés pour l'analyse de fréquence :

Pour réaliser une analyse de fréquence efficace, plusieurs outils mathématiques et informatiques sont utilisés. Ces outils permettent d'automatiser l'analyse statistique et d'accélérer le processus de décryptage.

- **Outils mathématiques :**
  - **Distribution de fréquence des lettres** : permet de calculer la probabilité d'apparition de chaque lettre dans le texte chiffré.
  - **Histogramme de fréquence** : utilisé pour visualiser la répartition des lettres et identifier les lettres dominantes.
  - **Indice de coïncidence (IC)** : une mesure statistique permettant de détecter si un texte a été chiffré avec un chiffrement monoalphabétique ou polyalphabétique.
- **Outils informatiques :**
  - **Python** : Utilisé pour écrire des scripts d'analyse de fréquence et automatiser le comptage des lettres.
  - **NumPy et Matplotlib** : Bibliothèques Python pour l'analyse statistique et la visualisation des résultats sous forme de graphiques.
  - **SageMath** : Un outil puissant pour l'analyse cryptographique et l'analyse de fréquence sur des systèmes plus complexes.
  - **CryptoCrack** : Un logiciel spécialisé pour casser les chiffrements classiques comme César, Vigenère et Affine.
- **Outils linguistiques :**
  - **Fréquence des lettres en français** : Base de données de fréquence des lettres dans la langue française.
  - **Corpus textuel** : Utilisé pour comparer les fréquences obtenues avec celles d'un texte en clair.

Exemple : Voici par ordre décroissant des fréquences la répartition des lettres en français :

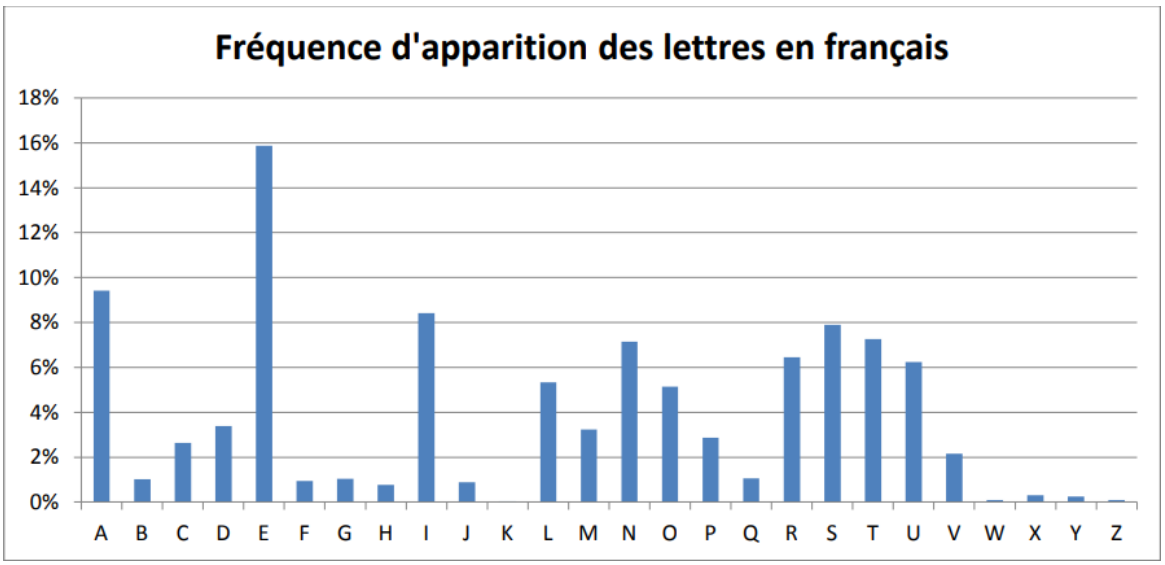

FIGURE 1.9 – Fréquence d'apparition des lettres en français
[7]

Lettre	Fréquence (%)	Lettre	Fréquence (%)	Lettre	Fréquence (%)
E	17,76	A	7,68	S	8,23
R	6,81	U	6,05	L	5,89
P	3,24	M	2,72	Q	1,34
B	0,80	H	0,64	X	0,54
K	0,01	W	0,00		
N	7,61	T	7,30	I	7,23
O	5,34	D	3,60	C	3,32
V	1,27	G	1,10	F	1,06
Y	0,21	J	0,19	Z	0,07

Voici un texte chiffré en utilisant le chiffre de César, dont la clé est inconnue :

SEGELAZEW AOP QJ LNKFAP Z AJUYUHKLAZEA CNWPQEPA AYNEPA YKKLANWPERAIAJP

Analyse

- La lettre la plus fréquente dans le texte chiffré est **A**.
- En français, la lettre la plus fréquente est **E**.
- On suppose donc que $\mathbf{A = E}$.

Calcul du décalage

$$\textit{Décalage} = (26 + (E - A)) \mod 26 = (26 + 4) \mod 26 = 4 \tag{1.1}$$

Message déchiffré

En appliquant le décalage de 4 à tout le texte, on obtient :

Wikipédia est un projet d'encyclopédie gratuite écrite coopérativement.

1.7 Méthodes de cryptanalyse moderne

Avec l'évolution des algorithmes de chiffrement, les techniques classiques de cryptanalyse se sont révélées inefficaces face aux chiffrements modernes tels que DES, AES ou encore RSA. Pour répondre à ces défis, de nouvelles méthodes d'analyse ont émergé, s'appuyant sur des concepts mathématiques avancés. Parmi les approches les plus puissantes figurent la cryptanalyse linéaire et la cryptanalyse différentielle.

1.7.1 Cryptanalyse Linéaire

La cryptanalyse linéaire est une technique introduite par Matsui en 1993 pour attaquer les chiffrements par blocs, notamment le DES. Elle consiste à exploiter des approximations linéaires entre les bits du texte clair, du texte chiffré et de la clé secrète. Cette méthode est particulièrement efficace contre les chiffrements basés sur des réseaux de substitution-permutation (SPN). [15]

Objectifs de la cryptanalyse linéaire L'objectif principal est d'identifier des relations linéaires biaisées qui ne se produisent pas avec une probabilité uniforme (i.e., différente de 0,5), afin d'en déduire des informations sur la clé secrète. Plus précisément, la cryptanalyse linéaire permet de :

- Évaluer la résistance d'un algorithme face aux attaques analytiques.
- Exploiter les biais statistiques dans les approximations linéaires.
- Réduire l'espace de recherche en éliminant les clés incompatibles avec les biais observés.
- Développer des attaques pratiques contre des chiffrements comme le DES ou certains SPN.

1.7.2 Cryptanalyse différentielle

La cryptanalyse différentielle a été développée par Biham et Shamir au début des années 1990 pour analyser la sensibilité des chiffrements par blocs aux variations du texte clair. Elle repose sur l'étude de la propagation des différences entre deux textes clairs à travers le processus de chiffrement, dans le but d'en déduire des informations sur la clé. [14]

Objectifs de la cryptanalyse différentielle Cette méthode vise à :

- Étudier comment une différence dans le texte clair influence la différence obtenue dans le texte chiffré.
- Identifier des paires d'entrée produisant des différences de sortie avec une forte probabilité.
- Détecter des schémas récurrents dans la propagation des différences.
- Exploiter ces schémas pour retrouver la clé secrète ou en réduire l'espace de recherche.

1.7.3 Comparaison avec la cryptanalyse différentielle

La cryptanalyse linéaire est souvent comparée à une autre technique classique d'attaque appelée cryptanalyse différentielle. Bien que les deux approches visent à retrouver des in formations sur la clé secrète en exploitant des propriétés du chiffrement, leurs principes et méthodes diffèrent fondamentalement. Le tableau ci-dessous résume les principales différences entre ces deux méthodes.

Critère	Cryptanalyse différentielle	Cryptanalyse linéaire
Principe	Étudie la manière dont les différences dans les paires de textes clairs influencent les différences dans les textes chiffrés	Approxime les relations entre les bits d'entrée, de sortie et de clé à l'aide d'équations linéaires (XOR)
Méthode utilisée	Analyse la propagation des différences (différences en entrée vs différences en sortie)	Utilise des équations linéaires approximatives et calcule leur biais statistique
Objectif	Identifier des paires différentielles à forte probabilité pour déduire des bits de la clé	Exploiter des approximations biaisées pour retrouver une partie de la clé
Efficacité	Très efficace contre les chiffrements ayant des structures de diffusion faibles ou des S-boites vulnérables	Particulièrement efficace sur certains algorithmes comme DES, mais dépend fortement du biais et de la structure du chiffrement
Volume de données requis	Utilise un grand nombre de paires avec une différence choisie dans le texte clair	Nécessite généralement encore plus de paires (texte clair - texte chiffré) pour obtenir des résultats statistiquement significatifs

TABLE 1.4 – Comparaison entre la cryptanalyse différentielle et la cryptanalyse linéaire

1.8 Conclusion

Ce chapitre nous a permis de comprendre les bases de la cryptographie et de découvrir les différents types de cryptosystèmes. Ces connaissances sont essentielles pour aborder la suite de notre travail, qui porte sur les techniques d'attaque contre ces systèmes.

Chapitre 2

Cryptanalyse linéaire

2.1 Introduction

Dans ce chapitre, nous présentons la cryptanalyse linéaire, une méthode statistique utilisée pour analyser la sécurité des algorithmes de chiffrement. Nous allons expliquer son principe de fonctionnement et les différentes étapes permettant de l'appliquer, notamment sur un réseau SPN.

2.2 Histoire

la cryptanalyse linéaire a été développée au début des années 1990 par Mitsuru Matsui, chercheur chez Mitsubishi Electric, dans le but de tester la robustesse du standard de chiffrement DES (Data Encryption Standard). Avant cette période, la plupart des attaques connues étaient basées sur la cryptanalyse différentielle, introduite en 1990 par Biham et Shamir. Matsui a proposé une approche nouvelle, fondée sur l'exploitation de relations linéaires approximatives entre les bits du texte clair et du texte chiffré, pour en déduire des informations sur la clé secrète.

En 1993, lors de la conférence EUROCRYPT, Matsui publie les premiers résultats concrets de cette approche. Il applique la cryptanalyse linéaire sur une version réduite de DES (avec moins de tours), montrant qu'il est possible de récupérer des sous-clés à l'aide d'un grand nombre de paires clair/chiffré. L'année suivante, en 1994, il réussit à casser le DES complet (16 tours) en utilisant environ 2^{43} paires de données, ce qui constitue à l'époque une avancée majeure dans le domaine de la cryptanalyse pratique. [15]

L'impact de cette attaque est considérable. Elle pousse la communauté cryptographique à repenser la sécurité de nombreux chiffrements et contribue à affaiblir la confiance dans DES, déjà menacé par sa courte taille de clé (56 bits). Depuis, la cryptanalyse linéaire devient un outil standard dans l'analyse des algorithmes symétriques. Elle est systématiquement prise en compte lors de la conception de nouveaux chiffrements, comme AES ou les chiffrements légers destinés aux objets connectés.

Au fil des années, plusieurs variantes ont vu le jour pour renforcer la puissance de cette méthode, comme la cryptanalyse linéaire multidimensionnelle, la boomerang linear cryptanalysis ou encore la combinaison avec la cryptanalyse différentielle. Ces extensions ont permis de tester des chiffrements plus résistants et d'élargir le champ d'application de l'approche linéaire.

Aujourd'hui, plus de trente ans après sa création, la cryptanalyse linéaire reste une référence incontournable dans le domaine de la sécurité symétrique. Elle est enseignée dans les cursus en cryptographie, utilisée comme base de comparaison dans les publications scientifiques, et continue d'évoluer au rythme des avancées en analyse mathématique et en modélisation statistique.

2.3 Principe de la cryptanalyse linéaire

La cryptanalyse linéaire est une technique puissante d'attaque à *texte clair connu*, appliquée aux chiffrements par blocs, tels que le *Data Encryption Standard* (DES) et les réseaux de substitution-permutation (SPN). Elle repose sur l'exploitation de relations **linéaires approximatives** entre les bits du texte clair (x), du texte chiffré (y) et de la clé secrète (k), dans le but de réduire l'espace de recherche et retrouver la clé.

2.3.1 Concept de base

Dans une attaque à texte clair connu, l'attaquant dispose d'un grand nombre de paires (x, y), où x est le texte clair et y le texte chiffré, tous chiffrés avec la même clé inconnue. L'objectif est de retrouver cette clé en exploitant des failles statistiques dans la structure du chiffrement. La cryptanalyse linéaire repose sur des équations binaires basées sur l'opération XOR ($\oplus$), de la forme :

$$x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus y_{j_1} \oplus y_{j_2} \oplus \cdots \oplus k_{h_1} \oplus k_{h_2} \oplus \cdots = 0 \tag{2.1}$$

qui représentent respectivement des bits spécifiques du texte clair, du texte chiffré, et de la clé .

Ces équations, appelées *approximations linéaires*, ne sont pas toujours exactes, mais elles sont vraies avec une certaine probabilité p. Le *biais linéaire* est défini par :

$$\varepsilon = \left| p - \frac{1}{2} \right|, \tag{2.2}$$

et mesure l'écart par rapport à un comportement purement aléatoire ($p = 0.5$). Un biais significatif (ε élevé) indique que l'équation est plus souvent vraie (ou fausse) que prévu, ce qui peut révéler une faiblesse exploitable du chiffrement.

2.3.2 Rôle des sous-clés

Les équations linéaires font souvent intervenir des bits de sous-clés intermédiaires, généralement ceux des derniers tours, qui ne sont pas directement observables. Leur influence est absorbée dans le biais observé. La valeur des bits de sous-clé (k_h) influence notamment le signe du biais : positif ou négatif. En testant statistiquement plusieurs hypothèses de sous-clés sur un grand nombre de paires (x, y), l'attaquant peut déterminer la valeur la plus probable de la sous-clé.

2.3.3 Algorithmes de Matsui

Mitsuru Matsui a proposé deux algorithmes de cryptanalyse linéaire. Ces algorithmes exploitent des approximations linéaires entre les bits de x, y et k, et s'appuient sur le principe du maximum de vraisemblance : on retient la valeur de clé la plus cohérente avec les données observées.

2.3.3.1 Algorithme 1 de Matsui

Cet algorithme [15]permet d'estimer un ou plusieurs bits de la clé à partir d'une seule équation linéaire approximative.

Algorithm 1 Procédure de cryptanalyse linéaire (estimation du membre droit)

Require: Une approximation linéaire E_j de la forme :

$$x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus y_{j_1} \oplus y_{j_2} \oplus \cdots = k_{h_1} \oplus k_{h_2} \oplus \cdots$$

Require: Un ensemble de N paires (x_j, c_j)
Ensure: Estimation du membre droit (0 ou 1)

```
Initialiser T ← 0
for chaque paire (xçj, c_j) do
    if E_j = 0 then
        T ← T + 1
    end if
end for
if T/N > 1/2 et p > 1/2 ou T/N < 1/2 et p < 1/2 then
    return 0
else
    return 1
end if
```

Cet algorithme donne une information binaire sur la clé. Pour déterminer plusieurs bits, il faut plusieurs équations indépendantes avec un biais suffisant.

2.3.3.2 Algorithme 2 de Matsui

Cet algorithme [15], plus efficace en pratique, permet de tester toutes les valeurs possibles d'une sous-clé (souvent celle du dernier tour) et de choisir celle qui maximise le biais observé.

Algorithm 2 Recherche du biais maximal d'une approximation linéaire sans clé

Require: Une approximation linéaire E_j de la forme :

$$x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus y_{j_1} \oplus y_{j_2} \oplus \cdots = 0$$

Require: Un ensemble de N paires (x_j, c_j)
Ensure: Le biais maximal $\varepsilon_{\max}$

```
Initialiser ε_max ← 0
for haque valeur possible de la sous-clé k_i do
    Initialiser χ ← 0
    for chaque paire (x_j, c_j) do
        if E_j = 0 then
            χ ← χ + 1
        end if
    end for
    Calculer le biais : ε ← |χ/N − 1/2|
    if ε > ε_max then
        ε_max ← ε
    end if
end for
return ε_max
```

Cet algorithme permet de comparer plusieurs hypothèses de sous-clés à l'aide d'un test statistique basé sur le biais. C'est celui qui est le plus souvent utilisé dans les attaques réelles sur DES ou SPN.

2.4 Présentation du chiffrement DES

2.4.1 Définition

Le **DES** (*Data Encryption Standard*) est un ancien algorithme de chiffrement symétrique par blocs conçu pour protéger les données confidentielles. Il a été adopté comme standard par le gouvernement des États-Unis en 1977. Son objectif est de transformer un texte clair (message lisible) en un texte chiffré (incompréhensible) à l'aide d'une clé secrète partagée.

Les principales caractéristiques du chiffrement DES sont les suivantes :

- **Type de chiffrement** : Symétrique (la même clé est utilisée pour chiffrer et déchiffrer).
- **Taille du bloc** : 64 bits (le message est traité par blocs de 64 bits).
- **Taille de la clé** : 64 bits, mais seulement 56 bits sont réellement utilisés (les 8 bits restants servent à la détection d'erreurs de parité).
- **Nombre de tours** : 16 tours de transformation pour renforcer la sécurité.
- **Structure** : Utilise une structure appelée **réseau de Feistel**, où le bloc est divisé en deux moitiés qui sont traitées de manière alternée à chaque tour.

L'algorithme repose sur plusieurs opérations simples : des **permutations**, des **substitutions** (grâce à des boîtes appelées **S-boîtes**), et des opérations logiques comme le **OU exclusif (XOR)**. L'objectif est de rendre le texte chiffré totalement différent du texte original, même en cas de légère modification du message ou de la clé.

Bien que le DES ait été largement utilisé pendant plusieurs décennies, il est aujourd'hui considéré comme insuffisamment sécurisé en raison de la petite taille de sa clé, qui peut être cassée par force brute avec les capacités informatiques modernes. Il a été remplacé par des algorithmes plus puissants comme l'**AES (Advanced Encryption Standard)**. Toutefois, il reste un excellent outil pédagogique pour comprendre les bases de la cryptographie moderne.

2.4.2 Limites de la cryptanalyse linéaire sur DES

Malgré sa puissance, la cryptanalyse linéaire présente plusieurs limitations :

- **Grand volume de données requis** : Des millions de paires texte clair/texte chiffré sont souvent nécessaires.
- **Temps de calcul non négligeable** : Le traitement de toutes les données demande des ressources importantes.
- **Biais très faibles** : Les approximations sur plusieurs tours ont des biais proches de zéro, donc difficiles à exploiter.
- **S-boites bien conçues** : Les S-boites du DES sont résistantes, ce qui limite la puissance de cette attaque.

2.5 Substitutions et permutations

Dans les systèmes de chiffrement modernes, deux opérations fondamentales sont largement utilisées pour renforcer la sécurité : la **substitution** et la **permutation**. Ces deux transformations jouent un rôle essentiel dans la complexification du lien entre le texte clair et le texte chiffré.

2.5.1 S-boîte (boîte de substitution)

Une *S-boîte* (Substitution Box) est une fonction utilisée pour substituer un petit bloc de bits par un autre, selon une table de correspondance prédéfinie. Elle représente une opération non linéaire, essentielle pour introduire la **confusion** dans les algorithmes de chiffrement, rendant ainsi la relation entre l'entrée (texte clair) et la sortie (texte chiffré) difficile à inverser.

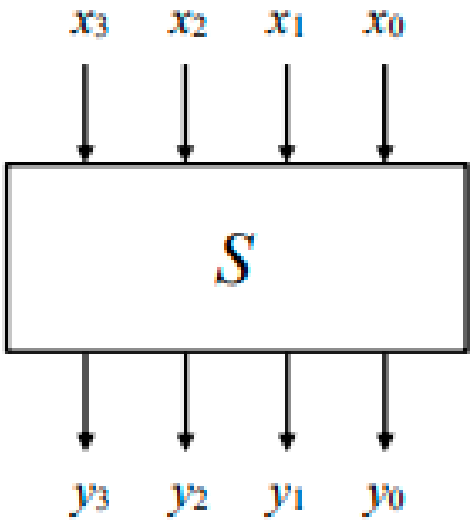

FIGURE 2.1 – S-boîte de 4 bits.

Par exemple, une S-boîte à 3 bits peut être définie comme suit :

x	0	1	2	3	4	5	6	7
S(x)	7	6	5	4	3	2	0	1

TABLE 2.1 – Table de substitution (S-boîte)

Cela signifie que l'entrée 0 donne 7, l'entrée 1 donne 6, et ainsi de suite.
Le principal inconvénient des S-boîtes réside dans leur coût mémoire, notamment lorsque leur taille est importante .

2.5.2 P-boîte (boîte de permutation)

Une *P-boîte* (Permutation boîte) est une boîte qui réorganise les bits d'un bloc sans modifier leurs valeurs. Elle applique une permutation fixe, spécifiant la position de chaque bit de sortie en fonction de sa position d'entrée.
Par exemple, une P-boîte de 8 bits peut être définie ainsi :

x	0	1	2	3	4	5	6	7
$\pi(x)$	3	5	1	0	4	7	6	2

TABLE 2.2 – Table de permutation (P-boîte)

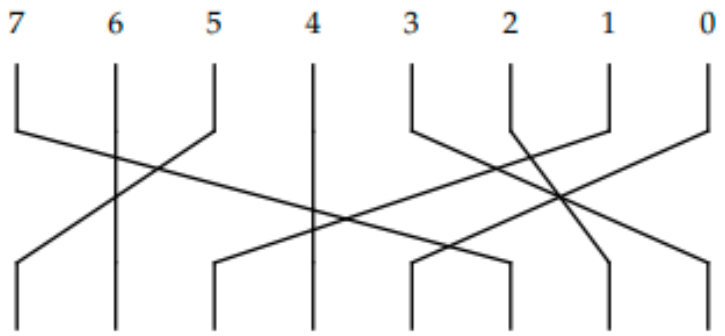

FIGURE 2.2 – Exemple d'une P-boîte de 8 bits.

Contrairement aux S-boîtes, les P-boîtes sont linéaires, simples à implémenter et peu coûteuses en mémoire. Dans certains algorithmes comme le DES, elles peuvent aussi effectuer des expansions ou des réductions de taille, en dupliquant ou en supprimant certains bits.

Cependant, ni la substitution seule ni la permutation seule ne suffisent à garantir un chiffrement sécurisé. C'est pourquoi les algorithmes modernes combinent ces deux opérations dans des structures appelées **réseaux de substitution-permutation**.

2.6 Le réseau de substitution-permutation (SPN)

2.6.1 Définition

Un **réseau de substitution-permutation (SPN)** est une architecture cryptographique qui combine plusieurs couches de substitutions (via des S-boîtes) et de permutations (via des P-boîtes), intercalées avec des opérations sur une clé secrète.
Chaque cycle du SPN est appelé une *tour*. Chaque tour comprend :

- une substitution non linéaire (S-boîte),
- une permutation linéaire (P-boîte),
- une combinaison avec une sous-clé (généralement via un XOR).

La répétition de plusieurs tours permet de renforcer la **confusion** et la **diffusion**.

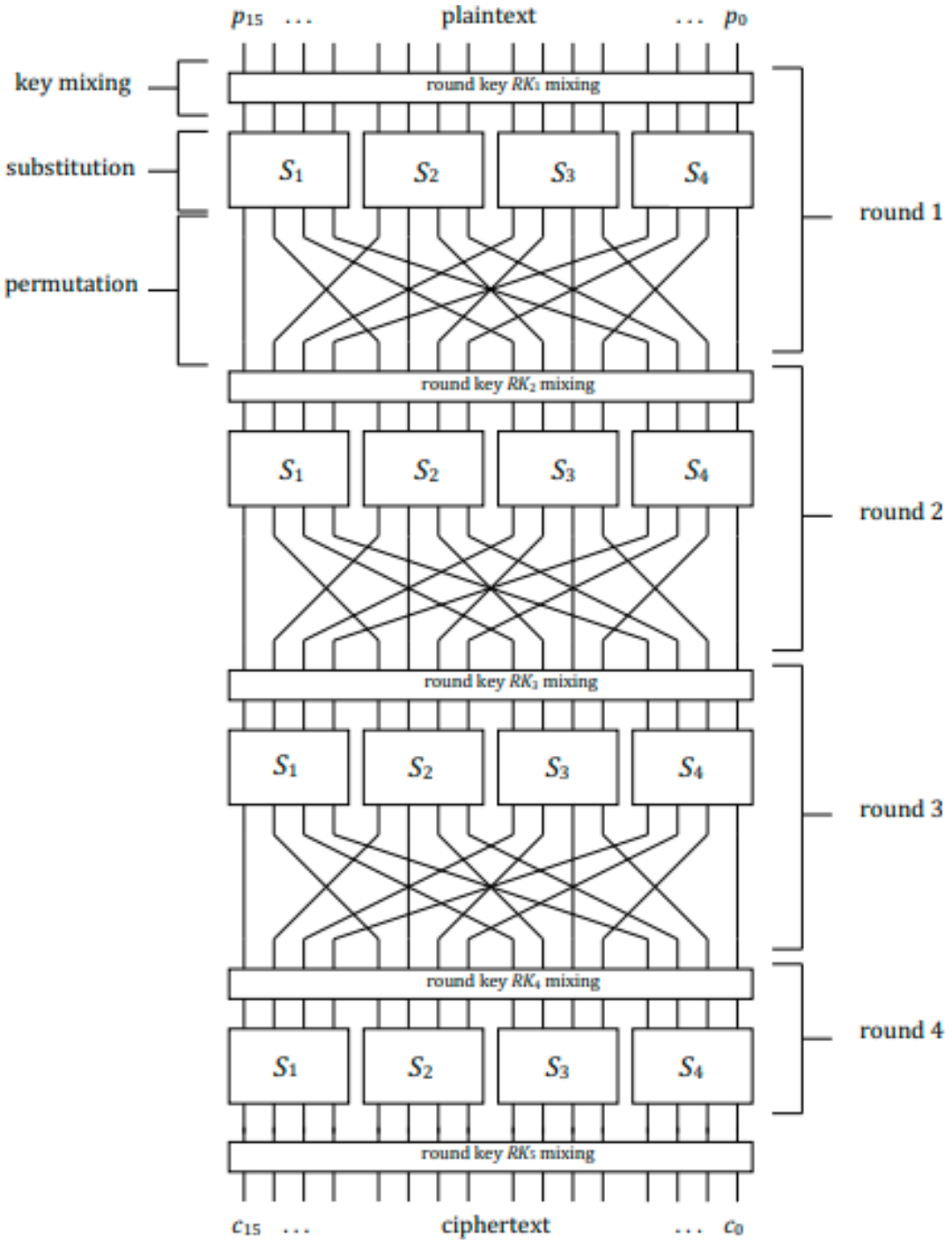

FIGURE 2.3 – Un SPN de 16 bits(4 tours)

2.6.2 Algorithme de chiffrement d'un réseau SPN

La fonction f_S désigne l'application parallèle de la S-boîte à chaque sous-bloc du message clair. Si le bloc s est divisé en m sous-blocs $x_{(1)}, x_{(2)}, \ldots, x_{(B)}$, alors :

$$f_S(s) = (S(x_{(1)}), S(x_{(2)}), \ldots, S(x_{(B)})).$$

Par exemple, dans un réseau SPN comportant 3 S-boîtes, on a :

$$f_S(x_{(1)}, x_{(2)}, x_{(3)}) = (S(x_{(1)}), S(x_{(2)}), S(x_{(3)})).$$

Algorithm 3 Chiffrement d'un réseau SPN (adapté de [25])

Require: Nombre de tours r, bloc clair $s \in \mathbb{B}^n$, sous-clés $k_1, \ldots, k_{r+1}$, S-boîte S, P-boîte π
Ensure: Bloc chiffré s

1: **for** $i = 1$ à $r - 1$ **do**
2: $s \leftarrow s \oplus k_i$ ▷ Ajout de la sous-clé
3: $s \leftarrow f_S(s)$ ▷ Substitution via la S-box
4: $s \leftarrow \pi(s)$ ▷ Permutation sauf au dernier tour
5: **end for**
6: **for** $i = r$ **do**
7: $s \leftarrow s \oplus k_r$
8: $s \leftarrow f_S(s)$
9: $s \leftarrow s \oplus k_{r+1}$
10: **end for** ▷ Ajout de la dernière sous-clé
11: **return** s

2.7 Analyse d'un réseau SPN

Le réseau de substitution-permutation (SPN) est une architecture fondamentale en cryptographie symétrique, particulièrement utilisée dans la conception des chiffrements par blocs modernes. Son fonctionnement repose sur l'application répétée de plusieurs transformations élémentaires organisées en tours successifs. Chaque tour est constitué d'opérations déterministes : l'addition de sous-clés, une substitution non linéaire à l'aide de S-boîte, et une permutation linéaire via une P-boîte. Cette combinaison vise à renforcer les propriétés de confusion et de diffusion, essentielles à la sécurité cryptographique.

2.7.1 Paramètres initiaux d'un SPN

- Nombre de tours r (par exemple, $r = 3$).
- Taille des blocs : n bits (par exemple, $n = 12$ bits).
- S-boîte : Substitution non linéaire $S : \mathbb{B}^n \to \mathbb{B}^n$.
- P-boîte : Permutation des bits du bloc.
- Clé principale k découpée en $r + 1$ sous-clés $k_1, k_2, \ldots, k_{r+1}$, chacune de taille n bits.

2.7.2 Structure générale du chiffrement

Un chiffrement SPN opère sur un bloc de texte clair x de taille fixe (par exemple 12 ou 16 bits), en appliquant r tours successifs, suivis d'une opération de chiffrement finale. Le processus peut être schématisé de manière abstraite comme suit :

$$x \xrightarrow{\oplus k_0} y_0 \xrightarrow{S} z_0 \xrightarrow{\pi} x_1 \xrightarrow{\oplus k_1} \cdots \xrightarrow{\pi} x_{r-1} \xrightarrow{\oplus k_{r-1}} y_{r-1} \xrightarrow{S} z_{r-1} \xrightarrow{\oplus k_r} c$$

- x est le texte clair initial.
- k_i est la sous-clé utilisée à l'étape i, dérivée de la clé principale.
- S désigne l'opération de substitution non linéaire à l'aide de S-boîtes
- π est la permutation linéaire appliquée sur les bits du bloc.
- c est le texte chiffré final.

Chaque tour du réseau applique ces trois transformations dans un ordre déterminé : d'abord l'addition de clé, suivie de la substitution, puis de la permutation. Le dernier tour est souvent modifié pour inclure seulement l'addition de la dernière clé après la dernière substitution, sans permutation, afin de préserver la structure réversible du chiffrement.

2.7.3 Étapes de transformation d'un tour

1. Addition de clé ($\oplus k_i$)

L'addition de clé est une opération bit-à-bit (XOR) entre l'état interne et une sous-clé k_i.

$$y_i = x_i \oplus k_i$$

- Cette opération est linéaire, simple à implémenter, mais cruciale car elle introduit la dépendance du chiffrement à la clé secrète.
- Chaque sous-clé est généralement extraite ou dérivée de la clé principale via un algorithme d'expansion de clé.

2. Substitution non linéaire (S)

Après l'addition de clé, l'état intermédiaire est transformé bloc par bloc à l'aide de S-boîte.

$$z_i = S(y_i)$$

- L'état est découpé en sous-blocs , chacun étant soumis à une fonction de substitution non linéaire appelée S-boîte.
- Cette transformation introduit de la confusion, rendant les relations entre l'entrée et la sortie du chiffrement non linéaires et donc plus difficiles à inverser sans connaître la clé.

3. Permutation linéaire (π)

L'état résultant de la substitution est ensuite réorganisé par une permutation fixe des bits, appelée P-boîte.

$$x_{i+1} = \pi(z_i)$$

- Chaque bit est déplacé selon une table de permutation prédéfinie, assurant la diffusion : un bit d'entrée affecte plusieurs bits de sortie au fil des tours.
- Cette diffusion permet de propager l'effet d'un seul bit à travers tout le bloc, ce qui augmente la complexité du chiffrement.

2.7.4 Tour final et chiffrement

Après le dernier tour complet, le chiffrement se termine par une dernière substitution (via S-boîte) suivie d'une ultime addition de clé :

$$c = S(y_{r-1}) \oplus k_r$$

Cette étape assure une dernière couche de confusion et masque le résultat final avec une nouvelle sous-clé.

2.7.5 Rôle cryptographique des composantes

L'efficacité d'un réseau SPN repose sur l'interaction harmonieuse entre ses composantes :

- **Confusion** : assurée par les S-boîtes (S), elle rend le lien entre texte clair et texte chiffré complexe et non linéaire.
- **Diffusion** : garantie par la permutation (π), elle disperse l'information à travers l'ensemble du bloc pour empêcher la localisation des effets d'un bit unique.
- **Séparation des clés** : l'usage de sous-clés distinctes ($k_0, \ldots, k_r$) pour chaque tour empêche l'attaquant de réduire la complexité du chiffrement à un seul XOR global.

2.7.6 Avantage pour la cryptanalyse linéaire

Cette structure modulaire rend les SPN particulièrement adaptés à l'étude théorique des attaques comme la cryptanalyse linéaire. Chaque couche (S-boîte, P-boîte, clé) peut être isolée et analysée quant à sa contribution au biais statistique entre les bits d'entrée et de sortie. Cela permet, par exemple, de concevoir des approximations linéaires exploitables dans des attaques par biais, tout en maintenant une structure suffisamment réaliste pour simuler des chiffrements modernes comme AES.

2.8 Étapes d'application de la cryptanalyse linéaire sur un réseau SPN

2.8.1 Définition des composants fondamentaux — S-boîte et P-boîte

S-boîte :

La **S-boîte** est une fonction de substitution non linéaire qui transforme un bloc d'entrée de taille fixe (souvent 4 bits) en un bloc de même taille, de manière déterministe et préétablie. Elle est responsable de l'introduction de la **non-linéarité** dans le chiffrement, essentielle pour résister aux attaques algébriques.
Pour une S-boîte de 4 bits, on a :

$$S : \mathbb{B}^4 \to \mathbb{B}^4$$

L'exemple suivant illustre une telle S-boîte, où chaque entrée $x \in \{0, \ldots, F\}$ est substituée par une valeur $S(x)$:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	0	4	5	F	1	2	7	E	6	C	A	3	9	B	8	D

TABLE 2.3 – Exemple de S-boîte pour un bloc de 12 bits

Cette fonction sera utilisée ultérieurement pour calculer les biais statistiques des approximations linéaires dans la Table d'Approximation Linéaire (LAT).

P-boîte :

La **P-boîte** est une permutation fixe qui réorganise les positions des bits d'un bloc. Elle vise à propager l'effet d'un bit d'entrée sur plusieurs bits en sortie, assurant ainsi la **diffusion** dans le réseau.
Pour un bloc de 12 bits, la P-boîte suivante déplace chaque bit de position i vers la position $P(i)$ donnée :

TABLE 2.4 – Exemple de P-boîte pour un bloc de 12 bits

i	0	1	2	3	4	5	6	7	8	9	A	B
$P(i)$	0	4	8	1	5	9	2	6	A	3	7	B

Cette permutation est appliquée après les substitutions à chaque tour du SPN, sauf au dernier. Elle joue un rôle clé dans l'expansion des caractéristiques linéaires à travers plusieurs S-boîtes.

Ces deux composants — S-boîte et P-boîte — constituent la base de l'architecture SPN et seront exploités dans les étapes suivantes de la cryptanalyse linéaire pour construire des approximations linéaires efficaces entre le texte clair et le texte chiffré.

2.8.2 Construction de la table d' approximation linéaire (LAT)

La **table d'approximation linéaire** (LAT — Linear Approximation Table) est un outil fondamental en cryptanalyse linéaire. Elle permet de quantifier, pour une S-boîte donnée, les corrélations linéaires existantes entre certaines combinaisons de bits d'entrée et de sortie.

Définition : Pour chaque couple de masques $(\alpha, \beta) \in \mathbb{B}^n \times \mathbb{B}^n$, on s'intéresse à la probabilité que l'équation suivante soit satisfaite [8] :

$$\alpha \cdot x = \beta \cdot S(x)$$

où x est un élément d'entrée de la S-boîte, $S(x)$ sa sortie, et $\cdot$ désigne le produit scalaire binaire (XOR des bits pondérés) défini par :

$$\alpha \cdot x = \bigoplus_{i=0}^{n-1} \alpha_i x_i$$

Formule de comptage : Pour chaque (α, β), on définit $N_L(\alpha, \beta)$ comme le nombre de cas où l'approximation est vérifiée parmi les 2^n entrées possibles :

$$N_L(\alpha, \beta) = |\{x \in \mathbb{B}^n \mid \alpha \cdot x \oplus \beta \cdot S(x) = 0\}|$$

Centrement de la table : Pour mieux détecter les biais, on recentre les valeurs autour de zéro :

$$\mathrm{LAT}[\alpha][\beta] = N_L(\alpha, \beta) - 2^{n-1}$$

Ainsi, une valeur LAT de 0 indique une absence de biais (probabilité exactement égale à 0.5), tandis qu'une valeur élevée en valeur absolue indique une forte corrélation exploitable.

2.8.2.1 Algorithme de construction de la table LAT

Cet algorithme est basé sur la méthode proposée dans, adaptée ici pour une S-boîte 4x4.

Algorithm 4 Construction de la LAT pour une S-boîte de 4 bits

Require: S-box $S : \{0, \ldots, 15\} \to \{0, \ldots, 15\}$
Ensure: LAT[16][16] contenant $N_L(\alpha, \beta) - 8$

```
for all α ∈ {0,...,15} do
    for all β ∈ {0,...,15} do
        N_L ← 0
        for all x ∈ {0,...,15} do
            x_bits ← bits(x)
            s_bits ← bits(S(x))
            if α · x_bits ⊕ β · s_bits = 0 then
                N_L ← N_L + 1
            end if
        end for
        LAT[α][β] ← N_L − 8
    end for
end for
return LAT
```

2.8.2.2 Tableau LAT obtenu

Voici la table LAT obtenue pour la S-boîte définie à l'étape précédente, avec $\text{LAT}[\alpha][\beta] = N_L - 8$. Les valeurs positives indiquent une corrélation, les négatives une anti-corrélation :

$\alpha\backslash\beta$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	V
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	-2	0	-2	2	0	2	0	4	-2	0	2	-2	0	2	4
2	0	2	4	-2	0	-2	0	-2	2	4	-2	0	2	0	2	0
3	0	0	0	0	2	-2	6	2	-2	2	2	-2	0	0	0	0
4	0	4	2	2	-2	2	0	0	0	0	2	-2	-2	-2	0	4
5	0	2	-2	-4	0	2	-2	4	0	2	2	0	0	2	2	0
6	0	2	-2	4	-2	-4	0	2	2	0	0	2	0	2	2	0
7	0	0	-2	2	0	4	2	2	2	2	-4	0	2	-2	0	0
8	0	0	0	0	-2	2	2	-2	4	0	4	0	2	2	-2	-2
9	0	-2	4	2	0	2	0	2	0	-2	0	-2	0	2	4	-2
A	0	2	0	2	6	0	-2	0	2	0	2	0	0	-2	0	-2
B	0	0	0	0	0	0	0	0	-2	-2	2	2	6	-2	2	2
C	0	-4	2	2	0	0	-2	2	0	4	2	2	0	0	-2	2
D	0	2	2	0	2	0	0	2	0	-2	-2	0	2	4	-4	2
E	0	2	2	0	0	2	2	0	-2	0	0	6	-2	0	0	-2
F	0	0	-2	2	2	2	0	-4	-2	2	0	0	0	4	2	2

TABLE 2.5 – Table d'approximation linéaire

2.8.3 Calcul du biais et de la probabilité

Le **biais** linéaire $\varepsilon(\alpha, \beta)$ mesure l'écart entre la probabilité réelle qu'une approximation linéaire soit satisfaite et la probabilité neutre de 0,5 (aucune corrélation). C'est une quantité centrale en cryptanalyse linéaire.

Après avoir construit la table d'approximations linéaires (LAT), par la formule [8] :

$$\text{LAT}[\alpha][\beta] = N_L(\alpha, \beta) - 2^{n-1}$$

on calcule le biais associé à chaque couple (α, β) comme suit :

$$\varepsilon(\alpha, \beta) = \frac{\text{LAT}[\alpha][\beta]}{2^n}$$

où n est le nombre de bits en entrée de la S-boîte.(ici $n = 4$).

Ainsi, un biais positif indique une corrélation directe entre les bits sélectionnés par α et ceux sélectionnés par β via la S-boîte, tandis qu'un biais négatif signale une anti-corrélation.

Voici le tableau des biais calculés pour chaque couple (α, β) avec la formule précédente :

TABLE 2.6 – Table des biais $\varepsilon(\alpha, \beta)$

$\alpha\backslash\beta$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0.500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	0.000	-0.125	0.000	-0.125	0.125	0.000	0.125	0.000	0.250	-0.125	0.000	0.125	-0.125	0.000	0.125	0.250
2	0.000	0.125	0.250	-0.125	0.000	-0.125	0.000	-0.125	0.125	0.250	-0.125	0.000	0.125	0.000	0.125	0.000
3	0.000	0.000	0.000	0.000	0.125	-0.125	0.375	0.125	-0.125	0.125	0.125	-0.125	0.000	0.000	0.000	0.000
4	0.000	0.250	0.125	0.125	-0.125	0.125	0.000	0.000	0.000	0.000	0.125	-0.125	-0.125	-0.125	0.000	0.250
5	0.000	0.125	-0.125	-0.250	0.000	0.125	-0.125	0.250	0.000	0.125	0.125	0.000	0.000	0.125	0.125	0.000
6	0.000	0.125	-0.125	0.250	-0.125	-0.250	0.000	0.125	0.125	0.000	0.000	0.125	0.000	0.125	0.125	0.000
7	0.000	0.000	-0.125	0.125	0.000	0.250	0.125	0.125	0.125	0.125	-0.250	0.000	0.125	-0.125	0.000	0.000
8	0.000	0.000	0.000	0.000	-0.125	0.125	0.125	-0.125	0.250	0.000	0.250	0.000	0.125	0.125	-0.125	-0.125
9	0.000	-0.125	0.250	0.125	0.000	0.125	0.000	0.125	0.000	-0.125	0.000	-0.125	0.000	0.125	0.250	-0.125
A	0.000	0.125	0.000	0.125	0.375	0.000	-0.125	0.000	0.125	0.000	0.125	0.000	0.000	-0.125	0.000	-0.125
B	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	-0.125	-0.125	0.125	0.125	0.375	-0.125	0.125	0.125
C	0.000	-0.250	0.125	0.125	0.000	0.000	-0.125	0.125	0.000	0.250	0.125	0.125	0.000	0.000	-0.125	0.125
D	0.000	0.125	0.125	0.000	0.125	0.000	0.000	0.125	0.000	-0.125	-0.125	0.000	0.125	0.250	-0.250	0.125
E	0.000	0.125	0.125	0.000	0.000	0.125	0.125	0.000	-0.125	0.000	0.000	0.375	-0.125	0.000	0.000	-0.125
F	0.000	0.000	-0.125	0.125	0.125	0.125	0.000	-0.250	-0.125	0.125	0.000	0.000	0.000	0.250	0.125	0.125

La **probabilité** que l'approximation linéaire soit correcte, c'est-à-dire que l'équation suivante soit satisfaite :

$$\alpha \cdot x = \beta \cdot S(x)$$

est donnée par la formule suivante :

$$\mathbb{P}[\alpha \cdot x = \beta \cdot S(x)] = \varepsilon(\alpha, \beta) + \frac{1}{2}$$

où $\varepsilon(\alpha, \beta)$ est le biais calculé précédemment.

2.8.4 Approximation linéaire finale et lemme d'empilement

2.8.4.1 Approximation finale et chemin linéaire

Lors d'une attaque par cryptanalyse linéaire sur un réseau SPN, l'objectif est de construire une **approximation linéaire globale** reliant certains bits du texte clair aux bits du texte chiffré. Cette approximation s'obtient en combinant des approximations locales sur plusieurs S-boîtes à travers les différents tours du chiffrement.
Une telle approximation permet, avec un biais non nul, de distinguer le chiffrement d'un comportement aléatoire, et d'en extraire des informations sur la clé.

Choix initial des masques et boîtes actives

On commence par analyser la table des biais $\varepsilon(\alpha, \beta)$ obtenue précédemment. On sélectionne une paire (α, β) correspondant à un biais maximal non nul (en général, celui dont $|\varepsilon|$ est le plus élevé).

- Ce choix détermine la ou les **boîtes actives du premier tour**, c'est-à-dire celles dont le masque d'entrée α est non nul.
- Les boîtes actives des tours suivants sont automatiquement déduites de la propagation du masque de sortie β à travers la permutation (P-boîte).

Remarque : Si plusieurs couples (α, β) présentent un biais égal maximal, on peut en choisir arbitrairement un parmi eux.

Propagation des masques et chemin d'approximation

Après le premier tour, le masque β est permuté selon la P-boîte, ce qui détermine le masque d'entrée α des S-boîte du tour suivant. Ce processus se répète à chaque tour jusqu'à la sortie du chiffrement.
Ainsi, la trajectoire du masque à travers le réseau SPN définit un **chemin linéaire** constitué d'une séquence de boîtes actives. Chaque boîte introduit une approximation locale.

Combinaison des approximations et élimination des états intermédiaires

Les approximations locales de chaque boîte active sont combinées à l'aide de l'opération XOR. Les états intermédiaires (bits internes du chiffrement) sont ainsi éliminés. L'équation résultante relie uniquement des bits du texte clair et des bits de la sortie (texte chiffré).
On obtient une équation de la forme :

$$x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus y_{j_1} \oplus \cdots \oplus y_{j_m} = 0$$

où x_i sont des bits du clair, y_j des bits du chiffré.

2.8.5 Lemme d'empilement

Le lemme d'empilement, proposé par Matsui, permet de calculer le biais total ε_{t} de l'approximation finale, lorsque plusieurs approximations locales indépendantes sont combinées.

$$\varepsilon_{\mathrm{t}} = 2^{n-1} \cdot \varepsilon_1 \cdot \varepsilon_2 \cdots \varepsilon_n \tag{2.3}$$

où :

- n est le nombre de boîtes actives,
- ε_i est le biais de l'approximation locale dans la $i_{\text{ème}}$ boîte.

2.8.6 Calcul du nombre de paires clair-chiffré

Le nombre N de paires clair-chiffré nécessaires pour distinguer statistiquement la bonne clé d'une mauvaise clé dépend du biais total ε_{t} de l'approximation linéaire utilisée. Il est donné par la formule suivante :

$$N = \frac{c}{\varepsilon_{\mathrm{t}}^2} \tag{2.4}$$

où :

- ε_{t} est le biais combiné obtenu avec le **lemme d'empilement**,
- c est une constante empirique de sécurité (par exemple 4 8 ou 16),
- N est le nombre de paires (x, y) nécessaires pour que la statistique soit significative.

Plus le biais est élevé, plus on peut détecter une différence significative avec un petit nombre d'échantillons.

Remarques

- Si ε_{t} est **faible** (proche de 0), alors N devient très grand : l'attaque devient impraticable.
- Si ε_{t} est **élevé** (proche de 0,1), alors N reste modéré, et l'attaque est efficace.

2.8.7 Test et estimation de la clé finale

Objectif de l'étape

Cette étape exploite l'équation d'approximation linéaire globale E_j obtenue précédemment afin d'identifier la bonne sous-clé du dernier tour du SPN. Cette méthode statistique repose sur le comptage du nombre de paires clair/chiffré satisfaisant $E_j = 0$ pour chaque combinaison de sous-clé candidate. La sous-clé ayant le plus fort biais empirique est considérée comme la meilleure estimation.

Déchiffrement partiel du dernier tour

Pour chaque combinaison candidate de sous-clé k, on effectue un déchiffrement partiel sur les sorties chiffrées c_j, limité aux positions correspondant aux S-boîtes actives :

(a) Application de la clé : $u_j = c_j \oplus k$
(b) Inversion des S-boîtes : $y_j = f_S^{-1}(u_j)$

Le vecteur y_j représente l'état interne juste avant la dernière couche de substitution.

Évaluation de l'équation d'approximation finale

L'équation linéaire finale E_j reliant certains bits de x_j (clair) et y_j (état interne) s'écrit sous la forme :

$$E_j = x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus y_{j_1} \oplus \cdots \oplus y_{j_m}$$

Cette équation est dite satisfaite si $E_j = 0$.

Comptage du nombre de satisfactions

Pour chaque sous-clé candidate, on compte :

$$\chi_k = \#\{j \in \{1, \ldots, N\} \mid E_j = 0\}$$

où N est le nombre total de paires (x_j, c_j) utilisées.

Estimation du biais empirique

On estime ensuite le biais empirique de la sous-clé candidate i par la formule :

$$\varepsilon_k = \left| \frac{\chi_k}{N} - \frac{1}{2} \right|$$

Un biais proche de zéro indique une clé incorrecte. Plus ε_i s'éloigne de 0,5, plus la sous-clé est probablement correcte.

Sélection de la meilleure sous-clé

La meilleure estimation de la clé est donnée par :

$$k^* = \arg\max_k \varepsilon_k$$

C'est cette sous-clé k^* qui est retenue comme estimation de la sous-clé réelle du dernier tour.

Remarques importantes sur les S-boîtes actives

- Le nombre de bits de clé à deviner dépend du nombre de **S-boîtes actives** au dernier tour :
  - 1 S-boîte active : $2^4 = 16$ combinaisons,
  - 2 S-boîtes actives : $2^8 = 256$ combinaisons,
  - 3 S-boîtes actives : $2^{12} = 4096$ combinaisons.
- Lorsque les 3 S-boîtes sont actives, l'équation d'approximation implique les 3 morceaux de la clé finale. Il est donc possible de retrouver la **clé complète** $k = (k_0, k_1, k_2)$.
- Si seules certaines S-boîtes sont actives (par exemple la première et la dernière), on ne peut estimer qu'une **sous-partie de la clé finale**, comme $(k_0, \_, k_2)$, les autres n'apparaissant pas dans l'équation E_j.

Récapitulatif de la procédure

(a) Parcourir toutes les sous-clés possibles pour les S-boîtes actives.

(b) Pour chaque paire (x_j, c_j) :
  - Effectuer un déchiffrement partiel de c_j,
  - Calculer l'état intermédiaire y_j,
  - Évaluer l'équation $E_j = 0$,
  - Incrémenter χ_i si E_j est satisfaite.

(c) Calculer le biais ε_i pour chaque sous-clé candidate.

(d) Retenir la clé qui maximise ε_i.

2.8.8 Algorithme d'éstimation de la clé finale

Algorithm 5 Estimation de la sous-clé finale par cryptanalyse linéaire

Require: Paires $(x_j, c_j)_{j=1}^N$, S-boîte inverse S^{-1}, équation E_j
Ensure: Sous-clé estimée $k^* = k$

Initialiser $\varepsilon_{\max} \leftarrow 0$, $k^* \leftarrow$ vide
for all sous-clés $k \in \{0, \ldots, 15\}^3$ **do**
  $\chi \leftarrow 0$
  for $j = 1$ à N **do**
    $u_j \leftarrow$ bits ciblés de $c_j \oplus k$ ▷ Supposons que les bits ciblés sont connus
    $y_j \leftarrow f_S^{-1}(u_j)$ ▷ Déchiffrement partiel du dernier tour
    if $E_j = 0$ **then**
      $\chi \leftarrow \chi + 1$
    end if
  end for
  $\varepsilon_k \leftarrow \left| \frac{\chi}{N} - \frac{1}{2} \right|$
  if $\varepsilon_k > \varepsilon_{\max}$ **then**
    $\varepsilon_{\max} \leftarrow \varepsilon$
    $k^* \leftarrow k$
  end if
end for
  return k^*

2.9 Conclusion

Dans ce chapitre, nous avons étudié le principe de la cryptanalyse linéaire et les différentes étapes de son application sur un SPN. Ces notions théoriques sont importantes pour comprendre la mise en œuvre pratique de cette attaque, qui sera présentée dans le prochain chapitre.

Chapitre 3

Implémentation et résultats

3.1 Introduction

Dans ce chapitre, nous mettons en œuvre la cryptanalyse linéaire étudiée précédemment sur un réseau SPN. Nous présentons le modèle de chiffrement, les étapes de l'attaque et les résultats obtenus à l'aide d'un programme Python.

3.2 Implémentation

3.2.1 Présentation générale de l'implémentation

L'implémentation a été réalisée en Python pour sa flexibilité et sa capacité à manipuler efficacement les calculs binaires et les structures de données. Le programme suit une méthodologie structurée, comprenant la construction de la table d'approximation linéaire (LAT), le calcul des biais, la sélection d'une approximation linéaire optimale, la génération de paires texte clair-texte chiffré, et l'estimation de la clé.

L'algorithme a été développé en suivant les étapes classiques de la cryptanalyse linéaire :

- Construction de la table des approximations linéaires (LAT) de la S-boîte.
- Sélection de l'approximation linéaire la plus biaisée.
- Génération de paires clair/chiffré aléatoires.
- Application de l'attaque linéaire pour estimer la clé.

3.2.2 Outils utilisés

L'implémentation de la cryptanalyse linéaire a été réalisée à l'aide de plusieurs outils et bibliothèques Python, sélectionnés pour leur efficacité dans le traitement des données, les calculs mathématiques et la visualisation des résultats. Voici une description détaillée des outils employés :

- **Python (version 3.x)** : Langage de programmation principal utilisé pour sa simplicité, sa flexibilité et son vaste écosystème de bibliothèques. Python a permis de coder les algorithmes de cryptanalyse, de manipuler les structures de données, et d'automatiser les calculs complexes nécessaires à la construction de la table d'approximations linéaires (LAT) et à l'exécution des attaques.

- **NumPy** : Bibliothèque essentielle pour les calculs numériques, utilisée pour manipuler des tableaux multidimensionnels (notamment la LAT de dimension 16×16). NumPy a permis d'optimiser les opérations matricielles et les calculs de biais, réduisant ainsi

le temps de traitement.

- **Random** : Module standard de Python utilisé pour générer des paires clair-chiffré aléatoires. La fonction `random.randint` a servi à simuler des textes clairs aléatoires sur 12 bits, garantissant une distribution uniforme pour les expériences d'attaque.

- **Math** : Module Python standard utilisé pour les calculs mathématiques de base, notamment ceux liés aux biais et à la formule du lemme d'empilement.

3.2.3 Objectif et description du SPN utilisé

L'objectif de ce travail est de mettre en œuvre une cryptanalyse linéaire sur un réseau de substitution-permutation (SPN) simplifié, afin d'évaluer l'efficacité de deux formules pour estimer le nombre de paires clair-chiffré nécessaires à une attaque réussie :

$$N_1 = \frac{4}{\varepsilon_t^2} \qquad \text{et} \qquad N_2 = \frac{16}{\varepsilon_t^2}$$

Cette analyse vise à identifier la bonne sous-clé d'un SPN à 2 tours, et à comparer la précision et la robustesse des deux formules dans un contexte pratique.
Le SPN utilisé est un modèle simplifié conçu pour rester proche des structures classiques de chiffrements par blocs, tout en permettant une étude académique accessible. Ses paramètres sont les suivants :

- **Taille des blocs** : 12 bits, divisés en trois nibbles de 4 bits, correspondant à trois S-boîtes par tour.

- **Nombre de tours** : 2 tours complets, un est utilisés pour propager des approximations linéaires, et le dernier pour le déchiffrement partiel .

- **S-boîte** : Une substitution non linéaire de 4 bits, définie comme suit :
  $[0, 4, 5, F, 1, 2, 7, E, 6, C, A, 8, 9, D, 3, B]$.

  Son inverse est automatiquement calculé pour le déchiffrement partiel.

- **P-boîte** : Permutation fixe des 12 bits, définie par la table :
  $[7, 0, 4, 8, 1, 5, 9, 2, 6, A, 3, B]$, assurant une diffusion efficace après chaque tour.

- **Clé** : Inialisation de 3 clés de 12 bits aléatoirement :$k_1 = [3,\ A,\ F]$, $k_2 = [E,\ 0,\ 2]$ et la clé réelle $k_3 = [4,\ C,\ B]$

- **Composants du chiffrement SPN** :Le réseau SPN utilisé dans cette implémentation est structuré comme suit : :

  (a) Il opère sur des blocs de 12 bits, divisés en trois sous-blocs de 4 bits, chacun traité par une S-boîte identique.

  (b) Une permutation fixe de 12 bits est appliquée à la sortie du premier tour pour mélanger les bits entre les sous-blocs.

  (c) Le chiffrement comporte deux tours complets (S-boîte + sous-clé + permutation), suivis d'un XOR final avec une dernière sous-clé.

3.3 Implementation de la méthodologie

Cette section présente les résultats obtenus lors de la cryptanalyse linéaire du réseau de substitution-permutation (SPN) à trois tours. Les données expérimentales incluent la construction de la table d'approximations linéaires (LAT), les biais calculés, l'application du lemme d'empilement, et les résultats des attaques pour les deux formules ($N_1 = \frac{8}{\varepsilon_t^2}$ et $N_2 = \frac{16}{\varepsilon_t^2}$). Les résultats sont organisés dans des tableaux pour une présentation claire et concise.

3.3.1 Construction de la table d'approximation linéaire (LAT)

La LAT a été générée a partie de programme suivant :

```
1                 def compute_lat_hex(sbox):
2                 n = 4  # 4 bits      16 entrées
3                 lat = {}
4
5                 for alpha in range(16):  # masques d'entrée     de 0x0 à 0xF
6                 for beta in range(16):  # masques de sortie     de 0x0 à 0xF
7                 count = 0
8                 for x in range(16):  # tous les x possibles sur 4 bits
9                 if parity(alpha & x) ^ parity(beta & sbox[x]) == 0:
10                count += 1
11                # Clé (  ,   ) en hexadécimal sous forme de chaînes : "A", "F", etc.
12                lat[(f"{alpha:X}", f"{beta:X}")] = count - 8
13                return lat
```

Listing 3.1 – Programme python de LAT [25]

Et les résultats sont comme suit :

$\alpha\backslash\beta$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	-2	0	-2	2	0	2	0	4	-2	0	2	-2	0	2	4
2	0	2	4	-2	0	-2	0	-2	2	4	-2	0	2	0	2	0
3	0	0	0	0	2	-2	6	2	-2	2	2	-2	0	0	0	0
4	0	4	2	2	-2	2	0	0	0	0	2	-2	-2	-2	0	4
5	0	2	-2	-4	0	2	-2	4	0	2	2	0	0	2	2	0
6	0	2	-2	4	-2	-4	0	2	2	0	0	2	0	2	2	0
7	0	0	-2	2	0	4	2	2	2	2	-4	0	2	-2	0	0
8	0	0	0	0	-2	2	2	-2	4	0	4	0	2	2	-2	-2
9	0	-2	4	2	0	2	0	2	0	-2	0	-2	0	2	4	-2
A	0	2	0	2	6	0	-2	0	2	0	2	0	0	-2	0	-2
B	0	0	0	0	0	0	0	0	-2	-2	2	2	6	-2	2	2
C	0	-4	2	2	0	0	-2	2	0	4	2	2	0	0	-2	2
D	0	2	2	0	2	0	0	2	0	-2	-2	0	2	4	-4	2
E	0	2	2	0	0	2	2	0	-2	0	0	6	-2	0	0	-2
F	0	0	-2	2	2	2	0	-4	-2	2	0	0	0	4	2	2

TABLE 3.1 – Table d'approximation linéaire (LAT)

3.3.2 Calcul des biais $\varepsilon(\alpha, \beta)$

À partir de la LAT précédente, nous calculons le biais associé à chaque approximation linéaire a partir de programme suivant :

```
def compute_bias_table(lat, n=4):
bias_table = {}
for (alpha, beta), val in lat.items():
bias = val / (2 ** n)  #     = LAT / 2^n
bias_table[(alpha, beta)] = bias
return bias_table
```

Listing 3.2 – Calcul du biais [25]

Le tableau suivant présente les biais pour chaque combinaison de masques d'entrée α et de sortie β :

$\alpha\backslash\beta$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0.500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	0.000	-0.125	0.000	-0.125	0.125	0.000	0.125	0.000	0.250	-0.125	0.000	0.125	-0.125	0.000	0.125	0.250
2	0.000	0.125	0.250	-0.125	0.000	-0.125	0.000	-0.125	0.125	0.250	-0.125	0.000	0.125	0.000	0.125	0.000
3	0.000	0.000	0.000	0.000	0.125	-0.125	0.375	0.125	-0.125	0.125	0.125	-0.125	0.000	0.000	0.000	0.000
4	0.000	0.250	0.125	0.125	-0.125	0.125	0.000	0.000	0.000	0.000	0.125	-0.125	-0.125	-0.125	0.000	0.250
5	0.000	0.125	-0.125	-0.250	0.000	0.125	-0.125	0.250	0.000	0.125	0.125	0.000	0.000	0.125	0.125	0.000
6	0.000	0.125	-0.125	0.250	-0.125	-0.250	0.000	0.125	0.125	0.000	0.000	0.125	0.000	0.125	0.125	0.000
7	0.000	0.000	-0.125	0.125	0.000	0.250	0.125	0.125	0.125	0.125	-0.250	0.000	0.125	-0.125	0.000	0.000
8	0.000	0.000	0.000	0.000	-0.125	0.125	0.125	-0.125	0.250	0.000	0.250	0.000	0.125	0.125	-0.125	-0.125
9	0.000	-0.125	0.250	0.125	0.000	0.125	0.000	0.125	0.000	-0.125	0.000	-0.125	0.000	0.125	0.250	-0.125
A	0.000	0.125	0.000	0.125	0.375	0.000	-0.125	0.000	0.125	0.000	0.125	0.000	0.000	-0.125	0.000	-0.125
B	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	-0.125	-0.125	0.125	0.125	0.375	-0.125	0.125	0.125
C	0.000	-0.250	0.125	0.125	0.000	0.000	-0.125	0.125	0.000	0.250	0.125	0.125	0.000	0.000	-0.125	0.125
D	0.000	0.125	0.125	0.000	0.125	0.000	0.000	0.125	0.000	-0.125	-0.125	0.000	0.125	0.250	-0.250	0.125
E	0.000	0.125	0.125	0.000	0.000	0.125	0.125	0.000	-0.125	0.000	0.000	0.375	-0.125	0.000	0.000	-0.125
F	0.000	0.000	-0.125	0.125	0.125	0.125	0.000	-0.250	-0.125	0.125	0.000	0.000	0.000	0.250	0.125	0.125

TABLE 3.2 – Table des biais

3.3.3 Approximations linéaires

Meilleures approximations linéaires

La table, de dimension 16×16, a permis d'identifier les approximations linéaires les plus significatives. Le tableau 3.3.3 présente les cinq meilleures approximations, triées par valeur absolue du biais $|\varepsilon(\alpha, \beta)|$.

α (hex)	β (hex)	$\lvert\varepsilon\rvert$	$P(\alpha, \beta)$
3	6	0.3750	0.8750
A	4	0.3750	0.8750
B	C	0.3750	0.8750
E	B	0.3750	0.8750
1	8	0.2500	0.7500

TABLE 3.3 – Meilleures approximations linéaires

Comme on a plusieurs approximations qui ont le mème maximum biais, nous avons choisi par defaut le couple $\alpha = E \ \ (1110), \beta = B \ \ (1011)$.

Approximations par tour

Pour l'attaque sur 2 tours :

- **Tour 1** :on a choisir par defaut la premiere S-boite , masques $\alpha = E$, $\beta = B$, biais $\varepsilon_1 = 0.375$.
- **Tour 2** : Déchiffrement partiel pour estimer la sous-clé.

L'équation d'approximation finale est :

$$x_0 \oplus x_1 \oplus x_2 \oplus y_1 \oplus y_7 \oplus y_{10} = 0.$$

où x représente les bits du texte clair et y les bits avant le dernier tour.

3.3.4 Application du lemme d'empilement

Dans notre attaque, une seule approximation linéaire est utilisée, concentrée sur une S-boîte active dans le premier tour. Par conséquent, aucune combinaison de biais n'a lieu, et le biais total se confond avec le biais individuel de l'approximation :

$$\varepsilon_{\mathrm{t}} = \varepsilon = 0{,}375$$

Dans notre exemple, le premier tour du SPN contient trois S-boîtes. Une seule est active avec un biais $\varepsilon = 0{,}375$, tandis que les deux autres ne sont pas actives, ce qui correspond à un biais de $\frac{1}{2}$ pour chacune. Ainsi, en appliquant le lemme du piling-up :

$$\varepsilon_{\mathrm{t}} = 2^{3-1} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot 0{,}375 = 0{,}375,$$

on retrouve exactement le biais de la S-boîte active. Cela confirme que lorsqu'une seule S-boîte est active, le biais total est égal au biais de cette approximation.

Remarque : Si le réseau SPN avait comporté trois tours ou plus avec plusieurs approximations, l'application du lemme aurait permis d'estimer un biais total réduit, ce qui aurait nécessité un nombre de paires clair/chiffré plus élevé pour conserver un bon taux de réussite dans l'estimation de la clé.

3.3.5 Génération des paires

Pour tester l'attaque, nous avons généré des paires (x_j, c_j), où xj est un texte clair aléatoire de 12 bits (représenté comme trois blocs de 4 bits) et c_j est le texte chiffré obtenu en utilisant la clé réelle $[4, C, B]$. La génération a été effectuée avec (`random.seed(42)`) pour garantir la reproductibilité :

```
random.seed(42)
N = 113
paires = []
for _ in range(N):
x = [random.randint(0, 15) for _ in range(3)]
c = spn_encrypt(x, keys)
paires.append((x, c))
```

Listing 3.3 – Génération de $N = 113$ paires clair/chiffré pour l'attaque [25]

Nous avons testé deux tailles d'échantillons : $N = 28$ (basé sur $4/\varepsilon_t^2$) et $N = 113$ (basé sur $16/\varepsilon_t^2$).

Chaque texte clair x est généré comme une liste de trois nombres hexadécimaux aléatoires (0 à 15), et le texte chiffré c est calculé à l'aide de la fonction `spn_encrypt`, qui applique les opérations de substitution, permutation et XOR avec les clés. Comme exprimé dans la figure suivant :

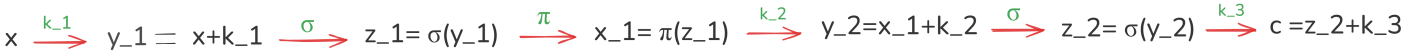

FIGURE 3.1 – structure de SPN de 2 tours

3.3.6 Estimation de la clé

Comme on a trouver Apres la permutation que il y a 3 s-boîte actiif donc l'estimation de la clé a été effectuée par une recherche exhaustive sur toutes les combinaisons possibles de la clé finale $(k^{(0)}, k^{(1)}, k^{(2)}) \in \{0, \ldots, 15\}^3$, soit $16^3 = 4096$ combinaisons. Pour chaque clé candidate, nous avons :

(a) Déchiffré partiellement le texte chiffré c en appliquant l'inverse de la S-boite après un XOR avec la clé candidate :

$$y_j = \left[S^{-1}(c_j^{(0)} \oplus k^{(0)}),\ S^{-1}(c_j^{(1)} \oplus k^{(1)}),\ S^{-1}(c_j^{(2)} \oplus k^{(2)})\right]$$

(b) Converti x et y en représentations binaires pour extraire les bits pertinents $(x_0, x_1, x_2, y_1, y_7, y_A)$.

(c) Vérifié si l'équation linéaire $x_0 \oplus x_1 \oplus x_2 \oplus y_1 \oplus y_7 \oplus y_A = 0$ est satisfaite.

(d) Compté le nombre de paires (χ_i) satisfaisant l'équation et calculé le biais expérimental :

$$\varepsilon_i = \left|\frac{\chi_i}{N} - 0.5\right|$$

La clé candidate avec le biais expérimental maximal est retenue comme estimation de la clé réelle.

```
# === Estimation de la clé ===
max_eps = 0
cle_estimee = None
resultats = []

for k0, k1, k2 in product(range(16), repeat=3):
chi = 0
```

```
                for x_hex, c_hex in paires:
                x_bin = hex_block_to_binlist(x_hex)
                c_xor_k = [c_hex[0] ^ k0, c_hex[1] ^ k1, c_hex[2] ^ k2]
                y = [SBOX_INV[b] for b in c_xor_k]
                y_bin = hex_block_to_binlist(y)
                bits = [x_bin[0], x_bin[1], x_bin[2], y_bin[1], y_bin[7], y_bin[10]]
                eq = bits[0] ^ bits[1] ^ bits[2] ^ bits[3] ^ bits[4] ^ bits[5]
                if eq == 0:
                chi += 1
                eps = abs(chi / N - 0.5)
                resultats.append(((k0, k1, k2), chi, eps))
                if eps > max_eps:
                max_eps = eps
                cle_estimee = (k0, k1, k2)
```

Listing 3.4 – Estimation de la clé finale par recherche du biais maximal [25]

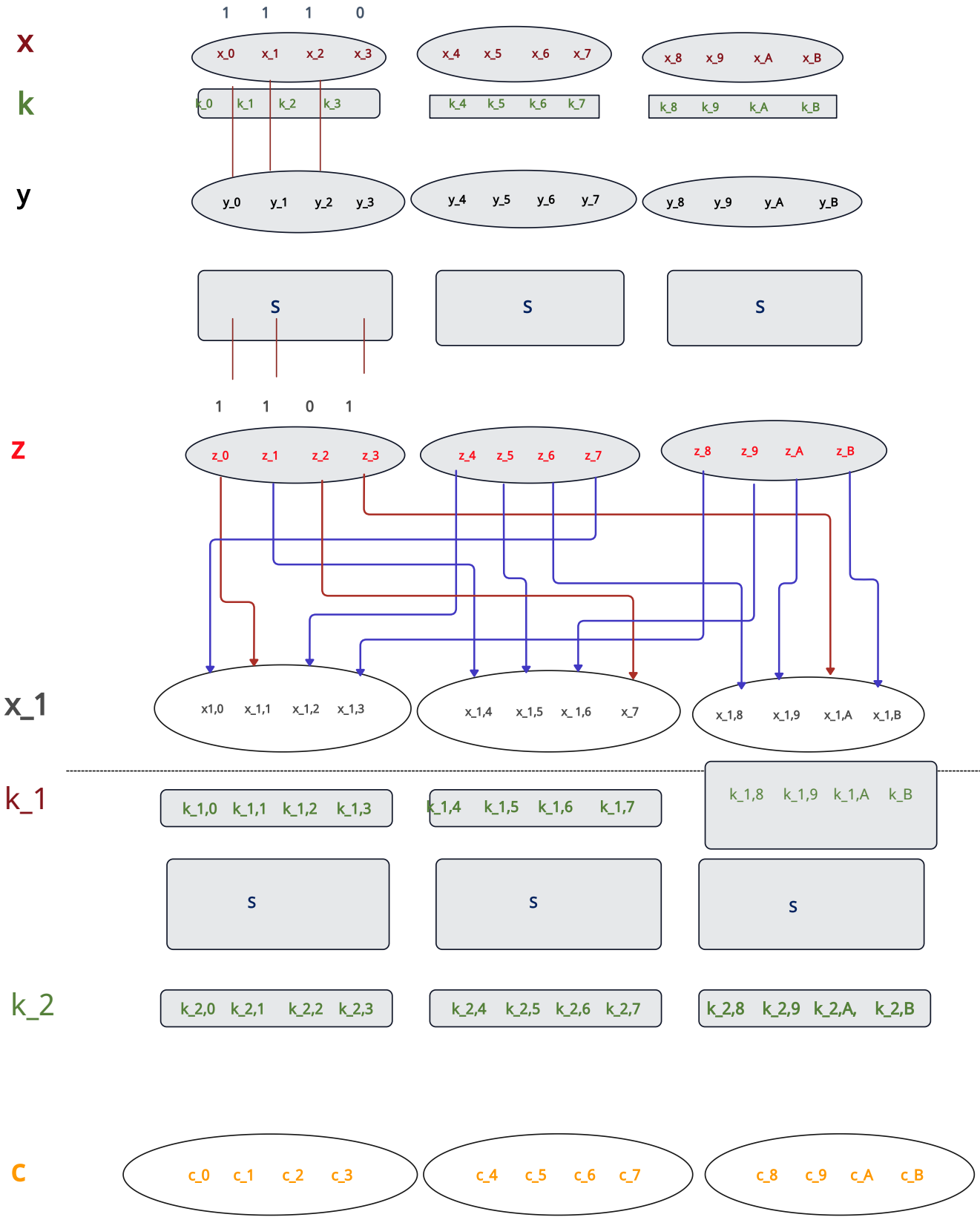

FIGURE 3.2 – Schéma de l'attaque linéaire appliquée au SPN implémenté

3.4 Résultats des attaques

Dans un premier temps, nous avons testé l'estimation de la clé finale en générant $N = 28$ paires clair/chiffré, conformément à la formule $N = \frac{4}{\varepsilon^2}$. Les résultats pour quelques clés candidates sont indiqués dans le tableau ci-dessous. On rappelle que la **clé réelle utilisée pour le chiffrement était $\mathbf{k_3} = (\mathbf{4}, \mathbf{C}, \mathbf{B})$**.

Clé testée	χ (sur 28)	ε
(0, 0, 0)	7	0,2000
(0, 0, 1)	5	0,0000
(0, 0, 2)	3	0,2000
(0, 0, 3)	4	0,1000
(0, 0, 4)	7	0,2000
(0, 0, 5)	5	0,0000
(0, 0, 6)	3	0,2000
(0, 0, 7)	2	0,3000
(0, 0, 8)	7	0,2000
(0, 0, 9)	5	0,0000
(0, 0, A)	4	0,1000
(0, 0, B)	5	0,0000
(0, 0, C)	7	0,2000
(0, 0, D)	5	0,0000
(0, 0, E)	6	0,1000
Clé réelle	—	(4, C, B)
Clé estimée	$\chi = 0$	(0, 9, F)
Estimation correcte ?	—	`FAUX`

TABLE 3.4 – Estimation de la clé avec $N = 28$

Dans un second temps, nous avons testé la formule plus stricte $N = \frac{16}{\varepsilon^2}$, qui donne $N = 113$ paires clair/chiffré.

Clé testée	χ (sur 114)	ε
(0, 0, 0)	49	0,0702
(0, 0, 1)	57	0,0000
(0, 0, 2)	53	0,0351
(0, 0, 3)	59	0,0175
(0, 0, 4)	54	0,0263
(0, 0, 5)	52	0,0439
(0, 0, 6)	54	0,0263
(0, 0, 7)	58	0,0088
(0, 0, 8)	53	0,0351
(0, 0, 9)	57	0,0000
(0, 0, A)	63	0,0526
(0, 0, B)	65	0,0702
(0, 0, C)	58	0,0088
(0, 0, D)	52	0,0439
(0, 0, E)	64	0,0614
(4, C, B)	12	0,3947
Clé réelle	—	(4, C, B)
Clé estimée	$\chi = 12$	(4, C, B)
Estimation correcte ?	—	`VRAI`

TABLE 3.5 – Estimation de la clé avec $N = 114$

3.5 Comparaison des résultats

Cette section compare les résultats des deux attaques réalisées avec deux formules différentes pour estimer le nombre de paires clair/chiffré nécessaires. Ces deux attaques visaient à retrouver la clé réelle de dernier tour $k_3 = (4, C, B)$ utilisée dans le chiffrement.

TABLE 3.6 – Comparaison des résultats

Critère	Formule 1 ($N = \frac{4}{\varepsilon_t^2}$)	Formule 2 ($N = \frac{16}{\varepsilon_t^2}$)
Biais utilisé ε	0.375	0.375
Nombre de paires N	28	113
Clé réelle k_3	(4, C, B)	(4, C, B)
Clé estimée k_3	(0, 9, F)	(4, C, B)
Biais ε_i trouvé	0.500	0.417
Résultat	**Échec**	**Succès**

Remarque : La clé estimée est obtenue après avoir exploré l'intégralité des $2^{12} = 4096$ combinaisons possibles de clés partielles (k_0, k_1, k_2). Les lignes affichées dans les tableaux précédents ne présentent qu'un extrait illustratif des premières combinaisons testées.

3.5.1 Analyse de la précision

L'attaque basée sur la formule $N = \frac{4}{\varepsilon_t^2}$ s'est révélée insuffisante pour extraire la bonne clé. Bien que certaines combinaisons aient présenté des biais non nuls, la distribution des résultats n'était pas suffisamment marquée pour identifier correctement la clé $(4, C, B)$. La clé estimée dans ce cas — $(0, 9, F)$ — correspondait à un biais élevé mais ne reflétait pas la clé réelle utilisée dans le chiffrement.
En revanche, l'attaque avec $N = \frac{16}{\varepsilon_t^2}$ a permis de retrouver précisément la clé finale. Le biais calculé pour cette clé était supérieur à celui des autres combinaisons, ce qui a facilité sa détection parmi les 4096 possibilités.

3.5.2 Convergence vers la clé réelle

Ces résultats montrent que la fiabilité d'une attaque linéaire dépend fortement du volume de données disponibles. Lorsque N est trop faible, les biais observés pour différentes clés peuvent être trop proches, entraînant des erreurs d'identification. À l'inverse, une valeur de N suffisamment grande permet de mieux séparer statistiquement les clés candidates et d'assurer une convergence correcte vers la clé réelle.

3.6 Interprétation et discussion

Les expériences menées sur le réseau SPN mettent en évidence plusieurs points importants :

- **Impact de la taille de l'échantillon** : Une valeur de N trop faible (comme pour $c = 4$) mène souvent à une estimation incorrecte, tandis qu'un nombre plus important de paires (comme pour $c = 16$) améliore significativement la précision de l'attaque.
- **Clé estimée vs clé réelle** : Seule l'approche avec $c = 16$ a permis d'estimer correctement la clé réelle k_3. Cela démontre l'importance de calibrer correctement le paramètre c dans la formule $N = \frac{c}{\varepsilon_t^2}$ selon la précision attendue.
- **Perspectives futures** : Des améliorations pourraient consister à combiner plusieurs approximations linéaires, à tester d'autres caractéristiques actives ou à étendre le SPN à un nombre de tours plus élevé pour observer l'évolution des biais dans un contexte plus complexe.

3.7 Conclusion

Dans ce chapitre, nous avons implémenté une attaque par cryptanalyse linéaire sur un réseau SPN. Les résultats obtenus montrent que cette méthode permet de retrouver une partie de la clé grâce à l'exploitation des approximations linéaires.

Conclusion générale

Ce mémoire a porté sur l'étude de la cryptanalyse linéaire appliquée à un réseau de chiffrement par substitution-permutation (SPN) , dans le but de mieux comprendre les principes de cette attaque et d'en évaluer l'efficacité dans un cadre contrôlé. Pour cela, nous avons d'abord construit un réseau SPN composé de deux tours, de trois boîtes de substitution identiques (S-boîtes), d'une boîte de permutation (P-boîte), et d'un schéma de l'attaque linéaire appliquée ce SPN implémenté . Ce réseau a servi de base pour mettre en œuvre toutes les étapes d'une attaque linéaire.

Dans la partie théorique, nous avons étudié les notions essentielles de la cryptanalyse linéaire : les approximations linéaires, les biais, la table d'approximation linéaire (LAT), ainsi que le lemme d'empilement, qui permet de combiner les biais sur plusieurs tours. Dans la partie pratique, nous avons implémenté l'ensemble de l'attaque en Python : construction de la LAT, sélection de l'approximation convenable, calcul du biais total, génération automatique des paires clair-chiffré, et estimation de la clé finale.

Ce travail confirme l'intérêt de la cryptanalyse linéaire comme outil d'évaluation de la sécurité des chiffrements par blocs, mais souligne également ses limites pratiques : elle reste sensible à la qualité des biais exploités et nécessite un grand nombre de données pour être efficace.

À la suite de cette étude, plusieurs développements sont envisageables. Il serait par exemple pertinent d'appliquer la cryptanalyse linéaire à des chiffrements plus complexes, tels que l'AES, afin d'évaluer leur niveau de résistance. L'utilisation conjointe de plusieurs approximations linéaires pourrait également permettre d'améliorer la précision de l'attaque. Enfin, l'analyse de l'influence du nombre de tours sur la sécurité d'un réseau SPN offrirait une meilleure compréhension de sa robustesse. Ces prolongements contribueraient à renforcer l'évaluation des algorithmes actuels et à améliorer la conception de systèmes de chiffrement plus sûrs.

Annexes

Valeurs associées aux lettres dans $\mathbb{Z}_{26}$

Dans le cadre du chiffrement de César, chaque lettre majuscule de l'alphabet est associée à une valeur entière dans l'ensemble $\mathbb{Z}_{26}$ (allant de 0 à 25). Le tableau suivant présente cette correspondance :

Lettre	A	B	C	D	E	F	G	H	I	J	K	L	M
Valeur	0	1	2	3	4	5	6	7	8	9	10	11	12

Lettre	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Valeur	13	14	15	16	17	18	19	20	21	22	23	24	25

TABLE 7 – Valeurs associées aux lettres de l'alphabet dans $\mathbb{Z}_{26}$

Bibliographie

[1] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer, 1993.

[2] A. Biryukov, C. De Cannière, and M. Quisquater, "On multiple linear approximations," *Advances in Cryptology—CRYPTO'04*, LNCS, vol. 3152, pp. 1–18, Springer, 2004.

[3] J.Y. Cho, "Linear cryptanalysis of reduced-round PRESENT," *CT-RSA 2010*, LNCS, vol. 5985, pp. 302–317, Springer, 2010.

[4] B. Collard, F.-X. Standaert, and J.-J. Quisquater, "Improved and multiple linear cryptanalysis of reduced round Serpent," *FSE 2007*, LNCS, vol. 4593, pp. 358–375, Springer, 2007.

[5] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," *IBM Journal of Research and Development*, vol. 38, no. 3, pp. 243–250, 1994.

[6] R. Djabri, "Cryptography Course," Master 1 in Data Science, University of Bejaia, Academic Year 2023–2024.

[7] C. Harpes, G.G. Kramer, and J.L. Massey, "A generalization of linear cryptanalysis and the applicability to DES," *EUROCRYPT'94*, LNCS, vol. 950, pp. 1–17, Springer, 1995.

[8] H. M. Heys, "A Tutorial on the Implementation of Block Ciphers : Software and Hardware Applications," Memorial University of Newfoundland, Dec. 2020.

[9] T. Jakobsen and L.R. Knudsen, "The interpolation attack on block ciphers," *FSE'97*, LNCS, vol. 1267, pp. 28–40, Springer, 1997.

[10] A. Joux, *Algorithmic Cryptanalysis*, CRC Press, 2009.

[11] P. Junod, "On the complexity of Matsui's attack," *SAC 2001*, LNCS, vol. 2259, pp. 199–211, Springer, 2001.

[12] B.S. Kaliski Jr., R.L. Rivest, and A.T. Sherman, "Is the Data Encryption Standard a group ?" *Journal of Cryptology*, vol. 4, no. 1, pp. 57–70, 1991.

[13] L.R. Knudsen and M.J.B. Robshaw, "Non-linear approximations in linear cryptanalysis," *EUROCRYPT'98*, LNCS, vol. 1403, pp. 224–236, Springer, 1998.

[14] S.K. Langford and M.E. Hellman, "Differential-linear cryptanalysis," *CRYPTO'94*, LNCS, vol. 839, pp. 17–25, Springer, 1994.

[15] M. Matsui, "Linear cryptanalysis method for DES cipher," *EUROCRYPT'93*, LNCS, vol. 765, pp. 386–397, Springer, 1994.

[16] M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard," *CRYPTO'93*, LNCS, vol. 773, pp. 1–11, Springer, 1993.

[17] S. Murphy and M.J.B. Robshaw, "Essential algebraic structure within the AES," *CRYPTO'02*, LNCS, vol. 2442, pp. 1–16, Springer, 2002.

[18] National Institute of Standards and Technology, "Data Encryption Standard (DES)," FIPS Publication 46, 1977.

[19] National Institute of Standards and Technology, "DES modes of operation," FIPS Publication 81, 1980.

[20] K. Nyberg, "Linear approximation of block ciphers," *EUROCRYPT'94*, LNCS, vol. 950, pp. 439–444, Springer, 1995.

[21] C. Paar and J. Pelzl, *Understanding Cryptography : A Textbook for Students and Practitioners*, Springer, 2010.

[22] B. Schneier, *Applied Cryptography : Protocols, Algorithms, and Source Code in C*, 2nd ed., Wiley, 1996.

[23] T. Shimoyama and T. Kaneko, "Quadratic relation of S-box and its application to the linear attack of full round DES," *CRYPTO'98*, LNCS, vol. 1462, pp. 200–211, Springer, 1998.

[24] D.R. Stinson, *Cryptography : Theory and Practice*, 3rd ed., Chapman and Hall/CRC, 2005.

[25] C. Swenson, *Modern Cryptanalysis : Techniques for Advanced Code Breaking*, 1st ed., Addison-Wesley, 2022.

[26] SageMath, *Cryptography Module Documentation, Release 10.6*, The Sage Development Team, 2025.

Résumé

Ce mémoire présente une étude théorique et expérimentale de la cryptanalyse linéaire appliquée à un réseau SPN . Dans un premier temps, nous avons construit un SPN à deux tours, en définissant des S-boîtes ainsi qu'une P-boîte. Nous avons ensuite étudié les concepts fondamentaux de la cryptanalyse linéaire, notamment les approximations linéaires, le calcul des biais et le lemme d'empilement. À l'aide du langage Python, nous avons mis en œuvre une attaque linéaire visant à retrouver la sous-clé du dernier tour. Enfin, les résultats obtenus ont montré que l'attaque permet d'estimer correctement la clé, à condition de disposer d'un volume de données suffisant.

Abstract

This thesis presents a theoretical and experimental study of linear cryptanalysis applied to an SPN. First, we designed a two-round SPN, defining the S-boxes and a P-box. We then explored the fundamental concepts of linear cryptanalysis, including linear approximations, bias computation, and the piling-up lemma. Using the Python programming language, we implemented a linear attack aimed at recovering the last round subkey. Finally, the results showed that the attack can accurately estimate the key, provided that a sufficiently large amount of data is available.