Université de Versailles
Saint-Quentin-en-Yvelines

Rapport de projet

Attaque par fautes sur le DES

Alexandre Mihet

Encadrant : Louis Goubin

Date de soumission : 11 octobre 2024

Préambule

Ce rapport présente le travail réalisé dans le cadre d'un projet d'une attaque dite "par fautes" appliquée au cryptosystème DES. Le projet a été mené sous la supervision de Louis Goubin, et s'inscrit dans le contexte du Master SeCReTS.

L'attaque par fautes est une forme d'attaque visant à compromettre un système cryptographique en exploitant les vulnérabilités physiques.

Dans le cadre de ce projet, j'ai choisi d'utiliser le langage de programmation **Python** étant donné que je le maîtrise bien. Il ne s'agit cependant pas du langage le plus rapide et le plus optimisé, mais pour ce projet il convient bien étant donné qu'il n'y a pas besoin d'avoir une puissance de calcul énorme.

Table des matières

Introduction

Le chiffrement des données constitue un pilier fondamental de la sécurité informatique, visant à protéger les informations sensibles contre les regards indiscrets. Dans ce contexte, le **D**ata **E**ncryption **S**tandard (**DES**) a longtemps été un algorithme de référence pour assurer la sécurité. Cependant, avec l'évolution des technologies et des techniques d'attaque, sa robustesse a été mise à l'épreuve. Parmi ces techniques, l'attaque par fautes s'est avérée être une menace très importante exploitant les failles physiques des dispositifs de chiffrement.

Dans cette introduction, nous éclaircissons d'abord les principes du DES, avant de plonger dans les subtilités des attaques par fautes.

0.1 Explication du DES

0.1.1 Historique

En 1973, le National Bureau of Standards des États-Unis a demandé la création d'un algorithme de chiffrement pour le milieu des affaires. IBM, la célèbre multinationale américaine a proposé son algorithme appelé "Lucifer", créé 2 ans plus tôt (en 1971) par Horst Feistel, le créateur du fameux schéma de Feistel.

Le cryptosystème Lucifer n'a pas entièrement convaincu et de ce fait, la NSA a demandé de faire certaines modifications à cet algorithme. Le DES est donc créé à partir de Lucifer et devient un standard de chiffrement en 1977.
L'article du DES a été publié par FIPS sous le nom "FIPS PUB 46".

0.1.2 Structure

Le DES est un chiffrement par blocs : les **blocs font 64 bits** et la longueur de la **clé fait 56 bits**.

L'algorithme est basé sur le réseau de Feistel, il y a en tout **16 tours**. Voici un schéma de l'algorithme DES en abrégé :

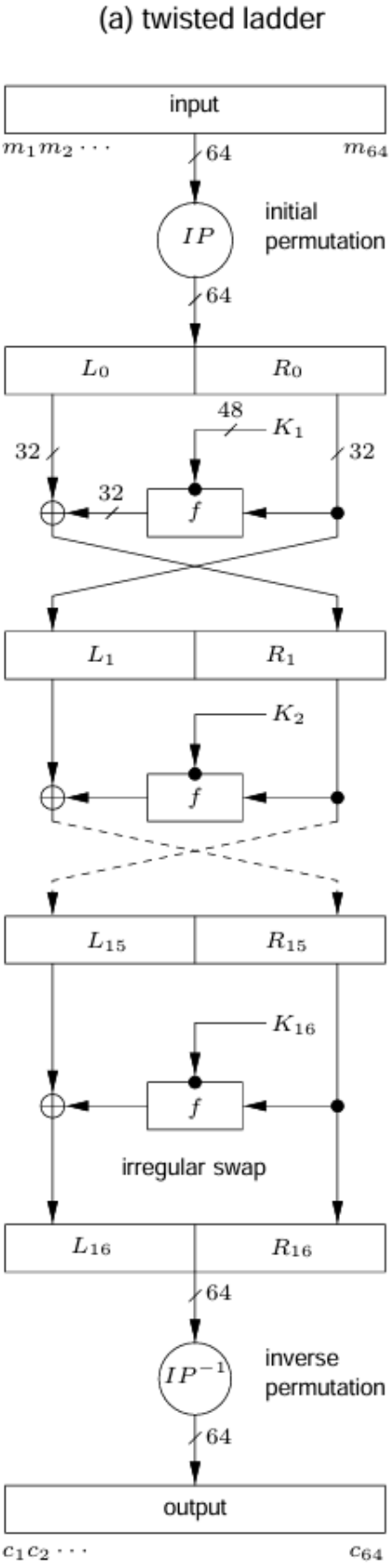

FIGURE 1 – Schéma du DES

Il y a beaucoup de choses à dire sur ce schéma, mais dans le contexte de ce projet,

nous allons nous focaliser sur la fonction f ainsi que sur les permutations au début et à la fin du chiffrement.
Voici le schéma de la fonction f :

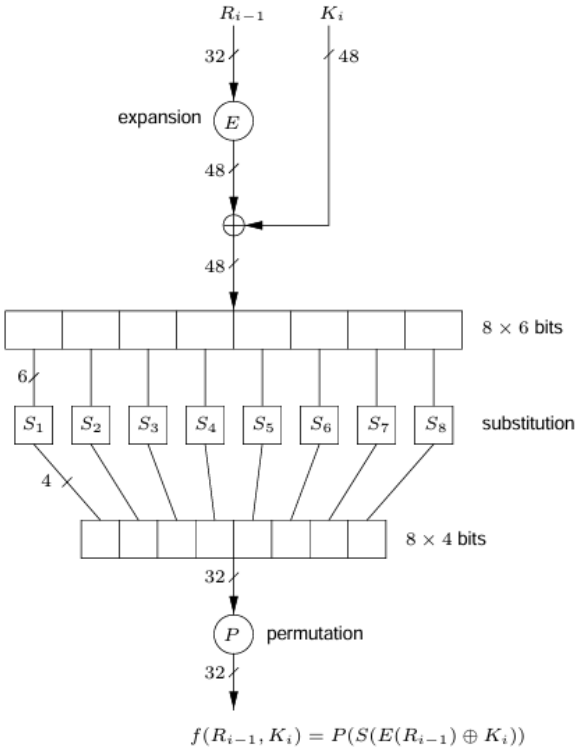

FIGURE 2 – Fonction f du DES

Nous allons nous focaliser plus en détail sur les caractéristiques de cette fonction dans les chapitres suivants.

0.1.3 Le DES de nos jours

Plusieurs attaques ont été développées à la suite de la parution du DES :
- En 1992, une attaque théorique est trouvée avec la cryptanalyse différentielle.
- En 1994, la première attaque pratique avec la récupération de la clé à l'aide de la cryptanalyse linéaire.
- En 1999, Deep Crack, une machine spécialisée dans le cassage du DES, arrive à retrouver une clé en moins de 24 heures.
- En 2004, le standard est officiellement abandonné.

Le problème majeur est la taille de la clé, 56 bits est bien trop court et comme l'avait énoncé Moore, la puissance des ordinateurs à travers les années s'est décuplée énormément et aujourd'hui une attaque exhaustive fonctionne.

Dans le présent projet, nous n'allons pas faire une recherche exhaustive mais nous allons voir comment une attaque par fautes permet de retrouver la clé du DES plus efficacement.

0.2 Explication des attaques par fautes

Les attaques "par faute" sont une catégorie d'attaques non invasives par canal auxiliaire qui exploitent les vulnérabilités physiques des dispositifs de chiffrement pour compromettre leur sécurité.

Plutôt que de se concentrer sur la faiblesse algorithmique, ces attaques ciblent les implémentations réelles des algorithmes, cherchant à perturber leur fonctionnement normal en introduisant délibérément des erreurs, ou "fautes", dans le processus de chiffrement. En manipulant ces fautes, les attaquants peuvent obtenir des informations sensibles sur la clé ou sur le chiffré.

Chapitre 1

Question 1 : Description de l'attaque par fautes sur le DES

1.1 Injection de faute

Comme expliqué précédemment, nous allons faire une attaque par faute sur le DES. Dans un premier temps, il faut savoir où exactement la faute sera injectée. Dans notre cas, nous allons effectuer la faute sur la valeur de sortie R_{15} du 15ème tour. Nous allons voir comment l'introduction de cette faute nous permet d'obtenir des informations sur la clé d'une manière plus rapide que la recherche exhaustive.
Voici ce que cela donne sur un schéma :

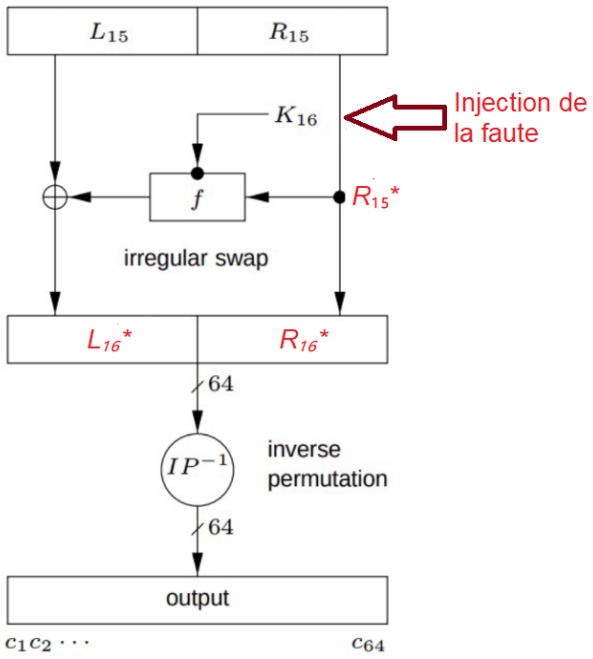

FIGURE 1.1 – Schéma injection de la faute

Concrètement, cela veut dire que l'attaquant a pu faire en sorte que la sortie de R_{15} ait une faute. Nous notons R^*_{15} la sortie faussée de R_{15}. La faute se propage sur R_{16} et L_{16} donnant R^*_{16} et L^*_{16}.

Mettons-nous dans la peau de l'attaquant et regardons désormais les équations des résultats des membres fautés et non fautés :

1. **Membres sans faute :**
   — $L_{16} = L_{15} \oplus f(K_{16}, R_{15})$

   — $R_{16} = R_{15}$

2. **Membres avec faute :**
   — $L^*_{16} = L_{15} \oplus f(K_{16}, R^*_{15})$

   — $R^*_{16} = R^*_{15}$

L'objectif ici est de retrouver la clé K_{16}. Pour cela, nous pouvons remarquer que K_{16} est présent à la fois dans les équations de L_{16} et L^*_{16}. Pour pouvoir éliminer le terme qui est présent dans les deux équations L_{15}, on va effectuer l'opération XOR entre les équations de L^*_{16} et L_{16}. On obtient dans ce cas l'équation suivante :

$$L_{16} \oplus L^*_{16} = L_{15} \oplus L_{15} \oplus f(K_{16}, R_{15}) \oplus f(K_{16}, R^*_{15})$$

Ce qui donne étant donné que $L_{15} \oplus L_{15} = 0$:

$$L_{16} \oplus L^*_{16} = f(K_{16}, R_{15}) \oplus f(K_{16}, R^*_{15})$$

1.2 Fonction f

Désormais, nous avons besoin de comprendre plus en détail le fonctionnement de la fonction f décrite plus haut dans le schéma. Nous allons maintenant écrire les équations en y ajoutant les détails de la fonction f :

$f(K_{16}, R_{15}) = P(S(E(R_{15}) \oplus K_{16}))$ -> Sans faute
$f(K_{16}, R^*_{15}) = P(S(E(R^*_{15}) \oplus K_{16}))$ -> Avec faute

Où P est la permutation, S les différentes boîtes-S et E la fonction d'expansion. Explicitons ces derniers :

Dans un premier temps, la fonction f prend en entrée R_{15} qui fait 32 bits et lui applique une fonction d'expansion E que nous n'allons pas détailler ici mais qui permet d'avoir une sortie de 48 bits. Cela permet que la sortie soit de la même taille que la clé et de faire en sorte qu'on puisse appliquer l'opération XOR entre R_{15} et K_{16}.

Maintenant que nous avons une sortie de 48 bits, nous allons la répartir sur 8 boîtes-S (S-Box) ce qui nous donne 8 S-Box de 6 bits chacune. La particularité des S-Box est que la sortie fait 4 bits. Au total, les 8 S-Box nous renvoient donc, après concaténation des résultats, une sortie de 32 bits. C'est ici la partie importante de l'attaque : Les S-Box. Nous y reviendrons un tout petit peu plus tard.

Au fait, à cause des S-Box, les équations sont un peu plus complexes que celles citées précédemment. En prenant en compte le détail des S-Box, voici ce que ça donne :

Sans faute :

$$f(K_{16}, R_{15}) \;=\; P\ (S_1(E(R_{15}) \oplus K_{16})_{\text{ bits 1 à 6}} || \ ... \ || S_8(E(R_{15}) \oplus K_{16})_{\text{ bits 43 à 48}})$$

Avec faute :

$$f(K_{16}, R^*_{15}) \;=\; P\ (S_1(E(R^*_{15}) \oplus K_{16})_{\text{ bits 1 à 6}} || \ ... \ || S_8(E(R^*_{15}) \oplus K_{16})_{\text{ bits 43 à 48}})$$

Où S_1 ... S_8 sont les 8 S-Box

1.3 Permutation P

Enfin, une permutation P est appliquée au résultat de 32 bits qui a une sortie de la même taille. Cette permutation est toujours la même, c'est-à-dire qu'elle vient avec la documentation du DES. Ce qu'on désire faire est de "remonter" le schéma pour attaquer les S-BOX, on connaît L_{16} et L^*_{16} . Pour pouvoir se débarrasser de cette permutation, il faut simplement appliquer la permutation inverse P^{-1}.

Ce qui est également intéressant de noter est que les permutations ont une propriété particulière qui est que pour toutes les permutations P, on a :
$P(\alpha \oplus \beta) = P(\alpha) \oplus P(\beta)$.
Ce qui nous permet de faire :

$$P^{-1}(L_{16} \oplus L^*_{16}) = P^{-1}(P(Sbox(E(R_{15}) \oplus K_{16}))) \oplus P^{-1}(P(Sbox(E(R^*_{15}) \oplus K_{16})))$$

$$P^{-1}(L_{16} \oplus L^*_{16}) = Sbox(E(R_{15}) \oplus K_{16}) \oplus Sbox(E(R^*_{15}) \oplus K_{16})$$

Rappelons qu'il y a 8 S-Box donc il faut découper cette équation en 8 "sous-équations" : Pour les bits 1 à 4, de 5 à 8, ainsi de suite jusqu'aux bits de 29 à 32 :

$$\begin{cases} P^{-1}(L_{16} \oplus L^*_{16}) = S_1(E(R_{15}) \oplus K_{16}) \oplus S_1(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 1 à 4} \\ P^{-1}(L_{16} \oplus L^*_{16}) = S_2(E(R_{15}) \oplus K_{16}) \oplus S_2(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 5 à 8} \\ P^{-1}(L_{16} \oplus L^*_{16}) = S_3(E(R_{15}) \oplus K_{16}) \oplus S_3(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 9 à 12} \\ P^{-1}(L_{16} \oplus L^*_{16}) = S_4(E(R_{15}) \oplus K_{16}) \oplus S_4(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 13 à 16} \\ P^{-1}(L_{16} \oplus L^*_{16}) = S_5(E(R_{15}) \oplus K_{16}) \oplus S_5(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 17 à 20} \\ P^{-1}(L_{16} \oplus L^*_{16}) = S_6(E(R_{15}) \oplus K_{16}) \oplus S_6(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 21 à 24} \\ P^{-1}(L_{16} \oplus L^*_{16}) = S_7(E(R_{15}) \oplus K_{16}) \oplus S_7(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 25 à 28} \\ P^{-1}(L_{16} \oplus L^*_{16}) = S_8(E(R_{15}) \oplus K_{16}) \oplus S_8(E(R^*_{15}) \oplus K_{16}), & \text{pour les bits 29 à 32} \end{cases}$$

1.4 Attaque des S-Box

Nous allons maintenant expliciter l'attaque exhaustive sur les S-Box. Pour cela, nous allons partir du système d'équations précédemment énoncé et nous allons aller de gauche à droite : L_{16} et L_{16}^* sont connus. On fait un XOR entre les deux et on applique un P^{-1}. Ce qui nous fait obtenir un résultat de 32 bits (8 blocs de 4 bits) qu'on va mettre de côté car ce n'est pas dépendant des S-Box.

D'un autre côté, nous allons prendre R_{15} et R_{15}^*, et on leur applique la fonction d'expansion E ce qui nous fait obtenir une sortie de 48 bits également qu'on va découper en 8 blocs de 6 bits.

Maintenant, nous allons prendre K_{16} mais seulement un bout de 6 bits pour chaque itération et l'incrémentons de 1 à chaque fois pour tester toutes les possibilités (000000, 000001, ..., 111111). À chaque itération, on va XOR ce bout de K_{16} avec un bout du résultat de $E(R_{15})$ que nous passons dans toutes les S-box une par une. On fait le même traitement pour $E(R_{15}^*)$.

Cela va nous donner deux résultats. Après le passage dans chaque S-Box, le résultat sera de taille 4 (la même taille qu'à gauche de l'égalité). On compare les deux uniquement dans les cas différents de 0000 (les cas qui n'ont pas été impactés par la faute donc qui ne nous apportent pas d'informations). À chaque égalité qu'on trouve, on stocke les bouts de clés utilisés pour l'itération. On fait ça sur tous les 32 bits et pour tous les chiffrés.

Par exemple, si la S-Box 1 a envoyé un résultat valide, on stocke les 6 bits de clés en position 1 et ainsi de suite et on obtient K_{16}.

1.5 Complexité de l'attaque

Cela nous permet de construire un système d'équations (8 équations) et la seule inconnue de ces équations sont les 6 bits de K_{16}. Il nous reste donc à faire une recherche exhaustive sur toutes les valeurs possibles de chacune des S-Box qui font 6 bits chacune, ce qui nous donne 2^6 opérations pour chaque S-Box. Étant donné qu'il y a en tout 8 S-Box, nous avons 8×2^6 ce qui au total nous donne 2^9 opérations.
Il ne faut pas oublier que nous avons en tout 32 (2^5) chiffrés fautés.
Au total, l'attaque pour trouver K_{16} a pour complexité $O(2^{9+5}) = O(2^{14})$

Nous allons voir comment ceci est possible en pratique dans les questions suivantes.

Chapitre 2

Question 2 : Application concrète de l'attaque

2.1 Description du protocole

Dans la partie précédente, nous avons vu qu'un attaquant pouvait trouver 8 équations, une par S-Box, chacune révélant 6 bits de K_{16}. Nous allons désormais voir comment cela a été implémenté et retrouvé en pratique.

Pour effectuer cette attaque par faute, nous disposons de :

— 1 chiffré juste

— 32 chiffrés faux

Ces 33 chiffrés découlent d'un seul et même message et d'une seule et même clé.

Pour effectuer cette attaque, nous allons suivre les différentes étapes expliquées dans le premier chapitre pas à pas. Le code est divisé en plusieurs parties, chacune ayant son importance pour effectuer l'attaque. Voici ces parties :

2.1.1 Fonctions utilitaires

Dans un premier temps, j'ai défini quelques fonctions qui me seront utiles par la suite. Ces fonctions incluent :

— **Conversion entre binaire et hexadécimal** : Ces fonctions permettent de convertir des chaînes hexadécimales en chaînes binaires de 64 bits et vice versa.

— **Opération XOR** : Cette fonction effectue une opération XOR bit à bit entre deux chaînes de bits.

— **Manipulation des bits et octets** : Des fonctions pour incrémenter des chaînes de bits et convertir des chaînes de bits en séquences d'octets.

2.1.2 Fonctions du DES

Dans un second temps, j'ai implémenté les fonctions natives du DES. Cela inclut :

— **Permutations initiale (IP) et inverse** : Ces fonctions appliquent la permutation initiale définie dans le DES et son inverse.

— **Expansion** E : Cette fonction étend une chaîne de 32 bits en une chaîne de 48 bits en appliquant la table d'expansion.

— **Permutation** P **et inverse** : Ces fonctions appliquent la permutation P définie dans le DES et son inverse.

— **S-Box** : Cette fonction applique la substitution définie par une des 8 S-Box du DES.

2.1.3 Fonction principale de l'attaque

La fonction principale implémente l'attaque par faute sur DES pour récupérer les clés fautives. Voici les étapes principales de cette fonction :

— Conversion des textes chiffrés fautifs et du texte chiffré correct en binaire.
— Application de la permutation initiale pour obtenir les bits R_{16} et L_{16}.
— Calcul de la faute en effectuant un XOR entre les valeurs correctes et fautives de L_{16}.
— Utilisation de l'expansion E et des S-Box pour essayer toutes les combinaisons possibles de K_{16} et déterminer les bits corrects.

Il suffit juste d'appliquer la théorie vue précédemment en python. Bien que ce ne soit peut-être pas le langage de programmation le plus rapide, avec ses types de base comme les listes et les dictionnaires, il est simple d'utilisation. Dans le cadre de cette attaque, c'est suffisant.

2.2 Trouver les 48 bits à partir de K_{16}

Maintenant, nous partons du fait que nous connaissons K_{16} qui fait 48 bits, il faut trouver la clé maîtresse K de 56 bits. Pour faire cela, il faut se plonger plus en détails dans l'algorithme de cadencement des clés du DES :

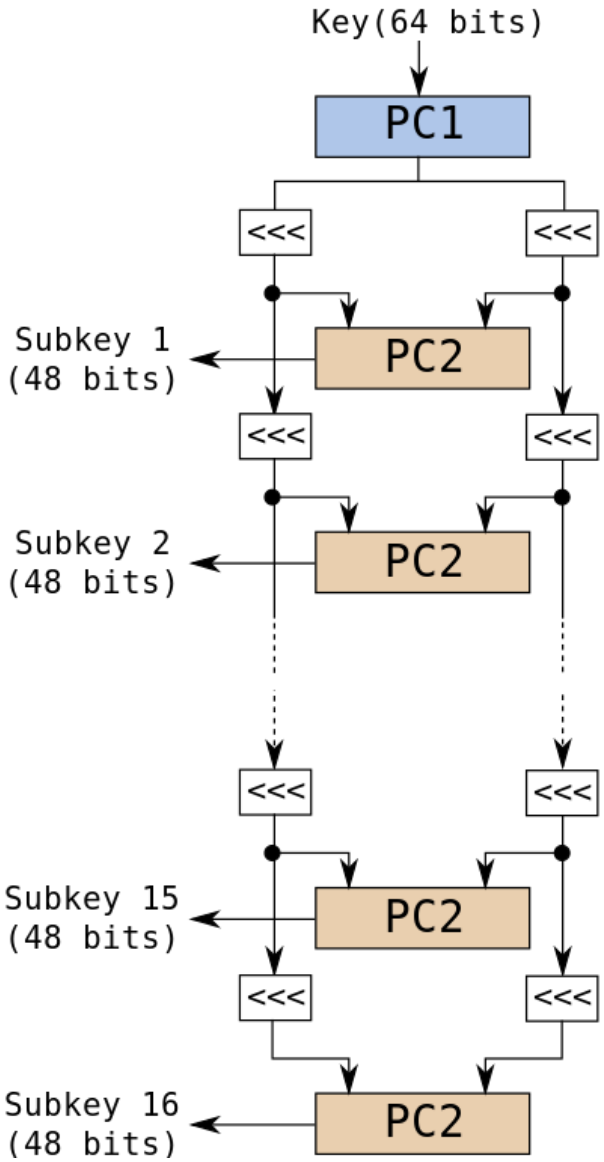

FIGURE 2.1 – Algorithme cadencement clé du DES

Dans le schéma les "««" représentent des décalages circulaires vers la gauche et les bits sont décalés d'un total de 28 positions à la fin de l'exécution de l'algorithme de cadencement des clés. Il est bon de noter que les deux parties (gauche et droite) font 28 bits chacune. Donc on a :

Pour avoir la clé maîtresse K de 48 bits, il suffit donc de calculer : $K = PC1^{-1}(PC2^{-1}(K_{16}))$

Toutefois, lorsqu'il s'agit d'effectuer l'inversion de la permutation PC2, nous étendrons de 48 à 56 bits, ce qui se traduira par une perte de 8 bits. Après avoir analysé la permutation PC2, nous pourrons ainsi déterminer la position de ces 8 bits dans PC2 inverse. Les bits 9,18,22,25,35,38,43,54 sont perdus donc on laisse un " ?".

Récupération de la clé dans le code

Après avoir déterminé les bits fautifs, les fonctions suivantes permettent de reconstruire la clé originale :

— Inversion des permutations PC2 et PC1 pour obtenir les bits de la clé.

— Vérification et test des bits de parité : Une boucle vérifie et teste les bits de parité pour trouver la clé complète.

2.3 48 bits de clé trouvée lors de cette attaque

Voici la valeur de K16 trouvée par le code :

F15408FCAB4E

Voici les 48 bits de la clé trouvés jusqu'à présent :

0011111x00011xxx11xx100x0001011x0001000x0111111x01x00x0x0x1x011x

Où x représente un bit que nous ne connaissons pas.

Chapitre 3

Question 3 : Clé complète du DES trouvée

3.1 8 bits manquants

Il faut se rappeler que nous connaissons uniquement 48 bits de la clé maîtresse K, on va maintenant rechercher les 8 derniers bits afin d'avoir la clé.

La première étape consiste à faire une recherche exhaustive à l'aide de la clé de 48 bits obtenue à l'étape précédente :
Il s'agit des bits notés x dans la clé de 48 bits précédemment énoncée donc on a 256 possibilités pour la clé finale. Étant donné que nous avons un couple clair-chiffré juste, il suffit de tester avec toutes les valeurs de la clé finale et on essaie de chiffrer le clair juste jusqu'à ce que son chiffré corresponde au chiffré juste et à ce moment-là, on aura trouvé la clé finale K.

3.2 Clé complète (56 bits) du DES trouvée

Voici ma clé trouvée lors du lancement de l'algorithme :

3E1CC816107F6476

Voici ma clé en binaire :

00111110000111001100100000010110000100000111111011001000111011O

Nous pouvons vérifier qu'il s'agit de la bonne clé en testant sur le site fourni avec le sujet du projet, nous avons un message clair ainsi qu'un message chiffré juste :

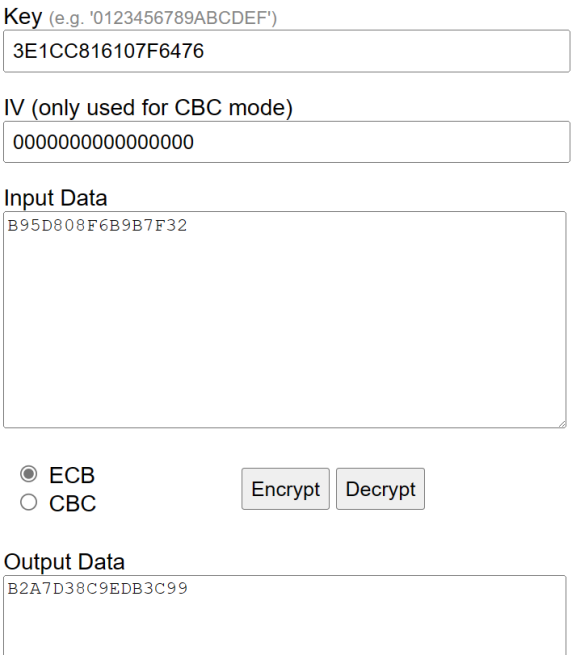

FIGURE 3.1 – Vérification de la clé

Nous pouvons voir que le chiffré associé à la clé trouvée correspond bien au chiffré juste fourni dans l'énoncé prouvant qu'il s'agit de la bonne clé.

Chapitre 4

Question 4 : Fautes sur les tours précédents

Plus tôt dans le présent rapport, j'ai évoqué la loi de Moore, il s'agit d'une prédiction faite par Gordon Moore :

> "Par circuit informatique, le nombre de transistors devait être dorénavant en mesure de doubler tous les deux ans, de quoi renforcer de façon exponentielle la puissance des ordinateurs ou de tout appareil qui utilise l'informatique"

Jusqu'à maintenant en pratique, cet énoncé s'est avéré vrai et de nos jours, pour rendre un calcul infaisable en pratique, il faudrait qu'il demande dans les environs de 2^{80} opérations élémentaires. Grâce à cette estimation, on a une limite à partir de laquelle on sait si un calcul est réalisable en pratique ou non.

Précédemment dans ce rapport, nous avons vu qu'une attaque par fautes sur le 15ème tour du DES a une complexité d'environ 2^{14} opérations.

Il est possible d'utiliser la même méthodologie qu'on a utilisée quand la faute était sur R_{15} pour les autres tours mais il faut savoir quelque chose qui peut paraître évident : quand on faute un tour, cette faute se propage dans tous les autres tours et donc cela rend les calculs beaucoup plus complexes.

Nous avons par exemple les équations suivantes si la faute est faite sur R_{14} :

Sans faute :
- $R_{15} = L_{14} \oplus f(K_{15}, R_{14})$
- $L_{15} = R_{14}$

Avec faute :
- $R^*_{15} = L_{14} \oplus f(K_{15}, R^*_{14})$
- $L^*_{15} = R^*_{14}$

Si on XOR R_{15} et R^*_{15}, on obtient (par élimination de L_{14}) :

$$R_{15} \oplus R^*_{15} = f(K_{15}, R_{14}) \oplus f(K_{15}, R^*_{14})$$

Au fait, si on refait la même chose qu'on a faite pour R_{15}, on se rend compte qu'à chaque fois qu'on remonte d'un tour, la complexité est multipliée par 2^{14}.

On obtient donc :

— 14ème tour, la complexité sera environ de 2^{28}
— 13ème tour, la complexité sera environ de 2^{42}
— 12ème tour, la complexité sera environ de 2^{56}
— 11ème tour, la complexité sera environ de 2^{70}
— 10ème tour, la complexité sera environ de 2^{84} => Irréalisable

Étant donné la loi de Moore énoncée précédemment, nous pouvons voir qu'une attaque est réalisable en pratique si jamais la faute est faite au maximum sur le 11ème tour du réseau de Feistel.

Chapitre 5

Question 5 : Contre-mesures

Il existe plusieurs contre-mesures possibles contre ce type d'attaque par fautes sur le DES

— Effectuer le calcul deux fois pour vérifier si le même résultat est obtenu à deux reprises. Cela permettrait de détecter et d'annuler le calcul en cas d'attaque de ce type. Il y aurait un impact sur le temps de calcul : puisqu'on effectuerait deux fois le même calcul, le temps nécessaire pour le calcul serait doublé par rapport à une mise en œuvre non sécurisée.

— Mettre en place un blindage comme (par exemple une cage de Faraday) pour se protéger des interférences extérieures. En effet, ce type d'attaque par faute peut être exécuté en altérant physiquement les composants électroniques, comme avec un laser, une augmentation de la température ou une modification du champ magnétique environnant. Ainsi, un blindage physique serait une solution pour contrer ce genre d'attaque. Cela n'aurait aucun impact sur le temps de calcul par rapport à une mise en œuvre non sécurisée, car il n'y aurait pas de modification du logiciel. Cependant, l'installation d'un tel dispositif est coûteuse et peut être inappropriée pour certains appareils exécutant le DES.

— Intégrer des morceaux de code de test qui s'exécutent simultanément avec l'algorithme principal. En cas de détection d'une erreur, le système est réinitialisé. Ces programmes de test peuvent vérifier le calcul en effectuant un calcul différent comme par exemple la "Preuve par neuf". Cependant, cela ajoute un certain coût supplémentaire en temps de calcul en raison du nombre d'opérations de test effectuées.