Université de Versailles
Saint-Quentin-en-Yvelines

Rapport de projet

Chiffrement et Attaque PRESENT24

Alexandre Mihet
Billal Medour

Encadrant : Christina Bourra

Préambule

Ce rapport présente le travail réalisé dans le cadre d'un projet de cryptanalyse, portant spécifiquement sur une attaque dite "par le milieu" appliquée au cryptosystème PRESENT, un chiffrement léger adapté aux environnements à faibles ressources comme l'Internet des Objets. Ce projet, supervisé par Mme Christina Bourra, a été mené dans le cadre de la troisième année de Licence en Informatique.

Afin d'implémenter cette attaque et de reproduire les expérimentations, le langage de programmation **Python** a été utilisé, pour sa flexibilité et ses bibliothèques riches en outils d'analyse et de cryptographie.

Table des matières

Chapitre 1

Introduction

L'objectif de ce projet est d'implémenter et d'analyser un chiffrement par bloc, ainsi que de concevoir une attaque "par le milieu" pour évaluer la résistance d'une version modifiée du chiffrement PRESENT, appelée PRESENT24.

Le chiffrement PRESENT a été conçu en 2007 par une équipe de chercheurs, dont Andrey Bogdanov et Christof Paar, pour répondre aux besoins de sécurité des dispositifs embarqués et de l'Internet des Objets, où les ressources sont limitées. Avec une structure en réseau de substitution-permutation, PRESENT offre un chiffrement léger et robuste face aux attaques courantes tout en maintenant une faible empreinte mémoire. Standardisé en 2009 sous la norme ISO/IEC 29192-2, il est devenu un pilier de la cryptographie légère.

Une partie clé du projet consiste à appliquer une attaque "par le milieu" (Meet-in-the-Middle) sur le chiffrement double 2PRESENT24. Dans ce schéma de double chiffrement, un message est d'abord chiffré avec une première clé $k1$ puis à nouveau chiffré avec une seconde clé $k2$.

Chapitre 2

Structure de PRESENT24

PRESENT24 est une version simplifiée du chiffrement PRESENT, où un message et une clé de 24 bits sont transformés sur 10 tours pour obtenir un texte chiffré de 24 bits. Chaque tour comporte trois étapes : une addition de sous-clé, une substitution via une S-Box et une permutation bit-à-bit. Ces étapes successives garantissent les principes de confusion et de diffusion, essentiels à la sécurité du chiffrement.

Ce chapitre détaille la fonction de chiffrement de PRESENT24 et son algorithme de cadencement de clé, permettant d'analyser en profondeur les mécanismes de protection implémentés.

2.1 Fonction de chiffrement

Algorithm 1 Fonction de chiffrement

Require: Un message m de 24 bits et 11 sous-clés K_i, $1 \leq i \leq 11$, produites par l'algorithme de cadencement de clé
Ensure: Un message chiffré c de 24 bits
1: Etat $\leftarrow m$
2: **for** $i = 1$ **to** 10 **do**
3: Etat $\leftarrow$ Etat $\oplus K_i$
4: Etat $\leftarrow$ Substitution(Etat)
5: Etat $\leftarrow$ Permutation(Etat)
6: **end for**
7: Etat $\leftarrow$ Etat $\oplus K_{11}$
8: $c \leftarrow$ Etat
9: **return** c

La fonction de chiffrement dans PRESENT24 applique une série d'opérations successives sur un message clair de 24 bits afin de produire un texte chiffré. Cette fonction suit le schéma de chiffrement SPN, structuré autour des trois étapes suivantes, répétées dans chaque tour du chiffrement :

1. **Addition** de la sous-clé : Chaque tour commence par une opération de XOR entre l'état courant (ou registre de 24 bits contenant le message en cours de chiffrement) et une sous-clé spécifique au tour en question. La sous-clé est dérivée de la clé maîtresse grâce à un algorithme de cadencement de clé (décrit dans une section

ultérieure). Cette étape permet d'ajouter une composante de secret à chaque tour du chiffrement, contribuant à la sécurité de l'algorithme.

2. **Substitution** : Une fois la sous-clé ajoutée, l'état est transformé via une substitution non-linéaire, effectuée à l'aide d'une boîte S, ou S-Box. L'état de 24 bits est divisé en six segments de 4 bits, et chacun de ces segments est remplacé par une autre valeur de 4 bits en fonction de la table de substitution de la S-Box. Cette substitution introduit de la confusion dans le chiffrement, rendant difficile la corrélation entre le texte clair et le texte chiffré.
3. **Permutation** : Enfin, une permutation bit-à-bit est appliquée pour diffuser les informations sur l'ensemble de l'état, garantissant ainsi que chaque bit du texte clair a une influence étendue sur le texte chiffré. Chaque bit de l'état est déplacé en fonction d'une table de permutation prédéfinie. Cette diffusion rend l'algorithme plus robuste en dispersant les dépendances entre les bits, ce qui complique les tentatives d'analyse linéaire ou différentielle.

Au terme des 10 tours de chiffrement, un dernier XOR est appliqué entre l'état final et une sous-clé supplémentaire pour produire le texte chiffré. Cette étape finale renforce davantage la sécurité en appliquant une couche supplémentaire de secret.

En suivant ce processus, la fonction de chiffrement de PRESENT24 garantit une transformation complexe et non-linéaire du message clair, assurant ainsi une protection renforcée du contenu.

2.1.1 Addition de la sous-clé à l'état

L'étape d'addition de la sous-clé est la première transformation appliquée au message clair à chaque tour du chiffrement. Cette opération utilise la sous-clé du tour en cours pour modifier l'état intermédiaire du message de manière non réversible. Le but de cette étape est d'introduire la clé secrète dans le processus de chiffrement, assurant ainsi que le résultat dépend bien de la clé et qu'il sera difficile de retrouver le texte clair sans elle.

Étant donnée la sous-clé du tour i, $K_i = \kappa^i_{23}\kappa^i_{22}\cdots\kappa^i_0$ pour $1 \leq i \leq 10$ et l'état actuel Etat $= b_{23}b_{22}\cdots b_1b_0$, cette étape consiste juste en l'opération

$$b_j \leftarrow b_j \oplus \kappa^i_j,$$

où $\oplus$ représente l'opération XOR.

2.1.2 Fonction de Substitution

La substitution dans PRESENT24 est une étape clé qui utilise une boîte-S (ou S-Box), qu'on notera S, pour introduire de la confusion dans le message. Cette boîte-S prend un bloc de 4 bits en entrée et le transforme en un autre bloc de 4 bits selon une table prédéfinie :

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S[x]$	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

Chaque entrée et sortie de la boîte-S est constituée de 4 bits et peut être donc représentée par un chiffre hexadécimal. C'est cette notation hexadécimale qui est utilisée pour décrire l'action de la boîte-S. Par exemple, $0011_2 = 3_{16} \rightarrow b_{16} = 1011_2$.

Pour cette étape, l'état de 24 bits $b_{23}b_{22}\cdots b_1b_0$ est divisé en 6 mots de 4 bits $w_5w_4\cdots w_0$, où $w_i = b_{4i+3}||b_{4i+2}||b_{4i+1}||b_{4i}$ pour $0 \leq i \leq 5$. La boîte-S est appliquée alors à chaque mot w_i de l'état, transformant tous les mots w_i en $S[w_i]$, pour $0 \leq i \leq 5$.
Par exemple, si un segment de 4 bits est 0011_2 (ou 3_{16} en hexadécimal), il sera transformé en b_{16} (ou 1011_2 en binaire) selon la S-Box.

Cette transformation non-linéaire rend difficile la prédiction des liens entre le texte clair et le texte chiffré, augmentant ainsi la sécurité du chiffrement en introduisant de la complexité dans l'algorithme.

2.1.3 Fonction de Permutation

La fonction de permutation dans PRESENT24 est conçue pour réaliser la **diffusion** en réorganisant les bits de l'état intermédiaire selon une table prédéfinie. Chaque bit i est déplacé à une nouvelle position $P(i)$, dispersant ainsi les informations sur l'ensemble de l'état.

i	0	1	2	3	4	5	6	7	8	9	10	11
$P(i)$	0	6	12	18	1	7	13	19	2	8	14	20
i	12	13	14	15	16	17	18	19	20	21	22	23
$P(i)$	3	9	15	21	4	10	16	22	5	11	17	23

Voici à quoi ressemble deux tours de PRESENT24 graphiquement :

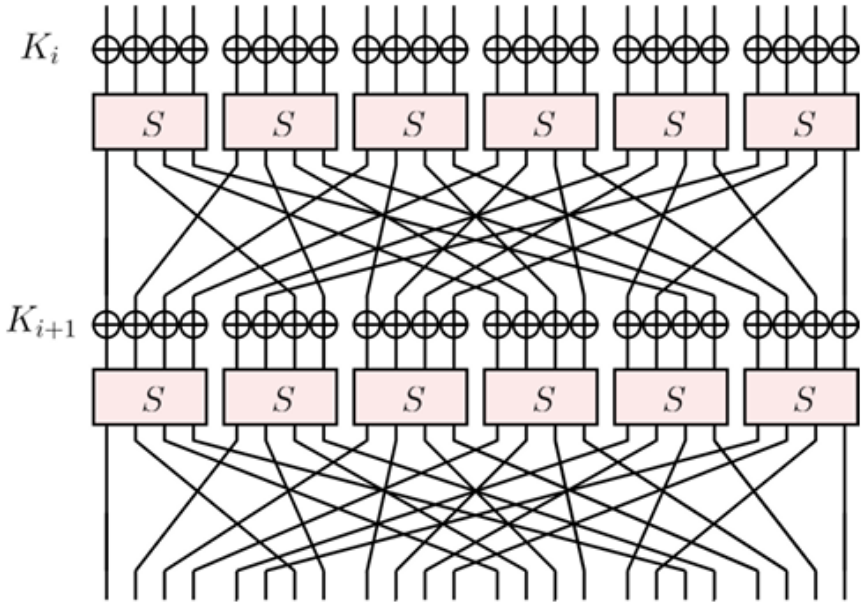

FIGURE 2.1 – Deux tours de PRESENT24 (tours i et $i+1$)

2.2 Algorithme de cadencement de clé

L'algorithme de cadencement de clé de PRESENT24 est directement inspiré de celui utilisé dans la version standard de PRESENT, qui repose sur une clé de 80 bits. Pour

cette raison, un registre K de 80 bits est utilisé dans cette version également. Ce registre, noté $K = k_{79}k_{78}\dots k_0$, est initialisé en plaçant la clé maîtresse de 24 bits dans les bits de poids fort ($k_{79}\dots k_{56}$) et en remplissant les 56 bits restants avec des zéros. Cette méthode permet de conserver la structure de l'algorithme de cadencement tout en adaptant le chiffrement à une clé de 24 bits.

Exemple : Si la clé maîtresse est `f34ab7`, alors, au début de l'algorithme, le registre K contiendra la valeur suivante (en notation hexadécimale) :

`f34ab700000000000000`.

Ici, comme dans tout le chiffrement, les bits de poids faible sont à droite, et les bits de poids fort sont à gauche.

À chaque tour i du chiffrement, une sous-clé K_i de 24 bits est dérivée à partir du registre K. Cette sous-clé est composée des bits de K situés entre les positions k_{39} et k_{16} incluses, ce qui s'écrit mathématiquement comme :

$$K_i = k_{39}k_{38}\dots k_{17}k_{16}.$$

2.2.1 Mise à jour du registre de clé

Après avoir extrait la sous-clé K_i pour le tour en cours, le registre K est mis à jour en trois étapes principales :

— **Rotation** : Le registre K est pivoté de 61 positions vers la gauche. Cela signifie que chaque bit est déplacé vers une position plus basse de 61 places, avec les bits de poids fort revenant en boucle à la position de poids faible. Mathématiquement, cette opération peut être représentée par :

$$[k_{79}k_{78}\dots k_1k_0] = [k_{18}k_{17}\dots k_{20}k_{19}]$$

— **Substitution des bits de poids fort** : Les 4 bits les plus à gauche du registre K, soit $k_{79}k_{78}k_{77}k_{76}$, sont transformés en utilisant la même boîte-S que celle employée dans la fonction de chiffrement de PRESENT24. Cette substitution introduit une non-linéarité dans la génération de la sous-clé, renforçant ainsi la sécurité contre les attaques. La substitution est notée :

$$[k_{79}k_{78}k_{77}k_{76}] = S[k_{79}k_{78}k_{77}k_{76}]$$

— **XOR avec le numéro de tour** : Enfin, un XOR est appliqué entre les bits $k_{19}k_{18}k_{17}k_{16}k_{15}$ du registre K et une représentation binaire du numéro du tour actuel, notée i. Cela ajoute une composante de variabilité dépendant du tour, assurant que chaque sous-clé est unique. Par exemple, pour le tour 1, on fait un XOR avec $i = 1 = (00001)$ en binaire, et pour le tour 2, avec $i = 2 = (00010)$, etc. Cette étape se formalise comme suit :

$$[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus i$$

2.2.2 Rôle de l'algorithme de cadencement de clé

L'algorithme de cadencement de clé de PRESENT24 assure que chaque tour utilise une sous-clé différente, ce qui complexifie le chiffrement et renforce sa résistance aux

attaques. La rotation et la substitution introduisent une non-linéarité et une diffusion dans la génération des sous-clés, tandis que l'ajout du numéro de tour garantit que les sous-clés varient à chaque itération. Cette approche rend difficile toute tentative d'attaque qui reposerait sur des répétitions ou des motifs dans les sous-clés.

Chapitre 3

Explication des attaques par le milieu

Les attaques par le milieu, également connues sous le nom d'attaques *Meet-in-the-Middle*, sont des techniques de cryptanalyse efficaces pour casser les systèmes de chiffrement qui utilisent un double chiffrement. Cette méthode a été initialement développée pour attaquer le *Double DES*, mais elle peut également s'appliquer à d'autres schémas de double chiffrement comme le *2PRESENT24*.

3.1 Principe de l'attaque par le milieu

L'attaque par le milieu repose sur l'observation qu'un chiffrement double, qui applique un même algorithme de chiffrement deux fois avec deux clés différentes, n'offre pas nécessairement une sécurité deux fois plus élevée. Au lieu de chercher directement les deux clés, cette attaque divise le processus en deux moitiés, permettant ainsi de retrouver les clés en deux étapes distinctes.

Dans le cas d'un double chiffrement, un message m est chiffré avec une première clé k_1 pour produire un intermédiaire, puis ce résultat intermédiaire est chiffré à nouveau avec une seconde clé k_2 pour obtenir le message chiffré final c. L'équation peut s'écrire comme suit :

$$c = 2PRESENT24_{k_1,k_2}(m) = \text{PRESENT24}_{k_2}(\text{PRESENT24}_{k_1}(m)).$$

L'attaque par le milieu permet de réduire le nombre d'essais nécessaires pour retrouver les clés k_1 et k_2, en exploitant des couples clair-chiffré (m, c) connus.

3.2 Étapes de l'attaque par le milieu

Voici les étapes principales de l'attaque par le milieu appliquée au chiffrement 2PRESENT24 :

1. **Génération de l'espace des clés intermédiaires** :
   — On commence par générer toutes les valeurs intermédiaires possibles pour le message m chiffré avec la première clé k_1. Pour chaque clé candidate k_1, on chiffre le message m et on enregistre le résultat intermédiaire dans une table sous la forme (clé k_1, valeur intermédiaire).
2. **Déchiffrement à partir du texte chiffré** :
   — Ensuite, pour chaque clé candidate k_2, on prend le message chiffré c et on le déchiffre en supposant que c a été chiffré avec k_2. Cela produit une valeur

intermédiaire possible qui aurait pu être générée après le premier chiffrement avec k_1. On enregistre également ces résultats dans une table sous la forme (clé k_2, valeur intermédiaire).

3. **Recherche de correspondances** :
   — L'étape clé de l'attaque consiste à comparer les valeurs intermédiaires générées lors des étapes précédentes. Si une valeur intermédiaire correspond entre les deux tables, cela signifie qu'il existe une clé k_1 dans la première table et une clé k_2 dans la seconde table telles que :

$$\text{PRESENT24}_{k_2}(\text{PRESENT24}_{k_1}(m)) = c.$$

   — Cette correspondance indique que (k_1, k_2) est une paire de clés candidates qui pourrait avoir produit le couple clair-chiffré (m, c).

4. **Vérification des clés** :
   — Puisque l'attaque peut produire plusieurs paires de clés candidates, il est nécessaire de vérifier chaque paire (k_1, k_2) avec d'autres couples clair-chiffré pour identifier la paire correcte.

3.3 Complexité de l'attaque

L'attaque par le milieu permet de réduire significativement le nombre de tentatives nécessaires par rapport à une recherche exhaustive. Dans un chiffrement double avec deux clés de n bits, une recherche brute nécessiterait 2^{2n} essais. Cependant, en divisant le processus en deux étapes, l'attaque par le milieu ne nécessite que $2^n + 2^n = 2^{n+1}$ opérations, ce qui est beaucoup plus faisable pour des valeurs de n raisonnables.

Dans le cas de 2PRESENT24, où chaque clé est de 24 bits, une recherche exhaustive demanderait 2^{48} essais. Avec l'attaque par le milieu, ce nombre est réduit à 2^{25} essais (soit 2^{24} pour chaque moitié), ce qui est une amélioration significative en termes de complexité.

3.4 Limites de l'attaque

Bien que l'attaque par le milieu réduise la complexité, elle nécessite un stockage considérable pour enregistrer les valeurs intermédiaires de chaque clé candidate. Cette exigence de mémoire peut limiter son application dans des contextes où la mémoire est une ressource critique.

De plus, cette méthode n'est efficace que pour les schémas de chiffrement à double application avec deux clés indépendantes. Elle n'est pas directement applicable aux systèmes à clé unique ou aux algorithmes qui ne suivent pas une structure similaire.

Chapitre 4

Conception du projet

Ce chapitre présente la structure et les choix de conception du projet, qui vise à implémenter le chiffrement PRESENT24, son déchiffrement, et une attaque par le milieu sur une version double du chiffrement, nommée 2PRESENT24. Le projet est structuré autour de quatre fichiers principaux : `CHIFFREMENT.py`, `DECHIFFREMENT.py`, `ATTAQUE.py`, et `main.py`, chacun ayant un rôle spécifique dans l'organisation du code.

4.1 Structure des fichiers

4.1.1 CHIFFREMENT.py

Le fichier `CHIFFREMENT.py` contient les fonctions principales nécessaires pour réaliser le chiffrement d'un message clair en utilisant l'algorithme PRESENT24. Il inclut :

- **Cadencement de clé (Key Scheduling)** : La fonction `Cadencement_de_cle` génère les sous-clés nécessaires pour chaque tour de chiffrement. À partir de la clé maîtresse de 24 bits, un registre de 80 bits est initialisé (en remplissant les bits de poids faible avec des zéros), puis pivoté et transformé pour créer une sous-clé de 24 bits pour chaque tour.
- **Permutation et Substitution** : Les fonctions `Permutation` et `Boite_s` appliquent respectivement la permutation bit-à-bit et la substitution (boîte S) à l'état courant du message. Ces étapes assurent la diffusion et la confusion, des principes essentiels dans la conception de chiffrements sécurisés.
- **Chiffrement** : La fonction principale `Chiffrement` prend un message clair (sous forme d'état de 24 bits) et une clé maîtresse, puis applique 10 tours de chiffrement en utilisant les sous-clés générées, la substitution et la permutation.

4.1.2 DECHIFFREMENT.py

Le fichier `DECHIFFREMENT.py` implémente le processus inverse du chiffrement, permettant de déchiffrer un message chiffré en utilisant la même clé maîtresse. Il comprend :

- **Inverse de la permutation et de la substitution** : Les fonctions `INV_Permutation` et `INV_Boite_s` appliquent les transformations inverses à celles utilisées pour le chiffrement, permettant de revenir à l'état initial du message clair.
- **Déchiffrement** : La fonction `Dechiffrement` effectue les 10 tours de déchiffrement en appliquant les sous-clés dans l'ordre inverse et en utilisant les fonctions inverses de substitution et de permutation pour retrouver le message clair.

4.1.3 ATTAQUE.py

Ce fichier est consacré à l'attaque par le milieu (MITM) sur le chiffrement double 2PRESENT24. Voici les éléments principaux :

- **Recherche des éléments communs** : La fonction `Trouver_elements_communs` compare les résultats intermédiaires de chaque moitié de l'attaque pour identifier les valeurs communes entre le chiffrement partiel et le déchiffrement partiel, indiquant une potentielle paire de clés (k_1, k_2) correcte.
- **Simulation de toutes les clés possibles** : La fonction génère toutes les combinaisons de 24 bits pour simuler chaque clé possible, puis réalise un chiffrement partiel (avec k_1) et un déchiffrement partiel (avec k_2) sur le couple clair-chiffré. Ces valeurs intermédiaires sont ensuite comparées pour trouver une correspondance.

4.1.4 main.py

Le fichier `main.py` est le script principal qui permet de lancer les différentes opérations de chiffrement, déchiffrement, et attaque par le milieu depuis la ligne de commande. Il guide l'utilisateur en lui demandant de saisir le message et la clé, puis utilise les fonctions des autres fichiers pour exécuter l'opération choisie.

4.2 Choix de conception et justifications

Le choix de structurer le projet en plusieurs fichiers permet une modularité et une lisibilité accrue. Chaque fichier se concentre sur une fonctionnalité spécifique, facilitant ainsi les modifications et les tests individuels de chaque composant.

4.2.1 Modularité du code

La séparation des fichiers `CHIFFREMENT.py`, `DECHIFFREMENT.py` et `ATTAQUE.py` permet de regrouper les fonctions en fonction de leur rôle. Ce découpage favorise également la réutilisabilité des fonctions, en particulier si des modifications sont nécessaires pour ajuster les paramètres ou tester de nouvelles variantes de PRESENT.

4.2.2 Gestion des sous-clés

L'algorithme de cadencement de clé joue un rôle crucial dans la sécurité du chiffrement. Le choix de maintenir un registre de 80 bits pour générer les sous-clés s'aligne sur la méthode utilisée dans la version originale de PRESENT, assurant ainsi une robustesse similaire. Chaque sous-clé est générée à partir de transformations complexes (pivotement, boîte S, et ajout du numéro de tour) pour éviter les répétitions et augmenter la sécurité contre les attaques par force brute.

4.2.3 Utilisation de Python et de ses bibliothèques

Python a été choisi pour sa flexibilité et la lisibilité de son code, ce qui en fait un langage adapté pour un projet académique de cryptographie. La bibliothèque `itertools` est utilisée dans `ATTAQUE.py` pour générer toutes les combinaisons de bits lors de l'attaque, ce qui simplifie l'implémentation de la recherche exhaustive.

Bibliographie

[1] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, C. Vikkelsoe, *PRESENT : An Ultra-Lightweight Block Cipher*, in Proceedings of CHES 2007, pp. 450–466, *Lecture Notes in Computer Science*, vol. 4727, Springer, 2007.
`https://iacr.org/archive/ches2007/47270450/47270450.pdf`

[2] G. Leander, *Small Scale Variants Of The Block Cipher PRESENT*, 2010.
`https://eprint.iacr.org/2010/143.pdf`