UE IN608N (CRYPTO): Attaque intégrale sur une version réduite d'un AES-128

Rapport IN608

Docherty Ronan, Ejjed Zakaria, Gago Jérémy, Guerin Raphaël, Thienard Rodolphe

10 Mai 2022

INTRODUCTION

Le but de ce Projet était de d'implémenter une version réduite d'un AES-128 avec un mode de chiffrement OFB et/ou CTR en C. Dans notre cas nous avons choisit le chiffrement CTR. Nous avons choisit le mode CTR pour des questions d'optimisations, car il est parallélisable et plus rapide que le mode OFB.
Mais aussi et surtout réalisé l'attaque intégrale sur 4 tours de l'AES-128 en mode de chiffrement CTR.

IMPLÉMENTATION DE L'AES-128

2.1 Gestion des commandes

Nous avons créer le fichier command.c et command.h pour faire la gestion des commandes. En fonction de la demande...

Un tour d'AES-128 est composé de 4 fonctions :

1. SubOctet : Substitue les octets de la matrice d'état.
2. DecaleLignes : Décale les lignes de la matrice d'état.
3. MelangeColonnes : Mélanges les colonnes de la matrice d'état.
4. AjouteCleTour : Ajoute la clé de tour à la matrice d'état.

Nous commencerons donc par la fonction SubOctet puisque c'est la première fonction du tour.

2.2 SubOctet

La fonction `void suboctet(int **matrix)` nous permet de substituer l'octet contenu dans une case de notre matrice, à l'aide de la table de substitution de l'AES, qui nous donnera un octet différent du précédent.

2.3 DecaleLignes

La fonction `decale_lignes(int **matrix);` permet de décaler les lignes de la matrice d'état en fonction de la ligne. Pour faire cela on créer une matrices (`int **`) temporaire qui stockera le décalage des lignes. Ces modifications serons recopié dans la matrices d'état.

2.4 MelangeColonnes

La fonction `void melange_colonnes(int **matrix)` permet, à l'aide de deux matrices(`operation_double` et `operation_triple`), d'effectuer certaines opérations pour donner de nouvelles valeurs aux lignes de chaque colonne de notre matrice. Les opérations sont les suivantes :

1. $$B'_{0,1} = (2 \star B_{0,1}) \oplus (3 \star B_{1,1}) \oplus B_{2,1} \oplus B_{3,1}$$
   La première ligne prendra la valeur de : l'octet renvoyé par la case correspondant à l'octet de la ligne 0 de cette colonne dans `operation_double` XOR l'octet renvoyé par la case correspondant à l'octet de la ligne 1 dans `operation_triple` XOR l'octet de la ligne 2 XOR l'octet de la ligne 3.

2. $$B'_{1,1} = B_{0,1} \oplus (2 \star B_{1,1}) \oplus (3 \star B_{2,1}) \oplus B_{3,1}$$
   La seconde prendra la valeur de : l'octet de la ligne 0 XOR l'octet renvoyé par la case correspondant à l'octet de la ligne 1 dans `operation_double` XOR l'octet renvoyé par la case correspondant à l'octet de la ligne 2 dans `operation_triple` XOR l'octet de la ligne 3

3. $$B'_{2,1} = B_{0,1} \oplus B_{1,1}(2 \star B_{2,1}) \oplus (3 \star B_{3,1})$$
   La troisième prendra la valeur de : l'octet de la ligne 0 XOR l'octet de la ligne 1 XOR l'octet renvoyé par la case correspondant à l'octet de la ligne 2 dans `operation_double` XOR l'octet renvoyé par la case correspondant à l'octet de la ligne 3 dans `operation_triple`

4. $$B'_{3,1} = (3 \star B_{0,1}) \oplus B_{1,1} \oplus B_{2,1}(2 \star B_{3,1})$$
   La dernière prendra la valeur de : l'octet renvoyé par la case correspondant à l'octet de la ligne 0 dans `operation_triple` XOR l'octet de la ligne 1 XOR l'octet de la ligne 2 XOR l'octet renvoyé par la case correspondant à l'octet de la ligne 3 dans `operation_double`

2.5 AjouteCleTour

La fonction `addroundkey(int **matrix, int **key);` fait un XOR entre la matrice d'état courante et la matrice de la clé de tour :
`matrix[i][j] = matrix[i][j] ^ key[i][j];`
Comme les deux matrices sont en hexadécimales il existe le XOR en C qui est le : " ^ ".

2.6 Les Tours

Pour les tours d'AES nous avons une fonction `void turn(int matrix, int key)`, qui effectue les fonctions précédemment cité (dans le même ordre), nous avons ensuite une fonction `void last_turn(int matrix, int key)` qui effectue les même fonctions que `turn` à part la fonction `melange_colonne` et enfin `void aes(int matrix, int exten` qui effectue un nombre définit de fois la fonction `turn` et qui termine par une itération de `last_turn`, dans notre cas 10 tour.

2.7 Gestion des clé

L'AES-128 utilise 2 clés différentes :

1. - **La clé Maitre**
2. - **La clé de Tour**

La clé Maitre est donnée par l'utilisateur en entrée doit faire 128 bits. La clé de tour quand a elle n'est pas donné en entrée mais utilise la clé maitre pour être créer et change à chaque tour qu'effectue l'AES.

2.7.1 Clé Maitre

Comme dit précédemment la clé maitre est la clé principale de l'AES-128. Cette clé est donnée par l'utilisateur et doit obligatoirement faire 128 bits. Cette clé est utilisé pour la création des clé de Tour. La clé Maitre étant donné en argument elle est de type `char*`, nous avons donc créé une fonction qui convertit les `char*` en matrices (en `int **`).

2.7.2 Clé de Tour

Pour générer la clé de Tour nous avons besoin d'étendre la clé de maitre de 128 bits à 1408 bits. Puis chaque portion de 128 bits de la clé maitre étendue deviens une clé de tour. Comme il y a dix tours il y a 10 clé de tour, plus une pour le dernier tour.

2.8 Chiffrement CTR

Pour faire le chiffrement CTR nous utilisons une fonction
`void counter_mode(int matrix, int extended_key, int **plaintext, int turns)` qui après avoir fait des tours d'AES sur une matrice, XOR cette matrice avec une matrice contenant 16 bits de texte clair, nous utilisons aussi une fonction `void loop_ctr(struct chained_matrixchained, int matrix, int extended_key, int turns)` qui quant à elle nous place à la fin du message clair et qui effectue

la fonction `counter_mode` avec les bouts de texte clair autant de fois qu'il y'a de bouts.