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

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

10 Mai 2022

Sommaire

Chapitre 1

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. Mais aussi et surtout réalisé l'attaque intégrale sur 4 tours de l'AES-128.

Chapitre 2

AES-128

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.

2.1 Suboctet

La fonction `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.

2.2 Decale lignes

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.

2.3 Melange colonnes

La fonction `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 :

Avec Bx,y l'octet contenu dans la ligne x et la colonne y de la matrice entré en argument, B' étant la matrice résultant des opérations, $2 \star Bx, y$ correspondant à l'octet renvoyé par la case de `operation_double`, à la ligne égale à la première partie de l'octet renvoyé par Bx,y et égale à

la colonne de la deuxième partie de cet octet, idem pour $3 \star B_{x,y}$ mais cette fois dans la matrice `operation_triple`

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`

2.4 Ajout cle tour

La fonction `addroundkey(int **matrix, int **key);` fait un XOR entre la matrice d'état courante et la matrice de la clé de tour.

2.5 Les Tours

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

2.6 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 et 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.

2.6.1 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 **`) `str_to_matrix(char *key)`.

2.6.2 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. La clé de tour est donc généré chaque tour par la fonction `key_extension(int **master_key, int **extended_key)` qui à partir de la clé maitre effectue un SubOctet sur la RotationOctet d'une clé tampon puis un XOR avec les constantes Rcon. L'action RotationOctet est effectué dans `rotation_left(int *tampon)` et les constantes Rcon dans un variable du même nom.

2.7 Mode de chiffrement

Pour faire le chiffrement CTR nous utilisons une fonction
`counter_mode(int matrix, int extended_key, int **plaintext, int turns)` qui après avoir fait les tours d'AES sur une matrice, XOR cette matrice avec une autre matrice contenant 16 bits de texte clair. Cette fonction est ensuite utilisé dans la fonction
`loop_ctr(struct chained_matrixchained, int matrix, int extended_key, int turns)` qui quant à elle, nous place au début d'une liste chainée(ici le message clair) et effectue la fonction `counter_mode` avec les bouts de 16 bits de texte clair, autant de fois qu'il y'a de bouts.

2.8 NONCE

Le nonce ou number used once, est un nombre aléatoire. Il a pour utiliter de rendre le chiffrement unique comme la cle maitre. Dans notre programme, nous avons decide de le generer a partir du random genere par le kernel linux au demarage. Si celui-ci n'est pas generer ou pas correctement, il bloque le demarrage de l'OS jusqu'a optenir une bonne generation. Il est considere comme etant un vrai aleatoire. Il utilise pour se generer differents nombre aleatoire issue du hardware.

Chapitre 3

Attaque sur AES-128 4 tours

3.1 L'attaque

On ne connait pas la cle maitre donc pour retrouver on doitr avoir le meme chiffre Nous cherchons a mettre en application la particularite mise en evidence sur AES-128 a 4 tours. Pour se faire, nous avons mis en place une structure de liste chaine sous la forme :

$$\begin{matrix} \text{i } 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \end{matrix} \quad \text{avec } i \in \{0, .., 255\}$$

3.2 Matrice inverse

Nous avons mis en place un calcul nous permettant de calculer la matrice inverse de AES_TABLE, la fonction n'est execute qu'une seul fois.

```
1     int *foo = calloc(256, 4);
2     int *tmp = aes_table;
3     for(int i = 0; i < 256; i++)
4     {
5       *(foo + tmp[i]) = i;
6     }
7
8     for(int j = 0; j < 16; j++)
9     {
10      for(int i = 0; i< 16; i++)
11      {
12        printf("0x%02X,", foo[j*16 + i]);
13      }
14      printf("\n");
15    }
16    free(foo);
17
```

3.3 Retrouver la cle K4

La cle K4 est la cle d'encryption du 4eme tour. Pour obtenir cette cle, nous avons besoin de retrouver tous les a $a \in \{0, .., 255\}$ verifiant l'egalite suivante :

$$\oplus_{i=0}^{255} SubOctetInverse(D_{k,I}^{i} \oplus a) = 0$$

3.4 De K4 a K

Pour reduire la cle K4 a K, nous appliquons la formule de reduction de cle. Faite a partir de cle_extention, nous avons juste reapplique pour que le Xor s'annule et ainsi retourver les cles inferieurs jusqu'a K.

3.5 Verification de K

Nous avons maintenant obtenu une cle possible parmis les differentes obtenable. Nous faisons donc une encryption des matrices :

$$\begin{matrix} \text{i } 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \end{matrix} \quad \text{avec } i \in \{0, .., 255\}$$

Nous regardons par la suite si les matrices obtenues sont egales aux matrices que nous avons tente de decrypter. Si elles sont egalent, nous avons trouver la cle maitre, sinon, nous recommencons a partir de l'etape precedente en changeant l'un des 'a' obtenu parmis les colones.

Chapitre 4

Compilation

Tout d'abord pour lancer le programme il vous suffit de taper `make` dans le terminal. La commande `make` vous affiche les différents arguments a mettre et les modes disponibles. Les arguments disponibles sont les suivant :

1. -a : qui permet de faire l'attaque.
2. -e : qui permet de faire l'encryptage et le décryptage d'un fichier.
3. -out : qui permet de nommer le fichier de sortie, par defaut: "output.txt".
4. -h : qui permet d'afficher les aides.
5. -nonce : qui génère un nombre aléatoire pour le CTR.

Dans le `Makefile` il existe déjà des commandes types comme `make enc` qui effectue un encryptage sur le fichier `matrices.txt`. Nous gérons les erreurs possibles d'arguments, ainsi que les arguments nécessaire qui ne sont pas présent.

Après avoir tapé la commande :
`./$(BIN) "aes128 clemaitre" "azertyuiopqsdfgh" e test.txt -out sortie.txt`
Le programme va effectué l'encryptage du fichier `test.txt` avec la clé maitre `"aes128 clemaitre"` en utilisant la nonce `azertyuiopqsdfgh` pour le chiffrement CTR et `sortie.txt` comme fichier de sortie d'encryptage.