Compte rendu

8 mai 2023

Algorithme AES

Enzo reale
Celine djadel

Dihia derbal
Amel lassal

Table des matières

1 Introduction

En informatique, la cryptographie fait référence à des techniques sécurisées d'information et de communication dérivées de concepts mathématiques, et à un ensemble de calculs basés sur des règles appelées algorithmes pour transformer les messages d'une manière difficile à déchiffrer.
La cryptographie moderne se préoccupe des trois objectifs suivants :

- Confidentialité : les informations ne peuvent être comprises que par le destinataire.
- Intégrité : les informations ne peuvent pas être modifiées dans le stockage ou le transit entre l'expéditeur et le destinataire prévu, sans que l'altération ne soit détectée.
- Authentification : l'expéditeur et le destinataire peuvent confirmer l'identité de l'autre et l'origine / la destination des informations.

On distingue deux types de chiffrements cryptographiques : le chiffrement symétrique et le chiffrement asymétrique. Le chiffrement symétrique repose sur un principe simple ; une clé est partagée par l'expéditeur et le destinataire, celle-ci servira à chiffrer et à déchiffrer le message qu'ils s'échangent. Un chiffrement asymétrique est un peu plus complexe : chaque personne dispose d'un jeu de clés (une privée et une publique) ; pour chiffrer un message on utilisera la clé privée du destinataire, alors que pour en déchiffrer un, le destinataire utilisera sa clé privée.

Dans le cadre de notre projet, nous nous intéresserons à L'Advanced Encryption Standard (AES) aussi connu sous le nom de Rijndael, qui est un algorithme de chiffrement par bloc. Il est encore aujourd'hui le chiffrement symétrique le plus utilisé au monde. Il a été développé après un appel international à candidatures lancé en janvier 1997 par le NIST (National Institute of Standards and Technology). Au terme de cet examen, le candidat Rijndael du nom de ses deux créateurs, Joan Daemen et Vincent Rijmen fut retenu. Ainsi, L'AES a été adopté par le NIST en 2001, puis a finalement été approuvé par la NSA (National Security Agency). Il remplace le DES (choisi comme standard dans les années 1970) qui de nos jours devenait obsolète.

L'idée de ce projet est donc d'implémenter le chiffrement et le déchiffrement de l'AES-128 qui utilise, comme son nom l'indique, une clé de 128 bits. Nous développerons aussi une attaque sur 4 tours de ce chiffrement (contre 10 tours au total) car il n'existe pas d'attaques "efficaces" connues sur l'AES complet. Cette attaque s'appelle attaque intégrale (ou encore attaque par saturation ou square attack). Elle exploite des propriétés structurelles de l'AES et permet de retrouver la clé secrète.

2 Description du chiffrement AES

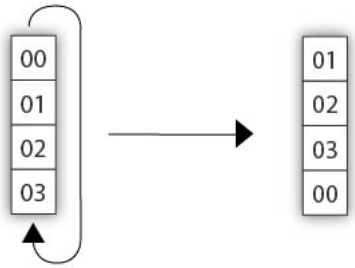
La version que nous avons implementé de l'AES utilise une clé maitre de 128 bits ainsi qu'une clé ronde de 128 bits également. Le nombre de tours est de 10 au total. Le texte clair donné en entrée à l'AES afin d'obtenir le texte chiffré est représenté par une matrice de 4x4 et cette représentation est appelée "état". Un tour de l'AES est une composition de plusieurs transformations SubBytes, ShiftRows, MixColumns ainsi que AddRoundKey.

2.1 Extension de la clé

Étant donné que nous ne fournissons à l'AES qu'une seule clé, nous avons besoin d'en dériver un certain nombre de clés rondes qui varie en fonction de la version de l'AES utilisée. Puisque nous utilisons l'AES-128, nous avons besoin de génerer exactement 11 clés.

Nous avons implementé trois fonctions différentes pour nous aider à construire la fonction KeyExpension() :

- RotWord : prend en entrée 4 octets et effectue une rotation de ces 4 octets de manière à ce que celui en dernière postion se retouve en première position après la transformation.



— SubWord : fais la substituion de 4 octets et renvoie également une sortie sur 4 octets où chaque octet est remplacé par sa valeur associée dans la Sbox.
— Rcon : prend un entier en entrée et renvoie un tableau de 4 octets avec les 3 octets les moins significatifs mis à 0. Nous avons simplement utilisé une table de recherche rcon contenant déjà les résultats des cacluls nécessaires.

2.2 PrintState

La fonction printState permet de convertir une chaîne de caractères en une clé hexadécimale et de la mettre sous forme de matrice 4x4. Cette étape est nécessaire pour pouvoir appliquer les transformations sur l'état de manière cohérente.

2.3 SubBytes

La fonction SubBytes doit prendre en entrée l'état et renvoyer le nouvel état modifié après la transformation. La table S-Box est utilisée pour effectuer cette transformation en remplaçant chaque octet par la valeur correspondante dans la table.

2.4 ShiftRows

La fonction ShiftRows effectue une rotation horizontale de chaque ligne de l'état. Plus précisément, la première ligne n'est pas modifiée, la deuxième ligne est décalée d'une position vers la gauche, la troisième ligne de deux positions vers la gauche et la quatrième ligne de trois positions vers la gauche. Cette étape permet de mélanger les données de l'état de manière à rendre la tâche de décryptage plus difficile pour un attaquant. La fonction np.roll est utilisée pour appliquer la rotation sur chaque ligne de l'état.

2.5 MixColumns

Cette transformation prend un état en entrée et multiplie chaque colonne de cet état par une matrice spéciale. Cela permet de diffuser les bits à travers l'état et ajoute de la confusion afin de renforcer la sécurité du chiffrement. Après la multiplication, la fonction MixColumns renvoie le nouvel état. Cette transformation est réversible, ce qui signifie qu'elle peut être annulée avec une transformation inverse lors du processus de déchiffrement.

2.6 AddRoundKey

Il s'agit de la dernière transformation d'un tour, qui est la transformation finale avant de passer au tour suivant. Elle consiste simplement en un XOR entre les valeurs de l'état et les valeurs de la clé ronde correspondante. L'objectif est de "mélanger" la clé avec l'état de telle sorte que chaque bit de la clé influence chaque bit de l'état. Le paramètre "round" de la fonction AddRoundKey représente la clé ronde correspondante à ce tour. Chaque tour utilise une clé différente, qui est dérivée de la clé principale en utilisant la fonction KeyExpansion.

3 Déchiffrement

3.1 Chiffrement inverse

Le processus de déchiffrement de l'AES se base sur l'algorithme de chiffrement, mais cependant il prend en argument un message chiffré et la clé puis détermine le message clair en faisant dans l'ordre inverse, l'inverse des fonctions de chiffrement. Le résultat obtenu sera affiché. Le déchiffrement commencera donc par ajouter la clé ronde (AddRoundKey), faire le ShiftRows inversé et le SubBytes inversé pour le premier round et pour les 9 derniers on ajoutera en plus le MixColumns inversé après l'AddRoundKey.
Fonctionnalités :

— Arguments en entrée : message chiffré, clé
— Sortie : Message déchiffré en clair

3.1.1 ShiftRowsInverse

La fonction ShiftRowsInverse est l'inverse de la transformation ShiftRows : Les octets des trois dernières lignes sont déplacés cycliquement sur différent nombres d'octets. La première ligne **L = 0** n'est pas déplacée. Les trois dernières lignes sont décalés selon une rotation horizontale cette fois-ci vers la **droite**.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 5 & 6 & 7 & 4 \\ a & b & 8 & 9 \\ f & c & d & e \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & a & b \\ c & d & e & f \end{bmatrix}$$

3.1.2 SubBytesInverse

La fonction SubBytesInverse est l'inverse de la transformation de substitution de bits, dans laquelle l'inverse de la Sbox est appliqué à chaque bits de l'état. On obtient le résultat en appliquant l'inverse de la transformation puis en prenant l'inverse multiplicatif de GF(2^8). On utilise la même Sbox que celle utilisée dans SubWord.

3.1.3 MixColumnsInverse

La fonction MixColumnsInverse est l'inverse de la transformation MixColumns. MixColumnsInverse est opérée sur l'état colonne par colonne, traitant ainsi chaque colonne comme un polynôme de degré 4. Les colonnes sont considérées comme des polynômes sur GF(2^8) et multipliées par $x^4 + 1$ avec un polynôme fixe $a^{-1}(x)$, donné par : $a^{-1}(x) = 0bx^3 + 0dx^2 + 09x + 0e$. C'est donc plus simplement une multiplication de matrice de chaque colonne, et on utilise donc l'inverse de la matrice de MixColumns qui est :

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} a0 \\ a1 \\ a2 \\ a3 \end{bmatrix}$$

3.1.4 AddRoundKeyInverse

La fonction AddRoundKeyInverse n'a pas été implémenté car elle est son propre inverse, puisqu'elle utilise l'opération XOR (ou exclusif).

3.2 Interface interactive

L'interface interactive permet à l'utilisateur le choix de la fonctionnalité souhaitée : Effectuer un chiffrement, un déchiffrement ou en encore lancer une attaque, et interagit avec toutes les autres fonctions suivant la demande.
Fonctionnalités :

- Choix de l'action à effectuer : 'c' pour chiffrement, 'd' pour déchiffrement et 'a' pour attaque.
- Saisie des données nécessaires : dans le cas où l'utilisateur choisit de lancer un chiffrement/déchiffrement, il lui est demandé de saisir le message ainsi que la clé à utiliser.
- Affichage des résultats.

3.3 Gestion des fichiers

Le fichier **key.txt** contient la clé générée aléatoirement par la fonction **generatekey** utilisée pour l'attaque. Le fichier sera créé s'il n'existe pas déjà.

La fonction **generatekey** du fichier attaque génère une clé grâce à la bibliothèque **secrets** de python qui fournit un accès à la source d'aléa la plus sûre disponible sur le système d'exploitation. Nous l'avons préféré à l'utilisation par défaut du générateur pseudo-aléatoire du module random.

4 Algorithmes utilisés et estimation de la complexité

4.1 Principe et complexité

Le tableau ci-dessous indique le principe général de chaque algorithme ainsi que leur complexité.

4.2 Chiffrement

Algorithme 1 : Cipher(byte in[4*10], byte out[4*10], word w[10*(10+1)])

Entrées : Etat (message clair), clé
Sorties : Message chiffré
Appliquer AddRoundKey(state, w[0, 10-1]);
for *round de 1 à 9* **do**
    Appliquer SubBytes(Etat);
    Appliquer ShiftRows(Etat);
    Appliquer MixColumns(Etat);
    Appliquer AddRoundKey(Etat, w[round*Nb, (round+1)*Nb-1]);
end
Appliquer SubBytes(Etat);
Appliquer ShiftRows(Etat);
Appliquer AddRoundKey(Etat, w[10*Nb, (10+1)*10-1]);
retourner "Message chiffré";

4.2.1 Compléxité

Pour chacun des 10 tours de l'AES, il est nécessaire de faire 4 opérations différentes : SubBytes, ShiftRows, MixColumns et AddRoundKey sauf pour le dernier tour où MixColumns n'est pas appliquée.

Ainsi, pour chiffrer un bloc de données de 128 bits en utilisant la version AES-128, le nombre total d'opérations est :

Nombre total d'opérations = (SubBytes + ShiftRows + MixColumns + AddRoundKey) * 9 + (SubBytes + ShiftRows + AddRoundKey)

— SubBytes : pour chaque octet du bloc de données de 128 bits, une opération de recherche dans la table de substitution S-box est nécessaire. Comme il y'a 16 octets dans le bloc de données, le nombre total d'opérations pour SubBytes est de 16 x 256 = 4 096.
— ShiftRows : la première ligne n'est pas modifiée et les troise autres sont décalées vers la gauche ce qui nous fait donc 3 opérations.
— MixComlumns : On effectue un produit matriciel utilisant chacun des 4 octets d'une colonne. Ce qui nous fait donc 4 X (1 X 4 X 4) opérations.
— AddRoundKey : un XOR est appliqué entre chacun des octets de l'état et de la clé ronde. Ce qui nous fait 4 X 4 = 16 opérations.

Nombre d'opérations par tour = SubBytes + ShiftRows + MixColumns + AddRoundKey = 4 096 + 3 + 64 + 16

Le nombre total d'opérations pour chiffrer un bloc de données de 128 bits en utilisant AES 128 bits avec 10 tours est donc :

(4096 + 3 + 64 + 16) X 9 + (4096 + 3 + 16) = 41 726 opérations au total.

4.3 Déchiffrement (Chiffrement Inverse)

Algorithme 2 : Déchiffrement

Entrées : Etat (message chiffré), clé
Sorties : Message déchiffré en clair
clé ← Expansion de la clé;
message chiffré ← AddRoundKey avec comme arguments le message chiffré et les 4 premieres colonnes de la clé;
message chiffré ← ShiftRowsInverse du message chiffré;
message chiffré ← SubBytesInverse du message chiffré;
for *i de 1 à 9* **do**
    message chiffré ← AddRoundKey avec comme arguments le message chiffré et la clé les colonnes allant de i*4 à i*4+4;
    message chiffré ← MixColumnsInverse du message chiffré;
    message chiffré ← ShiftRowsInverse du message chiffré;
    message chiffré ← SubBytesInverse du message chiffré;
end
message clair ← AddRoundKey avec comme arguments le message chiffré et la clé les 4 dernieres colonnes;
retourner "Message clair";

4.3.1 Complexité

Le message de 256 bits est découpé en blocs de données de 16 (matrice 4 x 4) bits et on le fait passer par :

— AddRoundKey qui fait une operation XOR entre un bloc de données et la clé de tour, qui donne comme nombre d'operations 4 x 4 = 16.
— ShiftRowsInverse : Décale les lignes sauf la première donc le nombre d'operations est de 3.
— SubBytesInverse : Substitue les valeurs du bloc de données avec celle de la Sbox, donc il y en a 16 (car bloc de donnée de taille 4 x 4) x 256 (la Sbox est de taille 16 x 16) = 4096.
— MixColumnsInverse : Multiplication de matrices : MixColumns a 16 x 4 colonnes = 64.

Le nombre total d'opérations pour déchiffrer un bloc de données de 128 bits en utilisant AES 128 bits avec 10 tours est donc de (16 + 3 + 4096) + 9 x (16 + 3 + 4096 + 64) = 41 726 opérations en tout.

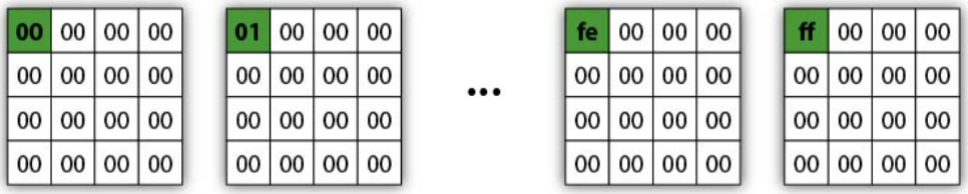
5 Attaque intégrale contre 4 tours de l'AES (Square attaque)

Nous commençons par prendre un ensemble de 256 textes clairs où le premier octet prend les 2^8 = 256 valeurs possibles et qui sont égaux à 0 sur les 15 octets restants.

Le but de l'attaque sur 4 tours de l'AES est de retrouver la clé du dernier tour octet par octet. Pour cela, on inverse partiellement nos calculs et on teste à la fin si la somme des 256 valeurs est bien égale à 0. La clé pour laquelle cette somme s'annule est avec une grande probabilité la bonne clé. On répète ensuite le même processus afin de récupérer les octets restants de la sous-clé.

5.1 Setup

Prend en argument une clé et génére un Δ set à partir de celle-ci. La case en vert représente la valeur active du Δ set.



5.2 CheckKeyGuess

Vérifie que le XOR entre tous les octets est bien égal à zéro en prenant la supposition pour un octet et l'ensemble des octets renvoyés par la fonction reverseState.

5.3 ReverseState

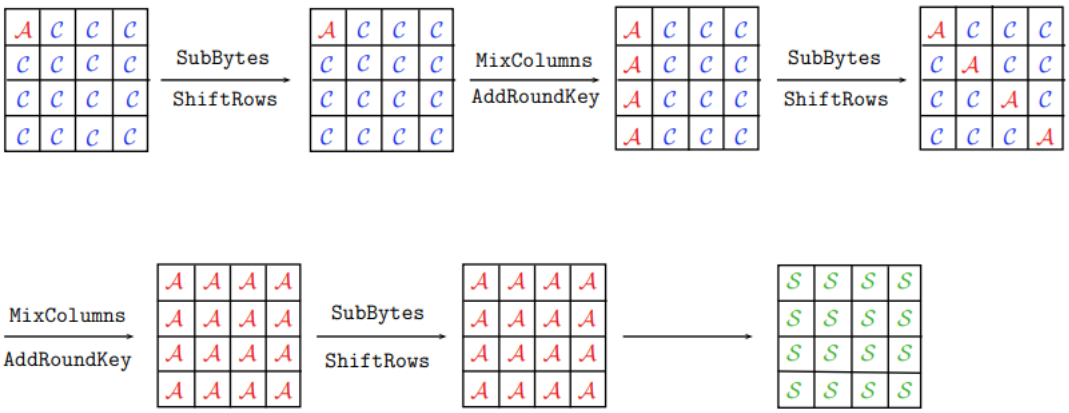
Prend en entrée une estimation d'un octet de la clé, sa position ainsi que le Δ set renvoyée par la fonction setup puis inverse l'octet à la position donnée pour chaque élément du Δ set.

5.4 InvertKeyExpansion

Permet de retrouver la clé originale utilisée pour l'AES à partir de l'indice de la clé ronde retrouvée pour le dernier tour.

5.5 Attaque

Vérifie qu'il y a bien une seule valeur valide d'un octet. S'il y en a plusieurs, on teste chacune des valeurs dans notre liste avec un nouveau Δ set chiffré généré grâce à la fonction setup jusqu'à ce qu'il ne reste plus qu'une seule valeur. Il ne nous reste plus qu'à itérer sur toutes les positions des octets de la dernière clé afin de tous les retrouver. À la fin de la fonction, on appelle InvertKeyExpansion afin d'obtenir la clé utilisée pour notre AES.



6 Conclusion

6.1 Résumé des principales conclusions du projet

En définitive, nous délivrerons un programme capable de chiffrer et déchiffrer avec l'algorithme AES sur 128 bits, mais aussi de faire une attaque sur une version réduite de celui-ci (AES avec 4 tours).

En ce qui concerne l'implémentation, nous avons utilisé le langage Python qui rend aisé la manipulation et le calcul des grands nombres en binaire. La grande variété de fonctions mathématiques et d'algorithmes complexes nous pousse également à choisir le langage procédural afin de faciliter la manipulation de ces fonctions. Il existe de plus des bibliothèques qui peuvent faciliter le travail de codage de l'algorithme AES et des fonctions de chiffrement connexes.

Nous avons utilisé la bibliothèque Numpy afin de manipuler plus facilement les blocs de données en array. Elle nous permet aussi d'appliquer directement certaines opérations de calcul ou de tri sur les blocs de données. Ainsi, nous réduisons la complexité du code en mémoire et optimisons au maximum l'espace de stockage.

6.2 Évaluation de la sécurité de l'AES

L'AES est donc un bon moyen de sécuriser les données car incassable sur 10 tours. Jusqu'à nos jours, le maximum de tours atteint est 8. Nous pourrions cependant renforcer la sécurité en utilisant une clé de 256 bits.

7 Annexes

1. https://www.davidwong.fr/blockbreakers/index.html

2. https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

3. https://christinaboura.files.wordpress.com/2019/11/boura-these-2012.pdf
4. https://numpy.org/doc/stable/index.html