UNIVERSITE TECHNOLOGIQUE DE TROYES

Projet GS15

Super AES – Super El Gamal

Théodore APAPOULLE –Thibault CHATIRON

INTRODUCTION

Le but de ce projet est de créer deux programmes de chiffrement et de déchiffrement : SUPER AES et SUPER EL GAMAL. (Voir sujet de l'énoncé)
Ce projet a été effectué sur Matlab avec l'aide des fonctions sur MuPaD qui reprend les fonctions de Maple. Cela nous a facilité la tache, en effet, pour créer nos polynômes, nous n'avons pas eu besoin de créer une fonction, pour effectuer l'exponentiation modulaire, nous avons utilisé une fonction powermod de Mupad.

SUPER AES

Avant de commencer à créer les différentes fonctions de super AES, nous devons construire le corps GF2^16 = Z/2Z[X] /<P[X]> avec P un polynôme irréductible. Pour cela, il faut créer des algorithmes permettant de trouver un polynôme irréductible de degré 16 et trouver un élément appartenant au corps qui est générateur.

1) Algorithme pour le polynôme irréductible

Soit n le degré de P alors P est irréductible si P divise X^ (P^n) –X et si pour tout d<n divisant n, P ne divise pas X^ (P^d) –X

Tout d'abord, nous avons crée une fonction permettant de générer des polynômes de degré T. Dans notre cas, cela sera des polynômes de degré 16. Ensuite nous avons écrit une fonction qui prend en entrée un polynôme P de degré 16 crée aléatoirement et détermine si P est irréductible dans $F_2[X]$.

2) Algorithme pour le polynôme générateur

Ensuite dès qu'on nous avons trouvé le polynôme irréductible. Il nous faut trouver le polynôme générateur. Pour trouver un élément générateur, il faut trouver un élément primitif. Nous allons créer un polynômede degré inférieur ou égal à 15 avec les fonctions crées précédemment. Ensuite,notons-le a. Cet élément appartient au corps. Les facteurs premiers de 65535 sont 3/5/7/17/257

Si a^(65535)/3 =1 on arrête. Cet élément n'est pas primitif
Sinon Si a^(65535)/5 =1 on arrête .Cet élément n'est pas primitif
Sinon Si a^(65535)/17 =1 on arrête. Cet élément n'est pas primitif
Sinon Si a^(65535)/257 =1 on arrête. Cet élément n'est pas primitif.

Si parmi tous les tests a^(65535)/facteurs premiers de 65535 est différent de 1 alors cette élément un élément primitif et donc générateur.

Maintenant, nous pouvons choisir un polynôme irréductible et un polynôme générateur afin de construire notre corps de Galois.

3) Algorithme pour générer une clé aléatoire

Nous générons plusieurs fois un chiffre aléatoire compris entre 0 et 65535 que l'on retranscrit en code ASCII et que l'on stocke ensuite dans un fichier texte,cela permettra d'obtenir une clé de longueur de 320, 480 ou 640 bits. La longueur de la clé est choisie par l'utilisateur.

4) Chiffrement

Code Matlab

```
fprintf('***************************** \n')
fprintf('*******  Chiffrement  ******* \n')
fprintf('***************************** \n')

taille=size(texte_clair_hexa,1);

tour_texte=ceil(taille/20); %permet de chiffre le texte en son entier

for k=1:tour_texte

fprintf('State initial');
[state] = aes_init(taille,k,texte_clair_hexa) %state initial

fprintf('State apresAddRoundKey');
state = AddRoundKey(state, 0, Nb)

fori=1:Nr-1
fprintf('State au round n°%d apresSubBytes :', i)
      state=SubBytes(state,W)
fprintf('State au round n°%d apresShiftRows :', i)
      state = ShiftRows(state)
fprintf('State au round n°%d apresmixColumns :', i)
      state = MixColumns(state)
fprintf('State au round n°%d apresAddRoundKey :', i)
      state = AddRoundKey(state, i, Nb)
end

fprintf('State au dernier tour apresSubBytes :')
state=SubBytes(state,W)
fprintf('State au dernier tour apresShiftRows :')
 state=ShiftRows(state)
fprintf('State final :')
 state = AddRoundKey(state,Nr,Nb)

afficher_texte_chiffre(state,k);

end
```

On commence par initialiser notre « state », c'est-à-dire que l'on retranscrit le texte par bloc de 16 bits que l'on convertit en décimal. Le texte en décimal alimentera une matrice, ce sera notre state.

Ensuite, nous effectuerons un AddRoundKey.

Selon la longueur de la clé, on fera 10, 12 ou 14 tours.

Les différentes étapes dans la boucle sont les fonctions suivantes :

- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey

La dernière étape consiste à effectuer les fonctions SubBytes, ShiftRows et AddRoundKey.

Finalement, on pourra retranscrire notre state en fichier texte.

5) AddRoundKey

Cette fonction consiste à effectuer un xor entre une clé et notre state. Cette clé changera au cours de l'étape de chiffrement et de déchiffrement à l'aide de la fonction KeyExpansion.

6) KeyExpansion

Cette fonction permet de « changer » de clé pour chaque AddRoundKey. Elle effectuera un certain nombre de fois la fonction ShiftRows.

7) ShiftRows

ShiftRows fonctionne sur les lignes de notre state. Elle déplace de manière cyclique les éléments de chaque ligne.
La première ligne est inchangée. Chaque élément de la deuxième ligne est décalé d'une colonnevers la gauche. De même, chaque élément des troisièmes, quatrièmes et cinquièmeslignes sont décalés de deux, trois et quatre colonnes vers la gauche.

8) SubBytes

Cette fonction consiste à multiplier chaque élément inverse de notre state par la matrice A. On effectue ensuite un xor de ce résultat avec le vecteur c.

Avec :

A= $\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$

et

c= $\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$

9) MixColumns

Le polynôme irréductible choisi est X^16+X^5+X^3+X^2+1.

Nous avons choisi d'implémenter la matrice X

```
'0002'   '0004'   '0001'   '0002'   '0002'
'0002'   '0002'   '0004'   '0001'   '0002'
'0002'   '0002'   '0002'   '0004'   '0001'
'0001'   '0002'   '0002'   '0002'   '0004'
'0004'   '0001'   '0002'   '0002'   '0002'
```

A cette étape, chaque colonne est multipliée par la matrice X.

10) Déchiffrement

Code Matlab

```
fprintf('****************************** \n')
fprintf('******* Déchiffrement ******* \n')
fprintf('****************************** \n')

taille=size(texte_chiffre_hexa,1);

tour_texte=ceil(taille/20); %permet de chiffrer le texte en son entier

for k=1:tour_texte

fprintf('State initial');
[state] = aes_init(taille,k,texte_chiffre_hexa) %state initial

fprintf('State apresAddRoundKey');
state = AddRoundKey(state, Nr, Nb)

fprintf('State apresInvShiftRows');
state=InvShiftRows(state)

fprintf('State apresInvSubBytes :')
state=InvSubBytes(state,W)

fori=Nr-1:-1:1
fprintf('State au round n°%d après AddRoundKey :',Nr-i)
    state = AddRoundKey(state,i, Nb)
fprintf('State au round n°%d après InvMixColumns :',Nr-i)
    state = InvMixColumns(state)
fprintf('State au round n°%d après InvShiftRows :',Nr-i)
    state = InvShiftRows(state)
fprintf('State au round n°%d après InvSubBytes :',Nr-i)
    state = InvSubBytes(state,W)
end

fprintf('State final :')
 state = AddRoundKey(state,0,Nb)

afficher_texte_dechiffre(state,k);
```

```
end
```

On commence par initialiser notre « state », c'est-à-dire que l'on retranscrit le texte par bloc de 16 bits que l'on convertit en décimal. Le texte en décimal alimentera une matrice, ce sera notre state.

Ensuite, nous effectuerons un AddRoundKey, un InvShiftRows et un InvSubBytes.

Selon la longueur de la clé, on fera 10, 12 ou 14 tours.

Les différentes étapes dans la boucle sont les fonctions suivantes :

- AddRoundKey
- InvMixColumns
- InvShiftRows
- InvSubBytes

La dernière étape consiste à effectuer la fonctionAddRoundKey.

Finalement, on pourra retranscrire notre state en fichier texte et vérifier que le message déchiffré est bien le même que le message initial envoyé.

11) InShiftRows

ShiftRows fonctionne sur les lignes de notre state. Elle déplace de manière cyclique les éléments de chaque ligne.
La première ligne est inchangée. Chaque élément de la deuxième ligne est décalé d'une colonnevers la droite. De même, chaque élément des troisièmes, quatrièmes et cinquièmeslignes sont décalés de deux, trois et quatre colonnes vers la droite.

12) InvMixColumns

Nous avons choisi d'implémenter la matrice Xprim

```
Xprim=[{'51EB' '3ABE' '8F02' 'D5DC' '8741'};...
       {'8741' '51EB' '3ABE' '8F02' 'D5DC'};...
       {'D5DC' '8741' '51EB' '3ABE' '8F02'};...
       {'8F02' 'D5DC' '8741' '51EB' '3ABE'};...
       {'3ABE' '8F02' 'D5DC' '8741' '51EB'}];
```

A cette étape, chaque colonne est multipliée par la matrice Xprim.

13) InvSubBytes

On effectue un xor du state avec le vecteur cprim. Ensuite, on multiplie chaque élément inverse de notre résultat par la matrice Aprim.

Avec :

Aprim= $\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$

et

cprim=c=$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$

SUPER EL GAMAL

Pour effectuer les différents algorithmes suivants, nous avons repris les fonctions utilisées dans **SUPER AES.**

- Algorithme de chiffrement d'El Gamal

Il faut choisir un polynôme irréductible de degré 16, un polynôme générateur et une clé secrète x choisi aléatoirement entre 0 et 65535. Ensuite nous avons appliqué l' algorithme de l'énoncé du sujet.

- Algorithme de création de la signature

Soit la clé publique [P,g,y] avec P polynôme irréductible, g polynôme générateur, y=g^x modulo P
M étant un bloc de message en décimal
On choisit un k aléatoire compris entre 0 et 65534, il faut qu'il soit premier avec 65535
Ensuite on pose r= g^k modulo P
On convertit r en décimal.
On trouve l'inverse de k modulo 65535 avec les fonctions crées
Ensuite on pose s=k^-1(M-a*r) modulo 65535
On a le couple [r,s]

- Algorithme de déchiffrement d'El Gamal

Nous avons appliqué le même algorithme de l'énoncé du sujet.

- Algorithme de vérification de la signature

Pour vérifier la signature il faut que **y^r*r^s = g^M [P]**