UE IN608 CRYPTO :

Attaque intégrale sur une version réduite d'AES-128

COMPTE RENDU

ACHERIR Mohand Arezki

AMMOUR Mahdi

AMRANE Younes

BUAUD Benjamin

FEURGARD Martin

TEKLAL Massil

TABLE DES MATIERES

Compilation et utilisation :

Chiffrement

Pour utiliser la fonction de chiffrement, la commande sur le terminal est un simple :

make

Cette commande compile les fichiers sources et exécute le programme.

Cela requiert :

→ la taille du texte clair à chiffrer *, représenté par un entier

→ le texte clair en hexadécimal, des couples d'octets non séparés par des espaces.

Exemple : un texte de taille 4 = "61626364", sans les guillemets

→ la clé maître avec le même format que le texte clair,

→ le nonce avec le même format que le texte clair.

A la fin du programme cela renvoie le chiffrement du texte clair en mode CTR ainsi qu'en mode OFB.

(*) Si la taille du texte clair n'est pas multiple de 16, on utilise une matrice de taille égale au multiple de 16 le plus proche, le reste est remplie par un padding de 0x00.

Attaque

Pour utiliser la fonction d'attaque il suffit d'entrer la commande suivante dans le terminal :

make attaque

Cette commande compile les fichiers sources et exécute le programme qui demande :

→ la clé maître en hexadécimal (des couples d'octets non séparés par des espaces.

Exemple : un texte de taille 4 = "61626364", sans les guillemets

Il en résulte l'affichage de la 4ème sous-clé calculée K4, et la clé finale qui est forcément égale à la clé maître fournie.

1. Implémentation de l'AES à 10 tours :

1.1. Chiffrement d'un seul bloc à 128 bits

```
AES_128(cle_maitre, matrice_etat):
        générer les sous-clés ki
        matrice_etat ^= cle_maitre
        faire 9 fois:
                Tour(matrice_etat, ki)
        DernierTour(matrice_etat, k(dernière))


Tour(matrice_etat, cle_tour):
        SubOctet(matrice_etat)
        DecaleLignes(matrice_etat)
        MelangeColonnes(matrice_etat)
        AjoutCleTour(matrice_etat, cle_tour)
```

Les fonctions *SubOctet*, *DecaleLignes* et *AjoutCleTour* sont implémentées de la même manière qu'elles sont décrites dans les slides du sujet.

La fonction *MelangeColonnes* est similaire à celle présentée dans les slides, mais avec la permutation suivante en plus :

```
unsigned char a00 = matrice_etat[0][0];
unsigned char a01 = matrice_etat[0][1];
unsigned char a02 = matrice_etat[0][2];
unsigned char a03 = matrice_etat[0][3];

unsigned char a10 = matrice_etat[1][0];
unsigned char a11 = matrice_etat[1][1];
unsigned char a12 = matrice_etat[1][2];
unsigned char a13 = matrice_etat[1][3];

unsigned char a20 = matrice_etat[2][0];
unsigned char a21 = matrice_etat[2][1];
unsigned char a22 = matrice_etat[2][2];
unsigned char a23 = matrice_etat[2][3];

unsigned char a30 = matrice_etat[3][0];
unsigned char a31 = matrice_etat[3][1];
unsigned char a32 = matrice_etat[3][2];
unsigned char a33 = matrice_etat[3][3];

Matrice_Etat[0][0] = a00;
Matrice_Etat[0][1] = a10;
Matrice_Etat[0][2] = a20;
Matrice_Etat[0][3] = a30;
```

```
Matrice_Etat[1][0] = a13;
Matrice_Etat[1][1] = a23;
Matrice_Etat[1][2] = a33;
Matrice_Etat[1][3] = a03;

Matrice_Etat[2][0] = a22;
Matrice_Etat[2][1] = a32;
Matrice_Etat[2][2] = a02;
Matrice_Etat[2][3] = a12;

Matrice_Etat[3][0] = a31;
Matrice_Etat[3][1] = a01;
Matrice_Etat[3][2] = a11;
Matrice_Etat[3][3] = a21;
```

1.2. Modes de chiffrement CTR / OFB:

Le mode CTR est implémenté par la fonction suivante :

```
void CTR(unsigned char Matrice_Etat[][4], unsigned char Cle_Maitre[16], unsigned char Nonce[4][4]) {

for(int i = 0 ; i < nbLignes ; i=i+4) {
        unsigned char X[4][4];
        for(int x = 0 ; x < 4 ; x++) {
                for(int y = 0 ; y < 4 ; y++) {
                        X[x][y] = Nonce[x][y];      // Stocke la Nonce dans X pour ne pas le modifier
                }
        }
}}

AES_128(X, Cle_Maitre){          // Chiffre le bloc de 128 bits X

for(int x = 0 ; x < 4 ; x++) {
        for(int y = 0 ; y < 4 ; y++) {
                Matrice_Etat[i+x][y] = Matrice_Etat[i+x][y] ^ X[x][y]; // Addition
        }
}
Inc_Nonce(Nonce);                // Incrémente le Nonce
}}
```

Le mode OFB est implémenté par la fonction suivante:

```
void OFB(unsigned char Matrice_Etat[][4],unsigned char Cle_Maitre[16],unsigned char IV[4][4]) {

unsigned char X[4][4];
for(int i = 0 ; i < 4 ; i++) {
        for(int j = 0 ; j < 4 ; j++) {
                X[i][j] = IV[i][j];
        }
}
for(int i = 0 ; i < nbLignes ; i=i+4) {
        AES_128(X, Cle_Maitre);
        for(int x = 0 ; x < 4 ; x++) {
                for(int y = 0 ; y < 4 ; y++) {
                        Matrice_Etat[i+x][y] = Matrice_Etat[i+x][y] ^ X[x][y];
                        X[x][y] = Matrice_Etat[i+x][y];
                }
        }
}}
```

2. Attaque sur l'AES réduit à 4 tours :

2.1. Calcul de K4 :

L'idée est d'effectuer une recherche exhaustive sur les 256 possibilités, pour chacun des 16 octets de la matrice de la 4ème sous-clé. Et pour chaque possibilité, on calcule en utilisant les matrices D (chiffrées avec l'AES réduit à 4 tours) la case correspondante dans les matrices C (chiffrées par 3 tours complets).

Après avoir calculé les octets de la case correspondante pour les 256 matrices C, on fait le

XOR entre ceux-ci et si l'on obtient 0 alors la recherche exhaustive aura trouvé un octet de la 4ème sous-clé.

On est sûr que toutes les cases de la sous-clé 4 sont calculées, car les "cases correspondantes" mentionnées au-dessus sont déduites avec les fonctions *DecaleLignesXY* et *permXY*, qui sont bijectives sur $\{0, 1, 2, 3\}^2$ dans $\{0, 1, 2, 3\}^2$.

2.2. Déduction de K0, .ie la clé-maitre principale :

La fonction *ExtensionCle1* (équivalente à *ExtensionCle* dans l'AES réduit à 4 tours) calcule les sous-clés 1, 2, 3 et 4, ligne par ligne en commençant par la ligne 4 et finissant à la ligne 19, sachant que les lignes de 0 à 3 représentent la clé maître.

Enfin, pour déduire dans notre cas la clé maitre à partir de la sous-clé 4 (les lignes de 16 à 19), on fait l'opération inverse : *ExtensionCle*^(-1), en commençant par la ligne 15 et en descendant jusqu'à la ligne 0, en suivant l'instruction suivante :

cle_etendue[i-4] = cle_etendue[i] ^ tampon pour i= 19, …, 4

Et par invariance, on conclut qu'après l'itération i = 4, on aura calculé les lignes 0, 1, 2 et 3. Donc on aura trouvé la clé maître.