

Sécurité des systèmes informatiques
Le chiffrement symétrique : AES

Olivier Markowitch

AES

Algorithme de chiffrement symétrique itératif

Le nombre de tours dépend de la longueur de la clé

Pour des blocs de 128 bits nous avons :

- une clé de 128 bits → 10 tours
- une clé de 192 bits → 12 tours
- une clé de 256 bits → 14 tours

Nous ne considérerons que des blocs et des clés de 128 bits

State

State est la structure qui contient les résultats intermédiaires

C'est un tableau de 4×4 octets :

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix}$$

Une telle structure a toujours 4 lignes et le nombre de colonnes égale la longueur de la clé divisée par 32

Assignment

L'assignation $state = x$ est réalisée ainsi :

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \leftarrow \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix}$$

où x_i indique le i^{me} octet de x

ByteSub

ByteSub est une substitution non linéaire appliquée sur un octet

Chaque octet de state est transformé suivant une Sbox

La Sbox a une explication algébrique

Soit z l'octet à modifier

```
byte ByteSub(byte z)
{
    if(z!=0)
        z=z^-1 dans GF(2^8)
    c=011000111
    for(i=0;i<8;++i)
        b[i]=z[i]+z[i+4]+z[i+5]+
            z[i+6]+z[i+7]+c[i] mod 2
    return(b)
}
```

ShiftRow

State est réorganisé ainsi :

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

MixColumn

On manipule les colonnes de state de la manière suivante :

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

où les éléments de la matrice sont exprimés en hexadécimal et où s_i représente la i^{me} colonne de state

Gestion de clés

Pour chaque tour, une clé (la roundkey) est dérivée de la clé secrète

L'opération « AddRoundKey » correspond à un xor bit à bit entre state et la roundkey

State et la roundkey sont exprimés sous forme de deux matrices 4×4 d'octets

Les octets de deux cellules situées en même position, disons i et j , dans state et dans la roundkey sont « xorés » bit à bit pour former la cellule résultat de coordonnées i et j

Gestion des clés (suite)

Le nombre de bits de l'ensemble des roundkeys est égal à la longueur d'un bloc multiplié par le nombre de tours plus un

Pour 128 bits cela donne : $128 \times (10+1) = 1408$ bits

La clé secrète est tout d'abord étendue en une expandedkey, grâce à l'algorithme KeyExpansion et les roundkeys sont extraites de cette clé étendue

Le résultat de l'algorithme d'expansion est composé de 44×32 bits = 1408 bits, notés $w[0] \dots w[43]$

Dans l'algorithme d'expansion, $key[i]$ correspond au i^{me} octet de la clé secrète

Gestion des clés (suite)

$$\text{Rotword}(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0)$$

où B_i est un octet

$$\text{Subword}(B_0, B_1, B_2, B_3) = (B'_0, B'_1, B'_2, B'_3)$$

où B'_i correspond au résultat de la Sbox appliquée à l'octet B_i

Le chiffrement

```
void AES(state &,key)
{
    KeyExpansion(key,expandedkey)
    AddRoundKey(state,expandedkey)
    for(i=1;i<10;++i)
    {
        ByteSub(state)
        ShiftRow(state)
        MixColumn(state)
        AddRoundKey(state,expandedkey+4*i)
    }
    ByteSub(state)
    ShiftRow(state)
    AddRoundKey(state,expandedkey+4*10)
}
```

Le chiffré est alors dans *state*

Le déchiffrement

```
void InvAES(state &,key)
{
    KeyExpansion(key, expandedkey)
    AddRoundKey(state, expandedkey+4*10)
    for(i=9;i>=1;i=i-1)
    {
        InvShiftRow(state)
        InvByteSub(state)
        AddRoundKey(state, expandedkey+4*i)
        InvMixColumn(state)
    }
    InvShiftRow(state)
    InvByteSub(state)
    AddRoundKey(state, expandedkey)
}
```

Le message clair est alors dans *state*

InvShiftRow réalise les rotations circulaires inverses des rotations circulaires gauches réalisées dans ShiftRow

AddRoundKey est son propre inverse