COURS ID12 « INTRODUCTION À LA CRYPTOLOGIE »
COURS 4
CHIFFREMENT SYMÉTRIQUE : AES

PAUL ZIMMERMANN (CM), LAURENT FOUSSE (TD)

2. Cryptologie symétrique : DES et AES

2.1. **DES.** Adopté en 1977, DES (*Data Encryption Standard*) prend en entrée un texte clair de 64 bits, une clé de 64 bits également (dont 8 bits de parité, donc seuls 56 sont utilisables), et retourne un texte chiffré de 64 bits. Le déchiffrement se fait avec la même clé, mais en renversant l'ordre des « boites ».

Le tableau ci-dessous résume les meilleures attaques connues contre DES :

attaque	paires connues/choisies	mémoire	temps
précalcul	1	2^{56}	1
recherche exhaustive	1	1	2^{55}
cryptanalyse linéaire	2^{43}	textes	2^{43}
cryptanalyse diff.	2^{47}	textes	2^{47}

2.2. **AES.** AES (*Advanced Encryption Standard*) est le successeur de DES, qui commençait à être dépassé.

Il y a eu en fait un appel à propositions pour AES, en deux phases de 1998 à 1999. La proposition retenue fut celle appelée Rijndael, inventée par deux belges, Joan Daemen et Vincent Rijmen. Il y a quelques petites différences entre Rijndael et AES. La taille de bloc (*block length*) est la taille des entrées, sorties et des calculs intermédiaires. La taille de clé (*key length*) est comme son nom l'indique la taille de la clé.

Rijndael considère une taille de bloc et une taille de clé pouvant être fixées à un multiple quelconque de 32 bits, avec un minimum de 128 et un maximum de 256, la taille de bloc pouvant différer de celle de la clé. Au contraire, AES fixe la taille du bloc à 128 bits, la clé pouvant faire 128, 192 ou 256 bits seulement.

Nous décrivons ici la version d'AES-128 (clé de 128 bits).

2.3. **AES-128.** Un *état* d'AES est un bloc de 128 bits, soit 16 octets, qu'on peut représenter par un tableau 4×4 :

$$\begin{array}{|cccc|}\hline a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \\ \hline\end{array},$$

que l'on notera aussi de façon linéaire :

$$a_0 a_1 a_2 a_3 \dots a_{12} a_{13} a_{14} a_{15},$$

où les a_i sont des octets notés par deux chiffres hexadécimaux.

Nous illustrerons les différentes phases avec l'exemple suivant : soit le texte clair

$$i = \texttt{3243f6a8885a308d313198a2e0370734}$$

Date: zimmerma@loria.fr,laurent@komite.net.

et la clé

$$k = \texttt{2b7e151628aed2a6abf7158809cf4f3c}$$

représentés par le tableau :

32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

Avant de décrire en détail AES, il faut quelques préliminaires mathématiques.

2.4. **Arithmétique dans** GF(2^8)**.** L'objet de base pour AES est l'élément du corps GF(2^8). Ce corps contient 2^8 éléments. Un élément de GF(2^8) peut être représenté par un polynôme de degré au plus 7, à coefficients dans GF(2), le corps à deux éléments. Un tel polynôme s'écrit donc

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0,$$

où $b_i \in \{0, 1\}$, et on peut le stocker en mémoire par le nombre binaire $(b_7b_6b_5b_4b_3b_2b_1b_0)_2$, que l'on notera aussi sous forme hexadécimale. Par exemple, la valeur hexadécimale `57` correspond à $(01010111)_2$, ou encore au polynôme

$$x^6 + x^4 + x^2 + x + 1.$$

2.4.1. *Addition dans* GF(2^8). La somme de deux éléments de GF(2^8) correspond à la somme des polynômes correspondants, les coefficients étant réduits modulo 2. Par exemple

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2,$$

ou en notation binaire : $(01010111)_2 + (10000011)_2 = (11010100)_2$, ou encore en hexadécimal `57` + `83` = `D4`. Il est clair que l'addition peut s'effectuer efficacement au niveau des octets via le « ou exclusif ».

2.4.2. *Multiplication dans* GF(2^8). Le produit de deux éléments de GF(2^8) correspond au produit des polynômes correspondants, modulo un polynôme irréductible $m(x)$ — fixé au préalable — de degré 8 sur GF(2). Pour AES, ce polynôme est $m(x) = x^8 + x^4 + x^3 + x + 1$, ou encore `11B` en hexadécimal :

```
> m := x^8 + x^4 + x^3 + x + 1:
> Factor(m) mod 2;
                          8    4    3
                         x  + x  + x  + x + 1
```

Par exemple, la multiplication de `57` par `83` donne :

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 = (x^5 + x^3)m + x^7 + x^6 + 1.$$

Il n'y a pas de moyen simple de calculer ce produit au niveau des octets. La meilleure méthode procède par additions et décalages.

2.4.3. *Inverse dans* GF(2^8). Tout élément non nul $p(x)$ de GF(2^8) admet un inverse. Celui-ci peut se calculer via l'algorithme d'Euclide étendu, qui détermine deux polynômes $u(x)$ et $v(x)$ tels que :

$$u(x)p(x) + v(x)m(x) = 1,$$

avec $\deg u(x) < \deg m(x)$. L'élément $u(x)$ est l'inverse de $p(x)$. Par exemple, l'inverse de $x^6 + x^4 + x^2 + x + 1$ est $x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$:

```
> p := x^6 + x^4 + x^2 + x + 1:
> Gcdex (p, m, x, 'u') mod 2:
> u;
                          7    5    4    3    2
                         x  + x  + x  + x  + x  + x + 1
```

ce que l'on notera aussi : l'inverse de `57` est `BF`.

2.4.4. *Multiplication par x dans* $\mathrm{GF}(2^8)$. La multiplication par x est un cas particulier de la multiplication. Soit $b(x) = b_7x^7 + \cdots + b_0$, alors $xb(x) = b_7x^8 + \cdots + b_0x$. Soit b_7 est nul, et $xb(x)$ est déjà réduit modulo $m(x)$, soit $b_7 = 1$, et il suffit de retrancher — ou d'ajouter puisqu'on est sur $\mathrm{GF}(2)$ — $m(x)$. Au niveau des octets, cette opération s'implante facilement par un décalage vers la gauche suivi d'une soustraction éventuelle de $m(x) - x^8$ soit `1B`.

AES considère en fait des polynômes de degré < 4 sur $\mathrm{GF}(2^8)$, chaque colonne de 4 octets d'un état (ou d'une clé) correspondant à un tel polynôme (le coefficient de degré i en ligne i).

2.5. **Chiffrement par AES.** Le chiffrement s'effectue de la manière suivante :

(1) À chaque taille de clé correspond un nombre N d'étapes, $10 \leq N \leq 14$. Pour AES-128, on a $N = 10$.

(2) La clé est « étendue » de 4 colonnes (ou mots de 32 bits) à $4(N + 1) = 44$ colonnes, ce qui fait $N + 1 = 11$ clés de session différentes.

(3) La première clé de session (la clé initiale dans le cas où bloc et clé sont de même taille) est « xor-ée » à l'état courant.

(4) On effectue N étapes de chiffrement comme suit — sauf la dernière qui ne comporte pas de `MixColumn` — où `AddRoundKey` correspond au xor avec la clé de session courante :

```
void Round (state_t State, unsigned char *RoundKey)
{
  ByteSub (State);
  ShiftRow (State);
  MixColumn (State);
  AddRoundKey (State, RoundKey);
}
```

(5) Le texte chiffré correspond à l'état courant obtenu au final.

2.5.1. *L'opération* `ByteSub`. Cette opération agit indépendamment sur chaque octet. Il suffit donc de décrire son action sur un élément X de $\mathrm{GF}(2^8)$. On commence par prendre l'inverse de X dans $\mathrm{GF}(2^8)$ — $X = 0$ reste inchangé — puis étant donnés $x_7, \ldots, x_0$ les coefficients de cet inverse, on applique la transformation affine ci-dessous :

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

et on reforme l'octet $(y_7 \ldots y_0)_2$. Par exemple, partant de `00`, on obtient `63`. Partant de `57`, l'inverse est `BF`, et on obtient `5B`.

2.5.2. *L'opération* `ShiftRow`. Cette opération consiste simplement à décaler cycliquement les lignes d'un bloc, chaque ligne étant décalée d'une valeur différente, ne dépendant que de la taille du bloc. La ligne 0 (coefficients constants) est toujours inchangée. Pour AES-128 (bloc de 4 mots de 32 bits, soit 16 octets), la ligne 1 est décalée de 1 octet vers la gauche, la ligne 2 de 2 octets vers la gauche, la ligne 3 de 3 octets vers la gauche :

$$\begin{array}{|cccc|} \hline a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \\ \hline \end{array} \Longrightarrow \begin{array}{|cccc|} \hline a_0 & a_4 & a_8 & a_{12} \\ a_5 & a_9 & a_{13} & a_1 \\ a_{10} & a_{14} & a_2 & a_6 \\ a_{15} & a_3 & a_7 & a_{11} \\ \hline \end{array}$$

2.5.3. *L'opération* `MixColumn`. Cette opération, quant à elle, agit sur les colonnes. Elle multiplie chaque colonne, considérée comme un polynôme de degré 3 à coefficients dans $\mathrm{GF}(2^8)[y]$, par le polynôme $3y^3 + y^2 + y + 2$, et réduit le produit modulo $y^4 + 1$.

2.5.4. *L'expansion de clé.* Pour AES-128, cette opération transforme une clé de 4 mots de 32 bits (128 bits) en 11 clés de 4 mots (soit 10 supplémentaires, la première étant celle de départ). L'ensemble de ces clés est concaténé en 44 mots. Le mot i est produit par « ou exclusif » entre le mot $i-4$ et le mot $i-1$, ce dernier ayant subi la transformation suivante lorsque i est multiple de 4 :

```
temp = W[i-1];
temp = ByteSub(RotByte(temp)) ^ R[i / 4];
```

où `ByteSub` est l'opération agissant au niveau des octets déjà décrite ci-dessous, `RotByte` remplace le mot (a, b, c, d) par (b, c, d, a), et $R[i] = (x^i, 0, 0, 0)$, où x^i désigne l'octet correspondant au polynôme x^i de $\mathrm{GF}(2^8)$.

2.5.5. *Exemple.* En partant du texte clair i et de la clé k donnés plus haut, on obtient après le premier xor $i \oplus k$:

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

puis respectivement après `ByteSub`, `ShiftRow`, `MixColumn` et `AddRoundKey` avec la deuxième clé de session :

d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

Le texte chiffré obtenu est finalement :

39	02	dc	19
25	dc	11	6a
84	09	85	0b
1d	fb	97	32

Pour déchiffrer, il suffit d'inverser le procédé : l'expansion de clé est identique mais on les considère dans l'ordre inverse, on commence par inverser l'étape finale (sans `MixColumn`), et on fait le xor avec la clé initiale à la fin. L'inverse de `AddRoundKey` est lui-même, pour `MixColumn` on multiplie par le polynôme inverse dans $\mathrm{GF}(2^8)[y]$ de $3y^3 + y^2 + y + 2$ modulo $y^4 + 1$, soit $11y^3 + 13y^2 + 9y + 14$. Pour `ShiftRow`, on décale cycliquement les lignes dans l'autre direction. Enfin pour `ByteSub` il suffit de calculer la transformation affine inverse.

Exercice : avec la clé k ci-dessus, trouver le chiffré du texte `0123456789abcdef0123456789abcdef`.