AES: Criptosistema Rijndael

Rijndael és un criptosistema de xifrat de bloc de clau secreta. Dissenyat per Joan Daemen i Vincent Rijmen, va ser elegit pel NIST (National Institute of Standards and Technology) del Estats Units com a estàndard de xifrat de bloc, l'AES (Advanced Encryption Standard, FIPS 187), en substitució del DES. En aquest criptosistema, tant la longitud de la clau com la dels blocs de text a xifrar pot ser de 128, 192 o 256 bits. Finalment, l'estàndard ha fixat la longitud del bloc en 128 bits, deixant la possibilitat d'escollir el tamany de la clau entre els tres valors esmentats.

El cos finit $GF(2^8)$

Els elements d'aquest cos són els **bytes**. Els expressaren en forma binària, hexadecimal o polinòmica, segons convingui.

El byte $\mathtt{b}_7\mathtt{b}_6\mathtt{b}_5\mathtt{b}_4\mathtt{b}_3\mathtt{b}_2\mathtt{b}_1\mathtt{b}_0$ serà el polinomi $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$.

Per exemple, `01010111=0x57` serà $x^6 + x^4 + x^2 + x + 1$.

Suma

La suma de dos elements del cos és la suma de polinomis binaris. Per exemple, `01010111+10000011` serà

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 = \mathtt{11010100}$$

Es correspon amb la operació XOR, que es denotarà $\oplus$. L'element neutre de la suma és `00000000=0x00`.

Multiplicació

Per fer el producte de dos elements del cos cal fer el producte de polinomis binaris i després prendre el residu de la divisió per $\mathbf{m} = \mathbf{x^8} + \mathbf{x^4} + \mathbf{x^3} + \mathbf{x} + \mathbf{1}$. Per exemple,

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) &= x^{13} + x^{11} + x^9 + x^8 + x^7 + \\ &\quad x^7 + x^5 + x^3 + x^2 + x + \\ &\quad x^6 + x^4 + x^2 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1\end{aligned}$$

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \pmod{x^8 + x^4 + x^3 + x + 1} = x^7 + x^6 + 1.$$

L'element neutre de la multiplicació és `00000001=0x01`.

A $GF(2^8)$, tot element diferent del `0x00`, té invers multiplicatiu. L'invers del polinomi a és l'únic polinomi b tal que

$$a\,b = 1 \mod m.$$

Es pot calcular usant l'algorisme d'Euclides estès.

També podem escriure els elements diferents del `0x00` com a potència d'un generador. Per exemple, si $g = x + 1 = \mathtt{00000011} = \mathtt{0x03}$, llavors

$$GF(2^8) = \{g, g^2, \ldots, g^{254}, g^{255}(= g^0 = 1)\} \cup \{0\}$$

El producte de dos elements $a = g^i$ i $b = g^j$, diferents de `0x00`, és $ab = g^i g^j = g^{i+j}$, i l'invers de a és $a^{-1} = (g^i)^{-1} = g^{-i} = g^{255-i}$. En aquest cas, la multiplicació i el càlcul de l'invers es redueixen a la cerca en una taula de 255 elements.

Entrada del missatge i de la clau

Convertim el text en clar en una llista de bits de longitud múltiple de 128, la qual es trenca en blocs de longitud 128. Els bits de cada bloc es posen en una matriu, que anomenem **estat**. L'estat té 4 files i 4 columnes. Omplim l'estat *per columnes* de manera que en cada casella hi hagi un byte. Així, si el bloc és la llista de bytes $a_{00}a_{10}a_{20}a_{30}a_{01}a_{11}a_{21}a_{31}\dots a_{23}a_{33}$, l'estat és:

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Anàlogament, els bits de la clau es van col.locant en una matriu de 4 files i **Nk** columnes. El nombre de columnes Nk serà 4, 6 o 8, depenent de si la longitud de la clau és 128, 192 o 256, respectivament. Per exemple, per a una clau de longitud 192 tindrem Nk = 6 i si la clau és la llista de bytes $k_{00}k_{10}k_{20}k_{30}k_{01}\dots k_{25}k_{35}$, aleshores la matriu és:

clau =

k_{00}	k_{01}	k_{02}	k_{03}	k_{04}	k_{05}
k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
k_{20}	k_{21}	k_{22}	k_{23}	k_{24}	k_{25}
k_{30}	k_{31}	k_{32}	k_{33}	k_{34}	k_{35}

Algoritme de xifrat

Consisteix en una transformació inicial seguida de **Nr** tombs. El nombre de tombs depèn de de Nk:

Nk	4	6	8
Nr	10	12	14

A partir de la clau K es construeixen Nr + 1 subclaus K_i, obtingudes mitjançant un procés d'expansió (KeyExpansion) que explicarem més endavant. Aquestes subclaus seran matrius 4×4. La primera s'usa en la transformació inicial del bloc i la resta, una en cada tomb.

Sigui M el bloc del missatge a xifrar, representat en bits i en forma d'estat. L'algorisme de xifrat segueix l'esquema següent:

estat = AddRoundKey(M, $\mathrm{K_0}$)
Per i=1 **fins** Nr − 1
  estat = $\mathrm{Tomb_i}$ (estat, $\mathrm{K_i}$)
FiPer
C = Tombfinal (estat, $\mathrm{K_{Nr}}$)

on C és el missatge xifrat. L'esquema dels tombs $\mathrm{Tomb_i}$ és el següent, per a $\mathrm{i} = 1, ..., \mathrm{Nr} - 1$:

estat = ByteSub(estat)
estat = ShiftRow(estat)
estat =MixColumn(estat)
estat = AddRoundKey(estat, $\mathrm{K_i}$)

I per a Tombfinal:

estat = ByteSub(estat)
estat = ShiftRow(estat)
estat = AddRoundKey(estat, $\mathrm{K_{Nr}}$)

La funció **AddRoundKey** s'aplica a dues matrius 4×4 i consisteix a fer el seu XOR:

$$A \oplus B = \big(a_{ij} \oplus b_{ij}\big)_{i,j}$$

ByteSub

Aquesta funció opera byte a byte, realitzant les accions següents:
1) substituir cada byte diferent de `0x00` pel seu invers a $GF(2^8)$
2) aplicar la transformació lineal següent, tenint en compte que les operacions es fan mòdul 2: el byte $\mathtt{b}_7\mathtt{b}_6\mathtt{b}_5\mathtt{b}_4\mathtt{b}_3\mathtt{b}_2\mathtt{b}_1\mathtt{b}_0$, es transforma en el byte $\mathtt{u}_7\mathtt{u}_6\mathtt{u}_5\mathtt{u}_4\mathtt{u}_3\mathtt{u}_2\mathtt{u}_1\mathtt{u}_0$, on

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

ShiftRow

La funció ShiftRow fa rotar cada fila de l'estat un nombre determinat de posicions cap a l'esquerra: la fila superior es deixa igual, la segona es desplaça una posició, la tercera dues posicions i la quarta tres posicions. És a dir, comptant de 0 a 3, la fila i rota i posicions cap a l'esquerra.

MixColumn

En aquesta funció les operacions es fan columna a columna, considerant cada columna com a polinomi amb coeficients en el cos $GF(2^8)$. La funció consisteix a multiplicar cada polinomi pel polinomi $c(Y) = \mathtt{0x03}Y^3 + \mathtt{0x01}Y^2 + \mathtt{0x01}Y + \mathtt{0x02}$ i prendre després el residu de la divisió per $Y^4 + 1$. Aquesta funció es pot calcular matricialment: si $(v_{0i},\ v_{1i},\ v_{2i},\ v_{3i})$ és una columna de l'estat a la qual s'aplica MixColumn, la seva transformada $(w_{0i},\ w_{1i},\ w_{2i},\ w_{3i})$ és

$$\begin{pmatrix} w_{0i} \\ w_{1i} \\ w_{2i} \\ w_{3i} \end{pmatrix} = \begin{pmatrix} \mathtt{0x02} & \mathtt{0x03} & \mathtt{0x01} & \mathtt{0x01} \\ \mathtt{0x01} & \mathtt{0x02} & \mathtt{0x03} & \mathtt{0x01} \\ \mathtt{0x01} & \mathtt{0x01} & \mathtt{0x02} & \mathtt{0x03} \\ \mathtt{0x03} & \mathtt{0x01} & \mathtt{0x01} & \mathtt{0x02} \end{pmatrix} \begin{pmatrix} v_{0i} \\ v_{1i} \\ v_{2i} \\ v_{3i} \end{pmatrix}.$$

KeyExpansion

La clau K es disposarà en forma de bits en una matriu de 4 files per Nk columnes. Les columnes les denotarem per $\mathrm{clau}(0), \mathrm{clau}(1), ..., \mathrm{clau}(\mathrm{Nk}-1)$.

La funció KeyExpansion genera $\mathrm{Nr}+1$ subclaus a partir de la clau K. Les subclaus són emmagatzemades a la matriu **W**, que tindrà 4 files per $4(\mathrm{Nr}+1)$ columnes. Cada 4 columnes constituiran una subclau, de forma que $\mathrm{W}(0), \ldots, \mathrm{W}(3)$ és la clau K_0, $\mathrm{W}(4), \ldots, \mathrm{W}(7)$ la clau K_1, etc. Les columnes de W les construirem a partir de les anteriors seguint l'algoritme següent[1]:

Per i = 0 **fins** Nk − 1
  W(i) = clau(i)
FiPer
Per i = Nk **fins** 4 ∗ (Nr + 1) − 1
  temp = W(i − 1)
  Si i = 0 mod Nk
    temp = ByteSub(RotByte(temp)) ⊕ Rcon(i/Nk)
  FSi
  Si Nk = 8 **i** i = 4 mod Nk
    temp= ByteSub(temp)
  FSi
  W(i) := W(i − Nk) ⊕ temp
Fiper

[1]**ByteSub** actua sobre cada element de la columna.

La funció **RotByte** indica una rotació: la columna (a, b, c, d) es converteix en la columna (b, c, d, a). La funció **Rcon** aplicada a i retorna la columna $(x^{i-1}, 0, 0, 0)$, en notació polinòmica (és el residu de la divisió d'aquest per $m(x)$).

Algoritme de desxifrat

L'algoritme de desxifrat es pot fer seguint un esquema igual al del xifrat, amb una transformació inicial seguida de Nr tombs. Caldrà utilitzar subclaus obtingudes mitjançant la funció **InvKeyExp**, que denotem $\mathrm{invK_0}$, $\mathrm{invK_1}, \ldots,$ $\mathrm{invK_{Nr}}$. Si C indica el bloc a desxifrar, representat en bits i en forma d'estat, l'algoritme de desxifrat és:

estat = AddRoundKey(C, $\mathrm{invK_{Nr}}$)
Per i = Nr − 1 **fins** 1
    estat = $\mathrm{InvTomb_i}$ (estat, $\mathrm{invK_i}$)
FiPer
M = InvTombfinal (estat, $\mathrm{invK_0}$)

L'esquema dels $\mathrm{InvTomb_i}$ és el següent, per a $\mathrm{i} = \mathrm{Nr} - 1, \ldots, 1$:

estat = InvByteSub(estat)
estat = InvShiftRow(estat)
estat = InvMixColumn(estat)
estat = AddRoundKey(estat, $\mathrm{invK_i}$)

I per a InvTombfinal:

estat = InvByteSub(estat)
estat = InvShiftRow(estat)
estat = AddRoundKey(estat, $\mathrm{invK_0}$)

Les noves funcions són exactament les inverses de les que teníem a l'algoritme de xifrat:

- **InvByteSub**. Cal canviar l'ordre de la funció ByteSub: si abans hem fet $\mathbf{y} = \mathbf{Ax} + \mathbf{v}$, ara cal fer $\mathbf{y} = \mathbf{A^{-1}}(\mathbf{x} - \mathbf{v})$, on
$$A^{-1} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$
Després s'ha de substituir cada byte pel seu invers en $GF(2^8)$, si és diferent de `0x00`.
- **InvShiftRow**. Ara cal rotar les files de l'estat el mateix nombre de posicions però en sentit contrari.
- **InvMixColumn**. La multiplicació de cada columna es fa pel polinomi invers de l'anterior, que és
$$d(Y) = \mathtt{0x0B}\, Y^3 + \mathtt{0x0D}\, Y^2 + \mathtt{0x09}\, Y + \mathtt{0x0E}.$$
Igual que abans, farem una multiplicació matricial:
$$\begin{pmatrix} v_{0i} \\ v_{1i} \\ v_{2i} \\ v_{3i} \end{pmatrix} = \begin{pmatrix} \mathtt{0x0E} & \mathtt{0x0B} & \mathtt{0x0D} & \mathtt{0x09} \\ \mathtt{0x09} & \mathtt{0x0E} & \mathtt{0x0B} & \mathtt{0x0D} \\ \mathtt{0x0D} & \mathtt{0x09} & \mathtt{0x0E} & \mathtt{0x0B} \\ \mathtt{0x0B} & \mathtt{0x0D} & \mathtt{0x09} & \mathtt{0x0E} \end{pmatrix} \begin{pmatrix} w_{0i} \\ w_{1i} \\ w_{2i} \\ w_{3i} \end{pmatrix}.$$
- **InvKeyExp.** Aquesta funció aplica a la clau K primer la funció KeyExpansion, i després aplica InvMixColumn a totes les columnes de W excepte a les 4 primeres i les 4 últimes, de manera que $\mathrm{invK_0} = \mathrm{K_0}$, $\mathrm{invK_{Nr}} = \mathrm{K_{Nr}}$, i $\mathrm{invK_i} = \mathrm{InvMixColumn(K_i)}$, si $1 \leq \mathrm{i} \leq \mathrm{Nr} - 1$.

Missatges i *padding*

Els missatges seran cadenes de bytes. Cada byte correspon a 8 bits, per tant calen 16 bytes per a omplir un bloc de 128 bits. Al missatge li afegirem el mateix *padding* que al SHA, ara per a blocs de 128 bits.

Implementació: signatures

Definiu la classe `aes` amb el següents mètodes:

public static byte[ ][ ][ ] `keyExpansion`(BigInteger `K` , int `Nk`, int `Nr`)

entrada: `K` és un enter que representa la clau, `Nk` és el nombre de columnes de la clau i `Nr` és el nombre de tombs;
sortida: llista de les $\mathrm{Nr}+1$ subclaus per al xifrat, el primer index fa referència a la subclau, el segon a les files de la subclau i el darrer a les columnes de la subclau.

public static byte[ ][ ][ ] `invKeyExpansion` (BigInteger `K`, int `Nk`, int `Nr`)

entrada: `K` és un enter que representa la clau, `Nk` és el nombre de columnes de la clau i `Nr` és el nombre de tombs;
sortida: llista de les $\mathrm{Nr}+1$ subclaus per al desxifrat, el primer index fa referència a la subclau, el segon a les files de la subclau i el darrer a les columnes de la subclau.

public static byte `byteSub`(byte `subestat`)

entrada: `subestat` és un byte;
sortida: un byte, resultat d'aplicar la transformació ByteSub al `subestat`.

public static byte `invByteSub` (byte `subestat`)

entrada: `subestat` és un byte;
sortida: un byte, resultat d'aplicar la transformació InvByteSub al `subestat`.

public static byte[ ][ ] `shiftRow`(byte[ ][ ] `estat`)

entrada: `estat` és una matriu de bytes de 4 files;
sortida: una matriu de bytes de 4 files i 4 columnes resultat d'aplicar la transformació ShiftRow a `estat`.

public static byte[ ][ ] `invShiftRow`(byte[ ][ ] `estat`)

entrada: `estat` és una matriu de bytes de 4 files i 4 columnes;
sortida: una matriu de bytes de 4 files i 4 columnes resultat d'aplicar la transformació InvShiftRow a `estat`.

public static byte[ ][ ] `mixColumn`(byte[ ][ ] `estat`)

entrada: `estat` és una matriu de bytes de 4 files i de 4 columnes;
sortida: una matriu de bytes de 4 files, resultat d'aplicar la transformació MixColumn a l'`estat`.

public static byte[ ][ ] `invMixColumn`(byte[ ][ ] `estat`)

entrada: `estat` és una matriu de bytes de 4 files i de 4 columnes;
sortida: una matriu de bytes de 4 files, resultat d'aplicar la transformació InvMixColumn a l'`estat`.

public static byte[ ][ ] `addRoundKey`(byte[ ][ ] `estat`, byte[ ][ ] K_i)

entrada: `estat` i K_i són matrius de 4 files i el mateix nombre columnes tals que els seus elements són bytes;
sortida: una matriu de 4 files i amb el mateix nombre de columnes que les de entrada, resultat de sumar les matrius `estat` i K_i bit a bit.

public static byte[ ][ ] `rijndael`(byte[ ][ ] `estat`, byte[ ][ ][ ] `W`, int `Nk`, int `Nr`)

entrada: `estat` és una matriu de 4 files i de 4 columnes, els elements de la qual són bytes, `Nk` és la longitud de la clau partit per 32, `Nr` és el nombre de tombs i `W` és la matriu que emmagatzema les subclaus;
sortida: una matriu de 4 files i de 4 columnes obtinguda a partir de l'algorisme de xifrat aplicat a `estat`.

public static byte[ ][ ] `invRijndael`(byte[ ][ ] `estat`, byte[ ][ ][ ] `InvW`, int `Nk`, int `Nr`)

entrada: `estat` és una matriu de 4 files i de 4 columnes, els elements de la qual són bytes, `Nk` és la longitud de la clau partit per 32, `Nr` és el nombre de tombs i `InvW` és la matriu que emmagatzema les subclaus de desxifrat;
sortida: una matriu de 4 files i de 4 columnes obtinguda a partir de l'algorisme de desxifrat aplicat a `estat`.

public static byte[] `xifrarAES` (byte[] `M`, BigInteger `K`, int `Lk`)

entrada: `M` és una llista de bytes que representa el missatge a xifrar, `K` és un enter que representa la clau i `Lk` és la longitud de la clau (128, 192 o 256);
sortida: llista de bytes que és el criptograma obtingut xifrant el missatge `M` (després d'afegir-li el padding) en **mode CBC** amb la clau `K`.

public static byte[] `desxifrarAES` (byte[] `C`, BigInteger `K`, int `Lk`)

entrada: `C` és una llista de bytes, que representa el missatge xifrat amb el criptosistema Rijndael, `K` és un enter que representa la clau i `Lk` és la longitud de la clau (128, 192 o 256);
sortida: una llista de bytes obtinguda al desxifrar el missatge i eliminar-li el *padding*.

Valors de test

Al FIPS 197 trobareu exemples pel `keyExpansion`, `rijndael` i `invRijndael`.