Rijndael

Em 2/10/00 o *National Institute of Standards and Technology* (NIST) do *US Department of Commerce* recomendou a adopção do algoritmo designado por **Rijndael** como **Advanced Encryption Standard** (AES).

O Rijndael é uma cifra iterada com blocos de tamanho variável ($N_b \times 32$ bits, com $N_b = 4, 6$ ou 8) e chaves de tamanho variável ($N_k \times 32$ bits, com $N_k = 4, 6$ ou 8). O algoritmo tem 10 iterações quando $N_b = N_k = 4$, tem 14 iterações quando $N_b = 8$ ou $N_k = 8$ e tem 12 iterações nos restantes casos.

O Rijndael **não é** um circuito de Feistel. Cada *round* é uma substituição composta por três transformações uniformes[a] invertíveis chamadas *layers* (camadas):

Camada não linear (*non-linear layer*) aplicação paralela de S-*boxes* destinada a evitar correlações cruzadas,

Camada linear (*linear mixing layer*) garante a difusão dos bits transformados,

Mistura de chaves (*key addition layer*) mistura das sub-chaves por simples operações **XOR**

Antes do primeiro *round* é aplicada uma mistura de chaves prévia. A motivação para este tipo de operação (conhecido por *whitening*) reside no facto que qualquer transformação invertível que ocorra antes da primeira mistura de chaves pode ser retirada sem afectar as propriedades de segurança da cifra.

De modo a melhorar a fase de decifragem a camada linear do último *round* é diferente da dos restantes.

[a] todos os bits são transformados de modo análogo.

Round do Rijndael

Para facilitar a apresentação a descrição do algoritmo será aqui feita para o caso mais simples: blocos e chaves de 128 bits ($N_b = N_k = 4$) e 10 iterações.

A definição de *round* é uma transformação sobre um **estado** de 128 bits organizado como uma matriz 4×4 de bytes. Cada *round* usa uma única sub-chave de 128 bits também organizada como uma matriz 4×4 de bytes. Ambas matrizes são sequencialmente organizadas "em colunas": i.e. a posição (i, j) da matriz corresponde à posição $i + 4 \times j$ num vector uni-dimensional.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Estado

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Chave

Cada byte (organizado do bit mais significativo para o menos significativo) $b_7\, b_6\, \cdots\, b_2\, b_1\, b_0$ é interpretado como um polinómio em Y do 7º grau

$$b_7 \times Y^7 + b_6 \times Y^6 + \cdots + b_2 \times Y^2 + b_1 \times Y + b_0$$

São definidas operações de soma e multiplicação de bytes do seguinte modo:

Soma $A \oplus B$ calcula-se fazendo o **XOR** bit-a-bit dos respectivos coeficientes.

Multiplicação $A \otimes B$ calcula-se multiplicando os polinómios e reduzindo o resultado módulo $Y^8 + Y^4 + Y^3 + Y + 1$.

Exemplo

$$\begin{aligned}\texttt{ff} \otimes \texttt{03} &= (Y^7 + Y^6 + Y^5 + Y^4 + Y^3 + Y^2 + Y + 1) \otimes (Y + 1) = \\ &= (Y^8 + 1) \pmod{Y^8 + Y^4 + Y^3 + Y + 1} \;=\; Y^4 + Y^3 + Y = \\ &= \texttt{1a}\end{aligned}$$

As operações de multiplicação e similares (como o cálculo da inversa) podem ser realizadas usando uma *tabela de logaritmos*: cada byte $b \neq 0$, pode ser escrito na forma

$$b = g^e \pmod{Y^8 + Y^4 + Y^3 + Y + 1} \qquad \text{com} \quad g = \mathtt{ff}$$

para um único expoente $e \in \{0 \cdots 254\}$ determinado pela seguinte tabela. Para $e > 254$, tem-se $g^e \equiv g^{e \bmod 255}$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	–	0	40	73	80	146	113	174	120	247	186	197	153	34	214	219
1	160	37	32	2	226	93	237	107	193	89	74	65	254	15	4	64
2	200	137	77	208	72	245	42	88	11	69	133	162	22	138	147	90
3	233	116	129	110	114	75	105	78	39	180	55	192	44	21	104	166
4	240	239	177	27	117	139	248	94	112	86	30	253	82	10	128	8
5	51	68	109	189	173	183	202	14	62	151	178	131	187	238	130	148
6	18	125	156	210	169	26	150	228	154	161	115	12	145	19	118	63
7	79	163	220	3	95	132	232	211	84	176	61	142	144	235	206	231
8	25	49	24	230	217	252	67	53	157	207	179	249	33	215	134	106
9	152	140	126	236	70	76	38	35	122	29	50	98	168	97	48	205
a	91	111	108	198	149	28	229	143	213	46	223	225	242	199	54	99
b	102	136	191	195	218	123	171	92	227	121	23	234	170	85	188	241
c	58	244	165	57	196	100	250	36	209	167	66	175	190	9	13	212
d	194	81	201	45	155	96	52	83	185	6	59	159	158	71	103	243
e	119	221	203	31	5	41	43	56	135	127	172	224	17	204	251	60
f	124	47	216	141	101	7	182	222	184	87	20	164	246	181	16	1

A entrada $(\mathtt{d_1}, \mathtt{d_2})$ contém o expoente e tal que $b = g^e$ se escreve, em hexadecimal $\mathtt{d_1 d_2}$

Exemplo

Vamos usar a tabela para calcular o produto $\mathtt{ff} \otimes \mathtt{03}$ e o inverso $\mathtt{aa}^{-1}$.

Vemos que $\mathtt{ff} = g^1$ e $\mathtt{03} = g^{73}$. Logo $\mathtt{ff} \otimes \mathtt{03} = g^1 \otimes g^{73} = g^{74} = \mathtt{1a}$.

Vemos que $\mathtt{aa} = g^{223}$; logo $\mathtt{aa}^{-1} = g^{-223 \pmod{255}}$. Como $-223 \equiv (255 - 223) \equiv 32 \pmod{255}$ o inverso será $\mathtt{aa}^{-1} = g^{32} = \mathtt{12}$.

Uma tabela de logaritmos e a sua inversa podem ser facilmente armazenadas numa S-Box (que se resume a um vector de 256 bytes), o que torna muito eficiente a execução de operações não lineares como multiplicação, inversão e similares.

Transformações Rijndael

Cada uma dos primeiros 9 *rounds* do Rijndael é uma sequência de 4 transformações sobre o estado.
O último *round* é análogo mas não tem a transformação **MixColumn**.
Adicionalmente, antes do 1º *round* há uma transformação **AddRoundKey**.

$$\boxed{\textbf{ByteSub}} \\ \Downarrow \\ \boxed{\textbf{ShiftRow}} \\ \Downarrow \\ \boxed{\textbf{MixColumn}} \\ \Downarrow \\ \boxed{\textbf{AddRoundKey}}$$

As duas transformações `ByteSub` e `ShiftRow` formam a *Non Linear Layer*. A transformação `MixColumn` forma a *Linear Mixing Layer* enquanto que `AddRoundKey` implementa a *Key Addition Layer*.

Transformação `ByteSub`

Actua em paralelo e de modo uniforme sobre cada um dos bytes da matriz do estado transformando-o do modo seguinte

$$b(Y) \quad \Longrightarrow \quad u(Y) \oplus v(Y) \times b^{-1}(Y) \pmod{Y^8 + 1}$$
$$\text{com} \quad u \equiv \texttt{c6} \quad \text{e} \quad v \equiv \texttt{f1}$$

Notas

1. A característica não linear é basicamente implementada na transformação $b \Longrightarrow b^{-1}$.
2. A multiplicação desta inversa por v usa um polinómio $(Y^8 + 1)$ diferente do usado nas operações algébricas gerais sobre os bytes (aumentando a não-linearidade intrínseca). v é escolhido de modo que $v^{-1} \pmod{Y^8 + 1}$ exista.
3. A transformação pode ser implementada eficientemente numa S-Box 8×8 invertível. A sua inversa implementa-se usando a S-Box em sentido oposto.
4. A constante u é escolhida de modo que a S-Box não tenha pontos fixos nem anti-pontos fixos.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	c6	37	87	47	df	46	6	ac	f3	e0	86	42	1f	8d	4a	97
1	5c	d8	6c	27	5f	65	84	ff	2a	bd	da	a	39	ba	d7	fc
2	8b	2f	c9	92	93	3	8f	3c	b3	aa	ae	ef	e7	7d	e3	a1
3	b0	8c	c2	cc	71	99	a0	59	80	d1	f8	de	4e	82	db	a7
4	60	c8	32	51	41	16	55	fa	d5	43	9d	cb	62	ce	2	b8
5	c5	ed	f0	2e	f2	3f	eb	45	56	4c	1b	63	54	34	75	c
6	fd	e	5a	4f	c4	24	c3	a8	a4	6f	d0	7	f5	33	9	7a
7	e5	ca	f4	8	d9	29	73	af	3b	9b	5d	e2	f1	f	cf	dd
8	2c	30	c1	3e	5	89	b4	81	bc	8a	17	23	b6	25	61	c7
9	f6	e8	4	3d	d2	52	f9	78	94	1e	7b	b1	1d	15	40	4d
a	fe	d3	53	50	64	90	b2	35	dc	cd	3a	d6	e9	a9	be	67
b	8e	7c	83	26	28	ad	14	6a	36	95	bf	5e	a6	57	1a	70
c	5b	77	a2	12	31	9a	bb	9c	7e	2d	b7	1	44	2b	48	58
d	f7	13	ab	96	74	c0	9f	10	e6	a3	85	6b	98	ec	21	19
e	ee	7f	79	e1	66	6d	18	b9	49	11	88	6e	1c	a5	72	d
f	38	ea	68	20	b	9e	d4	76	e4	69	22	0	fb	b5	4b	91

Tabela 1: S-Box da cifra Rijndael

Transformação `ShiftRow`

As linhas da matriz que representa o estado sofrem uma rotação circular: a primeira não é rodada, a segunda roda uma posição, a terceira roda duas posições e a última roda 3 posições

$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \quad \Longrightarrow \quad \begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,3} & a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,0} \\ \hline \end{array}$$

A transformação inversa efectua-se rodando as linhas o mesmo número de posições mas em sentido inverso.

Transformação `MixColumn`

Nesta transformação do estado cada coluna da matriz que representa o estado é interpretada como um polinómio em X do 3º grau

$$\mathbf{a}_j(X) \;\equiv\; a_{0,j} + a_{1,j} \times X + a_{2,j} \times X^2 + a_{3,j} \times X^3$$

Nestes polinómios a soma é efectuada componente a componente e a multiplicação é efectuada módulo $(X^4 + 1)$. As operações básicas são a multiplicação de cada $\mathbf{a}(X)$ por uma matriz $\mathbf{C}$ ou pela sua inversa $\mathbf{C}^{-1}$ dadas por

$$\mathbf{C}\,\mathbf{a}_j \equiv \begin{bmatrix} \mathtt{02} & \mathtt{03} & \mathtt{01} & \mathtt{01} \\ \mathtt{01} & \mathtt{02} & \mathtt{03} & \mathtt{01} \\ \mathtt{01} & \mathtt{01} & \mathtt{02} & \mathtt{03} \\ \mathtt{03} & \mathtt{01} & \mathtt{01} & \mathtt{02} \end{bmatrix} \otimes \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \qquad \mathbf{C}^{-1}\,\mathbf{a}_j \equiv \begin{bmatrix} \mathtt{0e} & \mathtt{0b} & \mathtt{0d} & \mathtt{09} \\ \mathtt{09} & \mathtt{0e} & \mathtt{0b} & \mathtt{0d} \\ \mathtt{0d} & \mathtt{09} & \mathtt{0e} & \mathtt{0b} \\ \mathtt{0b} & \mathtt{0d} & \mathtt{09} & \mathtt{0e} \end{bmatrix} \otimes \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

A operação **MixColumn** multiplica por $\mathbf{C}$ cada uma das 4 colunas do estado

$$\begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix} \mathbf{C}\,\mathbf{a}_0 & \mathbf{C}\,\mathbf{a}_1 & \mathbf{C}\,\mathbf{a}_2 & \mathbf{C}\,\mathbf{a}_3 \end{bmatrix}$$

A operação inversa de **MixColumn** é análoga mas usa a matriz $\mathbf{C}^{-1}$.

Transformação `AddRoundKey`

É efectuado um **XOR** byte-a-byte do estado e da subchave

$$\begin{array}{|cccc|}\hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \Longrightarrow \begin{array}{|cccc|}\hline a_{0,0}\oplus k_{0,0} & a_{0,1}\oplus k_{0,1} & a_{0,2}\oplus k_{0,2} & a_{0,3}\oplus k_{0,3} \\ a_{1,0}\oplus k_{1,0} & a_{1,1}\oplus k_{1,1} & a_{1,2}\oplus k_{1,2} & a_{1,3}\oplus k_{1,3} \\ a_{2,0}\oplus k_{2,0} & a_{2,1}\oplus k_{2,1} & a_{2,2}\oplus k_{2,2} & a_{2,3}\oplus k_{2,3} \\ a_{3,0}\oplus k_{3,0} & a_{3,1}\oplus k_{3,1} & a_{3,2}\oplus k_{3,2} & a_{3,3}\oplus k_{3,3} \\ \hline \end{array}$$

A transformação `AddRoundKey` coincide com a sua inversa.

A cifra inversa

Como cada transformação de um *round* é invertível, a sua inversa constrói-se por aplicação das inversas das diversas componentes pela ordem inversa. Notando que $\texttt{AddRoundKey}^{-1} = \texttt{AddRoundKey}$ a transformação será

$$\texttt{AddRoundKey} \Rightarrow \texttt{MixColumn}^{-1} \Rightarrow \texttt{ShiftRow}^{-1} \Rightarrow \texttt{ByteSub}^{-1}$$

O primeiro *round* não tem a transformação $\texttt{MixColumn}^{-1}$ e o último *round* é seguido de uma aplicação extra de `AddRoundKey` . As subchaves são usadas pela ordem inversa da usada na cifragem.

Note-se que

1. A ordem relativa das transformações `ByteSub` e `ShiftRow` é indiferente porque `ByteSub` do mesmo modo para qualquer byte e `ShiftRow` apenas muda a ordem relativa dos bytes.
2. A ordem relativa de `MixColumn` e `AddRoundKey` é indiferente porque `MixColumn` é uma transformação linear, `AddRoundKey` é uma soma e as transformações lineares $f(\cdot)$ verificam $f(a+k) = f(a) + f(k)$.

Se notar-mos que o 1° *round* não tem a transformação $\texttt{MixColumn}^{-1}$ vemos então (usando estas duas transposições) que a decifragem é dada pela sequência exactamente análoga à da cifragem

$$\texttt{AddRoundKey} \Rightarrow \texttt{ByteSub}^{-1} \Rightarrow \texttt{ShiftRow}^{-1} \Rightarrow \texttt{MixColumn}^{-1} \Rightarrow \texttt{AddRoundKey} \Rightarrow \cdots$$

Programador de Chaves

O programador de chaves gera, a partir da chave principal, tantas subchaves (com o mesmo tamanho da chave principal) quantos os *rounds* mais 1. Note-se que cada *round* contém uma transformação **AddRoundKey** e, adicionalmente, antes do primeiro round existe uma operação **AddRoundKey** extra.

Vamos ilustrar a geração das subchaves para o caso em que $N_b = N_k = 4$ e são usados 10 *rounds*. Os princípios gerais são os seguintes:

- A `RoundKey` é um vector unidimensional de palavras de 32 bits (4 bytes) e de tamanho 4×11. Sequencialmente são usadas 4 palavras deste vector para formar cada uma das subchaves.
- A chave principal (designada `CipherKey`) é expandida para a `RoundKey` segundo o algoritmo seguinte.
  1. O vector `RoundKey` é inicializado com 11 cópias do vector `CipherKey`.
  2. Para $i = 4$ até 44 a palavra na posição i é substituída pelo **XOR** da palavra na posição $i - 4$ e pela palavra w calculada da seguinte forma.
     (a) Se i não é múltiplo de 4 usa-se para w a palavra na posição $i - 1$,
     (b) Se i é múltiplo de 4 e se for $k = i/4$ e s a palavra na posição $i - 1$, então

     $$w = \texttt{ByteSub4}(\texttt{rot4}(s)) \oplus \texttt{cons}(k)$$

     em que:
     i. $\texttt{cons}(k)$ é uma palavra constante

     $$(Y^k, \texttt{00}, \texttt{00}, \texttt{00})$$

     em que o 1º byte é representado pelo polinómio Y^k e os restantes três bytes são `00`.
     ii. $\texttt{rot4}(s)$ roda os 4 bytes de s uma posição:

     $$(b_0, b_1, b_2, b_3) \Rightarrow (b_1, b_2, b_3, b_0)$$

     iii. $\texttt{ByteSub4}(\cdot)$ aplica a S-Box `ByteSub` a cada um dos 4 bytes da palavra.