Introdução

O algoritmo Advanced Encryption Standard, abreviado com as siglas AES, é o algoritmo mais popular usado para proteção eletrónica de dados. Desde 2006, é o algoritmo mais popular dentro dos algoritmos usados no que se chama *criptografia simétrica*. A criptografia simétrica também é denominada criptografia de um código, e é que o código que se usa tanto para cifrar uma mensagem como para a decifrar é o mesmo. Portanto, recetor e emissor concordam neste mesme código.

Com este artigo, pretende-se dar um esquema sobre como funciona este tipo de algoritmos, qual a matemática que o suporta, e formar uma ideia de que é o que acontece, por exemplo, quando passamos um cartão de pré-pagamento para o pagamento do bilhete do autocarro, ou como funcionam internamente os cartões pré-pagamento de energia como os que explicamos na Secção de Novidades deste número.

Em termos muito genéricos, temos um emissor e um recetor. O emissor dá uma informação (**input**) que é encriptada pelo algoritmo que apresentamos, o recetor recebe esta informação encriptada (**output**), e junto com o código secreto (**pin**) que é conhecido por ambos realiza o processo de desencriptação. Deste modo, foi transferida uma informação com alta segurança de que terceiros não possam ter acesso a ela. Veremos como é processada essa informação no algoritmo, como processa o código e como realiza o processo de encriptação, e em que tipo de matemática está suportado todo este jogo de encriptação. O processo de desencriptação não o veremos por ser bastante similar, mas a trabalhar de forma inversa.

1 Bytes e vetor de bytes

A unidade básica para processar no algoritmo AES é um **b**yte: isso é uma sequência de oito bits tratado como uma unidade inteira. Um bit é um dígito do sistema binário. No sistema decimal, que é o que usualmente se usa na vida quotidiana, usamos 10 dígitos. No entanto, para o sistema binário apenas se necessitam dois dígitos, 0 e 1. A nível de placa informática, podemos imaginar o zero como desligado e o 1 como ligado. Ver um exemplo de um byte de oito bits representado na Figura 1.

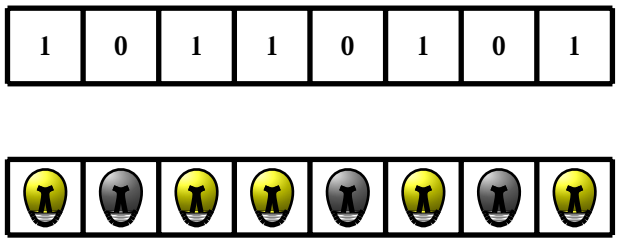

Figura 1: Exemplo de um byte de 8 bits em representação animada e binária

A partir da unidade do byte podemos definir uma lista de bytes ou uma matriz de bytes como se mostra nas Definições 1.1 e 1.2.

Definição 1.1. *Uma lista de bytes é uma sequência finita de $n \in \mathbb{N}$ bytes que podemos denotar como se segue,*

$$a_0 a_1 ... a_{n-1}$$

em que a_i é um byte para $i \in \{0, 1, ..., n-1\}$.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \left[\begin{array}{c|c|c|c} 10001001 & 11110000 & 10101010 & 10010011 \\ \hline 10011111 & 10010100 & 01001001 & 10111111 \\ \hline 00001001 & 01010101 & 10101011 & 10011001 \\ \hline 01011101 & 10010011 & 10110001 & 01011101 \end{array}\right]$$

Figura 2: Exemplo de uma matriz de bytes

Definição 1.2. *Uma matriz de bytes é um conjunto ordenado em filas e colunas cujos elementos são bytes. Mais particularmente, diremos que A é uma matriz de m filas e n colunas e os sus elementos serão denotados como a_{ij} para $i \in \{1, 2, \ldots, m\}$ e $j \in \{1, 2, \ldots, n\}$ sendo a_{ij} um byte.*

Ver um exemplo de uma matriz quadrada de tamanho 4 na Figura 2.

1.1 Entradas(input), saídas(output) e código secreto

A entrada e a saída para o algoritmo AES consiste em um array de 16 bytes, isto perfaz um total de 128 bits. No processo interno do algoritmo, estes 128 bits serão operados como uma matriz de bytes de 4 por 4 como foi visto na Figura 2. Desta forma, o input ou entrada do algoritmo é representado como o seguinte array $in_0 in_1 \ldots in_{15}$ e o output ou saída como $out_0 out_1 \ldots out_{15}$, em que cada in_i e out_i é um byte para $i \in \{0, 1, \ldots, 15\}$. Internamente, a entrada e todas as modificações, que denominamos estados intermédios, que produzirá o algoritmo serão trabalhados como matrizes de bytes de tamanho 4 por 4, como se pode ver na Figura Figura 3.

$$\underbrace{in_0 in_1 \ldots in_{15}}_{\text{entrada}} \rightarrow \begin{bmatrix} in_0 & in_4 & in_8 & in_{12} \\ in_1 & in_5 & in_9 & in_{13} \\ in_2 & in_6 & in_{10} & in_{14} \\ in_3 & in_7 & in_{11} & in_{15} \end{bmatrix} \rightarrow \underbrace{\begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix}}_{\text{estados intermedios}} \rightarrow \begin{bmatrix} out_0 & out_4 & out_8 & out_{12} \\ out_1 & out_5 & out_9 & out_{13} \\ out_2 & out_6 & out_{10} & out_{14} \\ out_3 & out_7 & out_{11} & out_{15} \end{bmatrix} \rightarrow \underbrace{out_0 out_1 \ldots out_{15}}_{\text{salida}}$$

Figura 3: Entrada, saída e estados intermédios.

No algoritmo os estados intermédios podem ver-se de outro ponto de vista. Os quatro bytes que formam cada coluna dão um total de 32 bits. Portanto, o estado pode ser interpretado como um vetor coluna de quatro palavras, cada uma delas formada com 32 bits. Assim, por exemplo, o estado intermédio representado na Figura 3 pode ser considerado como um vetor coluna das seguintes palavras:

$$w_0 = s_{00}s_{10}s_{20}s_{30}, \quad w_1 = s_{01}s_{11}s_{21}s_{31}, \quad , w_2 = s_{02}s_{12}s_{22}s_{32}, \quad w_3 = s_{03}s_{13}s_{23}s_{33}.$$

O código secreto também é um array de bytes, mas temos três possibilidades: 16 bytes(128 bits), 24 bytes(192 bits) ou 32 bytes(256 bits). No caso de 128 bits, o código inicial e os posteriores processados no algoritmo serão processados como blocos de tamanho 4 por 4; no caso de 192 bits blocos de tamanho 4 por 6, e finalmente no caso de 256 bites blocos de tamanho 4 por 8.

1.2 Interpretações diferentes de um byte

Parece divertido que um byte de oito bits possa ser interpretado como um polinómio ou como um par de elementos hexadecimais. Vamos ver em que é que consiste isto.

Notação polinomial

Representamos um byte de oito dígitos como se segue $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Este byte pode ser representado como um polinómio p(x) de grau sete como se segue:

$$p(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{k=0}^{7} b_kx^k. \tag{1.1}$$

Por exemplo, o byte $\{01101011\}$ representa o polinómio $x^6 + x^5 + x^3 + x + 1$.

Notação hexadecimal

O sistema hexadecimal consta de 16 dígitos:$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$. A letra A representa 10, a letra B onze, e assim sucessivamente. Por exemplo, o número 143 em notação decimal corresponde a 10011110 em notação binária e a $9E$ com notação hexadecimal. A representação hexadecimal que vamos fazer de um byte é realizada mediante a seguinte representação:

Padrão	Caráter	Padrão	Caráter	Padrão	Caráter	Padrão	Caráter
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

Usando este padrão, se tomarmos, por exemplo, o byte $\{10011110\}$ corresponde em notação hexadecimal aos primeiros quatro dígitos corresponde a 9 em notação hexadecimal e os últimos quatro dígitos a E. Isto é, corresponde ao par 9E. Usando este padrão, podemos representar a matriz dada na Figura 2 em notação hexadecimal como se segue:

$$\left[\begin{array}{c|c|c|c} 89 & F0 & AA & 93 \\ \hline 9F & 94 & 49 & BF \\ \hline 09 & 54 & AB & 99 \\ \hline 5D & 93 & B1 & 5D \end{array}\right].$$

2 A matemática do algoritmo

Todos os bytes no algoritmo AES são interpretados como polinómios de grau 7 com coeficientes 0 ou 1. De um ponto de vista mais matemático e próprio, diríamos que os bytes estão interpretados como elementos do campo finito $\mathcal{GF}(2^8)$ (ver [2]). Não vamos entrar em detalhes, e vamos falar da forma mais simples, mas se quiser saber mais do que é um corpo finito, os polinómios, as estruturas algebraicas, remeta-se às referências seguintes [2, 3, 4], é um mundo amazónico!

Vamos ver as operações que podemos fazer entre os bytes mediante a representação polinomial descrita na Equação (1.1).

2.1 Soma

Sejam dois bytes a $a = \{a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0\}$ e $b = \{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$, definimos a soma $c = \{c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0\}$ entre a e b que denotamos como $a \oplus b$ que denotamos como se segue:

Notação de byte: $c = a \oplus b = \{c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0\} = \{a_7 \oplus b_7, a_6 \oplus b_6, a_5 \oplus b_5, a_4 \oplus b_4, a_3 \oplus b_3, a_2 \oplus b_2, a_1 \oplus b_1, a_0 \oplus b_0\}$ em que $\oplus$ denota a operação XOR binária, isto é: $0 \oplus 0 = 0, 1 \oplus 0 = 0 \oplus 1 = 1, 1 \oplus 1 = 0$.

Notação polinomial: $\sum_{k=0}^{7} c_k x^k = \sum_{k=0}^{7} (a_k \oplus b_k) x^k$, sendo $\oplus$ a operação definida no anterior item.

Exercício 2.1. *A soma do polinómio* $x^7 + x^6 + x^4 + x + 1$ *com* $x^7 + x^4 + x + 1$ *dá* x^6 *como resultado o polinómio que corresponde ao byte* $\{01000000\}$.

2.2 Multiplicação

A multiplicação que vamos explicar não é tão intuitiva como a soma anterior. De facto, vamos abrir um intervalo para falar da operação *módulo*.

La multiplicación que vamos a explicar no es tan intuitiva como la suma anterior. De hecho vamos a hacer una parada para hablar de la operación *módulo*.

Vamos reparar nisto: quando dividimos qualquer número natural por 2, temos duas opções de resto, 0 e 1. Esta distinção não é outra que falar de números pares e ímpares. Mais formalmente, sejam todos os números inteiros $\mathbb{Z}$ e realizemos a divisão entre dois. Temos uma coleção de números cujo resto é 0, e outros cujo resto é 1. Deste ponto de vista, podemos falar de $\mathbb{Z}_2$ (espaço cociente) que é um corpo com dois únicos elementos: a classe dos elementos pares e a classe dos elementos ímpares. Repare que a operação XOR binária definida antes tem muita relação com isto. De facto, entender a operação $1 \oplus 1 = 0$ é muito simples agora se pensar que *a soma de dois números ímpares é um número par*.

Podemos pensar em fazer algo similar, por exemplo dividindo por 5. Desta forma, o conjunto de números inteiros poderia ser descomposto em 5 classes: os divisíveis por 5 (classe [0]), os que dão resto 1 (classe [1]), os que dão resto 2 (classe [2]) , os que dão resto 3 (classe [3]) e os que dão resto 4 (classe [4]). Em que classe se encontra o número 1234?

Acha a operação modular estranha? Imagina que podemos fazer o mesmo com polinómios? Pois, também se pode. Imaginemos o polinómio $x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ e dividamo-lo pelo polinómio irredutível de grau oito: $x^8 + x^4 + x^3 + x + 1$. Se dividirmos ambos os polinómios (faça-o como exercício para recordar como dividir dois polinómios) o resto que fica é o polinómio: $x^8 + x^4 + x^3 + x + 1$. De forma mais genérica: se tivermos um polinómio $p(x)$ e um outro polinómio de menor ou igual grau $q(x)$, diremos que $r(x)$ é $p(x)$ módulo $q(x)$ sendo $r(x)$ o resto de dividir $p(x)$ entre $q(x)$.

Dado este suposto salto ou parêntese, vamos entender o porquê disto tudo. Como se intitula esta subsecção, a multiplicação, isto é multiplicação de dois polinómios que cada um deles representa um byte. Mas se recordar como multiplicar dois polinómios, o grau do polinómio produto é a soma dos graus dos polinómios. Por exemplo, se multiplicar um polinómio de grau 3 por outro de grau 6, o resultado será um polinómio de grau 9. Mas em todo o momento dissemos que o algoritmo AES trabalhará com polinómios de grau menor ou igual que 7. Então se multiplicarmos polinómios de grau, no máximo sete, teremos na maioria dos casos polinómios de grau maior que sete. Esta é a razão pela qual a multiplicação não é a usual, mas que a multiplicação que é definida é a usual seguida de uma operação modular com o polinómio irredutível seguinte:

$$q(x) = x^8 + x^4 + x^3 + x + 1. \tag{2.1}$$

Ao ser $q(x)$ um polinómio de grau 8, o resto terá sempre um grau no máximo 7, obtendo o que queríamos, trabalhar com bytes. Esta multiplicação vamos denotá-la como $a(x) \bullet b(x)$ ou em notação de bytes como $a \bullet b$ entre a e b.

Exercício 2.2. *Sejam os polinómios* $a(x) = x^6 + x^4 + x^2 + x + 1$ *e* $b(x) = x^7 + x + 1$, *a sua multiplicação usual é* $x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ *e operação modular com* $q(x)$ *dado na Equação (2.1) dá como resultado o polinómio* $x^7 + x^6 + 1$. *Assim,*

$$a(x) \bullet b(x) = x^7 + x^6 + 1.$$

Observemos que na operação de multiplicar os polinómios se dá a multiplicação entre elementos binários, isto é: $1 \cdot 1 = 1, 1 \cdot 0 = 0 \cdot 1 = 0, 0 \cdot 0 = 0$.

2.3 Polinómios com coficientes que são bytes de oito dígitos

Na subsecção 1.1 falamos de palavras como uns vetores colunas de 32 bits concatenados. Vamos ver como trabalhar com tudo isto e a matemática que modeliza isto. Antes falámos de polinómios de grau menor ou igual a sete com coeficientes em $\mathbb{Z}_2$, isto é 0 ou 1. Agora vamos é definir o seguinte polinómio.

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0, \tag{2.2}$$

em que a_3, a_2, a_1 e a_0 são bytes de oito dígitos. Vamos ver como fizemos nas anteriores subsecções com este tipo de polinómios, como sejam as operações soma e multiplicação entre dois polinómios dados como a equação (2.2).

2.3.1 Soma

Esta é muito intuitiva, uma vez que é suficiente somar os bytes como vimos na subsecção 2.1. Desta forma, sejam $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ e $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ dois polinómios como na equação (2.2), então a soma denotada como será a seguinte:

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0).$$

Exercício 2.3. *Seja* $a(x) = \{10000001\}x^3 + \{11100001\}x^2 + \{10101010\}x + \{01010100\}$ *e* $b(x) = \{10000001\}x^3 + \{00000001\}x^2 + \{10101010\}x + \{00000000\}$ *polinómios com coeficientes bytes, é fácil conferir que*

$$a(x) + b(x) = \{00000000\}x^3 + \{11100000\}x^2 + \{00000000\}x + \{01010100\}.$$

2.3.2 Multiplicação

Esta operação segue um raciocínio similar ao realizado na subsecção 2.2. Mas é um pouco mais complicado.

Passo multiplicativo Criamos o polinómio produto $c(x)$ a partir de $a(x)$ e $b(x)$ dados como antes, como se segue:

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0,$$

em que

$$\begin{aligned} c_0 &= a_0 \bullet b_0, & c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1, \\ c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2, & c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3, \\ c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3, & c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3, \\ c_6 &= a_3 \bullet b_3. \end{aligned} \tag{2.3}$$

Passo modular Como o resultado na equação (2.3) não é um polinómio de quatro palavras, temos que realizar uma operação modular com um polinómio de grau 4. O algoritmo AES realiza-o com o polinómio $x^4 + 1$. Esta operação vamos deixá-la como exercício para que pratique (pode ver todos os detalhes em [5]).

Finalmente, se trabalhar o que propusemos, pode ver-se que realizar o produto entre $a(x)$ e $b(x)$, o qual denotaremos como $a(x) \otimes b(x)$, é equivalente a realizar uma operação matricial em que as somas e produtos são os explicados na subsecção anterior, isto é $\bullet$ e $\oplus$ entre bytes. Assim, $a(x) \otimes b(x) = d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$ em que

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Observemos que se tomamos como polinómio $a(x) = \{00000001\}x^3$ o resultado do produto de $a(x) \otimes b(x)$ é uma rotação da palavra $[b_0, b_1, b_2, b_3]$; isto é $[b_1, b_2, b_3, b_0]$. Esta operação vai denotar o algoritmo AES como *rotWord()*.

3 Um esquema do algoritmo

Antes de dar as ideias do algoritmo, queremos salientar vários pontos:

- A longitude da entrada e saída é de 128 bits representados em blocos de tamanho quatro por quatro, cada elemento um byte (usando notação hexadecimal será mais fácil representá-lo).
- A longitude do código secreto dependerá do tipo de algoritmo AES que use. Distinguem-se três AES-128, AES-192 e AES-256, os quais usam um código secreto, respetivamente de 128 bits, 192 bits e 256 bits.
- O número de voltas que se dará ao algoritmo dependerá também do tipo de AES. O AES.128 realizará 10 voltas, o AES-192 doze e o AES-256 realizará 14.

Para não complicar demais o algoritmo, apenas veremos o processo de encriptação. O processo de desencriptação é similar mas com operações inversas.

3.1 Input del algoritmo

O estado e o código secreto. Usando notação hexadecimal teríamos algo assim

ESTADO INICIAL

04	$E0$	48	28
66	CB	$F8$	06
81	19	$D3$	26
$E5$	$9A$	$7A$	$4C$

CODIGO SECRETA

$A0$	88	23	$2A$
FA	54	$A3$	$6C$
FE	$2C$	39	76
17	$B1$	39	05

Figura 4: Input del Algoritmo AES-128

3.2 Processo de encriptação do estado

Distinguimos três etapas:

Roda inicial

ESTADO INICIAL

04	$E0$	48	28
66	CB	$F8$	06
81	19	$D3$	26
$E5$	$9A$	$7A$	$4C$

,

CÓDIGO SECRETO

$A0$	88	23	$2A$
FA	54	$A3$	$6C$
FE	$2C$	39	76
17	$B1$	39	05

→ AddRoundKey →

ESTADO 0

$A4$	$G8$	$6B$	02
$9C$	$9F$	$5B$	$6A$
$7F$	35	EA	50
$F2$	$2B$	43	49

Figura 5: Roda Inicial do algoritmo AES-128

9 rodas

ESTADO $(i-1)$éSsimo

$A4$	$G8$	$6B$	02
$9C$	$9F$	$5B$	$6A$
$7F$	35	EA	50
$F2$	$2B$	43	49

→

1. SubBytes
2. ShiftRows
3. Mixcolums
4. AddRoundKey con

CLAVE (i)-éssima

*	*	*	*
*	*	*	*
*	*	*	*
*	*	*	*

→ ESTADO i+1-ésimo

$A4$	$G8$	$6B$	02
$9C$	$9F$	$5B$	$6A$
$7F$	35	EA	50
$F2$	$2B$	43	49

Figura 6: Roda i-éssima do algoritmo AES-128 para $i \in \{1, 2, ..., 9\}$

Roda final

ESTADO 9

$E9$	CB	$3D$	AF
09	31	32	$2E$
89	07	$7D$	$2C$
72	$5F$	94	$B5$

→

1. SubBytes
2. ShiftRows
3. AddRoundKey con

CLAVE 9-ésima

*	*	*	*
*	*	*	*
*	*	*	*
*	*	*	*

→ ESTADO FINAL

39	02	DC	19
25	DC	11	$6A$
84	09	85	$0B$
$1D$	FB	57	32

Figura 7: Última roda do algoritmo AES-128

Nas Figuras 5, 6 e 7 mostra-se um esquema do processo de encriptação do algoritmo AES e na. O processo de desencriptação é bastante análogo mas com operações inversas. Na secção em anexo vamos pôr claros exemplos do que fazem cada uma das operações que aparecem no esquema, estas são: SubBytes(), ShiftRows(), MixColumns(), AddRoundKey().

3.3 Processo de código

O código inicial irá modificando-se ao longo do processo para ser utilizado do modo que se apresenta nas figuras 6 e 7. Este processo consiste em, dado o código inicial, o qual vamos ver como um array de quatro palavras, cada palavra serão os 32 bits que encontramos em cada coluna. Para gerar a primeira palavra do código ronda 1, o que se fará é aplicar à palavra 4 do atual código a operação *RotWord()* que vimos na subsecção 2.3.2, e depois de ter feito isto, soma-se a primeira palavra e a primeira coluna da matriz Rcom que aparece na tabela 1. A segunda palavra é calculada a partir da calculada somando a segunda do código anterior; o mesmo com a terceira e a quarta. E este seria o código volta 1. As nove restantes são calculadas de forma análoga, utilizando sempre a anterior.

$$RCON = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{bmatrix}$$

Tabela 1: Matriz Rcon, cada coluna representa a potência x^i modulo $q(x)$ para $i \in \{0, 1, 2, \ldots, 9\}$.

4 Conclusões

O processo de desencriptação é muito similar ao mostrado, mas as operações que são utilizadas e que se explicam no Anexo 4 devem ser vistas de forma inversa. Todos os detalhes pode vê-los em 5. Se quiser ver exemplos, pode remeter-se a [5], em que aparece um programa e códigos das partes dos algoritmos.

Anexo: as operações do processo de encriptação

Aqui vamos representar de maneira informal e com simples exemplos cada uma das operações que aparecem no processo de encriptação do Algoritmo RSA-128. Por simplicidade vamos trabalhar com notação hexadecimal.

1. **SubBytes(·)**: é uma transformação não linear que substitui cada byte de um estado por um outro utilizando uma tabela de substituição. Esta tabela é denominada S-box (ver [5] ara ver a origem da mesma). Desta forma,

		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
x	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	**39**	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabela 2: Substituição do byte xy (formato hexadecimal).

se tivermos o estado seguinte,

19	$2F$	78	$4A$
AA	43	23	$5B$
23	11	$3B$	45
$F2$	08	$2B$	34

,

vamos substituir cada elemento deste bloco por um outro resultante da S-box. Por exemplo, o elemento posicionado na fila 2 e coluna 4 é $5B$. Portanto, tomamos o elemento resultante da S-box da fila que corresponde ao elemento 5 e a coluna B, este novo elemento é 39. De forma análoga é fácil ver que:

$$SubBytes\left(\begin{array}{|c|c|c|c|}\hline 19 & 2F & 78 & 4A \\ \hline AA & 43 & 23 & 5B \\ \hline 23 & 11 & 3B & 45 \\ \hline F2 & 08 & 2B & 34 \\ \hline\end{array}\right) = \begin{array}{|c|c|c|c|}\hline D4 & 15 & BC & D6 \\ \hline AC & 1A & 26 & 39 \\ \hline C3 & 82 & C2 & 6E \\ \hline 89 & 30 & F1 & 18 \\ \hline\end{array}$$

.

2. **ShiftRows(·)**: é uma transformação que a fila i-éssima $i-1$ mudanças cíclicas, para $i = \{1, 2, 3, 4\}$. Por exemplo, se $\boxed{23}\boxed{11}\boxed{3B}\boxed{45}$ é a fila três de um estado, realizam-se duas mudanças cíclicas, isto é $\boxed{3B}\boxed{45}\boxed{23}\boxed{11}$. Desta forma podemos ver:

$$ShiftRows\left(\begin{array}{|c|c|c|c|}\hline D4 & 15 & BC & D6 \\ \hline AC & 1A & 26 & 39 \\ \hline C3 & 82 & C2 & 6E \\ \hline 89 & 30 & F1 & 18 \\ \hline\end{array}\right) = \begin{array}{|c|c|c|c|}\hline D4 & 15 & BC & D6 \\ \hline 1A & 26 & 39 & AC \\ \hline C2 & 6E & C3 & 82 \\ \hline 18 & 89 & 30 & F1 \\ \hline\end{array}.$$

3. **M**ixColumns(·): é uma transformação que reflete a operação matemática vista na subsecção 2.3.2. Consideramos cada coluna de um estado como os coeficientes de um polinómio da forma como na Equação (2.2), e cada uma dela será multiplicado pelo polinómio fixo $a(x) = \{03\}x3 + \{01\}x2 + \{01\}x + \{02\}$.. Assim, a operação **M**ixColumns() realizada em cada coluna $s(x)$de um estado é a operação $s(x) \otimes a(x)$ definida em 2.3. Pode

tentar confirmar o seguinte:

$$MixColumns\left(\begin{array}{|c|c|c|c|}\hline D4 & E0 & B8 & 1E\\ \hline BF & B4 & 41 & 27\\ \hline 5D & 52 & 11 & 98\\ \hline 30 & AE & F1 & C5\\ \hline\end{array}\right) = \begin{array}{|c|c|c|c|}\hline 04 & E0 & 48 & 28\\ \hline 66 & CB & F8 & 06\\ \hline 81 & 19 & D3 & 26\\ \hline E5 & 9A & 7A & 4C\\ \hline\end{array}.$$

4. **A**ddRoundkey$(\cdot,\cdot)$: é uma transformação que tem dois argumentos, um estado e um código e a operação que se realiza é elemento a elemento a soma explicada na subsecção 2.1. O seguinte exemplo mostra o seguinte:

$$MixColumns\left(\underbrace{\begin{array}{|c|c|c|c|}\hline 04 & E0 & 48 & 28\\ \hline 66 & CB & F8 & 06\\ \hline 81 & 19 & D3 & 26\\ \hline E5 & 9A & 7A & 4C\\ \hline\end{array}}_{\text{estado}}, \underbrace{\begin{array}{|c|c|c|c|}\hline A4 & 88 & 23 & 2A\\ \hline 9C & 54 & A3 & 6C\\ \hline 7F & 2C & 39 & 76\\ \hline F2 & B1 & 39 & 05\\ \hline\end{array}}_{\text{código secreto}}\right) = \begin{array}{|c|c|c|c|}\hline A4 & 68 & 6B & 02\\ \hline 9C & 9F & 5B & 6A\\ \hline 7F & 35 & EA & 50\\ \hline F2 & 2B & 43 & 49\\ \hline\end{array}.$$

Referências

[1] Rijndael algorithm. *http://people.eku.edu/styere/Encrypt/JS-AES.html*.

[2] B. Hartley and T.O. Hawkes. *Rings, Modules and Linear Algebra*. Chapman and Hall, 1970.

[3] P.J. Hilton and Yel-Chiang Wu. *Curso de Álgebra Moderna*. Reverté, 1977.

[4] T.W. Hungerford. *Algebra*. Springer-Verlag, 1974.

[5] Federal Information. Advanced encryptation standard. *Processing Standards Publication 197*, 2001.