O padrão de criptografia simétrica AES

Raquel de Araújo de Souza
Fábio Borges de Oliveira
{rasouza,borges}@lncc.br

Resumo

Neste trabalho apresentamos o algoritmo AES, atual padrão de criptografia simétrica do governo dos EUA. Explicamos cada uma de suas etapas e suas respectivas inversas e, ao final, utilizamos um exemplo de cifragem e decifragem com o algoritmo.

1 Introdução

O atual padrão de criptografia dos EUA se originou de um concurso lançado em 1997 pelo NIST *(National Institute of Standards and Technology)*. Nesse momento havia a necessidade de escolher um algoritmo mais seguro e eficiente para substituir o DES *(Data Encryption Standard)*, que apresentou fragilidades. O novo algoritmo deveria atender a certos pré-requisitos como: ser divulgado publicamente e não possuir patentes; cifrar em blocos de 128 bits usando chaves de 128, 192 e 256 bits; ser implementado tanto em software quanto em hardware; ter maior rapidez em relação ao 3DES, uma variação recursiva do antigo padrão DES. Em 1998, na Primeira Conferência dos Candidatos AES, apresentaram-se 15 candidatos e, um ano depois, na Segunda Conferência, foram indicados 5 destes como finalistas: MARS, RC6, Rijndael, Serpent e Twofish. Em 2000, é conhecido o vencedor: Rijndael. O algoritmo, criado pelos belgas Vincent Rijmen e Joan Daemen, foi escolhido com base em qualidades como segurança, flexibilidade, bom desempenho em software e hardware etc.

2 O algoritmo AES

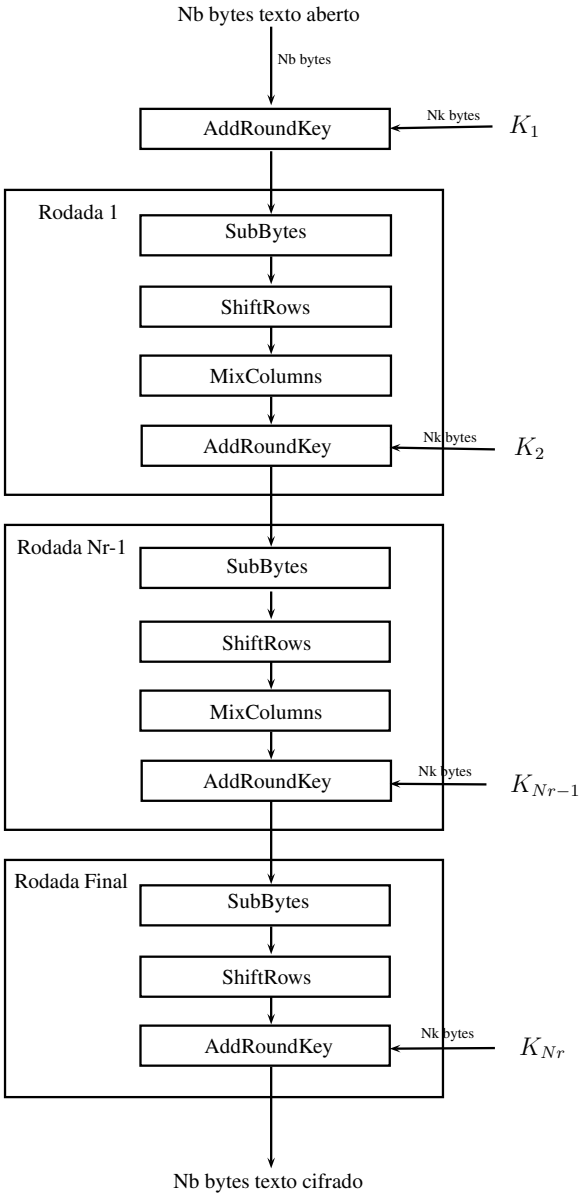
2.1 Aspectos Principais

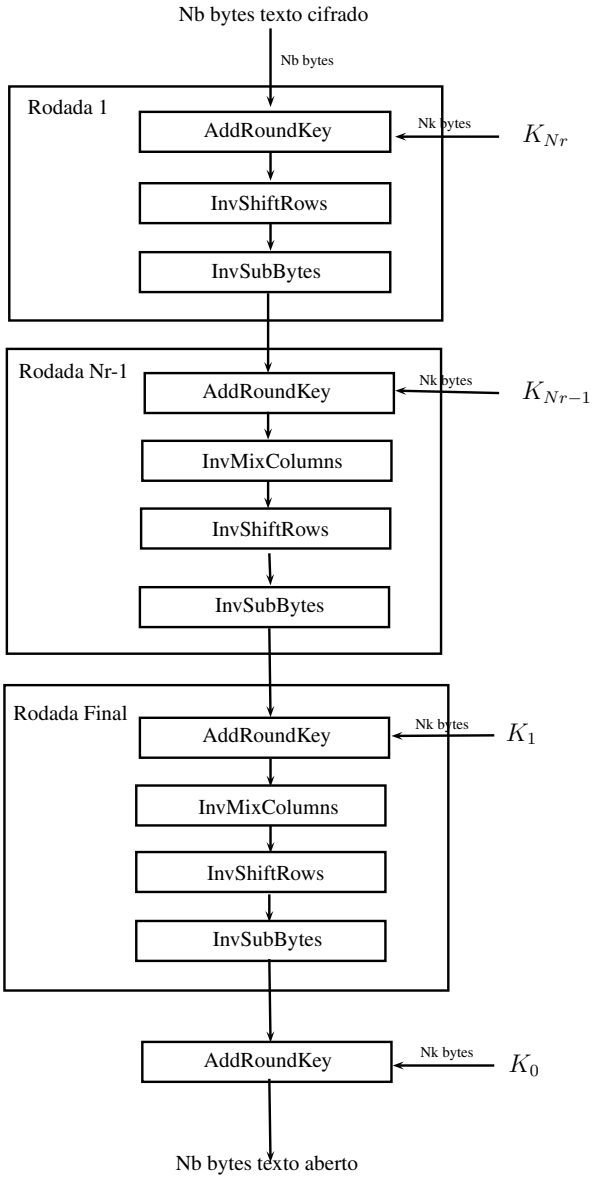
A pequena diferença entre o AES e o Rijndael é que este suporta tamanhos de chave e bloco variando entre 128, 160, 192, 224 e 256 bits. Ou seja, os dois possuem o mesmo funcionamento, sendo que o Rijndael pode utilizar uma variação maior de tamanhos de chave e bloco do que o AES. Usaremos exemplos com blocos e chaves 128 bits, apenas.

Antes de tudo, definiremos alguns termos utilizados com freqüência no algoritmo. *Estado* é uma matriz de bytes que inicialmente contém a mensagem e, após cada etapa, será modificada. *Mensagem* é o texto antes de ser criptografado, e cujo conteúdo deve ser acessível apenas ao destinatário. Para manter a segurança, é preciso tornar a mensagem um texto ilegível, também chamado *texto cifrado*, e isso é feito através da criptografia.

No Rijndael, o tamanho do estado vai depender do tamanho do bloco utilizado, sendo composta de 4 linhas e *Nb* colunas, onde *Nb* é o número de bits do bloco dividido por 32. O algoritmo possui rodadas, também chamadas de iterações, que, por sua vez, possuem 4 etapas: *AddRoundKey, SubBytes, ShiftRows e MixColumns*. Na última rodada, porém, a operação MixColumns não é realizada. Usaremos a sigla *Nr* (*number of rounds*) para designar o número de rodadas que serão utilizadas durante a execução do algoritmo. No AES o número de rodadas depende do tamanho da chave, sendo Nr igual a 10, 12 e 14, para Nk igual a 4, 6 e 8, respectivamente. O algoritmo possui uma chave principal e, a partir dela, são geradas $Nr + 1$ chaves, geralmente chamadas de *chaves de rodada*, pois cada uma será usada em uma rodada diferente. Além disso, a própria chave principal é usada antes da primeira rodada. A chave principal é alocada em uma matriz de 4 linhas e Nk colunas, e cada chave de rodada é agrupada da mesma maneira que o bloco de dados.

As figuras abaixo mostram de uma forma geral como ocorrem os processos de cifragem e decifragem com o AES.





2.1.1 Transformação SubBytes

Nesta etapa, cada byte do estado é substituído por outro em uma S-box (caixa de substituição), denotada por S_{RD}. Todos os valores dessa caixa são dados em hexadecimal. Os quatro primeiros e os quatro últimos bits do byte a ser substituído representam em hexadecimal, respectivamente, a linha e a coluna onde se

encontra o novo byte. Por exemplo, o valor hexadecimal 6a deverá ser substituído pelo byte que se encontra na linha "6"e na coluna "a"da S_{RD}, que é o valor 02. A S_{RD} é gerada a partir da composição de duas funções f e g constituídas sobre $GF(2^8)$. Se $a = a_0a_1a_2a_3a_4a_5a_6a_7$, temos que

$$g(a) = a^{-1},$$

onde a^{-1} é o inverso multiplicativo de a em $GF(2^8)$ e

$$f(a) = b = b_0b_1b_2b_3b_4b_5b_6b_7,$$

onde

$$b = \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \\ 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

A tabela 1 mostra a S-box usada no AES. A inversa da operação SubBytes chama-se **InvSubBytes**, e

Tabela 1: S_{RD}

		x															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
y	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

usa uma S-box inversa, denotada por S_{RD}^{-1}, que usa a composição de funções

$$S_{RD}^{-1}[b] = g^{-1}(f^{-1}(b)) = g(f^{-1}(b)).$$

Aplicando a S-box no valor 6a, obtemos o valor 02. Logo, aplicando a S-box inversa em 02 obtemos o valor 6a.

2.1.2 Transformação ShiftRows

Consiste em rotacionar à esquerda as linhas do estado, trocando assim a posição dos bytes. O número de posições a serem rotacionadas depende da linha e do tamanho do bloco nos quais estamos trabalhando. Na Tabela 3, Ci representa o número de posições a serem rotacionadas na linha de posição i (de cima para baixo) de um bloco com Nb colunas. Exemplificamos na Tabela 4 a transformação ShiftRows. A operação inversa correspondente chama-se **InvShiftRows** e consiste apenas em fazer o mesmo rotacionamento, porém à direita.

Tabela 2: S_{RD}^{-1}

		x															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
y	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabela 3: Deslocamento em função de Nb e Ci

Nb	$C0$	$C1$	$C2$	$C3$
4	0	1	2	3
6	0	1	2	3
8	0	1	3	4

Tabela 4: Exemplo da transformação ShiftRows

24	40	78	68
6e	63	96	48
13	b0	8e	53
ae	59	01	ee

$\xrightarrow{ShiftRows}$

24	40	78	68
63	96	48	6e
8e	53	13	b0
ee	ae	59	01

2.1.3 Transformação MixColumns

Nesta etapa, o resultado da operação em uma determinada coluna não influencia o resultado nas demais. Porém, a mudança de um byte em uma coluna influencia o resultado na coluna inteira. Os bytes do estado são tratados como polinômios sobre o corpo finito[1] $GF(2^8)$. Essa transformação pode ser representada por uma multiplicação de matrizes. Chamaremos de S' o estado após essa transformação. Ele será o resultado da multiplicação de uma matriz fixa C pela matriz S que representa o estado, ou seja,

$$\begin{bmatrix} S'_{1,1} & S'_{1,2} & S'_{1,3} & S'_{1,4} \\ S'_{2,1} & S'_{2,2} & S'_{2,3} & S'_{2,4} \\ S'_{3,1} & S'_{3,2} & S'_{3,3} & S'_{3,4} \\ S'_{4,1} & S'_{4,2} & S'_{4,3} & S'_{4,4} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \odot \begin{bmatrix} S_{1,1} & S_{1,2} & S_{1,3} & S_{1,4} \\ S_{2,1} & S_{2,2} & S_{2,3} & S_{2,4} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,4} \\ S_{4,1} & S_{4,2} & S_{4,3} & S_{4,4} \end{bmatrix},$$

onde $\odot$ é o produto matricial em $GF(2^8)$. A inversa dessa operação, denominada **InvMixColumns**, também é uma multiplicação usando uma matriz fixa $\mathbf{B} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$, que é a inversa da

[1] Para saber mais sobre corpos finitos consulte [2] e [3]

matriz $\mathbf{C} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$, usada na cifragem. Assim, para encontrar o estado inicial S, anterior à aplicação da operação MixColumns, devemos calcular $B \odot S' = S$.

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \odot \begin{bmatrix} S'_{1,1} & S'_{1,2} & S'_{1,3} & S'_{1,4} \\ S'_{2,1} & S'_{2,2} & S'_{2,3} & S'_{2,4} \\ S'_{3,1} & S'_{3,2} & S'_{3,3} & S'_{3,4} \\ S'_{4,1} & S'_{4,2} & S'_{4,3} & S'_{4,4} \end{bmatrix} = \begin{bmatrix} S_{1,1} & S_{1,2} & S_{1,3} & S_{1,4} \\ S_{2,1} & S_{2,2} & S_{2,3} & S_{2,4} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,4} \\ S_{4,1} & S_{4,2} & S_{4,3} & S_{4,4} \end{bmatrix}.$$

2.1.4 Transformação AddRoundKey

Esta é uma operação de XOR byte a byte entre o estado e a chave da rodada. Logo, essa é uma transformação que opera cada byte individualmente. Basicamente, se $s_{x,y}$ é um byte do estado e $k_{x,y}$ um byte da chave, temos que o byte $s'_{x,y}$ do novo estado é igual a $s_{x,y} \oplus k_{x,y}$. Como $(a \oplus b) \oplus b = a$, a transformação AddRoundKey é sua própria inversa.

2.2 Expansão de Chave

Como já foi dito, as chaves utilizadas em cada rodada são geradas a partir da chave principal do AES. O algoritmo usado gera $Nr + 1$ chaves, pois antes da primeira rodada é feita uma AddRoundKey. A geração de chaves, também conhecida como expansão de chave, resulta em um vetor com palavras (isto é, uma seqüência) de 4 bytes. Denotaremos cada palavra por w_i, onde i é a posição da palavra no vetor. Inicialmente, completamos as Nk primeiras palavras do vetor com os bytes da chave principal. Se i não

Tabela 5: Vetor composto pelas chaves de rodada

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	...
chave da rodada 0				chave da rodada 1				...			

é múltiplo de Nk, w_i será obtida através de uma operação de XOR entre temp $= w_{[i-1]}$ e $w_{[i-Nk]}$, as palavras 1 e Nk posições anteriores a ela, respectivamente. Caso i não seja múltiplo de Nk, usaremos as funções [4]:

1. RotWord - Essa função rotaciona a palavra uma posição à esquerda;
2. SubWord - Similar à SubBytes, substitui cada byte da palavra pelo byte correspondente na S-box;
3. Rcon(j) - É uma constante diferente a cada rodada (j). Essa constante é dada por Rcon(j)=(RC[j],00,00,00), onde RC[1]=1 e RC[j]=2 · RC[j-1], com a multiplicação sobre $GF(2^8)$. A tabela 2.2 mostra o valor de RC[j] a cada rodada.

Tabela 6: RC[j] em função da rodada (j)

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

A expansão de chave, para $Nk \leqslant 6$, pode ser descrita em pseudocódigo[2]:

```
KeyExpansion (byte key [4*Nk], word w[Nb*(Nr+1)], Nk)
      word temp
      for i from 0 to Nk-1
            w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
```

[2]O algoritmo para a geração de chaves, para Nk>6, pode ser encontrado em [1]

```
for i from Nk to Nb*(Nr + 1)-1
      temp = w[i - 1]
      if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      w[i] = w[i - Nk] xor temp
```

Exemplo Suponhamos que a chave principal tenha 128 bits e que a chave de rodada 4 seja: 2f 56 3a c5 78 bb de 47 14 f5 23 d7 a8 cf e2 35 . Na Tabela 7, indicamos passo a passo o cálculo da primeira palavra da chave de rodada 5.

Tabela 7: Exemplo do cálculo de w[i]

i	20
temp = w[i-1]	a8cfe235
RotWord	cfe235a8
SubWord	8a9896c2
Rcon(5)	10000000
temp = SubWord $\oplus$ Rcon(5)	9a9896c2
w[i-Nk]	2f563ac5
w[i] = temp $\oplus$ w[i-Nk]	b5ceac07

2.3 Cifrando e decifrando uma mensagem com o AES

O objetivo desta seção é demonstrar como funciona na prática o algoritmo AES. Para isso, iremos cifrar e decifrar uma mensagem usando esse algoritmo. Por questão de conveniência, utilizaremos um tamanho de 128 bits para a mensagem, ou seja, usaremos um estado com 4 linhas e 4 colunas. Todos os valores das tabelas serão representados no sistema hexadecimal e usaremos a tabela ASCII estendida. Vamos usar um texto com 16 caracteres, o que corresponde a 128 bits. Iremos cifrar e decifrar a mensagem PALESTRA␣NO␣LNCC, que aparece na Tabela 8 como um estado do AES. Escolhemos a chave principal:

Tabela 8: Estado da mensagem

P	S	␣	L
A	T	N	N
L	R	O	C
E	A	␣	C

$\xrightarrow{ASCII}$

50	53	20	4c
41	54	4e	4e
4c	52	4f	43
45	41	20	43

BOLSISTA␣DO␣CNPq. Utilizando a tabela ASCII após alocá-la em uma matriz, chegamos à Tabela 9. Após implementarmos o algoritmo de geração de chaves, obtivemos os valores mostrados na Tabela 10.

Tabela 9: Chave principal agrupada em bloco

B	I	␣	C
O	S	D	N
L	T	O	P
S	A	␣	q

$\xrightarrow{ASCII}$

42	49	20	43
4f	53	44	4e
4c	54	4f	50
53	41	20	71

Mostraremos os detalhes das etapas da primeira rodada e, a seguir, apenas o resultado da rodada 10. Começamos o algoritmo com a AddRoundKey antes da primeira rodada, usando a chave de rodada 0. A operação é vista na Tabela 11. Prosseguindo, aplicamos SubBytes ao estado obtido na operação anterior e, logo após, a operação ShiftRows. Podemos ver os resultados nas Tabelas 12 e 13.

Explicaremos agora alguns aspectos da transformação MixColumns. As operações de soma e multiplicação usadas nessa etapa são um pouco diferentes das usuais, pois são realizadas sobre um corpo

Tabela 10: Chaves de Rodada

Chave de Rodada 0	424f4c53	49535441	20444f20	434e5071
	w_0	w_1	w_2	w_3
Chave de Rodada 1	6c1cef49	254fbb08	050bf428	4645a459
	w_4	w_5	w_6	w_7
Chave de Rodada 2	00552413	251a9f1b	20116b33	6654cf6a
	w_8	w_9	w_{10}	w_{11}
Chave de Rodada 3	24df2620	01c5b93b	21d4d208	47801d62
	w_{12}	w_{13}	w_{14}	w_{15}
Chave de Rodada 4	e17b8c80	e0be35bb	c16ae7b3	86eafad1
	w_{16}	w_{17}	w_{18}	w_{19}
Chave de Rodada 5	7656b2c4	96e8877f	578260cc	d1689a1d
	w_{20}	w_{21}	w_{22}	w_{23}
Chave de Rodada 6	13ee16fa	85069185	d284f149	03ec6b54
	w_{24}	w_{25}	w_{26}	w_{27}
Chave de Rodada 7	9d913681	1897a704	ca13564d	c9ff3d19
	w_{28}	w_{29}	w_{30}	w_{31}
Chave de Rodada 8	0bb6e25c	13214558	d9321315	10cd2e0c
	w_{32}	w_{33}	w_{34}	w_{35}
Chave de Rodada 9	ad871c96	bea659ce	67944adb	775964d7
	w_{36}	w_{37}	w_{38}	w_{39}
Chave de Rodada 10	50c41263	ee624bad	89f60176	feaf65a1
	w_{40}	w_{41}	w_{42}	w_{43}

Tabela 11: Operação AddRoundKey anterior à primeira rodada

50	53	20	4c		42	49	20	43		12	1a	00	0f
41	54	4e	4e		4f	53	44	4e		0e	07	0a	00
4c	52	4f	43	$\oplus$	4c	54	4f	50	$=$	00	06	00	13
45	41	20	43		53	41	20	71		16	00	00	32

Tabela 12: Transformação SubBytes na primeira rodada

12	1a	00	0f		c9	a2	63	76
0e	07	0a	00		ab	c5	67	63
00	06	00	13	$\xrightarrow{SubBytes}$	63	6f	63	7d
16	00	00	32		47	63	63	23

Tabela 13: Transformação ShiftRows na primeira rodada

c9	a2	63	76		c9	a2	63	76
ab	c5	67	63		c5	67	63	ab
63	6f	63	7d	$\xrightarrow{ShiftRows}$	63	7d	63	6f
47	63	63	23		23	47	63	63

de Galois. Há várias representações possíveis para os elementos de um corpo de Galois, ou corpo finito. A mais simples para a multiplicação é a polinomial, que atribui a cada conjunto de 8 bits um polinômio de grau menor ou igual a 7, sendo cada bit o coeficiente de um termo do polinômio. Por exemplo, o byte AB tem representação binária igual a 10101011 e representação polinomial igual a $1x^7 + 0x^6 + 1x^5 + 0x^4 + 1x^3 + 0x^2 + 1x + 1 = x^7 + x^5 + x^3 + x + 1$.

Pelo fato de estarmos trabalhando sobre um corpo de característica 2, a soma e o xor coincidem. Assim, na soma de polinômios, reduziremos os coeficientes módulo 2. Na operação de multiplicação, o

resultado é reduzido módulo $m(x) = x^8 + x^4 + x^3 + x + 1$. O símbolo $\otimes$ será usado para representar o produto entre dois elementos de $GF(2^8)$.

As operações envolvendo congruência entre polinômios, que usaremos aqui, podem ser encontradas no livro de Stallings [5]. Relembrando, o nosso estado após a etapa MixColumns será o resultado de uma multiplicação de matrizes. Se denotarmos por S o novo estado, teremos:

$$S = \begin{bmatrix} S_{1,1} & S_{1,2} & S_{1,3} & S_{1,4} \\ S_{2,1} & S_{2,2} & S_{2,3} & S_{2,4} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,4} \\ S_{4,1} & S_{4,2} & S_{4,3} & S_{4,4} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \odot \begin{bmatrix} c9 & a2 & 63 & 76 \\ c5 & 67 & 63 & ab \\ 63 & 7d & 63 & 6f \\ 23 & 47 & 63 & 63 \end{bmatrix}.$$

Mostraremos aqui somente o cálculo do byte da primeira linha e primeira coluna, pois os demais foram obtidos de maneira similar.

$$\begin{aligned} S_{1,1} &= (02 \otimes c9) \oplus (03 \otimes c5) \oplus 63 \oplus 23 \\ &= (00000010 \otimes 11001001) \oplus (00000011 \otimes 11000101) \\ &\oplus (01100011) \oplus (00100011) \\ &= x \otimes (x^7 + x^6 + x^3 + 1) \oplus (x + 1) \otimes (x^7 + x^6 + x^2 + 1) \\ &\oplus (x^6 + x^5 + x + 1) \oplus (x^5 + x + 1) \\ &= (x^8 + x^7 + x^4 + x) \oplus (x^8 + x^7 + x^3 + x + x^7 + x^6 + x^2 + 1) \\ &\oplus (x^6 + x^5 + x + 1) \oplus (x^5 + x + 1) \\ &= x^7 + x^4 + x^3 + x^2 + 1 \\ &= 10011101 \\ &= 9d \end{aligned}$$

Terminando a primeira rodada da cifragem, realizamos uma AddRoundKey com a chave de rodada 1,

Tabela 14: Transformação MixColumns na primeira rodada

c9	a2	63	76
c5	67	63	ab
63	7d	63	6f
23	47	63	63

$\xrightarrow{MixColumns}$

9d	cc	63	06
de	ac	63	e9
af	f6	63	a6
a0	69	63	98

que é 6c1cef49 254fbb08 050bf428 4645a459. A Tabela 15 mostra o resultado.

Tabela 15: Transformação AddRoundKey na primeira rodada

9d	cc	63	06
de	ac	63	e9
af	f6	63	a6
a0	69	63	98

$\oplus$

6c	25	05	46
1c	4f	0b	45
ef	bb	f4	a4
49	08	28	59

$=$

f1	e9	66	40
c2	e3	68	ac
40	4d	97	02
e9	61	4b	c1

Após Rodada 1: f1 e9 66 40 c2 e3 68 ac 40 4d 97 02 e9 61 4b c1

O processo que realizamos na primeira rodada é repetido 9 vezes, sendo que na última a operação MixColumns não é realizada. Após a realização das 10 rodadas, obtivemos os resultados da Tabela 16. Para chegar novamente à mensagem, realizamos o processo de decifragem. Nele, a primeira rodada decifra a rodada 10 da cifragem; a rodada 2 decifra a rodada 9 etc. Claramente, essa ordem inversa ocorre porque a decifragem é um processo de "volta"ao texto original. Como queremos demonstrar como se dá esse caminho de cifragem/decifragem, iremos cifrar e decifrar a mesma rodada. Na Tabela 17, mostramos

Tabela 16: Rodada 10 do processo de cifragem

Etapas	**Resultado**
Estado após rodada 9	79 8c e2 b5 e1 b0 76 89 f3 26 a7 d9 07 ce fc 15
SubBytes	b6 64 98 d5 f8 e7 38 a7 0d f7 5c 35 c5 8b b0 59
ShiftRows	b6 e7 5c 59 f8 f7 b0 d5 0d 8b 98 a7 c5 64 38 35
AddRoundKey	e6 23 4e 3a 16 95 fb 78 84 7d 99 d1 3b cb 5d 94
Texto Cifrado	e6 23 4e 3a 16 95 fb 78 84 7d 99 d1 3b cb 5d 94

Tabela 17: Rodada 9 do processo de decifragem

Etapas	**Resultado**
Estado após rodada 8	9a 2d 17 82 bf 09 08 37 08 ef f1 3b 4b 9a f0 ca
AddRoundKey	9a 78 33 91 9a 13 97 2c 28 fe 9a 08 2d ce 3f a0
InvMixColumns	a1 11 88 78 1e 45 77 1e 33 91 09 ef 09 25 e3 b3
InvShiftRows	a1 25 09 1e 1e 11 e3 ef 33 45 88 b3 09 91 77 78
InvSubBytes	f1 c2 40 e9 e9 e3 4d 61 66 68 97 4b 40 ac 02 c1

Tabela 18: AddRoundKey - Rodada 10 da decifragem

f1	e9	66	40		6c	25	05	46		9d	cc	63	06
c2	e3	68	ac	$\oplus$	1c	4f	0b	45	=	de	ac	63	e9
40	4d	97	02		ef	bb	f4	a4		af	f6	63	a6
e9	61	4b	c1		49	08	28	59		a0	69	63	98

os resultados da rodada 9 da decifragem. Ao realizarmos a rodada 10 da decifragem, chegaremos à mensagem. Apresentamos na Tabela 18 a primeira operação: AddRoundKey. Agora calculamos o estado após InvMixColumns. Novamente, mostraremos apenas o cálculo do primeiro byte e, na Tabela 19, apresentamos todos os resultados.

$$
\begin{aligned}
S_{1,1} &= (0e \otimes 9d) \oplus (0b \otimes de) \oplus (0d \otimes af) \oplus (09 \otimes a0) \\
&= (00001110 \otimes 10011101) \oplus (00001011 \otimes 11011110) \\
&\oplus \ (00001101 \otimes 10101111) \oplus (00001001 \otimes 10100000) \\
&= (x^3 + x^2 + x) \otimes (x^7 + x^4 + x^3 + x^2 + 1) \\
&\oplus \ (x^3 + x + 1) \otimes (x^7 + x^6 + x^4 + x^3 + x^2 + x) \\
&\oplus \ (x^3 + x^2 + 1) \otimes (x^7 + x^5 + x^3 + x^2 + x + 1) \oplus (x^3 + 1) \otimes (x^7 + x^5) \\
&= (x^{10} + x^7 + x^6 + x^5 + x^3 + x^9 + x^6 + x^5 + x^4 + x^2 + \\
&+ \ x^8 + x^5 + x^4 + x^3 + x) \\
&\oplus \ (x^{10} + x^9 + x^7 + x^6 + x^5 + x^4 + x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + \\
&+ \ x^7 + x^6 + x^4 + x^3 + x^2 + x) \\
&\oplus \ (x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x^9 + x^7 + x^5 + x^4 + x^3 + x^2 + \\
&+ \ x^7 + x^5 + x^3 + x^2 + x + 1) \\
&\oplus \ (x^{10} + x^8 + x^7 + x^5) \\
&= x^9 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\
&= x^7 + x^6 + x^3 + 1 \\
&= 11001001 \\
&= c9
\end{aligned}
$$

Seguindo a ordem, chegamos à operação InvShiftRows, Tabela 20, e em seguida temos a etapa Inv-

Tabela 19: InvMixColumns - Rodada 10

9d	cc	63	06
de	ac	63	e9
af	f6	63	a6
a0	69	63	98

$\xrightarrow{InvMixColumns}$

c9	a2	63	76
c5	67	63	ab
63	7d	63	6f
23	47	63	63

SubBytes, Tabela 21.

Para terminar, fazemos uma AddRoundKey, correspondente àquela feita antes da rodada 1 do pro-

Tabela 20: InvShiftRows - Rodada 10

c9	a2	63	76
c5	67	63	ab
63	7d	63	6f
23	47	63	63

$\xrightarrow{InvShiftRows}$

c9	a2	63	76
ab	c5	67	63
63	6f	63	7d
47	63	63	23

Tabela 21: InvSubBytes - Rodada 10

c9	a2	63	76
ab	c5	67	63
63	6f	63	7d
47	63	63	23

$\xrightarrow{InvSubBytes}$

12	1a	00	0f
0e	07	0a	00
00	06	00	13
16	00	00	32

cesso de cifragem. A Tabela 22 mostra o xor entre o estado depois de InvSubBytes e a chave de rodada 0. Terminado o processo de decifragem, chegamos ao texto 50414c45 53545241 204e4f20 4c4e4343, que

Tabela 22: AddRoundKey após a rodada 10

12	1a	00	0f
0e	07	0a	00
00	06	00	13
16	00	00	32

$\oplus$

42	49	20	43
4f	53	44	4e
4c	54	4f	50
53	41	20	71

$=$

50	53	20	4c
41	54	4e	4e
4c	52	4f	43
45	41	20	43

corresponde à mensagem, representada na Tabela 23.

Tabela 23: Mensagem

50	53	20	4c
41	54	4e	4e
4c	52	4f	43
45	41	20	43

$\rightarrow$

P	S	∅	L
A	T	N	N
L	R	O	C
E	A	∅	C

3 Conclusões

Este artigo apresentou o algoritmo AES, descrevendo o funcionamento de suas etapas e os processos de cifragem e decifragem. O algoritmo trabalha sobre o corpo finito $GF(2^8)$ e possui funcionamento simples, usando caixas de substituição (S-box), rotações, a operação de xor, multiplicação de matrizes. Descrevemos ainda um exemplo de cifragem e decifragem com o algoritmo.

4 Agradecimentos

Gostaríamos de agradecer ao PIBIC/CNPq, pelo apoio financeiro para este trabalho.

Referências

[1] Joan Daemen and Vincent Rijmen, *The design of Rijndael: AES — The Advanced Encryption Standard*, Springer-Verlag, 2002.

[2] E. Horowitz, *Modular arithmetic and finite field theory: A tutorial*, SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation (New York, NY, USA), ACM Press, 1971, pp. 188–194.

[3] Richard E. Klima, Neil Sigmon, and Ernest Stitzinger, *Applications of abstract algebra with Maple*, 2000. MR MR1720327 (2000k:65004)

[4] NIST, *Specification for the Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, 2001.

[5] William Stallings, *Cryptography and network security principles and practices, fourth edition*, Prentice Hall, 2005.