# O algoritmo AES: Apresentação e Descrição da Estrutura

**Raquel de Araújo de Souza**[1], **Fábio Borges de Oliveira**[1], **Gerson Nunes da Cunha**[2]

[1]Laboratório Nacional de Computação Científica
Av. Getulio Vargas, 333, Quitandinha CEP: 25651-075 Petrópolis - Rio de Janeiro.

[2]Universidade Católica de Petrópolis - UCP
Rua Barão do Amazonas, 124, Centro - CEP: 25685-070 Petrópolis - Rio de Janeiro.

{rasouza, borges, gerson}@lncc.br

***Abstract.*** *This article has the purpose of show in a clear form the functioning of AES (Advanced Encryption Standard), algorithm which became the standard of cryptography, to the security of the American government data. There is a short introduction telling the story of the AES competition and later we will demonstrate its functioning, by the explanation of the stages of encryption/decryption algorithm and a simple example.*

***Resumo.*** *Este artigo tem a finalidade de apresentar de forma clara o funcionamento do AES (Advanced Encryption Standard), algoritmo que se tornou o padrão de criptografia para segurança de dados do governo americano. É feita uma breve introdução relatando a história do concurso AES e, em seguida, demonstramos o seu funcionamento através da explanação das etapas do algoritmo de cifragem/decifragem e de um exemplo simples.*

## 1. Introdução

Durante 20 anos, o DES *(Data Encryption Standard)* foi o algoritmo padrão usado pelo governo americano para proteger informações confidenciais. O surgimento do AES *(Advanced Encryption Standard)* decorreu da grande necessidade de se substituir o DES, que se tornou ultrapassado em virtude do pequeno tamanho da chave (56 bits) utilizada. Para isso, o NIST *(National Institute of Standards and Technology)* lançou em 1997 um concurso para adotar o novo algoritmo simétrico, que passaria a se chamar AES *(Advanced Encryption Standard)*.

O algoritmo deveria atender a alguns requisitos, tais como: direitos autorais livres; divulgação pública; maior rapidez em relação ao 3DES; cifrar em blocos de 128 bits com chaves de 128, 192 e 256 bits; possibilidade de implementação em software e hardware. Na Primeira Conferência dos Candidatos AES, em 98, apresentaram-se 15 candidatos e, em 1999, a Segunda Conferência indicou 5 destes para continuar na disputa: MARS, RC6, Rijndael, Serpent e Twofish. Em 2000, após análises rigorosas de especialistas na área de criptografia, é conhecido o vencedor: Rijndael. O algoritmo foi criado por Vincent Rijmen e Joan Daemen, dois belgas. Como tanto na primeira quanto na segunda etapa do concurso, todos os algoritmos descritos atendiam às exigências do concurso, a decisão foi tomada com base em outras qualidades como segurança, flexibilidade, bom desempenho em software e hardware etc.

## 2. O algoritmo AES

### 2.1. Estrutura

O Rijndael suporta tamanhos de chave e bloco variando entre 128, 160, 192, 224 e 256 bits, diferentemente do AES. Apesar de os dois possuírem o mesmo funcionamento, aqui usaremos exemplos com blocos e chaves 128 bits, apenas.

Para descrever a estrutura do AES é fundamental termos em mente a definição de *estado*, no contexto do algoritmo. *Estado* é a matriz de bytes que iremos manipular entre as diversas rodadas, ou *rounds*, e que portanto será modificada a cada etapa. No Rijndael, o tamanho dessa matriz 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 (pois 4 bytes representam 32 bits). Como o AES usa blocos de 128 bits, o estado terá 4 linhas e 4 colunas.

A chave é agrupada da mesma maneira que o bloco de dados, com número de colunas *Nk* [NIST 2001]. 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 varia de acordo com o tamanho da chave, sendo Nr igual a 10, 12 e 14, para Nk igual a 4, 6 e 8, respectivamente.

A cada rodada do algoritmo de cifragem, realizamos 4 etapas: *AddRoundKey, SubBytes, ShiftRows e MixColumns*. Na última rodada, porém, a operação MixColumns é suprimida. Cada uma destas etapas e suas respectivas inversas, que serão usadas no processo de decriptação, serão descritas mais detalhadamente nas próximas subseções. Todos os valores contidos nos estados dos exemplos serão dados em hexadecimal.

#### 2.1.1. Transformação SubBytes

Essa é a única transformação da cifra que não é linear [Daemen and Rijmen 2002]. Cada byte do estado é substituído por outro em uma S-box (caixa de substituição). Todos os valores dessa caixa são dados em hexadecimal. A substituição é feita da seguinte maneira: 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 c2 deverá ser substituído pelo byte que se encontra na linha c e na coluna 2 da S-box, que é o valor 25. A tabela 1 mostra a S-box usada no AES. A inversa da operação SubBytes chama-se **InvSubBytes**, e usa uma S-box inversa. Por exemplo, aplicando a S-box no valor a6, obtemos o valor 24. Então, aplicando a S-box inversa em 24 obtemos o valor a6.

#### 2.1.2. Transformação ShiftRows

É uma etapa simples do algoritmo e consiste em rotacionar à esquerda as linhas do estado, trocando assim a posição dos bytes. O número de posições a serem rotacionadas depende da posição da linha. Na primeira linha não há rotacionamento; na segunda, os bytes são rotacionados uma posição; na terceira, 2 posições e na última, 3 posições. A operação inversa correspondente chama-se **InvShiftRows** e para aplicá-la é necessário apenas fazer um rotacionamento similar àquele feito na operação de cifragem, porém à direita.

**Tabela 1. S-box usada no AES**

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

### 2.1.3. Transformação MixColumns

Nesta etapa, cada coluna é operada de forma independente. Isso quer dizer que o resultado da operação em uma determinada coluna não influencia o resultado nas demais. Entretanto, cada byte de uma coluna influencia todos os outros bytes da mesma. Nessa etapa, as colunas do estado são tratadas como polinômios sobre o corpo finito[1] $GF(2^8)$. Podemos representar essa transformação por uma multiplicação de matrizes. O novo estado $S'$ 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 produtório matricial em $GF(2^8)$. A inversa da operação MixColumns, denominada **InvMixColumns**, também consiste em uma multiplicação usando uma matriz fixa. Essa matriz deve ser a inversa de $\mathbf{C} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$ que é a matriz $\mathbf{B} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$. Ou seja, para encontrar $S$ devemos calcular $B \odot S' = S$. O modelo da transformação pode ser visto abaixo:

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

Veremos mais detalhes desta transformação no exemplo de cifragem/decifragem.

[1] Para saber mais sobre corpos finitos consulte [Horowitz 1971] e [Klima et al. 2000]

### 2.1.4. Transformação AddRoundKey

AddRoundKey é uma operação de XOR entre o estado e a chave da rodada, que possuem o mesmo número de bytes. Assim, essa transformação opera cada byte individualmente. O XOR é feito byte a byte, ou seja, 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 $s_{x,y} \oplus k_{x,y}$. A operação é representada na Tabela 2. A transformação AddRoundKey é sua própria inversa.

**Tabela 2. Transformação AddRoundKey**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_{1,1}$ | $s_{1,2}$ | $s_{1,3}$ | $s_{1,4}$ | $\oplus$ | $k_{1,1}$ | $k_{2,1}$ | $k_{3,1}$ | $k_{4,1}$ | = | $s'_{1,1}$ | $s'_{1,2}$ | $s'_{1,3}$ | $s'_{1,4}$ |
| $s_{2,1}$ | $s_{2,2}$ | $s_{2,3}$ | $s_{2,4}$ | | $k_{1,2}$ | $k_{2,2}$ | $k_{3,2}$ | $k_{4,2}$ | | $s'_{2,1}$ | $s'_{2,2}$ | $s'_{2,3}$ | $s'_{2,4}$ |
| $s_{3,1}$ | $s_{3,2}$ | $s_{3,3}$ | $s_{3,4}$ | | $k_{1,3}$ | $k_{2,3}$ | $k_{3,3}$ | $k_{4,3}$ | | $s'_{3,1}$ | $s'_{3,2}$ | $s'_{3,3}$ | $s'_{3,4}$ |
| $s_{4,1}$ | $s_{4,2}$ | $s_{4,3}$ | $s_{4,4}$ | | $k_{1,4}$ | $k_{2,4}$ | $k_{3,4}$ | $k_{4,4}$ | | $s'_{4,1}$ | $s'_{4,2}$ | $s'_{4,3}$ | $s'_{4,4}$ |

## 2.2. Geração de Chaves

As chaves usadas 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, conhecida também como expansão de chave, resulta em um vetor com palavras (isto é, uma seqüência) de 4 bytes. Cada palavra será representada aqui por $w_i$, sendo $i$ a posição da palavra no vetor. O algoritmo começa completando as 4

**Tabela 3. 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 | | | | ... | | | |

primeiras palavras do vetor com os bytes da chave principal. Depois, temos duas ocasiões a considerar:

**i não é múltiplo de Nk** - Nesse caso, $w_i$ 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;

**i é múltiplo de Nk** - Usamos outras operações [NIST 2001]:

1. RotWord - Rotaciona a palavra uma posição (correspondente a um byte) à esquerda;
2. SubWord - Aplica a S-box do AES em cada byte da palavra;
3. Rcon(j) - Faz uma operação de XOR entre o resultado dos dois primeiros itens com 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 abaixo mostra o valor de RC[j] a cada rodada.

**Tabela 4. 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]:

[2]O algoritmo para a geração de chaves, para Nk>6, pode ser encontrado em [Daemen and Rijmen 2002]

```
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])
      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** Vamos supor que a chave de rodada 6 seja: 5c 3a 11 02 14 6f 5b af bc 52 40 dd 50 f4 61 78. O cálculo em cada etapa dos 4 primeiros bytes (primeira palavra) da chave de rodada 7 pode ser visto na Tabela 5:

**Tabela 5. Exemplo do cálculo de w[i]**

| i | 28 |
|---|---|
| temp = w[i-1] | 50f46178 |
| RotWord | f4617850 |
| SubWord | bfefbc53 |
| Rcon(7) | 40000000 |
| temp = SubWord $\oplus$ Rcon(7) | fffebc53 |
| w[i-Nk] | 5c3a1102 |
| w[i] = temp $\oplus$ w[i-Nk] | acd5ad51 |

## 2.3. Exemplo de Cifragem/Decifragem com AES

Nesta seção vamos colocar em prática tudo o que foi abordado a respeito do algoritmo AES. Para isso, usaremos um exemplo simples, cifrando e decifrando uma mensagem. *Mensagem* é o texto legível, ou seja, antes de ser criptografado, e cujo significado deve ser descoberto apenas pelo destinatário; daí a necessidade do uso da criptografia, tornando a mensagem um texto ilegível, também chamado *texto cifrado*.

Como vimos, o AES é um cifrador de bloco, portanto vamos usar um tamanho de 128 bits para a mensagem, ou seja, usaremos um estado com 4 linhas e 4 colunas. Logicamente, podem ser cifrados textos maiores ou menores que 128 bits, esta escolha é feita apenas para simplificar o exemplo. Os valores serão representados no sistema hexadecimal e trabalharemos com a tabela ASCII extendida. Como cada caractere possui 8 bits, vamos usar um texto com 16 caracteres. Por exemplo, vamos cifrar e decifrar a mensagem PALESTRA $\not b$NO $\not b$LNCC. A Tabela 6 mostra a mensagem transformada em um estado do AES.

**Tabela 6. Estado da mensagem**

| | | | |
|---|---|---|---|
| P | S | $\not b$ | L |
| A | T | N | N |
| L | R | O | C |
| E | A | $\not b$ | C |

$\xrightarrow{ASCII}$

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

Como estamos trabalhando com uma chave de 128 bits, e portanto 10 rodadas, devemos ter 11 chaves de rodada. Em primeiro lugar, selecionamos a chave principal. A nossa será BOLSISTA $\not b$DO $\not b$CNPq. Agrupando em bloco e utilizando a tabela ASCII, podemos ver a matriz da chave na Tabela 7.

**Tabela 7. 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 |

Sabendo que a chave de rodada 0 é a própria chave principal, e procedendo como no exemplo da subseção anterior para a formação das demais chaves, obtivemos os valores mostrados na Tabela 8. Dela vamos retirar todas as chaves de rodada usadas nesse exemplo. Neste trabalho, porém, demonstraremos em detalhe as etapas da primeira rodada, e apenas o resultado da rodada 10. Na primeira rodada, começamos utilizando a

**Tabela 8. 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 |
| | $w_{40}$ | $w_{41}$ | $w_{42}$ | $w_{43}$ |

chave de rodada 0 através da operação AddRoundKey, apresentada na Tabela 9. Agora

**Tabela 9. Operação AddRoundKey anterior à primeira rodada**

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

$\oplus$

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

=

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

passamos à etapa SubBytes, utilizando o estado resultante da operação acima. Em seguida aplicamos a operação ShiftRows ao novo estado. As Tabelas 10 e 11 apresentam os resultados obtidos após as etapas SubBytes e ShiftRows, respectivamente.

A operação MixColumns é mais delicada. Como estamos trabalhando sobre um corpo finito, temos as operações de soma e multiplicação um pouco diferentes das usuais. Para simplificar os cálculos, usaremos uma notação 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 c2 tem representação binária igual a 11000010 e

**Tabela 10. Transformação SubBytes na primeira rodada**

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

$\xrightarrow{SubBytes}$

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

**Tabela 11. Transformação ShiftRows na primeira rodada**

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

$\xrightarrow{ShiftRows}$

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

representação polinomial igual a $1x^7+1x^6+0x^5+0x^4+0x^3+0x^2+x+0 = x^7+x^6+x$.

A operação de soma no corpo finito que estamos usando corresponde a uma operação de XOR entre os bits. Então, quando somarmos dois polinômios, eliminaremos os termos iguais dois a dois. A operação de multiplicação é uma multiplicação comum de polinômios, porém o resultado é obtido módulo $m(x) = x^8 + x^4 + x^3 + x + 1$. Isto quer dizer que iremos trabalhar com o resto da divisão do polinômio em questão (caso seu grau seja maior que 7) por $m(x)$. Para adiantar, temos que o resto da divisão de $x^8$ por $m(x)$ é igual a $x^4 + x^3 + x + 1$. Usaremos $\otimes$ para denotar o produto entre dois elementos de $GF(2^8)$.

A aritmética polinomial que usaremos aqui pode ser encontrada na seção 4.5 do livro de Stallings [Stallings 2005]. Com essas informações em mãos, podemos realizar a operação MixColumns. Como já vimos, o nosso estado após essa etapa será o resultado de uma multiplicação de matrizes. Chamando esse novo estado de S, teremos:

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

Como esta etapa é bastante longa, mostraremos apenas o cálculo do byte da primeira linha e primeira coluna. Os demais foram calculados de maneira análoga.

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

Para terminar o processo de cifragem da primeira rodada, é feita uma AddRoundKey

**Tabela 12. Transformação MixColumns na primeira rodada**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c9 | a2 | 63 | 76 | | 9d | cc | 63 | 06 |
| c5 | 67 | 63 | ab | $\xrightarrow{MixColumns}$ | de | ac | 63 | e9 |
| 63 | 7d | 63 | 6f | | af | f6 | 63 | a6 |
| 23 | 47 | 63 | 63 | | a0 | 69 | 63 | 98 |

com a chave de rodada 1, que é 6c1cef49 254fbb08 050bf428 4645a459. O resultado é exposto na Tabela 13.

**Tabela 13. Transformação AddRoundKey na primeira rodada**

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

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

Acabamos de cifrar a primeira rodada. Este processo é repetido 9 vezes, sendo que na última é omitida a operação MixColumns. A Tabela 14 mostra os resultados da rodada 10. No processo de decifragem, a rodada 1 corresponde à decifragem da rodada 10; a rodada

**Tabela 14. 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 |

2 decifra a rodada 9, e assim por diante. Isso ocorre, evidentemente, porque partimos do texto cifrado para chegar à mensagem, realizando um processo de "volta". Nosso objetivo neste trabalho é cifrar e decifrar a mesma rodada, para mostrar com detalhes como funciona cada operação e sua inversa. Assim, os resultados obtidos na rodada 9 podem ser consultados na Tabela 15. Agora vamos para a rodada 10, que corresponde à

**Tabela 15. 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 |

decifragem da rodada 1 de encriptação, ou seja, ao realizarmos esse processo, chegaremos à mensagem. Em primeiro lugar, é importante lembrar que na AddRoundKey da rodada 10, Tabela 16, usaremos a chave de rodada 1, pois esta foi a chave usada na rodada 1 de cifragem. Agora, passamos à etapa MixColumns. Como fizemos na etapa de cifragem,

**Tabela 16. AddRoundKey - Rodada 10 da decifragem**

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

$\oplus$

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

=

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

mostraremos apenas o cálculo do primeiro byte.

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

Na Tabela 17, temos o estado após a operação InvMixColumns. Prosseguindo, efetuamos

**Tabela 17. 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 |

a operação InvShiftRows, Tabela 18, e logo após temos a etapa InvSubBytes, Tabela 19. Finalmente, não podemos nos esquecer de que, durante o processo de cifragem, foi feita

**Tabela 18. 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 |

uma AddRoundKey antes da rodada 1. Portanto, faremos agora uma AddRoundKey com a chave de rodada 0, a mesma utilizada naquela ocasião. Veja o resultado na Tabela 20.

**Tabela 19. 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 |

**Tabela 20. 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 |

Como podemos ver, chegamos ao texto: 50414c45 53545241 204e4f20 4c4e4343. A Tabela 21 mostra a mensagem agrupada em bloco.

**Tabela 21. 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. Conclusão

Esse trabalho apresentou o algoritmo que é o atual padrão de criptografia dos EUA, descrevendo detalhadamente suas etapas e os processos de cifragem e decifragem. O algoritmo usa uma caixa de substituição (S-box), rotações, a operação de xor, multiplicação de matrizes, e trabalha sobre o corpo finito $GF(2^8)$.

Para facilitar a compreensão, mostramos um exemplo simples, cifrando e decifrando a mensagem PALESTRA ∅NO ∅LNCC com a chave BOLSISTA ∅DO ∅CNPq. Nesse exemplo, demonstramos praticamente como funciona cada uma das etapas do algoritmo, e suas respectivas inversas. Ao decifrar a décima rodada, chegamos à mensagem, como esperávamos.

## 4. Agradecimentos

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

## Referências

Daemen, J. and Rijmen, V. (2002). *The design of Rijndael: AES — The Advanced Encryption Standard*. Springer-Verlag.

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

Klima, R. E., Sigmon, N., and Stitzinger, E. (2000). *Applications of abstract algebra with Maple*. CRC Press, Boca Raton, FL.

NIST (2001). Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197.

Stallings, W. (2005). *Cryptography and Network Security Principles and Practices, Fourth Edition*. Prentice Hall.