

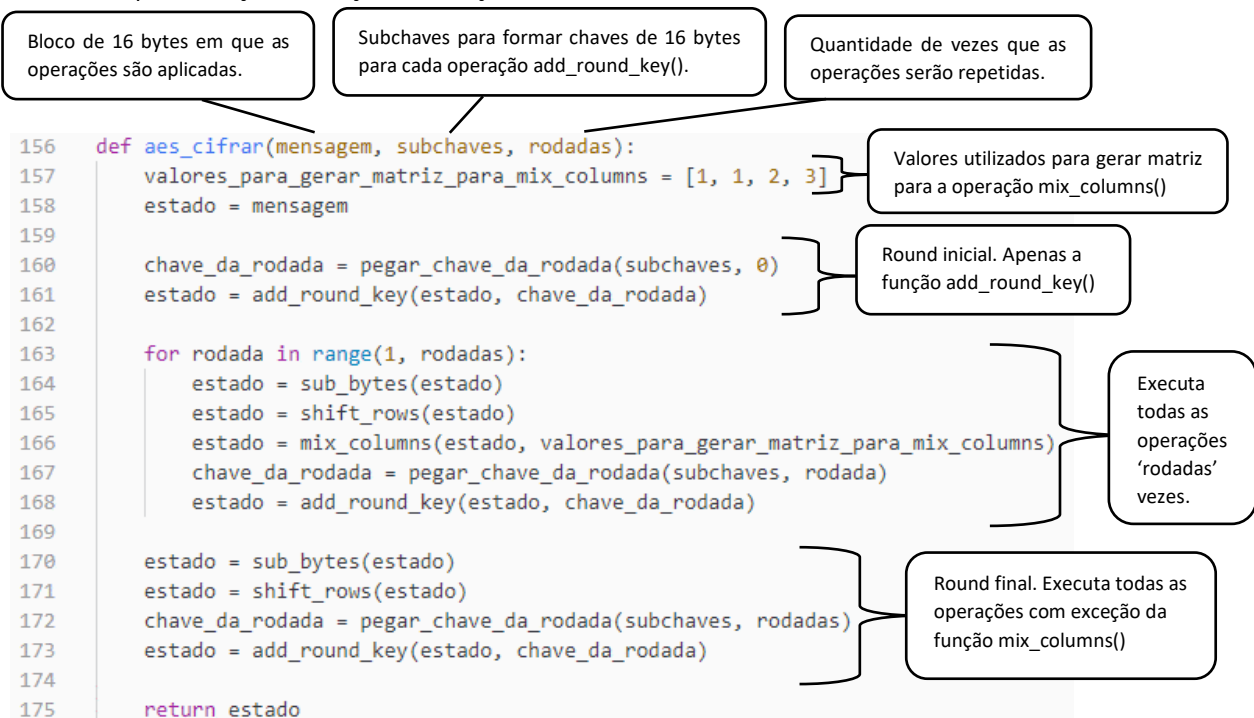
Grupo: Andre Ivan – 190084197
Fernando de Alcantara – 190125586

- Cifra AES - O Advanced Encryption Standard é um algoritmo de criptografia de blocos com chave simétrica, ou seja, a mesma chave que é usada para cifrar também é usada para decifrar. No caso do AES, o bloco de entrada tem tamanho de 16 bytes e a chave pode ter tamanho de 16, 24 e 32 bytes, o tamanho de chave usado também especifica o número de repetições sendo 10, 12 e 14 respectivamente. O funcionamento do algoritmo consiste em realizar uma série de operações repetidamente em um bloco que denominamos de “estado” (cada repetição é um round). As operações que são realizadas em cada round são: adicionar uma chave ao estado (add_round_key), substituir os bytes do estado (sub_bytes), deslocar as linhas do estado (shift_rows) e misturar as colunas do estado (mix_columns).

A decifração realiza tais operações em ordem inversa e as operações apresentam uma função inversa.

A implementação apresentada nesse relatório utilizará tamanho de chave de 16 bytes e um número de rodadas não fixo permitindo informar a quantidade de rodadas a realizar.

▪ Implementação da função de cifração AES:



▪ Função `add_round_key()`:

```
83 add_round_key = lambda estado, chave_da_rodada: [estado[i] ^ chave_da_rodada[i] for i in range(16)]
```

Essa função realiza uma operação xor bit a bit entre o estado e a chave. Ambos o estado e a chave contem 16 bytes. O operador `^` em python representa a operação xor. Por fim, nos retorna a lista de 16 bytes resultante.

- Função sub_bytes():

```
85 sub_bytes = lambda palavra: [s_box[pos] for pos in palavra]
```

Essa função substitui cada byte do estado por um byte presente na tabela de substituição [Rijndael S-box - Wikipedia](#).

- Função shift_rows():

```
68 # funcao que rotaciona a palavra n vezes
69 rotacionar = lambda palavra, n: palavra[n:] + palavra[0:n]
74 # funcao que nivela uma lista que tem sublista exemplo uma lista [[1], [2]] vira [1, 2]
75 flatten = lambda lista: [item for sublista in lista for item in sublista]
89 shift_rows = lambda estado: flatten([rotacionar(estado[i*4:(i*4)+4], i) for i in range(4)])
```

Essa função rotaciona cada linha da matriz n vezes, a primeira linha não rotaciona, a segunda rotaciona uma vez a terceira duas vezes e a quarta rotaciona 3 vezes. Note que utilizamos duas funções auxiliares, uma para efetuar a rotação de cada linha rotacionar() e outra para nivelar a lista resultante flatten().

- Função mix_columns():

```
76 # transpõe a matriz (é uma lista só que referenciada aqui como matriz)
77 pegar_matriz_transposta = lambda matriz: flatten([[matriz[(4*j)+i] for j in range(4)] for i in range(4)])

93 # funcao que realiza a operacao de multiplicacao -> galois multiplication
94 def gm(a, b):
95     p = 0
96     hiBitSet = 0
97
98     for _ in range(8):
99         if b & 1 == 1:
100             p ^= a
101             hiBitSet = a & 0x80
102             a <<= 1
103             if hiBitSet == 0x80:
104                 a ^= 0x1b
105             b >>= 1
106
107     return p % 256

135 # funcao que cria uma nova coluna dependendo do valor de vm 'valores da matriz'
136 # se vm = [1, 1, 2, 3] (gera matriz normal) realiza a operacao mix column normal
137 # se vm = [0x09, 0x0d, 0x0b, 0x0e] (gera matriz inversa) realiza a operacao mix column inversa
138 def mix_column(coluna, vm):
139     nova_coluna = [0] * 4
140     nova_coluna[0] = gm(coluna[0], vm[2]) ^ gm(coluna[1], vm[3]) ^ gm(coluna[2], vm[1]) ^ gm(coluna[3], vm[0])
141     nova_coluna[1] = gm(coluna[0], vm[0]) ^ gm(coluna[1], vm[2]) ^ gm(coluna[2], vm[3]) ^ gm(coluna[3], vm[1])
142     nova_coluna[2] = gm(coluna[0], vm[1]) ^ gm(coluna[1], vm[0]) ^ gm(coluna[2], vm[2]) ^ gm(coluna[3], vm[3])
143     nova_coluna[3] = gm(coluna[0], vm[3]) ^ gm(coluna[1], vm[1]) ^ gm(coluna[2], vm[0]) ^ gm(coluna[3], vm[2])
144     return nova_coluna

146 # funcao que embaralha a matriz de estado
147 def mix_columns(estado, valores_matriz):
148     novo_estado = pegar_matriz_transposta(estado) # transpor facilita as outras operacoes
149
150     for i in range(4):
151         novo_estado[i*4:(i*4)+4] = mix_column(novo_estado[i*4:(i*4)+4], valores_matriz)
152
153     return pegar_matriz_transposta(novo_estado) # transpõe novamente para voltar ao normal
```

Essa função realiza operações nas colunas, tal operação envolve multiplicação e adição no campo de Galois. Primeiramente, na função mix_columns() transpomos a matriz para efetuar uma manipulação mais simplificada da mesma. Depois disso, no for loop, é efetuado a operação de mix_column() para cada coluna da matriz onde vm = [1, 1, 2, 3] no caso da cifração, que será usado para gerar a matriz. Então, por fim, o que temos é uma multiplicação de matrizes 4x4 * 4x1 onde as operações de adição e multiplicação são feitas com base no campo de Galois em vez da aritmética usual.

Em código, tal adição é representada pelo operador xor (^) e a multiplicação pela função gm() (fonte da função gm() [AES Encryption \(asecuritysite.com\)](#)).

Que por fim nos gera uma nova matriz 4x1 que é a nossa nova coluna.

$$\begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

A matriz da esquerda do símbolo de = é a matriz resultante da multiplicação efetuada. Que no caso, é a nova coluna.

- Implementação da função de decifração AES:

```

178 def aes_decifrar(mensagem, subchaves, rodadas):
179     valores_para_gerar_matriz_para_mix_columns_inversa = [0x09, 0x0d, 0x0e, 0x0b]
180     estado = mensagem
181
182     chave_da_rodada = pegar_chave_da_rodada(subchaves, rodadas)
183     estado = add_round_key(estado, chave_da_rodada)
184     estado = shift_rows_inverso(estado)
185     estado = sub_bytes_inversa(estado)
186
187     for rodada in range(rodadas-1, 0, -1):
188         chave_da_rodada = pegar_chave_da_rodada(subchaves, rodada)
189         estado = add_round_key(estado, chave_da_rodada)
190         estado = mix_columns(estado, valores_para_gerar_matriz_para_mix_columns_inversa)
191         estado = shift_rows_inverso(estado)
192         estado = sub_bytes_inversa(estado)
193
194     chave_da_rodada = pegar_chave_da_rodada(subchaves, 0)
195     estado = add_round_key(estado, chave_da_rodada)
196
197     return estado

```

Para a decifração embora as operações executadas são inversas as da cifração, elas ainda são operações extremamente parecidas. Então, tendo isso em mente, a explicação sobre cada uma será breve.

- Função add_round_key():

A operação de xor é a inversa dela mesma. Então, por exemplo, executando ela duas vezes com a mesma chave, obtemos o resultado original.

- Função sub_bytes_inversa():

```

87 sub_bytes_inversa = lambda estado: [s_box_inversa[pos] for pos in estado]

```

A inversa da função sub_bytes(), simplesmente consulta a tabela de substituição inversa [Rijndael S-box - Wikipedia](#) na hora de substituir cada byte do estado.

- Função shift_rows_inverso():

```

91 shift_rows_inverso = lambda estado: flatten([rotacionar(estado[i*4:(i*4)+4], -i) for i in range(4)])

```

A inversa da função shift_rows() executa a rotação das linhas para o lado contrario a ela (-i).

- Função mix_columns():

A inversa da função mix_columns() é simplesmente efetuar as operações de mix_columns() utilizando a matriz inversa, ou seja $vm = [0x09, 0x0d, 0x0e, 0x0b]$ o que nos gera a matriz inversa.

$$\begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

A matriz utilizada agora é inversa a matriz apresentada para a etapa de cifração.

- Modos de operação ECB e CTR:

- Electronic Codebook - ECB:

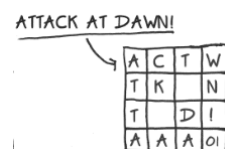
```

200 def ecb(chave, blocos, rodadas, modo):
201     subchaves = gerar_subchaves(chave, rodadas)
202
203     blocos_gerados = []
204     for bloco in blocos:
205         bloco = pegar_matriz_transposta(bloco)
206         if modo == 'cifrar':
207             novo_bloco = pegar_matriz_transposta(aes_cifrar(bloco, subchaves, rodadas))
208         elif modo == 'decifrar':
209             novo_bloco = pegar_matriz_transposta(aes_decifrar(bloco, subchaves, rodadas))
210         blocos_gerados.append(novo_bloco)
211
212     return blocos_gerados # retorna os blocos cifrados ou decifrados, depende do valor de modo = cifrar | decifrar

```

Nesse modo, a mensagem é dividida em blocos de 16 bytes e cada um é cifrado separadamente.

Note que, a função pegar_matriz_transposta() é utilizada para posicionar os bytes na direção vertical na matriz para utilização no aes.



▪ Contador – CTR:

```

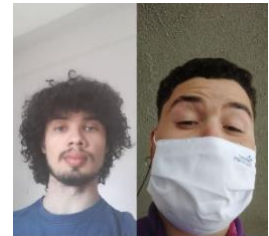
215 def ctr(chave, blocos, rodadas):
216     subchaves = gerar_subchaves(chave, rodadas)
217     contador = [0x0] * 16
218
219     blocos_gerados = []
220     for bloco in blocos:
221         res = pegar_matriz_transposta(aes_cifrar(pegar_matriz_transposta(contador.copy()), subchaves, rodadas))
222         novo_bloco = palavra_xor_outra_palavra(res, bloco)
223         blocos_gerados.append(novo_bloco)
224         contador = incrementar(contador.copy())
225
226     return blocos_gerados #

```

Nesse modo, os blocos de entrada da cifração são contadores que, após serem cifrados são utilizados para realizar a operação de xor com a mensagem que se queira cifrar ou decifrar. Certas implementações utilizam um nonce (que é um número arbitrário usado apenas uma vez) concatenado ao contador, não o utilizamos aqui.

• Testes:

O teste consiste em cifrar os bytes de uma imagem no modo ECB com 1, 5, 9 e 13 rodadas do AES e efetuar o mesmo para o modo CTR. Renderizando os resultados e gerando um hash para cada. Os formatos de imagem utilizados para os testes foram os jpg e bmp, no caso do jpg foram cifrados os bytes que correspondem ao entropy-coded segment (ciframos os bytes errados no que diz respeito ao que desejavamos obter cifrando uma imagem, no entanto deu uns resultados legais então resolvemos deixar) diferentemente de imagens no formato bmp em que ciframos os bytes da imagem obtendo os resultado que de fato nos interessa.



Chave utilizada:

0x5f 0xc3 0x7b 0xb7 0x16 0xd3 0xb5 0x2f 0x4c 0x4e 0x87 0xa9 0xb9 0x9c 0x77 0x50

Selfie utilizada:

Resultados que nos interessamos:

SELFIE EM BMP:


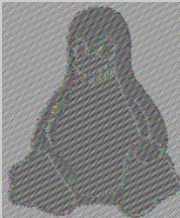
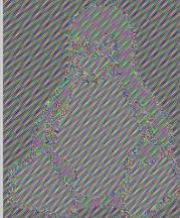

ECB:

Rodadas	1	5	9	13
Resultado				
Hash	ffd4c452b9555190c25c b5ff95b8e2de9774f9ea	9616983eead89ca4748 ea5ccf9786851ccb2ae5 0	77c4a21b889aee3f10d 7eda9248d3e237f5e4a d4	ef2279c60073d392994 4e9d060dca5bebc12e8 b9

CTR:

Rodadas	1	5	9	13
Resultado				
Hash	1b6e25b3f51e6d21e7a d586894507c32812a46 77	65a8062a703b315a77 8267cfb199438ba2306 8eb	fa6122c6da4a2a34699 5d2047df331ad8f0394 3b	c36bfd41f74fc247f83fb 6b6c06d0cc1b9de2c4a

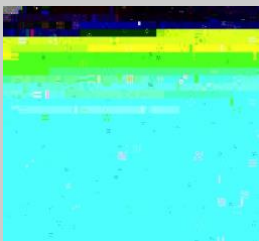



Resultados extras:

Rodadas	TUX	TUX ECB 1 RODADA	TUX ECB 5 RODADAS	TUX CTR 5 RODADAS
Resultado				
Hash		61310cf25c772887729 8ce48ea0242ad0bb5a3 fd	f551feeda029e7ba4ed 822966ea683b49658b 300	bc5c096a9f5ed7bf225 9d07c72d2a505e28777 fc




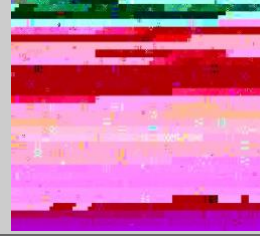
Resultado extra obtido cifrando bytes “errados” no formato jpg:

SELFIE EM JPG:

ECB:

Rodadas	1	5	9	13
Resultado				
Hash	d02ce3bea2e1bbaf089 da9107b5ba19b6b0fde ba	a0d1354d27e62ad21b d1af5cd81368885fd0b 238	d0d74a75b1f3d6ccad7 1df55130ce289a64b6d 73	8a3751588755849b72 a1893886be287dfc83a a4e

CTR:

Rodadas	1	5	9	13
Resultado				
Hash	240b30eddd6d62ec12 93c283a3b0fb4fcc4d3d 4f	5e126724a070c4be1c6 b71eb7d8c59ea377a7b cc	ded28ad05f18b627614 316ed571742bf68c6bd e9	8f74881c81273c4fe7a2 8496cd17a73e5f0b6a4 8

Referências:

Advanced Encryption Standard – Wikipédia, a enciclopédia livre ([wikipedia.org](https://pt.wikipedia.org))

Rijndael S-box - Wikipedia

Block cipher mode of operation - Wikipedia

Advanced Encryption Standard (AES) - YouTube

Como funciona o algoritmo de criptografia AES? - Stack Overflow em Português

AES Encryption (asecuritysite.com)

Análise do AES e sua criptoanálise diferencial (ufrgs.br)

Implementing AES (nindalf.com)

A Stick Figure Guide to the Advanced Encryption Standard (AES) (moserware.com)

Rijndael key schedule | Crypto Wiki | Fandom

Understanding and Decoding a JPEG Image using Python - Yasoob Khalid

Bitwise Operators in Python – Real Python