Relatorio: Cifra de Vigenère

Igor Bastos de Oliveira, 18/0122177

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)

1. Introdução

O processo de cifração e decifração AES, Advanced Encryption Standard (Padrão Avançado de Criptografia), faz parte da família dos algoritmos de criptografia simétrica e é um dos mais amplamente utilizados no mundo. Seu uso mais comum é na proteção de informações confidenciais em uma variedade de aplicações, sendo usadas em comunicações de segurança e armazenamento de dados sensíveis.

Em 2001 o AES foi usado como substituto do DES (data encryption standard) pelo instituto de padrões e tecnologia dos estados unidos, NIST, isso porque o AES é mais eficiente e seguro em comparação com o DES. Considerando que ele opera usando uma chave secreta compartilhada entre as partes que desejam trocar informação de forma segura. O processo de cifração envolve transformação dos dados originais em dados inelegíveis usando vários processos por camadas e a decifração reverter esse mesmo processo usando a mesma chave.

De maneira introdutória o funcionamento do AES utiliza do conceito da estrutura de blocos, ou seja, o sistema divide os dados em blocos fixos antes de aplicar a criptografia. As chaves de criptografia podem variar de tamanho entre 128, 192 e 256 bits dependendo do nível de segurança e isso torna o aes também um sistema de criptografia mais flexível. Neste relatório será trabalhado o processo de construção da criptografia e descriptografia usando o AES e também vai ser mostrado a teoria e a prática desse processo.

2. Criptografia

Como foi dito anteriormente, a criptografia AES envolve o processo de blocos fixos. O tamanho do bloco pode ser de 128 a 256 bits e os dados a serem criptografados precisam estar no tamanho apropriado. Usando um processo matemático complexo a criptografia AES transforma dados originais em dados inelegíveis que só podem ser revertidas tendo a chave de descriptografia em mãos. A criptografia envolve vários processos que foram utilizados na criação do codigo para uso desta aplicação.

Separação dos blocos: Já que o AES opera em blocos fixos. O tamanho do bloco escolhido foi de 128 bits mas poderia chegar ate 256. Foi usado o método ECB para fazer a criptografia usando a separação de blocos

Também foi feita a operação de blocos usando o CTR, um modo de operação. Usando um nonce e um contador para gerar uma sequência de blocos de cifra que são combinadas com os blocos de dados de entrada

```
def ECB(self, word, num, op):
  aux = []

  for i in range(len(word)):
```

```
        for j in range(len(word[i])):
            aux.append(word[i][j][0])
            aux.append(word[i][j][1])
            aux.append(word[i][j][2])

    while (len(aux) % 16 != 0):
        aux.append(0x00)

    wordfinal = []

    for i in range(int(len(aux) / 16)):
        blocos = []

        for j in range(16):
            blocos.append(aux[(i * 16) + j])

        if op == 1:
            wordaux = self.AES(blocos, num)

        else:
            wordaux = self.AES_inv(blocos, num)

        for k in range(16):
            wordfinal.append(wordaux[k])

    return wordfinal

    def CTR(self, word, num, noncekey):
    aux = []
    cont = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0

    for i in range(len(word)):
        for j in range(len(word[i])):
            aux.append(word[i][j][0])
            aux.append(word[i][j][1])
            aux.append(word[i][j][2])

    while (len(aux) % 16 != 0):
        aux.append(0x00)

    wordfinal = []
    noncekeyaux = noncekey

    for i in range(int(len(aux) / 16)):
        fon = []
        wordaux = []
```

```
        for j in range(16):
            fon.append(aux[(i * 16) + j])

        auxvec = self.AES(noncekeyaux, num)



        for j in range(16):
            wordaux.append(fon[j] ^ auxvec[j])

        for j in range(16):
            wordfinal.append(wordaux[j])

        for j in range(16):
            cont[j] = (cont[j] + 1) & 0xFF



        for j in range(16):
            noncekeyaux[j] = noncekey[j] ^ cont[j]



    return wordfinal
```

Chave de Criptografia: Antes do processo de cifração, foi criado uma chave para criptografar, o programa funciona com a opção de criação da chave, usar chave própria e uma chave já criada em um arquivo txt.

```
    def nonce(self):
    aux = []

    for i in range(16):
        aux.append(random.randrange(0, 255))

    return aux
```

Subchaves: Foi usada a chave principal para a criação de subchaves para cada etapa do processo de cifração.

```
    def key_expansion(self, key, num_rounds):
    expanded_key = []
    control = 0

    for i in range((num_rounds + 1) * 16):
        if i < 16:
```

```
            expanded_key.append(key[i])
        elif i % 16 == 0:
            round_constant = self.key_expansion_constants[int(i / 1
            expanded_key.append(
                self.substitution_box[expanded_key[i - 3]] ^ expand
            control = 1
        elif control == 1:
            expanded_key.append(self.substitution_box[expanded_key[
            control = 2
        elif control == 2:
            expanded_key.append(self.substitution_box[expanded_key[
            control = 3
        elif control == 3:
            expanded_key.append(self.substitution_box[expanded_key[
            control = -1
        else:
            expanded_key.append(expanded_key[i - 4] ^ expanded_key[

    self.exp = expanded_key

def around_key(self, word, round_number):
    j = round_number * 16

    word[:] = [word[i] ^ self.exp[j + i] for i in range(16)]
```

Rounds: Foi programado para gerar números de rounds para cifração que o usuário vai escolher, o programa funcionou bem até a numeração 14 de rounds dando resultados satisfatórios que serão explanados na etapa de teste.

Substituição de bytes: Em Cada rodada ocorre o processo de cada byte do bloco é substituído de acordo com a tabela específica para a substituição que chamamos de substitution box. Permutação: Os bytes dos blocos são permutados entre si fazendo com que os blocos entrem em um processo de reorganização mais robusto

```
    def sub_bytes(self, word):
    word[:] = [self.substitution_box[byte] for byte in word]
```

Misturas das colunas: Ocorre o processo de mistura das colunas para dispersar as informações deixando mais difícil a descoberta sem o uso da chave Por fim ocorre todos os processos novamente devido a quantidade de rodadas que foi definida até chegar na rodada final

```
        def mix_column(self, word):
    mixed_word = [0] * 16
    gmul = self.Gmul

    for i in range(4):
        base_idx = i * 4
        mixed_word[base_idx] = gmul(0x02, word[base_idx]) ^ gmul(0x
```

```
                                word[base_idx + 3]
        mixed_word[base_idx + 1] = word[base_idx] ^ gmul(0x02, word

                                    word[base_idx + 3]
        mixed_word[base_idx + 2] = word[base_idx] ^ word[base_idx +


        mixed_word[base_idx + 3] = gmul(0x03, word[base_idx]) ^ wor


    return mixed_word
```

Rodada final: Na última rodada não é realizada a mistura das colunas e apenas a substituição, permutação e adição da chave.

3. Descriptografia

O processo de descriptografia é muito similar ao da criptografia, em modos simples é basicamente o caminho inverso, ela tem como objetivo fundamental a recuperação dos blocos dos dados originais e desta forma é necessário tomar o caminho inverso da criptografia. Os passos empregados no código foram

AddRoundKey inverso: Etapa inicial onde ocorre a operação inversa do addRoundkey. Ela envolve a aplicação de subchaves da rodada final, seguindo de uma retroatividade atraves das subchaves das rodadas anteriores em ordem inversa.

ShiftRows inverso: A operação invertida da permutação, desfazendo e recuperando os dados permutados da etapa de criptografia

```
d   def shift_rows_inv(self, word):

aux = word[13]
word[13] = word[9]
word[9] = word[5]
word[5] = word[1]
word[1] = aux

aux = word[14]
aux2 = word[10]
word[14] = word[6]
word[10] = word[2]
word[6] = aux
word[2] = aux2

aux = word[15]
aux2 = word[11]
aux3 = word[7]
word[15] = word[3]
```

```
        word[11] = aux
        word[7] = aux2
        word[3] = aux3

    def mix_column_inv(self, word):
        aux = []

        mix_columns_matrix = [
            [14, 11, 13, 9],
            [9, 14, 11, 13],
            [13, 9, 14, 11],
            [11, 13, 9, 14]
        ]

        for i in range(4):
            aux.extend(
                [
                    self.Gmul(mix_columns_matrix[i][0], word[i * 4]) ^

                    self.Gmul(mix_columns_matrix[i][2], word[i * 4 + 2]

                ]
            )

        return aux
```

SubBytes inverso: Desta vez usando a tabela inversa do bytes para restaurar os bytes emparelhados MixColumns inverso: Restaura a posição original das colunas dos blocos de dados

```
def sub_bytes_inv(self, word):
substitution_box = self.inverse_substitution_box
word[:] = [substitution_box[b] for b in word]
```

O processo se repete na mesma lógica da criptografia até chegar no resultado final que é a descriptografia completa do objeto.

4. testes

Foi iniciado os testes de criptografia com imagem png para os cados de 1, 5, 9 e 13 blocos e segue os resultados:

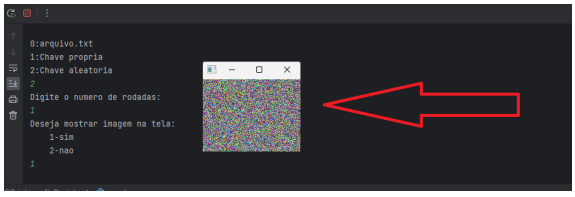

Figure 1. teste criptografia de uma rodada.

Figure 2. teste criptografia de cinco rodadas.

Figure 3. teste criptografia de nove rodadas.

Figure 4. teste criptografia de 13 rodada.

A descriptografia tambem se mostrou eficiente para o processo:

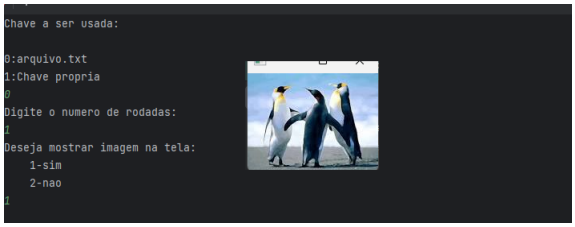

Figure 5. descriptografia teste.

5. Conclusão

O estudo e o desenvolvimento deste projeto nos proporcionaram uma visão profunda sobre as complexidades da criptografia simétrica, com foco especial no algoritmo AES. A implementação do AES não apenas demonstrou a importância de garantir a segurança das informações, mas também ressaltou a necessidade contínua de proteger a confidencialidade e integridade das comunicações em nosso mundo digital em constante evolução.

Como foi demonstrado, a segurança da informação depende, em grande parte, da maneira como as chaves de criptografia são tratadas, pois representam um elo importante

na cadeia de segurança. Além disso, a flexibilidade do AES em relação a tamanhos de chave e modos de operação ressalta a necessidade de abordagens de segurança adaptáveis que possam ser personalizadas para atender a diferentes níveis de proteção.

Resumindo, o projeto de implementação do AES não apenas enfatizou a complexidade da criptografia simétrica e seus princípios fundamentais de segurança, mas também destacou a importância de se manter atualizado em um cenário de ameaças em constante evolução. Proteger a confidencialidade e integridade das comunicações permanece uma responsabilidade crítica em nosso mundo digital, e a criptografia desempenha um papel essencial nesse esforço contínuo de garantir a segurança de nossos dados e informações.

References

AES. wikpedia. Disponível em: https://pt.wikipedia.org/wiki/Advanced$_Encryption_Standard_Aces$
$27 de outubro de 2023$.