MODELO DE CRIPTOGRAFIA POR SUBSTITUIÇÃO E PERMUTAÇÃO POR CHAVE

Djalma Farias Bastos Neto - 2222130020[1,†]

[1]*IESB, Brasília – <djalma.farias@iesb.edu.br>*

1. INTRODUÇÃO

A criptografia desempenha um papel crucial na segurança das comunicações digitais, transformando texto simples em texto cifrado e permitindo a recuperação da informação original. Neste artigo, são explorados os conceitos fundamentais de criptografia e decriptação, abrangendo algoritmos simétricos e assimétricos. A gestão de chaves e os riscos associados aos ataques de força bruta, técnica que testa todas as combinações possíveis de chaves, são discutidos, avaliando-se a eficácia dessa abordagem por meio da Notação Big O(7). Para ilustrar a aplicação prática, foi desenvolvido um algoritmo de criptografia simples em Python, sem a utilização de bibliotecas externas, com destaque para os processos de cifração e decifração de dados. Finalmente, são analisadas as implicações de segurança, proporcionando uma visão mais aprofundada sobre como a criptografia protege informações sensíveis e os desafios enfrentados nesse contexto.

Palavras-chave: *Criptografia; Decriptação; Ataques de força bruta; Gestão de chaves.*

2. MATERIAIS E MÉTODOS

Para o desenvolvimento dos algoritmos de criptografia por substituição e permutação, foi utilizado o Visual Studio Code (VSCode) (5) como IDE, juntamente com o compilador Python3 (4) para a execução do código. A implementação do algoritmo foi feita em Python, dada a sua simplicidade na manipulação de strings. O método adotado baseia-se nos princípios fundamentais da criptografia por substituição e permutação, aliados à geração de chaves.

3. REVISÃO DE LITERATURA

As técnicas de criptografia utilizadas neste trabalho baseiam-se em dois princípios fundamentais de segurança: substituição e permutação. Ambos os métodos têm uma longa história no campo da criptografia e são amplamente empregados em sistemas modernos para garantir a segurança de dados.

A ***substituição*** *é uma técnica criptográfica em que cada símbolo no texto original (plaintext) é substituído por outro símbolo de acordo com uma regra definida. O algoritmo desenvolvido faz uso dessa técnica através da função* `rodar_alfabeto()`*, que realiza a substituição de cada letra do alfabeto por outra deslocada de acordo com um valor baseado na chave gerada. Essa abordagem é similar ao clássico Cifra de César, onde um deslocamento fixo é aplicado a cada letra, mas com a diferença de que o deslocamento varia com base em uma chave aleatória, aumentando a complexidade do sistema de substituição.*

Por outro lado, a ***permutação*** *refere-se à reordenação de elementos em um texto ou sequência de acordo com um padrão específico. No código implementado, após a aplicação da substituição, os caracteres resultantes são reorganizados com base em uma chave de permutação gerada aleatoriamente pela função* `gerar_chave_aleatoria()`*. Essa chave é essencialmente uma sequência de números que determina a nova posição de cada caractere no texto criptografado. A reordenação (ou permutação) dos caracteres é realizada após a substituição, adicionando uma segunda camada de ofuscação aos dados. Isso aumenta a resistência contra ataques que buscam descobrir padrões simples no texto cifrado.*

Historicamente, a combinação de substituição e permutação é uma abordagem poderosa e amplamente utilizada em sistemas de criptografia, como a Cifra de Vigenère e o DES (Data Encryption Standard). A estratégia empregada no código deste artigo segue uma linha similar, criando uma estrutura de segurança robusta através de múltiplas camadas de cifração. Ao aplicar substituição seguida de permutação, o modelo garante que o texto cifrado seja difícil de decifrar sem a chave correta, uma vez que a inversão de uma dessas operações por si só não revelaria o texto original.

A chave de permutação, gerada de forma aleatória, e o uso de uma substituição variável garantem um grau elevado de entropia no processo, reduzindo a previsibilidade do texto cifrado e a suscetibilidade a ataques de força bruta ou análise de frequência. Dessa forma, as técnicas combinadas de substituição e permutação criam uma estrutura de criptografia mais segura e resistente, mesmo em implementações simples como a apresentada neste trabalho.

4. ENCRIPTAÇÃO

Atenção

O método utilizado deve ser aplicado somente em mensagens de texto plano que não contenham acentuação, caracteres especiais, pontuação ou letras maiúsculas.

O processo de criptografia neste modelo consiste em quatro etapas principais:

1. ***Leitura e Mapeamento:*** *Inicialmente, cada letra da mensagem M é mapeada para um vetor de tamanho M, com posições que vão de 0 até $M-1$. Esse vetor representa a mensagem em uma forma numérica, facilitando as etapas subsequentes (ver Figura 1).*

Figura 1. Leitura e Mapeamento.

0	1	2	3	4	5	6
b	o	m		d	i	a

2. ***Geração da Chave Aleatória:*** *Em seguida, uma chave C aleatória é gerada, com o mesmo tamanho que a mensagem M. A chave é representada como um vetor contendo uma combinação aleatória de números variando de 0 até $M-1$ (ver Figura 2).*

Figura 2. Geração da Chave.

5	1	0	3	2	4	6

3. ***Transposição com Base na Chave:*** *Nesta etapa, o primeiro número da chave $C[0]$ é utilizado para realizar a transposição da mensagem M. Cada letra da mensagem é rearranjada de acordo com a ordem especificada por $C[0]$. Espaços em branco são substituídos por "#"durante este processo (ver Figura 3).*

Figura 3. Processo de Transposição por Chave.

b	o	m		d	i	a
+5	+5	+5		+5	+5	+5
g	t	r	#	i	n	f

4. ***Reordenação com Base na Chave:*** *Finalmente, após a transposição, a posição de cada letra é ajustada com base na chave gerada. O resultado dessa etapa é a mensagem criptografada M_c, que reflete as alterações feitas pelas etapas anteriores (ver Figura 4).*

Figura 4. Processo de Reordenação por Chave.

0	1	2	3	4	5	6
g	t	r	#	i	n	f

5	1	0	3	2	4	6
n	t	g	#	r	i	f

O algoritmo de criptografia começa com a leitura da mensagem, a geração de uma chave aleatória e a criptografia de uma mensagem por meio de transposição e substituição de caracteres. Ele utiliza funções específicas para gerar a chave, ler o conteúdo de arquivos e executar a criptografia. Vamos explicar detalhadamente cada bloco do código e sua função no processo de criptografia.

```
1  import random
2
3  def gerar_chave_aleatoria(tamanho,
       nome_arquivo):
4      chave = list(range(tamanho))
5      random.shuffle(chave)
6      chave_string = ' '.join(map(str
       , chave))
7
8      with open(nome_arquivo, 'w') as
       arquivo:
9          arquivo.write(chave_string)
10     return chave
```

Explicação:

- `list(range(tamanho))`: *Cria uma lista de números inteiros de 0 até* `tamanho-1`*, onde* `tamanho` *é o número de caracteres da mensagem.*
- `random.shuffle(chave)`: *Embaralha essa lista de forma aleatória, gerando a chave que será usada na transposição dos caracteres.*
- `Gravação da chave`: *A chave é convertida em uma string e salva em um arquivo de texto. Esse arquivo pode ser reutilizado durante a decriptação.*

Para criptografar uma mensagem, é necessário primeiro ler o seu conteúdo de um arquivo. A função `ler_arquivo_para_string` *realiza essa tarefa.*

```
1  def ler_arquivo_para_string(
       nome_arquivo):
2      conteudo_completo = ""
3      try:
4          with open(nome_arquivo, 'r'
       ) as arquivo:
```

```
            for linha in arquivo:
                conteudo_completo += linha.strip() + " "
    except FileNotFoundError:
        print(f"Arquivo '{nome_arquivo}' não encontrado.")
    return conteudo_completo.strip()
```

Explicação:

- ***Leitura do arquivo:*** *A função tenta abrir o arquivo especificado e ler seu conteúdo, concatenando todas as linhas em uma única string.*
- ***Tratamento de exceções:*** *Se o arquivo não for encontrado, uma mensagem de erro é exibida. Isso é importante para garantir que o código lida adequadamente com possíveis problemas de I/O.*

A função `criptografar` *é responsável por realizar a criptografia da mensagem. Ela utiliza a chave gerada para transpor os caracteres e uma técnica de substituição simples para criptografar cada letra, utilizando a função auxiliar* `rodar_alfabeto`*.*

```
def criptografar(chave, conteudo):
    conteudo_criptografado = ''
    conteudo_reordenado = ''
    for indice, letra in enumerate(conteudo):
        if letra == ' ':
            conteudo_criptografado = conteudo_criptografado + '#'
        else:
            conteudo_criptografado = conteudo_criptografado + rodar_alfabeto(chave[0], letra)

    conteudo_reordenado = conteudo_criptografado
    conteudo_reordenado_list = list(conteudo_reordenado)
    for indice, numero in enumerate(chave):
        conteudo_reordenado_list[indice] = conteudo_criptografado[numero]
    conteudo_reordenado = ''.join(conteudo_reordenado_list)

    with open('mensagem_criptografada.txt', 'w') as arquivo:
        arquivo.write(conteudo_reordenado)
    return chave
```

Explicação:

- ***Substituição de espaço por #:*** *Durante a criptografia, os espaços na mensagem original são substituídos por #, preservando a estrutura do texto durante a transposição.*
- ***Substituição de letras:*** *A função* `rodar_alfabeto` *é usada para criptografar cada letra da mensagem de acordo com o valor da chave.*
- ***Reordenamento da mensagem:*** *Após a substituição, o conteúdo é reordenado utilizando a chave gerada. Cada caractere é realocado conforme o valor correspondente na chave, finalizando o processo de transposição.*

A função `rodar_alfabeto` *é responsável por alterar o valor de cada letra da mensagem, aplicando uma rotação no alfabeto com base em um índice fornecido pela chave.*

```
import string

def rodar_alfabeto(indice, letra):
    alfabeto = string.ascii_lowercase
    tamanho_alfabeto = len(alfabeto)

    if letra not in alfabeto:
        print(letra)
        raise ValueError("A letra fornecida não está no alfabeto.")

    indice_inicial = alfabeto.index(letra)

    indice_rodado = (indice_inicial + indice) % tamanho_alfabeto

    return alfabeto[indice_rodado]
```

Explicação:

- ***Rotação de letras:*** *Esta função aplica uma rotação no alfabeto usando o valor da chave como base. Por exemplo, se a chave for 3 e a letra for 'a', a função retornará 'd'.*
- ***Verificação de erros:*** *A função verifica se a letra está no alfabeto, lançando uma exceção se a entrada for inválida.*

No final do código, as funções são usadas para ler uma mensagem de um arquivo, gerar uma chave de tamanho adequado e criptografar o conteúdo.

```
arquivo_mensagem = 'mensagem.txt'
arquivo_chave = 'chave.txt'

conteudo = ler_arquivo_para_string(
    arquivo_mensagem)

tamanho_string = len(conteudo)

chave_aleatoria =
    gerar_chave_aleatoria(
    tamanho_string, arquivo_chave)

conteudo_criptografado =
    criptografar(chave_aleatoria,
    conteudo)
```

Explicação:

- ***Leitura da mensagem:*** *O conteúdo do arquivo* `mensagem.txt` *é lido e armazenado em* `conteudo`.
- ***Geração da chave:*** *A chave é gerada com base no tamanho da mensagem e armazenada no arquivo* `chave.txt`.
- ***Criptografia:*** *A mensagem é criptografada usando a chave gerada e salva no arquivo* `mensagem_criptografada.txt`.

5. DECRIPTAÇÃO

1. ***Leitura da Mensagem Criptografada:*** *Inicialmente, a mensagem criptografada M_c é lida e convertida novamente para um vetor de tamanho M, com as posições dos caracteres organizadas de acordo com a ordem em que foram transpostos. A mensagem ainda contém os espaços representados por "#"(ver Figura 5).*

Figura 5. Leitura da Mensagem Criptografada.

0	1	2	3	4	5	6
n	t	g	#	r	i	f

2. ***Reaplicação da Chave para Reordenação:*** *A mesma chave C usada na criptografia é essencial para a reordenação dos dados. O vetor da chave orienta a reversão do processo de reordenação, permitindo que os elementos do texto cifrado sejam reposicionados corretamente. Essa chave garante que o processo de decriptação mantenha a integridade dos dados, assegurando que o resultado final corresponda exatamente ao texto original. Portanto, a correta aplicação da chave C é fundamental para recuperar a informação de forma precisa e completa (ver Figura 6).*

Figura 6. Reaplicação da Chave para Reordenação.

5	1	0	3	2	4	6
n	t	g	#	r	i	f

0	1	2	3	4	5	6
g	t	r	#	i	n	f

3. ***Inversão da Transposição:*** *Com a mensagem reordenada, a transposição realizada na criptografia é revertida. Nesta etapa, o vetor de transposição da chave C[0] é aplicado inversamente para retornar as letras às suas posições originais. Os caracteres "#"são substituídos por espaços, recuperando a estrutura original da mensagem (ver Figura 7).*

Figura 7. Inversão da Transposição.

g	t	r	#	i	n	f
-5	-5	-5	-5	-5	-5	-5
b	o	m		d	i	a

4. ***Mensagem Original Recuperada:*** *Após reverter o processo de transposição e reordenação, a mensagem original M é recuperada. As letras e espaços estão agora na ordem original, correspondendo ao texto antes da criptografia (ver Figura 8).*

Figura 8. Mensagem Original Recuperada.

b	o	m		d	i	a

O algoritmo de decriptação começa com a leitura da chave, seguida pela reordenação da mensagem criptografada e, finalmente, a substituição de caracteres para restaurar a mensagem original. Vamos explicar detalhadamente cada bloco do código e sua função no processo de decriptação.

```
import string

def ler_arquivo(nome_arquivo):
    conteudo_completo = ""
    try:
```

```
        with open(nome_arquivo, 'r') as arquivo:
            for linha in arquivo:
                conteudo_completo += linha.strip() + " "
    except FileNotFoundError:
        print(f"Arquivo '{nome_arquivo}' não encontrado.")
        return conteudo_completo.strip()
```

Explicação:

- ***Leitura do arquivo:*** *A função tenta abrir o arquivo especificado e ler seu conteúdo, concatenando todas as linhas em uma única string.*
- ***Tratamento de exceções:*** *Se o arquivo não for encontrado, uma mensagem de erro é exibida, garantindo que o código lida adequadamente com possíveis problemas de I/O.*

A função **descriptografar** *é responsável por realizar a decriptação da mensagem criptografada. Ela utiliza a chave lida para reordenar os caracteres e substituir os caracteres especiais de volta aos espaços.*

```
def descriptografar(chave, mensagem_criptografada):
    chave_lista = list(map(int, chave.split()))
    reordenada = ''

    for i in range(len(chave_lista)):
        reordenada += mensagem_criptografada[chave_lista.index(i)]

    conteudo_descriptografado = ''

    for indice, letra in enumerate(reordenada):
        if letra == '#':
            conteudo_descriptografado += ' '
        else:
            conteudo_descriptografado += rodar_alfabeto(int(chave_lista[0]), letra)

    with open('mensagem_descriptografada.txt', 'w') as arquivo:
        arquivo.write(conteudo_descriptografado)
```

Explicação:

- ***Reordenação da mensagem:*** *A função reordena a mensagem criptografada com base na chave lida.*
- ***Substituição de letras:*** *Durante a decriptação, os caracteres são substituídos de volta usando a função* **rodar_alfabeto** *para restaurar a mensagem original.*

A função **rodar_alfabeto** *altera o valor de cada letra da mensagem, aplicando uma rotação no alfabeto com base em um índice fornecido pela chave.*

```
def rodar_alfabeto(indice, letra):
    alfabeto = string.ascii_lowercase
    tamanho_alfabeto = len(alfabeto)

    if letra not in alfabeto:
        raise ValueError("A letra fornecida não está no alfabeto.")

    indice_inicial = alfabeto.index(letra)

    indice_rodado = (indice_inicial - indice) % tamanho_alfabeto

    return alfabeto[indice_rodado]
```

Explicação:

- ***Rotação de letras:*** *Esta função aplica uma rotação no alfabeto usando o valor da chave como base, invertendo o processo de criptografia.*
- ***Verificação de erros:*** *A função verifica se a letra está no alfabeto, lançando uma exceção se a entrada for inválida.*

No final do código, as funções são usadas para ler a mensagem criptografada e a chave, e então realizar a decriptação.

```
arquivo_mensagem_criptografada = 'mensagem_criptografada.txt'
arquivo_chave = 'chave.txt'

chave = ler_arquivo(arquivo_chave)

mensagem_criptografada = ler_arquivo(arquivo_mensagem_criptografada)
```

```
descriptografar(chave,
    mensagem_criptografada)
```

Explicação:

- ***Leitura da mensagem criptografada:*** *O conteúdo do arquivo* `mensagem_criptografada.txt` *é lido e armazenado em* `mensagem_criptografada`.
- ***Leitura da chave:*** *A chave é lida do arquivo* `chave.txt`.
- ***Decriptação:*** *A mensagem é decriptografada usando a chave lida e o resultado é salvo no arquivo* `mensagem_descriptografada.txt`.

6. FORÇA BRUTA

Atenção

Este algoritmo realiza todas as combinações possíveis de uma mensagem, além de aplicar rotações em cada uma delas. É recomendado utilizar mensagens curtas, pois a complexidade computacional cresce exponencialmente com o tamanho da entrada.

A técnica de força bruta consiste em gerar todas as permutações de uma mensagem criptografada e, em seguida, aplicar rotações de caracteres para testar diferentes possibilidades de decifração. A seguir, o código correspondente a esse processo é apresentado e explicado em suas diversas etapas.

1. ***Leitura da mensagem criptografada:*** *O conteúdo da mensagem criptografada é lido a partir de um arquivo de texto (ver Figura 9).*

Figura 9. Leitura da Mensagem Criptografada.

0	1
h	j

2. ***Geração de combinações:*** *O algoritmo gera todas as combinações possíveis dos caracteres da mensagem criptografada (ver Figura 10).*

Figura 10. Geração de combinações.

1	h	j
2	j	h

3. ***Aplicação de rotações:*** *Em seguida, com base no tamanho da mensagem, aplica-se uma rotação de caracteres a cada combinação gerada (ver Figura 11).*

Figura 11. Aplicação de rotações.

h	j
-1	-1
g	i

h	j
-2	-2
f	h

j	h
-1	- 1
i	g

j	h
-2	- 2
h	f

A seguir, apresenta-se o código Python que implementa o método de força bruta:

```
def ler_arquivo(nome_arquivo):
    conteudo_completo = "" try: with
     open(nome_arquivo, 'r') as
    arquivo: for linha in arquivo:
    conteudo_completo += linha.strip
    () + " " except
    FileNotFoundError: print(f"
    Arquivo '{nome_arquivo}' não
    encontrado.") return
    conteudo_completo.strip()
```

Explicação:

- ***Leitura do arquivo:*** *A função* `ler_arquivo` *abre o arquivo de texto contendo a mensagem criptografada e lê seu conteúdo, tratando também possíveis erros relacionados à ausência do arquivo.*

O algoritmo de força bruta, a seguir, gera todas as combinações de uma string:

```
if len(mensagem) == 1:
    return [mensagem]

combinacoes = []

for i in range(len(mensagem)):

    char_atual = mensagem[i]

    resto = mensagem[:i] + mensagem
    [i+1:]

    for p in forcaBruta(resto):
        combinacoes.append(
    char_atual + p)

return combinacoes
```

Explicação:

- ***Geração de combinações:*** *A função recursiva* `forcaBruta` *gera todas as combinações possíveis dos caracteres da mensagem, dividindo a string em um*

caractere fixo e o restante da mensagem, o qual é processado novamente de forma recursiva.

A seguir, apresenta-se a função que realiza a rotação das letras:

```
alfabeto = string.ascii_lowercase
tamanho_alfabeto = len(alfabeto)

if letra not in alfabeto:
    return letra

indice_inicial = alfabeto.index(letra)

indice_novo = (indice_inicial - numero) % tamanho_alfabeto

return alfabeto[indice_novo]
```

Explicação:

- ***Rotação de letras:*** *A função* `rodar_letra` *ajusta cada letra da mensagem de acordo com um número específico de rotações, deslocando-a no alfabeto. Caracteres especiais, como #, são transformados em espaços.*

Por fim, aplica-se as rotações geradas a todas as combinações obtidas:

```
for combinacao in vetor_combinacoes:
    tamanho = len(combinacao)
    rotacoes_combinacao = []

    for n in range(0, tamanho):
        nova_combinacao = ''.join(rodar_letra(letra, n) for letra in combinacao)
        rotacoes_combinacao.append(nova_combinacao)

    resultado_final.append(rotacoes_combinacao)

return resultado_final
```

Explicação:

- ***Aplicação das rotações:*** *Para cada combinação gerada pela função de força bruta, são aplicadas diferentes rotações a cada uma das letras, e os resultados são armazenados em um vetor.*

Por fim, o código principal integra as diferentes etapas:

```

possibilidades = forcaBruta(mensagem_criptografada)
    resultado = aplicar_todas_as_rotacoes(possibilidades)

with open("combinacoes.txt", "w") as file:
for index, combinacao in enumerate(resultado):
    for indice, conteudo in enumerate(combinacao):
        file.write(f" - {combinacao[indice]}\n")
```

Explicação:

- ***Execução final:*** *Neste trecho, a mensagem criptografada é lida de um arquivo, as combinações de força bruta são geradas, as rotações são aplicadas, e os resultados finais são salvos em um arquivo de saída (*`combinacoes.txt`*).*

7. CONCLUSÕES

O desenvolvimento de um algoritmo de criptografia baseado em transposição e substituição de caracteres, implementado em Python, demonstrou a viabilidade de gerar e aplicar chaves aleatórias para proteger mensagens de texto simples. O método implementado permite a criptografia de conteúdo através da substituição de letras por posições alteradas no alfabeto e reorganização dos caracteres conforme uma chave gerada. No entanto, o algoritmo é limitado a mensagens sem acentuação, caracteres especiais, letras maiúsculas ou pontuação, o que impõe restrições ao seu uso em cenários reais mais complexos.

Esse código fornece uma abordagem educacional prática para a compreensão dos fundamentos da criptografia, incluindo a importância da geração de chaves aleatórias e a manipulação de strings. Futuras melhorias poderiam incluir o suporte para um conjunto mais amplo de caracteres e a adição de técnicas mais avançadas de criptografia para garantir maior segurança, como o uso de cifragem assimétrica.

A aplicação prática deste código destaca o poder da programação em Python para simular e demonstrar algoritmos criptográficos, fornecendo uma base sólida para o entendimento das técnicas envolvidas na proteção de informações sensíveis.

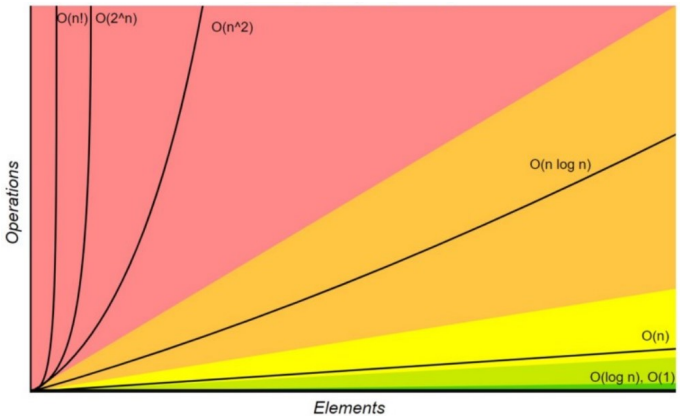
A técnica de força bruta consiste em tentar todas as combinações possíveis de uma mensagem criptografada até encontrar a solução correta. No caso do algoritmo desenvolvido, essa técnica implicaria testar todas as permu-

tações do conteúdo criptografado, verificando cada uma até descobrir a mensagem original.

A complexidade de quebrar esse algoritmo, seguindo a Notação Big O, é de $O(n! \cdot n)$ (ver Figura 12), onde n representa o tamanho da mensagem. Isso ocorre porque:

1. *O algoritmo gera todas as n! permutações possíveis da mensagem. A permutação de n elementos implica um crescimento exponencial, já que o número de combinações possíveis aumenta drasticamente conforme o tamanho da mensagem cresce.*
2. *Além disso, para cada permutação gerada, o algoritmo realiza uma operação adicional para verificar ou manipular cada um dos n caracteres da mensagem, resultando em um custo adicional linear, multiplicando o esforço em n.*

Figura 12. Gráfico de complexidade Big-O (7).



Portanto, à medida que o tamanho da mensagem aumenta, a quantidade de combinações e rotações cresce de maneira significativa, tornando o uso de força bruta inviável para mensagens longas. Esse tipo de análise evidencia a importância de projetar algoritmos criptográficos robustos, que dificultem significativamente ataques desse tipo.

APÊNDICE

A. CÓDIGOS COMPLETOS

A.1. criptografar.py

```
import string

def gerar_chave_aleatoria(tamanho, nome_arquivo):
    chave = list(range(tamanho))
    random.shuffle(chave)
    chave_string = ' '.join(map(str, chave))

    with open(nome_arquivo, 'w') as arquivo:
        arquivo.write(chave_string)
    return chave

def ler_arquivo_para_string(nome_arquivo):
    conteudo_completo = ""
    try:
        with open(nome_arquivo, 'r') as arquivo:
            for linha in arquivo:
                conteudo_completo += linha.strip() + " "
    except FileNotFoundError:
        print(f"Arquivo '{nome_arquivo}' não encontrado.")
    return conteudo_completo.strip()

def criptografar(chave, conteudo):
    conteudo_criptografado = ''
    conteudo_reordenado = ''
    for indice, letra in enumerate(conteudo):
        if letra == ' ':
            conteudo_criptografado = conteudo_criptografado + '#'
        else:
            conteudo_criptografado = conteudo_criptografado + rodar_alfabeto(chave[0], letra)

    conteudo_reordenado = conteudo_criptografado
    conteudo_reordenado_list = list(conteudo_reordenado)
    for indice, numero in enumerate(chave):
        conteudo_reordenado_list[indice] = conteudo_criptografado[numero]
    conteudo_reordenado = ''.join(conteudo_reordenado_list)

    with open('mensagem_criptografada.txt', 'w') as arquivo:
        arquivo.write(conteudo_reordenado)
    return chave
```

```
    return conteudo_reordenado

def rodar_alfabeto(indice, letra):
    alfabeto = string.ascii_lowercase
    tamanho_alfabeto = len(alfabeto)

    if letra not in alfabeto:
        print(letra)
        raise ValueError("A letra fornecida não está no alfabeto.")

    indice_inicial = alfabeto.index(letra)

    indice_rodado = (indice_inicial + indice) % tamanho_alfabeto

    return alfabeto[indice_rodado]

arquivo_mensagem = 'mensagem.txt'
arquivo_chave = 'chave.txt'

conteudo = ler_arquivo_para_string(arquivo_mensagem)

tamanho_string = len(conteudo)

chave_aleatoria = gerar_chave_aleatoria(tamanho_string, arquivo_chave)

conteudo_criptografado = criptografar(chave_aleatoria, conteudo)
```

A.2. descriptografar.py

```

def ler_arquivo(nome_arquivo):
    conteudo_completo = ""
    try:
        with open(nome_arquivo, 'r') as arquivo:
            for linha in arquivo:
                conteudo_completo += linha.strip() + " "
    except FileNotFoundError:
        print(f"Arquivo '{nome_arquivo}' não encontrado.")
    return conteudo_completo.strip()

def descriptografar(chave, mensagem_criptografada):
    chave_lista = list(map(int, chave.split()))
    indices = list(map(int, chave.replace(' ','')))
    reordenada = ''

    for i, index in enumerate(chave_lista):

        reordenada = reordenada + mensagem_criptografada[chave_lista.index(i)]


    conteudo_descriptografado = ''

    for indice, letra in enumerate(reordenada):
        if letra == '#':

            conteudo_descriptografado = conteudo_descriptografado + ' '
        else:

            conteudo_descriptografado = conteudo_descriptografado + rodar_alfabeto(int(chave_lista[0]), letra)

    with open('mensagem_descriptografada.txt', 'w') as arquivo:
        arquivo.write(conteudo_descriptografado)

def rodar_alfabeto(indice, letra):
    alfabeto = string.ascii_lowercase
    tamanho_alfabeto = len(alfabeto)

    if letra not in alfabeto:
        raise ValueError("A letra fornecida não está no alfabeto.")

    indice_inicial = alfabeto.index(letra)
```

```
    indice_rodado = (indice_inicial - indice) % tamanho_alfabeto

    return alfabeto[indice_rodado]

arquivo_mensagem_criptografada = 'mensagem_criptografada.txt'
arquivo_chave = 'chave.txt'

chave = ler_arquivo(arquivo_chave)

mensagem_criptografada = ler_arquivo(arquivo_mensagem_criptografada)

descriptografar(chave, mensagem_criptografada)
```

A.3. forcaBruta.py

```
import string

def ler_arquivo(nome_arquivo):
    conteudo_completo = ""
    try:
        with open(nome_arquivo, 'r') as arquivo:
            for linha in arquivo:
                conteudo_completo += linha.strip() + " "
    except FileNotFoundError:
        print(f"Arquivo '{nome_arquivo}' não encontrado.")
    return conteudo_completo.strip()

def forcaBruta(mensagem):

    if len(mensagem) == 1:
        return [mensagem]

    combinacoes = []

    for i in range(len(mensagem)):

        char_atual = mensagem[i]

        resto = mensagem[:i] + mensagem[i+1:]

        for p in forcaBruta(resto):
            combinacoes.append(char_atual + p)

    return combinacoes

def rodar_letra(letra, numero):
    if letra == '#':
        return ' '

    alfabeto = string.ascii_lowercase
    tamanho_alfabeto = len(alfabeto)


    if letra not in alfabeto:
        return letra

    indice_inicial = alfabeto.index(letra)

    indice_novo = (indice_inicial - numero) % tamanho_alfabeto

    return alfabeto[indice_novo]

def aplicar_todas_as_rotacoes(vetor_combinacoes):
    resultado_final = []

    for combinacao in vetor_combinacoes:
        tamanho = len(combinacao)
        rotacoes_combinacao = []

        for n in range(0, tamanho):
            nova_combinacao = ''.join(rodar_letra(letra, n) for letra in combinacao)
            rotacoes_combinacao.append(nova_combinacao)

        resultado_final.append(rotacoes_combinacao)

    return resultado_final

arquivo_mensagem_criptografada = 'mensagem_criptografada.txt'
mensagem_criptografada = ler_arquivo(arquivo_mensagem_criptografada)

```

```
possibilidades = forcaBruta(
    mensagem_criptografada)
resultado =
    aplicar_todas_as_rotacoes(
    possibilidades)

with open("combinacoes.txt", "w")
    as file:

     for index, combinacao in
    enumerate(resultado):
          for indice, conteudo in
    enumerate(combinacao):
               file.write(f" - {
    combinacao[indice]}\n")
```

REFERÊNCIAS

Absoluta. Cripty Básico: Transposição. Disponível em: <https://www.absoluta.org/cripty/cripty_basico.htm#:~:text=Transposi%C3%A7%C3%A3o%20%2D%20Cifra%20de%20transposi%C3%A7%C3%A3o%20(algumas,letra%20%C3%A9%20trocada%20por%20outra>.

IME-USP. Cifras de Substituição Simples. Disponível em: <https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-substitui%C3%A7%C3%A3o-simples>.

IME-USP. Cifras de Transposição. Disponível em: <https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o>.

Python. Python Official Website. Disponível em: <https://www.python.org/>.

Visual Studio Code. Visual Studio Code Official Website. Disponível em: <https://code.visualstudio.com/>.

Ganeshicmc. Guia de Python para Criptografia. Disponível em: <https://gitbook.ganeshicmc.com/criptografia/guia-de-python-para-criptografia>.

FreeCodeCamp. O que é a notação Big O: Complexidade de tempo e de espaço. Disponível em: <https://www.freecodecamp.org/portuguese/news/o-que-e-a-notacao-big-o-complexidade-de-tempo-e-de-espaco/>.