Relatório do Software Anti-plágio CopySpider

Para mais detalhes sobre o CopySpider, acesse:https://copyspider.com.br

Instruções

Este relatório apresenta na próxima página uma tabela na qual cada linha associa o conteúdo do arquivo de entrada com um documento encontrado na internet (para "Busca em arquivos da internet") ou do arquivo de entrada com outro arquivo em seu computador (para "Pesquisa em arquivos locais"). A quantidade de termos comuns representa um fator utilizado no cálculo de similaridade dos arquivos sendo comparados. Quanto maior a quantidade de termos comuns, combinada com o agrupamento desses termos, maior a similaridade entre os arquivos. É importante destacar que a classificação da semelhança como Alta, Moderada e Baixa não representa um "índice de plágio". Por exemplo, documentos que citam de forma direta (transcrição) outros documentos, podem ter uma similaridade Alta e ainda assim não podem ser caracterizados como plágio. Há sempre a necessidade do avaliador fazer uma análise para decidir se as semelhanças encontradas caracterizam ou não o problema de plágio ou mesmo de erro de formatação ou adequação às normas de referências bibliográficas. Para cada par de arquivos, apresenta-se uma comparação dos termos semelhantes, os quais aparecem em vermelho.
Veja também:
Analisando o resultado do CopySpider
Qual o percentual aceitável para ser considerado plágio?

Versão do CopySpider: 3.1.6
Relatório gerado por: luizsouza1411@outlook.com
Análise no modo: Web/Normal (97.5%) em 18:32
Idioma da busca: Português

Arquivos	Termos comuns	Semelhança	Agrupamento
Relatório APS 1º Semestre.docx X teses.usp.br/teses/disponiveis/3/3141/tde-30092008-182545/publico/DissertacaoRevisada_Marcos_Simplicio.pdf	99	Baixa	Baixo
Relatório APS 1º Semestre.docx X www.cos.ufrj.br/uploadfile/1364224388.pdf	94	Baixa	Baixo
Relatório APS 1º Semestre.docx X repositorio.ufsc.br/xmlui/bitstream/handle/123456789/85272/227681.pdf?sequence=1&isAllowed=y	92	Baixa	Baixo
Relatório APS 1º Semestre.docx X www.passeidireto.com/arquivo/143044716/sistemas-operacionais-funcoes-e-seguranca	91	Baixa	Baixo
Relatório APS 1º Semestre.docx X wiki.stoa.usp.br/images/c/cf/Stallings-cap2e3.pdf	78	Baixa	Baixo
Relatório APS 1º Semestre.docx X www.passeidireto.com/arquivo/76648298/aps-cifra-de-vigenere	77	Baixa	Baixo
Relatório APS 1º Semestre.docx X www.passeidireto.com/arquivo/85388953/aps-criptografia-vigenere	77	Baixa	Baixo
Relatório APS 1º Semestre.docx X www.estrategiaconcursos.com.br/curso/download/?aula=3337168	74	Baixa	Baixo
Relatório APS 1º Semestre.docx X www.egape.pe.gov.br/images/media/1665420043_Apostila Introducao Seguranca Informacao Corporativa.pdf	73	Baixa	Baixo
Relatório APS 1º Semestre.docx X www.passeidireto.com/arquivo/160778022/09-tanembaum-redes-de-computadores-4-ed-pt-br	60	Baixa	Baixo

Arquivos com problema de download

https://github.com/TomSchimansky/CustomTkinter/wiki/CTkButton - Não foi possível baixar o arquivo. É recomendável baixar o arquivo manualmente e realizar a análise em conluio (Um contra todos). - Tipo do arquivo não identificado

https://pt.stackoverflow.com/questions/43492/como-funciona-o-algoritmo-de-criptografia-aes - Não foi possível baixar o arquivo. É recomendável baixar o arquivo manualmente e realizar a análise em conluio (Um contra todos). - (22) The requested URL returned error: 403

https://pt.stackoverflow.com/questions/383328/elabora%C3%A7%C3%A3o-de-cifra-de-c%C3%A9sar-em-python - Não foi possível baixar o arquivo. É recomendável baixar o arquivo manualmente e realizar a análise em conluio (Um contra todos). - (22) The requested URL returned error: 403

https://comum.rcaap.pt/bitstreams/57e97369-0dd8-459c-b1df-b870a97dd43a/download - Não foi possível baixar o arquivo. É recomendável baixar o arquivo manualmente e realizar a análise em conluio (Um contra todos). - Tipo de arquivo não suportado: application/json

https://github.com/VILHALVA/CURSO-DE-CUSTOMTKINTER/blob/main/SINTAXE.md - Não foi possível baixar o arquivo. É recomendável baixar o arquivo manualmente e realizar a análise em conluio (Um contra todos). - Tipo do arquivo não identificado

https://pt.stackoverflow.com/questions/471352/critografia-em-python - Não foi possível baixar o arquivo. É recomendável baixar o arquivo manualmente e realizar a análise em conluio (Um contra todos). - (22) The requested URL returned error: 403

https://pt.stackoverflow.com/questions/480647/executar-evento-em-bot%C3%A3o-tkinter - Não foi possível baixar o arquivo. É recomendável baixar o arquivo manualmente e realizar a análise em conluio (Um contra todos). - (22) The requested URL returned error: 403

Arquivos com problema de conversão

https://cifradevigenere.vercel.app - Não foi possível converter o arquivo. É recomendável converter o arquivo para texto manualmente e realizar a análise em conluio (Um contra todos).

https://www.ibm.com/docs/pt-br/cobol-zos/6.4.0?topic=functions-converting-hexadecimal-bit-data-hex-bit - Não foi possível converter o arquivo. É recomendável converter o arquivo para texto manualmente e realizar a análise em conluio (Um contra todos).

https://estacio.br/cursos/graduacao/engenharia-da-computacao - Não foi possível converter o arquivo. É recomendável converter o arquivo para texto manualmente e realizar a análise em conluio (Um contra todos).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)

Arquivo 2: teses.usp.br/teses/disponiveis/3/3141/tde-30092008-182545/publico/DissertacaoRevisada_Marcos_Simplicio.pdf (33513 termos)

Termos comuns: 99

Índice de similaridade antigo: 0,27%

Novo índice de similaridade: 4,51%

Índice de agrupamento: Baixo

O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: 8a80c8d51493147x10

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Desenvolvimento de aplicação utilizando técnicas criptográficas

CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos

2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO

A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.

Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.

A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.

Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.

Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial.

Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave

repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo

da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de segurança iria precisar de recursos mais avançados.
Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA
A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.
O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.
O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.
O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para

bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',

```
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
```

```
            resultado += base[novo_indice]
        else:
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
```

```
resultado = ''
for char_texto, char_chave in zip(texto, chave_expandida):
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)
```

```
return etapa6
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
```

```
            print("Saindo...")
            break

        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
```

```
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)
```

```
campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: www.cos.ufrj.br/uploadfile/1364224388.pdf (20464 termos)
Termos comuns: 94
Índice de similaridade antigo: 0,41%
Novo índice de similaridade: 4,28%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: cf5ac851130b9acx12

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO
A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.
Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.
A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.
Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.
Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da

computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do

texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de segurança iria precisar de recursos mais avançados.

Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA

A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.

O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.

O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.

O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de

caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
```

```
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
        else:
```

```
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
for char_texto, char_chave in zip(texto, chave_expandida):
```

```
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)

return etapa6
```

```
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
            break
```

```
        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
```

```
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)

campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
```

```
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2:
repositorio.ufsc.br/xmlui/bitstream/handle/123456789/85272/227681.pdf?sequence=1&isAllowed=y
(26935 termos)
Termos comuns: 92
Índice de similaridade antigo: 0,31%
Novo índice de similaridade: 4,19%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: a5850c44701f1fbx16

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO

A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.

Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.

A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.

Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.

Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial.

Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave

repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo

da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de segurança iria precisar de recursos mais avançados.
Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA
A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.
O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.
O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.
O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para

bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
```

```
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
```

```
            resultado += base[novo_indice]
        else:
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
```

```
resultado = ''
for char_texto, char_chave in zip(texto, chave_expandida):
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)
```

```
return etapa6
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
```

```
            print("Saindo...")
            break

        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
```

```
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)
```

```
campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: www.passeidireto.com/arquivo/143044716/sistemas-operacionais-funcoes-e-seguranca (39014 termos)
Termos comuns: 91
Índice de similaridade antigo: 0,22%
Novo índice de similaridade: 4,14%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: f628c1c4180c809x9

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO

A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.

Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.

A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.

Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.

Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço

liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento

determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de

segurança iria precisar de recursos mais avançados.
Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA
A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.
O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.
O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.
O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de

Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
```

```
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
```

```
        else:
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
```

```
for char_texto, char_chave in zip(texto, chave_expandida):
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)
```

```
return etapa6
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
```

```
            break

        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
```

```
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)
```

```
campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: wiki.stoa.usp.br/images/c/cf/Stallings-cap2e3.pdf (17182 termos)
Termos comuns: 78
Índice de similaridade antigo: 0,40%
Novo índice de similaridade: 3,55%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: 2d0be97b0b1fab9x10

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO
A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.
Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.
A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.
Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.
Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da

computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do

texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de segurança iria precisar de recursos mais avançados.

Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA

A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.

O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.

O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.

O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de

caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
```

```
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
        else:
```

```
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
for char_texto, char_chave in zip(texto, chave_expandida):
```

```
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)

return etapa6
```

```
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
            break
```

```
        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
```

```
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)

campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
```

```
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: www.passeidireto.com/arquivo/76648298/aps-cifra-de-vigenere (6136 termos)
Termos comuns: 77
Índice de similaridade antigo: 0,93%
Novo índice de similaridade: 3,51%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: cd05c08bc86bc54x10

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO
A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.
Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.
A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.
Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.
Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da

computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do

texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de segurança iria precisar de recursos mais avançados.

Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA

A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.

O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.

O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.

O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de

caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
```

```
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
        else:
```

```
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
for char_texto, char_chave in zip(texto, chave_expandida):
```

```
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)

return etapa6
```

```
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
            break
```

```
        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
```

```
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)

campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
```

```
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: www.passeidireto.com/arquivo/85388953/aps-criptografia-vigenere (6099 termos)
Termos comuns: 77
Índice de similaridade antigo: 0,93%
Novo índice de similaridade: 3,51%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: 8306dce139f2869x10

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO
A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.
Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.
A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.
Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.
Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da

computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do

texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de segurança iria precisar de recursos mais avançados.

Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA

A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.

O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.

O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.

O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de

caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
```

```
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
        else:
```

```
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
for char_texto, char_chave in zip(texto, chave_expandida):
```

```
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)

return etapa6
```

```
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
            break
```

```
        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
```

```
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)

campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
```

```
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: www.estrategiaconcursos.com.br/curso/download/?aula=3337168 (14778 termos)
Termos comuns: 74
Índice de similaridade antigo: 0,43%
Novo índice de similaridade: 3,37%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: 7f9a5e107b9042fx14

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO

A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.

Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.

A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.

Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.

Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da

computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do

texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de segurança iria precisar de recursos mais avançados.

Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA

A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.

O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.

O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.

O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de

caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
```

'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)

from caracteres import BASE_CARACTERES

```
def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
        else:
```

```
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
for char_texto, char_chave in zip(texto, chave_expandida):
```

```
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)

return etapa6
```

```
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
            break
```

```
        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
```

```
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)

campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
```

```
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: www.egape.pe.gov.br/images/media/1665420043_Apostila Introducao Seguranca Informacao Corporativa.pdf (28561 termos)
Termos comuns: 73
Índice de similaridade antigo: 0,23%
Novo índice de similaridade: 3,32%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: c3a71958a07786cx12

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO
A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.
Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.
A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.
Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.
Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço

liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento

determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de

segurança iria precisar de recursos mais avançados.
Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA
A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.
O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.
O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.
O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de

Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
```

```
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
```

```
        else:
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
```

```
for char_texto, char_chave in zip(texto, chave_expandida):
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)
```

```
return etapa6
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
```

```
            break

        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
```

```
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)
```

```
campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).

==========

Arquivo 1: Relatório APS 1º Semestre.docx (2193 termos)
Arquivo 2: www.passeidireto.com/arquivo/160778022/09-tanembaum-redes-de-computadores-4-ed-pt-br (44923 termos)
Termos comuns: 60
Índice de similaridade antigo: 0,12%
Novo índice de similaridade: 2,73%
Índice de agrupamento: Baixo
O texto abaixo é o conteúdo do documento **Arquivo 1**. Os termos em vermelho foram encontrados no documento **Arquivo 2**. Id da comparação: c08476ef63122b4x7

==========

FACULDADE VANGUARDA

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

São José dos Campos
2025

LUIZ GUSTAVO FRANCISCO DE SOUZA

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS
Desenvolvimento de aplicação utilizando técnicas criptográficas
CIFRA DE VIGENÈRE COM TRANSPOSIÇÃO E PERMUTAÇÃO BIJETORA DE BYTES

Atividades Práticas Supervisionadas do curso de
ENGENHARIA DA COMPUTAÇÃO
da FACULDADE VANGUARDA, sob orientação de:
Profa. Dra. Ivana Yoshie Sumida ? Prof. Responsável:
Prof. MSc. André Yoshimi Kusumoto ? Coordenador:

São José dos Campos
2025

INTRODUÇÃO
A Atividade Prática Supervisionada (APS) tem como objetivo colocar em prática os conhecimentos adquiridos na disciplina de Algoritmos e Programação Estruturada, ampliando essa base por meio de pesquisas e da construção de uma aplicação criptográfica desenvolvida em Python. A proposta visa introduzir o aluno aos diversos contextos envolvidos na elaboração de um sistema computacional funcional.
Este trabalho busca proporcionar não apenas a aplicação dos conteúdos teóricos aprendidos em sala de aula, mas também o desenvolvimento de habilidades práticas fundamentais para a formação em Engenharia da Computação, como lógica de programação, estruturação de código, modularização e solução de problemas.
A escolha do tema criptografia está diretamente relacionada à sua crescente importância no cenário tecnológico atual, onde a proteção de dados e a privacidade da informação se tornaram prioridades. O desafio de implementar um algoritmo autoral, sem recorrer a bibliotecas prontas, permite um mergulho mais profundo na compreensão dos mecanismos criptográficos, contribuindo significativamente para a formação técnica do aluno.
Este relatório apresenta desde os fundamentos teóricos da criptografia até a descrição da técnica escolhida, finalizando com a solução prática implementada em Python.
Uma experiência pessoal que reforçou o interesse pelo tema foi o filme O Jogo da Imitação (The Imitation Game, 2014), que retrata a quebra da Máquina Enigma durante a Segunda Guerra Mundial. Essa máquina era utilizada pelos alemães para proteger suas comunicações militares, e o esforço

liderado por Alan Turing e sua equipe para decifrar seus códigos é um marco histórico na ciência da computação e na criptografia. Esse episódio ilustra a relevância da área e reforçou minha motivação para estudar e desenvolver técnicas criptográficas de forma prática.

FUNDAMENTOS DE CRIPTOGRAFIA

A criptografia é uma área da ciência da computação e da matemática aplicada que estuda técnicas para proteger a informação contra acessos não autorizados. Seu principal objetivo é garantir a confidencialidade, integridade, autenticidade e, em alguns casos, a irredutibilidade dos dados durante a comunicação ou armazenamento.

Como define a própria Google Cloud, ?No nível mais básico, a criptografia é o processo de proteger informações ou dados usando modelos matemáticos para embaralhá-los de modo que apenas as partes que têm a chave para decifrar possam acessá-lo.? (GOOGLE CLOUD, 2025).

Existem dois principais tipos de criptografia: simétrica e assimétrica. Na criptografia simétrica, o mesmo segredo (ou chave) é usado tanto para cifrar quanto para decifrar a informação. É um modelo eficiente em termos de desempenho, mas que exige um método seguro de compartilhamento da chave entre as partes. Algoritmos clássicos desse tipo incluem o AES (Advanced Encryption Standard) e o DES (Data Encryption Standard).

Já na criptografia assimétrica, são utilizadas duas chaves distintas: uma pública, que pode ser divulgada, e uma privada, que deve ser mantida em sigilo. O que é cifrado com uma chave só pode ser decifrado com a outra. Esse modelo é amplamente utilizado em sistemas de assinatura digital, autenticação e troca segura de chaves. Exemplos conhecidos incluem o RSA, o ECC (Criptografia de Curvas Elípticas) e o algoritmo ElGamal.

Figura 2 ? Ilustração dos modelos de criptografia simétrica e assimétrica

Fonte: ResearchGate. Disponível em: https://www.researchgate.net/figure/Figura-14-Modelo-simetrico-e-assimetrico-de-criptografia_fig4_266912212. Acesso em: 30 maio 2025.

Além das técnicas de cifra, a criptografia moderna envolve outros conceitos essenciais, como a função hash, usada para garantir integridade dos dados, e os vetores de inicialização (IVs), que garantem aleatoriedade mesmo quando a mesma chave é usada em múltiplas operações.

A segurança criptográfica não depende apenas do sigilo do algoritmo, mas da robustez matemática contra ataques, como força bruta, análise estatística ou ataques de criptoanálise. Por isso, é comum utilizar algoritmos públicos, amplamente testados pela comunidade científica, com segredos limitados apenas às chaves utilizadas.

TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica adotada neste projeto combina três princípios fundamentais: a cifra de Vigenère, a transposição de blocos fixos e uma permutação matemática bijetora aplicada diretamente sobre os bytes da mensagem. Essa composição resulta em um algoritmo simétrico, reversível e eficiente e prático.

A cifra de Vigenère, um método clássico de substituição polialfabética, baseia-se no uso de uma chave repetida ao longo do texto original. Para cada caractere da mensagem, é aplicado um deslocamento

determinado pela posição correspondente da chave, resultando em diferentes substituições ao longo do texto. Tradicionalmente, esse processo é ilustrado por meio da tabela de Vigenère, também conhecida como quadrado de Vigenère, composta por diversas linhas com o alfabeto deslocado progressivamente. A cada letra do texto simples, utiliza-se uma linha diferente da tabela, conforme a letra da chave.

Figura 2 ? Tabela de Vigenère para cifragem de texto

Fonte: Adaptado de GEEKSFORGEEKS. Vigenère Cipher. Disponível em: https://www.geeksforgeeks.org/vigenere-cipher/. Acesso em: 30 maio 2025.

Posteriormente o texto é submetido a uma transposição de blocos: ele é dividido em grupos de quatro caracteres, e cada bloco é invertido. Essa etapa reorganiza a posição dos elementos da mensagem, embaralhando sua estrutura e contribuindo para a não linearidade do processo.

A última etapa do algoritmo é uma permutação bijetora aplicada sobre os bytes da mensagem já cifrada. Cada byte é transformado pela fórmula:

f(b) = (3 * b + 7) mod 256

Essa função embaralha o conteúdo de forma determinística e reversível, utilizando uma operação aritmética modular inspirada em transformações não lineares típicas de cifradores modernos. Ao final, o resultado é convertido em hexadecimal, o que facilita a representação e o armazenamento da mensagem criptografada.

A descriptografia reverte exatamente cada uma das etapas na ordem inversa. A mensagem em hexadecimal é convertida em bytes, submetida à função inversa da permutação, depois decodificada e decifrada utilizando as mesmas chaves derivadas, desfazendo a transposição de blocos e a primeira cifra de Vigenère. Ao final, o texto original é restaurado com precisão, desde que a chave correta seja utilizada.

Entre as principais vantagens desta técnica estão a combinação de múltiplos mecanismos (substituição, transposição e permutação), que foi organizado em etapas:

Vigenère: O texto é criptografado com a cifra de Vigenère usando uma chave inicial.

Transposição: Os caracteres são reorganizados para embaralhar o conteúdo.

Vigenère com chave modificada: A cifra de Vigenère é aplicada novamente com uma chave derivada (_2).

Nova aplicação de Vigenère: Uma terceira cifra é feita com outra chave modificada (_3).

Conversão para bytes: O texto é transformado em bytes UTF-8.

Permutação bijetora: Os bytes são embaralhados de forma reversível.

Hexadecimal: O resultado é convertido para uma string hexadecimal.

A robustez do embaralhamento por camadas, a utilização de uma base de caracteres expandida (incluindo acentuação e símbolos especiais) e a clareza da estrutura, que permite adaptação e expansão conforme a necessidade do projeto. O uso de chaves derivadas também adiciona variabilidade e aumenta a resistência contra ataques de repetição.

Entre as limitações, destaca-se a ausência de mecanismos adicionais como autenticação ou verificação de integridade (MAC, hash), e o fato de a segurança do sistema depender diretamente da força e sigilo da chave utilizada pelo usuário. Por se tratar de um algoritmo autoral, seu uso em contextos críticos de

segurança iria precisar de recursos mais avançados.
Esta técnica pode ser aplicada em situações que exigem proteção leve a moderada de dados, como arquivos locais, mensagens confidenciais, proteção de configurações e informações sensíveis em sistemas que não requerem criptografia padronizada por grandes instituições.

SOLUÇÃO DESENVOLVIDA
A aplicação desenvolvida foi implementada na linguagem Python e estruturada em um modelo funcional, com duas interfaces: uma por linha de comando (terminal) e outra com interface gráfica, utilizando a biblioteca CustomTkinter. O projeto foi dividido em cinco arquivos principais: programa.py, criptografia.py, descriptografia.py, caracteres.py e interface.py, cada um com responsabilidades distintas dentro do processo criptográfico.
O arquivo programa.py contém o menu textual da aplicação, onde o usuário pode escolher entre criptografar, descriptografar ou encerrar o sistema. A interação ocorre via terminal, com entrada de texto digitada pelo usuário. Já o arquivo interface.py oferece uma versão visual da aplicação, com campos de entrada para o texto, chave e opção de escolha (criptografar, descriptografar ou sair), além de botões funcionais para executar e limpar os dados. A interface foi desenvolvida com a biblioteca CustomTkinter, que possibilita um design moderno.
O arquivo criptografia.py implementa o processo de cifragem com múltiplas camadas, organizadas da seguinte forma: cifra de Vigenère com a chave original, transposição de blocos com inversão, duas cifras de Vigenère com chaves modificadas e, por fim, aplicação de uma permutação matemática bijetora sobre os bytes da mensagem. O resultado é convertido em hexadecimal para facilitar a visualização e o armazenamento da mensagem cifrada.
O arquivo descriptografia.py realiza o processo reverso. Ele converte o texto hexadecimal de volta para bytes, aplica a função matemática inversa, decodifica os dados em UTF-8 e desfaz cada etapa de

Vigenère e da transposição, até recuperar o texto original. Ambas as operações utilizam uma base de caracteres personalizada definida no arquivo caracteres.py, que inclui letras com acentuação, símbolos e outros caracteres especiais.
A lógica de cifragem implementada segue as etapas descritas anteriormente na seção 3, com aplicação em camadas da cifra de Vigenère, transposição de blocos e permutação bijetora. Essas etapas foram convertidas em funções específicas dentro do código, cada uma com responsabilidade clara e sequencial.
A aplicação funciona de maneira robusta, permitindo que o usuário insira textos de até 1000 caracteres e uma chave personalizada. A saída da criptografia é apresentada como uma string hexadecimal, e a descriptografia é capaz de recuperar o texto original integralmente, desde que a mesma chave seja utilizada.

A seguir, estão imagens ilustrando a execução do programa nas duas versões:

Figura 3 ? Tela de menu principal no terminal

Figura 4 ? Entrada de texto e chave para criptografia no terminal

Figura 5 ? Interface gráfica da aplicação

Figura 6 ? Resultado da criptografia na interface gráfica

Figura 5 ? Processo de descriptografia na interface gráfica

CÓDIGO-FONTE

```
digitos = list('0123456789')

letras_maiusculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')

letras_minusculas = list('abcdefghijklmnopqrstuvwxyz')

pontuacoes = list('!"#$%&\'()*+,-./:;&lt;=&gt;?@[\\]^_`{|}~')

espacos = [' ']

letras_acentuadas = [
'á','à','â','ã','ä','é','è','ê','ë','í','ì','î','ï',
'ó','ò','ô','õ','ö','ú','ù','û','ü','ç',
```

```
'Á','À','Â','Ã','Ä','É','È','Ê','Ë','Í','Ì','Î','Ï',
'Ó','Ò','Ô','Õ','Ö','Ú','Ù','Û','Ü','Ç','ñ','Ñ'
]

simbolos_extra = ['?','£','¥','¢','§','©','®','?','°','±','×','÷','µ','¶']

BASE_CARACTERES = (
digitos +
letras_maiusculas +
letras_minusculas +
pontuacoes +
espacos +
letras_acentuadas +
simbolos_extra
)
```

```
from caracteres import BASE_CARACTERES

def cifra_vigenere(texto, chave, base):
    chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
    resultado = ''
    for char_texto, char_chave in zip(texto, chave_expandida):
        if char_texto in base and char_chave in base:
            indice_texto = base.index(char_texto)
            indice_chave = base.index(char_chave)
            novo_indice = (indice_texto + indice_chave) % len(base)
            resultado += base[novo_indice]
```

```
        else:
            resultado += char_texto
    return resultado

def transposicao_bloco(texto, tamanho_bloco=4):
    resultado = ''
    for i in range(0, len(texto), tamanho_bloco):
        bloco = texto[i:i+tamanho_bloco]

        resultado += bloco[::-1]
    return resultado

def permutacao_bijetora(bytes_seq):
    return bytes(( (b * 3 + 7) % 256 for b in bytes_seq ))
def modificar_chave(chave, modificador):
    return chave + modificador

def criptografar(texto, chave):
    etapa1 = vigenere(texto, chave, BASE_CARACTERES)
    etapa2 = transposicao_bloco(etapa1)
    chave2 = modificar_chave(chave, '_2')
    etapa3 = vigenere(etapa2, chave2, BASE_CARACTERES)
    chave3 = modificar_chave(chave, '_3')
    etapa4 = vigenere(etapa3, chave3, BASE_CARACTERES)
    etapa4_bytes = etapa4.encode('utf-8')
    etapa5_bytes = permutacao_bijetora(etapa4_bytes)
    texto_encriptado = etapa5_bytes.hex()
    return texto_encriptado
```

```
from caracteres import BASE_CARACTERES

def vigenere_decifrar(texto, chave, base):
chave_expandida = (chave * ((len(texto) // len(chave)) + 1))[:len(texto)]
resultado = ''
```

```
for char_texto, char_chave in zip(texto, chave_expandida):
if char_texto in base and char_chave in base:
indice_texto = base.index(char_texto)
indice_chave = base.index(char_chave)
novo_indice = (indice_texto - indice_chave) % len(base)
resultado += base[novo_indice]
else:
resultado += char_texto
return resultado

def permutacao_bijetora_inversa(bytes_seq):
return bytes(((171 * (b - 7)) % 256 for b in bytes_seq))

def transposicao_bloco_inversa(texto, tamanho_bloco=4):
resultado = ''
for i in range(0, len(texto), tamanho_bloco):
bloco = texto[i:i+tamanho_bloco]
resultado += bloco[::-1]
return resultado

def modificar_chave(chave, modificador):
return chave + modificador

def descriptografar(texto_hex, chave):

etapa1_bytes = bytes.fromhex(texto_hex)

etapa2_bytes = permutacao_bijetora_inversa(etapa1_bytes)

etapa2_str = etapa2_bytes.decode('utf-8')

chave3 = modificar_chave(chave, '_3')
etapa3 = vigenere_decifrar(etapa2_str, chave3, BASE_CARACTERES)

chave2 = modificar_chave(chave, '_2')
etapa4 = vigenere_decifrar(etapa3, chave2, BASE_CARACTERES)

etapa5 = transposicao_bloco_inversa(etapa4)

etapa6 = vigenere_decifrar(etapa5, chave, BASE_CARACTERES)
```

```
return etapa6
from caracteres import BASE_CARACTERES
from criptografia import criptografar
from descriptografia import descriptografar

def menu( ):
    while True:
        print("\n====== Menu Principal ======")
        print("1. Criptografar")
        print("2. Descriptografar")
        print("3. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            texto = input("Digite o texto (máx. 256 caracteres): ")
            chave = input("Digite a chave: ")
            if len(texto) &gt; 1000:
                print("Erro: texto muito longo.")
                continue
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                criptografado = criptografar(texto, chave)
                print("\nTexto criptografado (hexadecimal):\n" + criptografado)
            except Exception as e:
                print("Erro na criptografia:", e)

        elif opcao == '2':
            texto = input("Digite o texto criptografado (hexadecimal): ")
            chave = input("Digite a chave: ")
            if not chave:
                print("Erro: chave vazia.")
                continue
            try:
                original = descriptografar(texto, chave)
                print("\nTexto descriptografado:\n" + original)
            except Exception as e:
                print("Erro na descriptografia:", e)
        elif opcao == '3':
            print("Saindo...")
```

```
            break

        else:
            print("Opção inválida.")

if __name__ == "__main__":
    menu( )
import customtkinter as ctk
from criptografia import criptografar
from descriptografia import descriptografar

ctk.set_appearance_mode('dark')

app = ctk.CTk()
app.title('Criptografia APS')
app.geometry('500x600')

# Funções de ação
def executar_acao():
    opcao = campo_opcao.get()
    texto = campo_texto.get()
    chave = campo_chave.get()

    resultado.delete("1.0", "end")

    if opcao == '1':
        if len(texto) &gt; 1000:
            resultado.insert("end", "Erro: texto muito longo.")
            return
        if not chave:
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            criptografado = criptografar(texto, chave)
            resultado.insert("end", criptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na criptografia: {e}")

    elif opcao == '2':
        if not chave:
```

```
            resultado.insert("end", "Erro: chave vazia.")
            return

        try:
            descriptografado = descriptografar(texto, chave)
            resultado.insert("end", descriptografado)
        except Exception as e:
            resultado.insert("end", f"Erro na descriptografia: {e}")

    elif opcao == '3':
        app.destroy()
    else:
        resultado.insert("end", "Opção inválida. Escolha 1, 2 ou 3.")

def limpar_campos():
    campo_opcao.delete(0, "end")
    campo_texto.delete(0, "end")
    campo_chave.delete(0, "end")
    resultado.delete("1.0", "end")

def copiar_resultado():
    conteudo = resultado.get("1.0", "end").strip()
    app.clipboard_clear()
    app.clipboard_append(conteudo)

# Interface
label_menu = ctk.CTkLabel(app, text='MENU PRINCIPAL', font=('Arial', 18))
label_menu.pack(pady=10)

label_opcoes = ctk.CTkLabel(app, text='1. Criptografar
2. Descriptografar
3. Sair')
label_opcoes.pack(pady=3)

campo_opcao = ctk.CTkEntry(app, placeholder_text='Escolha uma opção:')
campo_opcao.pack(pady=5)

campo_texto = ctk.CTkEntry(app, placeholder_text='Digite o texto para Criptografar ou Descriptografar:',
width=400)
campo_texto.pack(pady=5)
```

```
campo_chave = ctk.CTkEntry(app, placeholder_text='Digite a chave:', width=400)
campo_chave.pack(pady=5)

botao_executar = ctk.CTkButton(app, text='Executar', command=executar_acao)
botao_executar.pack(pady=10)

botao_limpar = ctk.CTkButton(app, text='Limpar Tudo', command=limpar_campos)
botao_limpar.pack(pady=5)

resultado = ctk.CTkTextbox(app, width=400, height=200)
resultado.pack(pady=10)

botao_copiar = ctk.CTkButton(app, text='Copiar Resultado', command=copiar_resultado)
botao_copiar.pack(pady=5)

app.mainloop()
```

REFERÊNCIAS

MENEZES, Alfred; VAN OORSCHOT, Paul; VANSTONE, Scott. Handbook of Applied Cryptography. CRC Press, 1996.

USP IME. Cifras de Transposição. Disponível em: https://wiki.imesec.ime.usp.br/books/criptografia/page/cifras-de-transposi%C3%A7%C3%A3o. Acesso em: 30 maio 2025.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution...? (GEEKSFORGEEKS, 2025).