IPBeja
INSTITUTO POLITÉCNICO DE BEJA

INSTITUTO POLITÉCNICO DE BEJA
Escola Superior de Tecnologia e Gestão
Mestrado em Engenharia de Segurança Informática
Criptografia e Criptanálise Aplicadas

Desenvolvimento de Aplicação de Cifra Simétrica

Rafael Conceição Narciso - 24473
Hugo Diogo - 18803

IPBeja Escola Superior de Tecnologia e Gestão

Beja, dezembro de 2025

INSTITUTO POLITÉCNICO DE BEJA
Escola Superior de Tecnologia e Gestão
Mestrado em Engenharia de Segurança Informática

Desenvolvimento de Aplicação de Cifra Simétrica

Rafael Conceição Narciso - 24473
Hugo Diogo - 18803

Orientador: Prof. Rui Miguel Silva

Beja, dezembro de 2025

Resumo

Este relatório descreve o desenvolvimento de uma aplicação gráfica para a cifragem e decifragem de ficheiros, implementada na linguagem Python. O projeto visa consolidar conhecimentos de criptografia simétrica através da implementação manual de cifras clássicas (Vigenère e Playfair) e da integração segura de bibliotecas para cifras modernas (AES e DES). São abordadas decisões de arquitetura, modos de operação (CBC) e considerações de segurança.

Palavras-chave: Python, Criptografia Simétrica, AES, DES, Vigenère, Playfair.

Abstract

This report details the development of a graphical application for file encryption and decryption, implemented in Python. The project aims to consolidate symmetric cryptography knowledge through the manual implementation of classical ciphers (Vigenère and Playfair) and the secure integration of libraries for modern ciphers (AES and DES). Architectural decisions, modes of operation (CBC), and security considerations are discussed.

Keywords: Python, Symmetric Cryptography, AES, DES, Vigenère, Playfair.

Índice

Índice de Figuras

1 Introdução

A segurança da informação depende, em grande medida, da robustez dos algoritmos criptográficos utilizados para proteger a confidencialidade dos dados. Este projeto, desenvolvido no âmbito da unidade curricular de Criptografia e Criptanálise Aplicadas, consiste na criação de uma ferramenta em Python capaz de cifrar e decifrar ficheiros utilizando quatro algoritmos distintos: dois clássicos (Vigenère e Playfair) e dois modernos (DES e AES).

A aplicação fornece uma Interface Gráfica (GUI) para facilitar a interação do utilizador, abstraindo a complexidade das operações matemáticas e binárias subjacentes.

2 Enquadramento Teórico

2.1 Cifras Clássicas

«««< HEAD As cifras clássicas implementadas operam ao nível do caráter e pertencem à era pré-computacional. ======= As cifras clássicas implementadas operam ao nível do caracter (byte visível). »»»>

- **Vigenère:** Uma cifra polialfabética que utiliza uma chave e uma tabela (*Tabula Recta*) para substituir caracteres, dificultando a análise de frequências simples.
- **Playfair:** Uma cifra de substituição poligráfica que opera sobre digramas (pares de letras) utilizando uma matriz 5×5.

2.2 Cifras Modernas

As cifras modernas operam sobre blocos de bits e são desenhadas para resistir a ataques computacionais.

- **DES (Data Encryption Standard):** Algoritmo de bloco de 64 bits baseado numa Rede de Feistel. Utiliza uma chave de 56 bits.
- **AES (Advanced Encryption Standard):** Sucessor do DES, operando com blocos de 128 bits e chaves de 128, 192 ou 256 bits, baseado numa rede de substituição-permutação.

3 Desenvolvimento da Aplicação

A aplicação foi desenvolvida em **Python** devido à sua versatilidade na manipulação de *byte streams*. A interface gráfica foi construída com a biblioteca **Tkinter**, seguindo uma arquitetura orientada a eventos.

3.1 Interface Gráfica (GUI)

A GUI permite ao utilizador selecionar o algoritmo, carregar os ficheiros de chave/tabela e indicar os ficheiros de entrada e saída.

««««< HEAD

Figura 1: Interface principal da aplicação desenvolvida.

=======

3.2 Estrutura da Aplicação

A aplicação foi desenvolvida seguindo uma arquitetura modular que separa a interface gráfica da lógica criptográfica. O ficheiro principal, `gui.py`, implementa a interface utilizando a biblioteca **CustomTkinter**, gerindo os eventos do utilizador e invocando as classes específicas para cada cifra (`aes_cipher.py`, `des_cipher.py`, `playfair_cipher.py` e `vigenere_cipher.py`).

3.3 Implementação: Cifras Modernas (AES e DES)

Para as cifras modernas, utilizou-se a biblioteca **PyCryptodome**. Ambas as implementações partilham a mesma arquitetura de segurança:

- **Modo de Operação:** CBC (*Cipher Block Chaining*) para garantir segurança semântica.
- **Padding:** PKCS7 (via função `pad/unpad`) para ajustar os dados ao tamanho do bloco.
- **Gestão de IV:** O Vetor de Inicialização é gerado aleatoriamente e concatenado no início do ficheiro binário.

3.3.1 AES (Advanced Encryption Standard)

A classe `AESCipher` gere automaticamente chaves de 128, 192 ou 256 bits. O tamanho do bloco é fixo em 128 bits (16 bytes), o que se reflete na extração do IV durante a decifragem (`data[:16]`).

```
def encrypt_file(self, data):
    cipher = AES.new(self.key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(data, AES.block_size))
    # Concatena IV (16 bytes) + Criptograma
    return cipher.iv + ct_bytes

def decrypt_file(self, data):
    # Recupera o IV (primeiros 16 bytes)
    iv = data[:16]
    ct = data[16:]
    cipher = AES.new(self.key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(ct), AES.block_size)
```

Listing 1: AES: Manipulação de Ficheiros (Bloco de 16 bytes)

3.3.2 DES (Data Encryption Standard)

A classe `DESCipher` implementa a estrutura clássica de blocos de 64 bits (8 bytes). O código valida estritamente o tamanho da chave (8 bytes) e ajusta a extração do IV para ler apenas os primeiros 8 bytes (`data[:8]`).

```
def encrypt_file(self, data):
    cipher = DES.new(self.key, DES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(data, DES.block_size))
    # Concatena IV (8 bytes) + Criptograma
    return cipher.iv + ct_bytes

def decrypt_file(self, data):
    # Recupera o IV (primeiros 8 bytes)
    iv = data[:8]
    ct = data[8:]
    cipher = DES.new(self.key, DES.MODE_CBC, iv)
    return unpad(cipher.decrypt(ct), DES.block_size)
```

Listing 2: DES: Manipulação de Ficheiros (Bloco de 8 bytes)

3.4 Implementação: Vigenère

A cifra de Vigenère foi implementada de raiz, suportando tanto a geração automática de uma *Tabula Recta* padrão (26×26) como o carregamento de tabelas personalizadas a partir de ficheiro.

A implementação distingue-se pela assimetria algorítmica entre as operações de cifragem e decifragem. Enquanto a cifragem acede diretamente às coordenadas da matriz ($O(1)$), a decifragem necessita de percorrer a linha correspondente à chave ($O(N)$) para encontrar a coluna original. Esta abordagem foi necessária para suportar tabelas com alfabetos desordenados.

```
for i, char in enumerate(plaintext):
    if char.isalpha():
        # Use table for encryption
        row = ord(key[i]) - ord('A')
        col = ord(char) - ord('A')
        encrypted_char = self.table[row][col]
        ciphertext += encrypted_char
    else:
        ciphertext += char
```

Listing 3: Cifragem: Acesso direto à matriz

```
for i, char in enumerate(ciphertext):
    if char.isalpha():
        # Use table for decryption - find column in row
        row = ord(key[i]) - ord('A')
        # Find which column gives us the ciphertext character
        for col in range(26):
            if self.table[row][col] == char:
                plaintext += chr(col + ord('A'))
                break
    else:
        plaintext += char
```

Listing 4: Decifragem: Pesquisa linear na linha

3.5 Implementação: Playfair

A cifra Playfair envolveu a criação dinâmica da matriz 5×5, onde a letra 'J' é fundida com o 'I' para se ajustar à grelha de 25 caracteres. Antes do processamento criptográfico, foi implementado um método auxiliar `_prepare_text` que normaliza a entrada: converte para maiúsculas, remove caracteres não-alfabéticos e insere um caráter de enchimento ('X') entre letras repetidas num digrama ou no final do texto caso este tenha um comprimento ímpar.

3.5.1 Lógica de Cifragem e Decifragem

O núcleo do algoritmo reside na localização das coordenadas $(linha, coluna)$ de cada par de letras na matriz. A transformação segue três regras geométricas distintas, implementadas através de aritmética modular:

1. **Mesma Linha:** Na cifragem, as letras são substituídas pelas que se encontram imediatamente à direita (coluna + 1). Na decifragem, utiliza-se a letra imediatamente à esquerda (coluna - 1).
2. **Mesma Coluna:** Na cifragem, as letras são substituídas pelas que se encontram imediatamente abaixo (linha + 1). Na decifragem, utiliza-se a letra imediatamente acima (linha - 1).
3. **Retângulo:** As letras formam os cantos opostos de um retângulo. Neste caso, trocam-se as colunas das letras, mantendo-se as linhas originais. Esta operação é a mesma tanto na cifragem como na decifragem.

```
for i in range(0, len(plaintext), 2):
    row1, col1 = self._find_position(plaintext[i])
    row2, col2 = self._find_position(plaintext[i + 1])
```

```
    if row1 == row2:  # Mesma linha (shift direita)
        ciphertext += self.matrix[row1][(col1 + 1) % 5]
        ciphertext += self.matrix[row2][(col2 + 1) % 5]
    elif col1 == col2:  # Mesma coluna (shift baixo)
        ciphertext += self.matrix[(row1 + 1) % 5][col1]
        ciphertext += self.matrix[(row2 + 1) % 5][col2]
    else:  # Retangulo (troca cantos)
        ciphertext += self.matrix[row1][col2]
        ciphertext += self.matrix[row2][col1]
```

Listing 5: Cifragem: Aplicação das regras geométricas

```
for i in range(0, len(ciphertext), 2):
    row1, col1 = self._find_position(ciphertext[i])
    row2, col2 = self._find_position(ciphertext[i + 1])

    if row1 == row2:  # Mesma linha (shift esquerda)
        plaintext += self.matrix[row1][(col1 - 1) % 5]
        plaintext += self.matrix[row2][(col2 - 1) % 5]
    elif col1 == col2:  # Mesma coluna (shift cima)
        plaintext += self.matrix[(row1 - 1) % 5][col1]
        plaintext += self.matrix[(row2 - 1) % 5][col2]
    else:  # Retangulo (troca cantos - mantem-se igual)
        plaintext += self.matrix[row1][col2]
        plaintext += self.matrix[row2][col1]
```

Listing 6: Decifragem: Inversão do deslocamento

Para a decifragem, a lógica é idêntica, alterando apenas o operador aritmético nas regras de linha e coluna para subtração (ex: `(col1 - 1) % 5`), revertendo assim o deslocamento circular.

4 Análise de Segurança e Decisões de Projeto

Apesar da funcionalidade correta, identificam-se os seguintes aspetos críticos de segurança:

1. **Gestão de Chaves:** As chaves são lidas de ficheiros de texto simples. Em produção, dever-se-ia utilizar um KMS (Key Management System).
2. **Integridade:** O modo CBC garante confidencialidade, mas não integridade. A aplicação é vulnerável a ataques de modificação de bits ("bit-flipping"), pois não implementa HMAC para autenticação da mensagem.
3. **Memória:** Python não permite limpeza segura de memória, deixando chaves expostas em RAM durante a execução.

5 Testes e Resultados

Foram realizados testes de cifra e decifra garantindo que o ficheiro decifrado é binariamente idêntico ao original.

Figura 2: Exemplo de operação com AES.

6 Conclusão

O desenvolvimento deste projeto permitiu consolidar a distinção prática entre cifras de fluxo de texto (Clássicas) e cifras de bloco binário (Modernas). A implementação do modo CBC e a gestão manual de tabelas no Playfair foram os principais desafios superados, resultando numa ferramenta funcional e didática.