



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

(De)cipher™

Martinho José Novo Caeiro - 23917
Paulo António Tavares Abade - 23919



Beja, outubro de 2025

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

(De)cipher™

Martinho José Novo Caeiro - 23917
Paulo António Tavares Abade - 23919

Orientador: Rui Miguel Silva

Beja, outubro de 2025

Resumo

Este relatório descreve o desenvolvimento de uma aplicação de cifra e decifra de ficheiros, implementada em Python, uma linguagem de alto nível. O objetivo é explorar cifras de criptografia simétrica e demonstrar a aplicação prática das mesmas. Esta aplicação é desenvolvida no âmbito da unidade curricular de Criptografia e Criptanalise Aplicadas IPBeja, 2025.

Keywords: python, criptanalise, criptografia simétrica

Abstract

This report describes the development of a file encryption and decryption application, implemented in Python, a high-level language. The objective is to explore symmetric encryption ciphers and demonstrate their practical application. This application is developed within the scope of the course Cryptography and Applied Cryptanalysis IPBeja, 2025.

Keywords: python, cryptanalysis, symmetric cryptography

Índice

1	Introdução	1
2	Teoria	1
2.1	AES	1
2.2	DES	1
2.3	Vigenère	1
2.4	PlayFair	1
3	Desenvolvimento da Aplicação	2
3.1	GUI	2
3.2	AES	3
3.3	DES	4
3.4	Vigenère	5
3.5	PlayFair	6
3.6	Testes e Resultados	7
4	Conclusão	9
	Bibliografia	10

Índice de Figuras

1	Exemplo da interface gráfica da aplicação.	2
2	Cifragem e Decifragem com AES	3
3	Cifragem e Decifragem com DES	4
4	Cifragem e Decifragem com Vigenère	5
5	Cifragem e Decifragem com PlayFair	6
6	Testes com AES	7
7	Testes com DES	7
8	Testes com Vigenère	8
9	Testes com PlayFair	8

1 Introdução

Esta aplicação consistirá na junção de 4 cifras de criptografia simétrica, AES, DES, Vigenère e PlayFair. A aplicação será desenvolvida em Python (Python Software Foundation, 2025) e tem como base uma interface gráfica simples (GUI), permitindo ao utilizador interagir com o sistema de forma simples e eficiente. Todo o progresso do projeto será documentado no repositório GitHub (Martinho Caeiro & Paulo Abade, 2025).

2 Teoria

2.1 AES

O AES (Advanced Encryption Standard) (Wikipedia, 2025a) é um padrão de criptografia simétrica amplamente utilizado para proteger dados. Ele utiliza blocos de 128 bits e chaves de 128, 192 ou 256 bits, oferecendo alta segurança e eficiência.

2.2 DES

O DES (Data Encryption Standard) (Wikipedia, 2025d) é um algoritmo de criptografia simétrica que foi amplamente utilizado no passado. Ele opera em blocos de 64 bits e utiliza uma chave de 56 bits. Embora tenha sido substituído por algoritmos mais seguros, o DES ainda é relevante para fins educacionais.

2.3 Vigenère

A cifra de Vigenère (Wikipedia, 2025c) é um método de criptografia que utiliza uma palavra-chave para cifrar o texto. Ela é baseada na cifra de César, mas em vez de usar um único deslocamento, ela aplica deslocamentos diferentes com base nas letras da palavra-chave.

2.4 PlayFair

A cifra de PlayFair (Wikipedia, 2025b) é um método de criptografia que utiliza uma matriz de 5x5 para cifrar pares de letras. Ela é mais segura do que a cifra de César, pois não utiliza um deslocamento fixo.

3 Desenvolvimento da Aplicação

Todos os seguintes scripts foram desenvolvidos em Python com o uso do VS Code (Microsoft Corporation, 2025) e todos verificam se a maquina possui os pacotes necessários para o funcionamento da ferramenta, irá perguntar se quer instalar para poder continuar a utilizar. Caso o utilizador deseje prosseguir, o script irá instalar os pacotes necessários e depois irá ativá-los.

3.1 GUI

A GUI é o metodo principal de interação com a ferramenta. Para aceder a um dos metodos de cifra/decifra, o utilizador deve seleccionar a aba correspondente. Para executar a GUI apenas é necessário escrever 'python gui.py'.

A GUI apresenta-se da seguinte forma:

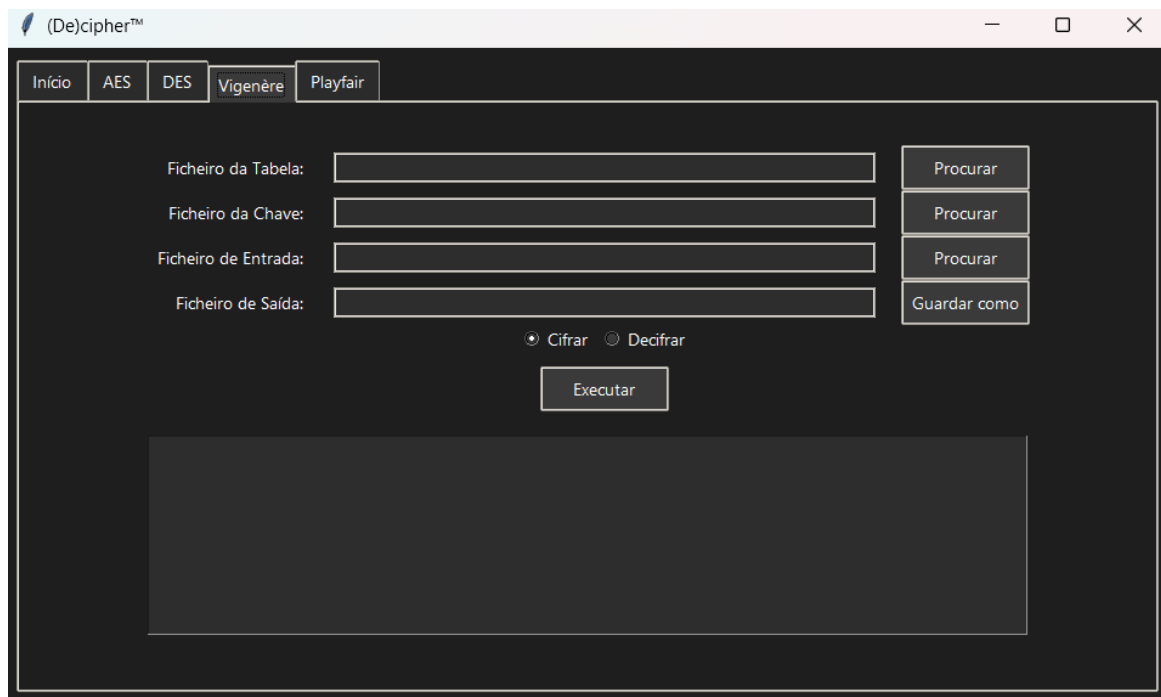


Figura 1: Exemplo da interface gráfica da aplicação.

3.2 AES

Implementou-se uma ferramenta para cifragem e decifragem de ficheiros utilizando AES em modo CBC (biblioteca PyCryptodome). A chave é carregada a partir de ficheiro através da rotina interna `_read_key_file`. Essa rotina aceita uma string hexadecimal (preferencial) ou texto UTF-8, converte para `bytes` e valida o comprimento (16, 24 ou 32 bytes). O formato do ficheiro cifrado utilizado pelo módulo é IV (16 bytes) concatenado com o ciphertext. A encriptação aplica preenchimento compatível com PKCS#7 para completar blocos de 16 bytes; a descifragem valida e remove esse preenchimento. Em caso de ficheiro de chave inexistente, tamanho de chave inválido ou padding inválido, são lançadas exceções apropriadas.

```
def encrypt_file(input_path: str, output_path: str, key: str = ""):
    """Encrypt a file with AES-CBC and write IV||ciphertext to output.

    Args:
        input_path: Path to the plaintext input file (opened in binary).
        output_path: Path to write the encrypted file (binary).
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    cipher = AES.new(key_bytes, AES.MODE_CBC)
    iv = cipher.iv

    with open(input_path, "rb") as f_in:
        data = f_in.read()

    pad_len = AES.block_size - (len(data) % AES.block_size)
    data += bytes([pad_len]) * pad_len

    ciphertext = cipher.encrypt(data)

    with open(output_path, "wb") as f_out:
        f_out.write(iv + ciphertext)
```

(a) Cifragem

```
def decrypt_file(input_path: str, output_path: str, key: str = ""):
    """Decrypt a file produced by :func:`encrypt_file`.

    Args:
        input_path: Path to the encrypted input file.
        output_path: Path to write the decrypted plaintext.
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
        ValueError: if padding is invalid (likely wrong key).
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    with open(input_path, "rb") as f_in:
        iv = f_in.read(16)
        ciphertext = f_in.read()

    cipher = AES.new(key_bytes, AES.MODE_CBC, iv=iv)
    data = cipher.decrypt(ciphertext)

    pad_len = data[-1]
    if pad_len < 1 or pad_len > AES.block_size:
        raise ValueError("Padding inválido ou chave incorreta.")
    data = data[:-pad_len]

    with open(output_path, "wb") as f_out:
        f_out.write(data)
```

(b) Decifragem

Figura 2: Cifragem e Decifragem com AES

3.3 DES

Implementou-se uma ferramenta para cifragem e decifragem de ficheiros utilizando DES em modo CBC (biblioteca PyCryptodome). A chave é carregada a partir de ficheiro pela rotina interna `_read_key_file`, que aceita uma string hexadecimal (preferencial) ou texto UTF-8, converte para `bytes` e valida o comprimento (exatamente 8 bytes). O formato do ficheiro cifrado é IV (8 bytes) concatenado com o ciphertext. A encriptação aplica preenchimento compatível com PKCS#5 (blocos de 8 bytes); a descifragem valida e remove esse preenchimento. Em caso de ficheiro de chave inexistente, tamanho de chave inválido ou padding inválido, são lançadas exceções apropriadas.

```
def encrypt_file(input_path: str, output_path: str, key: str = ""):
    """Encrypt a file using DES-CBC and write IV||ciphertext to output.

    Args:
        input_path: Path to the plaintext file (binary).
        output_path: Path where the encrypted file will be written.
        key: Path to the key file (hex or text).
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    cipher = DES.new(key_bytes, DES.MODE_CBC)
    iv = cipher.iv

    with open(input_path, "rb") as f_in:
        data = f_in.read()

    pad_len = DES.block_size - (len(data) % DES.block_size)
    data += bytes([pad_len]) * pad_len

    ciphertext = cipher.encrypt(data)

    with open(output_path, "wb") as f_out:
        f_out.write(iv + ciphertext)
```

(a) Cifragem

```
def decrypt_file(input_path: str, output_path: str, key: str = ""):
    """Decrypt a file produced by :func: encrypt_file and write plaintext.

    Args:
        input_path: Path to the encrypted input file.
        output_path: Path to write the recovered plaintext.
        key: Path to the key file (hex or text).
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    with open(input_path, "rb") as f_in:
        iv = f_in.read(8)
        ciphertext = f_in.read()

    cipher = DES.new(key_bytes, DES.MODE_CBC, iv=iv)
    data = cipher.decrypt(ciphertext)

    pad_len = data[-1]
    if pad_len < 1 or pad_len > DES.block_size:
        raise ValueError("Padding inválido ou chave incorreta.")
    data = data[:-pad_len]

    with open(output_path, "wb") as f_out:
        f_out.write(data)
```

(b) Decifragem

Figura 3: Cifragem e Decifragem com DES

3.4 Vigenère

Implementou-se uma ferramenta para cifragem e decifragem de ficheiros usando a cifra de Vigenère com base numa tabela 26×26 carregada através de um ficheiro. A rotina interna `read_table_from_file` lê e valida uma tabela com 26 linhas de 26 letras (A–Z); `read_key_from_file` lê a chave, filtra apenas letras A–Z e devolve uma string em maiúsculas (válida que não esteja vazia). O texto é normalizado para letras ASCII visíveis em maiúsculas antes do processamento; a função `process_text` avança o índice da chave apenas sobre letras e usa `encrypt_char/decrypt_char` para cifrar/decifrar carácter a carácter segundo a linha da tabela definida pelo carácter da chave. As funções `encrypt_file/decrypt_file` recebem `key` como o par `[table_path, key_path]`, lêem os ficheiros de entrada e escrevem ficheiros de saída contendo apenas letras maiúsculas A–Z. Em caso de ficheiro de tabela/chave inexistente, tabela com formato inválido ou chave vazia, são lançadas exceções apropriadas.

```
def process_text(text: str, key: str, table, mode: str = "encrypt") -> str:
    """Encrypt or decrypt `text` using Vigenère with the provided table.

    The function normalizes `text` to upper-case ASCII letters before
    processing and advances the key index across letters only.
    """
    text = ''.join(ch for ch in text.upper() if ch.isascii() and ch.isalpha())
    result = ""
    key_index = 0
    for ch in text:
        key_ch = key[key_index % len(key)]
        if mode == "encrypt":
            result += encrypt_char(ch, key_ch, table)
        else:
            result += decrypt_char(ch, key_ch, table)
        key_index += 1
    return result
```

Figura 4: Cifragem e Decifragem com Vigenère

3.5 PlayFair

Implementou-se uma ferramenta para cifragem e decifragem de ficheiros usando a cifra de Playfair com uma tabela 5×5 carregada através de um ficheiro. A rotina interna `read_board_from_file` lê e valida a tabela (linhas de letras; caracteres não-alfabéticos são ignorados), mapeia 'J' para 'I' e, se necessário, preenche os restantes caracteres A..Z (sem J). A normalização do texto é feita por `fix_message`: conserva apenas letras ASCII, converte para maiúsculas, insere 'X' entre letras duplicadas num digrafo e adiciona 'X' se necessário para obter comprimento par. O par (digrafo) é transformado por `process_pair` segundo as regras de Playfair (mesma linha → deslocamento horizontal; mesma coluna → deslocamento vertical; caso rectangular → troca de colunas). As funções `encrypt_text/decrypt_text` aplicam o processamento ao texto; `encrypt_file/decrypt_file` leem/escrevem ficheiros de texto usando o caminho da board como `key`. Em caso de ficheiro de board inexistente são lançadas exceções apropriadas.

```
def process_pair(board, a: str, b: str, mode: str = "encrypt") -> str:
    """Process a digraph (a,b) according to Playfair rules.

    Returns the transformed pair as a two-character string. Mode may be
    "encrypt" or "decrypt".
    """
    pos_a = search_letter(board, a)
    pos_b = search_letter(board, b)
    if not pos_a or not pos_b:
        return a + b

    if pos_a[0] == pos_b[0]:
        if mode == "encrypt":
            return board[pos_a[0]][(pos_a[1] + 1) % 5] + board[pos_b[0]][(pos_b[1] + 1) % 5]
        else:
            return board[pos_a[0]][(pos_a[1] - 1) % 5] + board[pos_b[0]][(pos_b[1] - 1) % 5]

    elif pos_a[1] == pos_b[1]:
        if mode == "encrypt":
            return board[(pos_a[0] + 1) % 5][pos_a[1]] + board[(pos_b[0] + 1) % 5][pos_b[1]]
        else:
            return board[(pos_a[0] - 1) % 5][pos_a[1]] + board[(pos_b[0] - 1) % 5][pos_b[1]]

    else:
        return board[pos_a[0]][pos_b[1]] + board[pos_b[0]][pos_a[1]]
```

Figura 5: Cifragem e Decifragem com PlayFair

3.6 Testes e Resultados

```
def encrypt_file(input_path: str, output_path: str, key: str = ""):
    """Encrypt a file with AES-CBC and write IV||ciphertext to output.

    Args:
        input_path: Path to the plaintext input file (opened in binary).
        output_path: Path to write the encrypted file (binary).
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    cipher = AES.new(key_bytes, AES.MODE_CBC)
    iv = cipher.iv

    with open(input_path, "rb") as f_in:
        data = f_in.read()

    pad_len = AES.block_size - (len(data) % AES.block_size)
    data += bytes([pad_len]) * pad_len

    ciphertext = cipher.encrypt(data)

    with open(output_path, "wb") as f_out:
        f_out.write(iv + ciphertext)
```

(a) Antes da Cifragem

```
def decrypt_file(input_path: str, output_path: str, key: str = ""):
    """Decrypt a file produced by :func:`encrypt_file`.

    Args:
        input_path: Path to the encrypted input file.
        output_path: Path to write the decrypted plaintext.
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
        ValueError: if padding is invalid (likely wrong key).
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    with open(input_path, "rb") as f_in:
        iv = f_in.read(16)
        ciphertext = f_in.read()

    cipher = AES.new(key_bytes, AES.MODE_CBC, iv=iv)
    data = cipher.decrypt(ciphertext)

    pad_len = data[-1]
    if pad_len < 1 or pad_len > AES.block_size:
        raise ValueError("Padding inválido ou chave incorreta.")
    data = data[: -pad_len]

    with open(output_path, "wb") as f_out:
        f_out.write(data)
```

(b) Depois da Decifragem

Figura 6: Testes com AES

```
def encrypt_file(input_path: str, output_path: str, key: str = ""):
    """Encrypt a file with AES-CBC and write IV||ciphertext to output.

    Args:
        input_path: Path to the plaintext input file (opened in binary).
        output_path: Path to write the encrypted file (binary).
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    cipher = AES.new(key_bytes, AES.MODE_CBC)
    iv = cipher.iv

    with open(input_path, "rb") as f_in:
        data = f_in.read()

    pad_len = AES.block_size - (len(data) % AES.block_size)
    data += bytes([pad_len]) * pad_len

    ciphertext = cipher.encrypt(data)

    with open(output_path, "wb") as f_out:
        f_out.write(iv + ciphertext)
```

(a) Antes da Cifragem

```
def decrypt_file(input_path: str, output_path: str, key: str = ""):
    """Decrypt a file produced by :func:`encrypt_file`.

    Args:
        input_path: Path to the encrypted input file.
        output_path: Path to write the decrypted plaintext.
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
        ValueError: if padding is invalid (likely wrong key).
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    with open(input_path, "rb") as f_in:
        iv = f_in.read(16)
        ciphertext = f_in.read()

    cipher = AES.new(key_bytes, AES.MODE_CBC, iv=iv)
    data = cipher.decrypt(ciphertext)

    pad_len = data[-1]
    if pad_len < 1 or pad_len > AES.block_size:
        raise ValueError("Padding inválido ou chave incorreta.")
    data = data[: -pad_len]

    with open(output_path, "wb") as f_out:
        f_out.write(data)
```

(b) Depois da Decifragem

Figura 7: Testes com DES

```
def encrypt_file(input_path: str, output_path: str, key: str = ""):
    """Encrypt a file with AES-CBC and write IV||ciphertext to output.

    Args:
        input_path: Path to the plaintext input file (opened in binary).
        output_path: Path to write the encrypted file (binary).
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    cipher = AES.new(key_bytes, AES.MODE_CBC)
    iv = cipher.iv

    with open(input_path, "rb") as f_in:
        data = f_in.read()

    pad_len = AES.block_size - (len(data) % AES.block_size)
    data += bytes([pad_len]) * pad_len

    ciphertext = cipher.encrypt(data)

    with open(output_path, "wb") as f_out:
        f_out.write(iv + ciphertext)
```

(a) Antes da Cifragem

```
def decrypt_file(input_path: str, output_path: str, key: str = ""):
    """Decrypt a file produced by :func:`encrypt_file`.

    Args:
        input_path: Path to the encrypted input file.
        output_path: Path to write the decrypted plaintext.
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
        ValueError: if padding is invalid (likely wrong key).
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    with open(input_path, "rb") as f_in:
        iv = f_in.read(16)
        ciphertext = f_in.read()

    cipher = AES.new(key_bytes, AES.MODE_CBC, iv=iv)
    data = cipher.decrypt(ciphertext)

    pad_len = data[-1]
    if pad_len < 1 or pad_len > AES.block_size:
        raise ValueError("Padding inválido ou chave incorreta.")
    data = data[:-pad_len]

    with open(output_path, "wb") as f_out:
        f_out.write(data)
```

(b) Depois da Decifragem

Figura 8: Testes com Vigenère

```
def encrypt_file(input_path: str, output_path: str, key: str = ""):
    """Encrypt a file with AES-CBC and write IV||ciphertext to output.

    Args:
        input_path: Path to the plaintext input file (opened in binary).
        output_path: Path to write the encrypted file (binary).
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    cipher = AES.new(key_bytes, AES.MODE_CBC)
    iv = cipher.iv

    with open(input_path, "rb") as f_in:
        data = f_in.read()

    pad_len = AES.block_size - (len(data) % AES.block_size)
    data += bytes([pad_len]) * pad_len

    ciphertext = cipher.encrypt(data)

    with open(output_path, "wb") as f_out:
        f_out.write(iv + ciphertext)
```

(a) Antes da Cifragem

```
def decrypt_file(input_path: str, output_path: str, key: str = ""):
    """Decrypt a file produced by :func:`encrypt_file`.

    Args:
        input_path: Path to the encrypted input file.
        output_path: Path to write the decrypted plaintext.
        key: Path to the key file (hex or text).

    Raises:
        FileNotFoundError: if the key file doesn't exist.
        ValueError: if padding is invalid (likely wrong key).
    """
    if not os.path.isfile(key):
        raise FileNotFoundError(f"Ficheiro de chave não encontrado: {key}")
    key_bytes = _read_key_file(key)

    with open(input_path, "rb") as f_in:
        iv = f_in.read(16)
        ciphertext = f_in.read()

    cipher = AES.new(key_bytes, AES.MODE_CBC, iv=iv)
    data = cipher.decrypt(ciphertext)

    pad_len = data[-1]
    if pad_len < 1 or pad_len > AES.block_size:
        raise ValueError("Padding inválido ou chave incorreta.")
    data = data[:-pad_len]

    with open(output_path, "wb") as f_out:
        f_out.write(data)
```

(b) Depois da Decifragem

Figura 9: Testes com PlayFair

4 Conclusão

O desenvolvimento desta aplicação permitiu explorar conceitos fundamentais sobre cifras simétricas. Apesar de alguns desafios encontrados, foi possível implementar as cifras solicitadas e configurar uma interface centralizada para facilitar a utilização das mesmas. Durante este processo, foram revistos conceitos previamente estudados na licenciatura de Engenharia Informática.

Bibliografia

- IPBeja. (2025). *Disciplina: Criptografia e Criptanalise Aplicadas / IPBeja* [Página LPD]. Obtido outubro 30, 2025, de <https://cms.ipbeja.pt/course/view.php?id=544>
- Martinho Caeiro & Paulo Abade. (2025). *Decipher-Tool - Repositório de Código* [Repositório da Aplicação Decipher-Tool]. Obtido outubro 30, 2025, de <https://github.com/MartinhoCaeiro/Decipher-Tool>
- Microsoft Corporation. (2025). *Visual Studio Code* [Editor de Texto Visual Studio Code]. Obtido outubro 30, 2025, de <https://code.visualstudio.com/>
- Python Software Foundation. (2025). *Welcome to Python.org* [Linguagem de Programação Python]. Obtido outubro 30, 2025, de <https://www.python.org/>
- Wikipedia. (2025a). *Advanced Encryption Standard*. Obtido outubro 30, 2025, de https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- Wikipedia. (2025b). *Cifra de Playfair*. Obtido outubro 30, 2025, de https://en.wikipedia.org/wiki/Playfair_cipher
- Wikipedia. (2025c). *Cifra de Vigenère*. Obtido outubro 30, 2025, de https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher
- Wikipedia. (2025d). *Data Encryption Standard*. Obtido outubro 30, 2025, de https://en.wikipedia.org/wiki/Data_Encryption_Standard