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

Relatório Técnico – Componente 1

Realizado por:
Rafael Narciso (n.º 24473)
Hugo Diogo (n.º 18803)

Docente:
Prof. Rui Miguel Silva

Beja, dezembro de 2025

Conteúdo

1 Introdução

Este relatório descreve o desenvolvimento de uma aplicação em modo gráfico para a cifragem e decifragem de ficheiros, realizada no âmbito da unidade curricular de Criptografia e Criptanálise Aplicadas.

O objetivo principal deste trabalho é consolidar os conhecimentos sobre criptografia simétrica através da implementação "de raiz"de algoritmos clássicos (Vigenère e Playfair) e da integração de bibliotecas para algoritmos modernos (DES e AES).

2 Enquadramento Teórico e Tecnológico

2.1 Cifras Clássicas

As cifras clássicas implementadas operam ao nível do carater (byte visível).

- **Vigenère:** Uma cifra polialfabética que utiliza uma chave e uma tabela (Tabula Recta) para substituir caracteres.
- **Playfair:** Uma cifra de digramas que utiliza uma matriz 5×5 gerada a partir de uma chave.

2.2 Cifras Modernas

- **DES (Data Encryption Standard):** Algoritmo de bloco de 64 bits.
- **AES (Advanced Encryption Standard):** Sucessor do DES, operando com blocos de 128 bits e chaves de 128, 192 ou 256 bits.

2.3 Tecnologias Utilizadas

A aplicação foi desenvolvida na linguagem **Python** devido à sua versatilidade na manipulação de *strings* e *bytes*. Para a interface gráfica utilizou-se a biblioteca **CustomTkinter**, e para as cifras modernas a biblioteca **Cryptography**/**PyCryptodome**.

3 Arquitetura e Implementação

3.1 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 **Tkinter**, 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`).

Esta separação permite que os algoritmos sejam testados independentemente da interface, facilitando a manutenção e a correção de erros ("debugging").

3.2 Implementação das Cifras Clássicas

As cifras clássicas foram implementadas sem recurso a bibliotecas externas de criptografia, manipulando diretamente os caracteres ASCII e as suas representações numéricas.

3.2.1 Cifra de Vigenère

A implementação da cifra de Vigenère (classe `VigenereCipher`) baseia-se na aritmética modular. A função `_extend_key` garante que a chave tem o mesmo comprimento que o texto a cifrar. A cifragem é realizada através da soma dos índices dos caracteres na tabela alfabética, módulo 26.

```
def encrypt(self, plaintext):
    for i, char in enumerate(plaintext):
        if char.isalpha():
            row = ord(key[i]) - ord('A')
            col = ord(char) - ord('A')

            encrypted_char = self.table[row][col]
            ciphertext += encrypted_char
    return ciphertext
```

Listing 1: Lógica central da cifra de Vigenère

3.2.2 Cifra de Playfair

A cifra de Playfair (classe `PlayfairCipher`) exigiu uma implementação mais complexa devido à necessidade de gerar uma matriz 5×5 e processar digramas (pares de letras).

- **Matriz:** A função `_create_matrix` remove duplicados da chave e preenche a matriz com o restante alfabeto, fundindo as letras 'J' e 'I' numa única posição para caber na grelha de 25 caracteres.
- **Preparação do Texto:** A função `_prepare_text` insere um caráter de enchimento ('X') caso existam letras repetidas num digrama ou se o texto tiver um comprimento ímpar.

```
if text[i] == text[i + 1]:  # Letras iguais no par
    prepared += 'X'         # Insere 'X'
else:
    prepared += text[i + 1]
```

Listing 2: Tratamento de digramas na cifra Playfair

3.3 Implementação das Cifras Modernas

Para as cifras modernas, utilizou-se a biblioteca **PyCryptodome**, garantindo uma implementação robusta e segura dos algoritmos padrão.

3.3.1 Modo de Operação e Padding

Tanto para o AES como para o DES, optou-se pelo modo de operação **CBC (Cipher Block Chaining)**. Este modo requer um Vetor de Inicialização (IV) aleatório para garantir que o mesmo texto simples produza criptogramas diferentes. O *padding* (preenchimento) é realizado segundo a norma PKCS7, através da função `pad()` da biblioteca, garantindo que os dados têm um tamanho múltiplo do bloco (8 bytes para DES, 16 bytes para AES).

3.3.2 Gestão do Vetor de Inicialização (IV)

Uma decisão crítica de implementação foi a forma de armazenamento do IV. Como o IV é necessário para a decifragem mas não é secreto, a aplicação concatena o IV (em binário) diretamente no início do ficheiro cifrado.

```
def encrypt_file(self, data):
    cipher = AES.new(self.key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(data, AES.block_size))
    # Retorna o IV concatenado com o texto cifrado
    return cipher.iv + ct_bytes
```

Listing 3: Cifragem de ficheiros com AES em modo CBC

Na operação de decifragem, a aplicação lê os primeiros bytes do ficheiro (8 para DES, 16 para AES) para recuperar o IV e inicializar o algoritmo para a decifragem do restante conteúdo.

4 Análise de Segurança e Decisões de Projeto

A implementação realizada cumpre os requisitos funcionais propostos, no entanto, uma análise de segurança rigorosa revela vulnerabilidades inerentes à natureza académica do projeto, bem como decisões de design tomadas para mitigar riscos nos algoritmos modernos.

1. **Armazenamento de Material Criptográfico:** A leitura de chaves a partir de ficheiros de texto simples (*cleartext*) viola princípios fundamentais de gestão de segredos. Num ambiente de produção, esta abordagem expõe o sistema a fugas de informação triviais. A solução adequada passaria pela utilização de cofres digitais (*Key Management Systems* - KMS) ou módulos de segurança de hardware (HSM), garantindo que as chaves nunca residem em disco de forma legível.

2. **Segurança Semântica e Gestão do IV:** Para os algoritmos AES e DES, optou-se pelo modo de operação CBC (*Cipher Block Chaining*) em detrimento do inseguro modo ECB. Para garantir a segurança semântica — assegurando que a cifragem do mesmo texto simples resulta em criptogramas distintos —, é gerado um Vetor de Inicialização (IV) pseudoaleatório e criptograficamente seguro para cada operação. O IV é concatenado ao início do ficheiro cifrado; esta prática é segura, pois o IV não é um segredo, necessitando apenas de ser único (nonce) e imprevisível.

3. **Integridade e Autenticidade (Limitação):** É importante notar que a aplicação garante a confidencialidade, mas não a integridade dos dados. O modo CBC é maleável a ataques de inversão de bits. A ausência de um mecanismo de autenticação de mensagem (como HMAC ou o uso de modos autenticados como GCM - *Galois/Counter Mode*) significa que a aplicação não deteta se o ficheiro cifrado foi adulterado por terceiros antes da decifragem.

4. **Gestão de Memória:** A utilização de Python, uma linguagem de gestão automática de memória, impede o controlo granular sobre a limpeza de variáveis sensíveis (chaves e texto simples) da RAM após o uso, o que poderia permitir ataques de *memory dump* em máquinas comprometidas.

5 Manual de Utilização e Testes

5.1 Interface Gráfica

A Figura 1 demonstra a interface principal da aplicação.

Figura 1: Interface Gráfica da Aplicação

5.2 Casos de Teste

Foram realizados testes com ficheiros de texto (ASCII) para as cifras clássicas e ficheiros binários (imagens PDF) para as cifras modernas.

6 Conclusão

O desenvolvimento deste projeto permitiu compreender as diferenças fundamentais entre a manipulação de texto em cifras clássicas e a manipulação de blocos de bits em cifras modernas. A implementação "de raiz"revelou a complexidade algoritmica por trás de métodos históricos.