Relatório de Implementação do Projeto S-DES

Giulia Moura Ferreira, 20/00018795

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0201 - Segurança Computacional

giulia.ferreira@aluno.unb.br

1. Introdução

Este relatório documenta a implementação do algoritmo Simplified Data Encryption Standard (S-DES), uma versão simplificada do DES projetada para fins educacionais. O projeto tem como objetivo demonstrar o funcionamento básico de um sistema de criptografia simétrica, utilizando chaves de 10 bits e blocos de dados de 8 bits.

O repositório do projeto pode ser encontrado a partir desse link.

2. Objetivos

- Implementar o S-DES usando a linguagem Python.
- Compreender conceitos de permutação, substituição, operação XOR e função de Feistel.
- Realizar testes de encriptação e decriptação para validar a implementação.

3. Estrutura do Projeto

O projeto foi estruturado em módulos independentes para melhorar a organização e a manutenção do código:

3.1. Permutar.py

```
def permutar(bits, tabela):  13 usages
    return [bits[pos - 1] for pos in tabela]
                                                ]
```

Figure 1. permutar.py

Realiza uma permutação de bits com base em uma tabela fornecida.

- **bits** é uma lista que representa a sequência de bits a ser permutada.
- **tabela** é uma lista de índices que define a nova ordem dos bits

A função percorre cada posição especificada na tabela e usa essas posições para reorganizar os elementos da lista bits; como o índice em Python começa em 0, a função ajusta cada posição da tabela subtraindo 1 (pos - 1).

3.2. Deslocar_Esquerda.py

```
def deslocar_esquerda(bits, deslocamentos):  5 usages
    return bits[deslocamentos:] + bits[:deslocamentos]
```

Figure 2. deslocar_esquerda.py

Realiza um deslocamento circular à esquerda em uma lista de bits.

- **bits** é uma lista que representa a sequência de bits a ser deslocada.
- **tabela** é a quantidade de posições para deslocar à esquerda.

A função reorganiza a lista de bits dividindo-a em duas partes: a porção que começa na posição especificada pelo número de deslocamentos até o final e a porção que vai do início até a posição de deslocamento. Em seguida, essas duas partes são concatenadas para formar a nova lista.

3.3. Gerar_Chaves.py

```
from permutar import permutar
from deslocar_esquerda import deslocar_esquerda


def gerar_chaves(chave):  2 usages
    p10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
    p8 = [6, 3, 7, 4, 8, 5, 10, 9]
    chave_permutada = permutar(chave, p10)
    esquerda, direita = chave_permutada[:5], chave_permutada[5:]
    esquerda = deslocar_esquerda(esquerda, deslocamentos: 1)
    direita = deslocar_esquerda(direita, deslocamentos: 1)
    K1 = permutar(esquerda + direita, p8)
    esquerda = deslocar_esquerda(esquerda, deslocamentos: 2)
    direita = deslocar_esquerda(direita, deslocamentos: 2)
    K2 = permutar(esquerda + direita, p8)
    return K1, K2
```

Figure 3. gerar_chaves.py

Implementa a geração das duas subchaves K1 e K2 necessárias para o processo de encriptação e decriptação no S-DES.

- **Permutação Inicial (P10)**: A chave de 10 bits é reorganizada de acordo com a tabela p10

- **Deslocamento Circular à Esquerda**: A chave permutada é dividida em duas partes de 5 bits; ambas as partes são deslocadas à esquerda por 1 posição.
- **Primeira Subchave (K1)**: As duas metades deslocadas são combinadas; a permutação p8 é aplicada, selecionando e reorganizando 8 dos 10 bits para gerar a K1.
- **Deslocamento Circular Duplo**: As duas metades são novamente deslocadas à esquerda, desta vez por 2 posições.
- **Segunda Subchave (K2)**: A permutação p8 é aplicada novamente para gerar a K2

3.4. XOR.py

```python
def xor(bits1, bits2):  3 usages
    return [b1 ^ b2 for b1, b2 in zip(bits1, bits2)]
```

Figure 4. xor.py

Realiza a operação lógica XOR (OU exclusivo) bit a bit entre dois vetores de bits bits1 e bits2.

- **bits1 e bits2** são as listas de bits de entrada 1 e 2.

Nessa função, cada bit de *bits1* é comparado com o bit correspondente de *bits2*. Se os bits forem diferentes, retorna 1. Se os bits forem iguais, retorna 0.

obs. A função zip() em Python é uma ferramenta que permite combinar elementos de duas ou mais sequências em uma única sequência de tuplas. [2]

3.5. Caixa_S.py

```python
def caixa_s(bits, caixa):  3 usages
    linha = (bits[0] << 1) | bits[3]
    coluna = (bits[1] << 1) | bits[2]
    valor = caixa[linha][coluna]
    return [int(b) for b in f'{valor:02b}']
```

Figure 5. caixa_s.py

Realiza uma substituição não linear usando uma matriz conhecida como “caixa S” (S-box). Ela é usada para embaralhar bits durante o processo de encriptação no S-DES. Ela funciona da seguinte forma:

1. **Determinando a linha**: Os bits 0 e 3 são usados para definir a linha na matriz.
2. **Determinando a coluna**: Os bits 1 e 2 determinam a coluna.
3. **Obtendo o valor**: O valor correspondente é extraído da matriz `caixa[linha][coluna]`.

4. **Convertendo para bits**: O valor extraído é convertido para uma representação binária de 2 bits e é retornado como uma lista de inteiros.

obs. O operador << retorna o valor de x com os bits deslocados para a esquerda em y posições. Os novos bits inseridos à direita são preenchidos com zeros. Esse deslocamento equivale a multiplicar x por 2 elevado à potência y. Deslocar um único bit para a esquerda corresponde a dobrar o valor de x. [1]

3.6. Funcao_FK.py

```python
from permutar import permutar
from xor import xor
from caixa_s import caixa_s


def funcao_fk(bits, subchave):  6 usages
    ep = [4, 1, 2, 3, 2, 3, 4, 1]
    p4 = [2, 4, 3, 1]
    s0 = [[1, 0, 3, 2], [3, 2, 1, 0], [0, 2, 1, 3], [3, 1, 3, 2]]
    s1 = [[0, 1, 2, 3], [2, 0, 1, 3], [3, 0, 1, 0], [2, 1, 0, 3]]
    esquerda, direita = bits[:4], bits[4:]
    direita_expandida = permutar(direita, ep)
    resultado_xor = xor(direita_expandida, subchave)
    esquerda_caixa = caixa_s(resultado_xor[:4], s0)
    direita_caixa = caixa_s(resultado_xor[4:], s1)
    combinacao = esquerda_caixa + direita_caixa
    return xor(esquerda, permutar(combinacao, p4))
```

Figure 6. funcao_fk.py

Implementa uma rodada da função de Feistel usada no processo de encriptação e decriptação no S-DES.

- **ep (expansão/permutação)** é uma lista de índices usados para expandir e reorganizar os bits da metade direita
- **p4 (permutação p4)** é uma lista de índices usada para permutar a saída das caixas S após a substituição.
- **s0 e s1 (caixas S)** são matrizes bidimensionais que representam as tabelas de substituição usadas na função.
- **esquerda e direita** os 8 bits de entrada são divididos em duas partes, sendo os 4 bits iniciais da esquerda e os 4 bits finais da direita.
- **direita_expandida** é o resultado da permutação e expansão da metade direita, usando a tabela ep.
- **resultado_xor** é o resultado da operação XOR entre `direita_expandida` e `subchave`

- **esquerda_caixa e direita_caixa** são as saídas das caixas S aplicadas em dois blocos de 4 bits extraídos de `resultado_xor`.
- **combinacao** é uma lista de 4 bits com os resultados das caixas S.
- **resultado_final** é a saída final, obtida a partir da operação XOR entre a metade da esquerda e a permutação entre *combinacao* e *p4*.

3.7. Trocar.py

```python
def trocar(bits):  4 usages
    return bits[4:] + bits[:4]
```

Figure 7. trocar.py

Realiza a troca das metades esquerda e direita de uma sequência de bits.

A função divide a lista *bits* em duas metades, a primeira são os últimos 4 bits e a segunda são os primeiros 4 bits. Em seguida, ela retorna essas duas partes concatenadas, efetivamente invertendo suas posições.

3.8. Encriptar.py e Decriptar.py

```python
from permutar import permutar
from funcao_fk import funcao_fk
from trocar import trocar


def encriptar(texto_claro, K1, K2):  2 usages
    ip = [2, 6, 3, 1, 4, 8, 5, 7]
    ip_inv = [4, 1, 3, 5, 7, 2, 8, 6]
    bits = permutar(texto_claro, ip)
    bits = funcao_fk(bits, K1) + bits[4:]
    bits = trocar(bits)
    bits = funcao_fk(bits, K2) + bits[4:]
    return permutar(bits, ip_inv)
```

```python
from permutar import permutar
from funcao_fk import funcao_fk
from trocar import trocar


def decriptar(texto_cifrado, K1, K2):  2 usages
    ip = [2, 6, 3, 1, 4, 8, 5, 7]
    ip_inv = [4, 1, 3, 5, 7, 2, 8, 6]
    bits = permutar(texto_cifrado, ip)
    bits = funcao_fk(bits, K2) + bits[4:]
    bits = trocar(bits)
    bits = funcao_fk(bits, K1) + bits[4:]
    return permutar(bits, ip_inv)
```

Figure 8. encriptar.py e decriptar.py

Ambas as funções implementam processos fundamentais do S-DES, mas com ordens de operações distintas.

Enquanto `encriptar.py` começa aplicando a subchave K1 na primeira rodada e K2 na segunda, `decriptar.py` realiza o processo inverso, usando K2 na primeira rodada e K1 na segunda. Além disso, ambas utilizam a permutação inicial e a permutação final inversa, porém o fluxo de execução em `decriptar.py` é ajustado para desfazer as operações realizadas durante a encriptação. Essa diferença garante que a mensagem original seja corretamente recuperada durante a decriptação.

3.9. Main.py

```python
from gerar_chaves import gerar_chaves
from encriptar import encriptar
from decriptar import decriptar


if __name__ == '__main__':
    chave = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
    texto_claro = [1, 1, 0, 1, 0, 1, 1, 1]
    K1, K2 = gerar_chaves(chave)

    texto_original = texto_claro
    print('Texto Original:', texto_original)

    texto_cifrado = encriptar(texto_claro, K1, K2)
    print('Texto Cifrado:', texto_cifrado)

    texto_decriptado = decriptar(texto_cifrado, K1, K2)
    print('Texto Decriptado:', texto_decriptado)
```

Figure 9. main.py

Esse arquivo funciona como o ponto de entrada principal para o programa de encriptação e decriptação usando o S-DES. Ele executa três etapas principais:

1. **Geração de Subchaves (K1 e K2)**: a função 3.3 é chamada, usando uma chave de 10 bits como entrada.
2. **Encriptação do Texto Claro**: a função "encriptar" 3.8 recebe um bloco de 8 bits como texto claro e as subchaves K1 e K2.
3. **Decriptação do Texto Cifrado**: a função "decriptar" 3.8 usa o texto cifrado e as mesmas subchaves para reverter o processo de encriptação e recuperar o texto original.

4. Descrição Técnica

O S-DES (Simplified Data Encryption Standard) é uma versão simplificada do algoritmo de criptografia DES. Foi criado para fins educacionais e permite entender conceitos como permutação, substituição e estrutura de Feistel, essenciais na criptografia simétrica. Ele trabalha com chaves de 10 bits e blocos de dados de 8 bits. [4]

Na criptografia, uma cifra de Feistel é uma estrutura simétrica usada na construção de cifras de bloco [5]. O processo de desencriptação de uma cifra de Feistel é igual ao processo de criptografia, usando-se nesse caso, o texto cifrado como entrada do algoritmo e as subchaves KN em ordem reversa [3].

5. Conclusão

A saída obtida após a execução do arquivo 3.9 foi:

```
Texto Original: [1, 1, 0, 1, 0, 1, 1, 1]
Texto Cifrado: [1, 0, 1, 0, 1, 0, 0, 0]
Texto Decriptado: [1, 1, 0, 1, 0, 1, 1, 1]
```

Figure 10. Saída

Os resultados foram consistentes com as especificações teóricas do algoritmo.

A implementação do S-DES foi concluída com sucesso, atendendo aos objetivos propostos. O projeto proporcionou uma compreensão prática dos fundamentos da criptografia simétrica e dos algoritmos de encriptação.

References

[1] Alex Barros. Operações bit-a-bit (bitwise) em python. https://ealexbarros.medium.com/operaμes-bit-a-bit-bitwise-em-python-e75624ce0611. Accessed: 2024-12-14.

[2] Ana Maria Gomes. A função zip em python: Um guia completo. https://hub.asimov.academy/tutorial/a-funcao-zip-em-python-um-guia-completo/. Accessed: 2024-12-14.

[3] Rita Peres. Cifra feistel. https://www.revista-programar.info/artigos/cifra-feistel/, 2017. Accessed: 2024-12-17.

[4] William Stallings. *Cryptography and Network Security*. Pearson, 5th edition, 2010.

[5] Wikipédia. Cifra feistel, 2024. [Online; accessed 17-dezembro-2024].