Trabalho 1
S-DES

Sofia Dy La Fuente Monteiro, 211055530

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

1. Introdução

O DES (*Data Encryption Standard*) é um algoritmo de criptografia simétrica desenvolvido na década de 1970 pela IBM, em colaboração com a NSA (Agência de Segurança Nacional dos EUA). Ele foi adotado como padrão de criptografia pelo governo dos EUA em 1976, porém, por conta do seu tamanho de chave pequeno (56 bits) o DES é considerado hoje como inseguro. Em 2002, o DES foi substituído por um novo e mais eficiente padrão de cifra de bloco chamado AES que usa chaves de 128 bits (ou mais) e opera em blocos de 128 bits.

O DES é baseado em dois atributos da cifra de Feistel: Substituição (também chamada de confusão) e Transposição (também chamada de difusão). O DES consiste em 16 etapas, cada uma delas chamada de rodada. Cada rodada executa as etapas de substituição e transposição, além de outras operações.

O funcionamento básico do DES começa com uma permutação inicial (IP), que reorganiza os bits do bloco de texto claro. Em seguida, o bloco é dividido em duas metades de 32 bits (esquerda e direita), que passam por 16 rodadas de transformação baseadas na estrutura de Feistel. Em cada rodada, a metade direita é expandida para 48 bits, combinada com uma subchave derivada da chave original usando operações de deslocamento e seleção de bits, e depois processada por caixas de substituição (S-boxes) que introduzem não-linearidade ao algoritmo. O resultado passa por uma permutação final e é combinado com a metade esquerda, trocando-se as duas metades para a próxima rodada. Após as 16 rodadas, as metades são reunidas e submetidas a uma permutação inversa (IP^{-1}), produzindo o texto cifrado final.

Esse trabalho tem como objetivo implementar o S-DES (Ou DES simplificado), que consiste em um algoritmo de criptografia educacional com estrutura fundamentais semelhantes ao DES, mas com parâmetros muito menores. Serão utilizados, por exemplo, uma chave de 10 bits e blocos de dados de 8 bits, além de realizar apenas duas rodadas da rede de Feistel.

Além do algoritmo, serão utilizados dois modos de operação de cifra de blocos: ECB (Electronic Codebook) e CBC (Cipher Block Chaining).

O S-DES é projetado para ajudar a entender os conceitos básicos de cifragem de blocos, utilizando chaves menores e um processo menos complexo.

Todo o código implementado foi colocado em um repositório no github e pode ser acessado através desse link: https://github.com/SofiaDyLaFuente/S-DES

2. S-DES

O Algoritmo de criptografia S-DES envolve cinco funções: uma permutação inicial (IP); uma função FK que envolve operações de permutação e substituição e depende de uma entrada chave; uma função de permutação simples (SW) que alterna as duas metades dos dados; uma função FK novamente; e uma função de permutação que é o inverso da permutação inicial (IP^{-1}).

2.1 Geração de Chaves Subjacentes

No algoritmo de encriptação implementado, a função de geração das chaves usa a chave de 10 bits para gerar duas subchave de 8 bits. Para isso, a chave será inicialmente permutada da seguinte maneira:

P10 (k1, k2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, k2, k7, k4, k10, k1, k9, k8, k6)

Podemos definir a primeira permutação como um vetor [3, 5, 2, 7, 4, 10, 1, 9, 8, 6] (Figura 1).

```python
# Geração de chaves subjacentes
def PermutacaoP10(chave):
    p10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
    chavePermutada = []

    for i in p10:
        chavePermutada.append(chave[i - 1])

    resultado = ''.join(chavePermutada)
    print(f'Chave P10: {resultado}')
    return resultado
```

Figura 1: Função para geração das chaves subjacentes

Para a primeira chave (K1) é feito um deslocamento circular em cada metade da chave permutada gerada, de modo que cada metade desloque 1 bit para a esquerda. Em seguida é feito uma segunda permutação (Figura 2) da forma:

P8
6 3 7 4 8 5 10 9

```python
def PermutacaoP8(chave):
    p8 = [6, 3, 7, 4, 8, 5, 10, 9]
    chavePermutada = []

    for i in p8:
        chavePermutada.append(chave[i - 1])

    resultado = ''.join(chavePermutada)
    print(f'Chave P8: {resultado}')
    return resultado
```

Figura 2: Função para permutação da chave

Gerando assim, a chave K1. Para a segunda chave (K2) é feito um novo deslocamento circular na primeira chave gerada, mas dessa vez deslocando 2 bits a esquerda. E por fim, é feito a segunda permutação (Figura 3).

```python
# Deslocamento Circular
def deslocamentoCircular(chave, shift):
    left = chave[:5]
    right = chave[5:]

    # Deslocamento circular à esquerda
    shiftedLeft = left[shift:] + left[:shift]
    shiftedRight = right[shift:] + right[:shift]

    print(f'Chave após deslocamento circular: {shiftedLeft} {shiftedRight}')
    return shiftedLeft + shiftedRight
```

Figura 3: Função para deslocamento circular

2.3 Permutação Inicial (IP)

A permutação inicial corresponde a permutação feita no bloco de dados de 8 bits. A ordem da permutação já está definida e corresponde a:

IP
2 6 3 1 4 8 5 7

Essa permutação é feita antes da rodada de Feistel (Figura 4).

```python
# Permutação Inicial (IP)
def PermutacaoInicial(bloco):
    ip = [2, 6, 3, 1, 4, 8, 5, 7]
    blocoPermutado = []

    for i in ip:
        blocoPermutado.append(bloco[i - 1])

    resultado = ''.join(blocoPermutado)

    print(f'Permutação inicial: {resultado}')
    return resultado
```

Figura 4: Função para permutação inicial (IP)

2.4 Rodadas de Feistel

O primeiro passo é pegar o bloco gerado na permutação inicial é dividi-lo em duas metades denominadas L e R (left e right). É feita uma expansão de bits e permutação (Figura 5) na metade direita, que já está pré definida e pode ser *conferida* a seguir:

EP
4 1 2 3 2 3 4 1

```python
# Expansão e Permutação (E/P)
def ExpansaoPermutacao(bits):
    ep = [4, 1, 2, 3, 2, 3, 4, 1]
    bitsPermutados = []

    for i in ep:
        bitsPermutados.append(bits[i - 1])

    resultado = ''.join(bitsPermutados)
    print(f'Expansão e Permutação: {resultado}')
    return resultado
```

Figura 5: Função de expansão e permutação

Em seguida é feito uma função XOR do resultado da expansão de permutação do bloco direito (right) com a subchave K1 (Figura 6).

```python
# XOR bit a bit entre x e y
def XOR(x, y):
    listaXor = []
    tamanho = len(x)

    for i in range(tamanho):
        bit1 = int(x[i])
        bit2 = int(y[i])
        xor = bit1 ^ bit2
        listaXor.append(str(xor))

    resultado = ''.join(listaXor)
    print(f'XOR: {resultado}')
    return (resultado)
```

Figura 6: Função XOR

Depois disso, o resultado do XOR passa pelas S-Boxes, que são responsáveis por introduzir não linearidade no processo de cifra. No S-DES existem duas S-box com tamanho 4x4, que recebem 4 bits de entrada e produzem 3 bits de saída. As S-box do S-DES funcionam da seguinte forma:

O primeiro e o quarto bit de entrada são tratados como um número de 2 bits, que especificam uma linha da S-box enquanto o segundo e o terceiro bits de entrada especificam uma coluna da S-box (Figura 7) As S-box são fixas e correspondem a:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

```python
def SBoxes(bits):
    left, right = bits[:4], bits[4:]

    def CalcularSBox(bloco, sbox):

        linha = int(bloco[0] + bloco[3], 2)  # Bits 1 e 4 para linha
        coluna = int(bloco[1] + bloco[2], 2)  # Bits 2 e 3 para coluna

        return format(sbox[linha][coluna], '02b')

    saidaS0 = CalcularSBox(left, S0)
    saidaS1 = CalcularSBox(right, S1)
    print(f'S-Boxes: {saidaS0} {saidaS1}')

    return saidaS0 + saidaS1
```

Figura 7: Função S-BOX

Depois de passar pelas S-box, o resultado produzido por S0 e S1 passam por uma nova permutação (Figura 8) da forma:

P4
2 4 3 1

```python
def PermutacaoP4(bits):
    p4 = [2, 4, 3, 1]
    bitsPermutados = []

    for i in p4:
        bitsPermutados.append(bits[i - 1])

    resultado = ''.join(bitsPermutados)
    print(f'Permutação P4: {resultado}')
    return resultado
```

Figura 8: Permutação P4

O próximo passo consiste em fazer novamente o XOR com a metade esquerda do bloco (left) e o resultado da permutação P4.

O resultado da função FK será a combinação de saída do XOR com o bloco direito (right) puro, sem permutações (Figura 9). Dessa forma temos a primeira rodada de Feistel.

```python
# Divisão das metades do bloco de dados + Função Fk
def Fk(bits, subchave, rodada):
    left, right = bits[:4], bits[4:]

    epBits = ExpansaoPermutacao(right)
    saidaXor = XOR(epBits, subchave)
    saidaSbox = SBoxes(saidaXor)
    saidaP4 = PermutacaoP4(saidaSbox)
    saidaFk = XOR(left, saidaP4)

    bloco = saidaFk + right
    print(f'{rodada}º rodada de Feistel: {bloco}')

    return bloco
```

Figura 9: Função FK

Para a segunda rodada de Feistel todo esse processo será repetido, com a diferença que o primeiro XOR será feito com a chave K2 e o bloco será o resultado da primeira rodada de Feistel com as metades esquerda e direita trocadas (Figura 10).

```python
def RodadasFeistel(bloco, K1, K2):
    # Primeira rodada com K1
    bloco1 = Fk(bloco, K1, 1)

    blocoInvertido = bloco1[4:] + bloco1[:4]

    # Segunda rodada com K2
    bloco2 = Fk(blocoInvertido, K2, 2)

    return bloco2
```

Figura 10: Função de rodadas de Feistel

2.5 Permutação Final

Após as rodadas de Feistel, uma permutação final (IP^{-1}), inversa a permutação inicial, é aplicada para obter o bloco cifrado (Figura 11). A ordem da permutação corresponde a:

(IP^{-1})
4 1 3 5 7 2 8 6

```python
# Permutação Final Inversa (IP⁻¹)
def PermutacaoFinal(bloco):
    ipInversa = [4, 1, 3, 5, 7, 2, 8, 6]
    blocoPermutado = []

    for i in ipInversa:
        blocoPermutado.append(bloco[i - 1])

    resultado = ''.join(blocoPermutado)
    print(f'Permutação final Inversa: {resultado}')
    return resultado
```

Figura 11: Função de Permutação Final Inversa

Observações relevantes:

- É importante destacar que a função de descriptografia é simétrica a de criptografia com a única diferença que a ordem das subchaves é inversa.
- Como especificado no enunciado do trabalho, todas as chaves possuem 10 bits e os blocos de dados têm exatamente 8 bits. Por esse motivo, não foi necessário implementar um esquema de padding. O algoritmo desenvolvido assume blocos de 8 bits como entrada; portanto, blocos com tamanho diferente não são suportados e resultarão em falha na execução.

O esquema mostrando todas as etapas do algoritmo S-DES pode ser conferido a seguir (Figura 12):

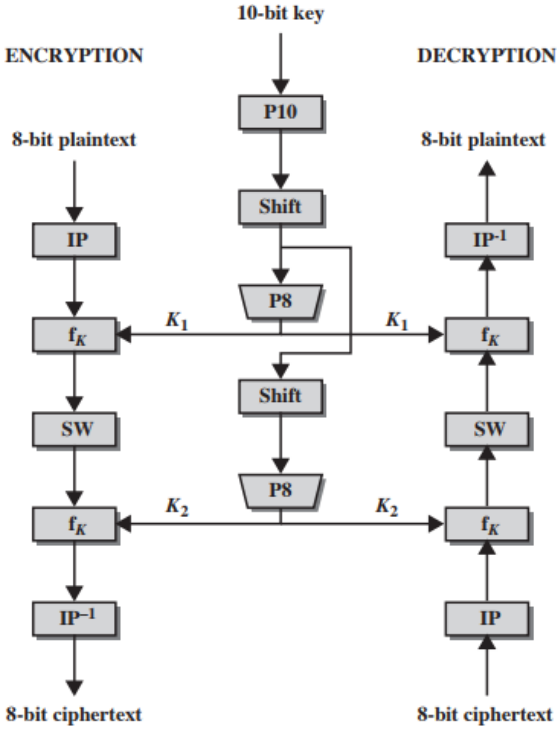


Figura 13: S-DES esquema

3. Modos de Operação

Um modo de operação de cifra de bloco é um método que usa uma cifra de bloco para fornecer um serviço de informação, confidencialidade ou autenticação, melhorando o efeito de um algoritmo criptográfico ou adaptando o algoritmo para uma aplicação. Existem 5 modos de operação que forma definidos pelo NIST, que são: Electronic codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), Counter (CTR). Para esse trabalho, apenas os modos de Electronic codebook (ECB) e Cipher Block Chaining (CBC) serão implementados.

3.1 Electronic codebook (ECB)

O modo electronic codebook corresponde ao modo mais simples no qual o texto claro é tratado por um bloco de cada vez, e cada bloco de texto claro é encriptado usando a mesma chave. O método ECB é ideal para uma pequena quantidade de dados, como uma chave de encriptação.

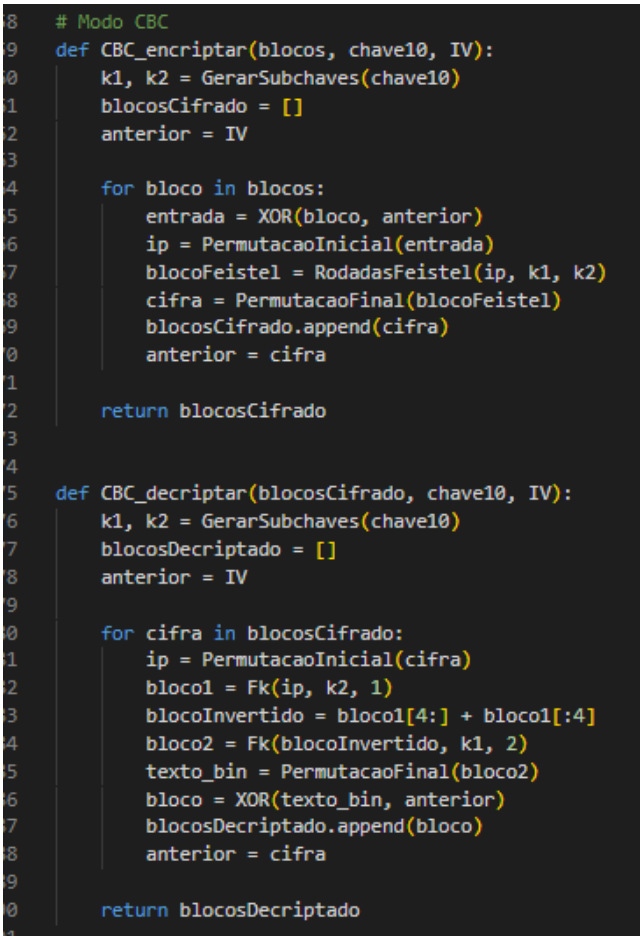
Para esse modo de operação, pegaremos o texto em claro fornecido e dividiremos em bloco de 8 bits. Cada bloco de 8 bits é cifrado/decifrado independentemente com a mesma chave (Figura 14)

```python
# Modo ECB
def ECB_encriptar(blocos, chave10):
    k1, k2 = GerarSubchaves(chave10)
    blocosCifrado = []

    for bloco in blocos:
        ip = PermutacaoInicial(bloco)
        blocoFeistel = RodadasFeistel(ip, k1, k2)
        cifra = PermutacaoFinal(blocoFeistel)
        blocosCifrado.append(cifra)

    return blocosCifrado


def ECB_decriptar(blocosCifrado, chave10):
    k1, k2 = GerarSubchaves(chave10)
    blocosDecriptado = []

    for cifra in blocosCifrado:
        ip = PermutacaoInicial(cifra)
        bloco1 = Fk(ip, k2, 1)
        blocoInvertido = bloco1[4:] + bloco1[:4]
        bloco2 = Fk(blocoInvertido, k1, 2)
        texto = PermutacaoFinal(bloco2)
        blocosDecriptado.append(texto)

    return blocosDecriptado
```

Figura 14: Modo de Operação ECB

3.2 Cipher Block Chaining (CBC)

Consiste em um modo de operação em que cada bloco de dados é cifrado utilizando a cifra do bloco anterior. Isso garante que blocos idênticos de dados gerem cifras diferentes, aumentando a segurança do algoritmo. A entrada do algoritmo de encriptação é o XOR do bloco de texto claro atual e do bloco de texto cifrado anterior. A mesma chave é usada para cada bloco, mas é necessário um vetor de inicialização. (Figura 15)



```python
# Modo CBC
def CBC_encriptar(blocos, chave10, IV):
    k1, k2 = GerarSubchaves(chave10)
    blocosCifrado = []
    anterior = IV

    for bloco in blocos:
        entrada = XOR(bloco, anterior)
        ip = PermutacaoInicial(entrada)
        blocoFeistel = RodadasFeistel(ip, k1, k2)
        cifra = PermutacaoFinal(blocoFeistel)
        blocosCifrado.append(cifra)
        anterior = cifra

    return blocosCifrado


def CBC_decriptar(blocosCifrado, chave10, IV):
    k1, k2 = GerarSubchaves(chave10)
    blocosDecriptado = []
    anterior = IV

    for cifra in blocosCifrado:
        ip = PermutacaoInicial(cifra)
        bloco1 = Fk(ip, k2, 1)
        blocoInvertido = bloco1[4:] + bloco1[:4]
        bloco2 = Fk(blocoInvertido, k1, 2)
        texto_bin = PermutacaoFinal(bloco2)
        bloco = XOR(texto_bin, anterior)
        blocosDecriptado.append(bloco)
        anterior = cifra

    return blocosDecriptado
```

Figura 15: Modo de operação CBC

4. Conclusão

Este trabalho teve como objetivo implementar o algoritmo S-DES (Simplified DES), uma versão educacional do DES (Data Encryption Standard), utilizando chaves de 10 bits e blocos de 8 bits, além de dois modos de operação: ECB (Electronic Codebook) e CBC (Cipher Block Chaining). A implementação do algoritmo S-DES permitiu aplicar na prática os conceitos fundamentais da criptografia de blocos, como permutação, substituição, geração de subchaves e estrutura de Feistel. Mesmo sendo uma versão simplificada do DES, o S-DES mantém os principais elementos que ajudam a entender o funcionamento de algoritmos de criptografia simétrica.

A implementação do S-DES contribuiu significativamente para o entendimento das operações fundamentais envolvidas em criptografia simétrica, fornecendo uma base sólida para o estudo de algoritmos mais complexos e aplicáveis a cenários reais, como o AES.