Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC0201 – Segurança Computacional – Turma 02 - 2024/2.
Prof. Lorena de Souza Bezerra Borges.

Integrante: Emanuel de Oliveira Barbosa. **Matrícula:** 211010403.

Trabalho de Implementação 1 – S-DES

1. Introdução

Data Encryption Standard (DES) é um algoritmo de criptografia simétrica desenvolvido na década de 1970. Utiliza uma chave de 56 bits para cifrar blocos de dados de 64 bits. Esta chave é conhecida tanto na origem quanto no destino e opera em 16 rodadas, nas quais o texto claro sofre permutações, expansões com a função de Feistel e substituições (*S-boxes*). Cabe citar que cada rodada gera uma subchave específica.

O S-DES (Simplified DES), por outro lado, é uma versão reduzida do DES. Seu objetivo é fornecer uma compreensão acessível do funcionamento dos cifradores de bloco, utilizando uma chave de 10 bits e operando em blocos de dados de 8 bits. Esta simplificação permite uma visualização detalhada dos processos de permutação, substituição e geração de subchaves, distribuídos em duas rodadas.

Este trabalho implementa o S-DES na linguagem Python, estruturado no formato de módulos que permitem o reaproveitamento dos métodos frequentemente utilizados no algoritmo, como as permutações e operações XOR. Cada método foi desenvolvido para ser independente e realizar uma única tarefa específica.

2. Implementação

Repositório: https://github.com/emanuelob/CIC0201_TP1_S-DES

O projeto foi desenvolvido em Python. Se você já tem a linguagem instalada no seu computador, basta rodar o comando *python sdes.py* no terminal. Caso contrário, você pode utilizar interpretadores online de Python, como o Replit.

Para exemplificar a saída de bits de cada etapa da Estrutura de Feistel, vamos considerar os seguintes dados:

- Chave de 10 bits: 1010000010 *(key)*;
- Bloco de dados de 8 bits: 11010111 *(plaintext)*.

Os parâmetros supracitados já estão configurados no código.

3. S-DES Encryption

A função de encriptação é a ***sdes_encrypt(plaintext, key)***. Nela, os seguintes processos são realizados sequencialmente:

3.1) Gera as subchaves K1 e K2;
3.2) Aplica a permutação inicial (IP);
3.3) Divide o bloco em duas metades;
3.4) Primeira rodada da função de Feistel:
    a) Aplica fk com K1;
    b) Realiza XOR com a metade esquerda;
    c) Troca as metades.
3.5) Segunda rodada da função de Feistel:
    a) Aplica fk com K2;
    b) Realiza XOR com a metade esquerda;

c) Troca as metades.

3.6) Aplica a permutação final (IP^{-1});

3.7) Retorna o texto cifrado.

i) Método Auxiliar: *permute(input_bits, permutation_table, perm_type)*

- ***input_bits:*** string de bits para permutar;
- ***permutation_table:*** tabela com as posições da permutação;
- ***perm_type:*** tipo de permutação ('P10', 'P8', 'IP', 'IP-1', 'EP', 'P4').

Este método realiza a reorganização dos bits de entrada de acordo com tabelas de permutação predefinidas, que variam conforme o contexto. Ele suporta diferentes tipos de permutação, como:

- **P10**: Reorganiza os 10 bits da chave para geração das subchaves.
- **P8**: Extrai e reorganiza 8 bits para formar subchaves.
- **IP** e **IP-1**: Realizam a permutação inicial e a inversa no bloco de dados.
- **EP**: Expande e permuta 4 bits para 8 bits na função fk.
- **P4**: Reorganiza os bits após o processamento nas S-boxes.

Como a permutação é um processo predominante no algoritmo, transformá-lo em um método auxiliar permite uma fácil manutenção e reutilização. Assim, cada permutação é implementada com base na tabela correspondente e utiliza os índices para reordenar os bits.

Finalmente, vamos passar por cada etapa individualmente.

3.1) Key Generation: *generate_subkeys(key)*

A geração de subchaves começa com a aplicação da permutação P10 (tabela pré-definida) na chave original, dividindo-a em duas metades de 5 bits. Considerando a chave 1010000010, a troca de posições dos bits teria o seguinte resultado:

P-10									
3	5	2	7	4	10	1	9	8	6
1	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

Em seguida, deslocamentos circulares são aplicados em cada metade da P10, e a permutação P8 é utilizada para formar as subchaves K1 e K2:

P-8							
6	3	7	4	8	5	10	9
Após LS-1 (00001 11000)							
1	**0**	**1**	**0**	**0**	**1**	**0**	**0**
Após LS-2 (00100 00011)							
0	**1**	**0**	**0**	**0**	**0**	**1**	**1**

Estas subchaves, K1= 10100100 e K2= 01000011, serão utilizadas em ordem durante o processo de criptografia e invertidas na descriptografia.

i)* Método Auxiliar: *left_shift(bits, positions)

- ***bits:*** string de bits a ser deslocada;
- ***positions:*** número de posições para deslocar os bits.

Este método é utilizado na geração das subchaves (K1 e K2) após a aplicação da permutação P10. Diferentes deslocamentos, como LS-1 (1 posição) e LS-2 (2 posições), são aplicados em cada etapa. Nesse contexto, ele realiza o deslocamento circular à esquerda dos bits pelo número de posições especificadas, garantindo que os bits deslocados retornem ao final da sequência.

3.2) Permutação Inicial (IP) e 3.3) Divisão do bloco em duas metades

A permutação inicial (IP) é aplicada ao bloco de dados de entrada (*plaintext*). Ela reorganiza os bits do bloco conforme uma tabela de permutação predeterminada. No caso do exemplo, temos:

IP							
2	6	3	1	4	8	5	7
1	**1**	**0**	**1**	**1**	**1**	**0**	**1**

Após, o bloco permutado é dividido em duas metades, esquerda (1101) e direita (1101), para serem processadas separadamente durante as rodadas da função de Feistel.

3.4) e 3.5) Aplicação das duas rodadas da função de Feistel (fk) com trocas de metades

O método ***function_fk(bits, subkey, round_num)*** combina o processamento de metade dos dados com uma subchave, onde:

- ***bits:*** string de bits (4 bits) a ser processada;
- ***subkey:*** subchave a ser usada na operação XOR;
- ***round_num:*** número da rodada (1 ou 2).

A função complexa fk inclui:

1. Expansão/permutação (EP) da metade direita para 8 bits.

EP (1101)							
4	1	2	3	2	3	4	1
1	**1**	**1**	**0**	**1**	**0**	**1**	**1**

2. Operação XOR com a subchave.

XOR (EP ⊕ k1)							
1	1	1	0	1	0	1	1
1	0	1	0	0	1	0	0
0	**1**	**0**	**0**	**1**	**1**	**1**	**1**

3. Divisão em dois blocos de 4 bits (S0 e S1) e substituição com S-boxes.

S-boxes (S0 = 0100 e S1= 1111)		
S0	Linha = bits externos = 00 Coluna = bits internos = 10	Valor correspondente na matriz = 11
S1	Linha = bits externos = 11 Coluna = bits internos = 11	Valor correspondente na matriz = 11

4. Reorganização dos resultados das S-boxes usando a permutação P4.

P-4			
2	4	3	1
1	**1**	**1**	**1**

5. Operação XOR com a metade esquerda original (IP ou SW).

XOR (P4 ⊕ IP)			
1	1	1	1
1	1	0	1
0	**0**	**1**	**0**

Após a operação XOR com o bloco permutado P4, o resultado 0010 é juntado com o bloco direito original do IP (1101), formando o bloco intermediário SW (0010 1101). Nesse estágio, ocorre uma troca das metades, de modo que o bloco direito (1101) se torna o bloco esquerdo, e o resultado da operação (0010) passa a ser o novo bloco direito.

Na segunda rodada da função, o bloco direito da SW (0010) é usado como entrada, junto com a subchave K2 (0010 0011), para repetir o processo de cifragem. Considerando os comportamentos acima exemplificados, os resultados intermediários seriam:

EP (0010)							
4	1	2	3	2	3	4	1
0	**0**	**0**	**1**	**0**	**1**	**0**	**0**
XOR (EP ⊕ k2)							
0	0	0	1	0	1	0	0
0	1	0	0	0	0	1	1
0	**1**	**0**	**1**	**0**	**1**	**1**	**1**

S-boxes (S0 = 0101 e S1= 0111)		
S0	Linha = bits externos = 01 Coluna = bits internos = 10	Valor correspondente na matriz = 01
S1	Linha = bits externos = 01 Coluna = bits internos = 11	Valor correspondente na matriz = 11

P-4			
2	4	3	1
1	**1**	**1**	**0**
XOR (P4 ⊕ SW)			
1	1	1	0
1	1	0	1
0	**0**	**1**	**1**

Após a operação XOR, o bloco esquerdo é atualizado com o valor de 0011, enquanto é concatenado com o bloco direito original do SW (0010), para a permutação final.

i) **Método Auxiliar: *xor_bits(bits1, bits2)***

- ***bits1:*** primeira string de bits para a operação XOR;
- ***bits2:*** segunda string de bits para a operação XOR.

Método que realiza a operação XOR entre duas strings de bits de igual comprimento. Para cada posição, se os bits são iguais, o resultado é '0'; se são diferentes, o resultado é '1'.

ii) **Método Auxiliar: *apply_sbox(bits, sbox, sbox_name)***

- ***bits:*** string de bits (4 bits) a ser processada;
- ***sbox:*** matriz S-box para a substituição;
- ***sbox_name:*** nome da S-box para exibição ('S0' ou 'S1').

Aplica a substituição usando uma S-box. A linha é determinada pelos bits externos (1º e 4º) e a coluna pelos bits internos (2º e 3º). O valor da S-box é convertido para uma representação binária de 2 bits.

3.6) Aplicação da permutação final (IP^{-1}) e 3.7) Retorno o texto cifrado
Finalmente, emprega a permutação final para obter o texto cifrado:

IP^{-1} (0011 0010)							
4	1	3	5	7	2	8	6
1	**0**	**1**	**0**	**1**	**0**	**0**	**0**

4. S-DES Decryption: *sdes_decrypt(ciphertext, key)*

A descriptografia segue a mesma estrutura da criptografia, mas as subchaves são aplicadas em ordem inversa (K2 na primeira rodada e K1 na segunda). Isso reverte o processo de criptografia, recuperando o texto original.

Do mesmo modo, ela gera subchaves, aplica a permutação inicial ao texto cifrado, divide em metades, realiza duas rodadas da função fk com trocas de metades, e finalmente aplica a permutação final para obter o texto plano.

5. Referências Bibliográficas

Material disponibilizado no Aprender3: STALLINGS, William. Appendix G Simplified DES.
RABELO, Rafael. *Criptografia Simétrica - O algoritmo Data Encryption Standard (DES)*. Programa de Mestrado Profissional em Engenharia Elétrica, Universidade de Brasília (UnB). Acesse aqui.
LECTURES BY SHREEDARSHAN K. *S-DES Encryption || Simplified data encryption standard (S-DES) || - Key Generation*. Acesse aqui.
LECTURES BY SHREEDARSHAN K. *S-DES Encryption || Simplified data encryption standard (S-DES) || - Encryption & Decryption*. Acesse aqui.
LECTURES BY SHREEDARSHAN K. *S - DES Decryption || Simplified data encryption standard (S-DES) || Explanation with example*. Acesse aqui.