Trabalho 1 - S-DES

Segurança Computacional - 2025/1

Pedro Avila Beneveli[1]

[1]Departamento de Ciência da Computação - Universidade de Brasília (UnB)
Brasília – DF – Brasil

pedro.beneveli@aluno.unb.br

A implementação desse trabalho está disponível no repositório `https://github.com/PedroABeneveli/S-DES`.

1. Parte I - Implementação do S-DES

O S-DES é uma versão simplificada do algortimo DES, desenvolvida pelo Professor Edward Schaefer da Universidade de Santa Clara para fins educacionais. Esse algoritmo funciona de forma extremamente similar ao DES, porém com parâmetros menores, já que o tamanho do bloco é de 8 bits, o tamanho da chave é 10 bits, e executa apenas 2 rodadas da rede de Feistel. A Figura 1, presente em [Stallings 2010], mostra a estrutura geral do S-DES.

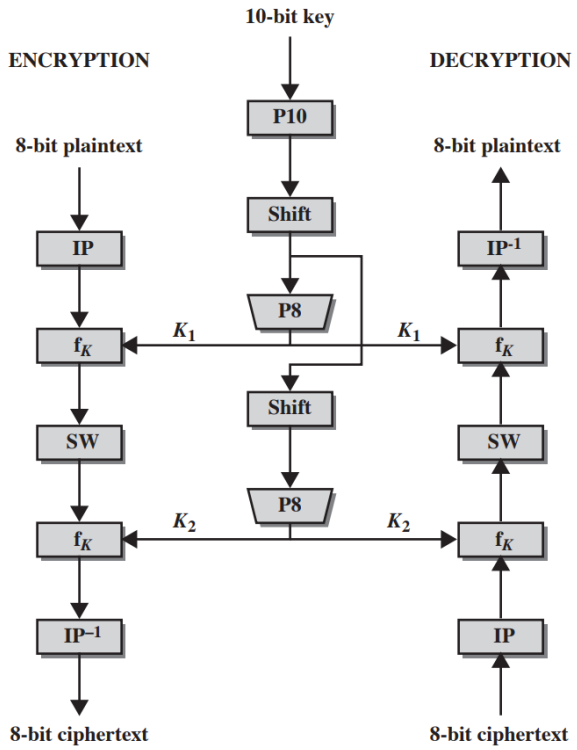

Figura 1. Estrutura do S-DES

Ao iniciar a implementação desse algoritmo em C++, notou-se um pequeno problema. Em [Stallings 2010], utilizado como principal referência para implementação, utilizou-se a notação de que o bit 1, o menos significativo (LSBit), se encontra na esquerda. Isso acaba diferindo do padronizado pelo objeto *bitset*, em que o LSBit se

encontra na direita. Por isso, foi necessário criar algumas funções para transformar as entradas no bitset correto, e formatar corretamente o resultado no bitset para a saída, além de levar isso em consideração na hora de combinar metades durante o algoritmo. As funções de transformação se encontram no arquivo *src/utils.cpp*.

1.1. Geração de Subchaves

Para realizar a geração das chaves, é necessária a implementação de 3 funções: a permutação inicial *P10* da chave, a seleção e permutação *P8*, e o shift cíclico *Shift*. As implementações dessas 3 funções se encontram no arquivo *src/bit_ops.cpp*.

Para realizar apenas a geração de chaves e observar todos os valores gerados no processo, é preciso apenas escolher a opção 1 no executável gerado na compilação do projeto. Um teste feito no executável está presente na Figura 2.

1.1.1. P10 e P8

A permutação P10 é aplicada inicialmente na chave. Para implementá-la, foi necessário apenas atribuir cada posição do resultado permutado o bit correspondente, seguindo a tabela apresentada na página 4 de [Stallings 2010].

Já P8, além de permutar os bits, seleciona apenas 8 dos 10 bits presentes na chave recebida, para formar a subchave correspondente. Essa subchave será utilizada posteriormente na função f_K, para realizar um XOR dentro da função.

1.1.2. Shift

Essa função realiza um shift circular na chave de 10 bits recebida pela função, recebendo também a quantidade de bits que serão shiftados. Porém, esse shift não é feito na chave inteira, e sim separadamente em cada metade. Ou seja, os primeiros 5 bits realizam um shift circular de x posições, e os 5 últimos bits também realizam separadamente um shift circular de x posições.

1.1.3. Subchaves

As subchaves K_1 e K_2 são geradas a partir da seguinte combinação das funções, onde K é a chave inicial.

$$K_1 = P8(Shift(P10(K), 1))$$
$$K_2 = P8(Shift(Shift(P10(K), 1), 2))$$

A função que gera ambas as subchaves está presente no arquivo *src/sdes.cpp*, chamada $subkey\_generation$.

1.2. Processo de Criptografia

Para criptografar uma mensagem, são necessárias diversas funções. Elas são IP, IP^{-1}, f_K, o mapeamento F e SW.

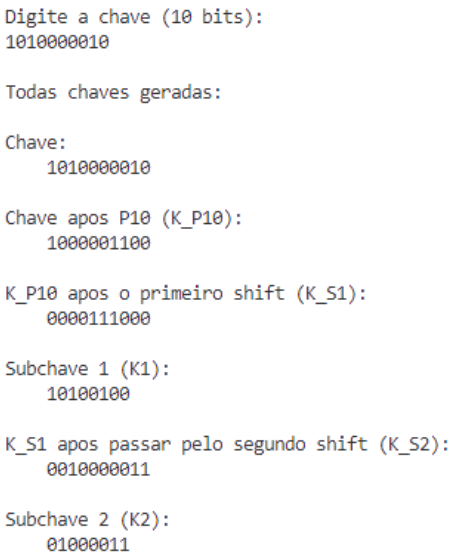

Figura 2. Teste de Geração das Chaves

1.2.1. IP e IP^{-1}

Essas funções são permutações realizadas no início e no final do processo de criptografia, respectivamente. IP^{-1} é a permutação inversa de IP, ou seja, $IP^{-1}(IP(x)) = x$. Similarmente à $P10$ e $P8$, essas funções foram implementadas com simples atribuições, e estão presentes no arquivo *src/bit_ops.cpp*, chamadas ip e $ip_inverse$ respectivamente.

1.2.2. Mapeamento F

Essa função será utilizada dentro de f_K em cada rodada de Feistel. Ela recebe uma sequência de 4 bits e uma subchave de 8 bits. Essa função segue o seguinte processo:

- A sequência de 4 bits passa pela função E/P (e_p em código), em que é feita uma expansão e permutação desse 4 bits para uma nova sequência de 8 bits, que chamaremos de N_e.
- $N_x = N_e \oplus K_x$, onde $\oplus$ é a operação binária XOR e K_x é a subchave recebida pelo mapeamento.
- N_x é dividida em duas metades, onde L são os primeiros 4 bits, e R são os últimos 4. Essas metades então são utilizadas como entradas das S-Boxes S0 e S1, onde o primeiro e último bit de cada metade são concatenadas para formar o índice da linha, e o segundo e terceiro são concatenados para formar o índice da coluna. Cada S-Box retorna um número de 2 bits.
- Os resultados das S-Boxes são concatenados (S0 + S1) e passam por uma permutação P4, que permuta esses 4 bits, assim formando o último retorno desse mapeamento.

A implementação das S-Boxes, de e_p, e de $P4$ estão presentes no arquivo *src/bit_ops.cpp*. Já a implementação final do mapeamento F está presente no arquivo *src/sdes.cpp*, na função $F\_mapping$.

1.2.3. f_K

A função f_K é uma combinação de permutações e funções de substituições, a maioria inclusa no mapeamento F. Ela gera a nova mensagem binária que irá fazer o *swap* com o lado direito da mensagem. Sua definição é:

$$f_K(L, R) = (L \oplus F(R, SK), R)$$

Onde F é o mapeamento descrito na seção 1.2.2, L são os primeiros bits da mensagem de 8 bits e R são os últimos. O retorno de f_K são duas metades novas, e nota-se que R não sofre alterações.

A implementação de f_K se encontra no arquivo *src/sdes.cpp*, na função fk.

1.2.4. SW

A função SW é responsável por trocar os dois lados de uma mensagem, de forma que o lado não alterado por f_K sofra alterações na próxima rodada do ciclo de Feistel. Nessa função, é feita a troca de L e R de posição. A implementação dessa função se encontra no arquivo *src/bit_ops.cpp*, na função sw.

1.2.5. Encriptação e Decriptação

As implementações da encriptação e decriptação foram feitas seguindo o processo da Figura 1, ou seja, combinando as funções descritas nas seções anteriores. Essas implementações se encontram no arquivo *src/sdes.cpp*, e podem ser testadas ao selecionar a opção 2 no arquivo executável do projeto, onde serão exibidos resultados parciais desses processos. Um teste de encriptação foi feito, o qual está apresentado na Figura 3.

2. Parte II - Modos de Operação

Um modo de operação é uma forma de melhorar o efeito criptográfico de uma cifra, ou possibilitar que um algoritmo seja utilizado em outra aplicação. No caso de cifras de blocos, modos de operação permitem que mensagens que possuem um tamanho maior que o bloco do algoritmo sejam criptografadas com esse mesmo algoritmo.

Nessa seção, trataremos rapidamente sobre os modos de operação implementados nesse projeto, o *Eletronic Codebook* (ECB) e *Cipher Block Chaining* (CBC). É importante mencionar que nesse trabalho não foi implementado *padding* para completar os blocos desses modos de operação, já que foi especificado que isso não seria necessário.

2.1. Eletronic Codebook

Esse modo de operação consiste em dividir a mensagem em diversos blocos de mesmo tamanho (de tamanho igual ao que o algoritmo recebe, no caso do S-DES são 8 bits), e encripta/decripta cada bloco de forma independente. Com isso, é possível paralelizar esse processo para que ele ocorra rapidamente, porém, padrões na mensagem ficam são facilmente detectados.

```
Digite a chave (10 bits):
1010000010

Digite 1 para realizar uma encriptacao, e 2 para realizar uma decriptacao.
1

Digite o texto em claro (8 bits):
11010111

Plain Text:                    11010111
Mensagem apos IP:              11011101
Primeiro fk:
        L = 1101, R = 1101
        F_mapping:
                Metade expandida = 11101011
                Metade expandida XOR 10100100 (subchave) = 01001111
                S0 retornou 11
                S1 retornou 11
                S0 + S1 = 1111
        Resultado F_mapping = 1111
        Novo L = 0010
Mensagem apos fk com k1:       00101101
Mensagem apos o switch:        11010010
        L = 1101, R = 0010
        F_mapping:
                Metade expandida = 00010100
                Metade expandida XOR 01000011 (subchave) = 01010111
                S0 retornou 01
                S1 retornou 11
                S0 + S1 = 0111
        Resultado F_mapping = 1110
        Novo L = 0011
Mensagem apos fk com k2:       00110010
Texto cifrado:
    10101000
```

Figura 3. Teste de Encriptação com o S-DES

Esse processo geralmente é indicado para mensagens menores, que não excedem muito o tamanho de bloco do algoritmo e que não sejam altamente estruturadas.

Para testar esse modo operação, no arquivo executável deve-se selecionar a operação 3 para realizar a encriptação ou decriptação de uma mensagem com esse modo de operação. O teste de encriptação com o ECB está presente na Figura 4.

2.2. Cipher Block Chaining

Diferentemente do ECB, o CBC é um modo em que a cifra de um bloco é utilizada para alterar a cifra do próximo bloco, assim eliminando o problema anterior dos padrões nas mensagens. Em contrapartida, essa dependência impossibilita que esse modo de execução seja paralelizado entre seus blocos, assim exigindo um maior tempo de processamento.

Para criar esse encadeamento, o CBC realiza um XOR do texto em plano de um bloco com o texto cifrado do bloco anterior antes de realizar a cifra desse bloco. No caso do primeiro bloco, é utilizado um Vetor de Inicialização (IV), de mesmo tamanho que o bloco, para realizar o XOR com o primeiro bloco em claro. Para realizar a decriptação, o XOR é realizado após o processo de decriptação, assim sendo possível recuperar o texto em claro.

Para executar o CBC implementado, ele está dentro da operação 4 do arquivo executável, onde será pedido o IV e o texto que se deseja cifrar/decifrar. Está presente na Figura 5 um teste de encriptação utilizando CBC.

Referências

Stallings, W. (2010). Appendix g - simplified des. http://mercury.webster.edu/aleshunas/COSC%205130/G-SDES.pdf.

```
Digite a chave (10 bits):
1010000010

Digite 1 para realizar uma encriptacao, e 2 para realizar uma decriptacao.
1

Digite o texto em claro (numero de bits multiplo de 8):
11010111 01101100 10111010 11110000

Bloco 1:
        Antes de encriptar:  11010111
        Depois de encriptar: 10101000

Bloco 2:
        Antes de encriptar:  01101100
        Depois de encriptar: 00001101

Bloco 3:
        Antes de encriptar:  10111010
        Depois de encriptar: 00101110

Bloco 4:
        Antes de encriptar:  11110000
        Depois de encriptar: 01101101

Texto cifrado:
    10101000 00001101 00101110 01101101
```

Figura 4. Teste de Encriptação com o modo de operação ECB

```
Digite a chave (10 bits):
1010000010

Digite o vetor de inicializacao (8 bits):
01010101

Digite 1 para realizar uma encriptacao, e 2 para realizar uma decriptacao.
1

Digite o texto em claro (numero de bits multiplo de 8):
11010111 01101100 10111010 11110000

|---------|---------------|-----------------------------|-----------------|---------------|
|  Bloco  | Bloco em Claro | IV ou bloco cifrado anterior | Resultado do XOR | Bloco  Cifrado |
|---------|---------------|-----------------------------|-----------------|---------------|
|1        |    11010111   |          01010101           |     10000010    |    00001011   |
|---------|---------------|-----------------------------|-----------------|---------------|
|2        |    01101100   |          00001011           |     01100111    |    10101001   |
|---------|---------------|-----------------------------|-----------------|---------------|
|3        |    10111010   |          10101001           |     00010011    |    10011011   |
|---------|---------------|-----------------------------|-----------------|---------------|
|4        |    11110000   |          10011011           |     01101011    |    01101010   |
|---------|---------------|-----------------------------|-----------------|---------------|

Texto cifrado:
    00001011 10101001 10011011 01101010
```

Figura 5. Teste de Encriptação com o modo de operação CBC