S-DES, ECB e CBC

Gabriel Henrique do Nascimento Neres
Arthur Diehl Barroso

May 17, 2025

Abstract

Texto detalhando o desenvolvimento de uma versão simplificada do algoritmo de criptografia DES, denominado de S-DES, que realiza a encriptação e decriptação utilizando dois modos de operação, o ECB e CBC. Nele é utilizado uma chave de 10 bits e uma entrada de convertida em uma lista de bytes.

Palavras-chave: Criptografia, DES, ECB, CBC.

Esse trabalho irá implementar e detalhar as etapas de confecção do algoritmo de criptografia simetria S-DES, um modelo simplificado do algoritmo DES, além de dois modos de operação (ECB e CBC). Para compreender melhor a implementação que está sendo feita, serão repassados os resultados de cada função baseado nos dados de entrada a baixo:

- Chave: 1010000010
- Mensagem: 11010111 01101100 10111010 11110000
- IV CBC: 01010101

1 S-DES

Em ambos os modos de operação que serão utilizados o mesmo algoritmo do S-DES será utilizado. Os próximos tópicos irão detalhar cada uma das etapas desenvolvidas internamente pelo algoritmo que é utilizado tanto para encriptar quanto decriptar aum bloco de 8 bits fornecido.

Para exemplificar cada um dos resultados intermediários obtidos ao longo do algoritmo, a **chave** utilizada será a mesma passada antes e o bloco de 8 bits será o primeiro da mensagem, ou seja , 11010111.

1.1 Geração de Chaves Subjacentes

No processo de geração das chaves subjacentes os 10 bits iniciais são transformados em 2 sub-chaves (k1 e k2) derivadas a partir da chave recebida. Para obter as sub-chaves são utilizadas 2 funções principais em conjunto com um array que representa as permutações e uma função que reunirá todo o processo de geração da chave.

```
SW = lambda s: permutation(s, [4, 5, 6, 7, 0, 1, 2, 3])
P10 = lambda s: permutation(s, [2, 4, 1, 6, 3, 9, 0, 8, 7, 5])
P8  = lambda s: permutation(s, [5, 2, 6, 3, 7, 4, 9, 8])
```

```
def permutation(s: str, indices: list[int]) -> str:
    return ''.join(s[i] for i in indices)
```

```
def Shift(key: str, n: int):
    fst_half = key[0:5]
    snd_half = key[5:10]

    return (fst_half[n:] + fst_half[:n] + snd_half[n:] + snd_half[:n])
```

```
def key_gen(key: str):
    intermediate_key = Shift((P10(key)), 1)

    K1 = P8(intermediate_key)
    K2 = P8(Shift(intermediate_key, 2))

    return (K1, K2)
```

Com a execução de todas essas operações, será obtido os seguintes valores a partir da chave recebida:

- Intermediaria: 0000111000
- K1: 10100100
- K2: 01000011

1.2 Permutação Inicial

Para realizar a permutação inicial, é utilizada a mesma função de permutação utilizada na geração das sub-chaves definindo o array [1, 5, 2, 0, 3, 7, 4, 6]. Ao aplicar ao bloco de entrada a permutação inicial será obtido 11011101.

1.3 Rodadas de Feistel

Para realização das rodadas de Feistel, a entrada de 8 bits recebida é dividida ao meio (L e R) e é aplicada a equação

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

e inverte o valor resultante, trocando L e R de lugar, usando a permutação [4, 5, 6, 7, 0, 1, 2, 3]. No modelo do SDES, serão realizadas apenas duas rodadas desse processo e por isso SK será, respectivamente, k1 e k2.

Das operações que serão utilizadas, falta apenas a definição da função F. Essa função tem como objetivo aplicar a **permutação de expansão**(EP) utilizando o array [3, 0, 1, 2, 1, 2, 3, 0], realizar o **XOR** entre a chave e a saída da permutação EP e por fim realizar a permutação P4 utilizando o array [1, 3, 2, 0] com os valores obtidos a partir das S-Boxes.

```
def F(R: str, SK: str):
  s = XOR(EP(R), SK)
  return P4(S0[s[0] + s[3]][s[1] + s[2]] + S1[s[4] + s[7]][s
    [5] + s[6]])
```

Listing 1: Função F

A EP é utilizada para transformar a entrada de 4 bits em uma saída de 8 bits, permitindo a realização do XOR com a chave. Já as S-Boxes estabelecidas pelo algoritmo, que estão presentes na imagem 1 e possuem valores para todas as combinações de 4 bits, permitem a conversão da entrada de 8 bits para uma saída de 4 bits que será utilizada pela permutação P4.

A fim de permitir um meio de comparar os resultados obtidos ao longo da f_k, o print do resultado das 2 rodadas de f_k está presente na imagem 2.

Figure 1: S-Boxes

```
S0 = {'00': {'00': '01', '01': '00', '10': '11', '11': '10'},
      '01': {'00': '11', '01': '10', '10': '01', '11': '00'},
      '10': {'00': '00', '01': '10', '10': '01', '11': '11'},
      '11': {'00': '11', '01': '01', '10': '11', '11': '10'}
      }

S1 = {'00': {'00': '00', '01': '01', '10': '10', '11': '11'},
      '01': {'00': '10', '01': '00', '10': '01', '11': '11'},
      '10': {'00': '11', '01': '00', '10': '01', '11': '00'},
      '11': {'00': '10', '01': '01', '10': '00', '11': '11'}
```

Figure 2: Resultados intermediários das rodadas de Feistel

```
EP(R): 11101011
XOR(EP(R), SK): 01001111
S0[00][10] + S1[11][11]
P4: 1111
Mensagem após o primeiro f_k: 00101101
Mensagem após o switch: 11010010
EP(R): 00010100
XOR(EP(R), SK): 01010111
S0[01][10] + S1[01][11]
P4: 1110
Mensagem após o segundo f k: 00110010
```

1.4 Permutação Final

Para finalizar o algoritmo, é realizada uma permutação final a qual é a inversa da permutação inicialmente. Desse modo, a permutação utilizada é a [3, 0, 2, 4, 6, 1, 7, 5] e o resultado obtido seguindo todo o algoritmo será 10101000, que também é a **mensagem encriptada** resultante do S-DES.

Como definimos agora todo o processo para encriptar a mensagem, devemos definir como será feito para decriptar o ciphertext. Dada a forma como o S-DES funciona, o processo de decriptação sera feito apenas invertendo a ordem de utilização das chaves, isto é, ao invés de k1 e k2 serão utilizadas k2 e k1 em f_k obtendo o plaintext.

2 Modos de operação

Ao iniciar o algoritmo, poderão ser utilizados dois modos de operação. Esses algoritmos são:

2.1 ECB

A técnica utilizada por esse modo de operação é bem simples, principalmente pela entrada fornecida será dividida em entradas do mesmo tamanho da utilizada pelo S-DES, ou seja, a entrada do ECB será de um conjunto de blocos de 8 bits que serão passados um a um para o algoritmo de encriptação do S-DES. Para isso, foi utilizada a função da imagem 3.

Figure 3: ECB Encript

```
def ecb_encrypt(message: str, key: str) -> list[str]:
    blocks = message.strip().split()
    return ' '.join([sdes_encryption(block, key) for block in blocks])
```

Para exemplificar, a mensagem fornecida ao programa ao passar pela função será convertida no array [11010111, 01101100, 10111010, 11110000]. Vale lembrar que esse **não** é o valor retornado pela função, pois este só será possível ao concatenar o resultado de cada bloco retornado ao utilizar o S-DES, que será 10101000 00001101 00101110 01101101.

2.2 CBC

Nesse modo de operação, além da mensagem dividida em blocos de 8 bits e da chave de 10 bits também será recebida um valor inicial (IV), que será utilizado para realizar a XOR com a mensagem a ser encriptada. Para isso foi utilizada a função:

```
1 def XOR(s1:str, s2:str):
2   return ''.join('1' if c1 != c2 else '0' for c1, c2 in zip(
      s1, s2))
```

Após a primeira iteração o valor utilizado para realizar o XOR com o bloco de mensagem será o valor retornado pela criptografia do SDES. Esse processo será repetido para todos os blocos de 8 bits da mensagem seguindo o loop da figura 4. Os resultados obtidos em cada parte do CBC foram os apresentados na figura 5.

Figure 4: Algoritmo CBC

```
for block in blocks:
    xored = XOR(block, prev)
    cipher = sdes_encryption(xored, key)
    ciphertext_blocks.append(cipher)
    prev = cipher
```

Figure 5: Resultados obtidos pelo CBC

```
1º Interação
Bloco: 11010111. Vetor usado no XOR: 01010101 -> XOR: 10000010
PlainText -> ciphertext: 11010111 -> 00001011
2º Interação
Bloco: 01101100. Vetor usado no XOR: 00001011 -> XOR: 01100111
PlainText -> ciphertext: 01101100 -> 10101001
3º Interação
Bloco: 10111010. Vetor usado no XOR: 10101001 -> XOR: 00010011
PlainText -> ciphertext: 10111010 -> 10011011
4º Interação
Bloco: 11110000. Vetor usado no XOR: 10011011 -> XOR: 01101011
PlainText -> ciphertext: 11110000 -> 01101010
00001011 10101001 10011011 01101010
```

3 Código

O código está disponível no github pelo link: https://github.com/Diehlgit/S-DES