Segurança Computacional
Trabalho 01

Eduardo Pereira de Sousa - 231018937
Luca Heringer Megiorin - 231003390
Universidade de Brasília

18 de Maio, 2025

1 Introdução

O algoritmo de encriptação DES (Data Encryption Standard), desenvolvido pela IBM, foi adotado como padrão em 1977 para as aplicações, e usa uma chave de 56 bits para encriptar blocos de 64 bits, até eventualmente ser quebrado e considerado inseguro [1]. Por ser um algoritmo mais complexo, foi-se criado um algoritmo simplificado para servir de intrumento didático, o Simplified Data Encryption Standard (S-DES), projetado pelo professor Edward Schaefer [2].

A necessidade da transferência de mensagens mais longas também se mostrou presente, pois as mensagens tendem a ser maiores que 64 bits. Para isso, foram criados diversos modos de operação, que, aliados a algoritmos de criptografia, como o DES, encriptam e decriptam mensagens mais longas, dividindo-as em blocos.

Neste documento, serão abordados os processos presentes no algoritmo S-DES (seção 2), bem como os modos de operação *Eletronic Codebook* (ECB), na seção 3, e *Cipher Block Chaining* (CBC), na seção 4. O código em python desenvolvido está disponível no GitHub (vide o apêndice A).

2 Simplified Data Encryption Standard

O S-DES é uma versão didática do DES (Data Encryption Standard) que opera em blocos de 8 bits e utiliza uma chave de 10 bits, projetada para facilitar o ensino dos conceitos de cifras de bloco, redes de Feistel, substituição e permutação de forma manual ou em simulações simples. O algoritmo consiste na geração de chaves (seção 2.1) e na encriptação/decriptação em si (seção 2.2)

2.1 Geração das chaves

Por ser uma versão simplificada do DES, o S-DES possui apenas duas rodadas de Feistel, o que implica na necessidade de apenas duas subchaves. Para a geração das subchaves, é feita uma permutação sobre os 10 bits (P10) sobre a chave original $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$:

$$P_{10}(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_1, k_9, k_8, k_6)$$

Em seguida, o resultado é dividido em duas metades de 5 bits, de forma que cada uma sofra uma rotação circular à esquerda de 1 bit (LS-1). Sendo $(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10})$ o resultado da junção entre as metades deslocadas, ele passará por uma escolha permutada de 8 bits (P8) e será a primeira subchave (K_1):

$$P_8(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}) = (p_6, p_3, p_7, p_4, p_8, p_5, p_{10}, p_9)$$

Para gerar K_2, divide-se mais uma vez em duas metades a chave que passou por LS-1, $(p_{1,2}, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10})$, e realiza-se uma rotação adicional de 2 bits (LS-2), e, com o resultado, aplica-se novamente P8, originando a segunda subchave.

O código 1 abaixo aplica os conhecimentos acima para gerar as chaves:

```
def generateKeys(key:bitarray):
    log_keys["key"] = key
    # P10 (10 bit-key permutation)
    p10_key = key[2:3] + key[4:5] + key[1:2] + key[6:7] + key[3:4] +
        key[9:10] + key[0:1] + key[8:9] + key[7:8] + key[5:6]
    log_keys["p10_key"] = p10_key
    subKeys = []
    shiftKey = p10_key
    for i in range(ROUNDS):
        # Circular Left Shift
        shiftKey = circular_left_shift(shiftKey, i)
        log_keys["leftShift"].append(shiftKey)
        # Permuted Choice (P8): 6 3 7 4 8 5 10 9
        p8_key = shiftKey[5:6] + shiftKey[2:3] + shiftKey[6:7] +
            shiftKey[3:4] + shiftKey[7:8] + shiftKey[4:5] + shiftKey
            [9:10] + shiftKey[8:9]
        subKeys.append(p8_key)
        log_keys["subKeys"].append(p8_key)
    return subKeys
```

Código 1: Função de Geração da Chave, presente no arquivo *sdes_functions.py*

2.2 Encriptação e Decriptação

O algoritmo de encriptação no S-DES recebe um bloco de 8 bits do texto em claro e a lista de subchaves, resultando no texto cifrado, também de 8 bits. O algoritmo de encriptação pode ser descrito como a composição das funções IP (seção 2.2.1), f_k (seção 2.2.2), SW (seção 2.2.3) e IP^{-1} (seção 2.2.4):

$$C = Enc(M, K_1, K_2) = IP^{-1} \circ f_{K2} \circ SW \circ f_{K1} \circ IP, \text{ ou}$$

$$C = Enc(M, K_1, K_2) = IP^{-1}(f_{K2}(SW(f_{K1}(IP(M)))))$$

De semelhante modo, a função de decriptação apenas requer que a ordem das subchaves seja invertida, então pode-se representá-la como:

$P = Dec(C, K_1, K_2) = Enc(C, K_2, K_1) = IP^{-1} \circ f_{K1} \circ SW \circ f_{K2} \circ IP$, ou

$P = Dec(C, K_1, K_2) = Enc(C, K_2, K_1) = IP^{-1}(f_{K1}(SW(f_{K2}(IP(C)))))$

Os códigos referentes a essa seção estão presentes no arquivo "'sdes_functions.py"', presentes no diretório "'helpers"' do GitHub (vide o apêndice A).

2.2.1 Permutação Inicial (IP)

Os 8 bits do texto recebido são permutados em uma ordem específica. Sejam os bits do texto recebido $(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$, então:

$IP(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8) = (p_2, p_6, p_3, p_1, p_4, p_8, p_5, p_7)$

2.2.2 Função K (F_k)

Esse componente do S-DES corresponde às rodadas de Feistel, combinando permutações e substituições. A função será executada duas vezes em uma encriptação ou decriptação, recebendo uma das duas subchaves geradas anteriormente e o resultado da Permutação Inicial. Esse resultado de IP é dividido em duas metades: da esquerda, composta pelos 4 bits mais significativos, e a da direita, composta pelos 4 bits menos significativos.

Primeiramente, a metade da direita (k_1, k_2, k_3, k_4) é utilizada em uma permutação expandida (E/P):

$E/P(k_1, k_2, k_3, k_4) = (k_4, k_1, k_2, k_3, k_2, k_3, k_4, k_1)$

Em seguida, é feito um XOR com uma das subchaves (na primeira rodada, a chave é K_1, para a encriptação ou K_2, para a decriptação) e os bits resultantes da permutação expandida. O resultado do XOR (de 8 bits), dado por $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ é dividido em duas metades de 4 bits, que serão usadas nas *Substitution Boxes* (S-Box):

$$S_0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{pmatrix} \end{array} \qquad S_1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{pmatrix} \end{array}$$

As *S-Boxes* mapeiam um conjunto de n bits de entrada para m bits de saída, utilizando uma tabela predefinida. Esse mapeamento torna a relação entre os dados de entrada e saída mais imprevisíveis e complexos. Com os 4 bits de cada metade do XOR feito anteriormente, são escolhidos os números de substituição de cada S-Box, de modo que o número gerado pela junção entre o primeiro e quarto bit (na metade da esquerda, por exemplo, x_1, x_4) representa a linha da *S-Box* e o formado pela junção entre o segundo e terceiro bit (na metade da esquerda, por exemplo, x_2, x_3) representa a coluna da *S-Box*.

O valor de saída será o valor em binário do número escolhido pela metade da esquerda aplicada à S_0 (s_{01}, s_{02}) concatenado ao binário do número escolhido pela metade da direita aplicada à S_1 (s_{11}, s_{12}). Estes valores serão combinados pela permutação $P4$:

$$P4(s_{01}, s_{02}, s_{11}, s_{12}) = (s_{02}, s_{12}, s_{11}, s_{01})$$

Por fim, a permutação P4 sofre um XOR com os 4 bits da esquerda da IP, e então a função F_k retorna o resultado do XOR concatenado com os 4 bits da direita de IP. O mesmo processo é repetido com a segunda chave (K_2, para a encriptação, ou K_1, para a decriptação) após o resultado da primeira rodada passar pela função SW.

2.2.3 Função Switch (SW)

A função Switch tem o papel de trocar os 4 bits da direita e da esquerda fornecidos pela primeira rodada da função F_k. Dessa forma, pode-se representá-la como:

$$SW(p_{1,2}\,, p_3, p_4, p_5, p_6, p_7, p_8) = (p_5, p_6, p_7, p_8, p_1, p_2, p_3, p_4)$$

2.2.4 Permutação Inversa (IP^{-1})

Por fim, após todos os passos sucedidos anteriormente, o resultado da segunda rodada da função F_k passa pela função inversa da permutação inicial IP:

$$IP^-1(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8) = (p_4, p_1, p_3, p_5, p_7, p_2, p_8, p_6)$$

O resultado dessa permutação será o resultado da encriptação/decriptação. Um log sobre a execução do algoritmo, com os passos, pode ser visto no terminal 1, presente no apêndice B.

3 Electronic Codebook Operation Mode

O modo de operação Electronic Codebook (ECB) trata cada bloco recebido de forma independente [3]. No caso do S-DES, para cada bloco de 8 bits, é aplicado separadamente o algoritmo de encriptação/decriptação para cada bloco. Essa independência torna a implementação relativamente simples, pois é possível processar vários blocos em paralelo ou testar cada um isoladamente sem manter qualquer estado entre eles. Em contrapartida, para mensagens longas, há uma maior facilidade para os criptoanalistas identificarem padrões e repetições, o que tornam-no um modo sucetível a um ataque de texto escolhido (CPA). A sua lógica principal, presente no arquivo “‘operation_modes.py“‘, é descrita no código 2 abaixo :

```
1 # ECB operation mode (enc and dec)
2 def ecb(blocks, subkeys, cryptMode = 0):
3     result=[]
4     numBlocks = len(blocks)
5     for i in range(numBlocks):
6         if cryptMode == 0:
7             result.append(sdes.sdes(subkeys,blocks[i].copy()))
```

```
        else:
            result.append(sdes.sdes(subkeys,blocks[i].copy(), 1))

    return result
```

Código 2: Lógica principal do ECB, presente no arquivo *operation_modes.py*

4 Cipher Block Chaining Operation Mode

O modo de operação Cipher Block Chaining (CBC), em contrapartida, torna a encriptação/decriptação de cada bloco de texto dependente da anterior. Essa dependência é gerada por meio da realização de um XOR entre cada bloco de texto em claro e o bloco de texto cifrado anterior antes de passar pelo algoritmo de encriptação. Um vetor de inicialização (IV), geralmente aleatório, é usado para o primeiro bloco, garantindo maior entropia e confusão no resultado final, pois essa aleatoriedade será propagada por meio da cadeia de XOR entre blocos de textos cifrados e em claro subsequentes, o que o torna mais seguro a ataques CPA, em comparação ao ECB [3].

Para o caso da decriptação, o processo inverso é feito, de modo que o um bloco de texto cifrado, após ser decriptografado, realiza um XOR com o bloco de texto cifrado anterior (ou com o IV, para o caso do primeiro bloco), voltando a ser um bloco de texto em claro. Por possuir uma dependência entre os blocos, nem a encriptação, nem a decriptação podem ser feitos de forma paralela para todos os blocos recebidos, o que pode acarretar em um escalonamento de tempo com mensagens maiores.

A lógica principal deste modo, presente no arquivo “‘operation_modes.py“‘, está descrita no código 3 abaixo :

```
# CBC operation mode (enc)
def cbc_encrypt(messageBlocks, iVector:bitarray, subkeys):
    cipherBlocks = []
    numBlocks = len(messageBlocks)
    cipherBlocks.append(sdes.sdes(subkeys, messageBlocks[0].copy() ^
        iVector))
    for i in range(1, numBlocks):
        cipherBlocks.append(sdes.sdes(subkeys, messageBlocks[i].copy()
            ^ cipherBlocks[i - 1]))
    return cipherBlocks
# CBC operation mode (dec)
def cbc_decrypt(cipherBlocks, iVector:bitarray, subkeys):
    result = []
    numBlocks = len(cipherBlocks)
    result.append(sdes.sdes(subkeys, cipherBlocks[0].copy(), 1) ^
        iVector)
    for i in range(1, numBlocks):
        result.append(sdes.sdes(subkeys, cipherBlocks[i].copy(), 1) ^
            cipherBlocks[i - 1])
    return result
```

Código 3: Lógica principal do CBC, presente no arquivo *operation_modes.py*

5 Padding

Tanto o ECB quanto o CBC são cifras de blocos que necessitam na completude destes para aplicar funções de encriptação/decriptação. Isso significa que todo texto deve ser, no caso do S-DES, múltiplo de 8 bits. Caso isso não ocorra, esses algoritmos necessitam de usar *padding*, ou seja, precisam de preencher o último bloco que está com uma quantidade de menor que 8 bits. Apesar de não ser necessário para os dados fornecidos pela professora, como visto no terminal 2 do apêndice B, foi desenvolvido um algoritmo que adiciona o preenchimento necessário, bem como um outro bloco de 8 bits ao final, indicando quantos bits foram adicionados no preenchimento.

Dessa forma, como pode ser visto nos logs terminal 4 e terminal 5, o texto cifrado é maior e é garantidamente um múltiplo de 8, enquanto que, ao decriptar, o texto em claro volta para o seu tamanho normal. Após verificar as execuções com e sem o padding, notou-se um leve crescimento no tempo de execução, porém não considera-se haver um *overhead* por existir padding, pois o tempo de adicionar um bloco a mais é constante, sendo que a única variação de tempo adicional dependeria no número de bits que são preenchidos ao final do último bloco de texo em claro.

6 Conclusão

Neste trabalho, ao implementar o S-DES e analisar seu funcionamento, foi possível compreender na prática a estrutura das Rodadas de Feistel, o papel das *S-Boxes* e o processo de geração das subchaves do algoritmo DES original. Além de permitir entender teoricamente cada etapa do algoritmo, foi visto como os modos de operação ECB e CBC impactam a segurança de um esquema de encriptação e a necessidade deles para a transmissão de mensagens mais longas. Ademais, é possível executar o código, presente no apêndice A, bem como validar os dados fornecidos pela professora no apêndice B.

Referências

[1] SIMMONS, G. *Data Encryption Standard*, 2025. Disponível em: <https://www.britannica.com/topic/Data-Encryption-Standard>. Acesso em: 16 de maio de 2025.

[2] STALLINGS, W. *Appendix G: Simplified DES, supplement to Cryptography and Network Security, Fifth Edition*, 2010.

[3] DWORKIN, M. Recommendation for Block Cipher Modes of Operation Methods and Techniques. Disponível em: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>. Acesso em: 17 de maio de 2025.

A Código do Exercício

O código foi desenvolvido em Python 3.13.0, testado em sistema operacional Windows. É necessário instalar a biblioteca “‘bitarray“‘ para o funcionamento.

- Link para repositório: https://github.com/Luke0133/Trabalho01-SegurancaComputacional.

B Logs da execução S-DES

Abaixo constam dados da simulação do algoritmo S-DES, usando o código do apêndice A e os dados de entrada fornecidos pela professora orientadora, no pdf do trabalho, presente no Aprender3. Os dados usados de entrada sempre estão referidos com um "|" antes (por exemplo, "| Key: 1010000010" indica a chave usada no algoritmo).

```
S-DES: Encryption and Decryption Mode - Project Data
| Key: 1010000010
| Plaintext: 11010111
Generating Sub-Keys:
> Key Used: 1010000010
> 10-Bit Permutation (P10): 1000001100
> Circular Left Shift (LS-1): 0000111000
> Permuted choice 1 (P8): 10100100
> Double Circular Left Shift (LS-2): 0010000011
> Permuted choice 2 (P8): 01000011
> Sub-Keys Generated: 10100100, 01000011
> Time Elapsed: 0.0552 ms
---------------------------------------
Encrypting Plaintext:
> Plaintext Used: bitarray('11010111')
> Initial Permutation: bitarray('11011101')
> Feistel Round 1: Left = 1101, Right = 1101
  >> Expansion/Permutation (E/P): 11101011
  >> XOR with SubKey (Right XOR K1): 01001111
  >> S-Boxes: S0 = 11, S1 = 11
  >> 4-Bit Permutation (P4): 1111
  >> XOR with Left (Left XOR P4): 0010
  >> Result (XOR with Left || Right): 00101101
> Switch Function (SW): 11010010
> Feistel Round 2: Left = 1101, Right = 0010
  >> Expansion/Permutation (E/P): 00010100
  >> XOR with SubKey (Right XOR K1): 01010111
  >> S-Boxes: S0 = 01, S1 = 01
  >> 4-Bit Permutation (P4): 1110
  >> XOR with Left (Left XOR P4): 0011
  >> Result (XOR with Left || Right): 00110010
> Inverse Initial Permutation (Ciphertext): 10101000
> Time Elapsed: 0.0723 ms
---------------------------------------
Decrypting Ciphertext:
> Ciphertext Used: bitarray('10101000')
> Initial Permutation: bitarray('00110010')
> Feistel Round 1: Left = 0011, Right = 0010
```

```
  >> Expansion/Permutation (E/P): 00010100
  >> XOR with SubKey (Right XOR K2): 01010111
  >> S-Boxes: S0 = 01, S1 = 01
  >> 4-Bit Permutation (P4): 1110
  >> XOR with Left (Left XOR P4): 1101
  >> Result (XOR with Left || Right): 11010010
> Switch Function (SW): 00101101
> Feistel Round 2: Left = 0010, Right = 1101
  >> Expansion/Permutation (E/P): 11101011
  >> XOR with SubKey (Right XOR K1): 01001111
  >> S-Boxes: S0 = 11, S1 = 11
  >> 4-Bit Permutation (P4): 1111
  >> XOR with Left (Left XOR P4): 1101
  >> Result (XOR with Left || Right): 11011101
> Inverse Initial Permutation (Plaintext): 11010111
> Time Elapsed: 0.0472 ms
----------------------------------------
```

Terminal 1: Exemplo de execução do código com os dados fornecidos na documentação do Aprender3.

```
OPERATION MODES FOR S-DES: Encryption and Decryption Mode - ECB,
    Project Data
| Key: 1010000010
| Plaintext: 11010111011011001011101011110000
Generating Sub-Keys:
> Key Used: 1010000010
> 10-Bit Permutation (P10): 1000001100
> Circular Left Shift (LS-1): 0000111000
> Permuted choice 1 (P8): 10100100
> Double Circular Left Shift (LS-2): 0010000011
> Permuted choice 2 (P8): 01000011
> Sub-Keys Generated: 10100100, 01000011
> Time Elapsed: 0.0371 ms
----------------------------------------
Encrypting Plaintext:
> Plaintext Used: 11010111011011001011101011110000
> Block 1: 11010111 --> Enc --> 10101000
> Block 2: 01101100 --> Enc --> 00001101
> Block 3: 10111010 --> Enc --> 00101110
> Block 4: 11110000 --> Enc --> 01101101
> Resulting Ciphertext: 10101000000011010010111001101101
> Time Elapsed: 0.1294 ms
----------------------------------------
Decrypting Ciphertext:
> Ciphertext Used: 10101000000011010010111001101101
> CipherBlock 1: 10101000 --> Dec --> 11010111
> CipherBlock 2: 00001101 --> Dec --> 01101100
> CipherBlock 3: 00101110 --> Dec --> 10111010
> CipherBlock 4: 01101101 --> Dec --> 11110000
> Resulting Plaintext: 11010111011011001011101011110000
> Time Elapsed: 0.1316 ms
----------------------------------------
```

Terminal 2: Execução do ECB com os dados fornecidos na documentação do Aprender3 (Sem Padding).

```
OPERATION MODES FOR S-DES: Encryption and Decryption Mode - CBC,
    Project Data
| Key: 1010000010
| Initialization Vector: 01010101
| Plaintext: 11010111011011001011101011110000
Generating Sub-Keys:
> Key Used: 1010000010
> 10-Bit Permutation (P10): 1000001100
> Circular Left Shift (LS-1): 0000111000
> Permuted choice 1 (P8): 10100100
> Double Circular Left Shift (LS-2): 0010000011
> Permuted choice 2 (P8): 01000011
> Sub-Keys Generated: 10100100, 01000011
> Time Elapsed: 0.0327 ms
----------------------------------------
Encrypting Plaintext:
> Plaintext Used: 11010111011011001011101011110000
> Initialization Vector Used: 01010101
> Block 1: 11010111 XOR IV (01010101)
  >> Enc --> 00001011
> Block 2: 01101100 XOR CipherBlock 1 (00001011)
  >> Enc --> 10101001
> Block 3: 10111010 XOR CipherBlock 2 (10101001)
  >> Enc --> 10011011
> Block 4: 11110000 XOR CipherBlock 3 (10011011)
  >> Enc --> 01101010
> Resulting Ciphertext: 00001011101010011001101101101010
> Time Elapsed: 0.1458 ms
----------------------------------------
Decrypting Ciphertext:
> Ciphertext Used: 00001011101010011001101101101010
> Initialization Vector Used: 01010101
> CipherBlock 1: 00001011 --> Dec --> XOR IV (01010101)
  >> Enc --> 11010111
> CipherBlock 2: 10101001 --> Dec --> 10111011
  >> XOR CipherBlock 1 (00001011) --> 01101100
> CipherBlock 3: 10011011 --> Dec --> 11010110
  >> XOR CipherBlock 2 (10101001) --> 10111010
> CipherBlock 4: 01101010 --> Dec --> 01001010
  >> XOR CipherBlock 3 (10011011) --> 11110000
> Resulting Plaintext: 11010111011011001011101011110000
> Time Elapsed: 0.1229 ms
----------------------------------------
```

Terminal 3: Execução do CBC com os dados fornecidos na documentação do Aprender3 (Sem Padding).

```
OPERATION MODES FOR S-DES: Encryption and Decryption Mode - ECB,
    Custom Data
| Key: 1010000010
| Plaintext: 11010111011011001011101011110000
Generating Sub-Keys:
> Key Used: 1010000010
> 10-Bit Permutation (P10): 1000001100
> Circular Left Shift (LS-1): 0000111000
```

```
> Permuted choice 1 (P8): 10100100
> Double Circular Left Shift (LS-2): 0010000011
> Permuted choice 2 (P8): 01000011
> Sub-Keys Generated: 10100100, 01000011
> Time Elapsed: 0.0402 ms
---------------------------------------
Encrypting Plaintext:
> Plaintext Used: 11010111011011001011101011110000
> Block 1: 11010111 --> Enc --> 10101000
> Block 2: 01101100 --> Enc --> 00001101
> Block 3: 10111010 --> Enc --> 00101110
> Block 4: 11110000 --> Enc --> 01101101
> Block 5: 00000000 --> Enc --> 11001110
> Resulting Ciphertext: 10101000000011010010111001101101110011110
> Time Elapsed: 0.1628 ms
---------------------------------------
Decrypting Ciphertext:
> Ciphertext Used: 10101000000011010010111001101101110011110
> CipherBlock 1: 10101000 --> Dec --> 11010111
> CipherBlock 2: 00001101 --> Dec --> 01101100
> CipherBlock 3: 00101110 --> Dec --> 10111010
> CipherBlock 4: 01101101 --> Dec --> 11110000
> CipherBlock 5: 11001110 --> Dec --> 00000000
> Resulting Plaintext: 11010111011011001011101011110000
> Time Elapsed: 0.1389 ms
---------------------------------------
```

Terminal 4: Execução do ECB com dados do Aprender3 (Com Padding).

```
OPERATION MODES FOR S-DES: Encryption and Decryption Mode - CBC,
    Custom Data
| Key: 1010000010
| Initialization Vector: 01010101
| Plaintext: 11010111011011001011101011110000
Generating Sub-Keys:
> Key Used: 1010000010
> 10-Bit Permutation (P10): 1000001100
> Circular Left Shift (LS-1): 0000111000
> Permuted choice 1 (P8): 10100100
> Double Circular Left Shift (LS-2): 0010000011
> Permuted choice 2 (P8): 01000011
> Sub-Keys Generated: 10100100, 01000011
> Time Elapsed: 0.0240 ms
---------------------------------------
Encrypting Plaintext:
> Plaintext Used: 11010111011011001011101011110000
> Initialization Vector Used: 01010101
> Block 1: 11010111 XOR IV (01010101)
  >> Enc --> 00001011
> Block 2: 01101100 XOR CipherBlock 1 (00001011)
  >> Enc --> 10101001
> Block 3: 10111010 XOR CipherBlock 2 (10101001)
  >> Enc --> 10011011
> Block 4: 11110000 XOR CipherBlock 3 (10011011)
  >> Enc --> 01101010
> Block 5: 00000000 XOR CipherBlock 4 (01101010)
```

```
  >> Enc --> 11111111
> Resulting Ciphertext: 0000101110101001100110110110101011111111
> Time Elapsed: 0.1636 ms
---------------------------------------
Decrypting Ciphertext:
> Ciphertext Used: 0000101110101001100110110110101011111111
> Initialization Vector Used: 01010101
> CipherBlock 1: 00001011 --> Dec --> XOR IV (01010101)
  >> Enc --> 11010111
> CipherBlock 2: 10101001 --> Dec --> 10111011
  >> XOR CipherBlock 1 (00001011) --> 01101100
> CipherBlock 3: 10011011 --> Dec --> 11010110
  >> XOR CipherBlock 2 (10101001) --> 10111010
> CipherBlock 4: 01101010 --> Dec --> 01001010
  >> XOR CipherBlock 3 (10011011) --> 11110000
> CipherBlock 5: 11111111 --> Dec --> 11110000
  >> XOR CipherBlock 4 (01101010) --> 00000000
> Resulting Plaintext: 1101011101101100101110101111000
> Time Elapsed: 0.1598 ms
---------------------------------------
```

Terminal 5: Execução do CBC com dados do Aprender3 (Com Padding).