Universidade de Brasília
Departamento de Ciência da Computação
CIC0201 - Segurança Computacional, Turma 02
Professora: Lorena Borges

Trabalho de Implementação 1
S-DES

Iasmim de Queiroz Freitas - 190108665
Lucas Issamu Hashimoto - 231003504

2025/01

Sumário

1 Introdução

O S-DES é uma versão simplificada do algoritmo DES, criado para fins didáticos para facilitar o entendimento dos conceitos básicos da criptografia simétrica com chave secreta. Trata-se de uma cifra de bloco que opera com chaves e blocos reduzidos, uma chave de 10 bits e blocos de 8 bits, e realiza apenas duas rodadas da rede de Feistel. Por ser simétrico, o mesmo algoritmo e chave são usados tanto para criptografar quanto para descriptografar os dados.

Apesar de manter a estrutura básica do DES, o S-DES possui limitações significativas em termos de segurança. Sua chave curta (apenas 1024 possibilidades) torna-o vulnerável a ataques de força bruta, onde um adversário poderia testar todas as combinações em tempo reduzido. Portanto, seu uso é restrito a contextos educacionais, servindo como ferramenta para entender processos como geração de subchaves, substituições por S-Boxes e modos de operação.

Neste trabalho vamos implementar o algoritmo S-DES, demonstrando o seu funcionamento na encriptação e decriptação de blocos de 8 bits, utilizando uma chave de 10 bits, e também aplicaremos os modos de operação ECB e CBC.

2 Funcionamento do S-DES

O S-DES realiza a cifragem em várias etapas, sendo elas:

2.1 Geração de Subchaves (K1, K2)

A partir da chave de 10 bits, são gerados duas subchaves **K1** e **K2**. Nesse processo, ocorre a permutação P10 que reorganiza os 10 bits da chave com base na tabela ([3,5,2,7,4,10,1,9,8,6]). Em seguida, ocorre o deslocamento circular que divide a chave em duas metades e aplica um deslocamento circular. Após isso, é feita a permutação P8, que seleciona e reorganiza 8 bits para formar **K1**, e novamente faz uma nova rotação, dessa vez com um deslocamento circular duplo, e P8 para gerar **K2**.

2.1.1 Implementação

A lógica foi implementada com três funções essenciais:

- `permutacao(bits, vetor_p, n_bits=10)` (1): reordena os bits baseado em um vetor de permutação (P10/P8).
- `deslocamento_circular(bits, tamanho, shift)` (2): executa deslocamentos circulares (LS-1/LS-2). A função recebe os bits como um inteiro, o tamanho dos bits que devem ser considerados no deslocamento, e o tamanho do deslocamento circular. Restrição: `tamanho` deve ser 5 e `shift` deve ser 1 para **K1** e 2 para **K2**.
- `gera_subchaves(chave)` (3): coordena o fluxo completo da geração de chave:

```
Chave Original (10 bits)
       ↓ P10
Chave Permutada
```

```
↓ LS-1 + P8 → K1
↓ LS-2 + P8 → K2
```

- Parâmetros de saída: `K1` e `K2` são atualizados por referência (tipo `int&` em C++).

Observação: Os códigos completos estão no **Anexo A**.

2.1.2 Resultados

A geração das subchaves para a chave `1010000010` foi testada e a seguir, é apresentado o resultado passo a passo, capturado do terminal:

Etapas ilustradas na Figura 1:

```
Geração de Subchaves (Chave: 1010000010)
--------------------------------------
1. Chave original:   1010000010
2. Após P10:         1000001100
3. Após LS-1:        L=00001, R=11000
4. Subchave K1:      10100100 (via P8)
5. Após LS-2:        L=00100, R=00011
6. Subchave K2:      01000011 (via P8)
```

2.2 Encriptação

O processo de encriptação opera sobre um bloco de 8 bits que passa pelas etapas:

- **Permutação Inicial (IP)**: reorganiza os bits do bloco com base na tabela IP = [2,6,3,1,4,8,5,7]
- **Divisão em Metades**: divide o bloco de 8 bits em: L (4 bits mais significativos) e R (4 bits menos significativos)
- **Rodadas de Feistel**: são executadas duas rodadas principais e em cada uma delas é aplicado:
  - Uma função não-linear na metade direita e é combinada com a subchave da rodada (na primeira rodada k1 e na segunda rodada k2). Nesta função ocorre permutações, S-Boxes (substituições) e uma permutação P4.
  - Em seguida, com o resultado da função anterior é feito uma combinação com a metade esquerda usando XOR.
  - Então as metades esquerda e direita são trocadas no final da primeira rodada e depois ocorre os mesmos passos anteriores com a subchave k2.
- Permutação Final (IP^{-1}): concluído as rodadas de Feistel, é aplicada uma permutação final com a tabela inversa IP^{-1} = [4,1,3,5,7,2,8,6], para obter o bloco cifrado.

2.2.1 Implementação

O processo de encriptação foi dividido em 7 funções principais, organizadas em três módulos:

1. Funções de Permutação e Divisão
   - `permutacaoInicial(n)` (4): Chama a função de permutação genérica (1), aplicando a tabela IP = [2,6,3,1,4,8,5,7] no bloco de 8 bits.
   - `permutacaoFinal(n)` (5): Também chama a função de permutação genérica (1), aplicando a permutação inversa IP^{-1} = [4,1,3,5,7,2,8,6].
   - `dividi(n)` (6): Divide o bloco em metades L (4 MSB) e R (4 LSB) usando operações bitwise.
2. Funções da Rede de Feistel
   - `s_box(n, s)` (7): Realiza substituições não-lineares usando as S-Boxes S0 e S1. Linha: bits 1 e 4 de `n`. Coluna: bits 2 e 3. Retorna `s[linha][coluna]` (2 bits). Exemplo: `0b1010` → `s[2][1]`.
   - `F(n, k)` (8): Combina expansão (EP), XOR com subchave, S-Boxes e permutação P4.
   - `feistel(l, r, k1, k2)` (9): Executa 2 rodadas, chamando `F(n, k)` e aplicando XOR K1 (primeira rodada) ou K2 (segunda rodada) (com troca de metades após a primeira).
3. Função de Encriptação
   - `encriptacao(n, k)` (10): Executa o S-DES completo. Gera K1/K2 (inverte se decriptando), aplica IP, Feistel(K1,K2), e IP^{-1}. Retorna bloco processado.

2.2.2 Resultados

A encriptação do bloco `11010111` para a chave `1010000010` foi validada e abaixo, é apresentado o resultado passo a passo, capturado do terminal:

Etapas ilustradas na Figura 2 e Figura 3:

```
S-DES - Encriptação
----------------------------------------
Entrada: Bloco=11010111, Chave=1010000010
Subchaves: K1=10100100, K2=01000011

1. Permutação Inicial (IP): 11010111 → 11011101
2. Divisão: L=1101, R=1101
3. Rodadas Feistel:
   - Rodada 1 (K1):
     E/P(R)=11101011  K1=01001111 → S-Boxes=1111 → P4=1111
     L  F=0010 → Troca: L=1101, R=0010
   - Rodada 2 (K2):
     E/P(R)=00010100  K2=01010111 → S-Boxes=0111 → P4=1110
     L  F=0011 → Final: L=0011, R=0010
4. Permutação Final (IP¹): 00110010 → Cifra=10101000
```

2.3 Decriptação

O processo de decriptação no S-DES é simétrico ao de encriptação, pois também se baseia na rede de Feistel. A única diferença está na ordem de aplicação das subchaves: durante a decriptação, aplica-se primeiro a subchave **K2**, seguida da **K1**. Isso é suficiente para reverter todo o processo de encriptação e recuperar o texto original.

2.3.1 Implementação

A implementação da decriptação 11 reutiliza a função `encriptacao()` com uma flag de controle chamada `encriptar`. Quando essa flag assume o valor 1, a ordem das subchaves é invertida no interior da função de encriptação.

2.3.2 Resultados

A seguir está o resultado obtido ao executar a função de decriptação sobre o bloco previamente cifrado `10101000` com a mesma chave `1010000010`:

Etapas ilustradas na Figura 4 e Figura 5:

```
S-DES - Decriptação
---------------------------------------
Entrada: Bloco=10101000, Chave=1010000010
Subchaves: K1=01000011, K2=10100100

1. Permutação Inicial (IP): 10101000 → 00110010
2. Divisão: L=0011, R=0010
3. Rodadas Feistel:
   - Rodada 1 (K2):
     E/P(R)=00010100  K2=00010100 → S-Boxes=0111 → P4=1110
     L  F=1101 → Troca: L=0010, R=1101
   - Rodada 2 (K1):
     E/P(R)=11101011  K1=11101011 → S-Boxes=1111 → P4=1111
     L  F=1101 → Final: L=1101, R=1101
4. Permutação Final (IP¹): 11011101 → Plaintext: 11010111
```

O resultado esperado era o bloco original do trabalho `11010111`, que foi o resultado obtido pela função, mostrando que a função de decriptação reverte corretamente o processo de encriptação, recuperando o bloco original de entrada.

3 Modos de Operação

Os modos de operação são responsáveis como organizar blocos de dados são processados sequencialmente por um algoritmo de cifra de bloco como o S-DES. Este trabalho implementa dois modos clássicos: **ECB (Electronic Codebook)** e **CBC (Cipher Block Chaining)**. Ambos operam sobre blocos de 8 bits, conforme a especificação do S-DES.

3.1 ECB - Electronic Codebook

No ECB, cada bloco da mensagem é criptografado separadamente utilizando a mesma chave. Isso significa que blocos de texto iguais sempre resultarão em blocos cifrados iguais, uma característica que torna esse modo de operação mais vulnerável a certos tipos de análise. Esse modo permite a criptografia de mensagens maiores do que o tamanho do bloco do algoritmo ao dividir a mensagem em blocos independentes.

Vantagens: simples de implementar e paralelizável.

Desvantagens: padrões na mensagem original podem ser revelados, tornando-o inseguro para muitos usos práticos.

3.1.1 Implementação

A implementação do ECB percorre a mensagem dividida em blocos, aplicando a função `encriptacao()` [12] ou `decriptacao()` [13 em cada bloco individualmente.

3.1.2 Resultados

Aqui está o resultado da encriptação e decriptação da chave usando o modo de operação ECB, com a mensagem e chave especificadas no roteiro deste trabalho:

Etapas ilustradas respectivamente nas Figura 6 e Figura 7:

```
ECB Encrypt (Key:1010000010)
----------------------------
Plaintext:  11010111 01101100 10111010 11110000
Ciphertext: 10101000 00001101 00101110 01101101

[1] 11010111 → 10101000  [2] 01101100 → 00001101
[3] 10111010 → 00101110  [4] 11110000 → 01101101

ECB Decrypt (Key:1010000010)
----------------------------
Ciphertext:  10101000 00001101 00101110 01101101
Plaintext: 11010111 01101100 10111010 11110000

[1] 10101000 → 11010111  [2] 00001101 → 01101100
[3] 00101110 → 10111010  [4] 01101101 → 11110000
```

Podemos observar que a decriptação recuperou corretamente o texto original, obtendo o resultado esperado.

3.2 CBC - Cipher Block Chaining

No modo CBC, o processo é mais seguro, pois cada bloco de entrada é combinado com o bloco cifrado anterior antes da encriptação. Para o primeiro bloco, utiliza-se um vetor de inicialização (IV). Isso garante que blocos idênticos de entrada resultem em blocos diferentes de saída, desde que o IV seja diferente.

Vantagens: oculta padrões na mensagem original, mais seguro que ECB.

Desvantagens: não permite paralelização na cifragem, exige IV confiável.

3.2.1 Implementação

Durante a cifragem [14], cada bloco é combinado com o bloco cifrado anterior via operação XOR antes de ser cifrado. Durante a decifragem [15], o bloco atual é decifrado e é combinado com o bloco anterior cifrado (ou IV) via XOR.

3.2.2 Resultados

Aqui está o resultado da encriptação e decriptação da chave usando o modo de operação ECB, com a mensagem, chave e vetor de inicialização especificados no roteiro deste trabalho: **Etapas ilustradas respectivamente nas Figura 8 e Figura 9:**

```
CBC Enc (K:1010000010, IV:01010101)
-----------------------------------
Bloco 1: (1101011101010101)→00001011
Bloco 2: (0110110000001011)→10101001
Bloco 3: (1011101010101001)→10011011
Bloco 4: (1111000010011011)→01101010
Saída: 00001011 10101001 10011011 01101010

CBC Dec (K:1010000010, IV:01010101)
-----------------------------------
1: 00001011→1000001001010101→11010111
2: 10101001→0110011100001011→01101100
3: 10011011→0001001110101001→10111010
4: 01101010→0110101110011011→11110000
Saída: 11010111 01101100 10111010 11110000
```

4 Conclusão

Embora o modo de operação CBC seja mais seguro que o ECB, ambos terão praticamente o mesmo tempo ao realizar um ataque de força bruta à chave. Isso ocorre porque depende da complexidade da chave e não do modo de operação, o ataque de força bruta vai testar todas as combinações possíveis, mas o números das combinações depende do tamanho da chave, neste caso, 1024 possibilidades. Vale ressaltar que no caso da tentativa de quebra da descriptografia por força bruta, ambos os modos de operações podem ser descriptografados de forma paralela, já que no CBC, para a descriptografia, o bloco a ser descriptografado só depende do ciphertext e do IV, ou seja, não há a necessidade de esperar o processamento do bloco anterior para processar o bloco atual.

Em suma, vimos o funcionamento do DES simplificado (como os processos de geração de chaves, S-Boxes e modos de operação) e também suas limitações devido à ataques de força bruta. Quanto ao modo de operação, percebemos que o CBC acaba sendo mais seguro, pois oculta padrões na mensagem original, já que no ECB o bloco da mensagem é criptografado separadamente usando a mesma chave, o que resulta em blocos cifrados iguais se tiverem blocos de textos iguais.

O código-fonte completo deste trabalho está disponível no repositório público do GitHub:

`https://github.com/hashimotz07/S-DES`

A Códigos Implementados

A.1 Permutação

Para permutação foi feito um código genérico que pode ser utilizado em todas as funções que utilizam permutações.

```
int permuta(int n, vector<int> &v, int total_bits = 10){
    int resultado = 0;
    for(int i = 0; i < (int)v.size(); i++){
        resultado = resultado << 1;
        resultado |= (n >> (total_bits - v[i])) & 1;
    }
    return resultado;
}
```

Listing 1: Função de permutação

A.2 Geração de Subchaves

```
int deslocamento_circular(int n, int tamanho, int shift){
    return ((n<<shift) | ((n>>(tamanho - shift)) & ((1 << shift)
       -1))) & ((1<<tamanho)-1);
}
```

Listing 2: Função de deslocamento circular

```
void gerarChaves(int chave, int &k1, int &k2){
    vector<int> p10 = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
    vector<int> p8 = {6, 3, 7, 4, 8, 5, 10, 9};

    // permutacao inicial (P10)
    int permutacao = permuta(chave, p10);
    int esquerda = (permutacao>>5); // 5 bits mais significativos
    int direita = (permutacao); // 5 bits menos significativos

    // geracao da k1 (shift de 1 posicao + P8)
    esquerda = deslocamento_circular(esquerda, 5, 1);
    direita = deslocamento_circular(direita, 5 , 1);
    k1 = permuta((esquerda << 5) | direita, p8);

    // geracao da k2 (shift de 2 posicoes + p8)
    esquerda = deslocamento_circular(esquerda, 5, 2);
    direita = deslocamento_circular(direita, 5 , 2);
    k2 = permuta((esquerda << 5) | direita, p8);
}
```

Listing 3: Função de geração de subchaves

A.3 Encriptação

A.3.1 Permutação e Divisão

```
int permutacaoInicial(int n){
    return permuta(n, ip, 8);
}
```

Listing 4: Função da permutação inicial - IP

```
int permutacaoFinal(int n){
    return permuta(n, ipi, 8);
}
```

Listing 5: Função da permutação final - IP^{-1}

```
pair<int, int> dividi(int n){
    int l = n >> 4;
    int r = n & ((1 << 4) - 1);
    return make_pair(l, r);
}
```

Listing 6: Função de dividir bloco de 8 bits

A.3.2 Funções da Rede de Feistel

```
int s_box(int n, vector<vector<int>>& s){
    int linha = ((n & 8) >> 2) | (n & 1);
    int coluna = ((n & 4) >> 1) | ((n & 2) >> 1);

    return s[linha][coluna];
}
```

Listing 7: Função de substituições usando S-Box

```
int F(int n, int k){
    n = permuta(n, ep, 4);
    n ^= k;
    int l = s_box(n >> 4, s0);
    int r = s_box(n, s1);
    n = permuta(((l << 2) | r), p4, 4);
    return n;
}
```

Listing 8: Função F (Expansão + S-Box + Permutação)

```
int feistel(int l, int r, int k1, int k2){
    l = F(r, k1) ^ l;
    swap(l, r);
    l = F(r, k2) ^ l;
    return ((l << 4) | r);
}
```

Listing 9: Função da Rede de Feistel completa (2 rodadas)

A.3.3 Função de encriptação

```
int encriptacao(int n, int k){
    int k1, k2;
    gerarChaves(k, k1, k2);
    if(encriptar&1) swap(k1, k2);

    n = permutacaoInicial(n);
    auto [l, r] = dividi(n);
    n = feistel(l, r, k1, k2);
    n = permutacaoFinal(n);

    return n;
}
```

Listing 10: Função de Encriptação

A.4 Decriptação

```
int decriptacao(int n, int k){
    encriptar = 1;
    n = encriptacao(n, k);
    encriptar = 2;
    return n;
}
```

Listing 11: Função de Decriptação

A.5 Modos de Operação

A.5.1 ECB - Eletronic Codebook

```
string ECB_encripta(vector<string>& plainText, int chave){
    string cipherText="";
    for(string blocoS : plainText){
        int bloco = toInt(blocoS, 8);
        cipherText += toBin(encriptacao(bloco, chave), 8) + "␣";
    }
    return cipherText;
}
```

Listing 12: Função de encriptação S-DES usando o modo ECB

```
string ECB_decripta(vector<string>& cipherText, int chave){
    string plainText="";
    for(string blocoS : cipherText){
        int bloco = toInt(blocoS, 8);
        plainText += toBin(decriptacao(bloco, chave), 8) + "␣";
    }
    return plainText;
}
```

Listing 13: Função de decriptação S-DES usando o modo ECB

A.5.2 CBC - Cipher Block Chain

```
string CBC_encripta(vector<string>& plainText, int chave, int IV)
    {
     string cipherText="";
     int last = IV;
     for(string blocoS : plainText){
         last = toInt(blocoS, 8) ^ last;
         last = encriptacao(last, chave);
         cipherText += toBin(last, 8) + "␣";
     }
     return cipherText;
}
```

Listing 14: Função de encriptação S-DES usando o modo CBC

```
string CBC_decripta(vector<string> cipherText, int chave, int IV)
    {
     int last, atual, bloco;
     string plainText="";
     last = IV;
     for(int i=0; i < (int)cipherText.size(); i++){
         atual = toInt(cipherText[i], 8);
         bloco = decriptacao(atual, chave) ^ last;
         last = atual;
         plainText += toBin(bloco, 8) + "␣";
     }
     return plainText;
}
```

Listing 15: Função de decriptação S-DES usando o modo CBC

B Resultados

B.1 Geração de Subchaves

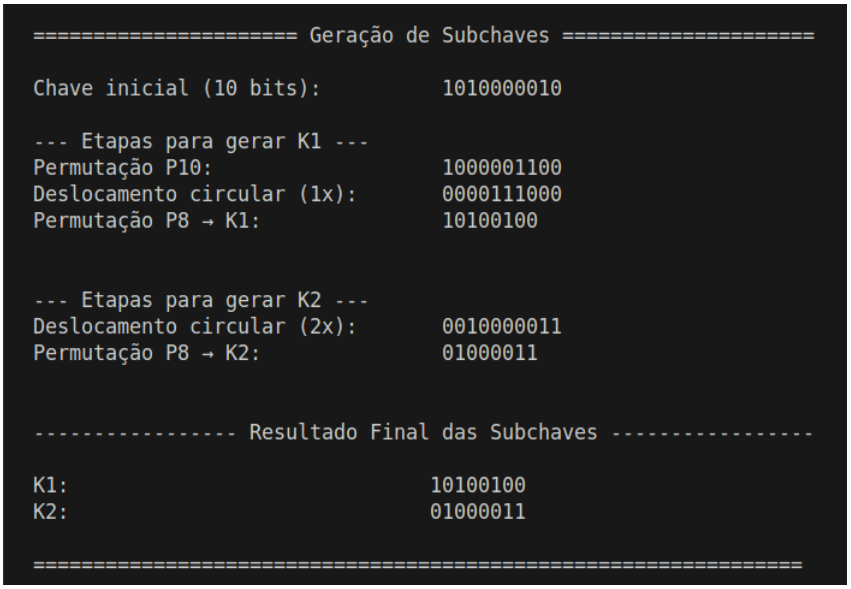

Figura 1: Processo de geração de subchaves no terminal.

B.2 Encriptação - (passo a passo)

```
====================== S-DES - Encriptação ======================

Bloco de entrada:       11010111
Chave de 10 bits:       1010000010

Subchaves geradas:
  K1 = 10100100
  K2 = 01000011

----------------------------------------------------------------
[1] Permutação Inicial (IP)

  Entrada:              11010111
  Após IP:              11011101

----------------------------------------------------------------
[2] Divisão em metades

  L (4 bits):           1101
  R (4 bits):           1101

----------------------------------------------------------------
[3] Rodadas de Feistel


=> Primeira Rodada (com K1)

  Expansão/Permutação de R:
    R =                 1101
    E/P(R):             11101011

  XOR com K1:
    K1:                 10100100
    Resultado:          11101011

  S-Boxes:
    S0(0010) -> 11
    S1(1111) -> 11
    Saída:              1111
    P4:                 1111

  XOR com L:
    L =                 1101
    L ⊕ F =             0010

  Resultado da rodada:
    L =                 0010
    R =                 1101
```

Figura 2: Encriptação: Geração de subchaves, IP, divisão em metades e 1ª rodada de Feistel

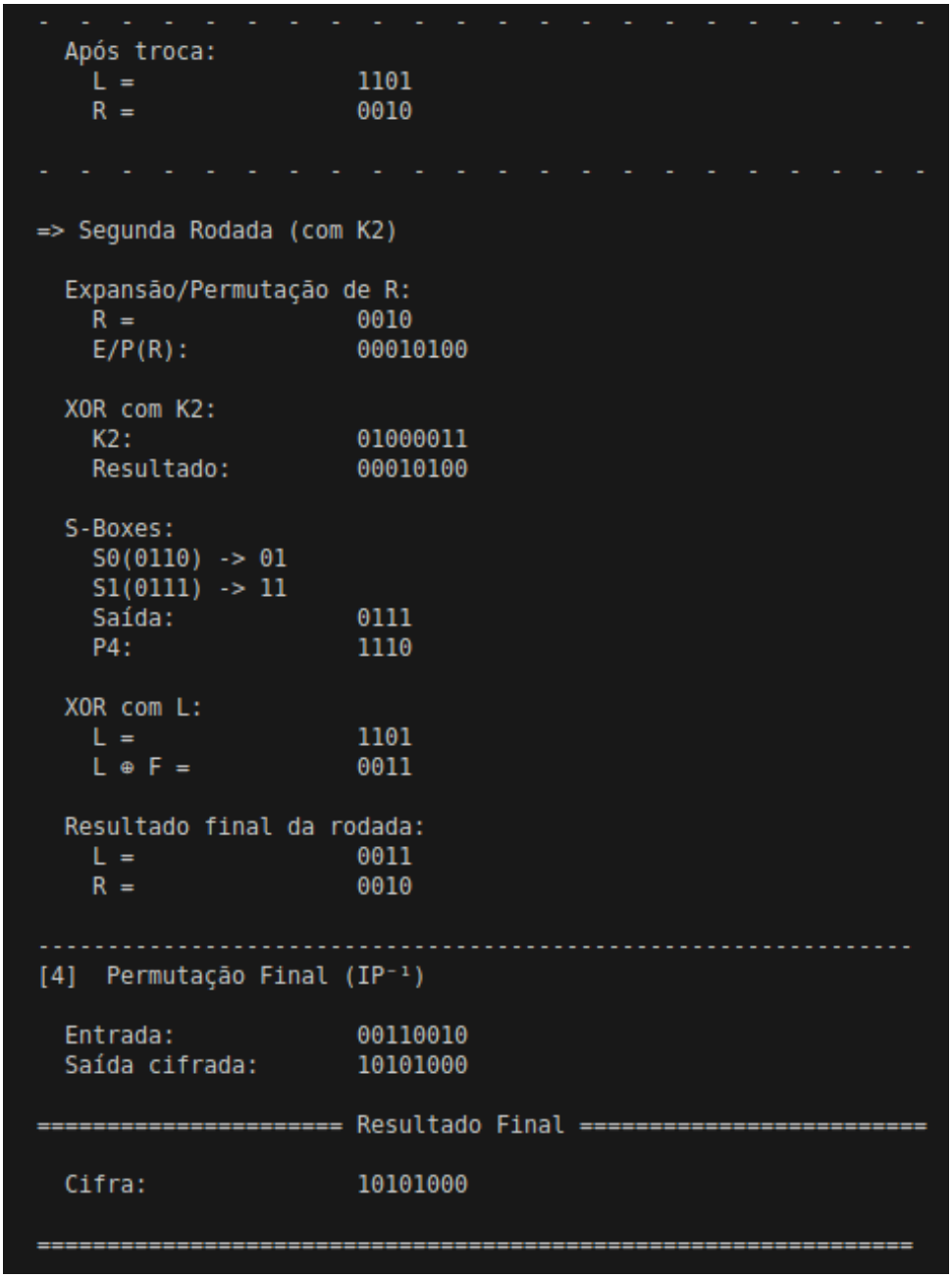

Figura 3: Encriptação: Inversão de L e R, 2ª rodada de feistel e IP^{-1}

B.3 Decriptação - (passo a passo)

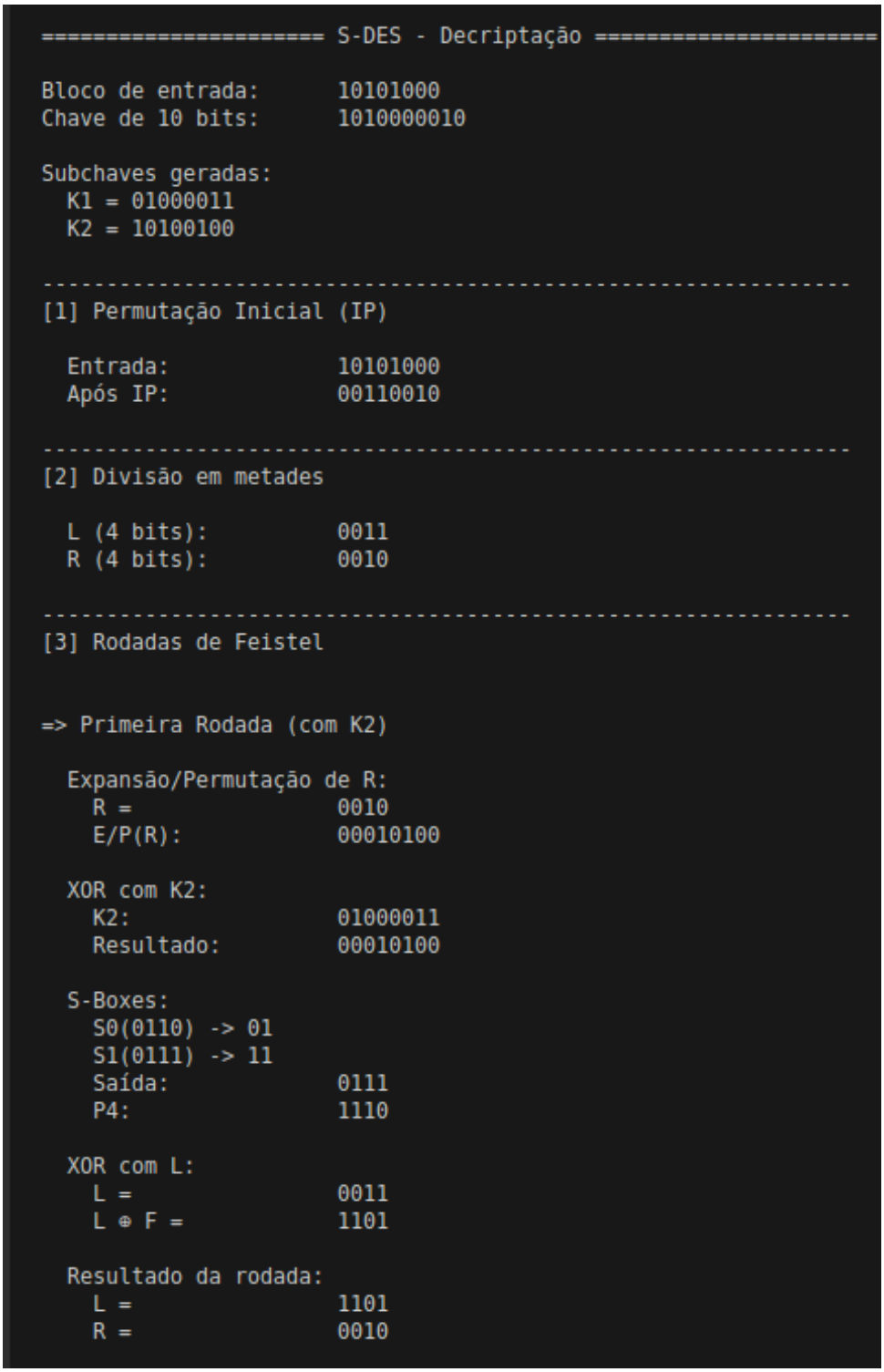

Figura 4: Decriptação: Geração de subchaves, IP, divisão em metades e 1ª rodada de Feistel

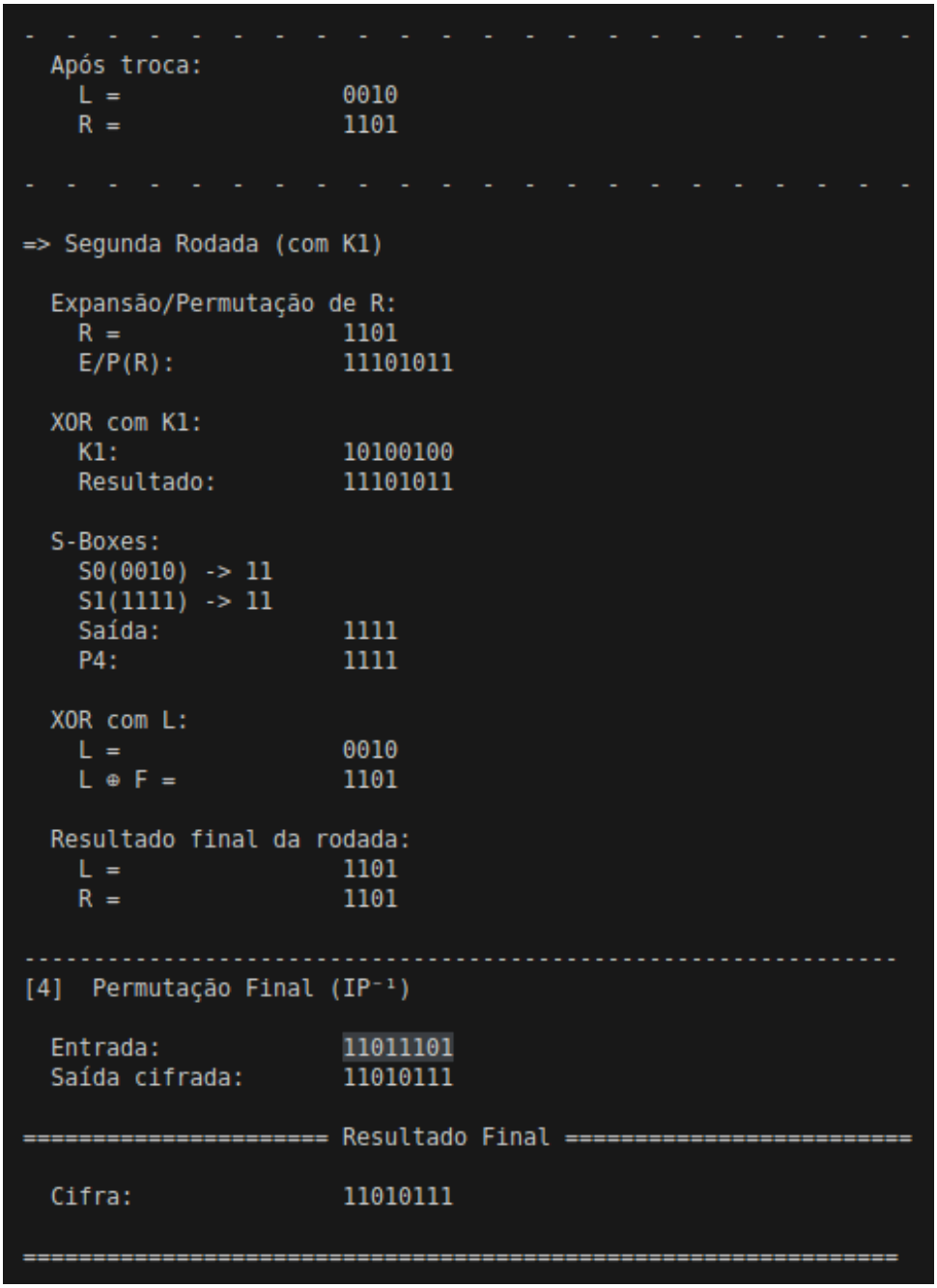

Figura 5: Decriptação: Inversão de L e R, 2ª rodada de feistel e IP^{-1}

B.4 ECB - Eletronic Codebook

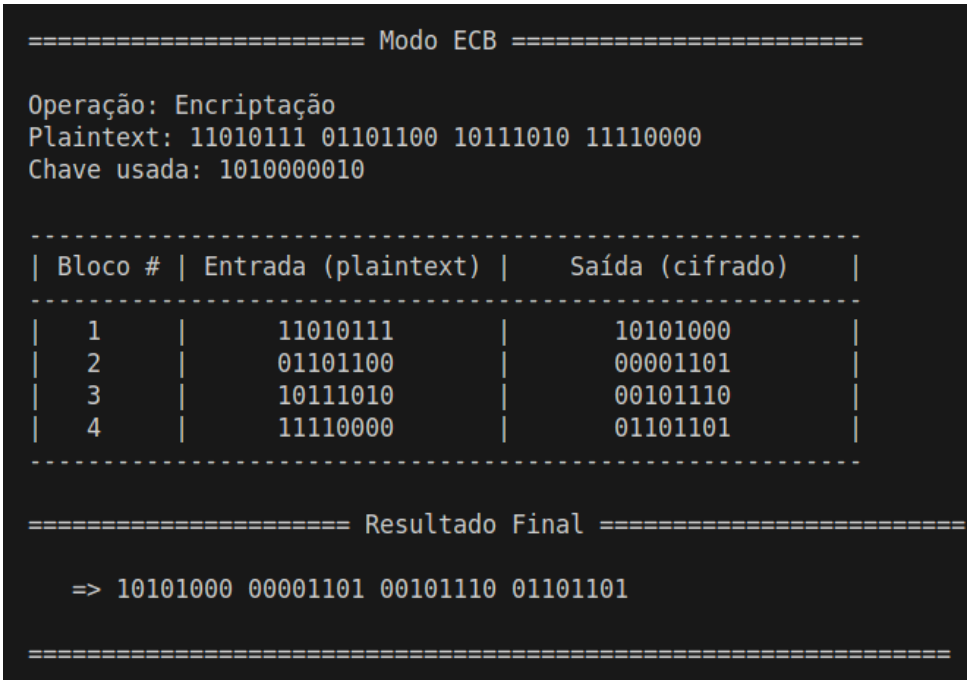

Figura 6: Teste encriptação de cifra de bloco usando S-DES - Modo ECB

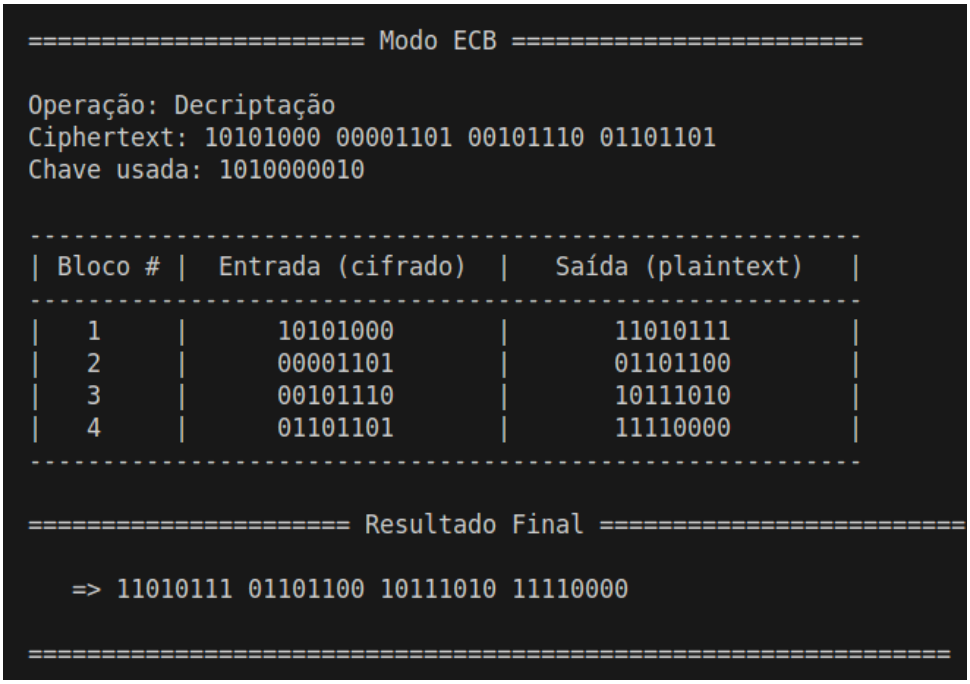

Figura 7: Teste decriptação de cifra de bloco usando S-DES - Modo ECB

B.5 CBC - Cipher Block Chain

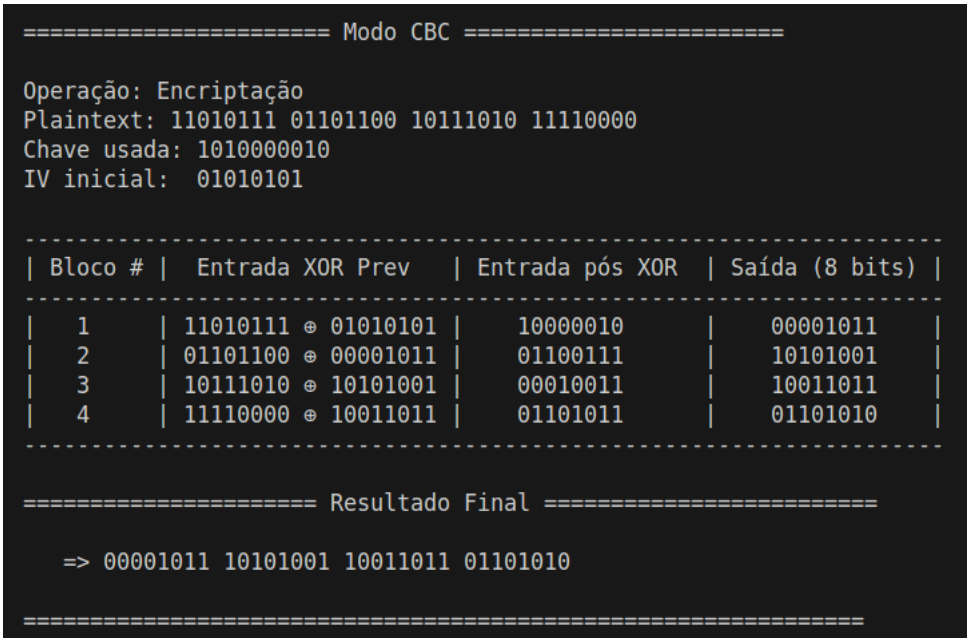

Figura 8: Teste encriptação de cifra de bloco usando S-DES - Modo CBC

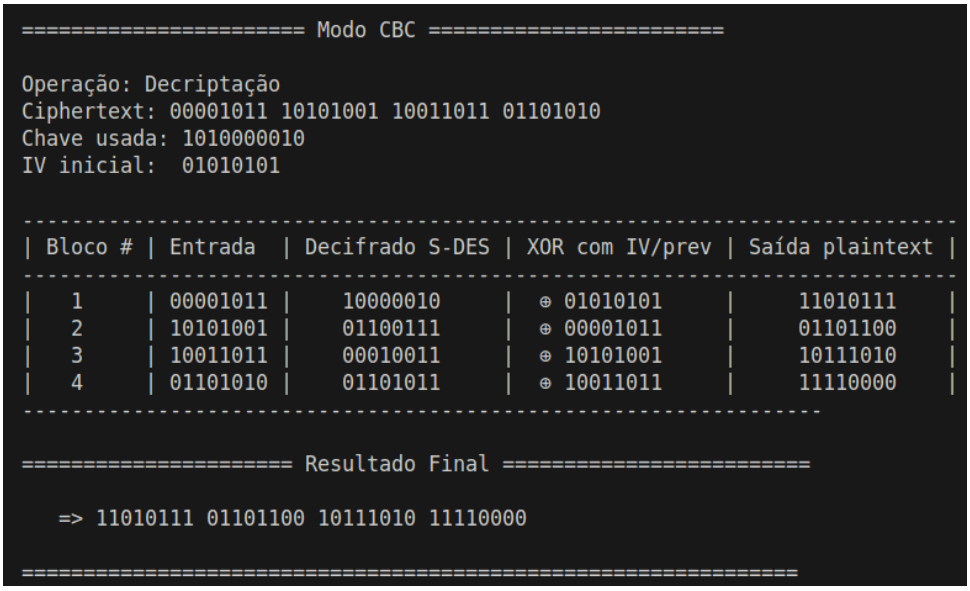

Figura 9: Teste decriptação de cifra de bloco usando S-DES - Modo CBC