Simplified - DES

December 18, 2024

1 Simplified - DES

O S-DES (Simplified Data Encryption Standard) é um algoritmo de criptografia simétrica utilizado exclusivamente para fins educacionais, com o objetivo de proporcionar entendimento sobre o algoritmo DES. O S-DES é uma versão reduzida e simplificada do DES, com uma chave de 10 bits e blocos de 8 bits, executando também apenas duas passagens pela função FK, que é uma versão simplificada da função do DES, principal responsável pela difusão da mensagem original. A difusão ocorre por meio de permutações, deslocamentos circulares, substituições e expansões para garantir a confidencialidade e a integridade das informações.

Neste notebook, faremos uma implementação do S-DES, seguindo as diretrizes do professor Edward Schaefer da Universidade de Santa Clara.

1.1 Geração das sub chaves K1 e K2

O S-DES é um algoritmo de criptografia simétrica, assim, teremos apenas uma chave privada, a qual será utilizada tanto no processo de criptografia quanto no processo de descriptografia. Esta chave possui 10 bits e será utilizada como entrada para a geração de duas subchaves de 8 bits, K1 e K2. Cada subchave será empregada apenas uma vez como uma das entradas na função F de difusão.

Para geração destas subchaves, é aplicado as seguintes funções na respectiva ordem:

1.1.1 Função de Permutação P10

Uma função simples de permutação que embaralha o valor da nossa chave privada de forma determinística.

[533]:
```
def P_10(key):
    """
    Performs the P10 permutation on a 10-bit key.
    """
    return [key[2], key[4], key[1], key[6], key[3], key[9], key[0], key[8],␣
↪key[7], key[5]]
```

Será a primeira etapa da geração das nossas subchaves.

1.1.2 Função para o Deslocamento Circular a Esquerda

Implementação básica de uma função de deslocamento circular a esquerda.

[534]:
```
def circular_left_shift(key):
    """
    Performs a circular left shift (LS-1) on both halves of a 10-bit key.
    """
    left  = key[:len(key)//2]
    right = key[len(key)//2:]

    left = left[1:] + left[:1]
    right = right[1:] + right[:1]

    return left+right
```

Após passar pela função P_1O, será feito uma rodada nesta função.

1.1.3 Função de Permutação P8

Uma função simples de permutação que embaralha o valor da nossa chave privada de forma determinística e reduzindo o tamanho da chave em dois bits.

[535]:
```
def P_8(key):
    """
    Performs the P8 permutation on a 10-bit key.
    """
    return [key[5], key[2], key[6], key[3], key[7], key[4], key[9], key[8]]
```

Após passar por uma rodada da circular_left_shift, esta função será aplicada e o seu resultado será o valor da primeira subchave K1.

Para geração da subchave K2, será aplicado mais duas rodadas da função circular_left_shift e depois aplicado novamente a função P_8, assim, o seu resultado será o valor da segunda subchave K2.

1.1.4 Executando as Funções para Geração de K1 e K2

Aqui temos a função s_des_generation_keys que implementa a ordem de execução das funções descritas acima.

[536]:
```
def s_des_generation_keys(key):
    """
    Generates the K1 and K2 keys for the S-DES encryption algorithm.
    """
    key_P10 = P_10(key)
    key_P10 = circular_left_shift(key_P10)
    k1 = P_8(key_P10)

    key_P10 = circular_left_shift(key_P10)
    key_P10 = circular_left_shift(key_P10)
    k2 = P_8(key_P10)
```

```
    return k1, k2
```

Para testarmos a operação desse algoritmo usaremos como chave o valor: 1010000010 e para o valor do bloco será utilizado: 11010111

[537]:
```
# Original 10-bit key
key = ["1","0","1","0","0","0","0","0","1","0"]
# Original 8-bit block
block = ["1","1","0","1","0","1","1","1"]
# The two subkeys generated
k1, k2 = s_des_generation_keys(key)

print(f'Key K1: {k1}\nKey K2: {k2}')
```

```
Key K1: ['1', '0', '1', '0', '0', '1', '0', '0']
Key K2: ['0', '1', '0', '0', '0', '0', '1', '1']
```

Após rodar o bloco de código acima, o resultado deverá ser:

Key K1: [‘1’, ‘0’, ‘1’, ‘0’, ‘0’, ‘1’, ‘0’, ‘0’]

Key K2: [‘0’, ‘1’, ‘0’, ‘0’, ‘0’, ‘0’, ‘1’, ‘1’]

Como nossa função de geração de subchave é totalmente deterministica, como deve ser,o resultado sempre deve ser este, caso não tenha sido esse caso, rode novamento todos os blocos acima para evitar algum problema de falta de declaração.

1.2 Aplicação da Criptografia e Descriptografia

Com as subchaves K1 e K2 geradas, seguimos para a etapa de criptografia e descriptografia da nossa implementação do algoritmo S-DES.

A criptografia é realizada pela sequência de funções: FP(Fk(k2, SW(Fk(k1, IP(block))))). Aqui, aplicamos as operações de permutação inicial (IP), seguida por Fk com K1 para realizar as operações de difusão. Depois, usamos a função SW para inverter as metades e aplicando Fk com K2. No final, uma permutação final (FP) reorganiza os bits da mensagem cifrada, terminando o processo de criptografia.

A descriptografia é similar, mas com a ordem das subchaves invertida: FP(Fk(k1, SW(Fk(k2, IP(cipher_text))))). A ordem das operações é a mesma, apenas aplicando Fk com K2 antes de Fk com K1 para restaurar a mensagem original.

Além das funções citadas acima, usaremos também duas tabelas de substituições, S0 e S1, e duas funções auxiliares, binary_to_decimal e exclusive_or, as quais são essenciais para o bom funcionamento do algoritmo.

[538]:
```
S0 = [["01", "00", "11", "10"],
      ["11", "10", "01", "00"],
      ["00", "10", "01", "11"],
      ["11", "01", "11", "10"]]

S1 = [["00", "01", "10", "11"],
```

```
      ["10", "00", "01", "11"],
      ["11", "00", "01", "00"],
      ["10", "01", "00", "11"]]

def binary_to_decimal(n):
    """Converts a binary string to its decimal equivalent."""
    decimal = 0
    for i, bit in enumerate(reversed(n)):
        decimal += int(bit) * (2 ** i)
    return decimal

def exclusive_or(n0, n1):
    """Performs a bitwise exclusive OR operation between two binary bits."""
    if n0 == n1:
        return "0"
    else:
        return "1"
```

PS: não esqueça de executar todos os blocos de código para evitar problemas de falta de definições.

1.2.1 Função de Permutação Inicial - IP

Uma função simples de permutação que embaralha o valor do bloco de entrada de forma determinística.

[539]:
```
def IP(block):
    """Initial Permutation (IP) rearranges the input block in a specific␣
 ↪arrange"""
    return [block[1], block[5], block[2], block[0], block[3], block[7],␣
 ↪block[4], block[6]]
```

Será a primeira função sobre nosso texto em claro.

1.2.2 Função de Permutação Final - FP (Inversa da IP)

Uma função simples de permutação que embaralha o valor do bloco de forma determinística.

[540]:
```
def FP(block):
    """Final Permutation (FP) is equivalent to the inverse function of IP"""
    return [block[3], block[0], block[2], block[4], block[6], block[1],␣
 ↪block[7], block[5]]
```

Será a ultima função aplicada sobre o nosso bloco de 8 bits.

1.2.3 Função de Expansão e Permutação - EP

A função EP realiza a expansão de um bloco de 4 bits para 8 bits e, em seguida, aplica a operação de XOR entre o bloco expandido e uma chave de 8 bits. A expansão é feita reorganizando os bits do bloco original em uma nova ordem, duplicando alguns deles para atingir o tamanho necessário.

[541]:
```
def EP(key, block):
    """Expansion Permutation (EP) expands and XORs a 4-bit block with an 8-bit␣
↪key"""
    return [exclusive_or(block[3], key[0]), exclusive_or(block[0], key[1]),␣
↪exclusive_or(block[1], key[2]), exclusive_or(block[2], key[3]),
            exclusive_or(block[1], key[4]), exclusive_or(block[2], key[5]),␣
↪exclusive_or(block[3], key[6]), exclusive_or(block[0], key[7])]
```

Essa função será chamada dentro da função F, utilizando como entrada o bloco de 4 bits correspondente à metade direita do bloco de 8 bits inicial, junto com uma das subchaves K1 ou K2. Ela é usada em ambas as passagens da função F durante os processos de criptografia e descriptografia.

1.2.4 Função de Permutação - P4

Uma função simples de permutação que embaralha um bloco de 4 bits de forma determinística.

[542]:
```
def P_4(block):
    """Performs the P4 permutation on a 4-bit block."""
    return [block[1], block[3], block[2], block[0]]
```

Será aplicada no valor do resultado das tabelas de substituição dentro da Função F.

1.2.5 Função F

Aqui chegamos no ponto mais crucial da nossa implementação, a funçao F. A função realiza três principais operações em um bloco de 4 bits: aplica a expansão e permutação (EP) para expandir o bloco de 4 bits para 8 bits com base na chave; usa as tabelas S0 e S1 para determinar as linhas e colunas correspondentes e obter um valor binário de saída; e, por fim, utiliza a permutação P4 para reorganizar os bits em um padrão específico, resultando em uma saída final do bloco.

[543]:
```
def F(key, block):
    """F function applies EP, S-box substitutions, and P4 permutations to a␣
↪4-bit block."""
    block = EP(key, block)

    row_S0    = (binary_to_decimal(block[0] + block[3]))
    column_S0 = (binary_to_decimal(block[1] + block[2]))
    row_S1    = (binary_to_decimal(block[4] + block[7]))
    column_S1 = (binary_to_decimal(block[5] + block[6]))

    return P_4(S0[row_S0][column_S0] + S1[row_S1][column_S1])
```

A função F é chamada dentro da função Fk para processar a metade direita do bloco de 8 bits. Ela é usada para aplicar substituições e permutações.

1.2.6 Função Fk

A função Fk combina a saída da função F com a metade esquerda de um bloco de 8 bits usando a operação XOR, enquanto a metade direita permanece inalterada. O resultado é um novo bloco de 8 bits, com a metade esquerda modificada.

[544]:
```
def Fk(key, block):
    """Fk function applies XOR on the output of F with the left half of the␣
 ↪block."""
    left = block[:4]
    right = block[4:]

    f_result = F(key, right)
    left_result = []
    for i in range(len(left)):
        left_result = left_result + [exclusive_or(left[i], f_result[i])]

    return left_result + right
```

Essa função será chamada duas vezes durante os processos de criptografia e descriptografia, utilizando uma das subchaves (K1 ou K2) como entrada junto com o bloco em processamento. Na criptografia, a ordem das subchaves é K1 seguida de K2, enquanto na descriptografia é K2 seguida de K1.

1.2.7 Função SW

A função SW realiza a troca das metades do bloco de 8 bits, trocando a metade esquerda com a metade direita.

[545]:
```
def SW(block):
    """Switches the left and right halves of the block."""
    return block[4:] + block[:4]
```

Esta função será chamada após a primeira aplicação da função Fk, trocando as metades do bloco de dados.

1.2.8 Executando Criptografia e Descriptografia

Para testarmos a operação desse algoritmo executaremos nosso algoritmo tanto para descriptografia quanto para a criptografia, utilizando as mesmas subchaves geradas anteriormente.

[546]:
```
cipher_text = FP(Fk(k2, SW(Fk(k1, IP(block)))))
plain_text = FP(Fk(k1, SW(Fk(k2, IP(cipher_text)))))

print(f'Cipher Text: {cipher_text}')
print(f'Plain Text: {plain_text}')
print()

if plain_text == block:
    print(f'Success! The decryption was successful, and the original block␣
 ↪matches the decrypted plaintext.')
else:
    print(f'Error: The decryption failed. The original block does not match the␣
 ↪decrypted plaintext.')
```

```
Cipher Text: ['1', '0', '1', '0', '1', '0', '0', '0']
Plain Text: ['1', '1', '0', '1', '0', '1', '1', '1']

Success! The decryption was successful, and the original block matches the
decrypted plaintext.
```

O resultado deve ser: Cipher Text: [‘1’, ‘0’, ‘1’, ‘0’, ‘1’, ‘0’, ‘0’, ‘0’]

e apresentar a mensagem: Success! The decryption was successful, and the original block matches the decrypted plaintext.

Assim, sinalizando que o nosso processo de geração de chaves, criptografia e descriptografia estão funcionando de forma adequada e conivente com o esperado. Caso não tenha obtido esse resultado, rode novamente todos os blocos de código deste arquivo notebook jupyter.

2 Repositório da Implementação

Para mais informações sobre implementação, confira o repositório: https://github.com/EduardoMarciano/Simplified-DES