Ministry of Education and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 4

Discipline: Cryptography and Security

Elaborated: Berco Andrei, FAF - 221

Checked: asist. univ., Nirca Dumitru

Chișinău 2024

Topic: Cypher Blocks. DES Algorithm

Tasks:

To develop a program in one of the preferred programming languages for implementing an element of the DES algorithm. The task will be chosen based on the student's ordinal number nn in the group list, according to the formula: nr_task=nmod 11nr_task = n \mod 11. For each task, the tables used and all intermediate steps must be displayed on the screen. The input data should either be user-provided or generated randomly.

Task 2.3: In the DES algorithm, K^+ is given. Determine the round key K_i for a given i.

Theoretical notes:

Data Encryption Standard (DES) - Overview

1. **Type**: Symmetric-key block cipher.
2. **Block Size**: 64-bit blocks.
3. **Key Size**: 56 bits (plus 8 parity bits).
4. **Structure**: Based on the Feistel network with 16 rounds.

DES Algorithm Steps:

1. **Initial Permutation (IP)**: Rearranges the input bits.
2. **16 Rounds**:
   - Expansion (E): Expands 32 bits to 48 bits.
   - Key Mixing: XOR with the 48-bit round key (KiK_iKi).
   - Substitution (S-Box): Reduces 48 bits to 32 bits.
   - Permutation (P): Rearranges bits.
3. **Final Permutation (FP)**: Inverse of the initial permutation.

Key Generation:

- 56-bit key (K+K^+K+) generates 16 round keys (KiK_iKi) via bit shifts and permutations.

Security:

- **Strength**: Simple design with strong diffusion and confusion principles.
- **Weakness**: 56-bit keys are vulnerable to brute-force attacks; replaced by AES and 3DES in modern use.

Implementation:

```
# Permutation tables for DES
PC1 = [
    57, 49, 41, 33, 25, 17, 9,
    1, 58, 50, 42, 34, 26, 18,
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4
]

PC2 = [
    14, 17, 11, 24, 1, 5,
    3, 28, 15, 6, 21, 10,
    23, 19, 12, 4, 26, 8,
    16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
]
```

1. Permutation Tables (PC1 and PC2):

- **PC1**: Used to permute the 64-bit input key (K^+ into a 56-bit key by removing parity bits.
- **PC2**: Reduces the combined 56-bit key (after shifts) to 48 bits for round keys (K_i).

```
# Shift schedule
SHIFT_SCHEDULE = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
```

2. Shift Schedule (SHIFT_SCHEDULE):

- Defines the number of left shifts to be applied to each half of the key (C and D) for each of the 16 DES rounds.

```python
def left_shift(bits, shift):
    """
    Perform a left circular shift on the given bits.
    """
    return bits[shift:] + bits[:shift]
```

left_shift(bits, shift):

- Performs a circular left shift on a binary string by a given number of positions.

```python
def apply_permutation(bits, table):
    return ''.join(bits[i - 1] for i in table)
```

apply_permutation(bits, table):

- Maps the input bits to new positions based on a permutation table (e.g., PC1 or PC2).

```python
def generate_round_key(k_plus, round_index):
    # Step 1: Apply PC1 to the 64-bit key to reduce it to 56 bits
    permuted_key = apply_permutation(k_plus, PC1)
    print(f"Key after PC1 permutation (56 bits): {permuted_key}\n")

    # Step 2: Split into two halves (C and D)
    c, d = permuted_key[:28], permuted_key[28:]
    print(f"Initial C0: {c}")
    print(f"Initial D0: {d}\n")

    # Step 3: Perform left shifts as per the shift schedule
    for i in range(round_index):
        c = left_shift(c, SHIFT_SCHEDULE[i])
        d = left_shift(d, SHIFT_SCHEDULE[i])
        print(f"After round {i + 1} shift:")
        print(f"C{i + 1}: {c}")
        print(f"D{i + 1}: {d}\n")

    # Step 4: Combine C and D after the final shift
    combined_key = c + d
    print(f"Combined key after final shift (56 bits): {combined_key}\n")

    # Step 5: Apply PC2 to derive the 48-bit round key
    round_key = apply_permutation(combined_key, PC2)
    print(f"Round Key K{round_index} after PC2 permutation (48 bits):
{round_key}\n")

    return round_key
```

generate_round_key(k_plus, round_index):

- Implements the DES key schedule for generating round keys:
    1. Applies PC1 to derive a 56-bit key.
    2. Splits the key into two 28-bit halves (C and D).
    3. Applies left shifts as per the shift schedule for the specified round.
    4. Combines shifted halves and applies PC2 to derive the 48-bit round key.

Results:

```
Enter 64-bit K+ (leave blank to generate randomly):
Generated random K+: 1110010001110100101010100111011010011100000100010001110101011111
Enter round index (1-16): 4
Key after PC1 permutation (56 bits): 11000101000110111000111100001101110011010011111111001010

Initial C0: 1100010100011011100011110000
Initial D0: 1101110011010011111111001010

After round 1 shift:
C1: 1000101000110111000111100001
D1: 1011100110100111111110010101

After round 2 shift:
C2: 0001010001101110001111000011
D2: 0111001101001111111100101011

After round 3 shift:
C3: 0101000110111000111100001100
D3: 1100110100111111110010101101

After round 4 shift:
C4: 0100011011100011110000110001
D4: 0011010011111111001010110111

Combined key after final shift (56 bits): 01000110111000111100001100010011010011111111001010110111

Round Key K4 after PC2 permutation (48 bits): 011100011101100000110001111111011000111110010001

Final Round Key K4: 011100011101100000110001111111011000111110010001
```

Conclusion

In this laboratory work, we explored the key generation process of the DES (Data Encryption Standard) algorithm, focusing on the derivation of round keys. We implemented critical steps, including the use of permutation tables (PC1 and PC2), bit shifts, and the combination of key halves. Through this, we gained hands-on experience with key concepts such as:

1. **Permutations**: The significance of rearranging bits to enhance security and randomness.
2. **Circular Shifting**: The role of left shifts in introducing variation between rounds.
3. **Round Key Generation**: The importance of deriving unique keys for each DES round to ensure cryptographic strength.

The implementation allowed us to visualize intermediate steps and understand how DES transforms the initial key into round-specific keys. This practical understanding reinforces the importance of structured transformations and permutations in cryptographic algorithms. Overall, this lab demonstrated the foundational principles behind symmetric encryption and provided valuable insights into the workings of the DES algorithm.