NSC Assignment 2–DES

Arpan Kumar (2021020)
Pranav Tanwar (2022368)

INDRAPRASTHA INSTITUTE *of* INFORMATION TECHNOLOGY
DELHI

Objectives

- To implement the DES encryption and decryption algorithm, with plaintext and key given as input.
- To verify whether the ciphertext obtained after encryption yields the same value as given in the plaintext using encryption using three test cases.
- To check whether the intermediate values match, i.e, output of round 1 of encryption is the same as round 15 of decryption, and that of round 14 of encryption is same as round 2 of decryption.

Encryption

Input- 64 bit plaintext

Key-64 bits, after parity drop- 56 bits

Output- 64 bit ciphertext

```python
for pair in range(3):
    print(f"\n=== Pair {pair+1} Encryption ===")

    # plaintext_input = input("Enter plaintext in hex: ")
    plaintext_input = test_cases[pair]['plaintext']
    # key_input = input("Enter key in hex: ")
    key_input = test_cases[pair]['key']

    key = hex_to_binary(key_input)
    # Removing parity bits
    key2 = ""
    for i in range(len(key)):
        if (i+1) % 8 == 0:
            continue
        key2 += key[i]
    key = key2
    key_after_perm = [key[key_perm_table[i+1]-1] for i in range(len(key))]
    key = key_after_perm
    plaintext = hex_to_binary(plaintext_input)

    # Applying initial permutation to plaintext
    plaintext_after_ip = [0]*64
    for i in range(1, 65):
        plaintext_after_ip[i-1] = int(plaintext[ip_table[i]-1])
    print(f"After initial permutation: {binary_to_hex(''.join(str(i) for i in plaintext_after_ip))}\n")

    # Splitting plaintext into halves
    L = plaintext_after_ip[:32]
    R = plaintext_after_ip[32:]
    R_orig = R.copy()
```

```python
enc_round_outputs = []  # To store per-round outputs for encryption
# 16 rounds of DES
for round in range(16):
    key = round_key_calculator(key, round)
    round_key = round_key_compressor(key)
    rk = ""
    for i in round_key:
        rk += str(i)
    print(f"Round key: {binary_to_hex(rk)}")
    out = R_function(R, round_key)
    R = []
    for j in range(0, 32):
        R.append(out[j] ^ L[j])
    L = R_orig
    R_orig = R
    round_output = ""
    temp = L + R
    for i in range(len(R+L)):
        round_output += str(temp[i])
    round_hex = binary_to_hex(round_output).upper()
    print(f"Output in Round {round+1} is {round_hex}\n")
    enc_round_outputs.append(round_hex)
output = R + L
final = [output[final_permutation[i+1]-1] for i in range(len(output))]
final_str = ""
for i in final:
    final_str += str(i)
final_ciphertext = binary_to_hex(final_str).upper()
print(f"Final encrypted output is {final_ciphertext}")

encryption_results.append({
    "round_outputs": enc_round_outputs,
    "ciphertext": final_ciphertext
})
```

Encryption– Output

```
=== Pair 2 Encryption ===
After initial permutation: 33ff33000f550f55

Round key: 365f93eb96a8
Output in Round 1 is 0F550F5502FC6459

Round key: 6e7b011bf6ea
Output in Round 2 is 02FC6459487C32A9

Round key: 0bbd7d3cdd25
Output in Round 3 is 487C32A9C200210F

Round key: cd64db8a6cf6
Output in Round 4 is C200210F390C5150

Round key: 77cfa8edeb91
Output in Round 5 is 390C51505EE61DFD

Round key: dab983b3465b
Output in Round 6 is 5EE61DFD9A52324F

Round key: 39ae5fdf9306
Output in Round 7 is 9A52324FD883EA78

Round key: 65748e9467ec
Output in Round 8 is D883EA78E11ECE47
```

```
Round key: 5437ec557cc5
Output in Round 9 is E11ECE47A18037D4

Round key: d2dc716aa0fd
Output in Round 10 is A18037D46409038E

Round key: cdeb66a3fd8f
Output in Round 11 is 6409038E075CDB62

Round key: a2f78f2e17b3
Output in Round 12 is 075CDB6277F0D6D1

Round key: 791763df4967
Output in Round 13 is 77F0D6D1CFC40095

Round key: e1d8f946cbd8
Output in Round 14 is CFC40095C9D4B3BD

Round key: 95e3d6d1b55d
Output in Round 15 is C9D4B3BDDFBD3951

Round key: b64f5eed965d
Output in Round 16 is DFBD395146485371

Final encrypted output is AFC4E0B8AF29D7A0
```

For input plaintext- FEDCBA9876543210

Key-

AABBCCDDEEFF0011

Ciphertext-

AFC4E0B8AF29D7A0

Decryption

```python
print(f"\n=== Pair {pair+1} Decryption ===")
ciphertext_input = final_ciphertext
key_input = test_cases[pair]['key']
ciphertext = hex_to_binary(ciphertext_input)
key = hex_to_binary(key_input)
# Removing parity bits
key2 = ""
for i in range(len(key)):
    if (i+1) % 8 == 0:
        continue
    key2 += key[i]
key = key2
key_after_perm = [key[key_perm_table[i+1]-1] for i in range(len(key))]
key = key_after_perm

ciphertext_after_ip = [0]*64
for i in range(64):
    ciphertext_after_ip[i] = int(ciphertext[ip_table[i+1]-1])

subkeys = []
for i in range(16):
    key = round_key_calculator(key, i)
    subkeys.append(round_key_compressor(key))

L = ciphertext_after_ip[:32]
R = ciphertext_after_ip[32:]
R_orig = R.copy()
```

```python
dec_round_outputs = []  # To store per-round outputs for decryption
for round in range(16):
    round_key = subkeys[15-round]
    rk = ""
    for i in round_key:
        rk += str(i)
    print(f"Round key: {binary_to_hex(rk)}")
    out = R_function(R, round_key)
    R = []
    for j in range(0, 32):
        R.append(out[j] ^ L[j])
    L = R_orig
    R_orig = R
    round_output = ""
    round_output_match = ""
    temp = L + R
    temp_match=R+L
    for i in range(len(R+L)):
        round_output += str(temp[i])
    for i in range(len(R+L)):
        round_output_match += str(temp_match[i])
    round_hex = binary_to_hex(round_output).upper()
    round_hex_match=binary_to_hex(round_output_match).upper()
    print(f"Output in Round {round+1} is {round_hex}")
    dec_round_outputs.append(round_hex_match)
output = R + L
output_match=R_orig+L
final = [output[final_permutation[i+1]-1] for i in range(len(output))]
final_str = ""
for i in final:
    final_str += str(i)
final_plaintext = binary_to_hex(final_str).upper()
print(f"Final output is {final_plaintext}")
decryption_results.append({
    "round_outputs": dec_round_outputs,
    "plaintext": final_plaintext
})
```

Decryption Output

```
=== Pair 2 Decryption ===
Round key: b64f5eed965d
Output in Round 1 is DFBD3951C9D4B3BD
Round key: 95e3d6d1b55d
Output in Round 2 is C9D4B3BDCFC40095
Round key: e1d8f946cbd8
Output in Round 3 is CFC4009577F0D6D1
Round key: 791763df4967
Output in Round 4 is 77F0D6D1075CDB62
Round key: a2f78f2e17b3
Output in Round 5 is 075CDB626409038E
Round key: cdeb66a3fd8f
Output in Round 6 is 6409038EA18037D4
Round key: d2dc716aa0fd
Output in Round 7 is A18037D4E11ECE47
Round key: 5437ec557cc5
Output in Round 8 is E11ECE47D883EA78
Round key: 65748e9467ec
```

```
Output in Round 9 is D883EA789A52324F
Round key: 39ae5fdf9306
Output in Round 10 is 9A52324F5EE61DFD
Round key: dab983b3465b
Output in Round 11 is 5EE61DFD390C5150
Round key: 77cfa8edeb91
Output in Round 12 is 390C5150C200210F
Round key: cd64db8a6cf6
Output in Round 13 is C200210F487C32A9
Round key: 0bbd7d3cdd25
Output in Round 14 is 487C32A902FC6459
Round key: 6e7b011bf6ea
Output in Round 15 is 02FC64590F550F55
Round key: 365f93eb96a8
Output in Round 16 is 0F550F5533FF3300
Final output is FEDCBA9876543210
```

We observe that the final output obtained is the same as the original plaintext.

Verification on Intermediate Values

```
=== VERIFICATION OF ROUND OUTPUTS ===
Pair 1: Encryption Round 1 equals Decryption Round 15: F0AAF0AAEF4A6544
Pair 1: Encryption Round 14 equals Decryption Round 2: 18C3155AC28C960D
Pair 2: Encryption Round 1 equals Decryption Round 15: 0F550F5502FC6459
Pair 2: Encryption Round 14 equals Decryption Round 2: CFC40095C9D4B3BD
Pair 3: Encryption Round 1 equals Decryption Round 15: 0000000038DBF9CB
Pair 3: Encryption Round 14 equals Decryption Round 2: 044D9D35472AC861
PS C:\Users\prana\Desktop\NSC_Assignments\Network-Security\Assignment2>
```

Encryption round 1 gives the same value in output as decryption round 15, and encryption round 14 gives same output as decryption round 2.