CSE350-
NETWORK SECURITY
ASSIGNMENT NO. 1

Data Encryption Standard

By –Angadjeet (2022071)
Arav amawate(2022091)

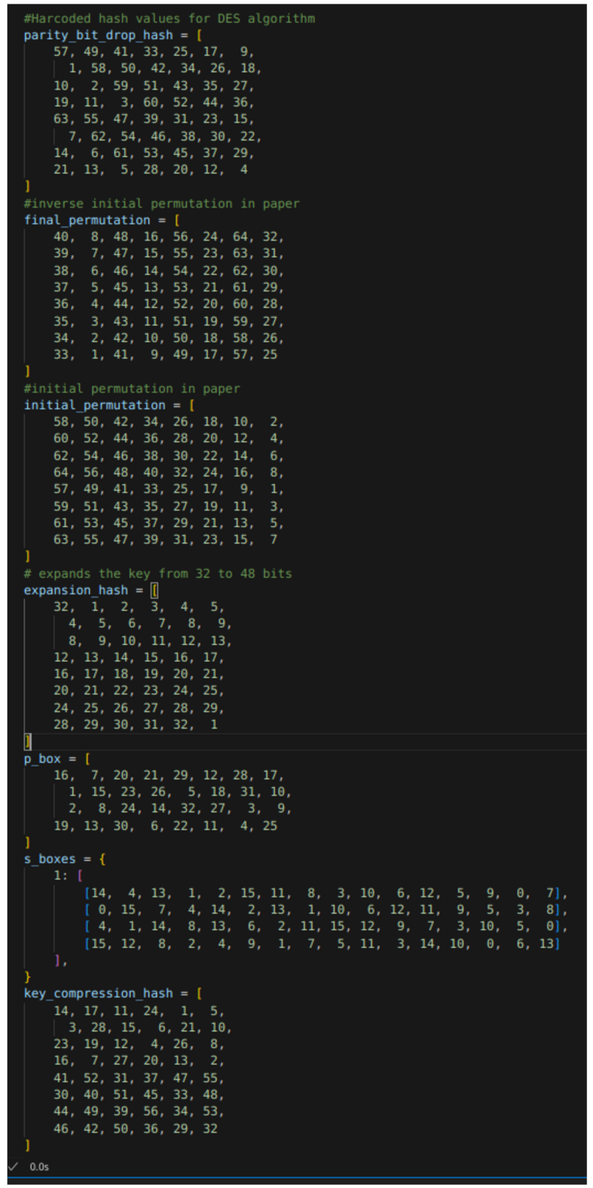
Introduction

Objective:

- Design and implemented a DES (Data Encryption Standard) algorithm in Python.
- Achieve secure encryption and decryption of plaintext messages.
- Demonstrate intermediate round outputs to verify that encryption and decryption are exact inverses.

System Overview

- **Permutation Tables:**Initial permutation, inverse (final) permutation, expansion, and P-box (straight permutation) tables.
- **S-Boxes:** Eight S-boxes for non-linear substitution.
- **Key Generation Module:** Converts an 8-character key to 64-bit binary.Drops parity bits and generates 16 round subkeys through left shifts and compression.
- **Encryption & Decryption Modules:**Encryption: Processes each 64-bit block through 16 rounds of the DES round function.
- **Decryption:** Reverses the encryption rounds using subkeys in reverse order.
- **Intermediate Verification:**Comparison of outputs from specific rounds (e.g., 1st vs. 15th and 14th vs. 2nd) to ensure correctness.



```
#Harcoded hash values for DES algorithm
parity_bit_drop_hash = [
    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
]
#inverse initial permutation in paper
final_permutation = [
    40,  8, 48, 16, 56, 24, 64, 32,
    39,  7, 47, 15, 55, 23, 63, 31,
    38,  6, 46, 14, 54, 22, 62, 30,
    37,  5, 45, 13, 53, 21, 61, 29,
    36,  4, 44, 12, 52, 20, 60, 28,
    35,  3, 43, 11, 51, 19, 59, 27,
    34,  2, 42, 10, 50, 18, 58, 26,
    33,  1, 41,  9, 49, 17, 57, 25
]
#initial permutation in paper
initial_permutation = [
    58, 50, 42, 34, 26, 18, 10,  2,
    60, 52, 44, 36, 28, 20, 12,  4,
    62, 54, 46, 38, 30, 22, 14,  6,
    64, 56, 48, 40, 32, 24, 16,  8,
    57, 49, 41, 33, 25, 17,  9,  1,
    59, 51, 43, 35, 27, 19, 11,  3,
    61, 53, 45, 37, 29, 21, 13,  5,
    63, 55, 47, 39, 31, 23, 15,  7
]
# expands the key from 32 to 48 bits
expansion_hash = [
    32,  1,  2,  3,  4,  5,
     4,  5,  6,  7,  8,  9,
     8,  9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32,  1
]
p_box = [
    16,  7, 20, 21, 29, 12, 28, 17,
     1, 15, 23, 26,  5, 18, 31, 10,
     2,  8, 24, 14, 32, 27,  3,  9,
    19, 13, 30,  6, 22, 11,  4, 25
]
s_boxes = {
    1: [
        [14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7],
        [ 0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8],
        [ 4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0],
        [15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13]
    ],
}
key_compression_hash = [
    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
]
✓ 0.0s
```

Helper Functions

```python
def s_box_transformation(self, right_48_bits):
    s_box_output_32_bits = []
    start_counter = 0
    end_counter = 6
    curr_block = 1
    for _ in range(8):
        six_bits = right_48_bits[start_counter:end_counter]
        row = int(str(six_bits[0]) + str(six_bits[5]), 2)
        col = int("".join(str(b) for b in six_bits[1:5]), 2)
        s_box_value = s_boxes[curr_block][row][col]
        s_box_bits = format(s_box_value, '04b')
        s_box_output_32_bits += [int(bit) for bit in s_box_bits]
        start_counter += 6
        end_counter += 6
        curr_block += 1
    return s_box_output_32_bits
```

s_box_transformation:

Splits the 48-bit input into eight 6-bit segments and, using each segment's row and column indices, maps them via the corresponding S-box to produce a 32-bit output.

```python
def F_box(self, right_32_bits, subkey):
    expanded = self.expansion_p_box(right_32_bits)
    xored = [self.xor(expanded[i], subkey[i]) for i in range(48)]
    s_out = self.s_box_transformation(xored)
    return self.straight_p_box(s_out)
```

F_box:

Expands a 32-bit block to 48 bits, XORs it with the subkey, passes the result through the S-box transformation, and finally applies a straight permutation.

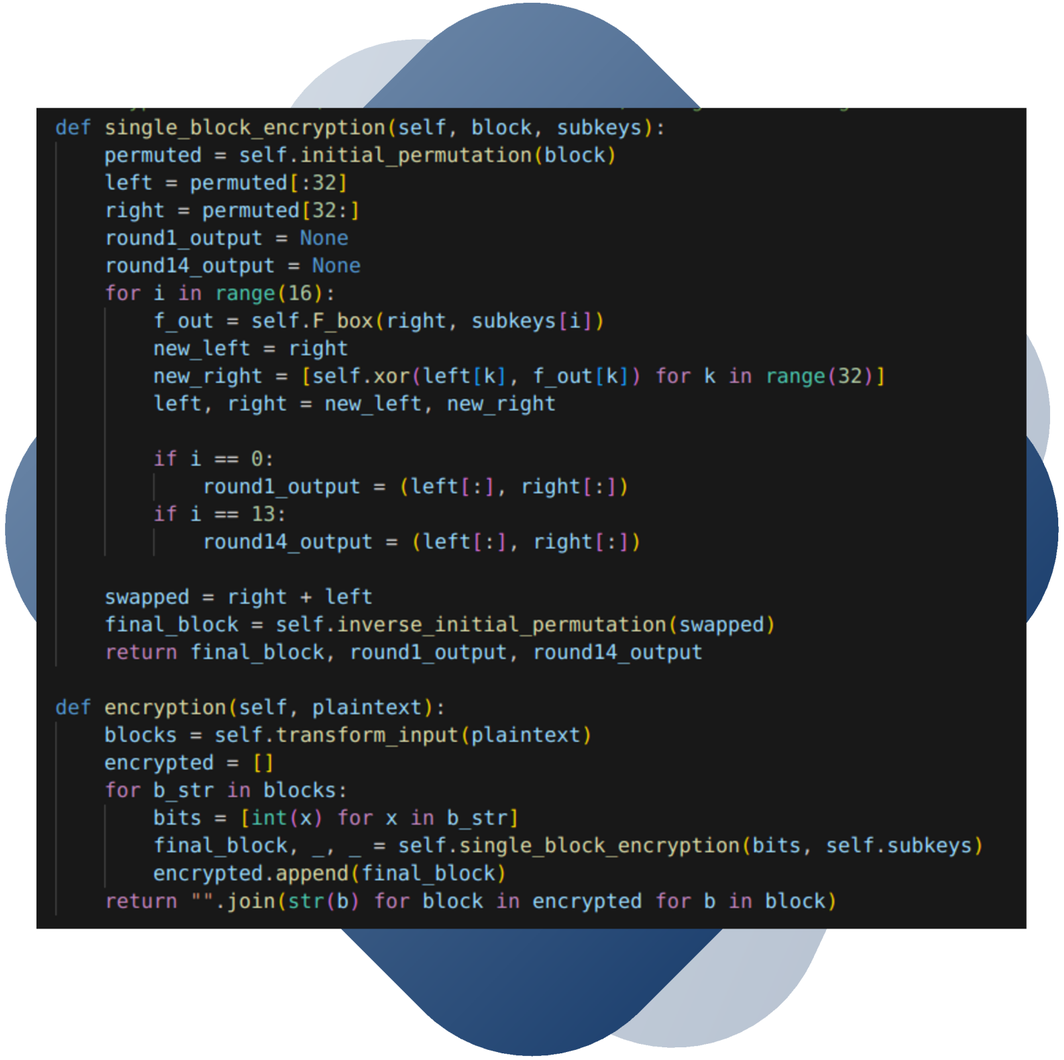
```python
def generate_subkeys(self, key):
    key_56 = self.parity_drop(key)
    subkeys = []
    left_key = key_56[:28]
    right_key = key_56[28:]
    shifting_hash = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
    for i in range(16):
        left_key = self.left_shift(left_key, shifting_hash[i])
        right_key = self.left_shift(right_key, shifting_hash[i])
        combined = left_key + right_key
        subkey = [combined[j - 1] for j in key_compression_hash]
        subkeys.append(subkey)
    return subkeys
```

generate_subkeys:

Drops parity bits from the 64-bit key, splits it, applies scheduled left shifts, and compresses the result to produce 16 unique 48-bit round subkeys.

Encryption Process

- **Initial Permutation:**Rearranges the 64-bit block using the predefined permutation table.
- **Splitting:**Divides the block into 32-bit left and right halves.
- **For each round :**Apply the F-box (which does expansion, XOR with the round subkey, S-box substitution, and a straight permutation).Update the halves (XOR with the left half and swap).Capture intermediate outputs at round 1 and round 14 for verification.
- **Final Permutation:**After the rounds, the halves are swapped and a final permutation is applied to yield the encrypted block.



```
def single_block_encryption(self, block, subkeys):
    permuted = self.initial_permutation(block)
    left = permuted[:32]
    right = permuted[32:]
    round1_output = None
    round14_output = None
    for i in range(16):
        f_out = self.F_box(right, subkeys[i])
        new_left = right
        new_right = [self.xor(left[k], f_out[k]) for k in range(32)]
        left, right = new_left, new_right

        if i == 0:
            round1_output = (left[:], right[:])
        if i == 13:
            round14_output = (left[:], right[:])

    swapped = right + left
    final_block = self.inverse_initial_permutation(swapped)
    return final_block, round1_output, round14_output

def encryption(self, plaintext):
    blocks = self.transform_input(plaintext)
    encrypted = []
    for b_str in blocks:
        bits = [int(x) for x in b_str]
        final_block, _, _ = self.single_block_encryption(bits, self.subkeys)
        encrypted.append(final_block)
    return "".join(str(b) for block in encrypted for b in block)
```

Decryption Process

- **Initial Permutation:**Rearranges the 64-bit ciphertext block.
- **Splitting:**Splits the block into left and right halves.
- **For each round(Reversed Order):**Uses the subkeys in reverse order (i.e., subkeys[15 - i]).Applies the F-box, updates the halves (XOR operation), and captures intermediate outputs (round 2 and round 15).
- **Final Permutation**:After swapping the halves, the inverse initial permutation is applied to yield the decrypted block.

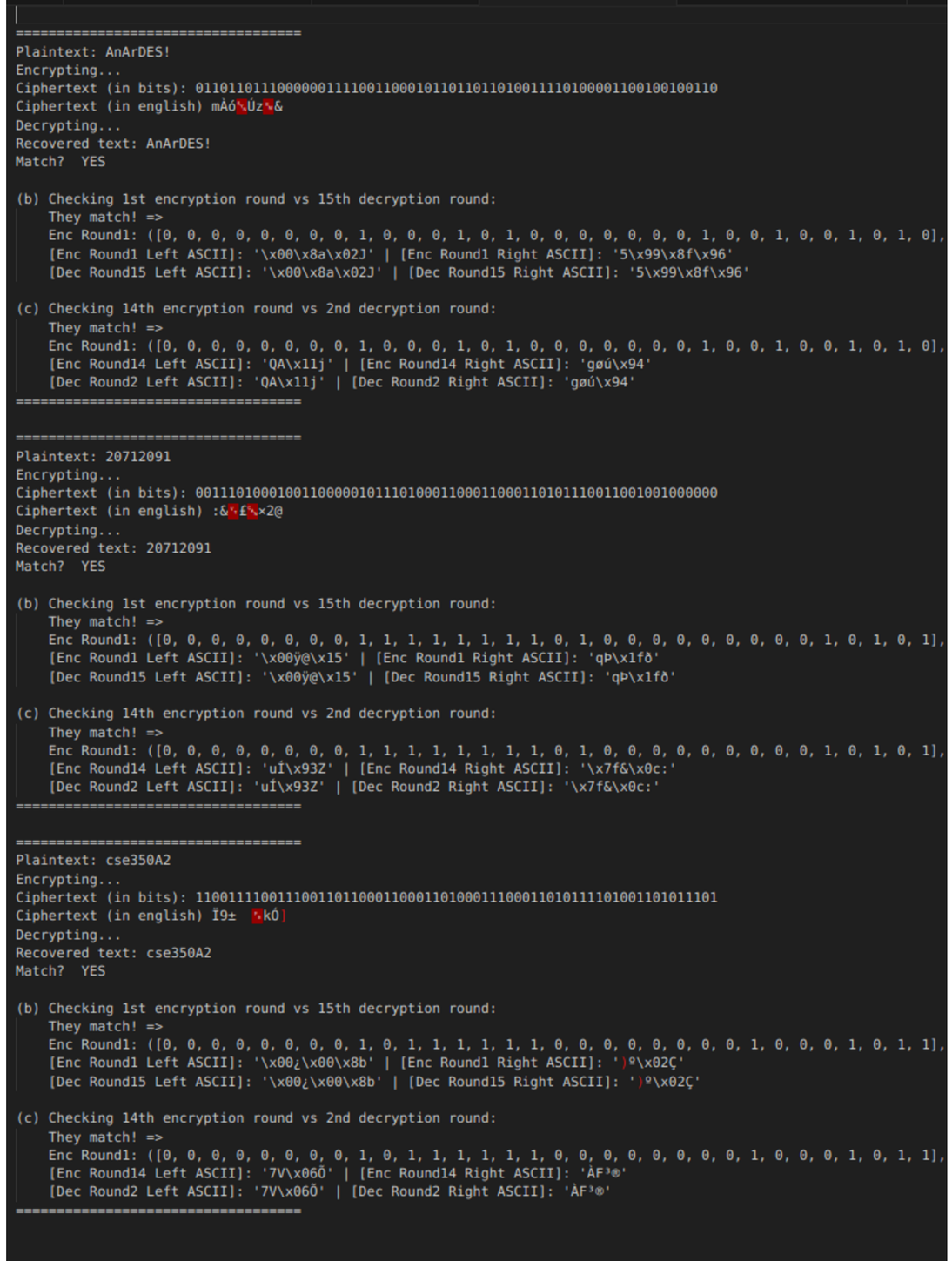
```python
def single_block_decryption(self, block, subkeys):
    permuted = self.initial_permutation(block)
    left = permuted[:32]
    right = permuted[32:]
    round2_output = None
    round15_output = None

    for i in range(16):
        f_out = self.F_box(right, subkeys[15 - i])
        new_left = right
        new_right = [self.xor(left[k], f_out[k]) for k in range(32)]
        left, right = new_left, new_right

        if i == 1:
            round2_output = (right[:], left[:])
        if i == 14:
            round15_output = (right[:], left[:])

    swapped = right + left
    final_block = self.inverse_initial_permutation(swapped)
    return final_block, round2_output, round15_output
```

```python
def decryption(self, ciphertext_bits):
    size = 64
    blocks = textwrap.wrap(ciphertext_bits, size)
    decrypted = []
    for b_str in blocks:
        bits = [int(x) for x in b_str]
        final_block, _, _ = self.single_block_decryption(bits, self.subkeys)
        decrypted.append(final_block)
    return "".join(str(b) for block in decrypted for b in block)
```



Output code

- Output Highlights:
- Encrypted Text:Ciphertext is shown both in its binary form and as its corresponding ASCII representation.
- Decrypted Text:The system successfully recovers the original plaintext, demonstrating the reversibility of the DES process.
- Verification Process:The code captures specific intermediate outputs (both in binary and ASCII) during encryption and decryption.These outputs are compared to confirm that the decryption process is correctly reversing the encryption steps.

Verification

- Example Output :
- Plaintext: "AnArDES!"
- Ciphertext (Binary):
- 0110110111000000111100110001011011011010011110100001100100100110
- Ciphertext (ASCII):
- mÀó Úz &
- Recovered Text: "AnArDES!"
- Intermediate Verification:Round 1 encryption output matches Round 15 decryption output.Round 14 encryption output matches Round 2 decryption output.

THANKS!