1. Data Encryption Standard (DES) Algorithm

1.1. Introduction

The Data Encryption Standard (DES) is a symmetric-key block cipher algorithm developed in the 1970s. It was widely used for secure electronic communication before being superseded by more advanced encryption methods.

1.2. Key Characteristics

- Block size: 64 bits
- Key size: 56 bits (technically 64 bits, but every 8th bit is used for parity)
- Number of rounds: 16

1.3. Algorithm Overview

1.3.1. Step 1: Initial Permutation

The 64-bit plaintext block undergoes an initial permutation.

1.3.2. Step 2: 16 Rounds of Encryption

Each round consists of:

- Splitting the block into two 32-bit halves (Left and Right)
- Expanding the Right half from 32 to 48 bits
- XORing with a 48-bit round key
- Passing through 8 S-boxes to reduce back to 32 bits
- Permutation of the resulting 32 bits
- XORing with the Left half
- Swapping Left and Right halves (except in the final round)

1.3.3. Step 3: Final Permutation

The output of the 16th round undergoes a final permutation (inverse of the initial permutation).

1.4. Key Schedule

- The 56-bit key is divided into two 28-bit halves
- For each round:
  - Both halves are rotated left by 1 or 2 bits (depending on the round)
  - 48 bits are selected from the 56 bits to form the round key

1.5. Security Considerations

While innovative for its time, DES is now considered insecure due to its short key length. Modern systems use more robust algorithms like AES.

“Strength”, “Weakness”	“Fast and easy to implement”, “Short key length (56 bits)”
“Well-studied and analyzed”, “Vulnerable to brute-force attacks”	“Resistant to differential cryptanalysis”, “Superseded by more secure algorithms”

Listing 1: DES-file.py

```
class ANSI:
    RED = "\033[0;31m"
    GREEN = "\033[0;32m"
    BLUE = "\033[0;34m"

    BOLD = "\033[1m"

    END = "\033[0m"

class DES:
    _IP = """
    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
    """
    IP = [int(x) for x in _IP.strip().split()]

    #IP^-1
    _FP = """
    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
    """
    FP = [int(x) for x in _FP.strip().split()]


    _PC_1 =  """
    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
    """
    PC_1 = [int(x) for x in _PC_1.strip().split()]
```

```
    _PC_2 = """
    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
    """
    PC_2 = [int(x) for x in _PC_2.strip().split()]

    # S-boxes
    _S1 = """
    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
    """
    S1 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S1.strip().split("\n")]

    _S2 = """
    15  1   8 14   6 11   3  4   9  7   2 13  12  0   5 10
    3  13   4  7  15  2   8 14  12  0   1 10   6  9  11  5
    0  14   7 11  10  4  13  1   5  8  12  6   9  3   2 15
    13  8  10  1   3 15   4  2  11  6   7 12   0  5  14  9
    """
    S2 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S2.strip().split("\n")]

    _S3 = """
    10  0   9 14   6  3  15  5   1 13  12  7  11  4   2  8
    13  7   0  9   3  4   6 10   2  8   5 14  12 11  15  1
    13  6   4  9   8 15   3  0  11  1   2 12   5 10  14  7
    1  10  13  0   6  9   8  7   4 15  14  3  11  5   2 12
    """
    S3 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S3.strip().split("\n")]

    _S4 = """
    7  13  14  3   0  6   9 10   1  2   8  5  11 12   4 15
    13  8  11  5   6 15   0  3   4  7   2 12   1 10  14  9
    10  6   9  0  12 11   7 13  15  1   3 14   5  2   8  4
    3  15   0  6  10  1  13  8   9  4   5 11  12  7   2 14
    """
    S4 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S4.strip().split("\n")]

    _S5 = """
    2  12   4  1   7 10  11  6   8  5   3 15  13  0  14  9
    14 11   2 12   4  7  13  1   5  0  15 10   3  9   8  6
    4   2   1 11  10 13   7  8  15  9  12  5   6  3   0 14
    11  8  12  7   1 14   2 13   6 15   0  9  10  4   5  3
    """
```

```
    S5 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S5.strip().split("\n")]

    _S6 = """
    12  1  10 15   9  2   6  8   0 13   3  4  14  7   5 11
    10 15   4  2   7 12   9  5   6  1  13 14   0 11   3  8
    9  14  15  5   2  8  12  3   7  0   4 10   1 13  11  6
    4   3   2 12   9  5  15 10  11 14   1  7   6  0   8 13
    """
    S6 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S6.strip().split("\n")]

    _S7 = """
    4  11   2 14  15  0   8 13   3 12   9  7   5 10   6  1
    13  0  11  7   4  9   1 10  14  3   5 12   2 15   8  6
    1   4  11 13  12  3   7 14  10 15   6  8   0  5   9  2
    6  11  13  8   1  4  10  7   9  5   0 15  14  2   3 12
    """
    S7 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S7.strip().split("\n")]

    _S8 = """
    13  2   8  4   6 15  11  1  10  9   3 14   5  0  12  7
    1  15  13  8  10  3   7  4  12  5   6 11   0 14   9  2
    7  11   4  1   9 12  14  2   0  6  10 13  15  3   5  8
    2   1  14  7   4 10   8 13  15 12   9  0   3  5   6 11
    """
    S8 = [ [int(x.strip()) for x in X.strip().split()] for X in
_S8.strip().split("\n")]

    S = [S1, S2, S3, S4, S5, S6, S7, S8]

    _E = """
    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
    """
    E = [int(x) for x in _E.strip().split()]

    _P = """
    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
    """
    P = [int(x) for x in _P.strip().split()]
```

```
    SK_SHIFT_SCHEDULE = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

    PAD = "~"

    def __init__(self, key: int):
        self.key = key
        self.subkeys = self._generate_subkeys()

    def _permute(self, block: int, table: list[int], block_size: int) -> int:
        result = 0
        for i, pos in enumerate(table):
            bit = (block >> (block_size - pos)) & 1
            result |= bit << (len(table) - 1 - i)
        return result

    def _generate_subkeys(self) -> list[int]:
        key = self._permute(self.key, self.PC_1, 64)
        left, right = key >> 28, key & 0xFFFFFFF
        subkeys = []
        for i in range(16):
            left = ((left << self.SK_SHIFT_SCHEDULE[i]) & 0xFFFFFFF) | (left >> 28 -
self.SK_SHIFT_SCHEDULE[i])
            right = ((right << self.SK_SHIFT_SCHEDULE[i]) & 0xFFFFFFF) | (right >> 28
- self.SK_SHIFT_SCHEDULE[i])
            subkey = self._permute((left << 28) | right, self.PC_2, 56)
            subkeys.append(subkey)
        return subkeys

    def f_function(self, right_half: int, subkey: int) -> int:
        expanded = self._permute(right_half, self.E, 32)
        result = expanded ^ subkey
        block = 0
        for i in (1, 2, 3, 4, 5, 6, 7, 8):
            bits = (result >> (48 - 6 * i)) & 0x3F
            row = (bits >> 4) & 0x2 | (bits & 1)
            col = (bits >> 1) & 0xF
            block = block << 4 | self.S[i - 1][row][col]
        block = self._permute(block, self.P, 32)
        return block

    def _des_encrypt(self, plaintext: int) -> int:
        block = self._permute(plaintext, self.IP, 64)
        left, right = block >> 32, block & 0xFFFFFFFF
        for i in range(16):
            new_right = left ^ self.f_function(right, self.subkeys[i])
            left, right = right, new_right
        final_block = (right << 32) | left
        return self._permute(final_block, self.FP, 64)

    def encrypt(self, plaintext: int | list[int] | list[str] | str) -> str |
list[str] | list[int] | int:

        if isinstance(plaintext, str):
            plaintext = [(plaintext[i:i+8]).ljust(8, self.PAD) for i in range(0,
len(plaintext), 8)]
            return "".join([hex(self._des_encrypt(int(block.encode("utf-8").hex(),
```

```
16)))[2:] for block in plaintext])

        if isinstance(plaintext, int):
            if len(bin(plaintext)[2:]) > 64:
                raise ValueError("Plaintext too long. Max length is 64 bits.")
            return self._des_encrypt(plaintext)

        return [self.encrypt(block) for block in plaintext]

    def _des_decrypt(self, ciphertext: int) -> int:
        subkeys = list(reversed(self.subkeys))
        block = self._permute(ciphertext, self.IP, 64)
        left, right = block >> 32, block & 0xFFFFFFFF
        for i in range(16):
            new_right = left ^ self.f_function(right, subkeys[i])
            left, right = right, new_right
        final_block = (right << 32) | left
        return self._permute(final_block, self.FP, 64)

    def decrypt(self, ciphertext: int | list[int] | list[str] | str):
        if isinstance(ciphertext, str):
            ciphertext = [(ciphertext[i:i+16]).rjust(16, "0") for i in range(0,
len(ciphertext), 16)]
            return "".join([bytes.fromhex(hex(self._des_decrypt(int(x, 16)))
[2:]).decode("utf-8").strip(self.PAD) for x in ciphertext])

        if isinstance(ciphertext, int):
            return self._des_decrypt(ciphertext)

        return [self.decrypt(block) for block in ciphertext]

    def encrypt_file(self, filename: str) -> None:
        with open("encrypted_" + filename, "wb") as e_file:
            with open(filename, "rb") as file:
                while True:
                    block = file.read(8)
                    if not block:
                        break
                    if len(block) < 8:
                        block += self.PAD.encode("utf-8") * (8 - len(block))
                    enc_block_int = self.encrypt(int.from_bytes(block))
                    e_file.write(hex(enc_block_int)[2:].encode("utf-8"))

    def decrypt_file(self, filename: str) -> None:
        with open("decrypted_" + filename, "wb") as d_file:
            with open(filename, "rb") as file:
                while True:
                    block = file.read(16)
                    if not block:
                        break
                    dec_block_int = self.decrypt(int(block, 16))
                    d_file.write(bytes.fromhex(hex(dec_block_int)
[2:]).strip(self.PAD.encode("utf-8")))

cipher = DES(0x133457799BBCDFF1)
```

```python
PT1 = "Devansh Parapalli Practical 04 - 02/08/2024"
PT2 = 0x0123456789ABCDEF
PT3 = [0x0123456789ABCDEF, 0x1234567890ABCDEF]
PT4 = ["Devansh", "Parapalli", "Practical", "04", "02/08/2024"]

print(f"{ANSI.BOLD}Plaintext:{ANSI.END}")
print(f"{ANSI.END}{ANSI.RED}P1:", PT1)
print(f"{ANSI.END}{ANSI.RED}P2:", f"{PT2:016X}")
print(f"{ANSI.END}{ANSI.RED}P3:", f"[{', '.join([f"{x:016X}" for x in PT3])}]")
print(f"{ANSI.END}{ANSI.RED}P4:", PT4)

encrypted1 = cipher.encrypt(PT1)
encrypted2 = cipher.encrypt(PT2)
encrypted3 = cipher.encrypt(PT3)
encrypted4 = cipher.encrypt(PT4)

print(f"{ANSI.END}{ANSI.BOLD}Encryption:{ANSI.END}")
print(f"{ANSI.END}{ANSI.GREEN}C1:", encrypted1)
print(f"{ANSI.END}{ANSI.GREEN}C2:", f"{encrypted2:016X}")
print(f"{ANSI.END}{ANSI.GREEN}C3:", f"[{', '.join([f"{x:016X}" for x in
encrypted3])}]")
print(f"{ANSI.END}{ANSI.GREEN}C4:", encrypted4)

print(f"{ANSI.END}{ANSI.BOLD}Decryption:")
print(f"{ANSI.END}{ANSI.BLUE}D1:", cipher.decrypt(encrypted1))
print(f"{ANSI.END}{ANSI.BLUE}D2:", f"{cipher.decrypt(encrypted2):016X}")
print(f"{ANSI.END}{ANSI.BLUE}D3:", f"[{', '.join([f"{x:016X}" for x in
cipher.decrypt(encrypted3)])}]")
print(f"{ANSI.END}{ANSI.BLUE}D4:", cipher.decrypt(encrypted4))

def display_file(filename):
    with open(filename, "r") as file:
        print(file.read())

print(f"{ANSI.END}{ANSI.BOLD}File Encryption:")
print(f"{ANSI.END}Original File: {ANSI.END}{ANSI.RED}")
display_file("test1.txt")

cipher.encrypt_file("test1.txt")
print(f"{ANSI.END}Encrypted File:{ANSI.END}{ANSI.GREEN}")
display_file("encrypted_test1.txt")

cipher.decrypt_file("encrypted_test1.txt")
print(f"{ANSI.END}Decrypted File:{ANSI.END}{ANSI.BLUE}")
display_file("decrypted_encrypted_test1.txt")
print(f"{ANSI.END}")
```

```
> python -u "c:\DevParapalli\Projects\RTMNU-SEM-7\CNS\practical\practical-04\DES-file.py"
Plaintext:
P1: Devansh Parapalli Practical 04 - 02/08/2024
P2: 0123456789ABCDEF
P3: [0123456789ABCDEF, 1234567890ABCDEF]
P4: ['Devansh', 'Parapalli', 'Practical', '04', '02/08/2024']
Encryption:
C1: 2f1b245c1cb27c054c8a1cf20def98fb6da9e28858f692d682332fcdf17e58a149cc38ce47c815417d061baff4425655
C2: 85E813540F0AB405
C3: [85E813540F0AB405, 0999BF92EB76BA0E]
C4: ['3cc490e3da8a6540', '4c8a1cf20def98fb53710be41f678c69', '4e70f0cf5e9b824f14dcf271a5f01429', '45b5c32b120dc714', '17fa37ed498235499977642ce8fd041f']
Decryption:
D1: Devansh Parapalli Practical 04 - 02/08/2024
D2: 0123456789ABCDEF
C3: [85E813540F0AB405, 0999BF92EB76BA0E]
C4: ['3cc490e3da8a6540', '4c8a1cf20def98fb53710be41f678c69', '4e70f0cf5e9b824f14dcf271a5f01429', '45b5c32b120dc714', '17fa37ed498235499977642ce8fd041f']
Decryption:
D1: Devansh Parapalli Practical 04 - 02/08/2024
D2: 0123456789ABCDEF
Decryption:
D1: Devansh Parapalli Practical 04 - 02/08/2024
D2: 0123456789ABCDEF
D1: Devansh Parapalli Practical 04 - 02/08/2024
D2: 0123456789ABCDEF
D2: 0123456789ABCDEF
D3: [0123456789ABCDEF, 1234567890ABCDEF]
D4: ['Devansh', 'Parapalli', 'Practical', '04', '02/08/2024']
File Encryption:
Original File:
DevanshParapalliTestFile1
Encrypted File:
543ff85a6083c6884a861623fd0fdc574d5aff4fa1e7422a869816cf2b130727
Decrypted File:
DevanshParapalliTestFile1

 pwsh  main ≡ ~1
13:02:59 | 5 Aug, Monday | in  C: →DevParapalli →Projects →RTMNU-SEM-7
❯
```

Figure 1: Output