```
#CS-ASSIGNMENT 2 - S-DES ALGORITHM
#HARSH MEHTA
#TA41
#BATCH B
class SDES:
    def __init__(self):
        self.key = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
        self.P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
        self.P8 = [6, 3, 7, 4, 8, 5, 10, 9]
        self.key1 = [0] * 8
        self.key2 = [0] * 8
        self.IP = [2, 6, 3, 1, 4, 8, 5, 7]
        self.EP = [4, 1, 2, 3, 2, 3, 4, 1]
        self.P4 = [2, 4, 3, 1]
        self.IP_inv = [4, 1, 3, 5, 7, 2, 8, 6]
        self.S0 = [
            [1, 0, 3, 2],
            [3, 2, 1, 0],
            [0, 2, 1, 3],
            [3, 1, 3, 2]
        ]
        self.S1 = [
            [0, 1, 2, 3],
            [2, 0, 1, 3],
            [3, 0, 1, 0],
            [2, 1, 0, 3]
        ]

    def key_generation(self):
        key_ = [0] * 10
        for i in range(10):
            key_[i] = self.key[self.P10[i] - 1]
        Ls = key_[:5]
        Rs = key_[5:]
        Ls_1 = self.shift(Ls, 1)
        Rs_1 = self.shift(Rs, 1)
        self.key1 = self.generate_key(Ls_1, Rs_1)
        Ls_2 = self.shift(Ls, 2)
        Rs_2 = self.shift(Rs, 2)
        self.key2 = self.generate_key(Ls_2, Rs_2)
        print("Your Key-1 :", self.key1)
        print("Your Key-2 :", self.key2)

    def shift(self, ar, n):
        return ar[n:] + ar[:n]

    def generate_key(self, Ls, Rs):
        key_ = Ls + Rs
        return [key_[i - 1] for i in self.P8]

    def encryption(self, plaintext):
        arr = [plaintext[i - 1] for i in self.IP]
        arr1 = self.function_(arr, self.key1)
        after_swap = self.swap(arr1, len(arr1) // 2)
        arr2 = self.function_(after_swap, self.key2)
        ciphertext = [arr2[i - 1] for i in self.IP_inv]
        return ciphertext

    def binary_(self, val):
        if val == 0:
            return "00"
        elif val == 1:
            return "01"
        elif val == 2:
            return "10"
        else:
            return "11"

    def function_(self, ar, key_):
        l = ar[:4]
        r = ar[4:]
        ep = [r[i - 1] for i in self.EP]
        ar = [key_[i] ^ ep[i] for i in range(8)]
        l_1 = ar[:4]
        r_1 = ar[4:]
        row = int(str(l_1[0]) + str(l_1[3]), 2)
        col = int(str(l_1[1]) + str(l_1[2]), 2)
        val = self.S0[row][col]
        str_l = self.binary_(val)
        row = int(str(r_1[0]) + str(r_1[3]), 2)
        col = int(str(r_1[1]) + str(r_1[2]), 2)
```

```
            val = self.S1[row][col]
            str_r = self.binary_(val)
            r_ = [int(str_l[i]) for i in range(2)] + [int(str_r[i]) for i in range(2)]
            r_p4 = [r_[i - 1] for i in self.P4]
            l = [l[i] ^ r_p4[i] for i in range(4)]
            output = l + r
            return output

        def swap(self, array, n):
            l = array[:n]
            r = array[n:]
            output = r + l
            return output

        def decryption(self, ar):
            arr = [ar[i - 1] for i in self.IP]
            arr1 = self.function_(arr, self.key2)
            after_swap = self.swap(arr1, len(arr1) // 2)
            arr2 = self.function_(after_swap, self.key1)
            decrypted = [arr2[i - 1] for i in self.IP_inv]
            return decrypted

    def main():
        obj = SDES()
        obj.key_generation()
        plaintext = [1, 0, 0, 1, 0, 1, 1, 1]
        print("Your plain Text is :", plaintext)
        ciphertext = obj.encryption(plaintext)
        print("Your cipher Text is :", ciphertext)
        decrypted = obj.decryption(ciphertext)
        print("Your decrypted Text is :", decrypted)

    if __name__ == "__main__":
        main()
```

```
Your Key-1 : [1, 0, 1, 0, 0, 1, 0, 0]
Your Key-2 : [1, 0, 0, 1, 0, 0, 1, 0]
Your plain Text is : [1, 0, 0, 1, 0, 1, 1, 1]
Your cipher Text is : [1, 0, 1, 1, 1, 0, 0, 0]
Your decrypted Text is : [1, 0, 0, 1, 0, 1, 1, 1]
```