# Assignment No 1

**Title-** Implementation of S-DES
**Name:** Thorve Avishkar Shrikrushna
**Roll No.:** 63

## Program:

```
from Crypto.Cipher import DES


# Pad or truncate the plaintext to a multiple of 8 bytes (DES block size)
def pad(text):
    while len(text) % 8 != 0:
        text += b' '
    return text


# Encrypt plaintext using S-DES
def sdes_encrypt(plaintext, key):
    cipher = DES.new(key, DES.MODE_ECB)
    plaintext = pad(plaintext)
    return cipher.encrypt(plaintext)


# Decrypt ciphertext using S-DES
def sdes_decrypt(ciphertext, key):
    cipher = DES.new(key, DES.MODE_ECB)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext.rstrip()  # Remove padding


# Example usage:
plaintext = b'Hello World by Swami Aher!'
key = b'01234567'  # 8-byte key (64 bits)


# Encryption
encrypted = sdes_encrypt(plaintext, key)
print("Encrypted:", encrypted)
```

```
# Decryption

decrypted = sdes_decrypt(encrypted, key)

print("Decrypted:", decrypted.decode())
```

**Output:**

PS C:\Users\Computer\Desktop\Sem6\Mp> python CS1.py

Encrypted:
b'$\xd6d\x9fK)\xa8\xf4\xa8iF;\xdf\x02\x07\xb6\x80M\x17\xf6\xe4\x1c\xbfg\x03\xe0}\xf3m\xa2+l'

Decrypted: Hello World by Swami Aher!

# Assignment No 2

**Title:** Implementation of S-AES
**Name:** Thorve Avishkar Shrikrushna
**Roll No.:** 63

## Program:

```
# Install the required library (if not installed)
# pip install pycryptodome

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

def encrypt(plaintext, key):
    # Create an AES cipher object with the key and AES.MODE_ECB mode
    cipher = AES.new(key, AES.MODE_ECB)
    # Pad the plaintext and encrypt it
    ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))
    return ciphertext

def decrypt(ciphertext, key):
    # Create an AES cipher object with the key and AES.MODE_ECB mode
    cipher = AES.new(key, AES.MODE_ECB)
    # Decrypt the ciphertext and remove the padding
    decrypted_data = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return decrypted_data

# Example usage
plaintext = b"Artificial Intelligance & Data Science "
key = get_random_bytes(32)  # Generating a random 32-byte key

print("Key:", key.hex())

# Encryption
encrypted_data = encrypt(plaintext, key)
```

```
print("Encrypted data:", encrypted_data)

# Decryption
decrypted_data = decrypt(encrypted_data, key)
print("Decrypted data:", decrypted_data.decode())
```

**Output:**

PS C:\Users\Computer\Desktop\Sem6\Mp> python CS2.py

Key: 53b3dd27c954ba354f3d6d345860b423f26ee3fbeef2b8600219c63fb201bed6

Encrypted data: b'\x17Z\xf7,\x82\x85\xcdv\xc1\xa3~\xe1).\x943G\xd1\x19\xb2]\xa1#E\xc5\xf4\xfekfI\xd1\x86N]u}\tH\x8a\xb5\xc5\xb4\xcc\xdc\x82\x06DJ'

Decrypted data: Artificial Intelligance & Data Science

# Assignment No 3

**Title:** Implementation of Diffie-Hellman key exchange
**Name:** Thorve Avishkar Shrikrushna
**Roll No.:** 63

## Program:

```
import random


class DHKE:
    def __init__(self, G, P):
        self.G_param = G
        self.P_param = P

    def generate_private_key(self):
        self.private_key = random.randrange(start=1, stop=10, step=1)

    def generate_public_key(self):
        self.public_key = pow(self.G_param, self.private_key) % self.P_param

    def exchange_key(self, other_public):
        self.shared_key = pow(other_public, self.private_key) % self.P_param

# Simulating the Key Exchange between two entities: Alice and Bob
Alice = DHKE(5, 22)
Bob = DHKE(5, 22)

Alice.generate_private_key()
Bob.generate_private_key()

print("------------Private Keys------------------\n")
print("Alice Private Key Generated is ", Alice.private_key, "\n")
print("Bob Private Key Generated is ", Bob.private_key, "\n")
```

```
print("------------End of Private Keys------------\n\n")

Alice.generate_public_key()
Bob.generate_public_key()

print("------------Public Keys------------------\n")
print("Alice Public Key Generated is ", Alice.public_key, '\n')
print("Bob Public Key Generated is ", Bob.public_key, '\n')
print("------------End of Public Keys-----------\n\n")

# Alice & Bob Exchange each other's keys now.
Alice.exchange_key(Bob.public_key)
Bob.exchange_key(Alice.public_key)

print("------------Shared Key Derived------------------\n")
print("Shared Key Generated now by Alice : ", Alice.shared_key, '\n')
print("Shared Key Generated now by Bob : ", Bob.shared_key, '\n')
print("------------End of Shared Key Derived-----------\n")
```

**Output:**

```
PS C:\Users\Computer\Desktop\Sem6\Mp> python CS3.py
------------Private Keys------------------

Alice Private Key Generated is  3

Bob Private Key Generated is  9

------------End of Private Keys------------



------------Public Keys------------------
```

Alice Public Key Generated is  15

Bob Public Key Generated is  9

------------End of Public Keys-----------

------------Shared Key Derived------------------

Shared Key Generated now by Alice :  3

Shared Key Generated now by Bob :  3

------------End of Shared Key Derived-----------

# Assignment No 4

**Title:** Implementation of RSA.
**Name:** Thorve Avishkar Shrikrushna
**Roll No.:** 63

## Program:

```
import math


# Prime numbers

p = 3

q = 7


# Compute n

n = p * q

print("n =", n)


# Compute Euler's Totient Function (phi)

phi = (p - 1) * (q - 1)


# Choose e (public exponent)

e = 2

while e < phi:

    if math.gcd(e, phi) == 1:  # Check if e is coprime with phi

        break

    else:

        e += 1


print("e =", e)


# Compute d (private exponent)

k = 2  # Multiplier for extended calculation

d = ((k * phi) + 1) / e  # Compute modular inverse of e

print("d =", d)


# Public and Private Keys
```

```
print(f'Public key: ({e}, {n})')
print(f'Private key: ({d}, {n})')

# Plain text message
msg = 12
print(f'Original message: {msg}')

# Encryption
C = pow(msg, e)
C = math.fmod(C, n)
print(f'Encrypted message: {C}')

# Decryption
M = pow(C, d)
M = math.fmod(M, n)
print(f'Decrypted message: {M}')
```

**Output:**

```
PS C:\Users\Computer\Desktop\Sem6\Mp> python CS4.py
n = 21
e = 5
d = 5.0
Public key: (5, 21)
Private key: (5.0, 21)
Original message: 12
Encrypted message: 3.0
Decrypted message: 12.0
```

# Assignment No 5

**Title:** Implementation of ECC algorithm.
**Name:** Thorve Avishkar Shrikrushna
**Roll No.:** 63

## Program:

```
# Install the required library (if not installed)

# pip install tinyec


from tinyec import registry

import secrets


# Choose an elliptic curve

curve = registry.get_curve('brainpoolP256r1')


def compress_point(point):

    return hex(point.x) + hex(point.y % 2)[2:]


def ecc_calc_encryption_keys(pubKey):

    ciphertextPrivKey = secrets.randbelow(curve.field.n)

    ciphertextPubKey = ciphertextPrivKey * curve.g

    sharedECCKey = pubKey * ciphertextPrivKey

    return (sharedECCKey, ciphertextPubKey)


def ecc_calc_decryption_key(privKey, ciphertextPubKey):

    sharedECCKey = ciphertextPubKey * privKey

    return sharedECCKey


# Generate ECC private and public keys

privKey = secrets.randbelow(curve.field.n)

pubKey = privKey * curve.g


print("Private key:", hex(privKey))

print("Public key:", compress_point(pubKey))
```

```
# Encryption
(encryptKey, ciphertextPubKey) = ecc_calc_encryption_keys(pubKey)
print("Ciphertext pubKey:", compress_point(ciphertextPubKey))
print("Encryption key:", compress_point(encryptKey))

# Decryption
decryptKey = ecc_calc_decryption_key(privKey, ciphertextPubKey)
print("Decryption key:", compress_point(decryptKey))
```

**Output:**

PS C:\Users\Computer\Desktop\Sem6\Mp> python CS5.py

Private key: 0x921ed31558aa514074e244c1165fb7b023f01000339116690e79a6178ef57288

Public key: 0x84e13ef392c62ce35c031f3db31464f3eefd3fbfbb9c26f5cc6103003e29e4d40

Ciphertext pubKey:
0x8e0274a9b9d27bca0d3421858873a4d3fa4a31139003eaf5c581052d139b6bba0

Encryption key: 0x6ad94496ce21767734062266f0a09fdd03f706f17c53e5c006fd2360c84ea6371

Decryption key: 0x6ad94496ce21767734062266f0a09fdd03f706f17c53e5c006fd2360c84ea6371