1.Implementation of Symmetric cipher algorithm AES

import hashlib

# Generate S-box from SHA256 of a constant string

def generate\_sbox():

seed = b"AES\_SBOX\_GENERATION\_SEED"

hash\_obj = hashlib.sha256(seed)

hash\_bytes = hash\_obj.digest()

# Create a permutation of 0-255 using the hash

sbox = list(range(256))

for i in range(256):

swap\_idx = (i + hash\_bytes[i % 32]) % 256

sbox[i], sbox[swap\_idx] = sbox[swap\_idx], sbox[i]

return sbox

Sbox = generate\_sbox()

InvSbox = [Sbox.index(i) for i in range(256)]

Rcon = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1B,0x36]

def sb(s): return [Sbox[b] for b in s]

def isb(s): return [InvSbox[b] for b in s]

def sr(s): return [s[0],s[5],s[10],s[15],s[4],s[9],s[14],s[3],s[8],s[13],s[2],s[7],s[12],s[1],s[6],s[11]]

def isr(s): return [s[0],s[13],s[10],s[7],s[4],s[1],s[14],s[11],s[8],s[5],s[2],s[15],s[12],s[9],s[6],s[3]]

def xt(a): return ((a<<1)^0x1B)&0xFF if a&0x80 else (a<<1)&0xFF

def mc(s):

for i in range(0,16,4):

s0,s1,s2,s3 = s[i],s[i+1],s[i+2],s[i+3]

t = s0^s1^s2^s3

s[i+0] ^= t^xt(s0^s1)

s[i+1] ^= t^xt(s1^s2)

s[i+2] ^= t^xt(s2^s3)

s[i+3] ^= t^xt(s3^s0)

return s

def mul(a, b):

p = 0

for \_ in range(8):

if b & 1: p ^= a

hi = a & 0x80

a = (a << 1) & 0xFF

if hi: a ^= 0x1B

b >>= 1

return p

def imc(s):

for i in range(0,16,4):

s0,s1,s2,s3 = s[i],s[i+1],s[i+2],s[i+3]

s[i+0] = mul(s0,0x0e)^mul(s1,0x0b)^mul(s2,0x0d)^mul(s3,0x09)

s[i+1] = mul(s0,0x09)^mul(s1,0x0e)^mul(s2,0x0b)^mul(s3,0x0d)

s[i+2] = mul(s0,0x0d)^mul(s1,0x09)^mul(s2,0x0e)^mul(s3,0x0b)

s[i+3] = mul(s0,0x0b)^mul(s1,0x0d)^mul(s2,0x09)^mul(s3,0x0e)

return s

def ark(s, k): return [a^b for a,b in zip(s,k)]

def ke(k):

w = [list(k[i:i+4]) for i in range(0,16,4)]

for i in range(4,44):

t = w[i-1][:]

if i%4==0:

t = t[1:]+t[:1]

t = [Sbox[b] for b in t]

t[0] ^= Rcon[i//4-1]

w.append([a^b for a,b in zip(w[i-4],t)])

return [bytes(sum(w[i\*4:i\*4+4],[])) for i in range(11)]

def enc(b, k):

s, rk = list(b), ke(k)

s = ark(s, rk[0])

for i in range(1,10): s = ark(mc(sr(sb(s))), rk[i])

return bytes(ark(sr(sb(s)), rk[10]))

def dec(b, k):

s, rk = list(b), ke(k)

s = ark(s, rk[10])

s = isr(isb(s))

for i in range(9,0,-1):

s = ark(s, rk[i])

s = imc(s)

s = isr(isb(s))

return bytes(ark(s, rk[0]))

def pad(d):

pad\_len = 16 - (len(d) % 16)

return d + bytes([pad\_len] \* pad\_len)

def unpad(d):

pad\_len = d[-1]

return d[:-pad\_len]

def enc\_ecb(p, k):

p = pad(p)

return b''.join(enc(p[i:i+16], k) for i in range(0,len(p),16))

def dec\_ecb(c, k):

return unpad(b''.join(dec(c[i:i+16], k) for i in range(0,len(c),16)))

key = b"\x2b\x7e\x15\x16\x28\xae\xd2\xa6\xab\xf7\x15\x88\x09\xcf\x4f\x3c"

plaintext = b"Hiiiiiii my name is Gayathri"

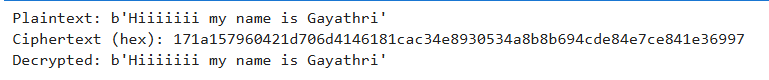
ciphertext = enc\_ecb(plaintext, key)

decrypted = dec\_ecb(ciphertext, key)

print(f"Plaintext: {plaintext}")

print(f"Ciphertext (hex): {ciphertext.hex()}")

print(f"Decrypted: {decrypted}")

output: 

1.Implementation of Symmetric cipher algorithm RC4

def rc4\_init(key: bytes):

"""Key Scheduling Algorithm (KSA)"""

key\_length = len(key)

S = list(range(256))

j = 0

for i in range(256):

j = (j + S[i] + key[i % key\_length]) % 256

S[i], S[j] = S[j], S[i]

return S

def rc4\_generate\_keystream(S, length):

"""Pseudo-Random Generation Algorithm (PRGA)"""

i = j = 0

keystream = []

for \_ in range(length):

i = (i + 1) % 256

j = (j + S[i]) % 256

S[i], S[j] = S[j], S[i]

K = S[(S[i] + S[j]) % 256]

keystream.append(K)

return keystream

def rc4\_encrypt(key: bytes, data: bytes) -> bytes:

"""Encrypt/Decrypt using RC4"""

S = rc4\_init(key)

keystream = rc4\_generate\_keystream(S, len(data))

return bytes([d ^ k for d, k in zip(data, keystream)])

if \_\_name\_\_ == "\_\_main\_\_":

key = b"secretkey"

plaintext = b"Hello RC4 Stream Cipher!"

# Encrypt

ciphertext = rc4\_encrypt(key, plaintext)

print("Plaintext:", plaintext)

print("Ciphertext (hex):", ciphertext.hex())

# Decrypt

decrypted = rc4\_encrypt(key, ciphertext)

print("Decrypted:", decrypted)

output:

