program for ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments). In other words, for these three modes, the total number of bits in the plaintext must be a positive multiple of the block (or segment) size. One common method of padding, if needed, consists of a 1 bit followed by as few zero bits, possibly none, as are necessary to complete the final block. It is considered good practice for the sender to pad every message, including messages in which the final message block is already complete. What is the motivation for including a padding block when padding is not needed?

# Simple demo: ECB, CBC, and CFB modes with 1-bit padding

# (No external libraries used)

def xor\_bytes(a, b):

return bytes([x ^ y for x, y in zip(a, b)])

def simple\_encrypt\_block(block, key):

"""Tiny fake block cipher (for demonstration only)"""

return bytes([(b + key[i % len(key)]) % 256 for i, b in enumerate(block)])

def simple\_decrypt\_block(block, key):

"""Inverse of simple\_encrypt\_block"""

return bytes([(b - key[i % len(key)]) % 256 for i, b in enumerate(block)])

def bit\_padding(data, block\_size=8):

"""Add 1 bit then 0 bits to complete final block"""

data\_bits = ''.join(f'{byte:08b}' for byte in data)

data\_bits += '1' # Add one '1' bit

while len(data\_bits) % (block\_size \* 8) != 0:

data\_bits += '0'

padded = int(data\_bits, 2).to\_bytes(len(data\_bits) // 8, byteorder='big')

return padded

def remove\_bit\_padding(data):

"""Remove 1-bit padding"""

bits = ''.join(f'{byte:08b}' for byte in data)

last\_one = bits.rfind('1')

bits = bits[:last\_one]

if not bits:

return b""

return int(bits, 2).to\_bytes((len(bits) + 7) // 8, byteorder='big')

# === ECB Mode ===

def ecb\_encrypt(data, key):

data = bit\_padding(data)

ciphertext = b""

for i in range(0, len(data), 8):

ciphertext += simple\_encrypt\_block(data[i:i+8], key)

return ciphertext

def ecb\_decrypt(data, key):

plaintext = b""

for i in range(0, len(data), 8):

plaintext += simple\_decrypt\_block(data[i:i+8], key)

return remove\_bit\_padding(plaintext)

# === CBC Mode ===

def cbc\_encrypt(data, key, iv):

data = bit\_padding(data)

ciphertext = b""

prev = iv

for i in range(0, len(data), 8):

block = xor\_bytes(data[i:i+8], prev)

enc = simple\_encrypt\_block(block, key)

ciphertext += enc

prev = enc

return ciphertext

def cbc\_decrypt(data, key, iv):

plaintext = b""

prev = iv

for i in range(0, len(data), 8):

dec = simple\_decrypt\_block(data[i:i+8], key)

plaintext += xor\_bytes(dec, prev)

prev = data[i:i+8]

return remove\_bit\_padding(plaintext)

# === CFB Mode ===

def cfb\_encrypt(data, key, iv):

data = bit\_padding(data)

ciphertext = b""

shift\_reg = iv

for i in range(0, len(data), 8):

enc = simple\_encrypt\_block(shift\_reg, key)

c\_block = xor\_bytes(data[i:i+8], enc)

ciphertext += c\_block

shift\_reg = c\_block

return ciphertext

def cfb\_decrypt(data, key, iv):

plaintext = b""

shift\_reg = iv

for i in range(0, len(data), 8):

enc = simple\_encrypt\_block(shift\_reg, key)

p\_block = xor\_bytes(data[i:i+8], enc)

plaintext += p\_block

shift\_reg = data[i:i+8]

return remove\_bit\_padding(plaintext)

# --- Main Program ---

key = b"key12345"

iv = b"initvect"

plaintext = input("Enter plaintext: ").encode()

# Encrypt & Decrypt in all three modes

for mode, enc\_func, dec\_func in [

("ECB", lambda d: ecb\_encrypt(d, key), lambda d: ecb\_decrypt(d, key)),

("CBC", lambda d: cbc\_encrypt(d, key, iv), lambda d: cbc\_decrypt(d, key, iv)),

("CFB", lambda d: cfb\_encrypt(d, key, iv), lambda d: cfb\_decrypt(d, key, iv))

]:

ciphertext = enc\_func(plaintext)

decrypted = dec\_func(ciphertext)

print(f"\n=== {mode} Mode ===")

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

print("Decrypted text:", decrypted.decode(errors='ignore'))

# --- Explanation ---

print("\n--- Explanation ---")

print("Even if plaintext fits exactly in a block, padding adds an extra block.")

print("This avoids ambiguity: the receiver always knows how to remove padding correctly.")

print("✅ Motivation: Prevent confusion between 'padded' and 'unpadded' messages of the same size.")

