Problem 1: Explore different modes of operation through manual encryption and decryption. Objective: Gain practical insights into the modes of operation discussed in class by manually encrypting and decrypting a given plaintext. Task Details:

1. Encryption Setup: a) Use a hypothetical block cipher with a block length of 4, defined as $Ek(b1b2b3b4)$ = $(b2b3b1b4)$. b) Convert English plaintext into a bit string using the table provided (A=0000 to P=1111). Assume we have a language that uses 16 letters only. If we want a more realistic exercise, we can have block size of 5 bits that can represent 32 cases (more than 26 letters) or 8 bits that use the ASCII.

2. Encryption Modes: Encrypt the plaintext 'FOO' using the following modes. Convert the final ciphertexts into letters. Show your work a) ECB (Electronic Codebook) b) CBC (Cipher Block Chaining) with IV=1010 c) CTR (Counter) with ctr=1010

In [ ]:
```
Encrypt the plaintext 'FOO' using "a) ECB (Electronic Codebook)"
```

Convert Plaintext to Bit String using plaintext -'FOO' needs to be converted to a bit string using the provided table

F = 0101 O = 1110 So, 'FOO' = 0101 1110 1110.

Given that our block cipher operates on 4-bit blocks, we'll handle 'FOO' as three separate blocks, though the last block might need padding if the mode requires it. However, since each letter perfectly fits into the 4-bit representation, we'll proceed without padding for simplicity.

In [ ]:
```
The plaintext 'FOO'.
Ek(b1b2b3b4)=(b2b3b1b4)
meaning the second and third bits swap places with the first and fourth bits
A mapping from characters to 4-bit representations, where F = 0101 and O = 1
Block 1: 'F' = 0101
Apply the block cipher's permutation rule
This changes '0101' (F) to '1010'.
Here, b1=0, b2=1, b3=0, and b4=1
After applying the permutation, we get '1010'.


Block 2: 'O' = 1110
The same permutation rule is applied to '1110' (O).
Here, b1=1, b2=1, b3=1, and b4=0
the permutation yields '1101'.
This is because the second and third bits (both 1s) stay in their positions
the fourth bit remains unchanged.


Block 3: Another 'O' = 1110
Applying the encryption to another 'O' yields the same result as Block 2, gi
```

```
Finally :

ECB mode encrypts each block individually.

Block 1 ('F' = 0101) becomes 1010 (applying
Ek(b1b2b3b4)=b2b3b1b4).
Block 2 ('O' = 1110) becomes 1101.
Block 3 ('O' = 1110) becomes 1101 again.
Thus, the ciphertext in ECB is 1010 1101 1101.
```

In [ ]:
```
b) CBC (Cipher Block Chaining) with IV = 1010
```

In [ ]:
```
Cipher Block Chaining (CBC) mode is a more complex and secure mode of operat

Encryption Process for 'FOO' using CBC Mode with IV = 1010
Let's break down the encryption of 'FOO' (with 'F' = 0101 and 'O' = 1110) ir

Initial Setup:
Plaintext 'FOO' = 'F' (0101) 'O' (1110) 'O' (1110)
Initialization Vector (IV) = 1010

Step-by-Step Encryption:
Step 1: Encrypt the First Block ('F' = 0101)
XOR the First Plaintext Block with IV: First, XOR 'F' (0101) with the IV (10
Result: 1111
Encrypt the Result: Apply the block cipher's permutation rule to 1111.
First Block Ciphertext: 1111
This encrypted block will be used as the "previous ciphertext block" for the

Step 2: Encrypt the Second Block ('O' = 1110)
XOR the Second Plaintext Block with the Previous Ciphertext Block: XOR 'O' (
Result: 0001
Encrypt the Result: Apply the block cipher's permutation rule to 0001.
Second Block Ciphertext: 0010


Step 3: Encrypt the Third Block (Another 'O' = 1110)
Repeat the Process for the Third Block, using the second block's ciphertext
XOR Operation: 1110 XOR 0010 = 1100.
Encrypt the Result: Apply the permutation to 1100.
Third Block Ciphertext: 1001

Final Ciphertext Blocks: 1111 0010 1001
```

In [ ]:
```
c) CTR (Counter) with ctr=1010 ??
```

In [ ]:
```
In CTR mode, encryption is done by encrypting a counter value and then XORir

Given:

Plaintext 'FOO' = 'F' (0101) 'O' (1110) 'O' (1110)
Initial counter (ctr) = 1010

Step-by-Step Encryption:
```

```
Step 1: Encrypt the First Block ('F' = 0101)
Encrypt the Counter: Encrypt the initial counter value (1010) using the bloc
XOR the Encrypted Counter with Plaintext: XOR the encrypted counter (1101) w
Result: 1000
Increment the Counter: After using it, increment the counter for the next bl

Step 2: Encrypt the Second Block ('O' = 1110)
Encrypt the Incremented Counter (1011): This gives us 1110 after applying th
XOR with the Second Plaintext Block: XOR 1110 with 'O' (1110).
Result: 0000
Increment the Counter: The counter increments to 1100 for the next block.

Step 3: Encrypt the Third Block (Another 'O' = 1110)
Encrypt the Further Incremented Counter (1100): Applying the permutation rul
XOR with the Third Plaintext Block: XOR 1001 with 'O' (1110).
Result: 0111

Counter for Subsequent Blocks: If there were more blocks, we would continue


Through CTR mode encryption, 'FOO' is encrypted without directly encrypting

Final Ciphertext Blocks: 1000 0000 0111
```

In [ ]:

```
Encryption Results
After performing the encryption for each mode, we have the following ciphert

ECB Ciphertext Bits: ['1001', '1110', '1110']
CBC Ciphertext Bits: ['1111', '0001', '1111']
CTR Ciphertext Bits: ['0011', '1001', '0100']
Converting Ciphertexts into Letters
Using the provided conversion table:

ECB Ciphertext: J (1001), O (1110), O (1110)
CBC Ciphertext: P (1111), B (0001), P (1111)
CTR Ciphertext: D (0011), J (1001), E (0100)
```

In [ ]:

```
e) Manually decrypt each ciphertext to recover the original plaintext. Show
```

In [ ]:

```
ECB (Electronic Codebook) Mode Decryption
In ECB mode, since each block is encrypted independently, the decryption pro
the decryption function is essentially the same due to the nature of the per

Steps to Decrypt ECB:

Apply the Decryption Function: For each ciphertext block, apply the permutat
directly to decrypt it.
Convert Back to Text: Translate the resulting bit strings back into their co




CBC (Cipher Block Chaining) Mode Decryption
CBC mode decryption requires the previous ciphertext block to decrypt the cu
```

```
Steps to Decrypt CBC:
Decrypt Each Block: Apply the decryption function to each ciphertext block t
XOR with Previous Ciphertext Block or IV: XOR the decrypted block with the p
Repeat for Each Block: Continue the process for each block in sequence.





CTR (Counter) Mode Decryption
CTR mode decryption is identical to its encryption process, as it involves X

Steps to Decrypt CTR:
Encrypt the Counter: For each block, encrypt the counter value using the sam
XOR with Ciphertext: XOR the encrypted counter with the ciphertext block to 
Increment the Counter: Increment the counter for each block, ensuring the sa

Example :
ciphertext blocks:

ECB Ciphertext Block: '1010' (Assuming it represents 'J')
Decrypt using the cipher's permutation to get '0101' ('F').

CBC Ciphertext Block: '1111', with IV = '1010'
Decrypt to '1111', XOR with IV ('1010') to get '0101' ('F').


CTR Ciphertext Block: '1000', with initial counter = '1010'
Encrypt counter ('1010') to get '1101', XOR with '1000' to recover '0101' ('


Decryption Task
The decryption process essentially reverses the encryption steps for each mo

a)ECB Decryption
JN (1010 1101) => Decrypt each block => FOO
b) CBC Decryption with IV=1010
O (1110) => Decrypt 1110 to 1111 => XOR with IV (1010) => 0101 => F
c) CTR Decryption with ctr=1010
I (1000) => Encrypt counter (1010) => 1101, then XOR with 1000 => 0101 => F
This simplification provides a conceptual understanding. In real scenarios,
```

In [ ]:
```
def encrypt_block(block):
    # This just rearranges our bits in a new order to mix them up
    return block[1] + block[2] + block[0] + block[3]

def xor_blocks(block1, block2):
    # This is a cool way to mix two bits together so they become a new secre
    return ''.join(str(int(a) ^ int(b)) for a, b in zip(block1, block2))
```

In [2]:
```
# For ECB, just scramble each block on its own
ecb_ciphertext_bits = [encrypt_block(block) for block in plaintext_bits]

# For CBC, start with an IV and chain the scrambles
```

```
cbc_iv = '1010'
cbc_ciphertext_bits = []
prev_cipher_block = cbc_iv
for block in plaintext_bits:
    xored_block = xor_blocks(block, prev_cipher_block)
    encrypted_block = encrypt_block(xored_block)
    cbc_ciphertext_bits.append(encrypted_block)
    prev_cipher_block = encrypted_block

# For CTR, scramble a counter and mix it with each block
ctr = '1010'
ctr_ciphertext_bits = []
for block in plaintext_bits:
    encrypted_ctr = encrypt_block(ctr)
    xored_block = xor_blocks(block, encrypted_ctr)
    ctr_ciphertext_bits.append(xored_block)
    ctr = format((int(ctr, 2) + 1), '04b')

(ecb_ciphertext_bits, cbc_ciphertext_bits, ctr_ciphertext_bits)
```

Out[2]:

```
(['1001', '1110', '1110'], ['1111', '0001', '1111'], ['0011', '1001', '010
0'])
```

In [8]:

```
# Define the block cipher's permutation function
def encrypt_block(block):
    # Since encryption and decryption are the same for our hypothetical ciph
    # we apply the same permutation: E_k(b1b2b3b4) = (b2b3b1b4)
    return block[1] + block[2] + block[0] + block[3]

# XOR two blocks of bits
def xor_blocks(block1, block2):
    return ''.join(str(int(a) ^ int(b)) for a, b in zip(block1, block2))

# Decryption for ECB mode
def decrypt_ecb(ciphertext_bits):
    # Apply the block cipher's permutation to each ciphertext block
    return [encrypt_block(block) for block in ciphertext_bits]

# Decryption for CBC mode
def decrypt_cbc(ciphertext_bits, iv):
    decrypted_bits = []
    prev_cipher_block = iv
    for block in ciphertext_bits:
        # Decrypt the block and XOR with the previous ciphertext block or IV
        decrypted_block = encrypt_block(block)
        plaintext_block = xor_blocks(decrypted_block, prev_cipher_block)
        decrypted_bits.append(plaintext_block)
        prev_cipher_block = block
    return decrypted_bits

# Decryption for CTR mode
def decrypt_ctr(ciphertext_bits, initial_ctr):
    decrypted_bits = []
    ctr = initial_ctr
    for block in ciphertext_bits:
        # Encrypt the counter, then XOR with the ciphertext block
```

```
        encrypted_ctr = encrypt_block(ctr)
        plaintext_block = xor_blocks(block, encrypted_ctr)
        decrypted_bits.append(plaintext_block)
        # Increment the counter
        ctr = format((int(ctr, 2) + 1), '04b')
    return decrypted_bits

# Example usage with placeholder ciphertext bits, IV, and initial counter
ecb_ciphertext_bits = ['1001', '1110', '1110']  # Placeholder for ECB cipher
cbc_ciphertext_bits = ['1111', '0001', '1111']  # Placeholder for CBC cipher
ctr_ciphertext_bits = ['0011', '1001', '0100']  # Placeholder for CTR cipher
iv = '1010'  # Initial Vector for CBC
initial_ctr = '1010'  # Initial counter for CTR

# Perform decryption
ecb_decrypted = decrypt_ecb(ecb_ciphertext_bits)
cbc_decrypted = decrypt_cbc(cbc_ciphertext_bits, iv)
ctr_decrypted = decrypt_ctr(ctr_ciphertext_bits, initial_ctr)

# Output the decrypted bits
print("ECB Decrypted:", ecb_decrypted)
print("CBC Decrypted:", cbc_decrypted)
print("CTR Decrypted:", ctr_decrypted)
```

```
ECB Decrypted: ['0011', '1110', '1110']
CBC Decrypted: ['0101', '1110', '1110']
CTR Decrypted: ['0101', '1110', '1110']
```

In [ ]:

```
```