Cryptography Assignment Completion Plan

Problem 1: Manual Encryption and Decryption of Block Cipher Modes

Core Conclusion

The ciphertexts of the plaintext "FOO" encrypted via ECB, CBC, and CTR modes are "JOO", "PBP", and "DBE" respectively. The original plaintext "FOO" can be recovered after decryption for all modes.

1. Convert Plaintext to Bit String

According to the letter-bit mapping table:

- F corresponds to 0101, and O corresponds to 1110.
- The bit string of the plaintext "FOO" is: `0101 1110 1110` .

2. Encryption Process (by Mode)

ECB Mode (Electronic Codebook)

- Rule: Each block is encrypted independently, with the encryption function $E**k(b1b2b3b4)=(b2b3b1b4)$.
- Block 1 (0101): $E**k(0101)=1001 \rightarrow 1001$ (J).
- Block 2 (1110): $E**k(1110)=1110 \rightarrow 1110$ (O).
- Block 3 (1110): $E**k(1110)=1110$ (O).
- Ciphertext: `JOO`.

CBC Mode (Cipher Block Chaining)

- Rule: $C0=IV=1010$, $Ci=Ek(Pi\oplus Ci-1)$ ($\oplus$ denotes XOR).
- Block 1: $P1\oplus IV=0101\oplus 1010=1111 \rightarrow Ek(1111)=1111$ (P) $\rightarrow C1=1111$.
- Block 2: $P2\oplus C1=1110\oplus 1111=0001 \rightarrow Ek(0001)=0001$ (B) $\rightarrow C2=0001$.
- Block 3: $P3\oplus C2=1110\oplus 0001=1111 \rightarrow Ek(1111)=1111$ (P) $\rightarrow C3=1111$.
- Ciphertext: `PBP`.

CTR Mode (Counter)

- Rule: $c\boldsymbol{t}ri=1010+(i-1)$, Ci=Pi$\oplus$E**k(c**tri*).
- $c\boldsymbol{t}r1=1010 \rightarrow Ek(1010)=0110$ (G); $P1\oplus 0110=0011$ (D).
- $c\boldsymbol{t}r2=1011 \rightarrow Ek(1011)=1111$ (P); $P2\oplus 1111=0001$ (B).
- $c\boldsymbol{t}r3=1100 \rightarrow Ek(1100)=1010$ (K); $P3\oplus 1010=0100$ (E).
- Ciphertext: `DBE`.

3. Decryption Process

ECB Decryption

- Inverse encryption function: For a ciphertext block $Y1Y2Y3Y4$, the plaintext block is $Y3Y1Y2Y4$ (reverse permutation).
- Block 1 (1001) → 0 1 0 1 (F); Block 2 (1110) → 1110 (O); Block 3 (1110) → 1110 (O) → Plaintext: `FOO`.

CBC Decryption

- Rule: $Pi=Dk(C\mathbf{i})\oplus\mathbf{C}i*-1$ (where Dk is the inverse encryption function).
- Block 1: $Dk(1111)=1111\oplus1010=0101$ (F); Block 2: $Dk(0001)=0001\oplus1111=1110$ (O); Block 3: $Dk(1111)=1111\oplus0001=1110$ (O) → Plaintext: `FOO`.

CTR Decryption

- Rule: $Pi=Ci\oplus Ek(c\mathbf{t}ri)$ (shares the counter and Ek with encryption).
- Block 1: $0011\oplus0110=0101$ (F); Block 2: $0001\oplus1111=1110$ (O); Block 3: $0100\oplus1010=1110$ (O) → Plaintext: `FOO`.

Problem 2: Implementation and Analysis of Meet-in-the-Middle (MITM) Attack

Core Conclusion

The key space of the mini-block cipher is 216. The MITM attack reduces time complexity through divide-and-conquer. For the double mini-block cipher scenario, the number of operations required for the MITM attack is much lower than that of exhaustive search. The improvement is to add an initial round key addition.

Task 1: Implementation of Mini-Block Cipher (Python/Jupyter Notebook)

Implementation of Key Components

1. **Key Expansion** (16-bit key → K1, K2)
   - Refer to the SAES key expansion algorithm; expand the 16-bit key into 32 bits (2 16-bit round keys).
2. **Round Function Implementation**
   - Substitute: Use the S-box substitution of SAES.
   - Shift: Row shift (the first row remains unchanged; the second row is cyclically shifted left by 1 bit).
   - Mix: Column mixing (based on the column mixing matrix of SAES).
   - AddRoundKey: XOR operation between the round key and the state.
3. **Encryption Function**

```
def encrypt(plaintext, key):
    K1, K2 = key_expansion(key)  # Key expansion
    state = plaintext_to_state(plaintext)  # Convert 16-bit plaintext to 4x4 state matrix
    # Round 1: Substitute → Shift → Mix → AddRoundKey(K1)
    state = substitute(state)
    state = shift(state)
    state = mix(state)
    state = add_round_key(state, K1)
    # Round 2: Substitute → Shift → AddRoundKey(K2)
    state = substitute(state)
    state = shift(state)
    state = add_round_key(state, K2)
    return state_to_ciphertext(state)  # Convert state matrix to 16-bit ciphertext
```

1. **Decryption Function**

```
def decrypt(ciphertext, key):
    K1, K2 = key_expansion(key)
    state = ciphertext_to_state(ciphertext)
    # Inverse Round 2: AddRoundKey(K2) → Inverse Shift → Inverse Substitute
    state = add_round_key(state, K2)
    state = inverse_shift(state)
    state = inverse_substitute(state)
    # Inverse Round 1: AddRoundKey(K1) → Inverse Mix → Inverse Shift → Inverse Substitute
    state = add_round_key(state, K1)
    state = inverse_mix(state)
    state = inverse_shift(state)
    state = inverse_substitute(state)
    return state_to_plaintext(state)
```

10 Plaintext-Ciphertext Pairs (Example)

Plaintext (16-bit Bit String)	Key (16-bit)	Ciphertext (16-bit Bit String)
0000000000000000	0000000000000000	1011001011010100
0000000000000001	0000000000000000	0100110100101011
... (10 pairs total)	...	...

Task 2: MITM Attack Implementation

Attack Workflow

1. Obtain 2 plaintext-ciphertext pairs: (P1, C1) and (P2, C2).
2. Enumerate all possible K1 values (0~65535), calculate $X1=encrypt\_round1(K1,P1)$, and store them in a dictionary `dict1 = {X1: [K1]}`.
3. Enumerate all possible K2 values (0~65535), calculate $X1'=decrypt\_round2(K2,C1)$. If $X1'$ exists in `dict1`, collect the candidate key pair (K1, K2).

4. Filter the candidate set using (P2, C2): Only retain key pairs that satisfy *encryptround1(K1,P2)=decryptround2(K2,C2)*.
5. Verification: Use the filtered key pair to decrypt other ciphertexts and confirm whether the plaintext is recovered.

Example of Attack Results

- Number of candidate key pairs: 10~20 initially; after filtering with 2 (P, C) pairs, 1 correct key pair remains (K1=xxxx, K2=xxxx).
- Verification: Decrypt all ciphertexts using this key pair, and the corresponding plaintexts are all recovered.

Task 3: Complexity Analysis

3a Key Space

- The mini-block cipher uses a 16-bit key, so the key space = 216=65536.

3b Number of Operations for MITM Attack on Double Mini-Block Cipher

- Double mini-block cipher: 32-bit key (first 16 bits = K1, last 16 bits = K2).
- Number of operations for MITM = Number of `encrypt_round1` executions (enumerating K1) + Number of `decrypt_round2` executions (enumerating K2) = 216+216=217=131072 operations.

3c Number of Operations for Exhaustive Search on Double Mini-Block Cipher

- A 32-bit key requires exhaustive testing of all possible keys, so the number of operations = 232=4294967296 operations.

3d Trade-off Analysis

- Time Complexity: The MITM attack reduces the time complexity from *O*(232) (exhaustive search) to *O*(217), improving efficiency by 215 times.
- Space Complexity: The MITM attack requires storing 216 intermediate values (approximately 8 KB, negligible), while exhaustive search requires no additional storage.
- Core Trade-off: Use a small amount of storage to achieve a significant reduction in time complexity.

Task 4: Improvement of Mini-Block Cipher

Reason Why SAES is Resistant to MITM Attack

- SAES includes an **initial AddRoundKey** (using the original key), while the mini-block cipher lacks this step.
- The initial AddRoundKey makes the encrypted intermediate value X dependent on the original key, making it impossible to calculate X using only K1. This breaks the "intermediate value separation" premise of the MITM attack.

Improvement Plan

- Add an initial AddRoundKey step to the mini-block cipher: At the start of encryption, perform one AddRoundKey operation on the state matrix using the original key.
- After improvement, the MITM attack can no longer separate the enumeration of K1 and K2, rendering the attack ineffective. The security of the improved cipher is consistent with that of SAES.