Exploring Block Cipher and the Meet-in-the-Middle Attack

Problem 1

1.1 Encryption Setup

1.1.a Block Cipher Definition

Use a hypothetical block cipher with a 4-bit block length, defined as： (E_k(b_1b_2b_3b_4) = (b_2b_3b_1b_4))

This is a simple permutation cipher that only adjusts bit positions without changing bit values.

- Input: 4-bit block (b_1b_2b_3b_4)
- Output: (b_2b_3b_1b_4)
- Core operation: Move the first bit to the third position

Example:Input: 1010 (i.e., (b_1=1, b_2=0, b_3=1, b_4=0))Output: (b_2b_3b_1b_4 = 0110)

1.1.b Letter-to-Bit Encoding

The plaintext is converted to a bit string using the following mapping (only 16 letters are used):

Letter	Bit String	Letter	Bit String
A	0000	I	1000
B	0001	J	1001
C	0010	K	1010
D	0011	L	1011
E	0100	M	1100
F	0101	N	1101
G	0110	O	1110
H	0111	P	1111

1.2 Encryption Modes (Plaintext: "FOO")

First, convert the plaintext "FOO" to a bit string based on the encoding table:

- F = 0101, O = 1110
- Plaintext bit string: (P_1=0101, P_2=1110, P_3=1110)

1.2.a ECB (Electronic Codebook)

Formula: (C_i = E_k(P_i))Calculation process:

1. (C_1 = E_k(P_1) = E_k(0101) = 0101 \oplus 1100 = 1001) (corresponds to J)
2. (C_2 = E_k(P_2) = E_k(1110) = 1110 \oplus 1100 = 0010) (corresponds to C)
3. (C_3 = E_k(P_3) = E_k(1110) = 1110 \oplus 1100 = 0010) (corresponds to C)

Ciphertext: "JCC"

1.2.b CBC (Cipher Block Chaining) with IV = 1010

Formula: (C_i = E_k(P_i \oplus C_{i-1})) (where (C_0 = IV))Calculation process:

1. Block 1: (C_1 = E_k(P_1 \oplus IV) = E_k(0101 \oplus 1010) = E_k(1111) = 1111 \oplus 1100 = 0011) (corresponds to D)
2. Block 2: (C_2 = E_k(P_2 \oplus C_1) = E_k(1110 \oplus 0011) = E_k(1101) = 1101 \oplus 1100 = 0001) (corresponds to B)
3. Block 3: (C_3 = E_k(P_3 \oplus C_2) = E_k(1110 \oplus 0001) = E_k(1111) = 1111 \oplus 1100 = 0011) (corresponds to D)

Ciphertext: "DBD"

1.2.c CTR (Counter) with ctr = 1010

Formula: (C_i = P_i \oplus E_k(ctr_i)) (where (ctr_i = 1010 + (i-1)))Calculation process:

1. Block 1: (ctr_1 = 1010), (E_k(1010) = 1010 \oplus 1100 = 0110); (C_1 = 0101 \oplus 0110 = 0011) (corresponds to D)
2. Block 2: (ctr_2 = 1011), (E_k(1011) = 1011 \oplus 1100 = 0111); (C_2 = 1110 \oplus 0111 = 1001) (corresponds to J)
3. Block 3: (ctr_3 = 1100), (E_k(1100) = 1100 \oplus 1100 = 0000); (C_3 = 1110 \oplus 0000 = 1110) (corresponds to O)

Ciphertext: "DJO"

1.3 Decryption Task

1.3.a Decrypt Each Ciphertext to Recover "FOO"

ECB Decryption

Formula: (P_i = D_k(C_i) = C_i \oplus K)Calculation process:

1. Block 1: (P_1 = D_k(1001) = 1001 \oplus 1100 = 0101) (corresponds to F)
2. Block 2: (P_2 = D_k(0010) = 0010 \oplus 1100 = 1110) (corresponds to O)
3. Block 3: (P_3 = D_k(0010) = 0010 \oplus 1100 = 1110) (corresponds to O)

Recovered plaintext: "FOO"

CBC Decryption

Formula: (P_i = D_k(C_i) \oplus C_{i-1}) (where (C_0 = IV = 1010))Calculation process:

1. Block 1: (P_1 = D_k(C_1) \oplus IV = (0011 \oplus 1100) \oplus 1010 = 1111 \oplus 1010 = 0101) (corresponds to F)
2. Block 2: (P_2 = D_k(C_2) \oplus C_1 = (0001 \oplus 1100) \oplus 0011 = 1101 \oplus 0011 = 1110) (corresponds to O)
3. Block 3: (P_3 = D_k(C_3) \oplus C_2 = (0011 \oplus 1100) \oplus 0001 = 1111 \oplus 0001 = 1110) (corresponds to O)

Recovered plaintext: "FOO"

CTR Decryption

Formula: (P_i = C_i \oplus E_k(ctr_i)) (same counters as encryption)Calculation process:

1. Block 1: (P_1 = 0011 \oplus E_k(1010) = 0011 \oplus 0110 = 0101) (corresponds to F)
2. Block 2: (P_2 = 1001 \oplus E_k(1011) = 1001 \oplus 0111 = 1110) (corresponds to O)
3. Block 3: (P_3 = 1110 \oplus E_k(1100) = 1110 \oplus 0000 = 1110) (corresponds to O)

Recovered plaintext: "FOO"

Problem 2

2.1 Implementing Mini Block Cipher (16-bit Key & 16-bit Block Size)

2.1.a Core Design & Implementation (Jupyter Notebook)

The mini block cipher is designed based on SAES, with the following core logic (pseudo-code):

Key Expansion:Use a 16-bit original key K to generate two 16-bit round keys (K_1) and (K_2) (following the standard SAES key expansion algorithm, ignoring (K_0)).

Encryption Process:

1. Initialize state with the 16-bit plaintext P
2. Execute Round 1 (using (K_1)): SubBytes → ShiftRows → MixColumns → AddRoundKey((K_1))
3. Execute Round 2 (using (K_2)): SubBytes → ShiftRows → AddRoundKey((K_2))
4. Output the 16-bit ciphertext C

Decryption Process:

1. Initialize state with the 16-bit ciphertext C
2. Execute Inverse Round 2 (using (K_2)): AddRoundKey((K_2)) → InvShiftRows → InvSubBytes
3. Execute Inverse Round 1 (using (K_1)): AddRoundKey((K_1)) → InvMixColumns → InvShiftRows → InvSubBytes
4. Output the 16-bit plaintext P

2.1.b Generate 10 Plaintext-Ciphertext Pairs

Use a fixed 16-bit key (e.g., 0x2b7e) to encrypt 10 distinct 16-bit plaintexts (e.g., (0x0000, 0x0001, ..., 0x0009)), generating 10 valid (Plaintext, Ciphertext) pairs.

2.2 MITM Attack Implementation

2.2.a Attack Strategy

The goal is to recover the 32-bit effective key (composed of two independent 16-bit keys (K_1) and (K_2)) for the ciphertext (C = E*{K_2}(E*{K_1}(P))). The attacker does not know the relationship between (K_1) and (K_2) via key expansion.

Step-by-step implementation:

1. Prepare two known (Plaintext, Ciphertext) pairs: ((P_1, C_1)) and ((P_2, C_2))
2. Forward phase: Enumerate all (2^{16}) possible (K_1) values, calculate the intermediate value (X = encrypt_round1(P_1, K_1)), and store in a lookup table: `table[X] = K_1`
3. Backward phase: Enumerate all (2^{16}) possible (K_2) values, calculate the intermediate value (X' = decrypt_round2(C_1, K_2))
4. Candidate screening: If (X' \in table), the pair ((table[X'], K_2)) is a candidate key pair
5. Verification: Use ((P_2, C_2)) to filter candidates—only retain pairs that satisfy (encrypt_round2(encrypt_round1(P_2, K_1), K_2) = C_2)

2.2.b Attack Results

After verification, the only candidate key pair that passes the test of both (P, C) pairs is the real key. False positives (invalid candidates) are eliminated by the second (P, C) pair.

2.3 Complexity Analysis (vs. Naive Exhaustive Search)

2.3.a Key Space of the Mini Block Cipher

The mini block cipher uses a 16-bit key, so its key space is:(2^{16} = 65,536) possible keys

2.3.b Operation Count for MITM Attack on Double Mini Block Cipher

The double mini block cipher uses a 32-bit key (first 16 bits for (K_1), last 16 bits for (K_2)). The MITM attack requires:

1. Forward phase: (2^{16}) encryption operations (enumerate (K_1))
2. Backward phase: (2^{16}) decryption operations (enumerate (K_2))
3. Matching phase: Negligible lookup operations

Total operations: (2^{16} + 2^{16} = 2^{17} = 131,072)

2.3.c Operation Count for Exhaustive Search on Double Mini Block Cipher

The 32-bit key has a key space of (2^{32}), so the exhaustive search requires:(2^{32} = 4,294,967,296) encryption operations (test all possible keys)

2.3.d Trade-off of the MITM Attack

The MITM attack trades **memory space** for **time efficiency**:

- Exhaustive search: Time complexity (O(2^{32})), Space complexity (O(1))
- MITM attack: Time complexity (O(2^{17})), Space complexity (O(2^{16}))

The attacker uses (2^{16}) memory to store intermediate values, reducing the time complexity from (2^{32}) to (2^{17}) (a 32,768-fold improvement).

2.4 Improvement of the Mini Block Cipher

2.4.a SAES vs. Mini Block Cipher (MITM Vulnerability)

SAES (Secure AES) differs from the mini block cipher by adding an **initial AddRoundKey operation** (using the original key K before Round 1).

SAES encryption process:AddRoundKey(K) → Round 1 (SubBytes/ShiftRows/MixColumns/AddRoundKey((K_1))) → Round 2 (SubBytes/ShiftRows/AddRoundKey((K_2)))

2.4.b Improvement Logic

The initial AddRoundKey(K) in SAES binds the intermediate state to the original key K, rather than directly using round keys (K_1) and (K_2). This breaks the "separability" of intermediate values required by the MITM attack—attackers can no longer independently enumerate (K_1) and (K_2) to calculate matching intermediate values.

Thus, adding the initial AddRoundKey operation (consistent with SAES) makes the mini block cipher resistant to MITM attacks, significantly improving its security.