AES-128

代码框架

以下是使用C语言实现AES-128加密和解密算法的代码。AES-128算法的密钥长度是128位（16字节），需要定义密钥扩展、加密和解密等函数。

AES算法包含四个主要操作：字节代替（SubBytes）、行移位（ShiftRows）、列混淆（MixColumns）和轮密钥加（AddRoundKey）。对于AES-128，加密和解密的过程是由10轮操作组成的。

以下是简化的C代码实现，包含AES-128的加密和解密功能。代码实现了密钥扩展、加密和解密函数。

```
#include <stdio.h>

#include <stdint.h>



#define Nb 4

#define Nk 4

#define Nr 10



// AES S-box

static const uint8_t sbox[256] = {

    // S-box values omitted for brevity

};



// AES inverse S-box

static const uint8_t inv_sbox[256] = {
```

```
    // Inverse S-box values omitted for brevity
};

// Rcon array for key expansion
static const uint8_t Rcon[11] = {
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36
};

// Substitute bytes function for encryption
void SubBytes(uint8_t state[4][4]) {
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            state[i][j] = sbox[state[i][j]];
}

// Inverse Substitute bytes function for decryption
void InvSubBytes(uint8_t state[4][4]) {
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            state[i][j] = inv_sbox[state[i][j]];
}
```

```
// Shift rows operation
void ShiftRows(uint8_t state[4][4]) {
    uint8_t temp;
    // Implement row shifts based on AES ShiftRows
}

// Inverse Shift rows operation
void InvShiftRows(uint8_t state[4][4]) {
    uint8_t temp;
    // Implement row shifts based on AES InvShiftRows
}

// MixColumns operation for encryption
void MixColumns(uint8_t state[4][4]) {
    // Implement MixColumns
}

// Inverse MixColumns operation for decryption
void InvMixColumns(uint8_t state[4][4]) {
    // Implement InvMixColumns
}
```

```
// Add round key operation
void AddRoundKey(uint8_t state[4][4], uint8_t roundKey[4][4]) {
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            state[i][j] ^= roundKey[i][j];
}

// Key expansion
void KeyExpansion(const uint8_t *key, uint8_t w[4 * Nb * (Nr + 1)]) {
    // Key expansion implementation based on AES specification
}

// AES encryption function
void AES_Encrypt(uint8_t *input, const uint8_t *key) {
    uint8_t state[4][4];
    uint8_t roundKeys[4 * Nb * (Nr + 1)];
    KeyExpansion(key, roundKeys);

    // Initial round
    AddRoundKey(state, roundKeys);

    for (int round = 1; round < Nr; round++) {
```

```
            SubBytes(state);
            ShiftRows(state);
            MixColumns(state);
            AddRoundKey(state, &roundKeys[round * 4 * Nb]);
        }

        // Final round (without MixColumns)
        SubBytes(state);
        ShiftRows(state);
        AddRoundKey(state, &roundKeys[Nr * 4 * Nb]);

        // Copy state to output
        for (int i = 0; i < 16; i++) input[i] = state[i % 4][i / 4];
}

// AES decryption function
void AES_Decrypt(uint8_t *input, const uint8_t *key) {
        uint8_t state[4][4];
        uint8_t roundKeys[4 * Nb * (Nr + 1)];
        KeyExpansion(key, roundKeys);

        // Initial round
```

```
    AddRoundKey(state, &roundKeys[Nr * 4 * Nb]);

    for (int round = Nr - 1; round > 0; round--) {
        InvShiftRows(state);
        InvSubBytes(state);
        AddRoundKey(state, &roundKeys[round * 4 * Nb]);
        InvMixColumns(state);
    }

    // Final round (without InvMixColumns)
    InvShiftRows(state);
    InvSubBytes(state);
    AddRoundKey(state, roundKeys);

    // Copy state to output
    for (int i = 0; i < 16; i++) input[i] = state[i % 4][i / 4];
}

int main() {
    uint8_t key[16] = { /* 128-bit AES key */ };
    uint8_t data[16] = { /* 128-bit plaintext */ };
```

```
    printf("Original data:\n");
    for (int i = 0; i < 16; i++) printf("%02x ", data[i]);

    AES_Encrypt(data, key);
    printf("\nEncrypted data:\n");
    for (int i = 0; i < 16; i++) printf("%02x ", data[i]);

    AES_Decrypt(data, key);
    printf("\nDecrypted data:\n");
    for (int i = 0; i < 16; i++) printf("%02x ", data[i]);

    return 0;
}
```

说明

1. **S-box和逆S-box**：用于`SubBytes`和`InvSubBytes`操作，需要将S-box和逆S-box的实际值填充进去。
2. **密钥扩展 (Key Expansion)**：根据AES规范生成轮密钥。
3. **加密过程**：先进行一次`AddRoundKey`操作，然后进行`SubBytes`、`ShiftRows`、`MixColumns`，并再次执行`AddRoundKey`。最后一轮省略`MixColumns`。
4. **解密过程**：与加密相反，需要先使用最后的轮密钥，然后逆序进行`InvShiftRows`、`InvSubBytes`、`InvMixColumns`等操作。

这段代码实现了AES-128的核心功能，但需要填充S-box、逆S-box、`MixColumns`和`InvMixColumns`的具体实现。

填充 ShiftRows

`ShiftRows` 是 AES 算法中的一个步骤，它通过行移位操作混淆字节的位置，增强加密强度。在 `ShiftRows` 操作中，AES的4x4状态矩阵的每一行根据行号进行移位操作：

- 第 0 行保持不变。
- 第 1 行左移 1 字节。
- 第 2 行左移 2 字节。
- 第 3 行左移 3 字节。

以下是 `ShiftRows` 和 `InvShiftRows` 的实现：

```
// ShiftRows operation for encryption
void ShiftRows(uint8_t state[4][4]) {
    uint8_t temp;

    // Row 1: left rotate by 1
    temp = state[1][0];
    state[1][0] = state[1][1];
    state[1][1] = state[1][2];
    state[1][2] = state[1][3];
    state[1][3] = temp;

    // Row 2: left rotate by 2
    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;
```

```
    temp = state[2][1];
    state[2][1] = state[2][3];
    state[2][3] = temp;

    // Row 3: left rotate by 3 (or right rotate by 1)
    temp = state[3][3];
    state[3][3] = state[3][2];
    state[3][2] = state[3][1];
    state[3][1] = state[3][0];
    state[3][0] = temp;
}

// Inverse ShiftRows operation for decryption
void InvShiftRows(uint8_t state[4][4]) {
    uint8_t temp;

    // Row 1: right rotate by 1
    temp = state[1][3];
    state[1][3] = state[1][2];
    state[1][2] = state[1][1];
    state[1][1] = state[1][0];
    state[1][0] = temp;
```

```
    // Row 2: right rotate by 2
    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;
    temp = state[2][1];
    state[2][1] = state[2][3];
    state[2][3] = temp;

    // Row 3: right rotate by 3 (or left rotate by 1)
    temp = state[3][0];
    state[3][0] = state[3][1];
    state[3][1] = state[3][2];
    state[3][2] = state[3][3];
    state[3][3] = temp;
}
```

解释

- **ShiftRows**: 每行按照其行号左移字节。例如，第 1 行左移 1 字节，第 2 行左移 2 字节，以此类推。
- **InvShiftRows**: 反向移位操作，恢复原来的顺序，以便解密时恢复初始状态。例如，第 1 行右移 1 字节，第 2 行右移 2 字节等。

这样，`ShiftRows` 和 `InvShiftRows` 函数可以在 AES 加密和解密中正确地处理状态矩阵的行移位。

填充 MixColumns

`MixColumns` 是 AES 算法中的一个列混淆操作，主要通过对状态矩阵的每一列进行线性变换来增强混淆效果。`MixColumns` 使用固定的 GF(2^8) 有限域上的多项式来对每一列进行矩阵乘法。

具体地，AES 算法中 `MixColumns` 使用了如下矩阵与状态矩阵的每一列相乘：

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

下面是 `MixColumns` 和 `InvMixColumns` 的实现。由于在 AES 中乘法是在 GF(2^8) 有限域中完成的，因此需要用到有限域上的乘法运算。

`MixColumns` 实现

c 复制代码

```c
// Helper function to perform multiplication in GF(2^8)
uint8_t gmul(uint8_t a, uint8_t b) {
    uint8_t p = 0;
    for (int i = 0; i < 8; i++) {
        if (b & 1) p ^= a;
        uint8_t high_bit = a & 0x80;
        a <<= 1;
        if (high_bit) a ^= 0x1b; // x^8 + x^4 + x^3 + x + 1
        b >>= 1;
    }
    return p;
}

// MixColumns operation for encryption
void MixColumns(uint8_t state[4][4]) {
    uint8_t temp[4];
    for (int c = 0; c < 4; c++) {
        temp[0] = gmul(0x02, state[0][c]) ^ gmul(0x03, state[1][c]) ^ state[2][c
        temp[1] = state[0][c] ^ gmul(0x02, state[1][c]) ^ gmul(0x03, state[2][c]
        temp[2] = state[0][c] ^ state[1][c] ^ gmul(0x02, state[2][c]) ^ gmul(0x0
        temp[3] = gmul(0x03, state[0][c]) ^ state[1][c] ^ state[2][c] ^ gmul(0x0

        for (int i = 0; i < 4; i++) {
            state[i][c] = temp[i];
        }
    }
}
```

`InvMixColumns` 实现

解密时使用的 `InvMixColumns` 操作需要使用不同的矩阵：

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

相应的代码实现如下：

```c
// Inverse MixColumns operation for decryption
void InvMixColumns(uint8_t state[4][4]) {
    uint8_t temp[4];
    for (int c = 0; c < 4; c++) {
        temp[0] = gmul(0x0e, state[0][c]) ^ gmul(0x0b, state[1][c]) ^ gmul(0x0d,
        temp[1] = gmul(0x09, state[0][c]) ^ gmul(0x0e, state[1][c]) ^ gmul(0x0b,
        temp[2] = gmul(0x0d, state[0][c]) ^ gmul(0x09, state[1][c]) ^ gmul(0x0e,
        temp[3] = gmul(0x0b, state[0][c]) ^ gmul(0x0d, state[1][c]) ^ gmul(0x09,

        for (int i = 0; i < 4; i++) {
            state[i][c] = temp[i];
        }
    }
}
```

代码解释

- **gfmul 函数**: 在 GF(2^8) 有限域上进行乘法运算。`gmul` 使用了“标准 AES 有限域乘法”，其中乘法结果超过 8 位时进行模多项式的异或运算。
- **MixColumns 和 InvMixColumns**：分别实现加密和解密时的列混淆操作，通过对每一列进行线性变换混淆字节。

这样，`MixColumns` 和 `InvMixColumns` 就可以正确地在 AES 加密和解密流程中进行列混淆操作了。

填充 KeyExpansion

`KeyExpansion` 是 AES 算法中的密钥扩展步骤，它用于生成加密和解密所需的轮密钥（Round Key）。在 AES-128 中，原始密钥长度为 128 位（16 字节），而每轮操作需要一个 128 位的轮密钥。AES-128 一共需要 10 轮密钥加上初始轮密钥，共 11 个轮密钥。

在密钥扩展过程中，会使用一个叫做 Rcon（轮常量）数组，以及 `SubWord` 和 `RotWord` 操作：

- **Rcon**：每一轮的常量，用于混淆。
- **SubWord**：使用 AES S-box 进行字节替换。
- **RotWord**：对 4 字节向量循环左移一个字节。

以下是 AES-128 `KeyExpansion` 的实现代码：

```
// Substitute word using S-box for key expansion
void SubWord(uint8_t *word) {
    for (int i = 0; i < 4; i++) {
        word[i] = sbox[word[i]];
    }
}

// Rotate word by one byte to the left
void RotWord(uint8_t *word) {
    uint8_t temp = word[0];
    word[0] = word[1];
    word[1] = word[2];
    word[2] = word[3];
    word[3] = temp;
```

```
}



// Key expansion function

void KeyExpansion(const uint8_t *key, uint8_t w[4 * Nb * (Nr + 1)]) {

    uint8_t temp[4];

    int i = 0;



    // The first Nk words are the original key

    while (i < Nk) {

        w[4 * i + 0] = key[4 * i + 0];

        w[4 * i + 1] = key[4 * i + 1];

        w[4 * i + 2] = key[4 * i + 2];

        w[4 * i + 3] = key[4 * i + 3];

        i++;

    }



    // Generate the rest of the words

    while (i < Nb * (Nr + 1)) {

        temp[0] = w[4 * (i - 1) + 0];

        temp[1] = w[4 * (i - 1) + 1];

        temp[2] = w[4 * (i - 1) + 2];

        temp[3] = w[4 * (i - 1) + 3];
```

```
            if (i % Nk == 0) {
                RotWord(temp);
                SubWord(temp);
                temp[0] ^= Rcon[i / Nk];
            }

            w[4 * i + 0] = w[4 * (i - Nk) + 0] ^ temp[0];
            w[4 * i + 1] = w[4 * (i - Nk) + 1] ^ temp[1];
            w[4 * i + 2] = w[4 * (i - Nk) + 2] ^ temp[2];
            w[4 * i + 3] = w[4 * (i - Nk) + 3] ^ temp[3];

            i++;
        }
    }
```

解释

- **初始化**：前 `Nk` 个字（在 AES-128 中 `Nk = 4`）直接从密钥中复制。
- **轮密钥生成**：从第 `Nk` 个字开始生成额外的轮密钥，直到生成总共 `Nb * (Nr + 1)` 个字。
  - **RotWord**：每 `Nk` 个字进行一次循环左移一个字节的操作。
  - **SubWord**：使用 S-box 替换字中的每个字节。
  - **Rcon 异或**：第一个字节与轮常量 `Rcon[i / Nk]` 异或，以增加混淆。
- **结果异或**：将 `temp` 与前一个 `Nk` 的字异或生成新的字。

示例调用

在加密函数中，`KeyExpansion` 会生成每一轮所需的轮密钥。例如，在 AES 加密的 `AddRoundKey` 操作中，每一轮的密钥可以直接从 `w` 中读取。

AES-192

区别

AES-192 的实现与 AES-128 非常相似，不同之处在于：

1. **密钥长度**：AES-192 使用 192 位（24 字节）的密钥。
2. **轮数**：AES-192 需要 12 轮加密，每轮使用不同的轮密钥。
3. **Nk 和 Nr**：对于 AES-192，`Nk = 6`（密钥包含 6 个 32 位字），`Nr = 12`（共 12 轮操作）。

修改 KeyExpansion

```
#define Nb 4      // AES 的列数是固定的

#define Nk 6      // AES-192 的密钥由 6 个 32 位字组成

#define Nr 12     // AES-192 有 12 轮



// Rcon 数组用于密钥扩展

static const uint8_t Rcon[13] = {
```

```
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36, 0x6C, 0xD8
};

// Substitute word using S-box for key expansion
void SubWord(uint8_t *word) {
    for (int i = 0; i < 4; i++) {
        word[i] = sbox[word[i]];
    }
}

// Rotate word by one byte to the left
void RotWord(uint8_t *word) {
    uint8_t temp = word[0];
    word[0] = word[1];
    word[1] = word[2];
    word[2] = word[3];
    word[3] = temp;
}

// Key expansion function for AES-192
void KeyExpansion(const uint8_t *key, uint8_t w[4 * Nb * (Nr + 1)]) {
    uint8_t temp[4];
```

```
int i = 0;



// The first Nk words are the original key

while (i < Nk) {

        w[4 * i + 0] = key[4 * i + 0];

        w[4 * i + 1] = key[4 * i + 1];

        w[4 * i + 2] = key[4 * i + 2];

        w[4 * i + 3] = key[4 * i + 3];

        i++;

}



// Generate the rest of the words

while (i < Nb * (Nr + 1)) {

        temp[0] = w[4 * (i - 1) + 0];

        temp[1] = w[4 * (i - 1) + 1];

        temp[2] = w[4 * (i - 1) + 2];

        temp[3] = w[4 * (i - 1) + 3];



        if (i % Nk == 0) {

                RotWord(temp);

                SubWord(temp);

                temp[0] ^= Rcon[i / Nk];
```

```
        } else if (Nk > 6 && i % Nk == 4) {

            SubWord(temp);

        }



        w[4 * i + 0] = w[4 * (i - Nk) + 0] ^ temp[0];

        w[4 * i + 1] = w[4 * (i - Nk) + 1] ^ temp[1];

        w[4 * i + 2] = w[4 * (i - Nk) + 2] ^ temp[2];

        w[4 * i + 3] = w[4 * (i - Nk) + 3] ^ temp[3];



        i++;

    }

}
```

解释

- **Nk = 6**：在 AES-192 中，密钥有 6 个 32 位字。
- **Nr = 12**：AES-192 需要 12 轮。
- **轮密钥生成逻辑**：与 AES-128 类似，但由于 `Nk = 6`，在每隔 `Nk = 6` 个字时会应用 `RotWord` 和 `SubWord`，以及 `Rcon` 的异或。
- **额外的 `SubWord` 操作**：在 `Nk > 6` 的情况下，每 4 个字时再应用一次 `SubWord`，用于增加混淆（此条件在 AES-256 中也适用）。

总结

- AES-192 的 `KeyExpansion` 生成了 12 + 1 = 13 个轮密钥，每个轮密钥占用 4 列（16 字节）。
- 其他加密、解密函数的 `ShiftRows`、`MixColumns` 等操作和 AES-128 一致，唯一的差别在于 AES-192 需要进行 12 轮（Nr = 12）的加密/解密。

AES-256

区别

AES-256 的实现与 AES-128 和 AES-192 类似。不同之处主要是：

1. **密钥长度**：AES-256 使用 256 位（32 字节）的密钥。
2. **轮数**：AES-256 需要 14 轮加密，每轮使用不同的轮密钥。
3. **Nk 和 Nr**：对于 AES-256，`Nk = 8`（密钥包含 8 个 32 位字），`Nr = 14`（共 14 轮操作）。

修改 KeyExpansion

```
#define Nb 4      // AES 的列数是固定的
#define Nk 8      // AES-256 的密钥由 8 个 32 位字组成
#define Nr 14     // AES-256 有 14 轮


// Rcon 数组用于密钥扩展
static const uint8_t Rcon[15] = {
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D
};


// Substitute word using S-box for key expansion
void SubWord(uint8_t *word) {
    for (int i = 0; i < 4; i++) {
        word[i] = sbox[word[i]];
    }
```

```
}



// Rotate word by one byte to the left

void RotWord(uint8_t *word) {

    uint8_t temp = word[0];

    word[0] = word[1];

    word[1] = word[2];

    word[2] = word[3];

    word[3] = temp;

}



// Key expansion function for AES-256

void KeyExpansion(const uint8_t *key, uint8_t w[4 * Nb * (Nr + 1)]) {

    uint8_t temp[4];

    int i = 0;



    // The first Nk words are the original key

    while (i < Nk) {

        w[4 * i + 0] = key[4 * i + 0];

        w[4 * i + 1] = key[4 * i + 1];

        w[4 * i + 2] = key[4 * i + 2];

        w[4 * i + 3] = key[4 * i + 3];
```

```
        i++;
    }

    // Generate the rest of the words
    while (i < Nb * (Nr + 1)) {
        temp[0] = w[4 * (i - 1) + 0];
        temp[1] = w[4 * (i - 1) + 1];
        temp[2] = w[4 * (i - 1) + 2];
        temp[3] = w[4 * (i - 1) + 3];

        if (i % Nk == 0) {
            RotWord(temp);
            SubWord(temp);
            temp[0] ^= Rcon[i / Nk];
        } else if (i % Nk == 4) {
            SubWord(temp);
        }

        w[4 * i + 0] = w[4 * (i - Nk) + 0] ^ temp[0];
        w[4 * i + 1] = w[4 * (i - Nk) + 1] ^ temp[1];
        w[4 * i + 2] = w[4 * (i - Nk) + 2] ^ temp[2];
        w[4 * i + 3] = w[4 * (i - Nk) + 3] ^ temp[3];
```

```
        i++;
    }
}
```

解释

- **Nk = 8**：AES-256 密钥有 8 个 32 位字。
- **Nr = 14**：AES-256 需要 14 轮。
- **轮密钥生成逻辑**：
  - 对于每 `Nk` 个字，进行 `RotWord` 和 `SubWord` 操作，并与轮常量 `Rcon[i / Nk]` 异或。
  - 当 `i % Nk == 4` 时，仅应用 `SubWord`，没有 `RotWord` 和 `Rcon` 异或。这用于在 AES-256 的密钥扩展中增加混淆。

说明

- AES-256 `KeyExpansion` 生成 15 个 128 位的轮密钥，每个轮密钥用于 AES-256 加密和解密的每一轮。
- 其他加密、解密函数的 `ShiftRows`、`MixColumns` 等操作与 AES-128 和 AES-192 相同，唯一的差别是 AES-256 进行 14 轮（Nr = 14）的加密/解密操作。