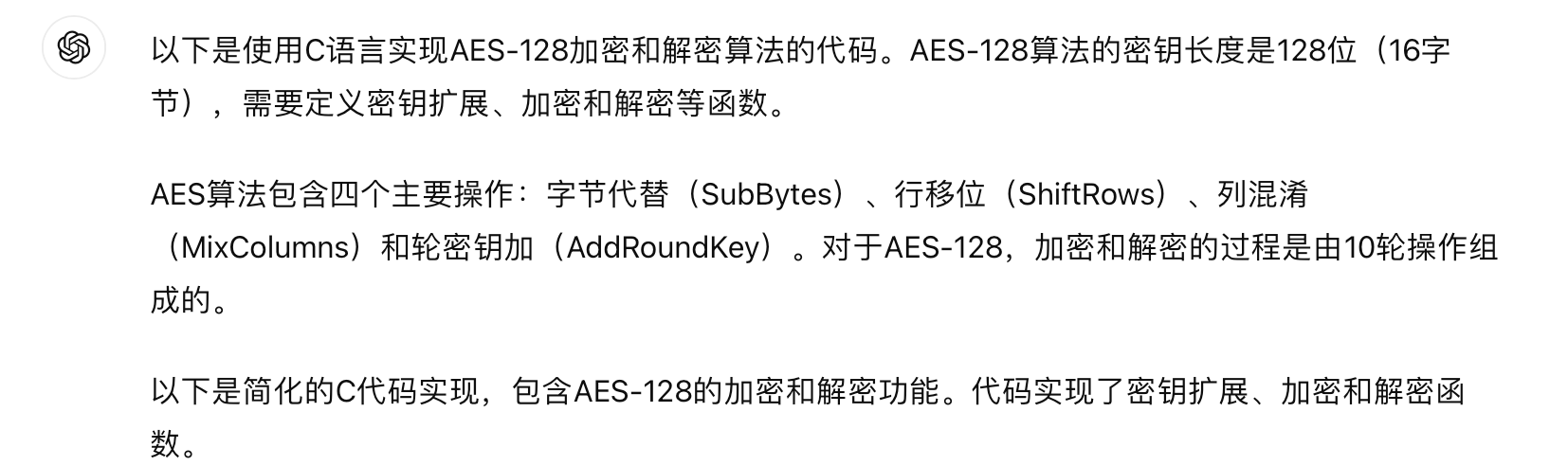
# AES-128

## 代码框架



#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]);

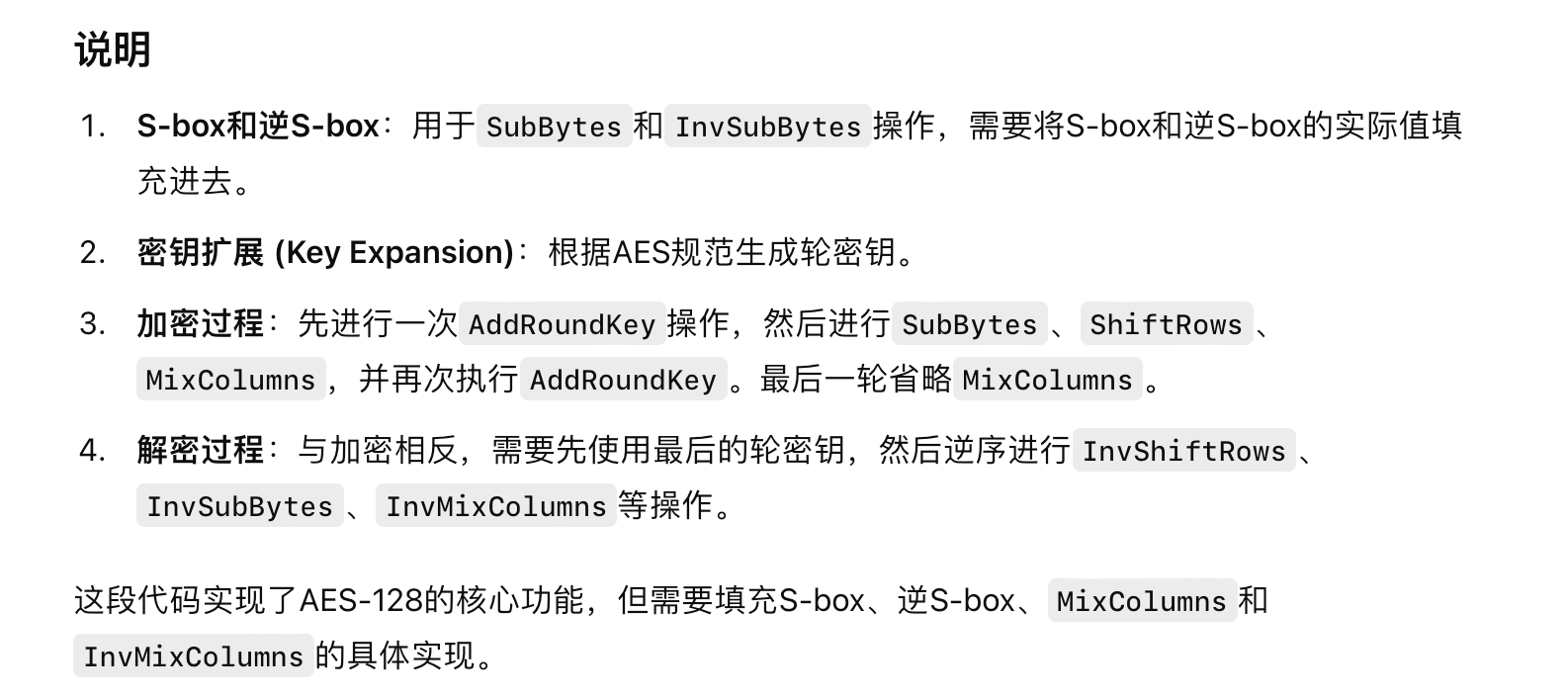
AES\_Decrypt(data, key);

printf("\nDecrypted data:\n");

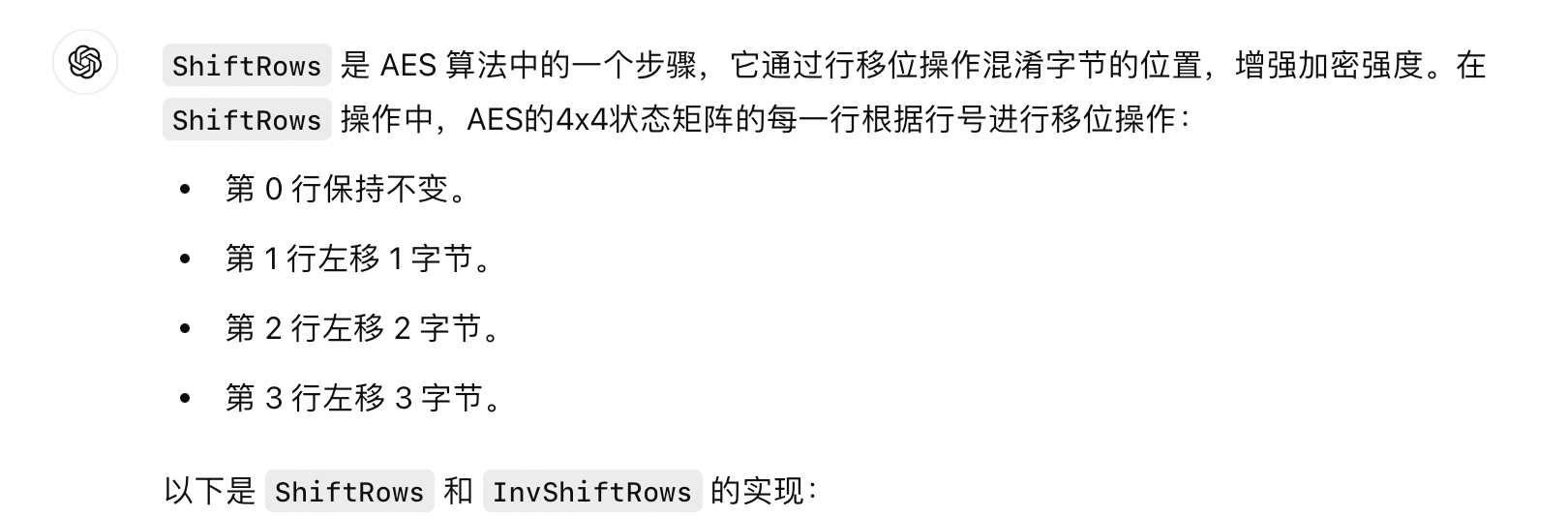
for (int i = 0; i < 16; i++) printf("%02x ", data[i]);

return 0;

}



## 填充ShiftRows



// 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];

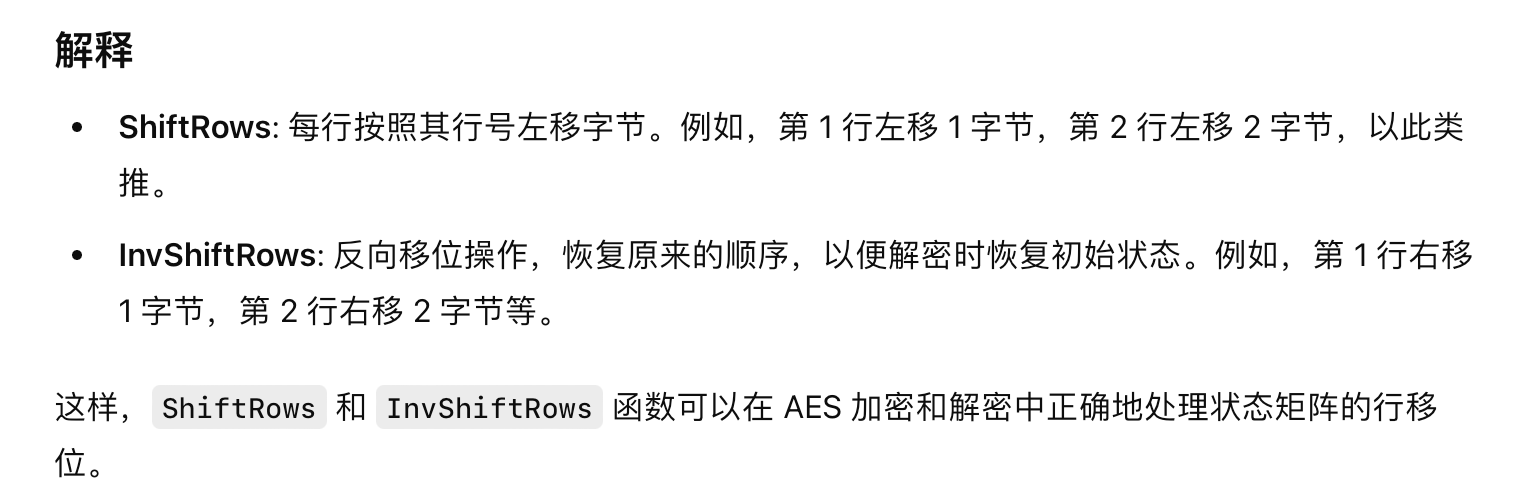
state[3][0] = state[3][1];

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

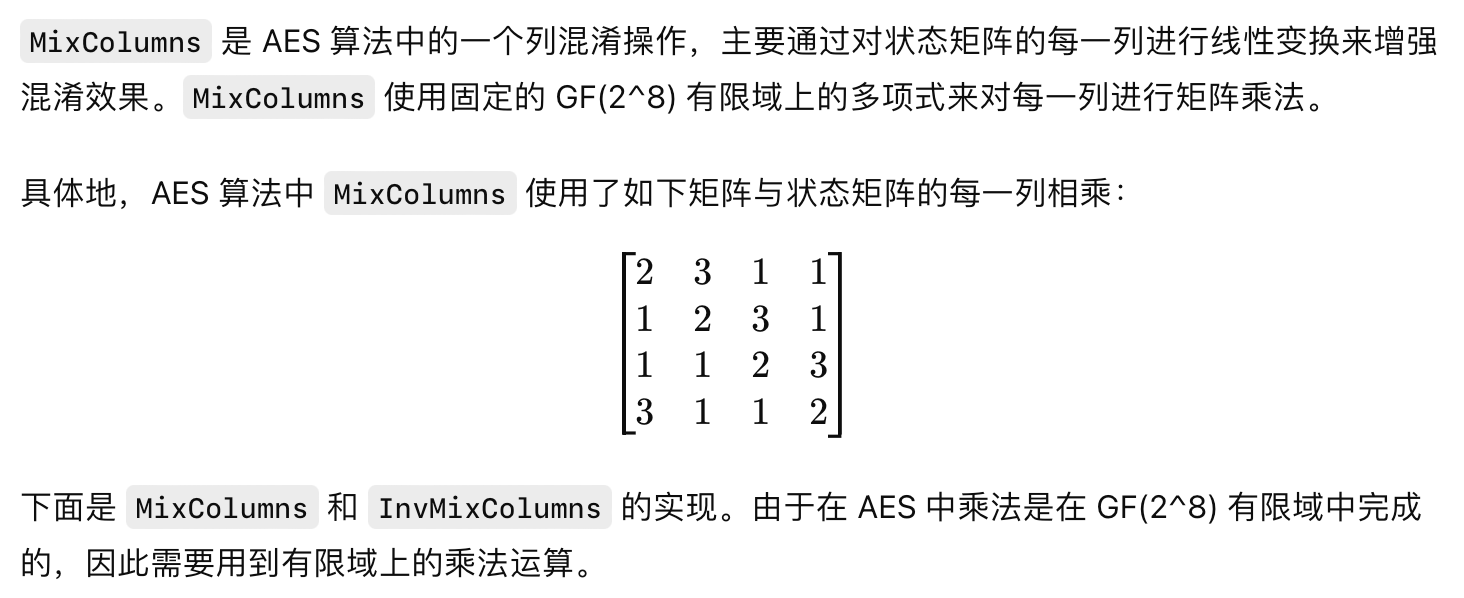
state[3][2] = state[3][3];

state[3][3] = temp;

}



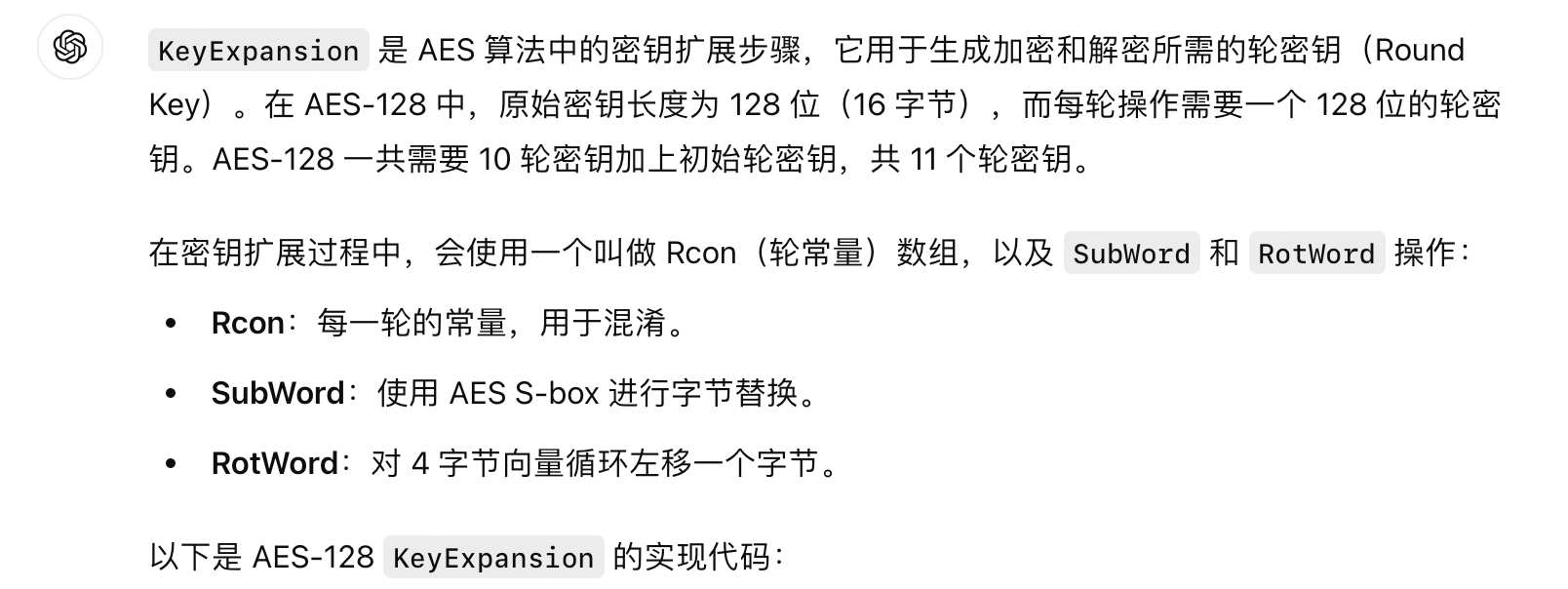
## 填充MixColumns







## 填充KeyExpansion



// 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];

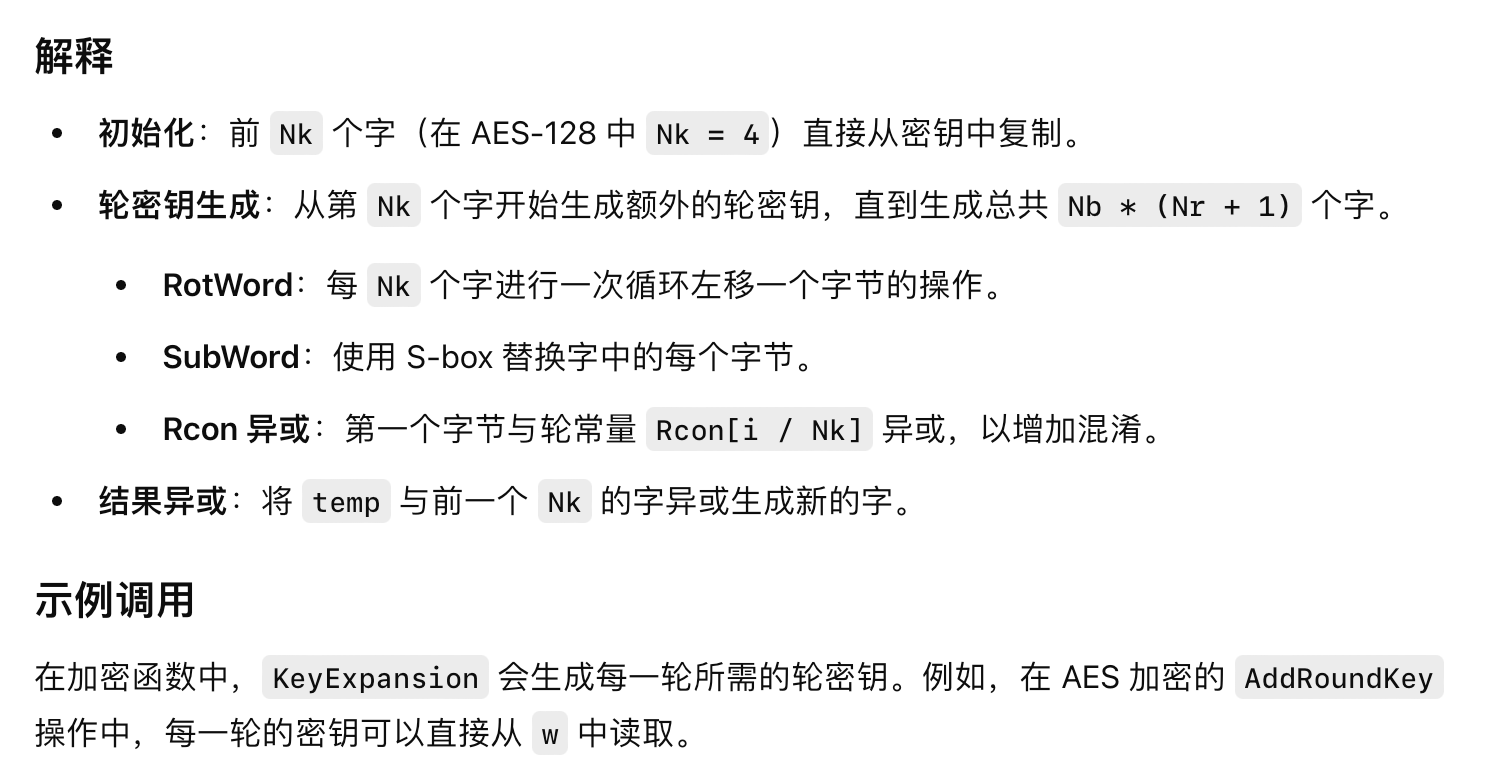
w[4 \* i + 2] = w[4 \* (i - Nk) + 2] ^ temp[2];

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

i++;

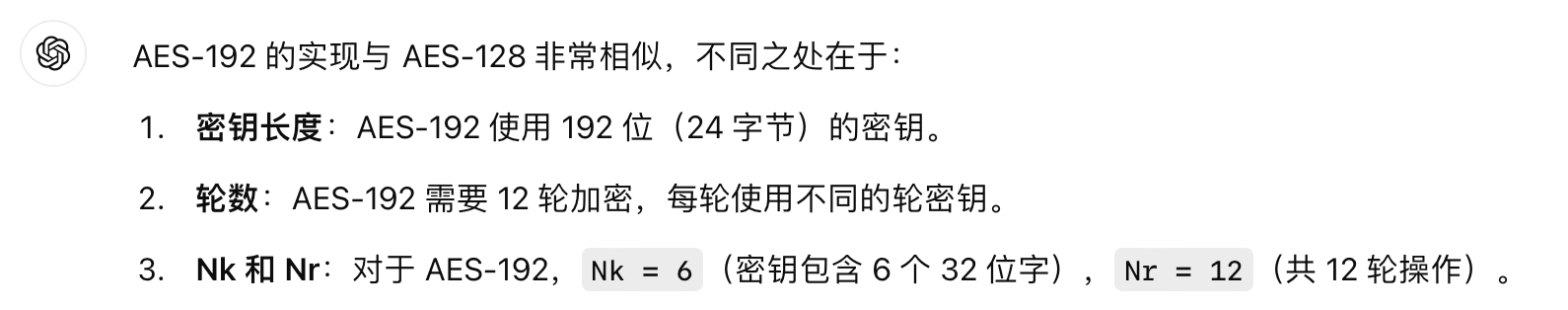
}

}



# AES-192

## 区别



## 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];

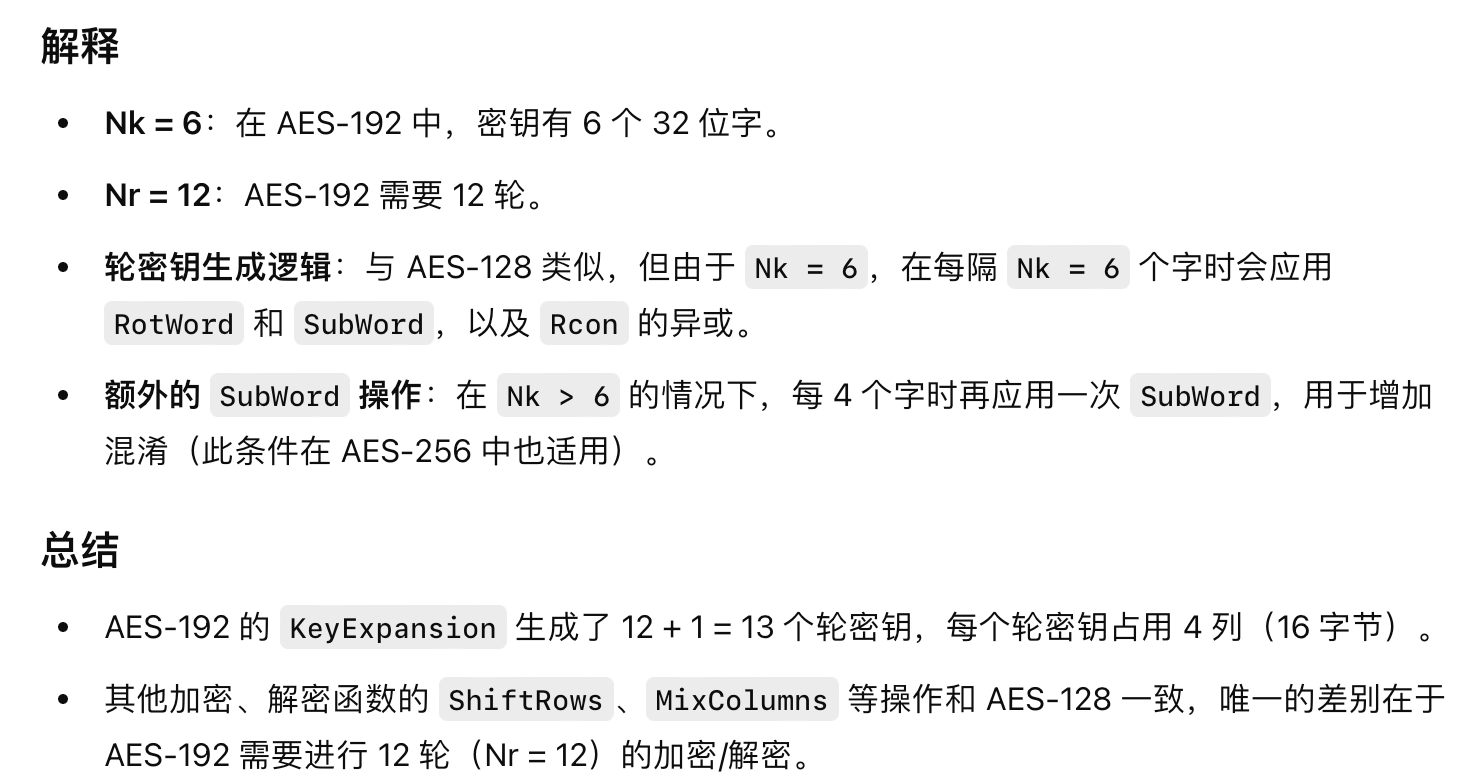
w[4 \* i + 2] = w[4 \* (i - Nk) + 2] ^ temp[2];

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

i++;

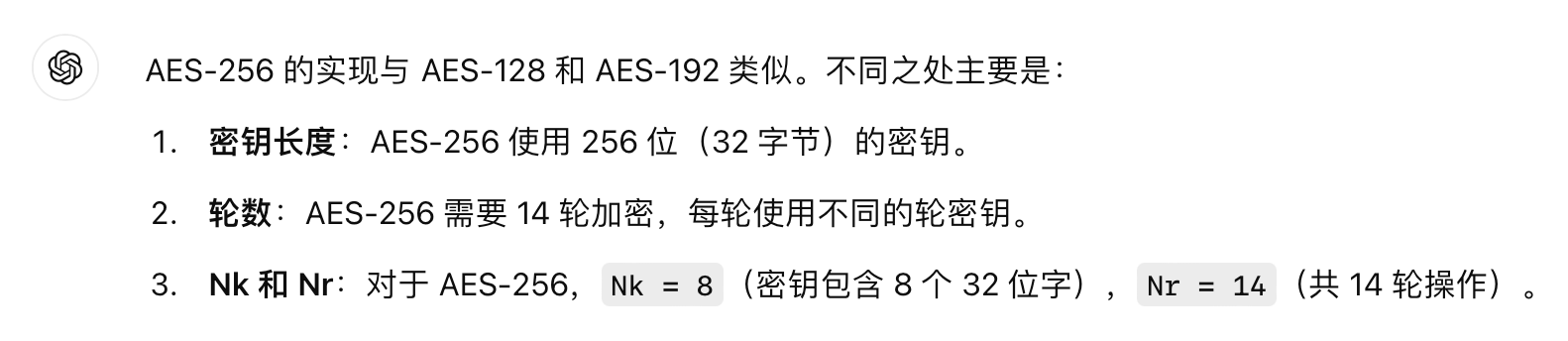
}

}



# AES-256

## 区别



## 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++;

}

}

