实验二 -- SM4分组密码的设计及实现

一、实验目的

本实验的目的是实现一个基于S-Box的加密算法-SM4。主要目标是：

1. 理解S-Box在加密算法中的作用。
2. 实现字节和32位字之间的转换。
3. 实现S-Box和线性变换。
4. 完成密钥扩展。
5. 实现加密和解密过程。

二、实验环境

- 编程语言：Python
- 运行平台：任意支持Python的环境

三、实验原理

1. 概述

SM4算法是中国国家密码管理局于2012年发布的一种分组密码算法，类似于DES和AES，SM4的分组长度和密钥长度均为128位。它采用32轮非线性迭代结构进行加密，每轮使用一个变换函数F。加密和解密过程结构相同，只是轮密钥使用顺序相反。

2. 参数产生

- **字节和字**：字节为8位二进制数，字为32位二进制数。
- **S盒**：固定的8位输入和输出置换表。
- **密钥**：128位加密密钥，表示为4个32位字（MKi, i=0,1,2,3）。
- **系统参数（FK）**：4个固定字（FK0, FK1, FK2, FK3）。
- **固定参数（CK）**：32个固定字（CK0, CK1, ..., CK31）。
- **轮密钥（rki）**：由密钥扩展算法生成的32个字。

3. 轮函数

加密函数采用合成置换T，由非线性变换和线性变换构成。

- **非线性变换**：由4个并行的S盒组成，输入和输出都是8位数据，S盒数据采用16进制表示。
- **线性变换**：应用于非线性变换后的32位字B。

4. 密钥扩展

密钥扩展由128位的加密密钥、系统参数和固定参数生成32个轮密钥。

- 初始变换：用加密密钥和系统参数进行初始计算。
- 轮密钥生成：通过迭代运算，将每一轮的结果作为下一轮的输入，生成32个轮密钥。

5. 加密/解密过程

加密过程通过32轮迭代，每轮使用一个变换函数F，并输出中间结果。最终的加密输出为中间结果的反序排列。解密过程通过逆向使用轮密钥进行32轮迭代，结构与加密过程相同但顺序相反。

SM4算法的整体流程

1. **密钥装入**：初始化系统参数和固定参数，将加密密钥扩展生成32个轮密钥。
2. **加密/解密**：
    - **加密**：对128位明文分组进行32轮迭代运算，最终输出加密后的密文。
    - **解密**：使用反向的轮密钥顺序，对128位密文分组进行32轮迭代运算，输出解密后的明文。

SM4算法安全性

- **强非线性**：通过S盒提供高非线性变换，增加加密强度。
- **复杂的密钥扩展**：轮密钥的生成过程复杂，增加了攻击难度。
- **高效性**：适合硬件和软件实现，保证了较高的加密和解密效率。

四、代码实现概述

代码包含多个功能模块，包括字节和32位字的转换、S-Box变换、线性变换、密钥扩展、轮函数、加密和解密过程。下面是对这些模块的详细解释：

字节和32位字之间的转换

```
def word_to_bytes(word, byte_list):
    byte_list.extend([(word >> i) & 0xff for i in range(24, -1, -8)])

def bytes_to_word(byte_list):
    result = 0
    for i in range(4):
        bits = 24 - i * 8
        result |= (byte_list[i] << bits)
    return result
```

- `word_to_bytes` 将32位的字转换为字节，并扩展到字节列表中。
- `bytes_to_word` 将字节列表转换为32位字。

S-Box变换

```
def s_box_transform(word):
    result = []
    for i in range(4):
        byte = (word >> (24 - i * 8)) & 0xff
        row = byte >> 4
        col = byte & 0x0f
        index = (row * 16 + col)
        result.append(S_BOX[index])
    return bytes_to_word(result)
```

- `s_box_transform` 对32位输入字进行S-Box变换，返回变换后的32位字。

线性变换

```
def rotate_left(word, bits):
    return (word << bits & 0xffffffff) | (word >> (32 - bits))

def linear_transform(word):
    return word ^ rotate_left(word, 2) ^ rotate_left(word, 10) ^
rotate_left(word, 18) ^ rotate_left(word, 24)
```

- `rotate_left` 对32位字进行循环左移。
- `linear_transform` 进行线性变换，结合了多个循环左移操作。

密钥扩展

```
def key_expansion_transform(k1, k2, k3, ck):
    xor_result = k1 ^ k2 ^ k3 ^ ck
    t = s_box_transform(xor_result)
    return t ^ rotate_left(t, 13) ^ rotate_left(t, 23)

def key_expansion(main_key):
    MK = [(main_key >> (128 - (i + 1) * 32)) & 0xffffffff for i in range(4)]
    keys = [FK[i] ^ MK[i] for i in range(4)]
    round_keys = []
    for i in range(32):
        t = key_expansion_transform(keys[i + 1], keys[i + 2], keys[i + 3],
CK[i])
        k = keys[i] ^ t
        keys.append(k)
        round_keys.append(k)
    return round_keys
```

- `key_expansion_transform` 对密钥进行扩展变换，生成轮密钥。
- `key_expansion` 执行密钥扩展，生成32个轮密钥。

加密和解密

```
def round_function(x1, x2, x3, rk):
    t = x1 ^ x2 ^ x3 ^ rk
    t = s_box_transform(t)
    return t ^ rotate_left(t, 2) ^ rotate_left(t, 10) ^ rotate_left(t, 18) ^
rotate_left(t, 24)

def format_output(x0, x1, x2, x3):
    return f"{x3:08x}{x2:08x}{x1:08x}{x0:08x}"

def encrypt(plaintext, round_keys):
    X = [plaintext >> (128 - (i + 1) * 32) & 0xffffffff for i in range(4)]
    for i in range(32):
        t = round_function(X[1], X[2], X[3], round_keys[i])
        c = t ^ X[0]
        X = X[1:] + [c]
    return format_output(X[0], X[1], X[2], X[3])

def decrypt(ciphertext, round_keys):
    ciphertext = int(ciphertext, 16)
```

```python
    X = [ciphertext >> (128 - (i + 1) * 32) & 0xffffffff for i in range(4)]
    for i in range(32):
        t = round_function(X[1], X[2], X[3], round_keys[31 - i])
        c = t ^ X[0]
        X = X[1:] + [c]
    return format_output(X[0], X[1], X[2], X[3])
```

- `round_function` 执行每一轮的加密变换。
- `format_output` 格式化输出，加密/解密后的结果。
- `encrypt` 使用轮密钥对明文进行加密。
- `decrypt` 使用轮密钥对密文进行解密。

实验结果

加密过程

```python
plaintext = "202420213004426c69686f72616e"  # 加密的明文
main_key = "0123456789abcdef0123456789abcdef"  # 主密钥
round_keys = key_expansion(int(main_key, 16))
ciphertext = encrypt(int(plaintext, 16), round_keys)
print("加密结果:")
display_output(ciphertext, "Ciphertext")
```

解密过程

```python
decrypted_text = decrypt(ciphertext, round_keys)
decrypted_text = decrypted_text.lstrip('0')  # 去除前导的0
print("\n解密结果:")
print("Plaintext:", decrypted_text)
```

输出：

```
加密结果:
Ciphertext: 8d 0c 96 99 62 f8 7c dd e6 e8 2b 16 62 b8 ef d6

解密结果:
Plaintext: 202420213004426c69686f72616e
```

五、总结

本实验通过实现一个加密算法，深入理解了S-Box、线性变换、密钥扩展以及加密解密过程。通过实验结果验证了加密和解密的正确性，实现了从明文到密文再到明文的转化过程。

附录:代码

```
S_BOX = [0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2,
0x28, 0xFB, 0x2C, 0x05,
         0x2B, 0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26,
0x49, 0x86, 0x06, 0x99,
         0x9C, 0x42, 0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43,
0xED, 0xCF, 0xAC, 0x62,
         0xE4, 0xB3, 0x1C, 0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA,
0x75, 0x8F, 0x3F, 0xA6,
         0x47, 0x07, 0xA7, 0xFC, 0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19,
0xE6, 0x85, 0x4F, 0xA8,
         0x68, 0x6B, 0x81, 0xB2, 0x71, 0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B,
0x70, 0x56, 0x9D, 0x35,
         0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58, 0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B,
0x01, 0x21, 0x78, 0x87,
         0xD4, 0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27, 0x52, 0x4C, 0x36, 0x02, 0xE7,
0xA0, 0xC4, 0xC8, 0x9E,
         0xEA, 0xBF, 0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5, 0xA3, 0xF7, 0xF2, 0xCE,
0xF9, 0x61, 0x15, 0xA1,
         0xE0, 0xAE, 0x5D, 0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD, 0x93, 0x32, 0x30,
0xF5, 0x8C, 0xB1, 0xE3,
         0x1D, 0xF6, 0xE2, 0x2E, 0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29, 0x23, 0xAB,
0x0D, 0x53, 0x4E, 0x6F,
         0xD5, 0xDB, 0x37, 0x45, 0xDE, 0xFD, 0x8E, 0x2F, 0x03, 0xFF, 0x6A, 0x72,
0x6D, 0x6C, 0x5B, 0x51,
         0x8D, 0x1B, 0xAF, 0x92, 0xBB, 0xDD, 0xBC, 0x7F, 0x11, 0xD9, 0x5C, 0x41,
0x1F, 0x10, 0x5A, 0xD8,
         0x0A, 0xC1, 0x31, 0x88, 0xA5, 0xCD, 0x7B, 0xBD, 0x2D, 0x74, 0xD0, 0x12,
0xB8, 0xE5, 0xB4, 0xB0,
         0x89, 0x69, 0x97, 0x4A, 0x0C, 0x96, 0x77, 0x7E, 0x65, 0xB9, 0xF1, 0x09,
0xC5, 0x6E, 0xC6, 0x84,
         0x18, 0xF0, 0x7D, 0xEC, 0x3A, 0xDC, 0x4D, 0x20, 0x79, 0xEE, 0x5F, 0x3E,
0xD7, 0xCB, 0x39, 0x48
         ]

FK = [0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc]
CK = [
    0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,
    0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
    0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,
    0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,
    0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,
    0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279
]


def word_to_bytes(word, byte_list):
    """
    Convert a 32-bit word to bytes and extend the byte_list.
    :param word: 32-bit word
    :param byte_list: List to extend with bytes
    """
    byte_list.extend([(word >> i) & 0xff for i in range(24, -1, -8)])
```

```python
def bytes_to_word(byte_list):
    """
    Convert a list of bytes to a 32-bit word.
    :param byte_list: List of bytes
    :return: 32-bit word
    """
    result = 0
    for i in range(4):
        bits = 24 - i * 8
        result |= (byte_list[i] << bits)
    return result


def s_box_transform(word):
    """
    Perform a non-linear transformation using the S-Box.
    :param word: 32-bit input word
    :return: Transformed 32-bit word
    """
    result = []
    for i in range(4):
        byte = (word >> (24 - i * 8)) & 0xff
        row = byte >> 4
        col = byte & 0x0f
        index = (row * 16 + col)
        result.append(S_BOX[index])
    return bytes_to_word(result)


def rotate_left(word, bits):
    """
    Perform a left circular rotation on a word.
    :param word: 32-bit word to rotate
    :param bits: Number of bits to rotate
    :return: Rotated 32-bit word
    """
    return (word << bits & 0xffffffff) | (word >> (32 - bits))


def linear_transform(word):
    """
    Perform a linear transformation (L transformation).
    :param word: 32-bit input word
    :return: Transformed 32-bit word
    """
    return word ^ rotate_left(word, 2) ^ rotate_left(word, 10) ^
rotate_left(word, 18) ^ rotate_left(word, 24)


def key_expansion_transform(k1, k2, k3, ck):
    """
    Perform the key expansion transformation.
    :param k1, k2, k3: 32-bit words
    :param ck: Constant word
    :return: Transformed 32-bit word
    """
    xor_result = k1 ^ k2 ^ k3 ^ ck
```

```
    t = s_box_transform(xor_result)
    return t ^ rotate_left(t, 13) ^ rotate_left(t, 23)


def round_function(x1, x2, x3, rk):
    """
    Perform the round function transformation for encryption.
    :param x1, x2, x3: 32-bit words
    :param rk: Round key
    :return: Transformed 32-bit word
    """
    t = x1 ^ x2 ^ x3 ^ rk
    t = s_box_transform(t)
    return t ^ rotate_left(t, 2) ^ rotate_left(t, 10) ^ rotate_left(t, 18) ^
rotate_left(t, 24)


def key_expansion(main_key):
    """
    Perform key expansion to generate round keys.
    :param main_key: 128-bit main key
    :return: List of 32 round keys
    """
    MK = [(main_key >> (128 - (i + 1) * 32)) & 0xffffffff for i in range(4)]
    keys = [FK[i] ^ MK[i] for i in range(4)]
    round_keys = []
    for i in range(32):
        t = key_expansion_transform(keys[i + 1], keys[i + 2], keys[i + 3],
CK[i])
        k = keys[i] ^ t
        keys.append(k)
        round_keys.append(k)
    return round_keys


def format_output(x0, x1, x2, x3):
    """
    Format the output of encryption/decryption.
    :param x0, x1, x2, x3: 32-bit words
    :return: Formatted 128-bit string
    """
    x0 &= 0xffffffff
    x1 &= 0xffffffff
    x2 &= 0xffffffff
    x3 &= 0xffffffff
    return f"{x3:08x}{x2:08x}{x1:08x}{x0:08x}"


def encrypt(plaintext, round_keys):
    """
    Encrypt the plaintext using the round keys.
    :param plaintext: 128-bit plaintext
    :param round_keys: List of 32 round keys
    :return: 128-bit ciphertext
    """
    X = [plaintext >> (128 - (i + 1) * 32) & 0xffffffff for i in range(4)]
    for i in range(32):
        t = round_function(X[1], X[2], X[3], round_keys[i])
```

```python
        c = t ^ X[0]
        X = X[1:] + [c]
    return format_output(X[0], X[1], X[2], X[3])


def decrypt(ciphertext, round_keys):
    """
    Decrypt the ciphertext using the round keys.
    :param ciphertext: 128-bit ciphertext
    :param round_keys: List of 32 round keys
    :return: 128-bit plaintext
    """
    ciphertext = int(ciphertext, 16)
    X = [ciphertext >> (128 - (i + 1) * 32) & 0xffffffff for i in range(4)]
    for i in range(32):
        t = round_function(X[1], X[2], X[3], round_keys[31 - i])
        c = t ^ X[0]
        X = X[1:] + [c]
    return format_output(X[0], X[1], X[2], X[3])


def display_output(hex_string, label):
    """
    Display the formatted output with a label.
    :param hex_string: Hexadecimal string to display
    :param label: Label for the output
    """
    formatted = " ".join([hex_string[i:i + 2] for i in range(0, len(hex_string),
2)])
    print(f"{label}: {formatted}")


if __name__ == '__main__':
    # 加密
    plaintext = "202420213004426c69686f72616e"  # 加密的明文
    main_key = "0123456789abcdef0123456789abcdef"  # 主密钥
    round_keys = key_expansion(int(main_key, 16))
    ciphertext = encrypt(int(plaintext, 16), round_keys)
    print("加密结果:")
    display_output(ciphertext, "Ciphertext")

    # 解密
    decrypted_text = decrypt(ciphertext, round_keys)
    decrypted_text = decrypted_text.lstrip('0')  # 去除前导的0
    print("\n解密结果:")
    print("Plaintext:", decrypted_text)
```