实验一

一、实验目的

1. 实现并理解密钥流生成算法的工作原理。
2. 验证生成的密钥流在加密和解密过程中的正确性。

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. ZUC算法简介

ZUC算法是一种同步序列密码，是我国自主设计并成为国际标准的加密算法之一。它包括ZUC算法本身、加密算法128-EEA3和完整性算法128-EIA3，广泛应用于4G LTE通信标准中。

2. ZUC算法结构

ZUC算法结构分为上层的线性反馈移位寄存器（LFSR）、中层的比特重组（BR）和下层的非线性函数F三部分。

2.1 线性反馈移位寄存器（LFSR）

LFSR由16个31比特的寄存器单元组成，采用有限域GF（231-1）上的16次本原多项式作为连接多项式，输出具有良好随机性的m序列。

- **初始化模式**：LFSR接收一个31比特字u，该字由非线性函数F的32比特输出W通过舍弃最低位得到。
- **工作模式**：LFSR不接受任何输入，持续生成序列。

2.2 比特重组（BR）

比特重组从LFSR的寄存器单元中抽取128比特，组成四个32比特字X0、X1、X2、X3，实现数据转换，破坏LFSR的线性结构。

2.3 非线性函数F

非线性函数F包含两个32比特存储单元R0和R1，输入为三个32比特字X0、X1、X2，输出为一个32比特字W。F函数通过模232加法、异或操作、S盒和L函数变换实现非线性压缩。

3. ZUC算法流程

3.1 密钥装入

将128比特的初始密钥KEY和初始向量IV扩展为16个31比特字，初始化LFSR的寄存器单元。

3.2 算法运行

- **初始化阶段**：将KEY和IV装入LFSR，非线性函数F的R1和R2置为0，重复比特运算和F运算32次，进行LFSR初始化。
- **工作模式**：执行一次F函数并舍弃输出，然后每个节拍输出一个32比特的密钥字Z。

4. ZUC算法安全性

- **LFSR设计**：采用素域GF（231-1）的m序列，周期长，统计特性好，安全且高效。
- **BR设计**：重组数据随机性好，重复概率低。
- **F函数设计**：F函数采用两个非线性变换S盒，提供混淆和扩散作用，确保算法安全。

总结

ZUC算法通过LFSR提供基础的线性序列，再通过BR和非线性函数F引入复杂性和随机性，确保了高安全性和高效性，是一个设计精良的同步序列密码。

四、常量和初始值

- S0, S1: 非线性变换中的S盒
- D: LFSR（线性反馈移位寄存器）的初始多项式系数
- lfsr_state: LFSR状态寄存器，初始化为全0
- keystream: 存储生成的密钥流
- recon_bits: 存储比特重构结果
- reg1, reg2, out_word: 非线性函数中的寄存器和输出字
- mod32: 32位模数，用于模加运算

五、算法步骤

1. 初始化LFSR

使用密钥和IV（初始化向量）对LFSR进行初始化：

```
def initialize_lfsr(key, iv):
    global reg1, reg2, out_word, lfsr_state
    for i in range(16):
        lfsr_state[i] = (key[i] << 23) | (D[i] << 8) | iv[i]
    for i in range(32):
        bit_reconstruct()
        nonlinear_function(recon_bits[0], recon_bits[1], recon_bits[2])
        lfsr_initialize_mode(out_word >> 1)
```

- 将密钥和IV与常数D进行结合，初始化LFSR状态。
- 进行32次LFSR初始化模式的迭代。

2. 比特重构

将LFSR状态寄存器中的部分比特位组合形成重构的比特位：

```python
def bit_reconstruct():
    recon_bits[0] = combine_high_low_bits(lfsr_state[15], lfsr_state[14])
    recon_bits[1] = combine_low_high_bits(lfsr_state[11], lfsr_state[9])
    recon_bits[2] = combine_low_high_bits(lfsr_state[7], lfsr_state[5])
    recon_bits[3] = combine_low_high_bits(lfsr_state[2], lfsr_state[0])
```

3. 非线性函数

执行非线性变换，生成输出字和更新寄存器状态：

```python
def nonlinear_function(x0, x1, x2):
    global reg1, reg2, out_word
    out_word = mod32_add(x0 ^ reg1, reg2)
    temp1 = mod32_add(reg1, x1)
    temp2 = reg2 ^ x2
    reg1 = sbox_transform(l1_transform(((temp1 << 16) | (temp2 >> 16)) &
0xffffffff))
    reg2 = sbox_transform(l2_transform(((temp2 << 16) | (temp1 >> 16)) &
0xffffffff))
```

4. LFSR工作模式

更新LFSR状态，确保其不为0：

```python
def lfsr_operate_mode():
    lfsr_state.append((2 ** 15 * lfsr_state[15] + 2 ** 17 * lfsr_state[13] + 2
** 21 * lfsr_state[10] +
                       2 ** 20 * lfsr_state[4] + (1 + 2 ** 8) * lfsr_state[0]) %
(2 ** 31 - 1))
    if lfsr_state[16] == 0:
        lfsr_state[16] = 2 ** 31 - 1
    lfsr_state.pop(0)
```

5. 密钥流生成

根据LFSR状态和非线性函数，生成指定长度的密钥流：

```python
def generate_keystream():
    bit_reconstruct()
    nonlinear_function(recon_bits[0], recon_bits[1], recon_bits[2])
    lfsr_operate_mode()
    for _ in range(keystream_length):
        bit_reconstruct()
        nonlinear_function(recon_bits[0], recon_bits[1], recon_bits[2])
        keystream.append(out_word ^ recon_bits[3])
        lfsr_operate_mode()
```

六、加密和解密

使用生成的密钥流，对给定的文本进行异或操作，实现加密和解密：

```
def encrypt_decrypt_text(text):
    text_bytes = text.encode('utf-8')
    encrypted_decrypted_bytes = bytes(
        [b ^ (keystream[i % keystream_length] & 0xff) for i, b in
enumerate(text_bytes)])
    return encrypted_decrypted_bytes.decode('utf-8', errors='ignore')
```

七、实验结果

```
# 示例密钥和IV
key = [0x3d, 0x4c, 0x4b, 0xe9, 0x6a, 0x82, 0xfd, 0xae, 0xb5, 0x8f, 0x64, 0x1d,
0xb1, 0x7b, 0x45, 0x5b]
iv = [0x84, 0x31, 0x9a, 0xa8, 0xde, 0x69, 0x15, 0xca, 0x1f, 0x6b, 0xda, 0x6b,
0xfb, 0xd8, 0xc7, 0x66]

# 初始化LFSR
initialize_lfsr(key, iv)
# 生成密钥流
generate_keystream()
# 打印密钥流
display_keystream()

# 要加密的消息
message = "2024shangyingmima"
print(f'\nmessage: {message}')

print('----------------------------------------------')
# 加密消息
encrypted_message = encrypt_decrypt_text(message)
print(f'Encrypted Message: {encrypted_message}')

# 解密消息
decrypted_message = encrypt_decrypt_text(encrypted_message)
print(f'Decrypted Message: {decrypted_message}')
```

输出：

```
KEY0: 14f1c272 KEY1: 3279c419
message: 2024shangyingmima
----------------------------------------------
Encrypted Message: @)@-□q□w□` □w□†□†□
Decrypted Message: 2024shangyingmima
```

八、结论

实验成功实现了密钥流生成算法，生成的密钥流在加密和解密过程中表现出良好的随机性和一致性。通过密钥流的异或操作，实现了对给定文本的加密和解密，验证了算法的正确性。

附录:代码

```
S0 = [[0x3e, 0x72, 0x5b, 0x47, 0xca, 0xe0, 0x00, 0x33, 0x04, 0xd1, 0x54, 0x98,
0x09, 0xb9, 0x6d, 0xcb],
      [0x7b, 0x1b, 0xf9, 0x32, 0xaf, 0x9d, 0x6a, 0xa5, 0xb8, 0x2d, 0xfc, 0x1d,
0x08, 0x53, 0x03, 0x90],
      [0x4d, 0x4e, 0x84, 0x99, 0xe4, 0xce, 0xd9, 0x91, 0xdd, 0xb6, 0x85, 0x48,
0x8b, 0x29, 0x6e, 0xac],
      [0xcd, 0xc1, 0xf8, 0x1e, 0x73, 0x43, 0x69, 0xc6, 0xb5, 0xbd, 0xfd, 0x39,
0x63, 0x20, 0xd4, 0x38],
      [0x76, 0x7d, 0xb2, 0xa7, 0xcf, 0xed, 0x57, 0xc5, 0xf3, 0x2c, 0xbb, 0x14,
0x21, 0x06, 0x55, 0x9b],
      [0xe3, 0xef, 0x5e, 0x31, 0x4f, 0x7f, 0x5a, 0xa4, 0x0d, 0x82, 0x51, 0x49,
0x5f, 0xba, 0x58, 0x1c],
      [0x4a, 0x16, 0xd5, 0x17, 0xa8, 0x92, 0x24, 0x1f, 0x8c, 0xff, 0xd8, 0xae,
0x2e, 0x01, 0xd3, 0xad],
      [0x3b, 0x4b, 0xda, 0x46, 0xeb, 0xc9, 0xde, 0x9a, 0x8f, 0x87, 0xd7, 0x3a,
0x80, 0x6f, 0x2f, 0xc8],
      [0xb1, 0xb4, 0x37, 0xf7, 0x0a, 0x22, 0x13, 0x28, 0x7c, 0xcc, 0x3c, 0x89,
0xc7, 0xc3, 0x96, 0x56],
      [0x07, 0xbf, 0x7e, 0xf0, 0x0b, 0x2b, 0x97, 0x52, 0x35, 0x41, 0x79, 0x61,
0xa6, 0x4c, 0x10, 0xfe],
      [0xbc, 0x26, 0x95, 0x88, 0x8a, 0xb0, 0xa3, 0xfb, 0xc0, 0x18, 0x94, 0xf2,
0xe1, 0xe5, 0xe9, 0x5d],
      [0xd0, 0xdc, 0x11, 0x66, 0x64, 0x5c, 0xec, 0x59, 0x42, 0x75, 0x12, 0xf5,
0x74, 0x9c, 0xaa, 0x23],
      [0x0e, 0x86, 0xab, 0xbe, 0x2a, 0x02, 0xe7, 0x67, 0xe6, 0x44, 0xa2, 0x6c,
0xc2, 0x93, 0x9f, 0xf1],
      [0xf6, 0xfa, 0x36, 0xd2, 0x50, 0x68, 0x9e, 0x62, 0x71, 0x15, 0x3d, 0xd6,
0x40, 0xc4, 0xe2, 0x0f],
      [0x8e, 0x83, 0x77, 0x6b, 0x25, 0x05, 0x3f, 0x0c, 0x30, 0xea, 0x70, 0xb7,
0xa1, 0xe8, 0xa9, 0x65],
      [0x8d, 0x27, 0x1a, 0xdb, 0x81, 0xb3, 0xa0, 0xf4, 0x45, 0x7a, 0x19, 0xdf,
0xee, 0x78, 0x34, 0x60]]

S1 = [[0x55, 0xc2, 0x63, 0x71, 0x3b, 0xc8, 0x47, 0x86, 0x9f, 0x3c, 0xda, 0x5b,
0x29, 0xaa, 0xfd, 0x77],
      [0x8c, 0xc5, 0x94, 0x0c, 0xa6, 0x1a, 0x13, 0x00, 0xe3, 0xa8, 0x16, 0x72,
0x40, 0xf9, 0xf8, 0x42],
      [0x44, 0x26, 0x68, 0x96, 0x81, 0xd9, 0x45, 0x3e, 0x10, 0x76, 0xc6, 0xa7,
0x8b, 0x39, 0x43, 0xe1],
      [0x3a, 0xb5, 0x56, 0x2a, 0xc0, 0x6d, 0xb3, 0x05, 0x22, 0x66, 0xbf, 0xdc,
0x0b, 0xfa, 0x62, 0x48],
      [0xdd, 0x20, 0x11, 0x06, 0x36, 0xc9, 0xc1, 0xcf, 0xf6, 0x27, 0x52, 0xbb,
0x69, 0xf5, 0xd4, 0x87],
      [0x7f, 0x84, 0x4c, 0xd2, 0x9c, 0x57, 0xa4, 0xbc, 0x4f, 0x9a, 0xdf, 0xfe,
0xd6, 0x8d, 0x7a, 0xeb],
      [0x2b, 0x53, 0xd8, 0x5c, 0xa1, 0x14, 0x17, 0xfb, 0x23, 0xd5, 0x7d, 0x30,
0x67, 0x73, 0x08, 0x09],
```

```python
        [0xee, 0xb7, 0x70, 0x3f, 0x61, 0xb2, 0x19, 0x8e, 0x4e, 0xe5, 0x4b, 0x93,
0x8f, 0x5d, 0xdb, 0xa9],
        [0xad, 0xf1, 0xae, 0x2e, 0xcb, 0x0d, 0xfc, 0xf4, 0x2d, 0x46, 0x6e, 0x1d,
0x97, 0xe8, 0xd1, 0xe9],
        [0x4d, 0x37, 0xa5, 0x75, 0x5e, 0x83, 0x9e, 0xab, 0x82, 0x9d, 0xb9, 0x1c,
0xe0, 0xcd, 0x49, 0x89],
        [0x01, 0xb6, 0xbd, 0x58, 0x24, 0xa2, 0x5f, 0x38, 0x78, 0x99, 0x15, 0x90,
0x50, 0xb8, 0x95, 0xe4],
        [0xd0, 0x91, 0xc7, 0xce, 0xed, 0x0f, 0xb4, 0x6f, 0xa0, 0xcc, 0xf0, 0x02,
0x4a, 0x79, 0xc3, 0xde],
        [0xa3, 0xef, 0xea, 0x51, 0xe6, 0x6b, 0x18, 0xec, 0x1b, 0x2c, 0x80, 0xf7,
0x74, 0xe7, 0xff, 0x21],
        [0x5a, 0x6a, 0x54, 0x1e, 0x41, 0x31, 0x92, 0x35, 0xc4, 0x33, 0x07, 0x0a,
0xba, 0x7e, 0x0e, 0x34],
        [0x88, 0xb1, 0x98, 0x7c, 0xf3, 0x3d, 0x60, 0x6c, 0x7b, 0xca, 0xd3, 0x1f,
0x32, 0x65, 0x04, 0x28],
        [0x64, 0xbe, 0x85, 0x9b, 0x2f, 0x59, 0x8a, 0xd7, 0xb0, 0x25, 0xac, 0xaf,
0x12, 0x03, 0xe2, 0xf2]]

D = [0x44d7, 0x26bc, 0x626b, 0x135e, 0x5789, 0x35e2, 0x7135, 0x09af,
     0x4d78, 0x2f13, 0x6bc4, 0x1af1, 0x5e26, 0x3c4d, 0x789a, 0x47ac]


# 常量和初始值
lfsr_state = [0] * 16  # LFSR状态寄存器，初始化为全0
keystream = []  # 密钥流列表
keystream_length = 2  # 密钥流长度
recon_bits = [0] * 4  # 重构的比特位
reg1, reg2, out_word = 0, 0, 0  # 寄存器1，寄存器2和输出字
mod32 = 2 ** 32  # 32位模数


def bit_reconstruct():
    """
    执行比特重构，生成重构的比特位。
    """
    recon_bits[0] = combine_high_low_bits(lfsr_state[15], lfsr_state[14])
    recon_bits[1] = combine_low_high_bits(lfsr_state[11], lfsr_state[9])
    recon_bits[2] = combine_low_high_bits(lfsr_state[7], lfsr_state[5])
    recon_bits[3] = combine_low_high_bits(lfsr_state[2], lfsr_state[0])


def lfsr_initialize_mode(u):
    """
    执行LFSR初始化模式。
    """
    v = (2 ** 15 * lfsr_state[15] + 2 ** 17 * lfsr_state[13] + 2 ** 21 *
lfsr_state[10] +
         2 ** 20 * lfsr_state[4] + (1 + 2 ** 8) * lfsr_state[0]) % (2 ** 31 - 1)
    lfsr_state.append((v + u) % (2 ** 31 - 1))
    if lfsr_state[16] == 0:
        lfsr_state[16] = 2 ** 31 - 1
    lfsr_state.pop(0)


def lfsr_operate_mode():
    """
    执行LFSR工作模式。
```

```
    """
    lfsr_state.append((2 ** 15 * lfsr_state[15] + 2 ** 17 * lfsr_state[13] + 2
** 21 * lfsr_state[10] +
                       2 ** 20 * lfsr_state[4] + (1 + 2 ** 8) * lfsr_state[0]) %
(2 ** 31 - 1))
    if lfsr_state[16] == 0:
        lfsr_state[16] = 2 ** 31 - 1
    lfsr_state.pop(0)


def nonlinear_function(x0, x1, x2):
    """
    执行非线性函数F。
    """
    global reg1, reg2, out_word
    out_word = mod32_add(x0 ^ reg1, reg2)
    temp1 = mod32_add(reg1, x1)
    temp2 = reg2 ^ x2
    reg1 = sbox_transform(l1_transform(((temp1 << 16) | (temp2 >> 16)) &
0xffffffff))
    reg2 = sbox_transform(l2_transform(((temp2 << 16) | (temp1 >> 16)) &
0xffffffff))


def l1_transform(x):
    """
    执行线性变换L1。
    """
    return x ^ rotate_left(x, 2) ^ rotate_left(x, 10) ^ rotate_left(x, 18) ^
rotate_left(x, 24) & 0xffffffff


def l2_transform(x):
    """
    执行线性变换L2。
    """
    return x ^ rotate_left(x, 8) ^ rotate_left(x, 14) ^ rotate_left(x, 22) ^
rotate_left(x, 30) & 0xffffffff


def sbox_transform(x):
    """
    执行S盒变换。
    """
    bytes_ = [0, 0, 0, 0]
    transformed = [0, 0, 0, 0]
    bytes_[0] = x >> 24
    bytes_[1] = (x >> 16) & 0xff
    bytes_[2] = (x >> 8) & 0xff
    bytes_[3] = x & 0xff
    for i in range(4):
        row = bytes_[i] >> 4
        column = bytes_[i] & 0xf
        transformed[i] = S0[row][column] if i % 2 == 0 else S1[row][column]
    return (transformed[0] << 24) | (transformed[1] << 16) | (transformed[2] <<
8) | transformed[3]
```

```python
def rotate_left(x, i):
    """
    执行32位整数的左旋转。
    """
    return ((x << i) & 0xffffffff) | (x >> (32 - i))


def mod32_add(r, x):
    """
    执行32位整数的模加。
    """
    return (r + x) % mod32


def high_bits(x):
    """
    获取32位整数的高16位。
    """
    return (x >> 15) & 0xffff


def low_bits(x):
    """
    获取32位整数的低16位。
    """
    return x & 0xffff


def combine_low_high_bits(w1, w2):
    """
    组合第一个字的低位和第二个字的高位。
    """
    return (low_bits(w1) << 16) | high_bits(w2)


def combine_high_low_bits(w1, w2):
    """
    组合第一个字的高位和第二个字的低位。
    """
    return (high_bits(w1) << 16) | low_bits(w2)


def initialize_lfsr(key, iv):
    """
    使用密钥和IV初始化LFSR。
    """
    global reg1, reg2, out_word, lfsr_state
    for i in range(16):
        lfsr_state[i] = (key[i] << 23) | (D[i] << 8) | iv[i]
    for i in range(32):
        bit_reconstruct()
        nonlinear_function(recon_bits[0], recon_bits[1], recon_bits[2])
        lfsr_initialize_mode(out_word >> 1)


def generate_keystream():
    """
    计算密钥流。
```

```
    """
    bit_reconstruct()
    nonlinear_function(recon_bits[0], recon_bits[1], recon_bits[2])
    lfsr_operate_mode()
    for _ in range(keystream_length):
        bit_reconstruct()
        nonlinear_function(recon_bits[0], recon_bits[1], recon_bits[2])
        keystream.append(out_word ^ recon_bits[3])
        lfsr_operate_mode()


def display_keystream():
    """
    打印生成的密钥流。
    """
    for i in range(keystream_length):
        print(f'KEY{i}: {hex(keystream[i]).replace("0x", "")} ', end='')


def encrypt_decrypt_text(text):
    """
    使用生成的密钥流加密或解密给定的文本。
    """
    text_bytes = text.encode('utf-8')
    encrypted_decrypted_bytes = bytes(
        [b ^ (keystream[i % keystream_length] & 0xff) for i, b in
enumerate(text_bytes)])
    return encrypted_decrypted_bytes.decode('utf-8', errors='ignore')


if __name__ == '__main__':
    # 示例密钥和IV
    key = [0x3d, 0x4c, 0x4b, 0xe9, 0x6a, 0x82, 0xfd, 0xae, 0xb5, 0x8f, 0x64,
0x1d, 0xb1, 0x7b, 0x45, 0x5b]
    iv = [0x84, 0x31, 0x9a, 0xa8, 0xde, 0x69, 0x15, 0xca, 0x1f, 0x6b, 0xda,
0x6b, 0xfb, 0xd8, 0xc7, 0x66]

    # 初始化LFSR
    initialize_lfsr(key, iv)
    # 生成密钥流
    generate_keystream()
    # 打印密钥流
    display_keystream()

    # 要加密的消息
    message = "2024shangyingmima"
    print(f'\nmessage: {message}')

    print('----------------------------------------------')
    # 加密消息
    encrypted_message = encrypt_decrypt_text(message)
    print(f'Encrypted Message: {encrypted_message}')

    # 解密消息
    decrypted_message = encrypt_decrypt_text(encrypted_message)
    print(f'Decrypted Message: {decrypted_message}')
```