实验三 -- SM3杂凑算法的设计与实现

一、实验原理

SM3算法是一种加密哈希算法，用于生成256位（32字节）杂凑值。它主要分为以下几个步骤：

1. **消息填充**
2. **消息扩展**
3. **迭代压缩**
4. **生成杂凑值**

1. 消息填充

将原始消息 `m` 填充为长度为512位的倍数。

- 首先在消息 `m` 的末尾添加一个比特 `1`。
- 然后添加若干比特 `0`，使得总长度满足 `l + 1 + k ≡ 448 (mod 512)`，即总长度减去64位应该是512的倍数。
- 最后，添加消息长度 `l` 的64位二进制表示（大端序存储）。

填充后的消息 `m'` 长度为512的倍数。

2. 消息扩展

将填充后的消息 `m'` 按照512位分组，每组生成132个字。

- **划分消息分组**：将消息分成 `n` 个512位的分组，记为 `B^(0), B^(1), ..., B^(n-1)`。
- **生成扩展字**：
  - 初始16个字：`W_0, W_1, ..., W_15` 直接从 `B^(i)` 分组中提取。
  - 递推生成剩余的 `W_j`（`j=16` 到 `67`）：

    ```
    W_j = P_1(W_{j-16} ⊕ W_{j-9} ⊕ (W_{j-3} <<< 15)) ⊕ (W_{j-13} <<< 7) ⊕ W_{j-6}
    ```

  - 递推生成 `W_j'`（`j=0` 到 `63`）：

    ```
    W_j' = W_j ⊕ W_{j+4}
    ```

3. 迭代压缩

对每个消息分组 `B^(i)`，使用压缩函数 `CF` 进行迭代压缩。

- 初始值 `IV`：

  ```
  IV = 7380166f 4914b2b9 172442d7 da8a0600 a96f30bc 163138aa e38dee4d b0fb0e4e
  ```

- 压缩函数 `CF`：

  ```
  V^{(i+1)} = CF(V^{(i)}, B^(i))
  ```

  其中，`V^{(0)}` 为初始值 `IV`，`B^(i)` 为消息分组。

4. 压缩函数 CF

压缩函数将当前值和消息分组进行处理，输出新的压缩值。

- **寄存器**：A, B, C, D, E, F, G, H 初始化为 `V^{(i)}`。
- **布尔函数**：

```
FF_j(X, Y, Z) = (X ⊕ Y ⊕ Z)  (0 ≤ j ≤ 15)
FF_j(X, Y, Z) = (X ∧ Y) ∨ (X ∧ Z) ∨ (Y ∧ Z) (16 ≤ j ≤ 63)

GG_j(X, Y, Z) = (X ⊕ Y ⊕ Z)  (0 ≤ j ≤ 15)
GG_j(X, Y, Z) = (X ∧ Y) ∨ (¬X ∧ Z) (16 ≤ j ≤ 63)
```

- **置换函数**：

```
P_0(X) = X ⊕ (X <<< 9) ⊕ (X <<< 17)
P_1(X) = X ⊕ (X <<< 15) ⊕ (X <<< 23)
```

- **常量**：

```
T_j =
{
  79cc4519 (0 ≤ j ≤ 15)
  7a879d8a (16 ≤ j ≤ 63)
}
```

- **压缩过程**：

```
FOR j=0 TO 63
  SS1 = ((A <<< 12) + E + (T_j <<< j)) <<< 7
  SS2 = SS1 ⊕ (A <<< 12)
  TT1 = FF_j(A, B, C) + D + SS2 + W_j'
  TT2 = GG_j(E, F, G) + H + SS1 + W_j
  D = C
  C = B <<< 9
  B = A
  A = TT1
  H = G
  G = F <<< 19
  F = E
  E = P_0(TT2)
ENDFOR
V^{(i+1)} = ABCDEFGH ⊕ V^{(i)}
```

5. 生成杂凑值

最终的杂凑值为 `V^{(n)}`，即 `ABCDEFGH` 的串联表示。

总结

SM3算法的结构简洁明了，包括消息填充、消息扩展、迭代压缩和生成最终杂凑值四个主要步骤。其安全性通过复杂的非线性布尔函数和置换函数以及多轮迭代压缩过程得到保障。

二、实验环境

- 编程语言：Python
- 运行平台：任意支持Python的环境

三、代码概述

此代码实现了中国国家密码管理局发布的密码算法——SM3哈希函数。该算法用于生成消息摘要，具有类似于SHA-256的功能。代码主要包括以下部分：

1. **初始向量**：用于初始化哈希值。
2. **常量 (T_j)**：在算法中使用的固定常量。
3. **辅助函数**：包括循环左移函数、布尔函数FF和GG、置换函数P0和P1。
4. **压缩函数CF**：对消息分组进行处理并更新哈希值。
5. **SM3哈希函数**：对输入消息进行预处理，分组，然后使用压缩函数生成最终哈希值。
6. **密钥派生函数**：使用SM3生成指定长度的密钥。

初始向量与常量 (T_j)

```
# 初始向量
INITIAL_VECTOR = [
    1937774191, 1226093241, 388252375, 3666478592,
    2842636476, 372324522, 3817729613, 2969243214,
]

# 常量 T_j
CONSTANT_T = [
                 2043430169] * 16 + [2055708042] * 48
```

初始向量和常量 (T_j) 是SM3算法的固定参数，用于初始化哈希状态和在压缩过程中使用。

循环左移函数

```
def rotate_left(x, n):
    """对x进行循环左移n位"""
    return ((x << n) & 0xffffffff) | ((x >> (32 - n)) & 0xffffffff)
```

该函数实现了对32位整数的循环左移操作。

布尔函数 FF 和 GG

```
def boolean_function_FF(x, y, z, j):
    """定义布尔函数FF"""
    if 0 <= j < 16:
        ret = x ^ y ^ z
    elif 16 <= j < 64:
        ret = (x & y) | (x & z) | (y & z)
    return ret

def boolean_function_GG(x, y, z, j):
    """定义布尔函数GG"""
    if 0 <= j < 16:
        ret = x ^ y ^ z
```

```
    elif 16 <= j < 64:
        ret = (x & y) | ((~ x) & z)
    return ret
```

布尔函数 FF 和 GG 是根据消息分组的不同阶段定义的，用于在压缩函数中处理输入数据。

置换函数 P0 和 P1

```
def permutation_function_P0(x):
    """定义置换函数P0"""
    return x ^ (rotate_left(x, 9 % 32)) ^ (rotate_left(x, 17 % 32))

def permutation_function_P1(x):
    """定义置换函数P1"""
    return x ^ (rotate_left(x, 15 % 32)) ^ (rotate_left(x, 23 % 32))
```

置换函数 P0 和 P1 在扩展消息分组时使用，用于增加数据的非线性。

压缩函数 CF

```
def sm3_compress(v_i, b_i):
    """定义CF函数"""
    w = []
    for i in range(16):
        weight = 0x1000000
        data = 0
        for k in range(i * 4, (i + 1) * 4):
            data = data + b_i[k] * weight
            weight = int(weight / 0x100)
        w.append(data)

    for j in range(16, 68):
        w.append(0)
        w[j] = permutation_function_P1(w[j - 16] ^ w[j - 9] ^ (rotate_left(w[j -
3], 15 % 32))) ^ (
            rotate_left(w[j - 13], 7 % 32)) ^ w[j - 6]

    w_1 = []
    for j in range(64):
        w_1.append(0)
        w_1[j] = w[j] ^ w[j + 4]

    a, b, c, d, e, f, g, h = v_i

    for j in range(64):
        ss_1 = rotate_left((rotate_left(a, 12 % 32) + e +
rotate_left(CONSTANT_T[j], j % 32)) & 0xffffffff, 7 % 32)
        ss_2 = ss_1 ^ (rotate_left(a, 12 % 32))
        tt_1 = (boolean_function_FF(a, b, c, j) + d + ss_2 + w_1[j]) &
0xffffffff
        tt_2 = (boolean_function_GG(e, f, g, j) + h + ss_1 + w[j]) & 0xffffffff
        d = c
        c = rotate_left(b, 9 % 32)
        b = a
        a = tt_1
        h = g
```

```
        g = rotate_left(f, 19 % 32)
        f = e
        e = permutation_function_P0(tt_2)

        a, b, c, d, e, f, g, h = map(
            lambda x: x & 0xFFFFFFFF, [a, b, c, d, e, f, g, h])

    v_j = [a, b, c, d, e, f, g, h]
    return [v_j[i] ^ v_i[i] for i in range(8)]
```

压缩函数 CF 对每个消息分组进行处理，并更新哈希状态。函数首先对消息分组进行扩展，然后通过64轮迭代对初始向量进行更新。

SM3哈希函数

```
def sm3_hash(message):
    """计算SM3哈希值"""
    length = len(message)
    reserve = length % 64
    message.append(0x80)
    reserve = reserve + 1

    range_end = 56
    if reserve > range_end:
        range_end = range_end + 64

    for i in range(reserve, range_end):
        message.append(0x00)

    bit_length = (length) * 8
    bit_length_str = [bit_length % 0x100]
    for i in range(7):
        bit_length = int(bit_length / 0x100)
        bit_length_str.append(bit_length % 0x100)
    for i in range(8):
        message.append(bit_length_str[7 - i])

    group_count = round(len(message) / 64)

    blocks = []
    for i in range(group_count):
        blocks.append(message[i * 64:(i + 1) * 64])

    V = []
    V.append(INITIAL_VECTOR)
    for i in range(group_count):
        V.append(sm3_compress(V[i], blocks[i]))

    y = V[i + 1]
    result = ""
    for num in y:
        result = '%s%08x' % (result, num)
    return result
```

SM3哈希函数首先对消息进行预处理（填充和附加长度），然后将消息分组，每组64字节，使用初始向量和压缩函数处理每个分组，最后返回哈希值。

密钥派生函数

```
def sm3_key_derivation_function(z, key_length):
    """使用SM3进行密钥派生"""
    key_length = int(key_length)
    count = 0x00000001
    rounds = ceil(key_length / 32)
    z_bytes = [i for i in bytes.fromhex(z.decode('utf8'))]
    derived_key = ""
    for i in range(rounds):
        msg = z_bytes + [i for i in binascii.a2b_hex(('%08x' %
count).encode('utf8'))]
        derived_key += sm3_hash(msg)
        count += 1
    return derived_key[0: key_length * 2]
```

该函数使用SM3哈希函数从输入值 (z) 派生指定长度的密钥。通过迭代SM3哈希函数，生成所需长度的密钥。

主程序

```
if __name__ == '__main__':
    # 要哈希的消息
    message = '20242021300442 6liuhaoran'

    # 将消息转换为字节数组
    message_bytes = bytearray(message, 'utf-8')

    # 使用SM3哈希函数进行哈希
    hash_result = sm3_hash(message_bytes)

    # 打印哈希结果
    print(f'哈希结果: {hash_result}')
```

输出:

```
哈希结果: ee9a3ff8b3707bdb731fba69f8c28179efb082eb9117b69336e00c7a4ee3fb7a
```

四、实验结论

通过上述代码，我们成功实现了SM3哈希算法，并计算了给定消息的哈希值。代码详细展示了SM3哈希函数的各个组成部分，包括预处理、分组、压缩以及哈希值的计算过程。最终生成的哈希值证明了SM3算法的正确性和有效性。

附录:代码

```
import binascii
from math import ceil

# 初始向量
INITIAL_VECTOR = [
    1937774191, 1226093241, 388252375, 3666478592,
```

```
    2842636476, 372324522, 3817729613, 2969243214,
]

# 常量 T_j
CONSTANT_T = [
                 2043430169] * 16 + [2055708042] * 48


# 循环左移函数
def rotate_left(x, n):
    """对x进行循环左移n位"""
    return ((x << n) & 0xffffffff) | ((x >> (32 - n)) & 0xffffffff)


# 定义布尔函数 FF
def boolean_function_FF(x, y, z, j):
    """定义布尔函数FF"""
    if 0 <= j < 16:
        ret = x ^ y ^ z
    elif 16 <= j < 64:
        ret = (x & y) | (x & z) | (y & z)
    return ret


# 定义布尔函数 GG
def boolean_function_GG(x, y, z, j):
    """定义布尔函数GG"""
    if 0 <= j < 16:
        ret = x ^ y ^ z
    elif 16 <= j < 64:
        ret = (x & y) | ((~ x) & z)
    return ret


# 定义置换函数 P0
def permutation_function_P0(x):
    """定义置换函数P0"""
    return x ^ (rotate_left(x, 9 % 32)) ^ (rotate_left(x, 17 % 32))


# 定义置换函数 P1
def permutation_function_P1(x):
    """定义置换函数P1"""
    return x ^ (rotate_left(x, 15 % 32)) ^ (rotate_left(x, 23 % 32))


# CF函数
def sm3_compress(v_i, b_i):
    """定义CF函数"""
    w = []
    for i in range(16):
        weight = 0x1000000
        data = 0
        for k in range(i * 4, (i + 1) * 4):
            data = data + b_i[k] * weight
            weight = int(weight / 0x100)
        w.append(data)
```

```
    for j in range(16, 68):
        w.append(0)
        w[j] = permutation_function_P1(w[j - 16] ^ w[j - 9] ^ (rotate_left(w[j -
3], 15 % 32))) ^ (
            rotate_left(w[j - 13], 7 % 32)) ^ w[j - 6]

    w_1 = []
    for j in range(64):
        w_1.append(0)
        w_1[j] = w[j] ^ w[j + 4]

    a, b, c, d, e, f, g, h = v_i

    for j in range(64):
        ss_1 = rotate_left((rotate_left(a, 12 % 32) + e +
rotate_left(CONSTANT_T[j], j % 32)) & 0xffffffff, 7 % 32)
        ss_2 = ss_1 ^ (rotate_left(a, 12 % 32))
        tt_1 = (boolean_function_FF(a, b, c, j) + d + ss_2 + w_1[j]) &
0xffffffff
        tt_2 = (boolean_function_GG(e, f, g, j) + h + ss_1 + w[j]) & 0xffffffff
        d = c
        c = rotate_left(b, 9 % 32)
        b = a
        a = tt_1
        h = g
        g = rotate_left(f, 19 % 32)
        f = e
        e = permutation_function_P0(tt_2)

        a, b, c, d, e, f, g, h = map(
            lambda x: x & 0xFFFFFFFF, [a, b, c, d, e, f, g, h])

    v_j = [a, b, c, d, e, f, g, h]
    return [v_j[i] ^ v_i[i] for i in range(8)]


# SM3哈希函数
def sm3_hash(message):
    """计算SM3哈希值"""
    length = len(message)
    reserve = length % 64
    message.append(0x80)
    reserve = reserve + 1

    range_end = 56
    if reserve > range_end:
        range_end = range_end + 64

    for i in range(reserve, range_end):
        message.append(0x00)

    bit_length = (length) * 8
    bit_length_str = [bit_length % 0x100]
    for i in range(7):
        bit_length = int(bit_length / 0x100)
        bit_length_str.append(bit_length % 0x100)
    for i in range(8):
        message.append(bit_length_str[7 - i])
```

```
    group_count = round(len(message) / 64)

    blocks = []
    for i in range(group_count):
        blocks.append(message[i * 64:(i + 1) * 64])

    V = []
    V.append(INITIAL_VECTOR)
    for i in range(group_count):
        V.append(sm3_compress(V[i], blocks[i]))

    y = V[i + 1]
    result = ""
    for num in y:
        result = '%s%08x' % (result, num)
    return result


# 定义SM3密钥派生函数
def sm3_key_derivation_function(z, key_length):
    """使用SM3进行密钥派生"""
    key_length = int(key_length)
    count = 0x00000001
    rounds = ceil(key_length / 32)
    z_bytes = [i for i in bytes.fromhex(z.decode('utf8'))]
    derived_key = ""
    for i in range(rounds):
        msg = z_bytes + [i for i in binascii.a2b_hex(('%08x' %
count).encode('utf8'))]
        derived_key += sm3_hash(msg)
        count += 1
    return derived_key[0: key_length * 2]


if __name__ == '__main__':
    # 要哈希的消息
    message = '20242021300442 6liuhaoran'

    # 将消息转换为字节数组
    message_bytes = bytearray(message, 'utf-8')

    # 使用SM3哈希函数进行哈希
    hash_result = sm3_hash(message_bytes)

    # 打印哈希结果
    print(f'哈希结果: {hash_result}')
```