1. 实验独自完成，小组成员：于奥玮（201900210032）。Git 账户名称：yuaowei。

2. 项目简介：

（1）完成了 SM3 加密算法的实现。（名称：SM3_achieve）

（2）完成了 SM3 加密算法的优化。(名称：SM3_optimize1,SM3_optimize2)

3.清单：完成了 SM3 加密算法的实现以及优化，无有问题的项目。

4.详细说明：

SM3：初步实现了 SM3 加密算法的部分过程，未给出正确结果。

SM3_update:完全实现了 SM3 加密算法，但是未实现分组功能，对于超过 512bit 的明文无法加密。

运行指导：直接运行即可。如果需要更改明文，在 plain_text 改即可。

```
start_time = time.time()
plain_text = b'abc'
result=convert(plain_text)
print(result)
```

运行截图：（IV 即为运行结果）

```
D:\python3.9\python.exe "E:/github market/SM3/SM3_update/main.py"
01100001011000100110001110000000000000000000000000000000000000000000000000000000000000000000000000000000
['01100001011000100110001110000000', '00000000000000000000000000000000', '00000000000000000000000
['01100001011000100110001110000000', '00000000000000000000000000000000', '00000000000000000000000
IV: [['01100110110001111111000011110100'], ['01100010111011101110110111011001'], ['11010001111100
time: 0.0034034252166748047 s

Process finished with exit code 0
```

SM3_achieve：完全实现了 SM3 的加密功能。

运行指导：直接运行即可。如果需要更改明文，在 plain_text 改即可。

```
start_time = time.time()

plain_text = b'abc'*30     #明文

result=convert(plain_text)
```

运行截图：

```
D:\python3.9\python.exe "E:/github market/SM3/SM3_achieve/main.py"
convert: 0110000101100010011000110110000101100010011000110110000101100010011000110110000
num: 2
result: ['0x01daa9d5', '0x8cf8328a', '0x48bd0032', '0xc137d875', '0x86071790', '0xcabd6126', '0xb7862abb', '0xf7
time: 0.00399017333984375 s
```

convert：为明文填充后的结果。

num：为分组的个数。

result：即为加密结果。

time：为运行时间。

SM3_optimize1：采用消息扩展的快速实现的优化方法进行优化，消息扩展的目的是利用 512 比特的消息分组 B 扩展得到 68 个字，快速实现时，为了尽可能减少不必要的数据加载和存储，将W_0，…,W_{67}和W_0'，…,W_{63}'的实现放在压缩函数中进行：

```
for j in range(64):
    if j<12:
        W.append(P[(j+4)*32:(j+5)*32])
    else:
        W.append(xor(P1(xor(_W[j-12],W[j-5],(shift(W[j+1],15)))),shift(W[j-9],7),W[j-2]_))
    W1.append(BIN(int(W[j],2)^int(W[j+4],2)))
    SS1=shift(BIN((int(shift(A,12),2)+int(E,2)+int(shift(T(j),j%32),2))%(2**32)),7)
    SS2=BIN(int(SS1,2)^int(shift(A,12),2))
    TT1=BIN((int(FF(A,B,C,j),2)+int(D,2)+int(SS2,2)+int(W1[j],2))%(2**32))
    TT2=BIN((int(GG(E,F,G,j),2)+int(H,2)+int(SS1,2)+int(W[j],2))%(2**32))
```

运行指导：直接运行即可，如果需要更改明文，在 plain_text 改即可。

```
start_time = time.time()

plain_text = b'abc'    #明文

result=convert(plain_text)
```

运行截图：

```
D:\python3.9\python.exe "E:/github market/SM3/SM3_optimize1/main.py"
convert: 01100001011000100110001110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
num: 1
result: ['0x66c7f0f4', '0x62eeedd9', '0xd1f2d46b', '0xdc10e4e2', '0x4167c487', '0x5cf2f7a2', '0x297da02b',
time: 0.0015263557434082031 s
```

此处和 SM3_achive 内容相同，运行时间缩短，代码得到优化。

SM3_optimize2：采用预计算的方法实现加速，预先计算并存储常数$t_i = T_i <<< i$。 这可以避免每个消息分组都去计算常数，且占用的存储空间也很少，仅256Byte.

```
def T(j):
    if j<16:
        result = "01111001110011000100010100011001"          #79cc4519
    if j>15:
        result = "01111010100001111001110110001010"          #7a879d8a
    return result
def T_new(j):
    if j<16:
        return shift("01111001110011000100010100011001",j%32)
    if j>15:
        return shift("01111010100001111001110110001010",j%32)
```

```
T_new_array = []
for i in range(64):
    T_new_array.append(T_new(i))
```

运行指导：直接运行即可。如果需要更改明文，在 plain_text 改即可。

```
start_time = time.time()

plain_text = b'abc'    #明文

result=convert(plain_text)
```

运行截图：

```
D:\python3.9\python.exe "E:/github market/SM3/SM3_optimize2/main.py"
convert: 01100001011000100110001110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
num: 1
result: ['0x66c7f0f4', '0x62eeedd9', '0xd1f2d46b', '0xdc10e4e2', '0x4167c487', '0x5cf2f7a2', '0x297da02b'
time: 0.0009982585906982422 s
```

此处和 SM3_achive 内容相同，运行时间缩短，代码再次得到优化。