

密码学实验实验报告六

18374480-黄翔

2021 年 4 月 26 日

1 实验目的

1. 通过本次实验, 了解并掌握各种工作模式。
2. 感受工作模式与填充方式对安全性的意义。

2 实验环境

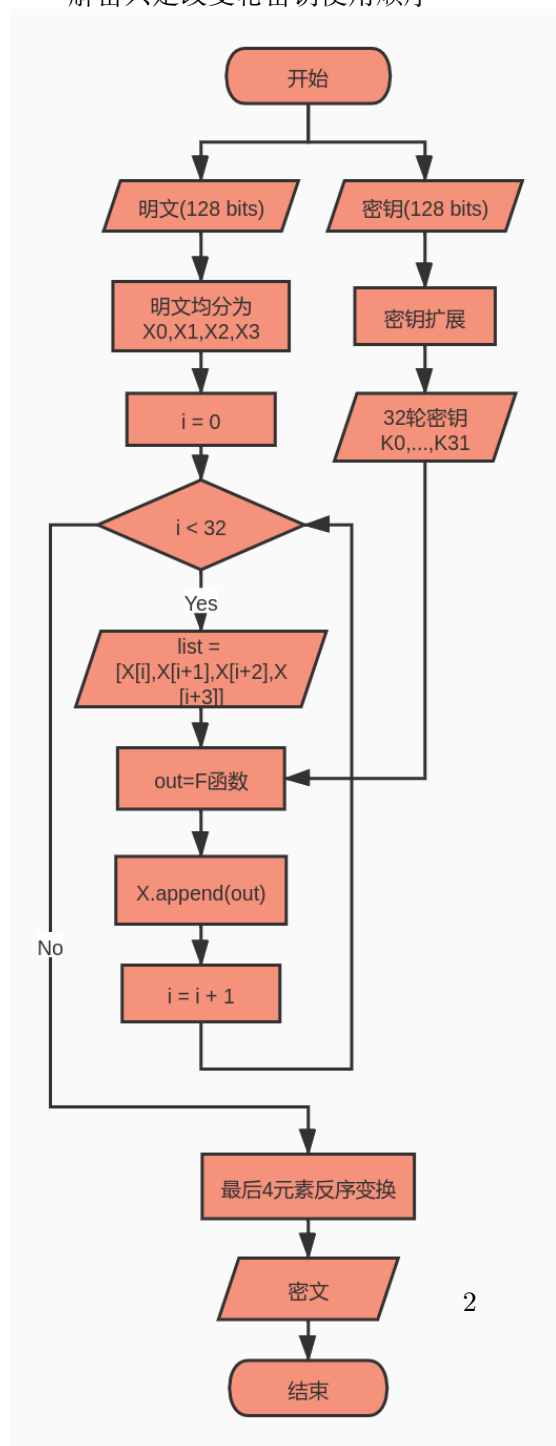
1. python 2 for required part
2. python 3 for optinal part

3 实验内容

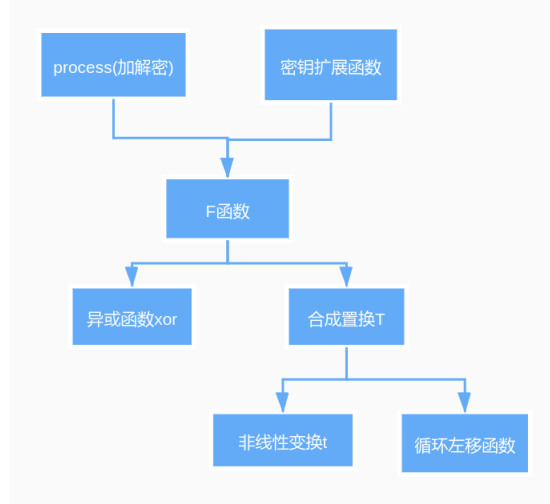
3.1 SM4 算法

3.1.1 算法流程图

解密只是改变轮密钥使用顺序



3.1.2 函数调用关系



3.1.3 算法伪代码

Algorithm 1 SM4 加密

Input: 明文 P (128 bits), 密钥 K (128 bits)

Output: 密文 C (128 bits)

```

1: function Encrypt( $P, K$ )
2:    $Key[32] \leftarrow$  密钥扩展函数  $getRoundKey(K)$ 
3:    $X \leftarrow [P[0 : 32], P[32 : 64], P[64 : 96], P[96 : 128]]$ 
4:   for  $i \leftarrow 0$  to 31 do
5:      $lst \leftarrow [X[i], X[i + 1], X[i + 2], X[i + 3]]$ 
6:      $out \leftarrow$  轮函数  $F(lst, K[i])$ 
7:      $X.append(out)$ 
8:   end for
9:   return  $C \leftarrow X[35] || X[34] || X[33] || X[32]$ 
10: end function
  
```

Algorithm 2 SM4 解密

Input: 密文 C (128 bits), 密钥 K (128 bits)

Output: 明文 P (128 bits)

```

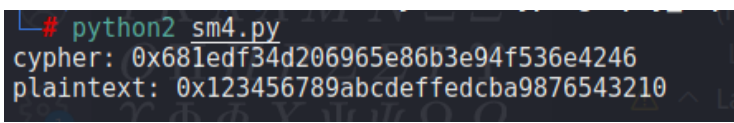
1: function Encrypt( $P, K$ )
2:    $Key[32] \leftarrow$  密钥扩展函数  $getRoundKey(K)$ 
  
```

```

3:    $X \leftarrow [C[0 : 32], C[32 : 64], C[64 : 96], C[96 : 128]]$ 
4:   for  $i \leftarrow 0$  to 31 do
5:        $lst \leftarrow [X[i], X[i + 1], X[i + 2], X[i + 3]]$ 
6:        $out \leftarrow \text{轮函数 } F(lst, K[31 - i])$ 
7:        $X.append(out)$ 
8:   end for
9:   return  $P \leftarrow X[35] || X[34] || X[33] || X[32]$ 
10: end function

```

3.1.4 测试样例及结果截图



```

# python2 sm4.py
cypher: 0x681edf34d206965e86b3e94f536e4246
plaintext: 0x123456789abcdeffedcba9876543210

```

3.1.5 总结

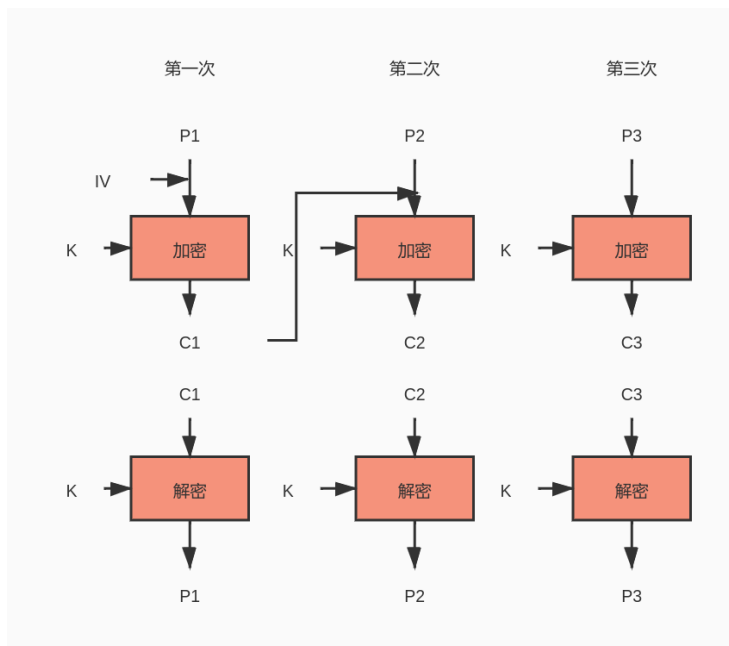
SM4 算法结构 SM4 分组密码算法采用非平衡 Feistel 结构，加密解密结构相同，只是轮密钥的使用顺序不同。这大大降低了软硬件实现难度

SM4 的混淆与扩散

1. **混淆**：轮密钥异或后进入 T 合成置换中的非线性变换 τ
2. **扩散**：轮函数 F 实现字间的大扩散，线性变换 L 实现字节间的中扩散，S 盒设计体现每个字节内的小扩散

3.2 ECB 工作模式

3.2.1 算法流程图



3.2.2 算法伪代码

Algorithm 3 SM4 ECB 加密

Input: 明文 P , 密钥 K (128 bits)

Output: 密文 C

- 1: $P \leftarrow$ 对 P 使用 $PKCS\#7$ 填充
 - 2: $lstP \leftarrow [P[0 : 128], P[128 : 258] \dots]$
 - 3: **for** e in $lstP$ **do**
 - 4: $C = C + SM4_encrypt(e, K)$
 - 5: **end for**
 - 6: **return** C
-

Algorithm 4 SM4 ECB 解密

Input: 密文 C , 密钥 K (128 bits)

Output: 明文 P

- 1: $lstC \leftarrow [C[0 : 128], C[128 : 258] \dots]$

- 2: **for** e in $lstC$ **do**
- 3: $P = P + SM4_decrypt(e, K)$
- 4: **end for**
- 5: **return** $P \leftarrow$ 对 P 使用 $PKCS\#7$ 去除填充

3.2.3 测试样例及结果截图

```

[ root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4 ]
# openssl dgst -sha256 -ecb cipher-ecb
SHA256(ecb)= c90ac687b9a32825d7f259747d6adde34f98ccfae116bbc2c9aee118741a2db1
SHA256(cipher-ecb)= c90ac687b9a32825d7f259747d6adde34f98ccfae116bbc2c9aee118741a2db1

[ root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4 ]
# openssl dgst -sha256 message message-ecb
SHA256(message)= df7ef99947051fba7b3708408939397cc613ce84916f25d7ab0fc7288fc449cf
SHA256(message-ecb)= df7ef99947051fba7b3708408939397cc613ce84916f25d7ab0fc7288fc449cf

[ root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4 ]
# xxd ecb
00000000: 681e df34 d206 965e 86b3 e94f 536e 4246  h..4...^...05nBF
00000010: 681e df34 d206 965e 86b3 e94f 536e 4246  h..4...^...05nBF
00000020: 681e df34 d206 965e 86b3 e94f 536e 4246  h..4...^...05nBF
00000030: ccd1 b5b3 6941 e70a 3916 d62e dc78 a969  ....1A..9....x.i

[ root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4 ]
# xxd cipher-ecb
00000000: 681e df34 d206 965e 86b3 e94f 536e 4246  h..4...^...05nBF
00000010: 681e df34 d206 965e 86b3 e94f 536e 4246  h..4...^...05nBF
00000020: 681e df34 d206 965e 86b3 e94f 536e 4246  h..4...^...05nBF
00000030: ccd1 b5b3 6941 e70a 3916 d62e dc78 a969  ....1A..9....x.i

```

3.2.4 总结

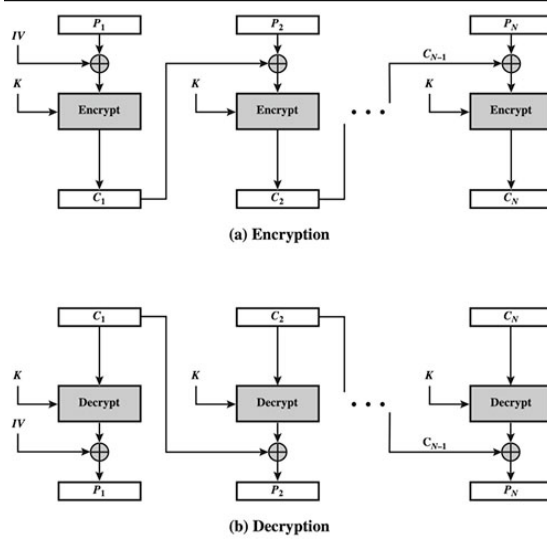
工作模式并行性 ECB 工作模式加密无法并行，解密可以并行

错误扩散 密文传输过程中的 1 bit 错误只会影响该组明文的恢复

安全性 同样的明文块会被加密成相同的密文块，因此 ECB 模式它不能很好的隐藏数据模式，如果消息是结构化的，密码分析者可以利用这些规律性特征来破译

3.3 CBC 工作模式

3.3.1 算法流程图



3.3.2 算法伪代码

Algorithm 5 SM4 CBC 加密

Input: 明文 P , 密钥 K (128 bits), 初始向量 IV

Output: 密文 C

```

1:  $P \leftarrow$  对  $P$  使用 PKCS#7 填充
2:  $lstP \leftarrow [P[0 : 128], P[128 : 258] \dots]$ 
3: for  $i \leftarrow 0$  to  $len(lstP)$  do
4:   if  $i$  is 0 then
5:      $lstP[i] = lstP[i] \oplus IV$ 
6:   else
7:      $lstP[i] = lstP[i] \oplus lstC[i - 1]$ 
8:   end if
9:    $lstC.append(SM4\_encrypt(lstP[i], K))$ 
10: end for
11: return  $C \leftarrow ".join(lstC)$ 

```

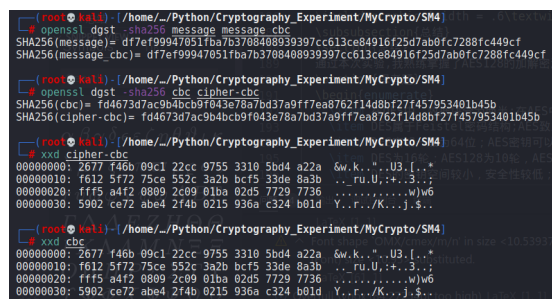
Algorithm 6 SM4 CBC 解密

Input: 密文 C , 密钥 K (128 bits), 初始向量 IV

Output: 明文 P

```
1:  $lstC \leftarrow [C[0 : 128], C[128 : 258] \dots]$ 
2: for  $i \leftarrow 0$  to  $len(lstC)$  do
3:    $temp = SM4\_decrypt(lstC[i], K)$ 
4:   if  $i$  is 0 then
5:      $lstP[i] = temp \oplus IV$ 
6:   else
7:      $lstP[i] = temp \oplus lstC[i - 1]$ 
8:   end if
9: end for
10:  $P \leftarrow ".join(lstP)$ 
11:  $P \leftarrow$  对  $P$  使用 PKCS#7 去除填充
```

3.3.3 测试样例及结果截图



```
(root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4)
# openssl dgst -sha256 message.cbc
SHA256(message)= df7ef99947051fba7b3708408939397cc613ce84916f25d7ab0fc7288fc449cf
SHA256(message_cbc)= df7ef99947051fba7b3708408939397cc613ce84916f25d7ab0fc7288fc449cf

(root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4)
# openssl dgst -sha256 cbc cipher-cbc
SHA256(cbc)= fd4673d7ac9b4bcb9f043e78a7bd37a9ff7ea8762f14d8bf27f457953401b45b
SHA256(cipher-cbc)= fd4673d7ac9b4bcb9f043e78a7bd37a9ff7ea8762f14d8bf27f457953401b45b

(root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4)
# xxd cipher-cbc
00000000: 2677 f46b 09c1 22cc 0755 3310 5bd4 a22a 6w.k...U3.[...
00000010: f612 5f72 75ce 552c 3a2b bcf5 33de 8a3b ..ru.U;+.3.;
00000020: fff5 a4f2 0809 2c09 01ba 02d5 7729 7736 .....W)w6
00000030: 5902 ce72 abe4 2f4b 0215 936a c324 b01d Y..r./K...j.$..

(root@kali:~/Python/Cryptography_Experiment/MyCrypto/SM4)
# xxd cbc
00000000: 2677 f46b 09c1 22cc 0755 3310 5bd4 a22a 6w.k...U3.[...
00000010: f612 5f72 75ce 552c 3a2b bcf5 33de 8a3b ..ru.U;+.3.;
00000020: fff5 a4f2 0809 2c09 01ba 02d5 7729 7736 .....W)w6
00000030: 5902 ce72 abe4 2f4b 0215 936a c324 b01d Y..r./K...j.$..
```

3.3.4 总结

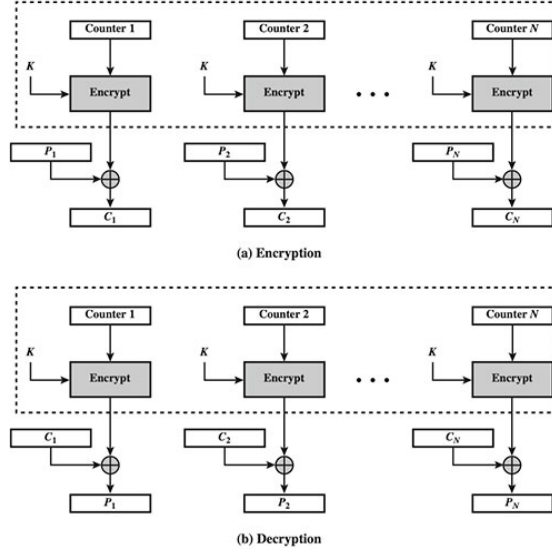
工作模式并行性 ECB 工作模式加密无法并行，解密可以并行

错误扩散 密文传输过程中的 1 bit 错误会影响该组明文的恢复以及下一组明文的恢复

其它 CBC 除了用来获得保密性，亦可用于认证。在加密时，明文中的微小改变会导致其后的全部密文块发生改变。

3.4 CTR 工作模式

3.4.1 算法流程图



3.4.2 算法伪代码

Algorithm 7 SM4 CTR 加密

Input: 明文 P , 密钥 K (128 bits), 计数器 $Counter$

Output: 密文 C

- 1: $lstP \leftarrow [P[0 : 128], P[128 : 258] \dots]$
 - 2: **for** $i \leftarrow 0$ to $len(lstP)$ **do**
 - 3: $temp \leftarrow SM4_encrypt(K, Counter[i])$
 - 4: $lstC[i].append(lstP[i] \oplus temp)$
 - 5: **end for**
 - 6: **return** $C \leftarrow ".join(lstC)$
-

Algorithm 8 SM4 CTR 解密

Input: 密文 C , 密钥 K (128 bits), 计数器 $Counter$

Output: 明文 P

- 1: $lstC \leftarrow [C[0 : 128], C[128 : 258] \dots]$
- 2: **for** $i \leftarrow 0$ to $len(lstC)$ **do**
- 3: $temp \leftarrow SM4_encrypt(K, Counter[i])$

```

4:     lstP[i].append(lstC[i] ⊕ temp)
5: end for
6: P ← ".join(lstP)

```

3.4.3 测试样例及结果截图

```

root@kali: ~/Python/Cryptography_Experiment/MyCrypto/SM4
# openssl dgst -sha256 message message_ctr
SHA256(message)= df7ef99947051fba7b3708408939397cc613ce84916f25d7ab0fc7288fc449cf
SHA256(message_ctr)= df7ef99947051fba7b3708408939397cc613ce84916f25d7ab0fc7288fc449cf

root@kali: ~/Python/Cryptography_Experiment/MyCrypto/SM4
# openssl dgst -sha256 ctr cipher-ctr
SHA256(ctr)= 0a976c964380965ec5ca9c15e6f06ab0fd6176e499c347e10398abeb848ba532
SHA256(cipher-ctr)= 0a976c964380965ec5ca9c15e6f06ab0fd6176e499c347e10398abeb848ba532

root@kali: ~/Python/Cryptography_Experiment/MyCrypto/SM4
# xxd ctr
00000000: 693d 9a53 5bad 5bb1 786f 53d7 253a 7056  1=.S[.].xoS.%:pV
00000010: bfb9 610e b9d1 5b16 2de1 6175 3aa7 ab84  ..a...[...au:...
00000020: c5c6 bc68 df1e 250c 0e99 244a 033e 4ef9  ...h.%...$j.>N.
00000030: d77e ca74 38bf ee95  --.t8...

root@kali: ~/Python/Cryptography_Experiment/MyCrypto/SM4
# xxd cipher-ctr
00000000: 693d 9a53 5bad 5bb1 786f 53d7 253a 7056  1=.S[.].xoS.%:pV
00000010: bfb9 610e b9d1 5b16 2de1 6175 3aa7 ab84  ..a...[...au:...
00000020: c5c6 bc68 df1e 250c 0e99 244a 033e 4ef9  ...h.%...$j.>N.
00000030: d77e ca74 38bf ee95  --.t8...

```

3.4.4 总结

工作模式并行性 ECB 工作模式加密、解密都可以并行（但需要能事先存储全部计数器值）

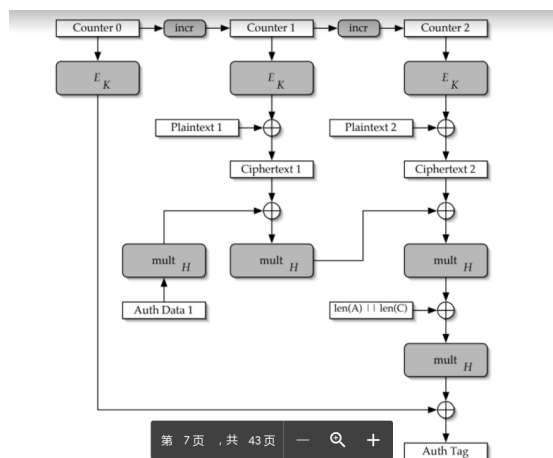
错误扩散 密文传输过程中的 1 bit 错误只会影响该组明文的恢复

其它

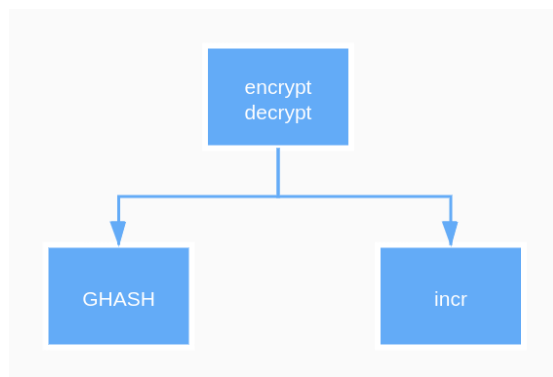
1. 由于 CTR 的结构的关系，并不需要对明文填充
2. 与 OFB 相似，CTR 将块密码变为流密码。它通过递增一个加密计数器以产生连续的密钥流，其中计数器可以是任意保证长时间不产生重复输出的函数

3.5 AES-GCM 加解密

3.5.1 算法流程图



3.5.2 函数调用关系



3.5.3 算法伪代码

Algorithm 9 AES-GCM 加密

Input: 明文 P , 密钥 K (128 bits), 初始向量 IV , A

Output: 密文 C , 认证标签 T

- 1: $H \leftarrow AES_encrypt(0, K)$
- 2: **if** $\text{len}(IV)$ is 24 **then**
- 3: $Y_0 \leftarrow IV || 0^{31}1$
- 4: **else**

```

5:    $Y_0 \leftarrow GHASH(H, IV)$ 
6: end if
7:  $P_1 || P_2 || \dots || P_n^* \leftarrow P$ 
8: for  $i \leftarrow 1$  to  $n$  do
9:    $Y_i \leftarrow incr(Y_{i-1})$ 
10: end for
11: for  $i \leftarrow 1$  to  $n - 1$  do
12:    $C_i \leftarrow P_i \oplus AES\_encrypt(Y_i, K)$ 
13: end for
14:  $C_n^* \leftarrow P_n^* \oplus MSB_u(AES\_encrypt(Y_n, K))$ 
15:  $C \leftarrow C_1 || C_2 || \dots || C_n^*$ 
16:  $T \leftarrow GHASH(H, A, C) \oplus AES\_encrypt(Y_0, K)$ 
17: return  $C, T$ 

```

Algorithm 10 AES-GCM 解密

Input: 密文 C , 密钥 K (128 bits), 初始向量 IV , A , 认证标签 T'

Output: 明文 P

```

1:  $H \leftarrow AES\_encrypt(0, K)$ 
2: if  $\text{len}(IV)$  is 24 then
3:    $Y_0 \leftarrow IV || 0^{31}1$ 
4: else
5:    $Y_0 \leftarrow GHASH(H, IV)$ 
6: end if
7:  $T \leftarrow GHASH(H, A, C) \oplus AES\_encrypt(Y_0, K)$ 
8: if  $T \neq T'$  then
9:   INVALID
10: end if
11: for  $i \leftarrow 1$  to  $n$  do
12:    $Y_i \leftarrow incr(Y_{i-1})$ 
13: end for
14: for  $i \leftarrow 1$  to  $n - 1$  do
15:    $P_i \leftarrow C_i \oplus AES\_encrypt(Y_i, K)$ 
16: end for
17:  $P_n^* \leftarrow C_n^* \oplus MSB_u(AES\_encrypt(Y_n, K))$ 

```

```

18:  $P \leftarrow P_1 || P_2 || \dots || P_n^*$ 
19: return  $C$ 

```

3.5.4 测试样例及结果截图

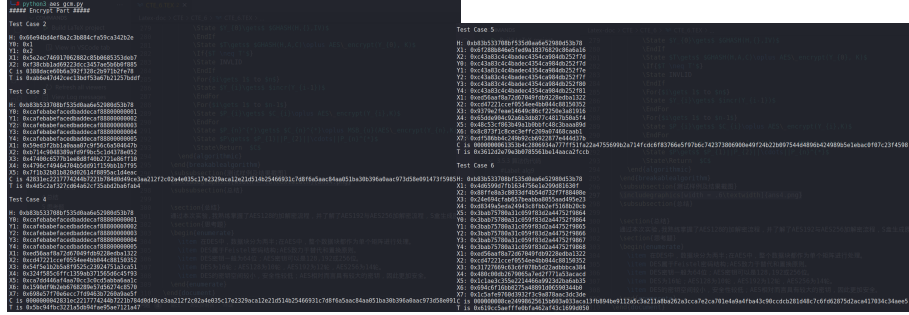


图 1: Encrypt_1

图 2: Encrypt_2

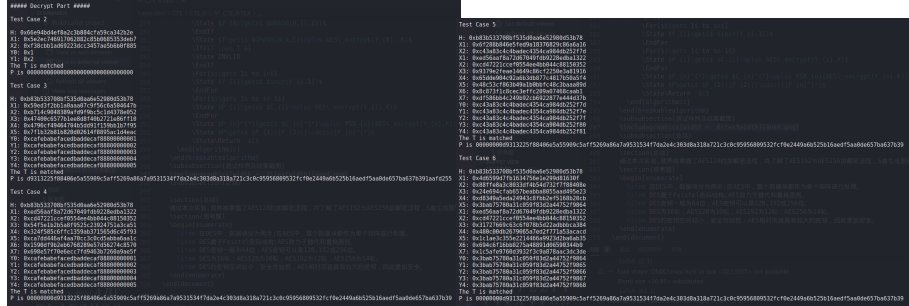


图 3: Decrypt_1

图 4: Decrypt_2

3.5.5 总结

数据相关 测试向量中 $GF2_{128}$ 数据是从左到右由低位到高位，和一般写法不同，经过长时间调试发现问题

编程实现相关 编程中的一个难点是实现 $GF2_{128}$ 域，在之前 $GF2_8$ 实现时实现有限域，只需要拓展子类为 $GF2_{128}$

GCM 实用性 GCM 是认证加密模式中的一种。GCM 中的 G 就是指 GMAC，C 就是指 CTR。GCM 可以同时确保数据的保密性、完整性及真实性，另外，它还可以提供附加消息的完整性校验。

4 总结

通过本次实验,我掌握了 SM4 国密算法的加解密流程,深刻体会到工作模式和填充方式对安全性的重要影响。体会到工作模式作为一项增强密码算法或使用算法适应具体应用的技术,在加解密和认证方面的应用。