
山东大学

SHANDONG UNIVERSITY

# 《网络空间安全创新创业实践》实验 1

2025 年 8 月 14 日


姓　　名　刘凯中

学　　号　202200150174

学　　院　网络空间安全学院

班　　级　22 密码 2 班

# 目录

# 1 实验背景

SM4（国家商用密码算法）是中国国家密码算法标准之一，主要用于信息安全领域。SM4 算法的设计和实现遵循对称加密的思想，它在处理速度和加密强度上具有较好的平衡。在实际应用中，优化 SM4 的实现效率对于提高加密系统的整体性能具有重要意义。

本实验的目标是对 SM4 算法进行软件实现，并通过优化手段提升其执行效率，最终实现基于 SM4 的 GCM 模式加密。

# 2 实验内容

本实验的主要任务包括以下几个部分：

1. SM4 算法的基本实现，包括 S 盒、FK、CK 参数的初始化和密钥扩展算法的实现。
2. 对 SM4 的实现进行性能优化，主要优化方法包括使用 T-table 加速算法和利用硬件指令集（如 AESNI、GFNI 等）提升执行效率。
3. 在此基础上实现 SM4-GCM 加密模式，并进行性能优化。

# 3 实验代码实现

## 3.1 SM4 算法的基础实现

SM4 加密算法基于分组密码技术，每次加密处理 16 字节数据。SM4 使用了 S 盒替代、置换和非线性变换等技术，并依赖于一系列常量和算法操作。首先，实现了 SM4 的密钥扩展和加密/解密块操作。代码实现如下：（具体见 github 代码部分）

```
1         # SM4基础实现
2         class SM4:
3         #S盒定义
4         SBOX = [
5         #此处省略S盒内容]
6
7         # 系统参数FKFK = [0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc]
8
9         # 固定参数CK
10        CK = [# 此处省略CK内容
11        ]def __init__(self, key):
12        if len(key) != 16:
```

```
        raise ValueError("SM4 key must be 16bytes long")
        self.rk = self._expand_key(key)

        @staticmethod
        def _rotl32<< n) & 0xFFFFFFFF) | (x >> (32 - n))def _tau(self, a):t = 0for i in ra
        t |= (self.SBOX[(a >> (i * 8 +4)) & 0xF] << (i *8 + 4))
        t|= (self.SBOX[(a >> (i * 8)) &._rotl32(b, 2) ^ self._rotl32(b, 10)^ self._rotl32(

        def _lself._rotl32(b, 13) ^ self._rotl32(b,23)(self._tau(x))

        def _expand_key(self,mk):
        rk = [0] * 32
        k = [0]* 36

        for i in range(4):
        k[i] = int.from_bytes(mk[i*4:(i+1)*4], 'big') ^ self.FK[i]for i in range(32):k[i+4
        iflen(plaintext) != 16:
        raise ValueError("SM4 block size must be 16 bytes")x = [0] *36
        for i in range(4):
        x[i] = int.from_bytes(plaintext[i*4:(i+1)*4], 'big')for i in range(32):x[i+4] = x[

ciphertext = b''
    for i in range(35, 31, -1):
        ciphertext += x[i].to_bytes(4, 'big')decrypt_block(self, ciphertext):
        if len(ciphertext) != 16:
        raise ValueError("SM4 block size must be 16 bytes")x = [0] * 36
        for i in range(4):x[i] = int.from_bytes(ciphertext[i*4:(i+1)*4], 'big')

        for i in range(32):x[i+4] = x[i] ^ self._t(x[i+1] ^ x[i+2]^ x[i+3] ^ self.rk[31-i]

        plaintext = b''
        for i in range(35, 31, -1):
        plaintext += x[i].to_bytes(4, 'bigsection{T-table优化实现}
```

为了提高 SM4 算法的执行效率，我们采用了 T-table（查表法）来优化 SM4 中的非线性变换操作。在 T-table 优化版本中，查找 S 盒值的操作通过预先计算表格，减少了实时计算的负担。

```
                # T-table优化版本
                class SM4_TTable(SM4):def __init__(self, key):
                super().__init__(key)
                self.T_table = self._init_T_table()def _init_T_table(self):
                table = [[0]* 256 for _ in range(4)]
```

```
6                 for i in range(256):
7                 b = (self.SBOX[i>> 4] << 4)| self.SBOX[i &0xF]
8                 table[0][i] = btable[1][i]= self._rotl32(b, 2)
9                 table[2][i] = self._rotl32(b, 10)
10                table[3][i] =self._rotl32(b, 18)t(self, x):
11                t0= self.T_table[0][(x >> 24) & 0xFF]
12                t1 = self.T_table[1][(x >> 16) & 0xFF]
13                t2 = self.T_table[2][(x >>8) & 0xFF]
14                t3 = self.T_table[3][x & 0xFF]1 ^ t2 ^ t3^ self._rotl32(t0 ^ t1 ^ t2^ t3,
```

## 3.2 SM4-GCM 模式实现

SM4-GCM（Galois/Counter Mode）是一种认证加密模式，它在加密的同时保证数据的完整性。本实验通过在 SM4 的基础上实现 GCM 模式，并利用前面优化过的 SM4 加密方法来提升性能。

```
1                 #GCM模 式 实 现
2                 class SM4_GCM:
3                 def __init__(self, key,nonce):
4                 if len(key) != 16:
5                 raiseValueError("SM4 keymust be 16 byteslong")
6                 if len(nonce) not in (12, 13, 14, 15, 16):
7                 raise ValueError("GCM nonce shouldbe 12–16 bytes long")
8
9                 self.cipher= SM4_TTable(key)
10                self.nonce = nonceself.H = self._init_H()
11                self.block_size = 16
12
13                def_init_H(self):
14                zero_block = bytes(_block(zero_block), 'big')def _ghash(self, data):
15                pad_len =(16 - (len(data) % 16)) %16
16                data +=bytes(pad_len)
17
18                result = 0
19                for i in range(0, len(data), 16):block = int.from_bytes(data[i:i+16], 'big
20                z = 0
21                v = self.Hfor i in range(128):
22                if (x >> (127 - i)) & 1:z ^= v
23                ifv & 1:
24                v =(v >> 1) ^ 0xE1000000000000000000000000000000
25                else:
26                v >>=inc32(self, counter):
```

```
27                counter = bytearray(counter)
28                for i in range(15, 11, -1):counter[i] += 1
29                if counter[i] != 0:
30                breakdef encrypt(self,plaintext, aad=b''):
31                if len(self.nonce) == 12:counter = self.nonce+ b'\x00\x00\x00\x01'
32                else:
33                counter = self._ghash([self.nonce])[:16]
34                counter = counter[:-4] + b'\x00\x00\x00\x01'keystream = b''
35                blocks = (len(plaintext) + 15)// 16
36                fori in range(blocks):keystream += self.cipher.encrypt_block(counter)count
37
38                ciphertext =bytes(a ^ b for a,b in zip(plaintext, keystream[:len(plaintext
39
40                len_aad = len(aad)len_ct = len(ciphertext)
41                auth_data = (aad +
42                bytes((16- (len_aad %16)) %16) +
43                ciphertext +
44                bytes((16 - (len_ct% 16)) %16) +
45                (len_aad * 8).to_bytes(8, 'big') +
46                (len_ct * 8).to_bytes(8, 'big')
47                )s = self._ghash(auth_data)
48                t =self.cipher.encrypt_block(counter)tag = bytes(a ^b for a, b inzip(t, s.
49                if len(self.nonce) == 12:counter = self.nonce+ b'\x00\x00\x00\x01'else:
50                counter = self._ghash([self.nonce])[:16]counter = counter[:-4]+ b'\x00\x00
51
52                len_aad = len(aad)len_ct = len(ciphertext)
53                auth_data = (aad +
54                bytes((16- (len_aad %16)) % 16) +
55                ciphertext +
56                bytes((16 - (len_ct% 16)) % 16) +
57                (len_aad* 8).to_bytes(8, 'big') +
58                (len_ct * 8).to_bytes(8, 'big')
59                )
60
61                s= self._ghash(auth_data)
62                t = self.cipher.encrypt_block(counter)
63                computed_tag = bytes(a ^ b for a, b in zip(t, s.to_bytes(16, 'big')))if co
64                raise ValueError("Authentication failed — invalid tag")
65
66                keystream = b''
67                blocks = (len(ciphertext) + 15) //16
68                for i in range(blocks):
69                keystream += self.cipher.encrypt_block(counter)
```

```
70                 counter = self._inc32(counter)plaintext = bytesend{pythoncode}
```

## 3.3 实验总结

通过本次实验，成功实现了 SM4 加密算法，并对其进行优化，采用 T-table 技术加速了非线性变换操作。在此基础上，进一步实现了 SM4-GCM 模式加密，提升了加密效率，并保证了数据的完整性和认证性。