# ISCC2025 WriteUp 提交模板

## MOBILE+ JpGsFlag

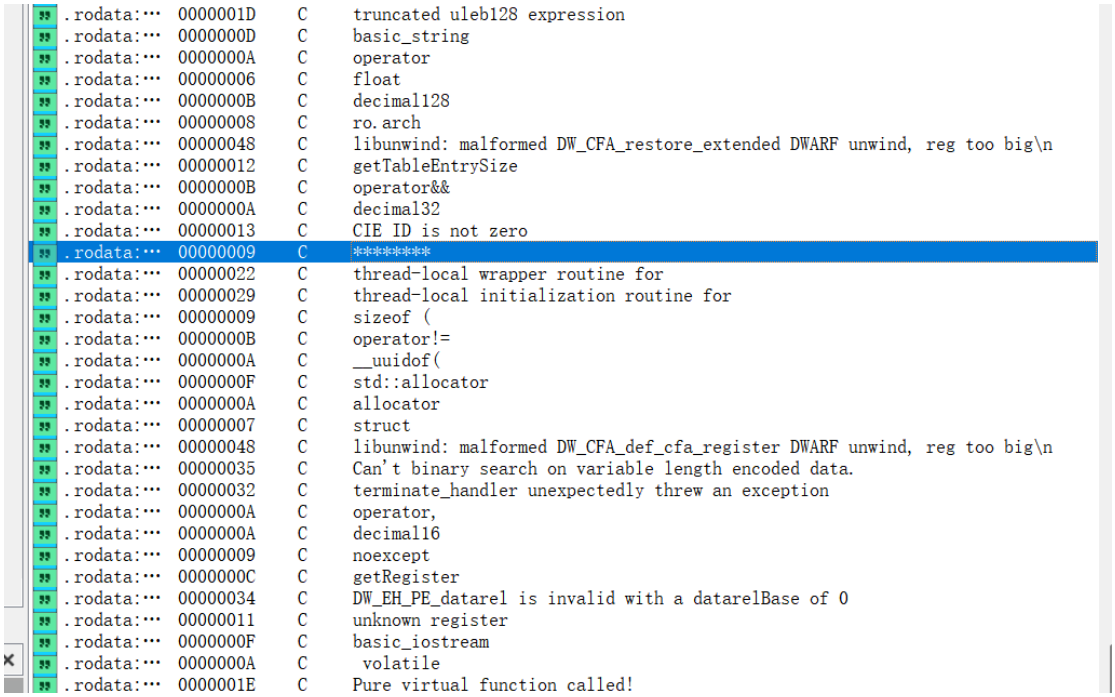
## 解题思路

1. 用 jadx-gui 打开 apk 文件

```
        @RequiresPermission("android.permission.USE_FINGERPRINT")
106     public void authenticate(@Nullable CryptoObject crypto, int flags, @Nullable CancellationSignal cancel, @NonNull Aut
            android.os.CancellationSignal cancellationSignal;
108         FingerprintManager fp = getFingerprintManagerOrNull(this.mContext);
109         if (fp != null) {
110             if (cancel != null) {
111                 cancellationSignal = (android.os.CancellationSignal) cancel.getCancellationSignalObject();
                } else {
112                 cancellationSignal = null;
                }
113             fp.authenticate(wrapCryptoObject(crypto), cancellationSignal, flags, wrapCallback(callback), handler);
        }
    }

        @Nullable
        @RequiresApi(23)
125     public static FingerprintManager getFingerprintManagerOrNull(@NonNull Context context) {
129         if (context.getPackageManager().hasSystemFeature("android.hardware.fingerprint")) {
130             return (FingerprintManager) context.getSystemService(FingerprintManager.class);
            }
132         return null;
        }

        @RequiresApi(23)
137     public static FingerprintManager.CryptoObject wrapCryptoObject(CryptoObject cryptoObject) {
            if (cryptoObject == null) {
138             return null;
            }
140         if (cryptoObject.getCipher() != null) {
141             return new FingerprintManager.CryptoObject(cryptoObject.getCipher());
            }
142         if (cryptoObject.getSignature() != null) {
143             return new FingerprintManager.CryptoObject(cryptoObject.getSignature());
            }
144         if (cryptoObject.getMac() == null) {
138             return null;
            }
145         return new FingerprintManager.CryptoObject(cryptoObject.getMac());
        }

        @RequiresApi(23)
152     public static CryptoObject unwrapCryptoObject(FingerprintManager.CryptoObject cryptoObject) {
            if (cryptoObject == null) {
153             return null;
            }
155         if (cryptoObject.getCipher() != null) {
156             return new CryptoObject(cryptoObject.getCipher());
            }
157         if (cryptoObject.getSignature() != null) {
158             return new CryptoObject(cryptoObject.getSignature());
            }
159         if (cryptoObject.getMac() == null) {
```

发现加密逻辑在 native 层

2. 用 ida 打开 so 文件，String 找到特殊字符




发现加密逻辑，完成后，代码将 v16 中的内容与全局变量 byte_49218 进行比较，所以我们的密文就是 byte_49218，sub_1E39C 是我们的加密函数

```
{
  int k; // [xsp+20h] [xbp-170h]
  int j; // [xsp+24h] [xbp-16Ch]
  __int64 v7; // [xsp+28h] [xbp-168h]
  int i; // [xsp+34h] [xbp-15Ch]
  __int64 v9; // [xsp+48h] [xbp-148h]
  unsigned int v10; // [xsp+54h] [xbp-13Ch]
  unsigned int v14; // [xsp+7Ch] [xbp-114h]
  __int64 v15; // [xsp+D8h] [xbp-B8h]
  _OWORD v16[3]; // [xsp+E0h] [xbp-B0h] BYREF
  __int16 v17; // [xsp+110h] [xbp-80h]
  _BYTE v18[8]; // [xsp+120h] [xbp-70h] BYREF
  __int64 v19; // [xsp+128h] [xbp-68h] BYREF
  _BYTE v20[84]; // [xsp+134h] [xbp-5Ch] BYREF
  __int64 v21; // [xsp+188h] [xbp-8h]

  v21 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
  v10 = sub_1EC44(a1, a4);
  v9 = sub_1EC78(a1, a4, 0LL);
  v15 = operator new[]((int)v10);
  __memcpy_chk(v15, v9, (int)v10, -1LL);
  sub_1DDA0(v15, v10, 57005LL);
  for ( i = 0; i <= 7; ++i )
    ;
  v7 = sub_1ECB4(a1, a3, 0LL);
  for ( j = 0; j <= 15; ++j )
    v20[j] = *(_BYTE *)(v7 + j);
  __memcpy_chk(v18, "0d6e1ff4", 8LL, 18LL);
  __memcpy_chk(&v19, "********", 8LL, 10LL);
  v17 = 0;
  memset(v16, 0, sizeof(v16));
  sub_1E39C(16LL, v18, v20, v16);
  for ( k = 0; k <= 15; ++k )
  {
    if ( *((unsigned __int8 *)v16 + k) != byte_49218[k] )
    {
      v14 = 0;
      goto LABEL_13;
    }
  }
  v14 = 1;
LABEL_13:
  _ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2));
  return v14;
}
```

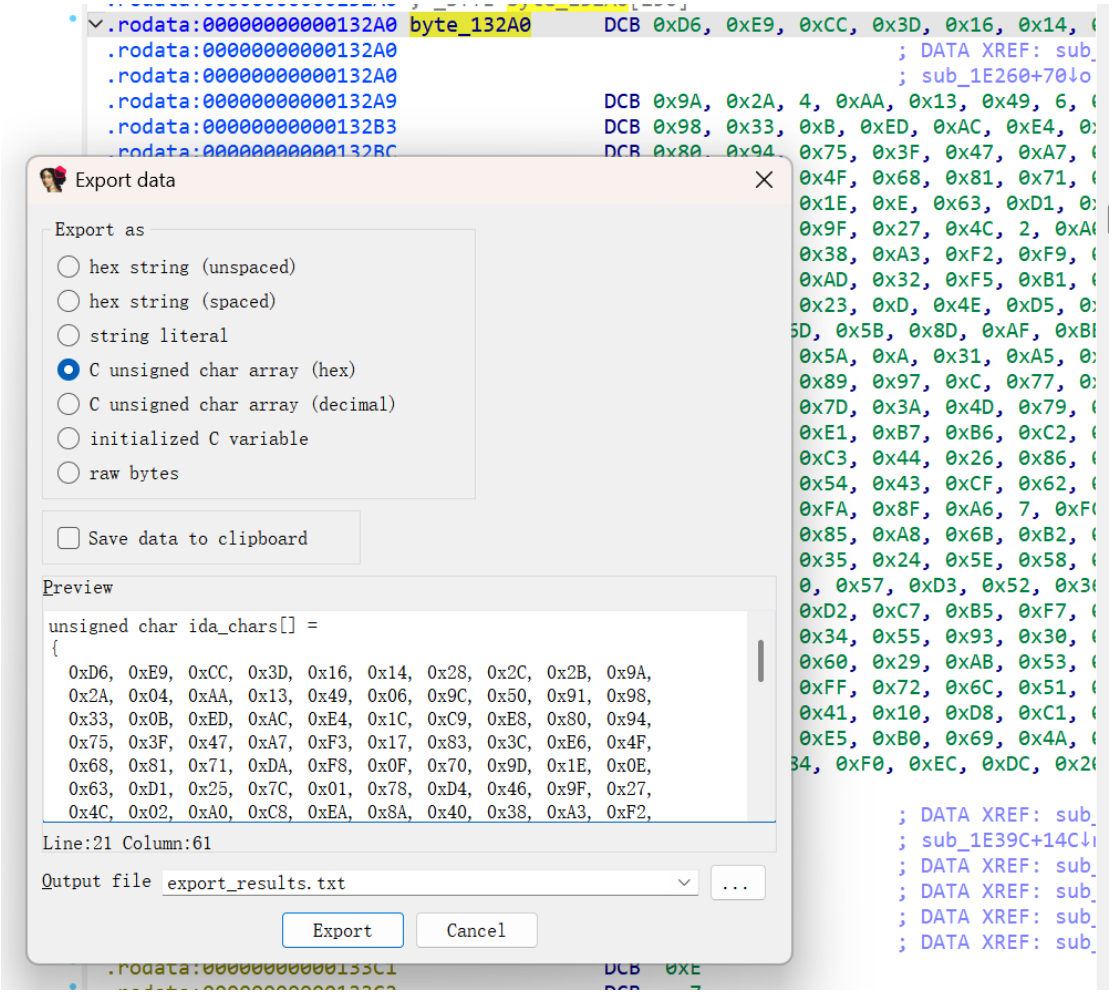
3. 分析加密函数 sub_1E39C，S 盒初始化，循环生成子密钥，轮函数与非线性变换进行 32 轮迭代，显然是 sm4 算法。

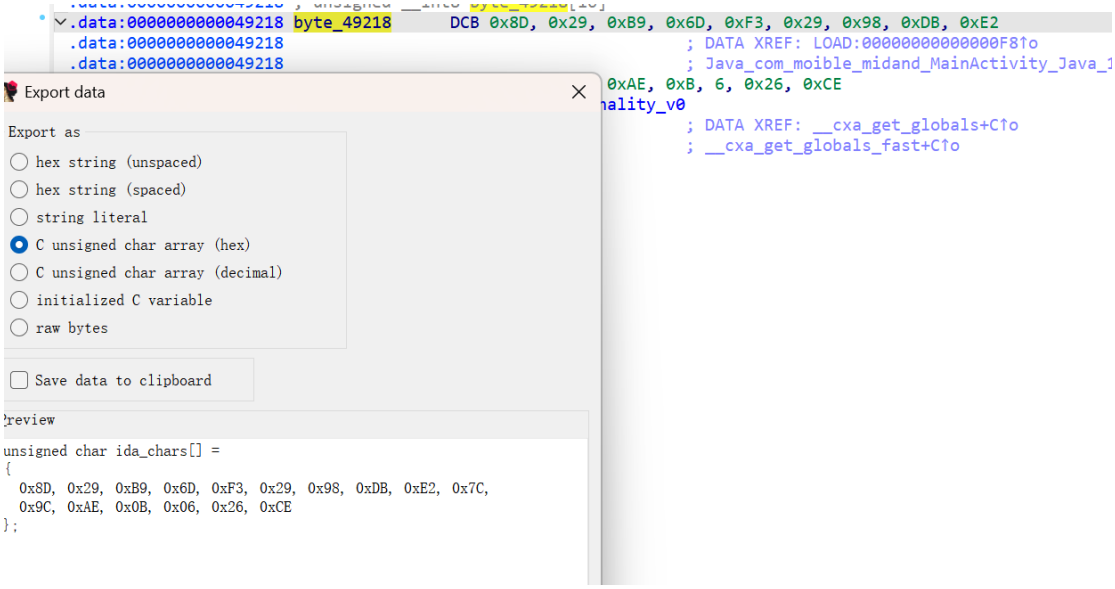
```
57  v20 = 0u;
58  v19 = 0u;
59  memset(v18, 0, sizeof(v18));
60  sub_1E030(a2);
61  sub_1E030(a2 + 4);
62  sub_1E030(a2 + 8);
63  sub_1E030(a2 + 12);
64  v21[0] = 2746333894LL;
65  v21[1] = 1453994832LL;
66  v21[2] = 1736282519LL;
67  v21[3] = 2993693404LL;
68  for ( i = 0; i <= 31; ++i )
69  {
70    v4 = i;
71    v7 = v21[v4];
72    v21[i + 4] = v7 ^ sub_1E154(v21[i + 1] ^ v21[i + 2] ^ v21[i + 3] ^ *(_QWORD *)(
73  }
74  for ( j = 0; j < a1; ++j )
75    ptr[j] = *(_BYTE *)(a3 + j);
76  for ( k = 0; k < 16 - (a1 & 0xF); ++k )
77    ptr[a1 + k] = 0;
78  for ( m = 0; ; ++m )
79  {
80    v5 = a1 >> 4;
81    if ( (a1 & 0xF) != 0 )
82      ++v5;
83    if ( m >= v5 )
84      break;
85    sub_1E030(&ptr[16 * m]);
86    sub_1E030(&ptr[16 * m + 4]);
87    sub_1E030(&ptr[16 * m + 8]);
88    sub_1E030(&ptr[16 * m + 12]);
89    for ( n = 0; n <= 31; ++n )
90    {
91      v6 = *((_QWORD *)v18 + n);
92      *((_QWORD *)&v18[2] + n) = v6 ^ sub_1E260(*((_QWORD *)v18 + n + 1) ^ *((_QWORD
93    }
94    sub_1E0AC(*((_QWORD *)&v20 + 1), a4 + 16 * m);
95    sub_1E0AC(v20, a4 + 16 * m + 4);
96    sub_1E0AC(*((_QWORD *)&v19 + 1), a4 + 16 * m + 8);
97    sub_1E0AC(v19, a4 + 16 * m + 12);
98  }
99  free(ptr);
100 _ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2));
101}
```

4. 我们先把 S 盒提取出来，sub_1E154 承担着 S 盒查找的任务，我们进入查找。byte_132A0 就是 S 盒

```
__int64 __fastcall sub_1E154(__int64 a1)
{
  __int64 v2; // [xsp+8h] [xbp-38h]
  __int64 v3; // [xsp+10h] [xbp-30h]
  int i; // [xsp+1Ch] [xbp-24h]
  int v5; // [xsp+30h] [xbp-10h] BYREF
  int v6; // [xsp+34h] [xbp-Ch] BYREF
  __int64 v7; // [xsp+38h] [xbp-8h]

  v7 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
  v6 = 0;
  v5 = 0;
  sub_1E0AC(a1, &v6);
  for ( i = 0; i <= 3; ++i )
    *((_BYTE *)&v5 + i) = byte_132A0[*((unsigned __int8 *)&v6 + i)];
  sub_1E030(&v5);
  v2 = ((__int64 (*)(void))sub_1E110)();
  v3 = v2 ^ sub_1E110(0LL, 23LL);
  _ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2));
  return v3;
}
```

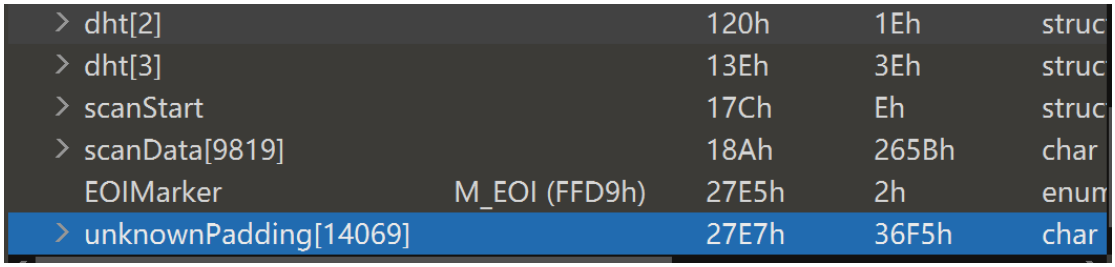
5. 我们将 S 盒和密文 byte_49218 提取出来

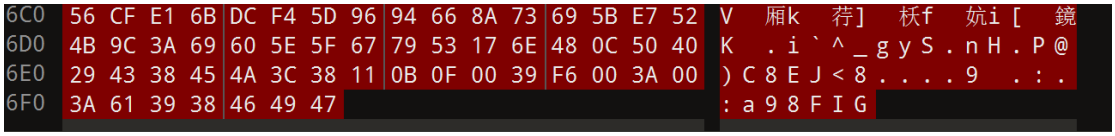


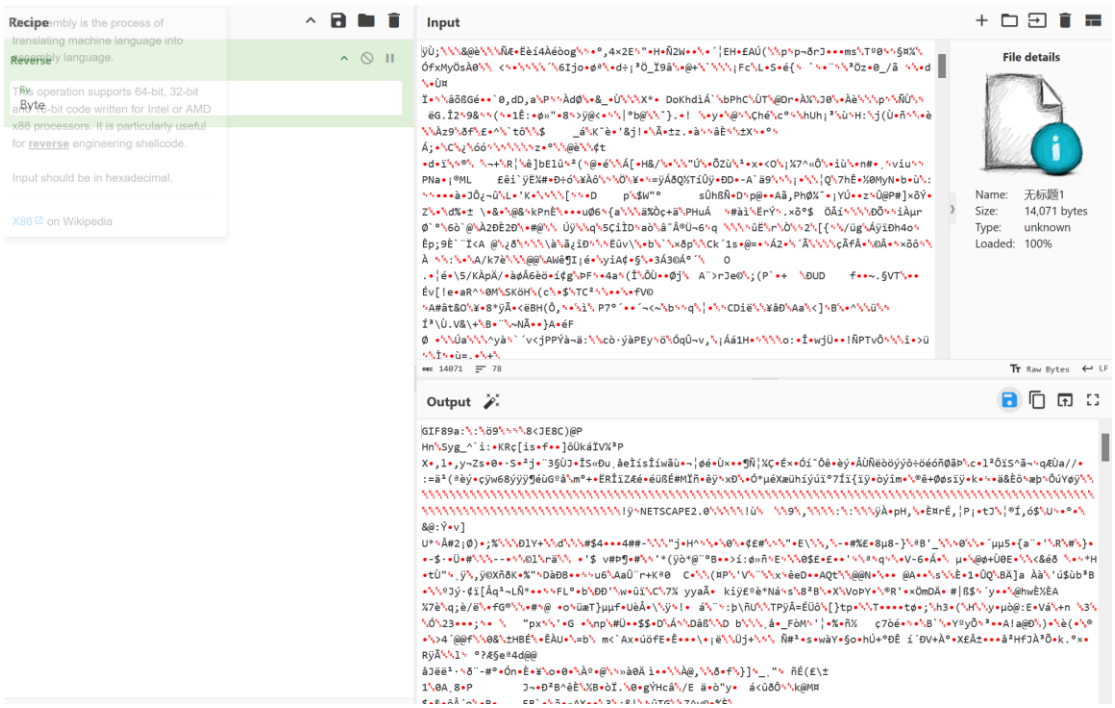



6. 在 apk 文件中还找到了一个图片，千万不要使用内置工具另保存，直接打开这个图片就行，010 查看

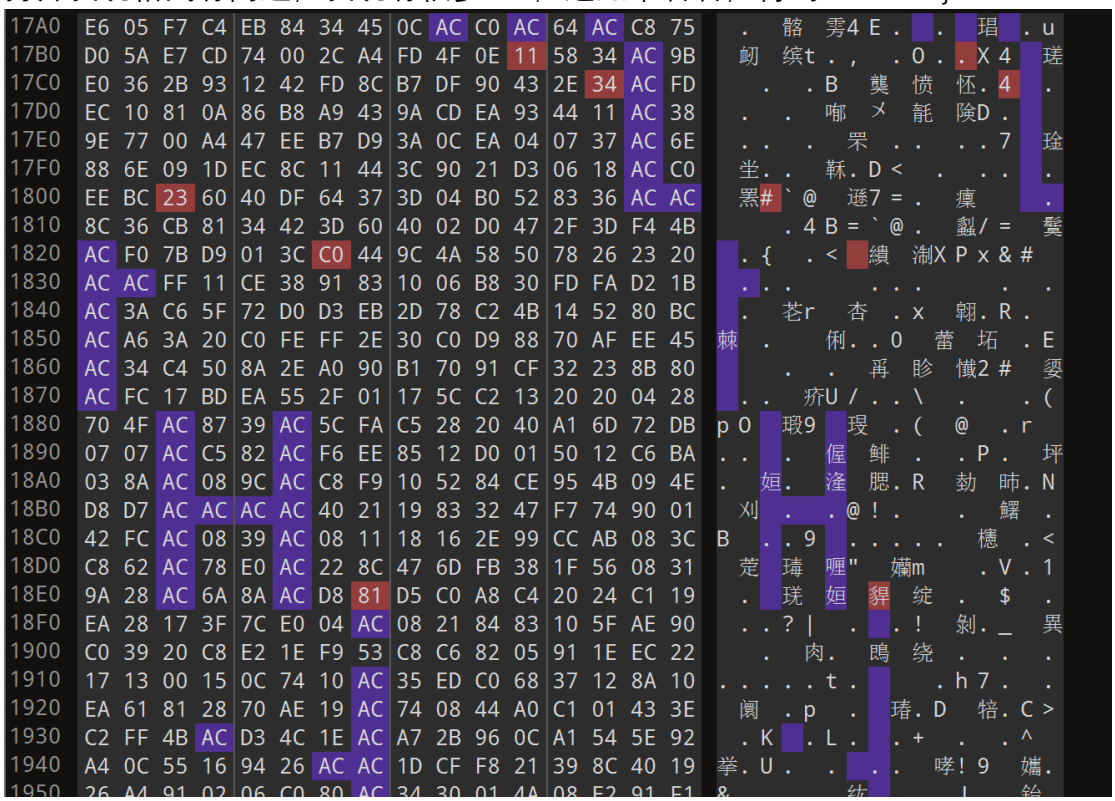

发现 unknownpadding，应该是一个 gif，倒序还原






7. 打开发现格式有问题，发现有很多 AC，连起来看看，得到 8cEmlrHj



结合之前的半个密钥我们就得到了 0d6e1ff48cEmlrHj 是加密算法的密钥

8. 写脚本进行 sm4 解密即可，得到 ISCC{a7f73e4419a1b3cc}

```
E:\reverse\python\venv\Scripts\python.exe E:\reverse\python\test\iscc\sm4.py
Decrypted plaintext: a7f73e4419a1b3cc
```

## Exp

```
import struct

# Constants
SM4_ENCRYPT = 1
SM4_DECRYPT = 0

# S-box
substitution_box = [
    [0xd6, 0xe9, 0xcc, 0x3d, 0x16, 0x14, 0x28, 0x2c, 0x2b, 0x9a, 0x2a, 0x4, 0xaa, 0x13, 0x49, 0x6],
    [0x9c, 0x50, 0x91, 0x98, 0x33, 0xb, 0xed, 0xac, 0xe4, 0x1c, 0xc9, 0xe8, 0x80, 0x94, 0x75, 0x3f],
    [0x47, 0xa7, 0xf3, 0x17, 0x83, 0x3c, 0xe6, 0x4f, 0x68, 0x81, 0x71, 0xda, 0xf8, 0xf, 0x70, 0x9d],
    [0x1e, 0xe, 0x63, 0xd1, 0x25, 0x7c, 0x1, 0x78, 0xd4, 0x46, 0x9f, 0x27, 0x4c, 0x2, 0xa0, 0xc8],
    [0xea, 0x8a, 0x40, 0x38, 0xa3, 0xf2, 0xf9, 0x15, 0xe0, 0x5d, 0x9b, 0x1a, 0xad, 0x32, 0xf5, 0xb1],
    [0x1d, 0xe2, 0x82, 0xca, 0xc0, 0x23, 0xd, 0x4e, 0xd5, 0x37, 0xde, 0x8e, 0x3, 0x6a, 0x6d, 0x5b],
    [0x8d, 0xaf, 0xbb, 0xbc, 0x11, 0x5c, 0x1f, 0x5a, 0xa, 0x31, 0xa5, 0x7b, 0x2d, 0xd0, 0xb8, 0xb4],
    [0x89, 0x97, 0xc, 0x77, 0x65, 0xf1, 0xc5, 0xc6, 0x18, 0x7d, 0x3a, 0x4d, 0x79, 0x5f, 0xd7, 0x39],
    [0x90, 0xfe, 0xe1, 0xb7, 0xb6, 0xc2, 0xfb, 0x5, 0x67, 0x76, 0xbe, 0xc3, 0x44, 0x26, 0x86, 0x99],
    [0x42, 0xf4, 0xef, 0x7a, 0x54, 0x43, 0xcf, 0x62, 0xb3, 0xa9, 0x8, 0x95, 0xdf, 0xfa, 0x8f, 0xa6],
    [0x7, 0xfc, 0x73, 0xba, 0x59, 0x19, 0x85, 0xa8, 0x6b, 0xb2, 0x64, 0x8b, 0xeb, 0x4b, 0x56, 0x35],
```

```
    [0x24, 0x5e, 0x58, 0xa2, 0x22, 0x3b, 0x21, 0x87, 0x0, 0x57, 0xd3, 0x52, 0x36, 0xe7, 0xc4, 0x9e],
    [0xbf, 0xd2, 0xc7, 0xb5, 0xf7, 0xce, 0x61, 0xa1, 0xae, 0xa4, 0x34, 0x55, 0x93, 0x30, 0x8c, 0xe3],
    [0xf6, 0x2e, 0x66, 0x60, 0x29, 0xab, 0x53, 0x6f, 0xdb, 0x45, 0xfd, 0x2f, 0xff, 0x72, 0x6c, 0x51],
    [0x1b, 0x92, 0xdd, 0x7f, 0xd9, 0x41, 0x10, 0xd8, 0xc1, 0x88, 0xcd, 0xbd, 0x74, 0x12, 0xe5, 0xb0],
    [0x69, 0x4a, 0x96, 0x7e, 0xb9, 0x9, 0x6e, 0x84, 0xf0, 0xec, 0xdc, 0x20, 0xee, 0x3e, 0xcb, 0x48]
]

system_parameters = [0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc]

fixed_parameters = [
    0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,
    0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
    0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,
    0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,
    0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,
    0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279
]


def substitute_byte(input_byte):
    row = (input_byte >> 4) & 0x0F
    col = input_byte & 0x0F
    return substitution_box[row][col]
```

```
def rotate_left(x, n, bits=32):
    return ((x << n) | (x >> (bits - n))) & 0xFFFFFFFF


def get_ulong_be(b, i):
    return (b[i] << 24) | (b[i + 1] << 16) | (b[i + 2] << 8) | b[i + 3]


def put_ulong_be(n, b, i):
    b[i] = (n >> 24) & 0xFF
    b[i + 1] = (n >> 16) & 0xFF
    b[i + 2] = (n >> 8) & 0xFF
    b[i + 3] = n & 0xFF


def linear_transformation(key_addition):
    input_bytes = bytearray(4)
    put_ulong_be(key_addition, input_bytes, 0)

    substituted_bytes = bytearray([
        substitute_byte(input_bytes[0]),
        substitute_byte(input_bytes[1]),
        substitute_byte(input_bytes[2]),
        substitute_byte(input_bytes[3])
    ])

    intermediate_value = get_ulong_be(substituted_bytes, 0)

    result = intermediate_value ^ rotate_left(intermediate_value, 2) ^ \
             rotate_left(intermediate_value, 10) ^ \
```

```
                rotate_left(intermediate_value, 18) ^ \
                rotate_left(intermediate_value, 24)

    return result & 0xFFFFFFFF


def round_function(x0, x1, x2, x3, round_key):
    return (x0 ^ linear_transformation(x1 ^ x2 ^ x3 ^ round_key)) & 0xFFFFFFFF


def calculate_round_key(key_addition):
    input_bytes = bytearray(4)
    put_ulong_be(key_addition, input_bytes, 0)

    substituted_bytes = bytearray([
        substitute_byte(input_bytes[0]),
        substitute_byte(input_bytes[1]),
        substitute_byte(input_bytes[2]),
        substitute_byte(input_bytes[3])
    ])

    intermediate_value = get_ulong_be(substituted_bytes, 0)

    round_key = intermediate_value ^ rotate_left(intermediate_value, 13) ^ \
                    rotate_left(intermediate_value, 23)

    return round_key & 0xFFFFFFFF


def generate_sub_keys(key):
    master_key = [
```

```
            get_ulong_be(key, 0),
            get_ulong_be(key, 4),
            get_ulong_be(key, 8),
            get_ulong_be(key, 12)
        ]

        key_schedule = [0] * 36
        key_schedule[0] = master_key[0] ^ system_parameters[0]
        key_schedule[1] = master_key[1] ^ system_parameters[1]
        key_schedule[2] = master_key[2] ^ system_parameters[2]
        key_schedule[3] = master_key[3] ^ system_parameters[3]

        sub_keys = [0] * 32
        for i in range(32):
            key_schedule[i + 4] = key_schedule[i] ^ calculate_round_key(
                key_schedule[i + 1] ^ key_schedule[i + 2] ^
                key_schedule[i + 3] ^ fixed_parameters[i]
            )
            sub_keys[i] = key_schedule[i + 4]

        return sub_keys


def process_block(sub_keys, input_block, output_block):
        state = [0] * 36

        state[0] = get_ulong_be(input_block, 0)
        state[1] = get_ulong_be(input_block, 4)
        state[2] = get_ulong_be(input_block, 8)
        state[3] = get_ulong_be(input_block, 12)
```

```
    for i in range(32):
        state[i + 4] = round_function(
            state[i], state[i + 1],
            state[i + 2], state[i + 3],
            sub_keys[i]
        )

    put_ulong_be(state[35], output_block, 0)
    put_ulong_be(state[34], output_block, 4)
    put_ulong_be(state[33], output_block, 8)
    put_ulong_be(state[32], output_block, 12)


def sm4_set_decryption_key(key):
    sub_keys = generate_sub_keys(key)
    # Reverse the subkeys for decryption
    for i in range(16):
        sub_keys[i], sub_keys[31 - i] = sub_keys[31 - i], sub_keys[i]
    return sub_keys


def sm4_crypt_ecb(sub_keys, input_data):
    length = len(input_data)
    output_data = bytearray(length)

    for i in range(0, length, 16):
        block = input_data[i:i + 16]
        output_block = bytearray(16)
        process_block(sub_keys, block, output_block)
        output_data[i:i + 16] = output_block
```

```
    return output_data


# Main decryption
secret_key = bytes([48, 100, 54, 101, 49, 102, 102, 52, 56, 99, 69, 109, 108, 114, 72,
106])
cipher_text = bytes([141, 41, 185, 109, 243, 41, 152, 219, 226, 124, 156, 174, 11, 6,
38, 206])

sub_keys = sm4_set_decryption_key(secret_key)
plain_text = sm4_crypt_ecb(sub_keys, cipher_text)

print("Decrypted plaintext:", plain_text.decode('latin-1'))
```