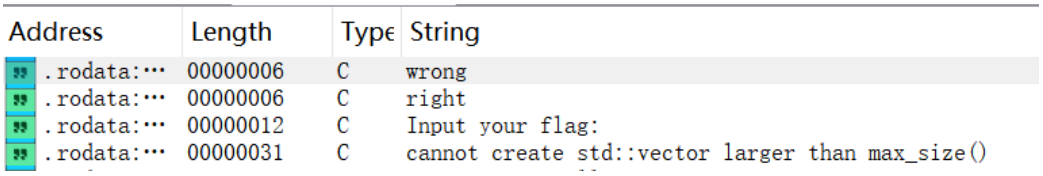
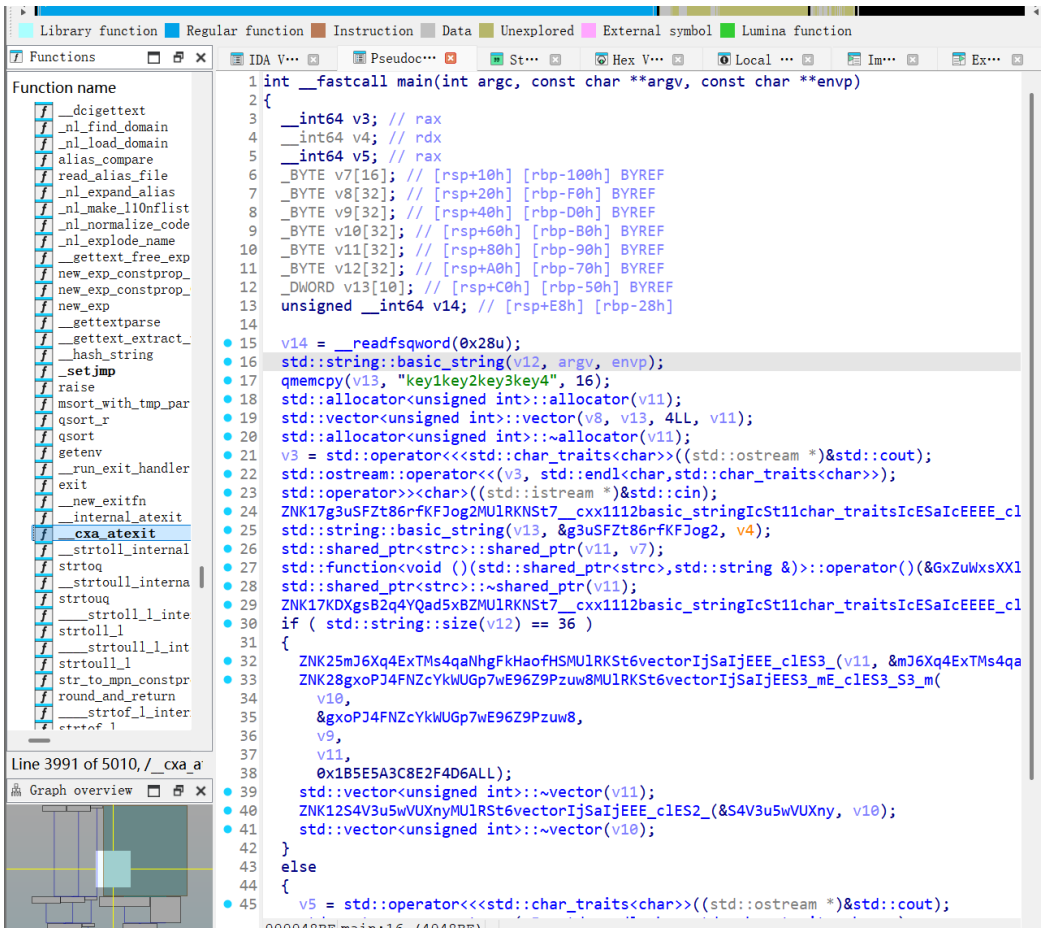
# ISCC2025 WriteUp 提交模板

## REVERSE+uglyCpp

## 解题思路

1. 用 ida 打开文件，在 strings 看到输入 flag，我们跳转对应地方（main 函数）




2. 能看到有一个密钥 key1key2key3key4，用户输入字符串存储到 v12，然后调用外部函数 GxZuWxsXXlsb 等处理，最后检查输入字符串长度是否为 36 字节。如果长度为 36 就执行相关函数。逐一查看。
3. ZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUlRKSt6vectorIjSaIjEEE_clES3_ 是密钥生成逻辑，对密钥进行扩展，通过 20 轮迭代生成密钥片段将原始密钥扩展到 44。生成初始序列，每轮通过 0x61C88647 递减生成数值，调用

ZZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUlRKSt6vectorIjSaIjEEE_clES3_ENKUlvE0_clEv 反转序列顺序。通过 ZZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUlRKSt6vectorIjSaIjEEE_clES3_ENKUlRmS5_RjS6_RS1_E1_clES5_S5_S6_S6_S7_进行双向量混合，最终生成 44 字节扩展密钥。

```
__int64 __fastcall ZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUlRKSt6vectorIjSaIjEEE_clES3_(__int64
{
  int v5; // [rsp+28h] [rbp-98h] BYREF
  int v6; // [rsp+2Ch] [rbp-94h] BYREF
  __int64 v7; // [rsp+30h] [rbp-90h] BYREF
  __int64 v8; // [rsp+38h] [rbp-88h] BYREF
  __int64 v9; // [rsp+40h] [rbp-80h] BYREF
  __int64 v10; // [rsp+48h] [rbp-78h] BYREF
  _QWORD v11[2]; // [rsp+50h] [rbp-70h] BYREF
  _QWORD v12[2]; // [rsp+60h] [rbp-60h] BYREF
  _QWORD v13[4]; // [rsp+70h] [rbp-50h] BYREF
  _BYTE v14[24]; // [rsp+90h] [rbp-30h] BYREF
  unsigned __int64 v15; // [rsp+A8h] [rbp-18h]

  v15 = __readfsqword(0x28u);
  v7 = std::vector<unsigned int>::size(a3);
  v11[1] = 20LL;
  v8 = 44LL;
  std::allocator<unsigned int>::allocator(v14);
  std::vector<unsigned int>::vector(a1, 44LL, v14);
  std::allocator<unsigned int>::~allocator(v14);
  v12[0] = a1;
  v12[1] = &v8;
  v9 = a3;
  v13[0] = &v8;
  v13[1] = &v7;
  v13[2] = a1;
  ZZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUlRKSt6vectorIjSaIjEEE_clES3_ENKUlvE_clEv(v12);
  ZZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUlRKSt6vectorIjSaIjEEE_clES3_ENKUlvE0_clEv(v14, &v9);
  v10 = 0LL;
  v11[0] = 0LL;
  v5 = 0;
  v6 = 0;
  ZZNK25mJ6Xq4ExTMs4qaNhgFkHaofHSMUlRKSt6vectorIjSaIjEEE_clES3_ENKUlRmS5_RjS6_RS1_E1_cl
    v13,
    &v10,
    v11,
    &v5,
    &v6,
    v14);
  std::vector<unsigned int>::~vector(v14);
```

4. 在数据段中发现硬编码的异或密钥，对应 9 个 32 位无符号整数，查看内存数据段提取数值：xor=[0x3ED6325B, 0xD709BF17,0xE3F27E18,0xA0870791,0x0146D6F9,0x7C6140FF, 0x10B69406, 0x94DDE0F6,0x40B2BB6C]。
5. 最后一个函数是验证函数，处理结果要和 v12 完全一致才输入有效。所以整个流程是用户输入→数据处理→与密钥异或→比对结果是否等于硬编码的 v12。

```
__int64 v9; // [rsp+20h] [rbp-80h] BYREF
__int64 v10; // [rsp+28h] [rbp-78h]
_BYTE v11[32]; // [rsp+30h] [rbp-70h] BYREF
_DWORD v12[10]; // [rsp+50h] [rbp-50h] BYREF
unsigned __int64 v13; // [rsp+78h] [rbp-28h]

v13 = __readfsqword(0x28u);
v12[0] = 1555645228;
v12[1] = -1805464030;
v12[2] = -776254107;
v12[3] = -1830657342;
v12[4] = 1912709250;
v12[5] = 1226013110;
v12[6] = 586150225;
v12[7] = -944784715;
v12[8] = 1927997992;
std::allocator<unsigned int>::allocator(&v9);
std::vector<unsigned int>::vector(v11, v12, 9LL, &v9);
std::allocator<unsigned int>::~allocator(&v9);
v2 = std::vector<unsigned int>::size(a2);
if ( v2 == std::vector<unsigned int>::size(v11) )
```

6. 写脚本运行，得到一个 32 位的字符串

```
v = [1555645228, -1805464030,
     -776254107, -1830657342, 1912709250,
     1226013110, 586150225, -944784715,
     1927997992]

v = [i & 0xffffffff for i in v]  # Convert to unsigned 32-bit integers

xor = [0x3ED6325B, 0xD709BF17,
       0xE3F27E18, 0xA0870791, 0x0146D6F9,
       0x7C6140FF, 0x10B69406, 0x94DDE0F6,
       0x40B2BB6C]

for i in range(len(v)):
    v[i] ^= xor[i]

# 确保flag被正确定义
flag = "".join([i.to_bytes(length=4, byteorder="little").decode() for i in v])
print("解密结果:", flag)
```

```
test
E:\reverse\python\venv\Scripts\python.exe E:\reverse\python\reverse\test.py
解密结果: wqob5qkC}3I2See2{vGsI9r5WeY2CZrSDQX2
```

7. 不一样是因为密钥扩展前进行ZNK17g3uSFZt86rfKFJog2MUlRKNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEE_clIES6_处理了输入，进行了置换提取处理打乱的置换表，"5p6h7q8d9risbtjuevkwaxlyfzm0c1n2g3o4"再进行映射

得到最终 flag：ISCC{WDbIeGrYrXqq32e2vs95e2ZSQ2wo5k}

```
E:\reverse\python\venv\Scripts\python.exe E:\reverse\python\reverse\test.py
解密结果: wqob5qkC}3I2See2{vGsI9r5WeY2CZrSDQX2
ISCC{WDbIeGrYrXqq32e2vs95e2ZSQ2wo5k}
```

## Exp

```
v = [1555645228, -1805464030,
     -776254107, -1830657342, 1912709250,
     1226013110, 586150225, -944784715,
     1927997992]

v = [i & 0xffffffff for i in v]

xor = [0x3ED6325B, 0xD709BF17,
       0xE3F27E18, 0xA0870791, 0x0146D6F9,
       0x7C6140FF, 0x10B69406, 0x94DDE0F6,
       0x40B2BB6C]

for i in range(len(v)):
    v[i] ^= xor[i]

#  确保 flag 被正确定义
flag = "".join([i.to_bytes(length=4, byteorder="little").decode() for i in v])
print("解密结果:", flag)
#
c_table = "5p6h7q8d9risbtjuevkwaxlyfzm0c1n2g3o4"
table = "abcdefghijklmnopqrstuvwxyz0123456789"

res = ""
for i in range(len(table)):
    tmp = c_table.index(table[i])
    res +=flag[tmp]
```

```
print(res)
```