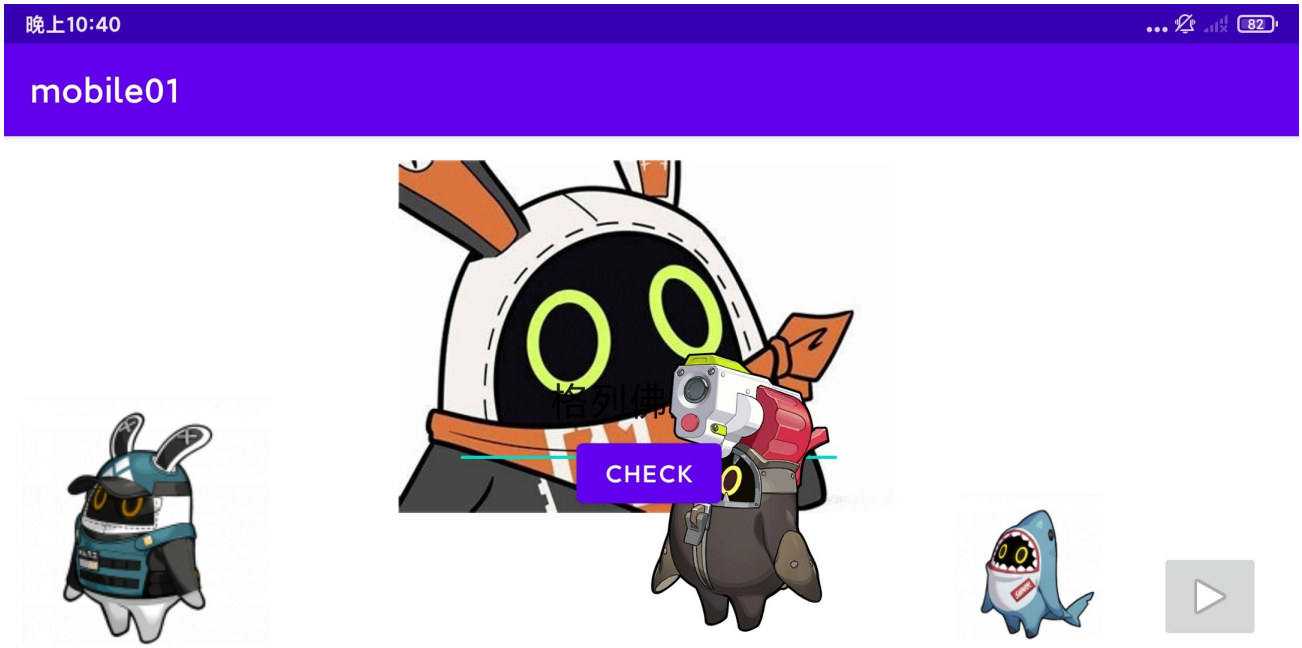
来源：https://xz.aliyun.com/news/18015
文章ID：18015

ISCC 练武初赛re+mobile wp-先知社区

mobile

ISCC mobile 邦布出击

安装apk



点击右下角的按钮，进入图鉴界面，百度各种邦布的种类，一个一个试，可以得到三段base64加密的文本
邦布图鉴 - 绝区零WIKI_BWIKI_哔哩

BBTUJIAN

名字： 鲨牙布

级别： S

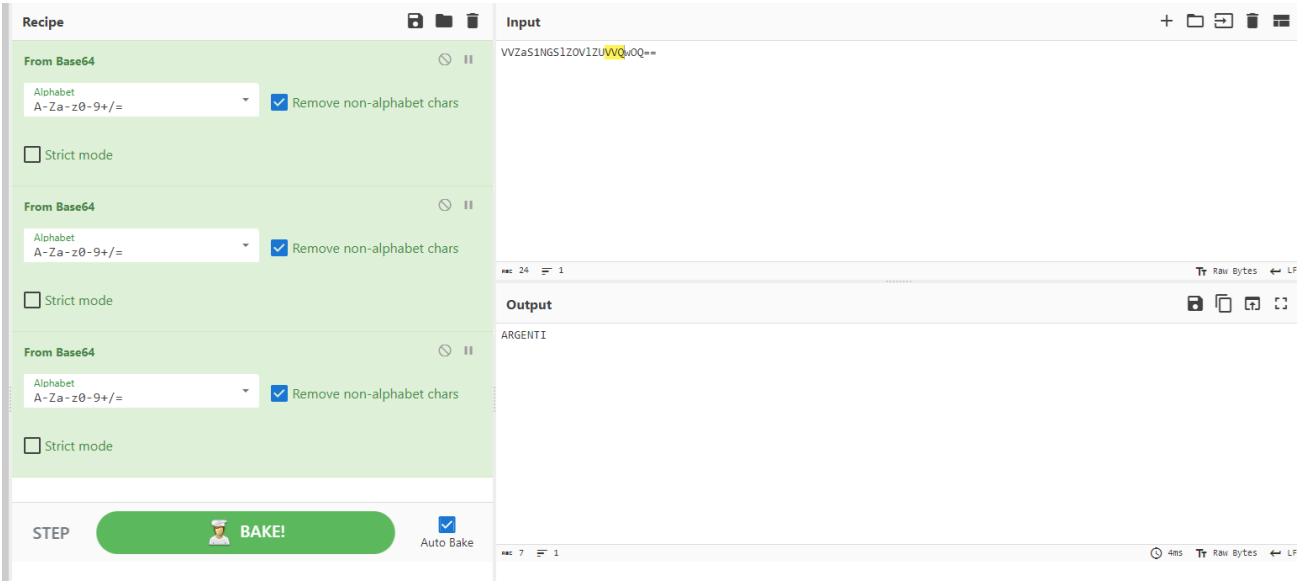
SUBMIT

VVZaS1NGS

哔哩

```
VVQwOQ==
lZOVlZU
VVZaS1NGS
```

然后将三段base64拼接起来，循环解码三次base64



得到一串明文
尝试打开解压得到的db文件，提示非数据库文件，经查询是经过sqlcipher加密，那么此前得到的明文应该就是解密的key

```
>sqlcipher enflag.db
SQLCipher version 3.8.0.2 2013-09-03 17:11:13
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> PRAGMA key = 'ARGENTI';
sqlite> ATTACH DATABASE 'plaintext.db' AS plaintext KEY '';
sqlite> SELECT sqlcipher_export('plaintext');

sqlite> DETACH DATABASE plaintext;
sqlite> .q
```

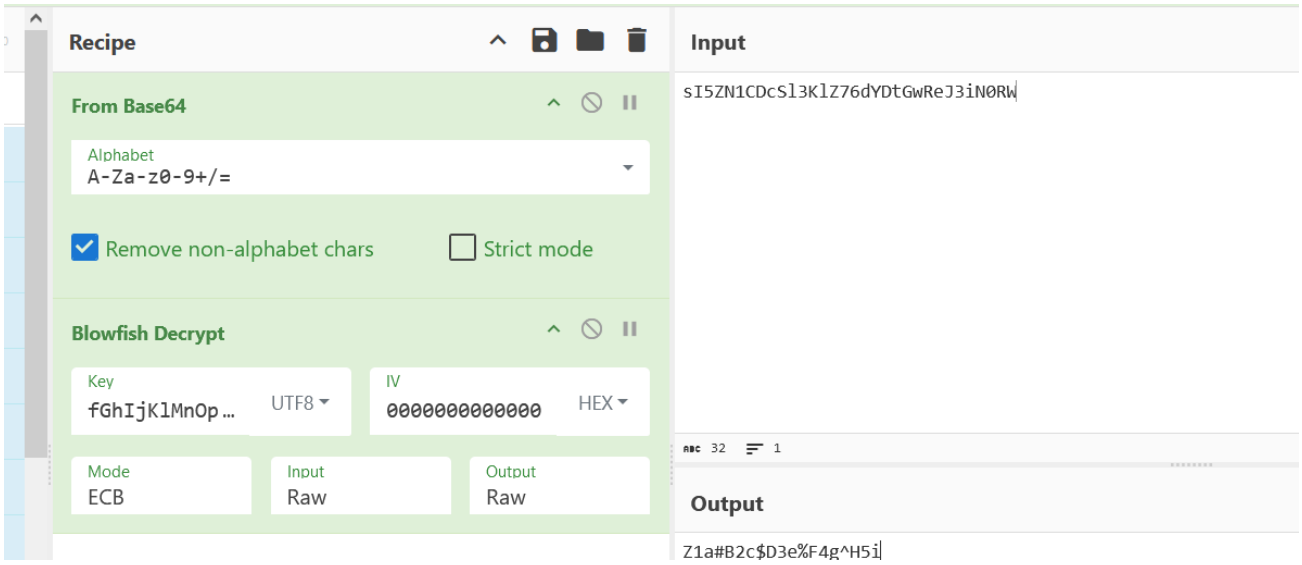
	id	name	value	info
1	1	flag!	102;108;97;103;123;121;111;11...	Congratulationo.0◆◆&#◆◆
2	2	key?	CdEfGhIjKlMnOpQr	!blowfish!
3	3	():flag?KEY	\u0074\u0068\u0065\u0020\u...	something crucial

flag是假的，实际应该留意的是key以及info中的blowfish（一种加密方式）
使用jadx打开apk

```
public boolean Jformat(String str) {
    if (str.length() < 7 || !str.substring(0, 5).equals("ISCC{") || str.charAt(str.length() - 1) != '}') {
        return false;
    }
    try {
        String a = a.a();
        Log.d("str1", "des加密明文: " + a);
        try {
            String encrypt = new DESHelper().encrypt(a, "WhItenet", getiv());
            Log.d("DEBUG_RES", "加密结果 res: " + encrypt);
            return str.substring(5, str.length() - 1).equals(encrypt);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    } catch (Exception e2) {
        throw new RuntimeException(e2);
    }
}
```

```
public class b {
    private static String hiddenString = "sI5ZN1CDcSl3KlZ76dYDtGwReJ3iN0RW";

    public static String b() {
        try {
```

将上图中的密文通过blowfish解密之后得到的内容就是DES的明文



根据apk的逻辑，只有当该明文DES加密的结果和输入内容去掉flag格式后的内容相同才正确
已知明文、key、加密方式，那么对于DES加密，还需要具备的就是iv，但是iv是通过native函数生成的

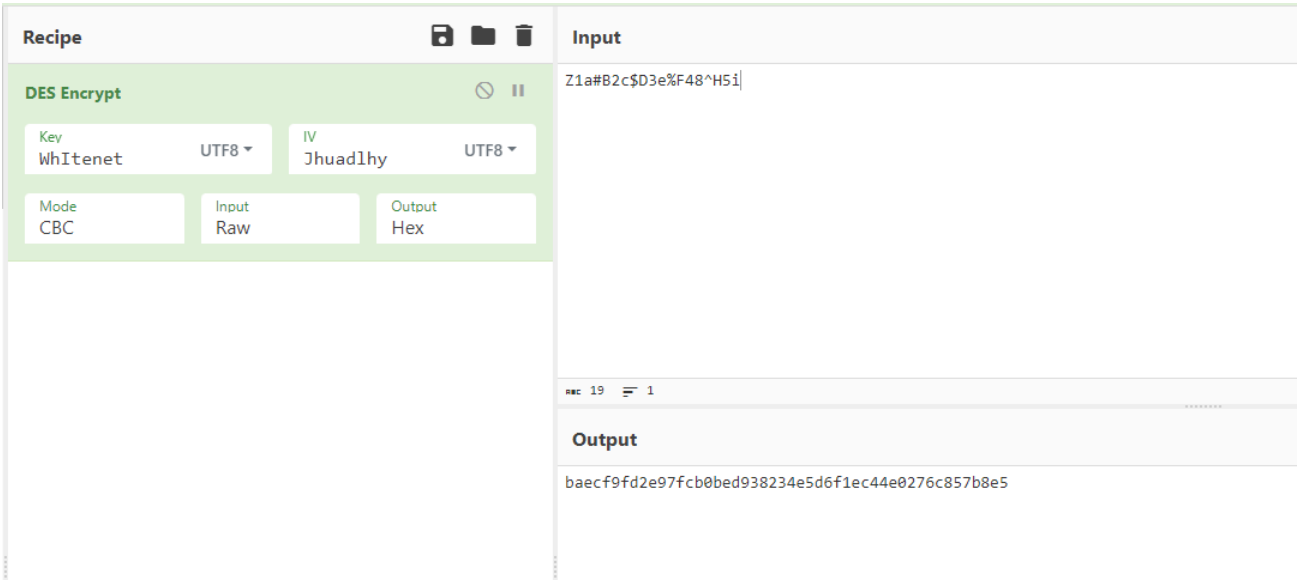
```
public native String getiv();
```

方法一：分析so文件iv的生成逻辑 -- 生成逻辑比较复杂，放弃
方法二：hook native function，在调用getiv时输出iv
这里使用frida hook（要在手机上先运行frida-server）

```
Java.perform(function () {
    try {
        var cls = Java.use("com.example.mobile01.MainActivity");
        cls.getiv.implementation = function () {
            var iv_val = this.getiv();
            console.log("[*] MainActivity.getiv() called, returned: " + iv_val);
            return iv_val;
        };
        console.log("[+] Hooked com.example.mobile01.MainActivity.getiv()");
    } catch (err) {
        console.error("[-] Failed to hook MainActivity.getiv: " + err);
    }
});
```

```
frida -U -f 进程名 -l hook.js
```

```
[*] MainActivity.getiv() called, returned: Jhuadlhykvdutfpssbzpvu
Jhuadlhykvdutfpssbzpvu
```



ISCC mobile detective

附件是一个apk文件，用jadx打开

```
ride // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity
cted void onCreate(Bundle bundle) {
uper.onCreate(bundle);
ctivityMainBinding inflate = ActivityMainBinding.inflate(getLayoutInflater());
his.binding = inflate;
etContentView(inflate.getRoot());
his.flagEditText = this.binding.editTextFlag;
utton button = this.binding.button;
his.submitButton = button;
utton.setOnClickListener(new View.OnClickListener() { // from class: com.example.detective.MainActivity.1
   @Override // android.view.View.OnClickListener
   public void onClick(View view) {
       if (MainActivity.this.Jformat(MainActivity.this.flagEditText.getText().toString())) {
           Toast.makeText(MainActivity.this, "Congratulations, you are right!", 1).show();
       } else {
           Toast.makeText(MainActivity.this, "PITY", 0).show();
       }
   }
);


DX INFO: Access modifiers changed from: private */
c boolean Jformat(String str) {
eturn str.length() >= 8 && (str.length() + 1) % 2 != 0 && str.substring(0, 5).equals("ISCC{") && str.charAt(str.length() - 1) == '}' && stringFromJNI(
```

可以看到关键是这个stringFromJNI函数，跟进之后发现是native函数，因此用IDA打开so文件

```
  intext2 = (char *)v22 + 1;
  LOBYTE(v22[0]) = 2 * len;
  if ( len )
LABEL_5:
    memmove(intext2, intext, len2);
  intext2[len2] = 0;
  (*a1)->ReleaseStringUTFChars(a1, (jstring)input, intext);
  v20 = 16;
  strcpy(v21, "Sherlock");
  xorEncrypt((__int64)v22, &v20, (__int64)&v17);
  v15[0] = 0LL;
  v15[1] = 0LL;
  if ( (v17 & 1) != 0 )
    v10 = v19;
  else
    v10 = v18;
  v16 = 0LL;
```

关键是这个xorEncrypt函数

```
}
v9 = *a2;
v10 = v3 >> 1;
if ( (v9 & 1) != 0 )
  v11 = *((_QWORD *)a2 + 1);
else
  v11 = v9 >> 1;
if ( !(_DWORD)v8 )
  v10 = *((_QWORD *)v5 + 1);
if ( v10 )
{
  v12 = 0LL;
  do
  {
    v13 = (unsigned __int8 *)*((_QWORD *)v5 + 2);
    v14 = (v8 & 1) == 0;
    v15 = (unsigned __int8 *)*((_QWORD *)a2 + 2);
    if ( !v14 )
      v13 = v5 + 1;
    if ( (*a2 & 1) == 0 )
      v15 = a2 + 1;
    v16 = v15[v12 % v11];
    v17 = v13[v12];
    if ( (*(_BYTE *)a3 & 1) != 0 )
      v18 = *(_QWORD *)(a3 + 16);
    else
      v18 = a3 + 1;
    *(_BYTE *)(v18 + v12++) = v16 ^ v17;
    v8 = *v5;
    v19 = *((_QWORD *)v5 + 1);
    v20 = v8 >> 1;
    LOBYTE(v8) = (v8 & 1) == 0;
    if ( (_BYTE)v8 )
      v19 = v20;
  }
  while ( v12 < v19 );
}
return result;
}
```

```
__int64 __usercall xorEncrypt@<X0>(__int64 result@<X0>, unsigned __int8 *a2@<X1>, __int64 a3@
{
  unsigned __int8 v3; // w9
  unsigned __int8 *v5; // x20
  __int64 v7; // x8
  unsigned __int64 v8; // x11
  unsigned __int64 v9; // x8
  __int64 v10; // x9
  unsigned __int64 v11; // x8
  unsigned __int64 v12; // x9
  unsigned __int8 *v13; // x15
  bool v14; // zf
  unsigned __int8 *v15; // x11
  unsigned __int8 v16; // w11
  unsigned __int8 v17; // w14
  __int64 v18; // x15
  unsigned __int64 v19; // x14
  unsigned __int64 v20; // x15

  v3 = *(_BYTE *)result;
  v5 = (unsigned __int8 *)result;
  if ( (*(_BYTE *)result & 1) != 0 )
  {
    result = sub_62B70(a3, *(_QWORD *)(result + 16), *(_QWORD *)(result + 8));
    v3 = *v5;
    LODWORD(v8) = (*v5 & 1) == 0;
  }
  else
  {
    v7 = *(_QWORD *)(result + 16);
    LODWORD(v8) = 1;
    *(_OWORD *)a3 = *(_OWORD *)result;
    *(_QWORD *)(a3 + 16) = v7;
  }
  v9 = *a2;
  v10 = v3 >> 1;
  if ( (v9 & 1) != 0 )
    v11 = *((_QWORD *)a2 + 1);
  else
00062754 _Z10xorEncryptRKNSt6__ndk112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEES7_:60 (62754)
```

通过分析代码可知，该函数先将字符串转换为十六进制，再将输入与key异或之后转为字符串，然后从每4个字符中提取前2个字符，然后再根据一定规律打乱字符串的位置信息，最后替换特定位置的字符

```
import re
from functools import reduce
import binascii

class CryptoSolver:
    @staticmethod
    def extract_alternate_chars(encoded_text):
        """提取每4个字符中的前2个字符"""
```

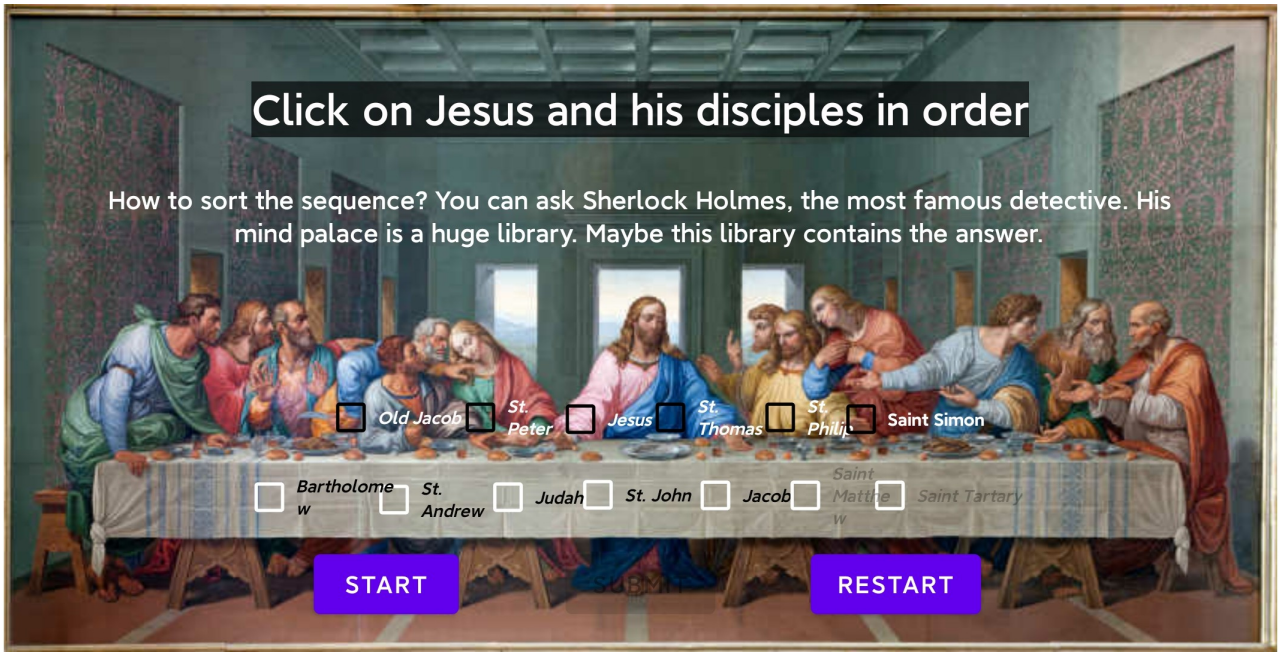
```
        if len(encoded_text) % 4 == 2:
            encoded_text += '00'
        return ''.join([encoded_text[i:i+2] for i in range(0, len(encoded_text), 4)])

    @staticmethod
    def hex_encode_chars(text):
        """将字符串转为十六进制表示"""
        hex_representation = ''.join([f'{ord(c):04x}' for c in text])
        return hex_representation[2:] if hex_representation.startswith('00') else hex_representation

    @staticmethod
    def process_pattern_swaps(text):
        """处理特定模式的字符交换"""
        result = []
        pattern = re.compile(r'(..)(..)')
        i = 0

        while i < len(text):
            if i + 3 < len(text) and text[i+2:i+4] == '21':
                result.append(text[i+1])
                result.append(text[i])
                i += 4
            else:
                result.append(text[i:i+2])
                i += 2

        return ''.join(result)

    @staticmethod
    def interleave_with_substitution(text):
        """分割、替换和交错合并处理"""
        mid_point = len(text) // 2
        first_half = list(text[:mid_point])
        second_half = list(text[mid_point:])

        # 替换特定位置的'3'为'0'
        for i in range(len(second_half)):
            if second_half[i] == '3' and (i == 0 or i % 3 == 0):
                second_half[i] = '0'

        for i in range(len(first_half)):
            if first_half[i] == '3' and (i == 1 or (i-1) % 3 == 0):
                first_half[i] = '0'

        # 交错合并
        merged = []
        for i in range(len(text)):
            if i % 2 == 0 and i//2 < len(second_half):
                merged.append(second_half[i//2])
            elif i % 2 == 1 and i//2 < len(first_half):
                merged.append(first_half[i//2])

        return ''.join(merged)

    @staticmethod
    def decode_to_chars(encoded):
        """将十六进制编码转换为字符"""
        chars = []
        index = 0

        while index < len(encoded):
            if encoded[index] == '0' and index + 2 < len(encoded):
                # 处理三位编码
```

```
                hex_val = encoded[index+1:index+3]
                chars.append(chr(int(hex_val, 16)))
                index += 3
            else:
                # 处理四位编码
                end = min(index+4, len(encoded))
                hex_val = encoded[index:end]
                if end - index == 4:  # 确保有足够的字符
                    chars.append(chr(int(hex_val, 16)))
                index += 4

        return ''.join(chars)


class XorDecoder:
    def __init__(self, key="Sherlock"):
        self.key = key.encode('utf-8')
        self.solver = CryptoSolver()

    def xor_bytes(self, hex_string):
        """XOR解密十六进制字符串"""
        # 将十六进制字符串转换为字节列表
        bytes_data = [int(hex_string[i:i+2], 16) for i in range(0, len(hex_string), 2)]

        # 应用XOR操作
        xor_result = []
        for i, byte in enumerate(bytes_data):
            key_byte = self.key[i % len(self.key)]
            xor_result.append(byte ^ key_byte)

        # 转换为UTF-8字符串
        try:
            return bytes(xor_result).decode('utf-8')
        except UnicodeDecodeError:
            # 处理解码错误
            return ''.join([chr(b) for b in xor_result])

    def process_layers(self, input_text):
        """应用多层处理"""
        layer1 = self.solver.hex_encode_chars(input_text)
        layer2 = self.solver.extract_alternate_chars(layer1)
        layer3 = self.solver.process_pattern_swaps(layer2)
        layer4 = self.solver.interleave_with_substitution(layer3)
        return self.solver.decode_to_chars(layer4)

    def decrypt(self, encrypted_hex):
        """完整解密流程"""
        # 先XOR解密
        intermediate = self.xor_bytes(encrypted_hex)
        # 然后应用多层解码
        return self.process_layers(intermediate)


def main():
    # 示例加密数据
    encrypted = "xxxxxxxxxxxxxxxxxxx"

    # 创建解码器并解密
    decoder = XorDecoder()
    result = decoder.decrypt(encrypted)
    print(f"解密结果: {result}")

    # 直接使用XOR解密检查中间结果
```

```
    xor_result = decoder.xor_bytes(encrypted)
    print(f"XOR中间结果: {xor_result}")


if __name__ == "__main__":
    main()
```

HolyGrail

附件为apk安装包



使用jadx打开apk，发现其中有许多checkbox，点击checkbox的响应如下

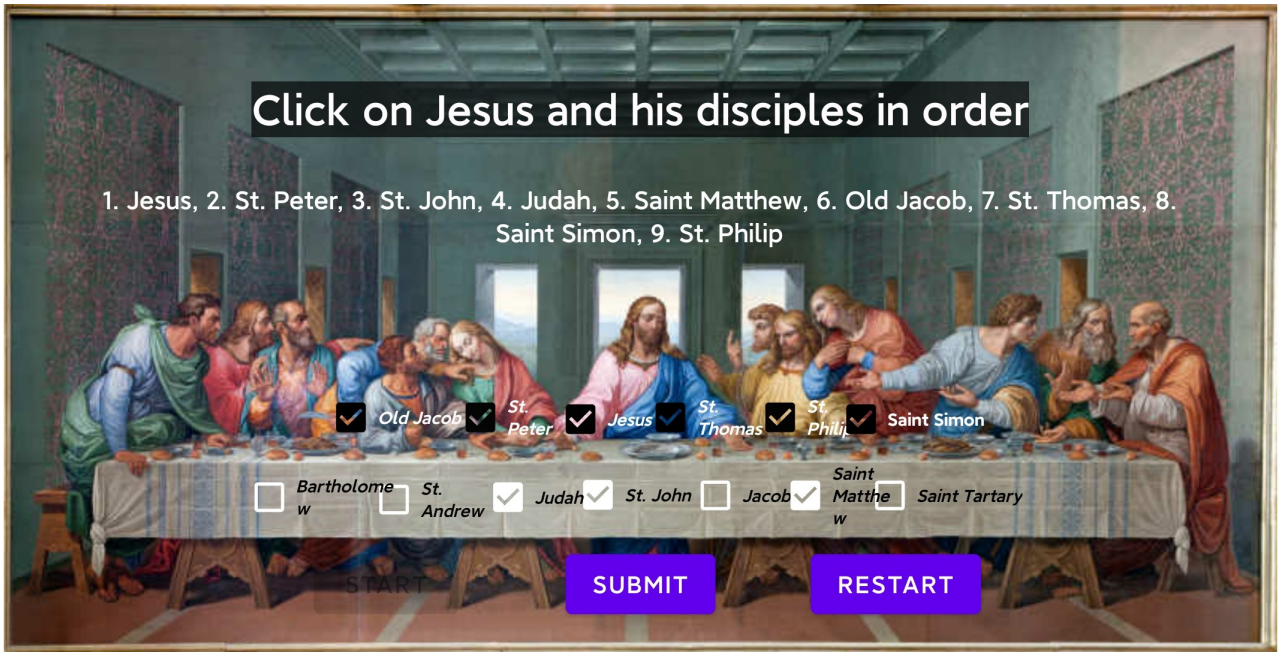
```
    /* JADX INFO: Access modifiers changed from: private */
    /* renamed from: onCheckBoxClicked, reason: merged with bridge method [inline-methods] */
119 public void m51lambda$enableCheckBoxes$3$comexampleholygrailMainActivity(CheckBox checkBox) {
120     String resourceEntryName = getResources().getResourceEntryName(checkBox.getId());
121     if (checkBox.isChecked()) {
123         this.userSequence.add(resourceEntryName);
        } else {
126         this.userSequence.remove(resourceEntryName);
        }
129     updateSelectedOrderTextView();
    }
```

每点击一个checkbox就会在userSequence末尾添加当前checkbox的资源名称

```
private void submitSequence() {
    CipherDataHandler.saveCipherText(this, CipherDataHandler.getCipherText(this.userSequence));
    GameData.userSequence.clear();
    GameData.userSequence.addAll(this.userSequence);
    goToNextPage();
}
```

而根据app的提示，需要按照特定顺序点击checkbox，才能进入验证flag的页面，并且返回在native层加密后的密文

关于顺序，可以自行百度，也可以问ai，最终顺序如下



如何获得密文：通过frida hook，手动传入特定顺序的参数（每个checkbox的参数也需要通过frida hook得到），然后输出返回的密文

```
var cipher = Java.use("com.example.holygrail.CipherDataHandler");
    var args = Java.array("java.lang.String", ["checkBox8","checkBox6","checkBox7","checkBox5","checkBox12","chec
console.log(cipher.generateCipherText(args));
```

然后分析验证flag的页面

```
    private void submitSequence() {
        String trim = this.flagInput.getText().toString().trim();
        if (!isCorrectFormat(trim)) {
            Toast.makeText(this, "Wrong flag format", 0).show();
            return;
        }
        String substring = trim.substring(5, trim.length() - 1);
        String string = this.sharedPreferences.getString("cipherText", "");
        String validateFlag = a.validateFlag(this, substring);
        if (validateFlag != null && validateFlag.equals(string)) {
            if ("af5c66387436b0c8cfa537be5751c4629c9e288966315c41ec07bf91658a32f4".equalsIgnoreCase(sha256(substring))) {
                Toast.makeText(this, "Success", 0).show();
                return;
            } else {
                Toast.makeText(this, "Correctly matched but in the wrong order.", 0).show();
                return;
            }
        }
        Toast.makeText(this, "Wrong flag", 0).show();
    }

    private boolean isCorrectFormat(String str) {
        return str.startsWith("ISCC{") && str.endsWith("}");
    }

    private String sha256(String str) {
        try {
            byte[] digest = MessageDigest.getInstance("SHA-256").digest(str.getBytes("UTF-8"));
            StringBuilder sb = new StringBuilder();
            for (byte b : digest) {
                String hexString = Integer.toHexString(b & UByte.MAX_VALUE);
                if (hexString.length() == 1) {
                    sb.append('0');
                }
                sb.append(hexString);
            }
            return sb.toString();
        } catch (Exception unused) {
            return "";
        }
    }
```

首先检查flag格式，然后调用a类的validateFlag方法

```
public class a {
    public static native String processWithNative(String str);

    static {
        System.loadLibrary("holygrail");
    }

    public static String validateFlag(Context context, String str) {
        return b.a(processWithNative(b(context, str)));
    }

    private static String b(Context context, String str) {
        return vigenereEncrypt(str, getEncryptionKey(context));
    }

    private static String getEncryptionKey(Context context) {
        String str = "";
        try {
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(context.getResources().openRawResource(R.raw.key)));
            str = bufferedReader.readLine();
            bufferedReader.close();
            return str;
        } catch (Exception e) {
            e.printStackTrace();
            return str;
        }
    }

    private static String vigenereEncrypt(String str, String str2) {
        StringBuilder sb = new StringBuilder();
        int length = str2.length();
        int i = 0;
        for (int i2 = 0; i2 < str.length(); i2++) {
            char charAt = str.charAt(i2);
            if (Character.isLetter(charAt)) {
                char c = Character.isLowerCase(charAt) ? 'a' : 'A';
                char charAt2 = str2.charAt(i % length);
                charAt = (char) ((((charAt - c) + (charAt2 - (Character.isUpperCase(charAt2) ? 'A' : 'a'))) % 26) + c);
                i++;
            }
            sb.append(charAt);
        }
        return sb.toString();
    }
}
```

大概流程

- getEncryptionKey
- vigenereEncrypt
- processWithNative
- b.a

```java
public class b {
    public static String a(String str) {
        StringBuilder sb = new StringBuilder();
        if (str.length() % 2 != 0) {
            Log.e("b", "Invalid input: Length must be even");
            return "Invalid input: Length must be even";
        }
        int i = 0;
        while (i < str.length()) {
            int i2 = i + 2;
            sb.append((char) Integer.parseInt(str.substring(i, i2), 16));
            i = i2;
        }
        return sb.toString();
    }
}
```

由于processWithNative是JNI函数，因此尝试frida hook该函数，尝试传入不同的值，发现每个字符对应的加密结果和顺序无关，因此可以直接生成所有字符加密的结果，再对目标字符串进行匹配

解密思路

- 转十六进制
- 字符替换
- 字符偏移

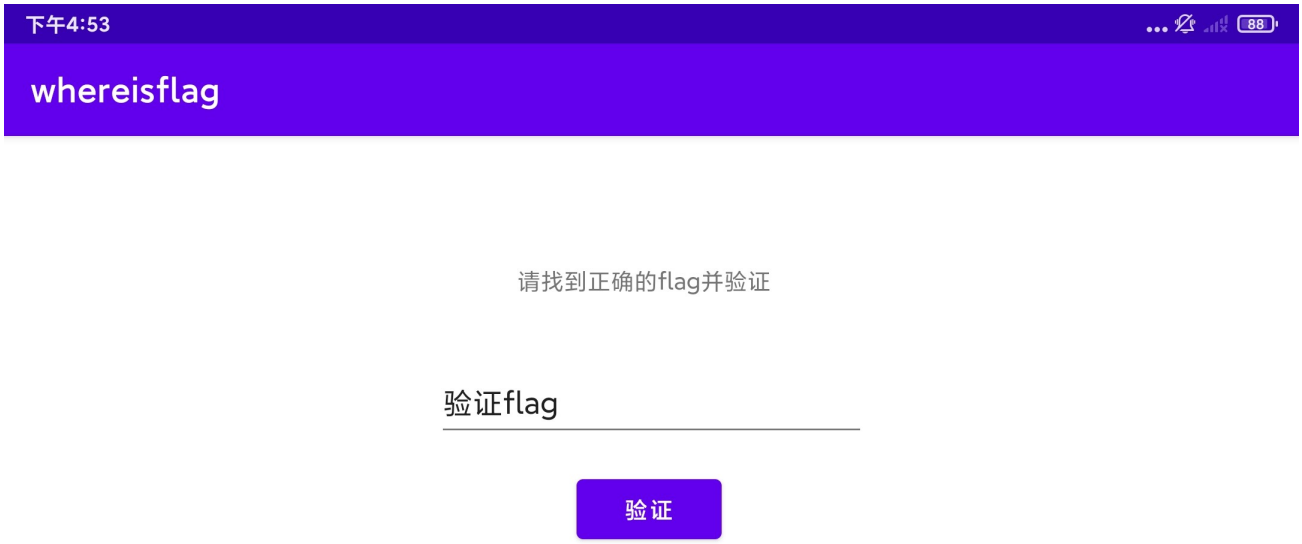
exp

```
from collections import defaultdict
import hashlib
import binascii
from rich.progress import track

CHARACTER_SET = r"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!#$%&'()*+,-./:;<=>?@[\]^_`{|}~"

def build_mapping_table():
    raw_data = "39213A213B213C2140214121422143214421452146474849 4A4B4C505152535455565758595A5B5C60616263646550215
    mapping = []

    idx = 0
    while idx < len(raw_data):
        if idx + 3 < len(raw_data) and raw_data[idx+2:idx+4] == "21":
            mapping.append(raw_data[idx:idx+4].lower())
            idx += 4
        else:
            mapping.append(raw_data[idx:idx+2].lower())
            idx += 2

    return mapping
```

```
class VigenereCipher:
    def __init__(self, key):
        self.key = key.lower()
        self.key_length = len(key)

    def decrypt(self, text):
        result = []
        key_position = 0

        for character in text:
            if not character.isalpha():
                result.append(character)
                continue

            base = ord('a') if character.islower() else ord('A')

            key_char = self.key[key_position % self.key_length]
            key_shift = ord(key_char) - ord('a')

            char_code = ord(character) - base
            decrypted_code = (char_code - key_shift) % 26
            result.append(chr(decrypted_code + base))

            key_position += 1

        return ''.join(result)

def compute_hash(content):
    """计算内容的SHA-256哈希值"""
    return hashlib.sha256(content.encode('utf-8')).hexdigest()

class CryptSolver:
    def __init__(self, cipher_mapping, charset):
        self.cipher_mapping = cipher_mapping
        self.charset = charset
        self.vigenere = VigenereCipher("TheDaVinciCode")

    def chunk_hexstring(self, hex_string):
        """将十六进制字符串分割为块"""
        chunks = []
        position = 0

        while position < len(hex_string):
            if (position + 3 < len(hex_string) and
                hex_string[position+2:position+4] == "21"):
                chunks.append(hex_string[position:position+4])
                position += 4
            else:
                chunks.append(hex_string[position:position+2])
                position += 2

        return chunks

    def decrypt_message(self, encrypted_bytes):
        """解密消息的主流程"""
        hex_data = binascii.hexlify(encrypted_bytes).decode() if isinstance(encrypted_bytes, bytes) else encrypte

        hex_chunks = self.chunk_hexstring(hex_data)
        print(f"解析后的块: {hex_chunks}")

        translated = []
        for chunk in hex_chunks:
            try:
```

```
                index = self.cipher_mapping.index(chunk.lower())
                translated.append(self.charset[index])
            except ValueError:
                translated.append('?')

        intermediate = ''.join(translated)
        print(f"中间结果: {intermediate}")

        plaintext = self.vigenere.decrypt(intermediate)
        print(f"解密结果: {plaintext}")

        return plaintext

def main():
    cipher_mapping = build_mapping_table()

    solver = CryptSolver(cipher_mapping, CHARACTER_SET)

    encrypted = b"xxxxxxxxxxxxxxxx"
    result = solver.decrypt_message(encrypted.hex())

    return result

if __name__ == "__main__":
    main()
```

whereisflag



jadx打开apk可以看到具体逻辑

```
public void onClick(View view) {
    String obj = MainActivity.this.flagEditText.getText().toString();
    if (obj.length() != 16) {
        Toast.makeText(MainActivity.this, "flag长度错误，请继续寻找", 0).show();
    } else if (new a().b(obj)) {
        Toast.makeText(MainActivity.this, "恭喜你找到了正确的flag", 1).show();
    } else {
        Toast.makeText(MainActivity.this, "flag错误，请继续寻找", 0).show();
    }
}
```

分析之后发现核心函数是native函数

Native 函数基本介绍

- **定义**：Native 函数通过 `native` 关键字在 Java 中声明，实际代码编译在 `.so` 动态库（ELF 格式）中。
- **JNI 桥梁**：Java 层通过 JNI（Java Native Interface）调用 Native 函数，函数名和参数需遵循 JNI 规范。

```
public class a {
    public native String compute(String str);

    public boolean b(String str) {
        if (str.startsWith("ISCC{") && str.endsWith("}")) {
            return compute(str.substring(5, 15)).equals("iB3A7kSISR");
        }
        return false;
    }
}
```

用解压软件直接解压apk文件，然后进入`\lib rm64-v8a`目录找到so文件，使用IDA64打开so文件，在其中找到`Java_`开头的函数便是native导出函数

在加密函数中首先将输入倒序

```
if ( v11 )
{
  v13 = &v12[v11 - 1];
  if ( v13 > v12 )
  {
    v14 = v12 + 1;
    do
    {
      v15 = *(v14 - 1);
      *(v14 - 1) = *v13;
      *v13-- = v15;
    }
    while ( v14++ < v13 );
    v7 = *((_QWORD *)&v20 + 1);
    v8 = (unsigned __int8 *)v21;
    v9 = (unsigned __int64)(unsigned __int8)v20 >> 1;
    v10 = v20 & 1;
  }
}
```

然后根据字符表查找输入的字符

```
  *(_QWORD *)v21 = v16 + 2;
  goto LABEL_5;
 }
 v15 = (char *)v21 + 1;
 LOBYTE(v21[0]) = 2 * v6;
 if ( v6 )
LABEL_5:
   memmove(v15, v5, (size_t)v14);
 *(( BYTE *)v14 + ( QWORD)v15) = 0;
 encrypt((int)v21, v7, v8, v9, v10, v11, v12, v13, v20, v21[0], v22);
 if ( (v21[0] & 1) != 0 )
   operator delete(v23);
 (*(void (__fastcall **)(__int64, __int64, const char *))(*(_QWORD *)a1 + 1360LL))(a1, a3, v5);
 if ( (v24 & 1) != 0 )
   v17 = v26;
 else
```

字符表需要动态调试得到

```
__int64 __fastcall charToIndex(unsigned int a1)
{
  __int64 v1; // x0

  v1 = std::string::find(&xmmword_52DF0, a1, 0LL);
  if ( v1 == -1 )
    return 0xFFFFFFFFLL;
  else
    return (unsigned int)(v1 + 1);
}
```

```
__int64 __fastcall indexToChar(int a1)
{
  unsigned __int64 v1; // x9
  char *v3; // x8

  if ( a1 < 1 )
    return 0LL;
  v1 = *((_QWORD *)&xmmword_52DF0 + 1);
  if ( (xmmword_52DF0 & 1) == 0 )
    v1 = (unsigned __int64)(unsigned __int8)xmmword_52DF0 >> 1;
  if ( v1 < (unsigned int)a1 )
    return 0LL;
  if ( (xmmword_52DF0 & 1) != 0 )
    v3 = (char *)qword_52E00;
  else
    v3 = (char *)&xmmword_52DF0 + 1;
  return (unsigned __int8)v3[a1 - 1];
}
```

而根据encrypt、charToIndex、indexToChar函数的逻辑，可以看到在索引转换时有固定偏移，为2

从jadx反编译的结果得到目标密文`iB3A7kSISR`，解密

exp

```
s = "WHEReISFLAGBCDJKMNOPQTUVXYZabcdfghijklmnopqrstuvwxyz01234567890"

ss = "iB3A7kSISR"

print("".join([s[(s.index(i)-2)%len(s)] for i in ss][::-1]))
```

RE

打出flag

从可执行程序的图标判断为pyinstaller编译的程序，使用pyinstxtractor反编译

```
python pyinstxtractor.py asd.exe
```

然后打开反编译的文件夹，打开同名pyc文件，反编译（uncompyle6或者在线）
python反编译 - 在线工具

```
#!/usr/bin/env python
# visit https://tool.lu/pyc/ for more information
# Version: Python 3.8

import lzma
import base64
exec(lzma.decompress(base64.b64decode('/Td6WFoAAATm1rRGAgAhARYAAAB0L+Wj4EzVCRVdADSbSme4Ujxz7+Hf194lj8gW1Q3vdmpD9b
```

可以将decompress之后的内容写入文件（以下为部分）

```
import base64
總□□硗□□□□□ך=base64.b64encode
總□□硗□□□□□ם()
總□□硗□□□□□680=ؤ
總□□硗□□□□□□=800
總□□硗□□□□□מ=總□□硗□□□□□□.set_mode((總□□硗□□□□□ؤ,總□□硗□□□□□□))
總□□硗□□□□□□.set_caption("打出flag")
總□□硗□□□□□255,255,255)=□)
總□□硗□□□□□0,0,0)=ر)
總□□硗□□□□□□=(255,0,0)
總□□硗□□□□□□=(0,255,0)
總□□硗□□□□□□=(128,128,128)
總□□硗□□□□□□=總□□硗□□□□□□+總□□硗□□□□□□+總□□硗□□□□□挣
總□□硗□□□□□ם=30
總□□硗□□□□□□='ZpmDBMytVs5Bi0NvBYN4CoA+AXV5AMR0EBp8BYy9'
總□□硗□□□□□□=5
def 總□□硗□□□□□□(text,shift):
 總□□硗□□□□□□=""
 for 總□□硗□□□□□ↄ in text:
  if 'A'<=總□□硗□□□□□ↄ<='Z':
   總□□硗□□□□□□+=總□□硗□□□□□90)אָ-(總□□硗□□□□□□(總□□硗□□□□□65-(ↄ))
  elif 'a'<=總□□硗□□□□□ↄ<='z':
   總□□硗□□□□□□+=總□□硗□□□□□122)אָ-(總□□硗□□□□□□(總□□硗□□□□□97-(ↄ))
  else:
   總□□硗□□□□□□+=總□□硗□□□□□ↄ
 總□□硗□□□□□□=總□□硗□□□□□ך(總□□硗□□□□□□.encode()).decode()
 總□□硗□□□□□□="""
 for 總□□硗□□□□□ↄ in 總□□硗□□□□□□:
  if 'A'<=總□□硗□□□□□ↄ<='Z':
   總□□硗□□□□□□□=總□□硗□□□□□אָ((總□□硗□□□□□□(總□□硗□□□□□65-(ↄ+shift)%26+65)
   總□□硗□□□□□□+=總□□硗□□□□□□
  elif 'a'<=總□□硗□□□□□ↄ<='z':
   總□□硗□□□□□□□=總□□硗□□□□□אָ((總□□硗□□□□□□(總□□硗□□□□□97-(ↄ+shift)%26+97)
   總□□硗□□□□□□+=總□□硗□□□□□□
  else:
   總□□硗□□□□□□+=總□□硗□□□□□ↄ
 return 總□□硗□□□□□□
class 總□□硗□□□□□□(總□□硗□□□□□□.Sprite):
 def __init__(總□□硗□□□□□□):
```

叫AI写个脚本去混淆

```
import base64

def decrypt(encrypted_text, shift):
    # 逆向凯撒移位
    decrypted_caesar = []
    for c in encrypted_text:
        if 'A' <= c <= 'Z':
            shifted = (ord(c) - ord('A') - shift) % 26
            decrypted_caesar.append(chr(shifted + ord('A')))
        elif 'a' <= c <= 'z':
            shifted = (ord(c) - ord('a') - shift) % 26
            decrypted_caesar.append(chr(shifted + ord('a')))
        else:
            decrypted_caesar.append(c)
    decrypted_caesar_str = ''.join(decrypted_caesar)

    # Base64解码
    decoded_bytes = base64.b64decode(decrypted_caesar_str)
    decoded_str = decoded_bytes.decode('utf-8')

    # 字符反转
    reversed_str = []
    for c in decoded_str:
        if 'A' <= c <= 'Z':
            reversed_char = chr(ord('Z') - (ord(c) - ord('A')))
        elif 'a' <= c <= 'z':
            reversed_char = chr(ord('z') - (ord(c) - ord('a')))
        else:
            reversed_char = c
        reversed_str.append(reversed_char)
    return ''.join(reversed_str)

target = "ZpmDBMytVs5Bi0NvBYN4CoA+AXV5AMR0EBp8BYy9"
flag = decrypt(target, 5)
print(flag)
```

有趣的小游戏

附件是一个exe和两个txt，其中txt内容为非打印字符
main函数中定义了许多常量

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _QWORD *v3; // rax
  _QWORD *v4; // rax
  _QWORD *v5; // rax
  __int64 v7[3]; // [rsp+20h] [rbp-60h] BYREF
  char v8; // [rsp+3Fh] [rbp-41h] BYREF
  int v9[24]; // [rsp+40h] [rbp-40h] BYREF
  char v10[32]; // [rsp+A0h] [rbp+20h] BYREF
  int v11[31]; // [rsp+C0h] [rbp+40h] BYREF
  char v12; // [rsp+13Fh] [rbp+BFh] BYREF
  char v13[48]; // [rsp+140h] [rbp+C0h] BYREF

  sub_40B270(argc, argv, envp);
  v11[0] = 0x18A550A;
  v11[1] = 0x840630DB;
  v11[2] = 0x3EC0C129;
  v11[3] = 0x175BDB99;
  v11[4] = 0x7FD5E3DB;
  v11[5] = 0xF99F6912;
  v11[6] = 0x199B32C1;
  v11[7] = 0x836C22BB;
  v11[8] = 0x440E4880;
  v11[9] = 0xE4EC8310;
  v11[10] = 0x2F00227A;
  v11[11] = 0xAB294A2A;
  v11[12] = 0x8EDB89F1;
```

通过查看附近函数，发现其他地方也定义了常数

```
  sub_48EC30(a1);
  *(_DWORD *)(a1 + 48) = 1000;
  *(_DWORD *)(a1 + 52) = 0;
  sub_48F820(a1 + 56);
  *(_DWORD *)(a1 + 80) = 0x12345678;
  *(_DWORD *)(a1 + 84) = 0x9ABCDEF0;
  *(_DWORD *)(a1 + 88) = 0xFEDCBA98;
  *(_DWORD *)(a1 + 92) = 0x76543210;
  v2 = time64(0i64);
  srand(v2);
  qmemcpy(v32, "##########", sizeof(v32));
  nullsub_4(&v33);
  v20.m128i_i64[0] = (__int64)v32;
  v20.m128i_i64[1] = 10i64;
  sub_48F050(v21, &v20, &v33);
  qmemcpy(v34, "#   #    #", sizeof(v34));
  nullsub_4(&v35);
  v20.m128i_i64[0] = (__int64)v34;
  v20.m128i_i64[1] = 10i64;
  sub_48F050(&v22, &v20, &v35);
  qmemcpy(v36, "# # #### #", sizeof(v36));
  nullsub_4(&v37);
  v20.m128i_i64[0] = (__int64)v36;
  v20.m128i_i64[1] = 10i64;
```

查看字符串表，可以在其中找到两个txt的文件名，交叉引用查看

```
.data:00…   00000005   C
.data:00…   0000005F   C
.rdata:0…   0000000A   C   file1.txt
.rdata:0…   0000000A   C   file2.txt
.rdata:0…   00000006   C   pause
.rdata:0…   0000001E   C   terminate called recursively\n
.rdata:0…   00000031   C   terminate called after throwing an instance of '
.rdata:0…   0000002E   C   terminate called without an active exception\n
.rdata:0…   0000000C   C     what():
.rdata:0…   00000024   C   __gnu_cxx::__concurrence_lock_error
.rdata:0…   00000026   C   __gnu_cxx::__concurrence_unlock_error
.rdata:0…   00000015   C   basic_string::append
.rdata:0…   00000031   C   locale::_S_normalize_category category not found
.rdata:0…   00000020   C   locale::_Impl::_M_replace_facet
.rdata:0…   00000024   C   __gnu_cxx::__concurrence_lock_error
```

```
void __fastcall process(__int64 a1, int a2, __int64 a3)
{
  __int64 v3; // [rsp+20h] [rbp-20h]
  void (__fastcall *v4)(__int64, _QWORD, __int64); // [rsp+28h] [rbp-18h]
  __int64 v5; // [rsp+30h] [rbp-10h]
  void (__fastcall *lpAddress)(__int64, _QWORD, __int64); // [rsp+38h] [rbp-8h]

  if ( a2 <= 1 )
  {
    if ( a2 < -1 )
    {
      v4 = (void (__fastcall *)(__int64, _QWORD, __int64))sub_41C090("file2.txt");
      if ( v4 )
      {
        v3 = sub_48F610(a1);
        v4(v3, (unsigned int)a2, a3);
        VirtualFree(v4, 0i64, 0x8000u);
      }
    }
  }
  else
  {
    lpAddress = (void (__fastcall *)(__int64, _QWORD, __int64))sub_41C090("file1.txt");
    if ( lpAddress )
    {
      v5 = sub_48F610(a1);
      lpAddress(v5, (unsigned int)a2, a3);
```

其中process是我重命名的结果

可以看到其中比较奇怪的一点是程序将文件的内容作为函数执行，也就是说原本内容不可见的txt其实是函数的二进制数据，要想知道该函数的具体逻辑，需要动态调试，在此处下断点，触发断点之后在汇编步进就可以看到其中逻辑

```
debug032:0000000000190000 sub     rsp, 30h
debug032:0000000000190004 mov     [rsp+28h], r8
debug032:0000000000190009 mov     [rsp+24h], edx
debug032:000000000019000D mov     [rsp+18h], rcx
debug032:0000000000190012 xor     eax, eax
debug032:0000000000190014 sub     eax, [rsp+24h]
debug032:0000000000190018 mov     [rsp+24h], eax
debug032:000000000019001C mov     eax, 34h ; '4'
debug032:0000000000190021 cdq
debug032:0000000000190022 idiv    dword ptr [rsp+24h]
debug032:0000000000190026 add     eax, 6
debug032:0000000000190029 mov     [rsp+4], eax
debug032:000000000019002D imul    eax, [rsp+4], 9E3779B9h
debug032:0000000000190035 mov     [rsp+0Ch], eax
debug032:0000000000190039 mov     rax, [rsp+18h]
debug032:000000000019003E mov     eax, [rax]
debug032:0000000000190040 mov     [rsp+14h], eax
debug032:0000000000190044
debug032:0000000000190044 loc_190044:                         ; CODE XREF: debug032:000000000019017A↓j
debug032:0000000000190044 mov     eax, [rsp+0Ch]
```

可以将汇编扔给ai判断函数逻辑
deekseek：“这段汇编代码实现的是 **XXTEA（eXtended TEA）算法的解密过程**……”
于是知道了加解密逻辑，并且根据xxtea的密钥格式可以判断先前的两处常量中位数较短的是key，而位数较长的是密文
接下来有两种解题方式：

1. 手动分析解密逻辑，自己编写代码
2. 交给ai
   xxtea的加解密逻辑网上有很多就不细说了，直接给出解密脚本

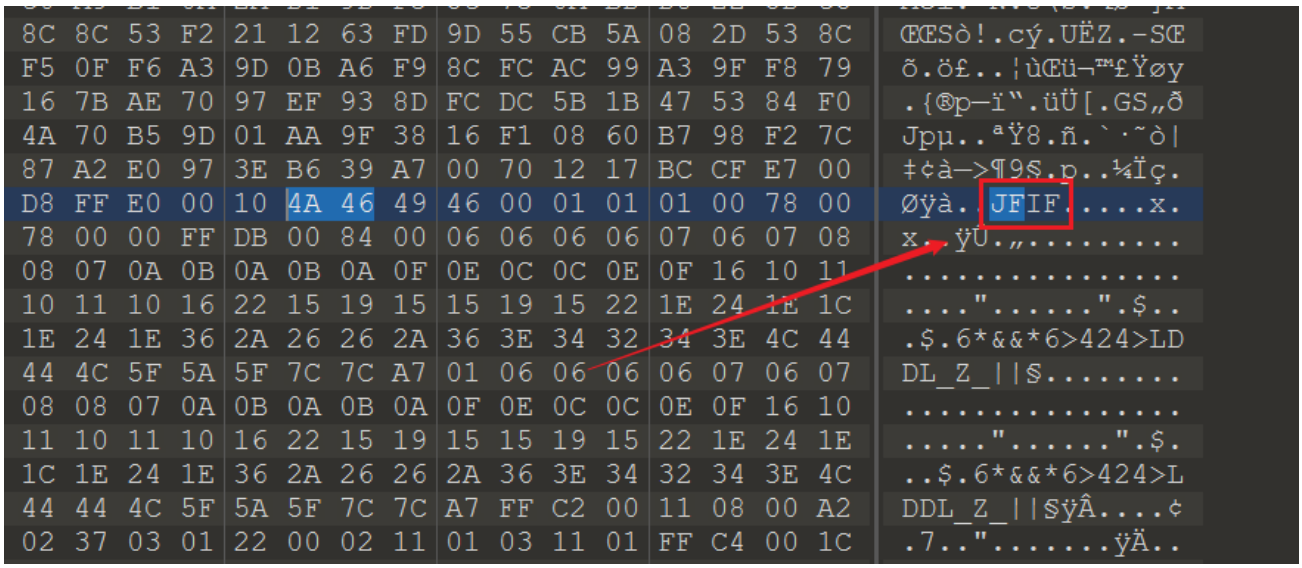
```
import base64
import struct
from typing import List

def mask_32bit(value):
    """Handle 32-bit unsigned integer overflow"""
    return value & 0xFFFFFFFF

def tea_decrypt(data: List[int], key: List[int]) -> List[int]:
    """
    Alternative implementation of XXTEA decryption

    Args:
        data: List of encrypted 32-bit integers
        key: Decryption key as 4x 32-bit integers

    Returns:
        List of decrypted 32-bit integers
    """
    data_len = len(data)
    rounds = 6 + 52 // data_len

    # Convert input data to a list that we can modify
    result = data.copy()

    # Initialize the sum value
    magic_constant = 0x9E3779B9
    accumulated_sum = mask_32bit(magic_constant * rounds)

    # Main decryption loop
    for _ in range(rounds):
        # Calculate the feistel key index
        mix_index = (accumulated_sum >> 2) & 3
```

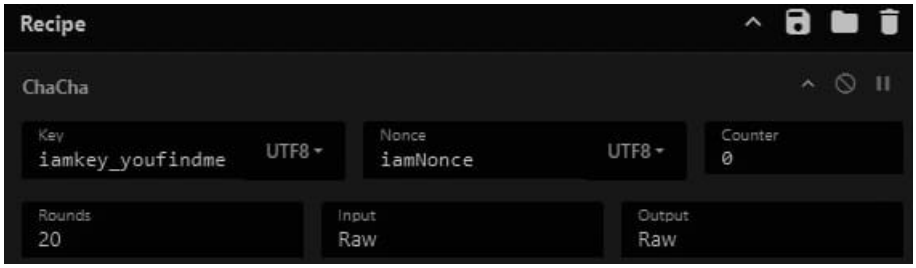
```
        # Process the data from end to beginning (except first element)
        for i in range(data_len - 1, 0, -1):
            # Get the values for the current operation
            current = result[i]
            previous = result[i - 1]
            next_val = result[0] if i == data_len - 1 else result[i + 1]

            # Calculate the mix value
            mx1 = mask_32bit((previous >> 5) ^ (next_val << 2))
            mx2 = mask_32bit((next_val >> 3) ^ (previous << 4))
            mx_sum = mask_32bit(mx1 + mx2)

            # Calculate the key part
            key_index = (i & 3) ^ mix_index
            key_mx = mask_32bit((accumulated_sum ^ next_val) + (key[key_index] ^ previous))

            # Apply the decryption transformation
            result[i] = mask_32bit(current - (mx_sum ^ key_mx))

        # Process the first element separately
        current = result[0]
        previous = result[data_len - 1]
        next_val = result[1]

        mx1 = mask_32bit((previous >> 5) ^ (next_val << 2))
        mx2 = mask_32bit((next_val >> 3) ^ (previous << 4))
        mx_sum = mask_32bit(mx1 + mx2)

        key_index = (0 & 3) ^ mix_index
        key_mx = mask_32bit((accumulated_sum ^ next_val) + (key[key_index] ^ previous))

        result[0] = mask_32bit(current - (mx_sum ^ key_mx))

        # Update the accumulated sum for next round
        accumulated_sum = mask_32bit(accumulated_sum - magic_constant)

    return result

def decrypt_and_check(encrypted_data, encryption_key, max_iterations=10000):
    """
    Repeatedly decrypt data and check for readable output

    Args:
        encrypted_data: List of encrypted 32-bit integers
        encryption_key: List of 4 32-bit integers
        max_iterations: Maximum number of decryption iterations
    """
    current_data = encrypted_data.copy()

    for iteration in range(max_iterations):
        # Decrypt one round
        current_data = tea_decrypt(current_data, encryption_key)

        # Try to interpret as text in different ways
        raw_bytes = b''.join([struct.pack("<I", val) for val in current_data])

        # Method 1: Try to decode the entire output as UTF-8
        try:
            decoded_text = raw_bytes.decode('utf-8')
            print(f"Iteration {iteration + 1}: Found valid UTF-8!")
            print(decoded_text)
        except UnicodeDecodeError:
            pass
```

```
        # Method 2: Try to extract first byte of each word
        try:
            first_bytes = bytes([raw_bytes[i] for i in range(0, len(raw_bytes), 4)])
            decoded_first = first_bytes.decode('utf-8')
            if any(c.isprintable() for c in decoded_first):
                print(f"Iteration {iteration + 1}: First bytes as text: {decoded_first}")
        except UnicodeDecodeError:
            pass

# Main execution
def main():
    # Same key and encrypted data as original
    encryption_key = [0x12345678, 0x9ABCDEF0, 0xFEDCBA98, 0x76543210]
    encrypted_data = [
        0x018A550A, 0x840630DB, 0x3EC0C129, 0x175BDB99,
        0x7FD5E3DB, 0xF99F6912, 0x199B32C1, 0x836C22BB,
        0x440E4880, 0xE4EC8310, 0x2F00227A, 0xAB294A2A,
        0x8EDB89F1, 0x28099186, 0xD04F421F, 0x23E7FD1C,
        0x6F48B862, 0x61796B6A, 0x857587A7, 0x33254C3A,
        0x06AAB088, 0x568A0B78, 0xAC64D9CF, 0xFB40A2C6,
        0x9082056A, 0x4FAAB834, 0x5D033C8B, 0x7D570A1C,
        0xCC81E29B, 0xCE1DE040
    ]

    decrypt_and_check(encrypted_data, encryption_key)

if __name__ == "__main__":
    main()
```

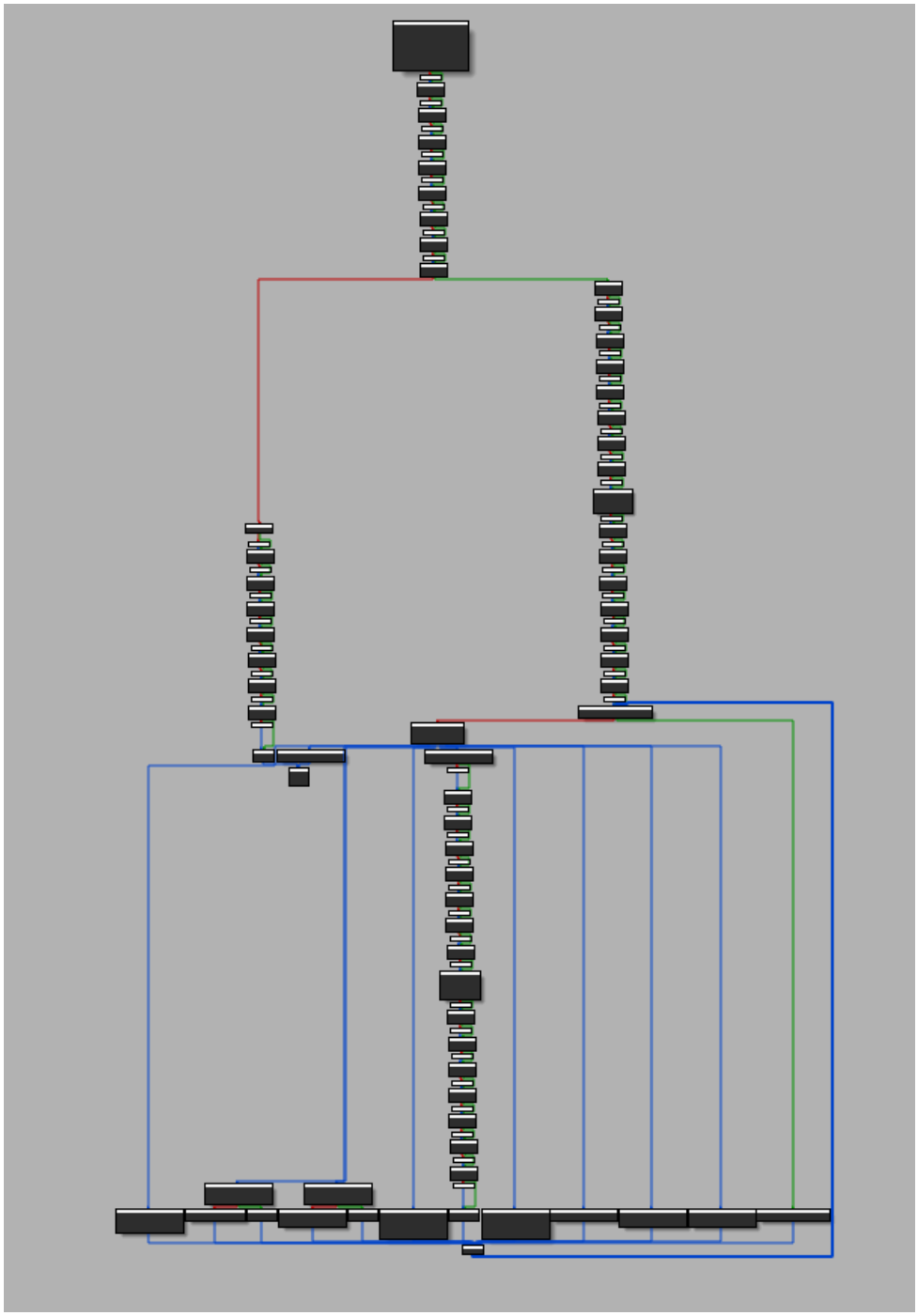
真？复杂

题目附件是一个raw文件，010editor查看发现JFIF文件头，提取图片





然后使用cyberchef解密，解密之前要先把原raw文件中附加的图片信息删除
解密之后得到压缩包一个，解密得exe文件和enc文件各一个



虽然流程图长这个样，但是是可以手动去除的

```
65      case 1:
66        if ( (unsigned int)sub_4015DE() )
67          sub_401550();
68        if ( (unsigned int)sub_401682() )
69          sub_40172A();
70        if ( (unsigned int)sub_40172A() )
71          sub_4015DE();
72        if ( (unsigned int)sub_4015DE() )
73          sub_401550();
74        if ( (unsigned int)sub_401550() )
75          sub_401682();
76        if ( (unsigned int)sub_40172A() )
77          sub_401550();
78        if ( (unsigned int)sub_401550() )
79          sub_40172A();
80        if ( (unsigned int)sub_401682() )
81          sub_40172A();
82        if ( (rand() & 1) != 0 )
83        {
84          if ( (rand() & 1) != 0 )
85          {
86            if ( (v4 & 1) != 0 )
87              v5 = 15;
88            else
89              v5 = 2;
90          }
91          else if ( (v4 & 1) != 0 )
92          {
93            v5 = 15;
94          }
95          else
96          {
97            v5 = 2;
98          }
99        }
100       else if ( (rand() & 1) != 0 )
101       {
102         if ( (v4 & 1) != 0 )
```

第一种方法：（直接忽略和输入无关的语句和函数，对于涉及到修改输入的语句统统下断点）
第二种方法：直接分析加密函数的switch逻辑，可以发现是对奇偶索引的字符做不同的变换，核心变量为v4（索引）和v5（控制跳转的case），通过`v4&1`的操作判断奇偶
通过分析exe文件可知原本逻辑是给定flag.txt，用exe加密得到enc文件，而现在只有enc文件，故需要逆向推解密逻辑
通过分析得到解密脚本

```
with open('flag.txt.enc', 'rb') as f:
    encrypted = f.read()

key = [0x88, 0x83, 0xA3, 0x7E, 0xEA, 0xA1, 0xBA, 0x25, 0x72, 0xCF, 0x1D, 0x6E, 0x79, 0x50, 0x17, 0x50]
decrypted = []
for v4, byte in enumerate(encrypted):
    if v4 % 2 == 0:  # 偶数索引处理
        temp = (~byte) & 0xFF      # 取消取反
        temp = (temp + v4) % 256   # 逆向减法
        temp ^= key[v4 % 16]       # 异或密钥
        orig = (temp - v4) % 256   # 逆向加法
    else:              # 奇数索引处理
        temp = byte ^ v4           # 取消异或v4
        temp = (temp - v4) % 256   # 逆向加法
        temp ^= key[v4 % 16]       # 异或密钥
        orig = (temp + v4) % 256   # 逆向减法
    decrypted.append(orig)

# 输出可打印字符（避免解码错误）
print(''.join([chr(b) if 32 <= b <= 126 else '.' for b in decrypted]))
```

faze

题目附件：faze.exe
使用IDA打开附件

```
text:0000000000401DBE loc_401DBE:                              ; CODE XREF: main+CD↑j
text:0000000000401DBE                 lea     rax, [rbp+390h+var_3B0]
text:0000000000401DC2                 mov     rcx, rax
text:0000000000401DC5                 call    _Z3xP9PA10_i    ; xP9(int (*)[10])
text:0000000000401DCA                 lea     rax, [rbp+390h+var_3BD]
text:0000000000401DCE                 mov     rcx, rax
text:0000000000401DD1                 call    _Z3mS2Pc        ; mS2(char *)
text:0000000000401DD6                 lea     rdx, [rbp+390h+var_3BD]
text:0000000000401DDA                 lea     rax, [rbp+390h+var_3D0]
text:0000000000401DDE                 mov     r8, rdx
text:0000000000401DE1                 lea     rdx, aIsccS     ; "ISCC{%s}"
text:0000000000401DE8                 mov     rcx, rax
text:0000000000401DEB                 call    _Z9sprintf_sILy19EEiRAT__cPKcz ; sprintf_s<19ull>(char (&)[19ull],char const*,...)
text:0000000000401DF0                 lea     rax, [rbp+390h+var_3F0]
text:0000000000401DF4                 mov     rcx, rax
text:0000000000401DF7                 call    _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1Ev ; std::__cxx11::basic_string<char,std::cha
text:0000000000401DFC                 lea     rdx, aEnterFlag ; "Enter flag:"
text:0000000000401E03                 mov     rcx, cs:_refptr__ZSt4cout
text:0000000000401E0A ;   try {
text:0000000000401E0A                 call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::operator<<<std::char_traits<char>>(st
text:0000000000401E0F                 mov     rdx, cs:_refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
text:0000000000401E16                 mov     rcx, rax
text:0000000000401E19                 call    _ZNSolsEPFRSoS_E ; std::ostream::operator<<(std::ostream & (*)(std::ostream &))
text:0000000000401E1E                 lea     rax, [rbp+390h+var_3F0]
text:0000000000401E22                 mov     rdx, rax
text:0000000000401E25                 mov     rcx, cs:_refptr__ZSt3cin
text:0000000000401E2C                 call    _ZSt7getlineIcSt11char_traitsIcESaIcEERSt13basic_istreamIT_T0_ES7_RNSt7__cxx1112basic_stringIS4_S5_T
text:0000000000401E31                 lea     rdx, [rbp+390h+var_3D0]
text:0000000000401E35                 lea     rax, [rbp+390h+var_3F0]
text:0000000000401E39                 mov     rcx, rax
text:0000000000401E3C                 call    _ZSteqIcSt11char_traitsIcESaIcEEbRKNSt7__cxx1112basic_stringIT_T0_T1_EEPKS5_ ; std::operator==<char>
text:0000000000401E41                 test    al, al
text:0000000000401E43                 jz      short loc_401E69
text:0000000000401E45                 lea     rdx, aCorrect   ; "Correct!"
text:0000000000401E4C                 mov     rcx, cs:_refptr__ZSt4cout
```

一眼C++，通过判断代码可以发现目标字符串在用户输入之前（getline）已经完成了目标字符串的初始化，所以这里有多种解法

1. 在sprintf上下断点，直接查看写入目标字符串的内容
2. 在比较的时候（ `operator==` ）下断点，查看比较的数据
   这里选择前者，在程序暂停时跳转到rcx所在地址

```
003C60]:000000000079FA62 db    0
003C60]:000000000079FA63 db  28h ; (
003C60]:000000000079FA64 db  3Ah ; :
003C60]:000000000079FA65 db  26h ; &
003C60]:000000000079FA66 db  5Bh ; [
003C60]:000000000079FA67 db  47h ; G
003C60]:000000000079FA68 db  42h ; B
003C60]:000000000079FA69 db  5Ch ; \
003C60]:000000000079FA6A db  58h ; X
003C60]:000000000079FA6B db  72h ; r
003C60]:000000000079FA6C db  46h ; F
003C60]:000000000079FA6D db  61h ; a
003C60]:000000000079FA6E db  29h ; )
003C60]:000000000079FA6F db    0
```

greeting

首先IDA打开可执行文件，会发现有些函数反编译的结果不正确，且提示错误，因此可以查看目标函数附近的汇编代码，找到类似加密逻辑的代码

```
.text:00000001400016B0 loc_1400016B0:                          ; CODE XREF: sub_140001220+4D0↓j
.text:00000001400016B0                 mov     r8, [rbp+70h+var_60]
.text:00000001400016B4
.text:00000001400016B4 loc_1400016B4:                          ; CODE XREF: sub_140001220+4C6↓j
.text:00000001400016B4                 mov     [r8+rsi], r12b
.text:00000001400016B8                 inc     rsi
.text:00000001400016BB                 mov     [rbp+70h+var_58], rsi
.text:00000001400016BF                 cmp     r14, rsi
.text:00000001400016C2                 jz      short loc_1400016F2
.text:00000001400016C4
.text:00000001400016C4 loc_1400016C4:                          ; CODE XREF: sub_140001220+482↑j
.text:00000001400016C4                 mov     rax, rsi
.text:00000001400016C7                 mul     r15
.text:00000001400016CA                 shr     dl, 2
.text:00000001400016CD                 movzx   eax, dl
.text:00000001400016D0                 lea     eax, [rax+rax*4]
.text:00000001400016D3                 lea     r12d, [rsi+5Ah]
.text:00000001400016D7                 xor     r12b, [rbx+rsi]
.text:00000001400016DB                 mov     ecx, esi
.text:00000001400016DD                 sub     ecx, eax
.text:00000001400016DF                 rol     r12b, cl
.text:00000001400016E2                 cmp     rsi, [rbp+70h+var_68]
.text:00000001400016E6                 jnz     short loc_1400016B4
.text:00000001400016E6 ; ---------------------------------------------------------------------------
.text:00000001400016E8                 db 48h
.text:00000001400016E8 ;   } // starts at 140001605
.text:00000001400016E9 ; ---------------------------------------------------------------------------
```

明显的异或和循环左移操作，大概率是加密逻辑
通过分析可知，代码首先是计算一个偏移，然后将目标数据对应索引的字节在异或`i+0x5a`之后（esi为索引）循环左移该计算出来的偏移，因此目标可以分为两步：

1. 分析该偏移的计算方式
2. 反推整个加密逻辑
   这里的r15其实是一个固定的值

```
loc_140001689:                          ; CODE XREF: sub_140001220+34D↑j
                add     rbx, rax
                mov     r8d, 1
                xor     esi, esi
                mov     r15, 0CCCCCCCCCCCCCCCDh
                lea     rdi, [rbp+70h+var_68]
                jmp     short loc_1400016C4
```

关于偏移量的计算

- 通过手动分析

- `mul r15` 和 `shr dl, 2` 的组合实际上执行的是整数除法 `i / 5`
- `lea eax, [rax+rax*4]` 计算的是 `(i/5)*5`
- `sub ecx, eax` 计算的是 `i - (i/5)*5`
- 以上逻辑等价于 `i%5`

- 直接动态调试可以发现rol操作中cl的取值是0、1、2、3、4、0……，所以其实偏移的计算方式是索引对5取余

然后就是逆向整个加密逻辑，有了偏移的计算方式，解密的逻辑很好推，就是对每个字节先循环右移再异或`(i+0x5a)`
对于密文，通过交叉引用和人肉分析等方式最终可以找到位于`0x014001B390`

因此完整的解密脚本如下

```
def encrypt(input_bytes):
    output = bytearray(len(input_bytes))
    for i in range(len(input_bytes)):
        div_result = (i // 10) * 5

        value = (i + 0x5A) & 0xFF
        value ^= input_bytes[i]

        rot_amount = (i - div_result) & 0x7
        value = ((value << rot_amount) | (value >> (8 - rot_amount))) & 0xFF

        output[i] = value

    return output

def decrypt(encrypted_bytes):
    output = ""
    for i in range(len(encrypted_bytes)):
        rot_amount = i % 5

        value = encrypted_bytes[i] & 0xFF
        value = ((value >> rot_amount) | (value << (8 - rot_amount))) & 0xFF

        value ^= (i + 0x5A)

        output += chr(value & 0xFF)

    return output

def main():
    encrypted_hex = "xxxxxxxxxxx"
    encrypted_bytes = bytes.fromhex(encrypted_hex)

    decrypted = decrypt(encrypted_bytes)
    print("Decrypted:", decrypted)

if __name__ == "__main__":
    main()
```