lab3：Crypto

这节课的主题是 “When Cryptography Meets Reality”。前面先讲了为什么很多 CTF Crypto 会显得无聊，因为有些题太像猜谜、脑洞或者很长的缝合题。比较好的方向应该是经典但自然的 RSA、DLP（Discrete Logarithm Problem，离散对数问题），新知识比如 FHE（Fully Homomorphic Encryption，全同态加密）、ZKP（Zero-Knowledge Proof，零知识证明）、PQC（Post-Quantum Cryptography，后量子密码），或者现实协议里的实现漏洞。后面讲了 FHE 的手套箱故事，意思是可以在不看到明文的情况下对密文做计算；ZKP 则是可以证明自己知道某件事，但不泄露具体内容。课件后半部分更贴近现实漏洞，比如 missing subgroup check（缺少子群检查）、missing curve check（缺少曲线检查）、ambiguous encoding（编码有歧义）、TSS（Threshold Signature Scheme，门限签名）协议里为了效率删检查导致的问题。

1. Role Token / Ambiguous Encoding

这题对应 Ambiguous encoding Vulnerability（第 21 页）。把它放到一个支付 token 的场景里，服务端有两个模式。`sign` 模式负责给 token 签名，它会检查这个 token 看起来是不是普通请求；`verify` 模式负责验签和发 flag，它会检查同一个 token 最后是不是变成了高额支付请求。漏洞点在于两个模式解析重复参数的方式不一样。

```
from hashlib import sha256
from urllib.parse import parse_qs
from secret import flag, key
def mac(s): return sha256(key + s).hexdigest()
op, raw = input("op = "), bytes.fromhex(input("token = "))
if op == "sign":
    q = parse_qs(raw); print(mac(raw) if q.get(b"amount", [b""])[0] != b"1000000"
and q.get(b"role", [b""])[0] != b"vip" else "no")
else:
    sig = input("sig = "); v = dict(x.split(b"=", 1) for x in raw.split(b"&"))
    print(flag if sig == mac(raw) and v.get(b"amount") == b"1000000" and
v.get(b"role") == b"vip" else "no")
```

这里输入的 token 是十六进制。`sign` 模式用 `parse_qs` 解析 URL 参数。如果一个参数出现多次，比如 `amount=10&amount=1000000`，`parse_qs` 会把它保存成列表，代码里取的是第一个值。这样它看到的就是 `amount=10` 和 `role=user`，所以愿意给这个 token 签名。

但是 `verify` 模式用了 `dict(x.split(...))`。同一个键出现多次时，Python 字典会保留后面的值，所以它看到的会变成 `amount=1000000` 和 `role=vip`。同一个 token，在签名的时候像普通请求，在验证的时候像高额请求，这就是编码歧义带来的问题。

我构造的原始 token 是 `user=bill&amount=10&role=user&amount=1000000&role=vip`。先把它交给 `sign`，服务端按第一个 `amount` 和第一个 `role` 判断，认为没问题，于是返回签名。然后把完全相同的 token 和签名交给 `verify`，这次服务端按最后一个 `amount` 和最后一个 `role` 判断，就会满足拿 flag 的条件。

```
token =
757365723d62696c6c26616d6f756e743d313026726f6c653d7573657226616d6f756e743d313030303
0303026726f6c653d766970
sig = 5ba850bb6051743d7e0a1a84190bcc2416f5d40b0a357e9b38426ef2f53532ec
flag = AAA{crypt0_3}
```

`secret.py` 里面有 flag 和签名用的 key。solve 不需要知道 key，只是先签名，再用同一个 token。

2. onelinecrypto

课件第 27 页。

```
assert __import__('re').fullmatch(r'SEE{\w{23}}',flag:=input()) and not
int.from_bytes(flag.encode(),'big')%13**37
```

它要求输入满足两个条件。第一，格式必须是 `SEE{...}`，中间正好 23 个 `\w` [A-Za-z0-9_] 字符。第二，把整个字符串按 big endian 转成整数后，必须能被 13^{37} 整除。

solve 里没有直接爆破 63^{23} 种可能。我先把每个未知字符写成 `85 + x_i`，因为 85 大概在可打印字符中间，合法字符对应的 `x_i` 通常不会特别大。然后把整除条件写成

```
固定部分 + Σ(85 + x_i) * 256^位置 = 0 mod 13^37
```

接着构造一个格。这个格里的短向量会同时让最后的模 13^{37} 余数变成 0，让每个 `x_i` 尽量小，也就是让字符落在正常可打印范围附近。脚本用 LLL 先把格基规约一下，再枚举短向量。枚举到候选以后，再用原题的正则和整除条件检查，最后得到真正合法的输入。

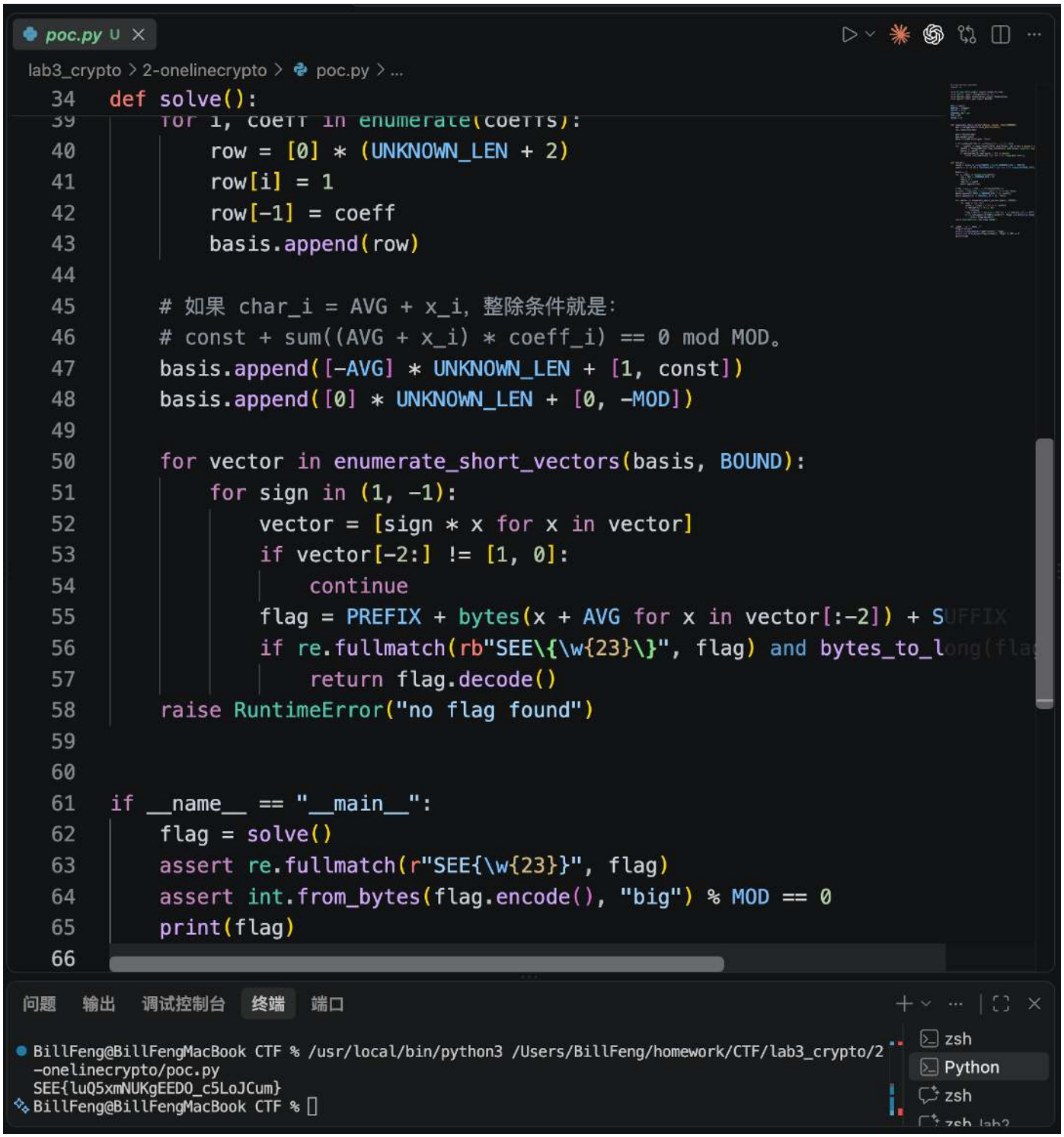

Image 1

SEE{luQ5xmNUKgEEDO_c5LoJCum}