Phòng thí nghiệm An toàn Thông tin InsecLab
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO THỰC HÀNH

Môn học: NT219

Tên chủ đề: LAB6

GVHD: Tô Trọng Nghĩa

Nhóm: GHI (tên nhóm tự đặt)

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT219.N21.ANTT

STT	Họ và tên	MSSV	Email
1	Huỳnh Nguyễn Uyển Nhi	21522424	

2. NỘI DUNG THỰC HIỆN:[1]

STT	Nội dung	Tình trạng	Trang
1	Yêu cầu 1	90%	1 - 3
2	Yêu cầu 2	50%	3 - 5
3	…	…	…
Điểm tự đánh giá			**10/10**

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

1. Thuật toán mã hoá DES

- **Bài tập 1: (1đ)** Hãy tìm ra cặp plaintext-ciphertext để khoá dưới đây trở thành khoá yếu.
- **FE FE 1F 1F FE FE 0E 0E**

Key: FEFE1F1FFEFE0E0E. Thuộc nhóm khóa yếu vì chỉ tạo ra 4 khóa con khác nhau (thay vì 16 khóa khác nhau). Em nghĩ mấu chốt ở việc nó tạo key lặp lại 1 byte. FE FE rồi 1F 1F... làm cho từ 8 bytes xuống còn 4 bytes.

Giả sử: chọn plaintext là 00 00 00 00 00 00 00 00

Thì ciphertext sẽ tìm được là: F412D63913F9E278

des_weak.py > ...

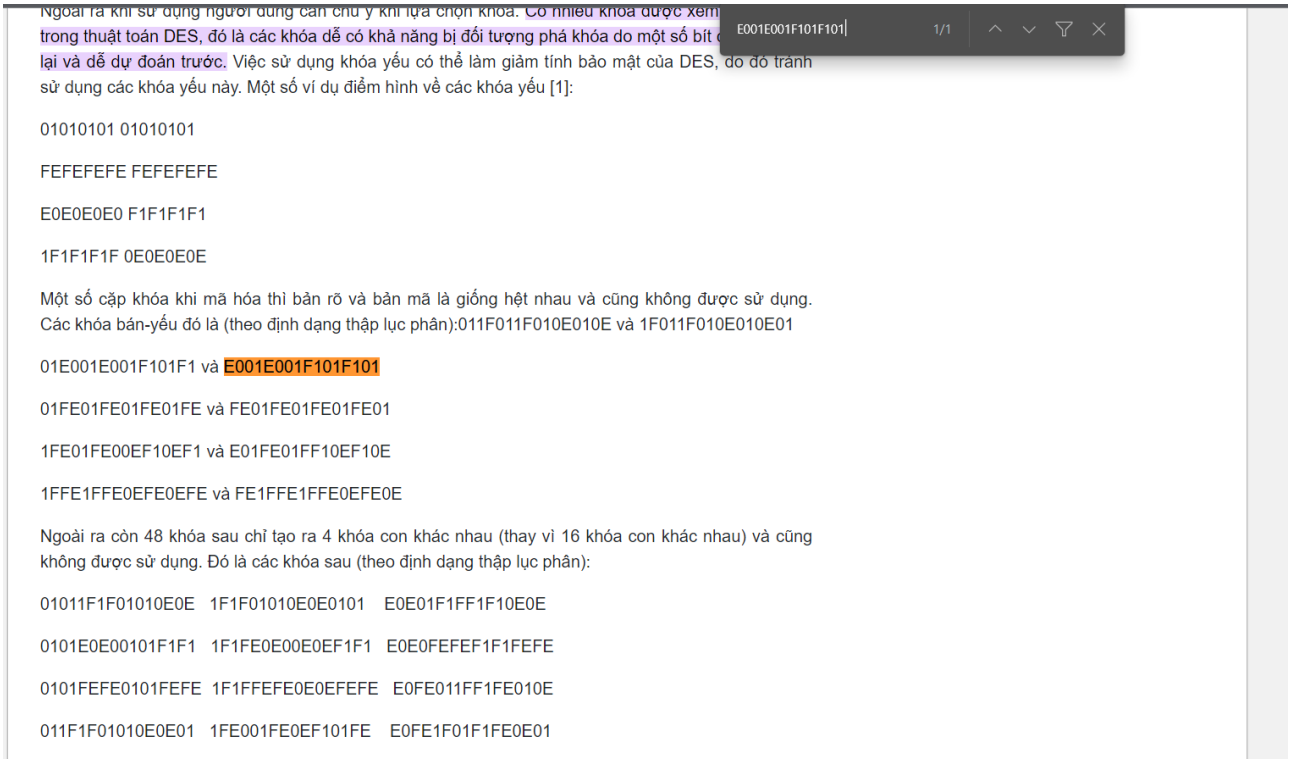
```python
import base64
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
def encrypt (plaintext, key):
    cipher = DES.new(key, DES.MODE_ECB)
    ciphertext = cipher.encrypt(plaintext)
    #plaintext = unpad(plaintext, DES.block_size)
    return ciphertext

plaintext = "0000000000000000"
key = "FEFE1F1FFEFE0E0E"
pl_b = bytes.fromhex(plaintext)
key_b = bytes.fromhex(key)
print (pl_b)
print(key_b)
cipher = encrypt(pl_b,key_b)
print(cipher.hex().upper())
```

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL

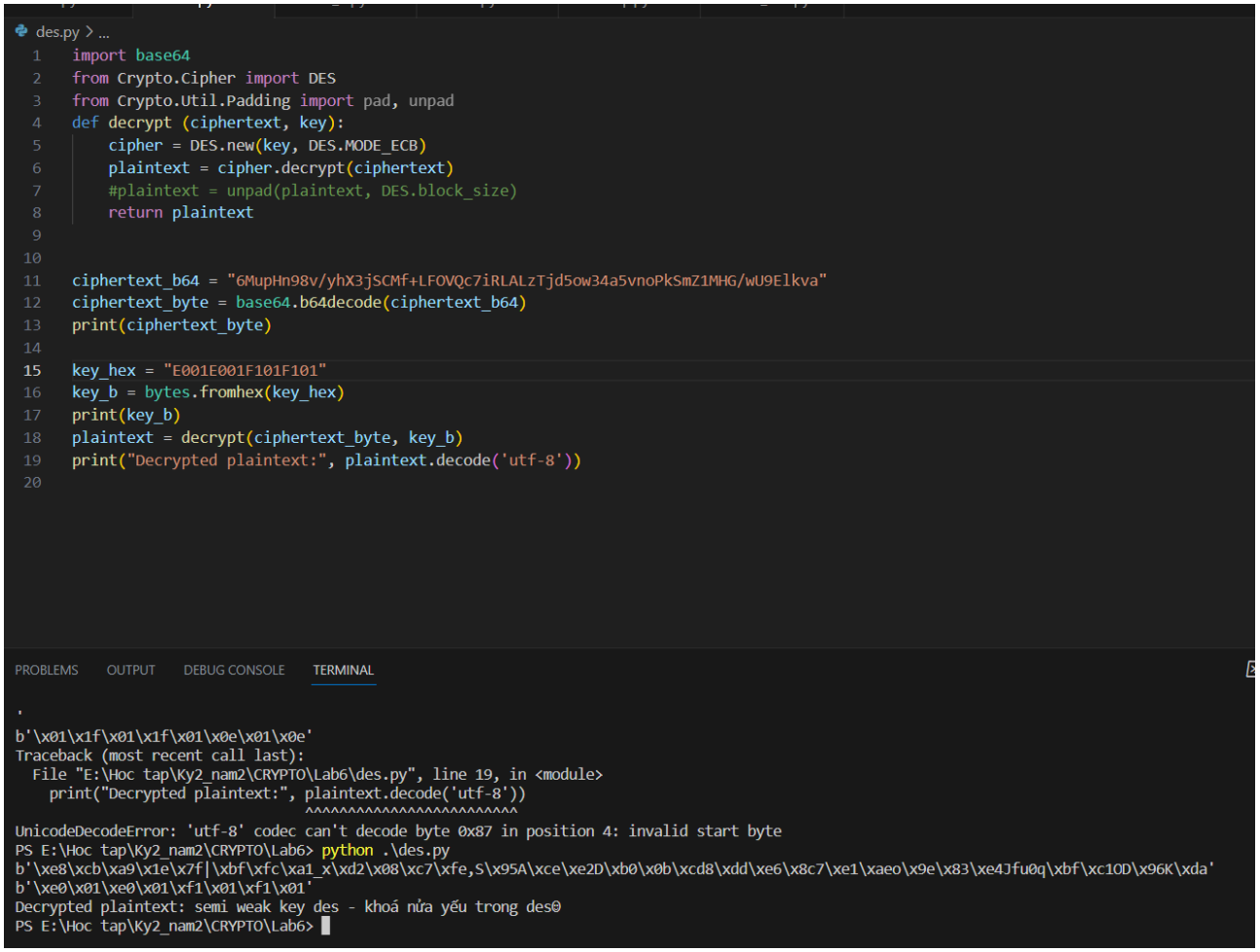
```
PS E:\Hoc tap\Ky2_nam2\CRYPTO\Lab6> python .\des_weak.py
b'\x00\x00\x00\x00\x00\x00\x00\x00'
b'\xfe\xfe\x1f\x1f\xfe\xfe\x0e\x0e'
F412D63913F9E278
```

- **Bài tập 2: (1đ)** Cho chương trình mã hoá **DES** bên dưới. Kết quả mã hoá được encode base64. Biết **DES** mã hoá **mode ECB** và khoá có liên quan đến khoá yếu và khoá nửa yếu. Hãy tìm ra khoá mà **mã hoá** ciphertext bên dưới ra kết quả là plaintext ban đầu.
- **Plaintext ban đầu**: "*semi weak key des - khoá nửa yếu trong des*"
- **cipher text:**
- 6MupHn98v/yhX3jSCMf+LFOVQc7iRLALzTjd5ow34a5vnoPkSmZ1MHG/wU9Elkva

Bruteforce các key trong list weak-key và semi-weak key.



Reference: [Giới thiệu về thuật toán DES, 3DES và khuyến nghị khi sử dụng trong các sản phẩm mật mã dân sự - Bkav Corporation - nacis.gov.vn - Cổng thông tin điện tử Ban Cơ yếu Chính phủ]



2. Thuật toán mã hoá AES

- **Bài tập 3: (1đ)** Các trường hợp nào nên sử dụng mode CBC, và trường hợp nào nên sử dụng mode OFB trên AES. Cho ví dụ.

- Mode CBC (Cipher Block Chaining):
Chế độ CBC thường được sử dụng cho các ứng dụng yêu cầu độ tin cậy và độ chống tấn công cao hơn.
CBC thích hợp khi cần mã hóa dữ liệu có kích thước lớn hoặc dữ liệu liên tục.
Các ứng dụng như VPN (Virtual Private Network), TLS (Transport Layer Security), IPsec
(Internet Protocol Security) thường sử dụng mode CBC.

Ví dụ: Trong một kết nối VPN, để bảo mật dữ liệu, mode CBC có thể được sử dụng để mã

hóa các gói tin trước khi gửi qua kênh truyền. Mỗi khối dữ liệu sẽ được XOR với khối mã

hóa trước đó trước khi được mã hóa, tạo ra một chuỗi dữ liệu được mã hóa liên tục.

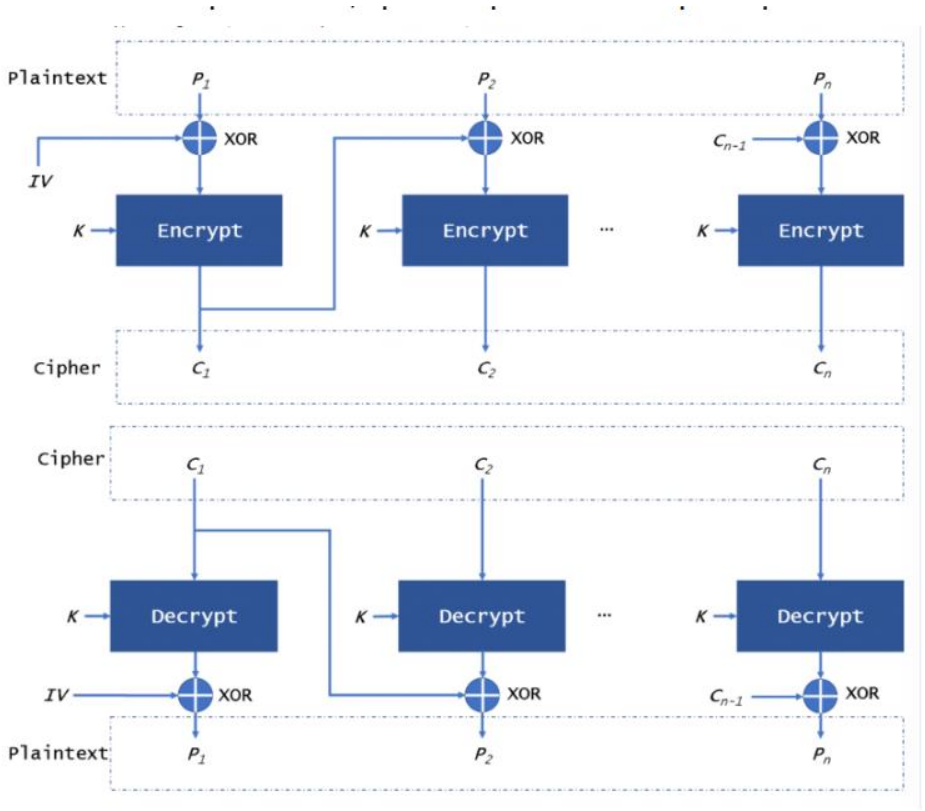


Figure 1: CBC mode

- Mode OFB (Output Feedback):

OFB thường được sử dụng trong các ứng dụng yêu cầu tính thời gian thực hoặc không đồng bộ, ví dụ như video streaming.

Nó cung cấp tính toàn vẹn dữ liệu, vì bất kỳ sự thay đổi nào trong dữ liệu mã hóa không ảnh hưởng đến các khối sau đó.

OFB thích hợp cho việc mã hóa dữ liệu có kích thước không cố định, ví dụ như mã hóa dữ liệu trực tuyến.

Ví dụ: Trong việc mã hóa video streaming, mode OFB có thể được sử dụng để mã hóa các

khung hình video một cách không đồng bộ. Mỗi khung hình video sẽ được mã hóa bằng việc XOR với một chuỗi dữ liệu ngẫu nhiên được tạo ra từ khóa. Điều này cho phép việc truyền tải video mà không cần chờ đến khi tất cả các khối video được mã hóa

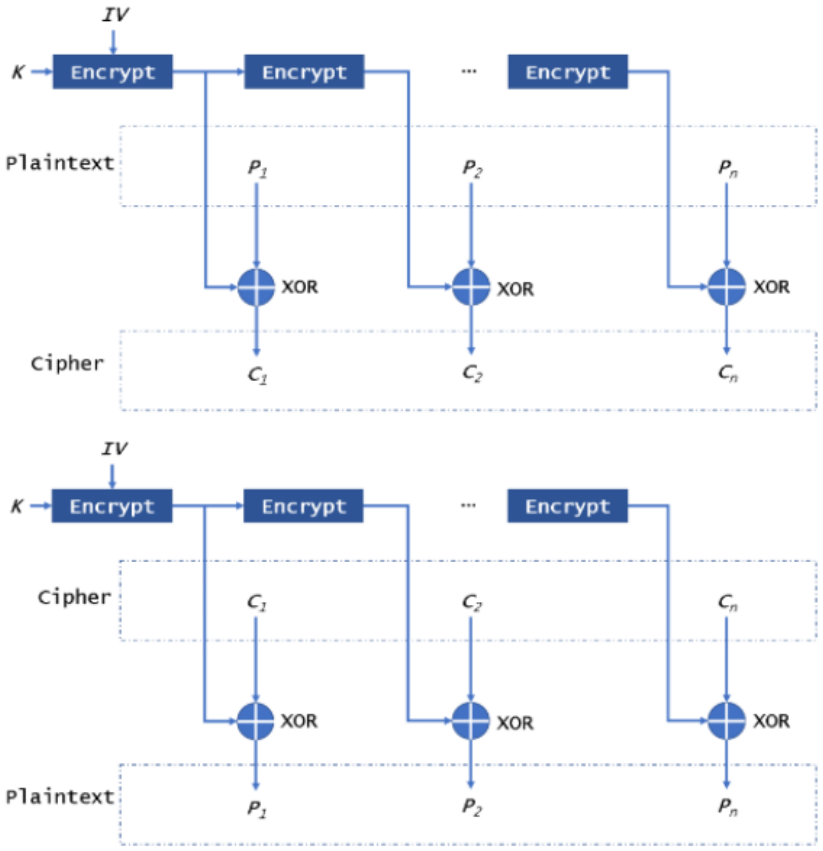


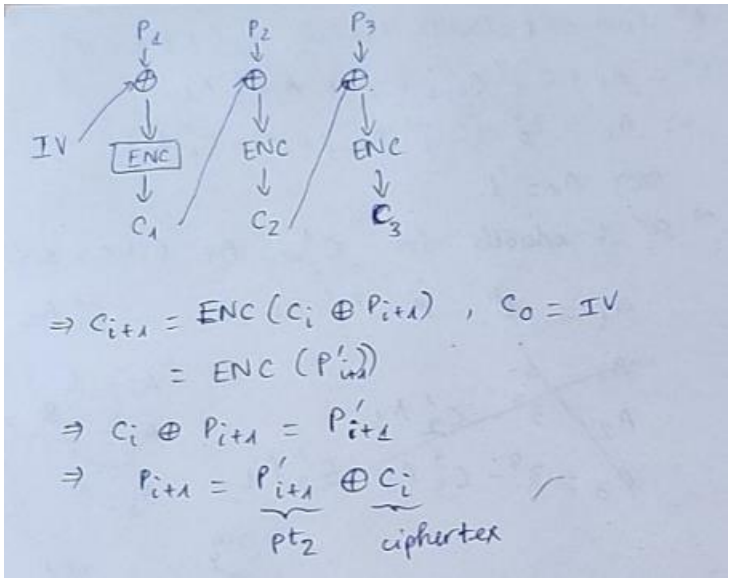
Figure 2: OFB mode

- **Bài tập 4: (1đ)** Cho chương trình mã hoá AES bên dưới. Kết quả mã hoá sử dụng AES mode CBC để mã hoá và sử dụng AES mode ECB để giải mã dựa trên cipher của kết quả mã hoá. Hãy tìm ra đoạn mã rõ ban đầu của chương trình
- Mã hoá sử dụng AES-256 và Block_size = 16

Idea: Vì block-size là 16 nên chia ciphertext và plaintext2 thành từng block có size tương ứng

```
list_cipher_block = []
list_plaintext2_block = []
list_plaintext_block = []
for i in range(0, len(cipher), 16):
    list_cipher_block.append(cipher[i:i+16])

for i in range(0, len(plaintext2), 16):
    list_plaintext2_block.append(plaintext2[i:i+16])
```

Ta phân tích quá trình Enc và Dec như sau:



Code:

```
for i in range(len(list_cipher_block)):
    if i == 0 :
        tmp = bytes([a ^ b for a , b in zip(list_plaintext2_block[0],iv) ])
        list_plaintext_block.append(tmp)
    else:
        tmp = bytes([a ^ b for a , b in zip(list_plaintext2_block[i],list_cipher_block[i-1]) ])
        list_plaintext_block.append(tmp)

flag = bytes().join(list_plaintext_block)


print(flag.decode())
```

Decrypt bằng cách xor, trong đó block đầu tiên xor với iv, các block còn lại xor với plaintext của block trước đó.

```python
iv = "yIwZn0RUXvlw9lxtVhCuOg=="
cipher = "XEkdm4AGP6oQK0OfKhN7a/fTBb+XyUMlKOHAKoPW7wUzrcbuJnj40jaCfVVilHmLU1V/dXIltUF8dsAhC8LDBIIBhvKzDucvYRwg1HwX9zAwxmeRIq9qFs6+ei80
plaintext2 = "hU+6vyyX7ZhQtxk+dtQ//OyopjjjJkvC8ZDy/Ap7EopMVGuf46GAkUaG4Fv2t88wE86uD53Rm/JYRv47BbQa45D4ER1SSXbhXDekRVmttmrmSuOLny60WgNel

iv = base64.b64decode(iv)

cipher = base64.b64decode(cipher)

plaintext2 = base64.b64decode(plaintext2)
print(len(iv))
print(len(cipher))
print(len(plaintext2))

list_cipher_block = []
list_plaintext2_block = []
list_plaintext_block = []
for i in range(0, len(cipher), 16):
    list_cipher_block.append(cipher[i:i+16])

for i in range(0, len(plaintext2), 16):
    list_plaintext2_block.append(plaintext2[i:i+16])

#bytes([a ^ b for a, b in zip(bytes1, bytes2)])
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS E:\Hoc tap\Ky2_nam2\CRYPTO\Lab6> python .\aes_cbc.py
16
224
224
Mã hóa AES được thực hiện thông qua 5 chức năng chính là AddRoundKey, SubBytes, ShiftRows, MixColumns và KeyExpansion. Năm chức năng này được sắp xếp để thực hiện ba bước cơ bản
.vvv
PS E:\Hoc tap\Ky2_nam2\CRYPTO\Lab6>
```

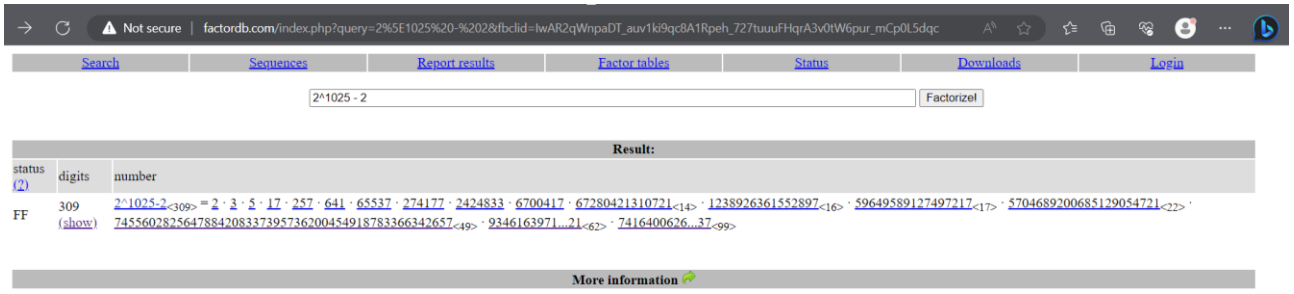
3. Thuật toán mã hoá RSA

- **Bài tập 5: (1đ)** Các tham số của public key rsa lần lượt là

Ý tưởng là cycle RSA attack, do 2^1025 - 2 mình factor ra đc.

Solve dựa trên bài này:

[GCTF 2022] Cycling | /dev/ur4ndom - Robin Jadoul



```python
# From factordb
factors = [2, 3, 5, 17, 257, 641, 65537, 274177, 2424833, 6700417, 67280421310721, 1238926361552897,
           59649589127497217, 5704689200685129054721,
           7455602825647884208337395736200454918783366342657, (2 ** 256 + 1) // 1238926361552897,
           (2 ** 512 + 1) // 180785917665242360085553923151981577020782265587640012 81]
assert 2 ** 1025 - 2 == prod(factors)

C = []
for ps in powerset(factors):
    v = prod(ps) + 1
    if isprime(v):
        C.append(prod(ps) + 1)
Ξ = prod(C)

e = 65537
n = 194326416639702286024546132569291140934764160534429254756332542456564420846769802418264203845642726291345024973700618220765757471 1
ct = 855170488574088974430975503298702010186351242562518088813862740360716130495382273899339168052034526559264504306866252070816936692 9

d = pow(e, -1, Ξ)
print (long_to_bytes(pow(ct,d,n)))
```

```
PROBLEMS 9  OUTPUT  DEBUG CONSOLE  TERMINAL                    Python
PS E:\Hoc tap\Ky2_nam2\CRYPTO\Lab6> python .\rsa_cycleattack.py
b'https://ur4ndom.dev/posts/2022-07-04-gctf-cycling/'
PS E:\Hoc tap\Ky2_nam2\CRYPTO\Lab6>
```

- **Bài tập 6: (1đ)** Luyện tập sử dụng RSA với các thông số sau.
- **Cho n =**

Vì n quá lớn và e = 13 nhỏ. Nên quá trình encrypt: C = m^e % N = m^e.

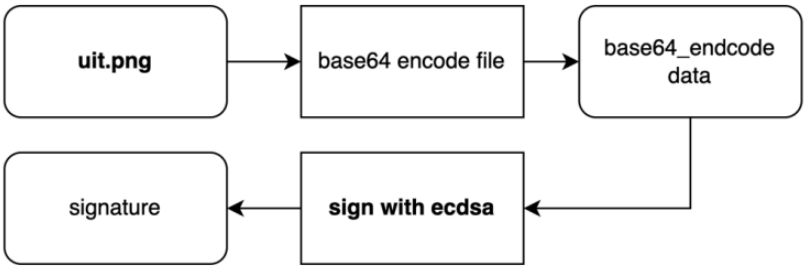
Vì thế có C rồi nên chỉ cần tính căn bậc e của C là ra m.

```python
from Crypto.Util.number import*
from gmpy2 import *
n =435227995514595233481777875128553267789518490019706910041264522576167370450264035329257425687482934218288024104078760071937867729665
c =476571097361060712805283430610657498271936961456822634648857175380529608803901802760993465949620967958626141563851005820592167730085
e = 13
import gmpy2
for i in range(10000000):
    m, is_true_root = gmpy2.iroot(i*n + c, e)
    if is_true_root:
        print(f"Found i = {i}")
        print("Message success: {}".format(bytearray.fromhex(format(m, 'x')).decode()))
        break
    else: continue

```

```
PS E:\Hoc tap\Ky2_nam2\CRYPTO\Lab6> python .\rsa_13.py
Found i = 0
Message success: wao. m^e < n is so bad !!!
```

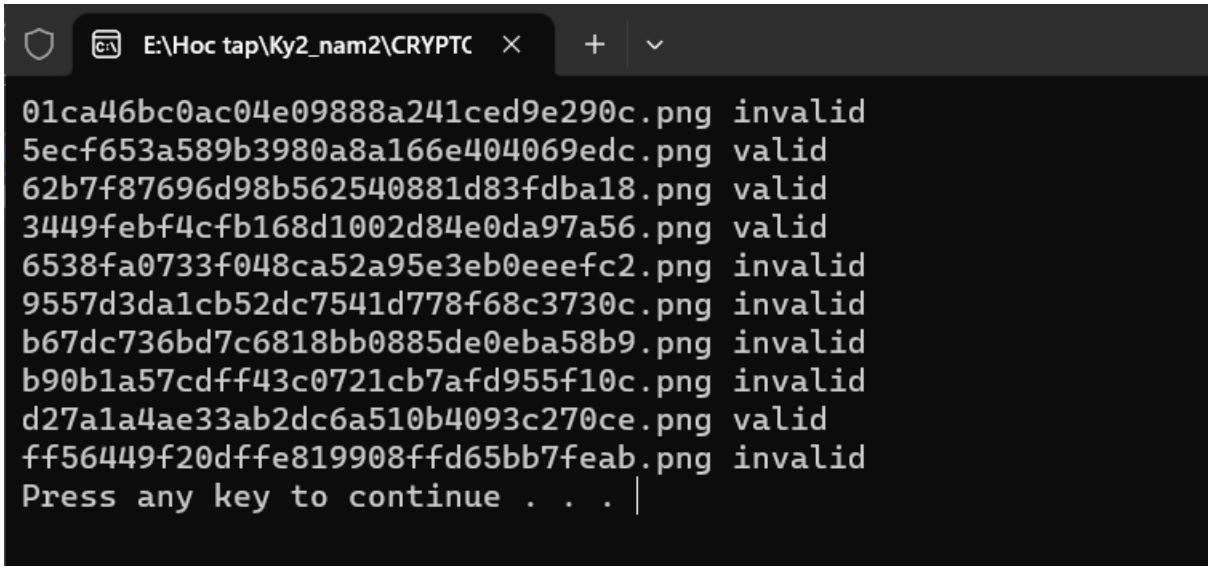
4. Thuật toán mã hoá Elliptic Curve

- **Bài tập 7:(2đ)** Cho chương trình mã hoá chữ ký bằng ECC. Hãy viết một chương trình xác thực chữ ký với các tập tin hình ảnh được cung cấp. Kiểm tra signature đính kèm nào là đúng với tập tin ban đầu.(có 10 signature cần kiểm tra)
- ECDSA<ECP, SHA1>




```
int main(int argc, char* argv[])
{
    //Load public key
    ECDSA<ECP, SHA1>::PublicKey publicKey;
    LoadPublicKey("ec.public.key", publicKey);

    //Verify message
    std::vector<string> lsdir = { "01ca46bc0ac04e09888a241ced9e290c.png", "5ecf653a589b3980a8a166e404069edc.png", "62b7f87696d98b562540881d83fdba18.png",

    for (int i = 0; i < 10; i++) {
        bool result = false;
        string filename = lsdir[i];
        string message;
        FileSource file(filename.c_str(), true, new CryptoPP::StringSink(message));

        string image = message.substr(0, message.length() - 42);
        string signature = message.substr(message.length() - 42);

        string encoded;
        StringSource(image, true, new Base64Encoder(new StringSink(encoded)));

        result = VerifyMessage(publicKey, encoded, signature);
        string result_str = (result) ? "valid" : "invalid";
        cout << filename << " " << result_str << endl;

    }
    system("pause");
    return 0;
}
```

```
void LoadPublicKey(const string& filename, ECDSA<ECP, SHA1>::PublicKey& key)
{
    key.Load(FileSource(filename.c_str(), true /*pump all*/).Ref());
}

bool VerifyMessage(const ECDSA<ECP, SHA1>::PublicKey& key, const string& message, const string& signature)
{
    bool result = false;

    StringSource(signature + message, true,
        new SignatureVerificationFilter(
            ECDSA<ECP, SHA1>::Verifier(key),
            new ArraySink((byte*)&result, sizeof(result))
        ) // SignatureVerificationFilter
    );

    return result;
}
```


E:\Hoc tap\Ky2_nam2\CRYPTC
01ca46bc0ac04e09888a241ced9e290c.png invalid
5ecf653a589b3980a8a166e404069edc.png valid
62b7f87696d98b562540881d83fdba18.png valid
3449febf4cfb168d1002d84e0da97a56.png valid
6538fa0733f048ca52a95e3eb0eeefc2.png invalid
9557d3da1cb52dc7541d778f68c3730c.png invalid
b67dc736bd7c6818bb0885de0eba58b9.png invalid
b90b1a57cdff43c0721cb7afd955f10c.png invalid
d27a1a4ae33ab2dc6a510b4093c270ce.png valid
ff56449f20dffe819908ffd65bb7feab.png invalid
Press any key to continue . . .