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

BÁO CÁO THỰC HÀNH

Môn học: Mật mã học

Tên chủ đề: Thi cuối kì

GVHD: Tô Trọng Nghĩa

Nhóm: Mệt mỏi

1. **THÔNG TIN CHUNG:**

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: XXX

STT	Họ và tên	MSSV	Email
1	Nguyễn Thị Minh Châu	21520645	21520645@gm.uit.edu.vn

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**

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

Task 1:

1. 01 E0 E0 01 01 F1 F1 01 đây là một khóa có thể-yếu trong DES.

"01 E0 E0 01 01 F1 F1 01" có cấu trúc đơn giản và không ngẫu nhiên, không đáp ứng yêu cầu về tính an toàn của một khoá DES. Nó dễ bị tấn công bằng cách brute-force trong một thời gian ngắn.

Hơn hết trên NIST đã công bố đây là 1 khóa có thể-yếu

There are also 48 keys that produce only four distinct subkeys (instead of 16) - these are called possibly weak keys and should be avoided. These possibly weak keys are (in hex):

01011F1F01010E0E	1F1F01010E0E0101	E0E01F1FF1F10E0E
0101E0E00101F1F1	1F1FE0E00E0EF1F1	E0E0FEFEF1F1FEFE
0101FEFE0101FEFE	1F1FFEFE0E0EFEFE	E0FE011FF1FE010E
011F1F01010E0E01	1FE001FE0EF101FE	E0FE1F01F1FE0E01
011FE0FE010EF1FE	1FE0E01F0EF1F10E	E0FEFEE0F1FEFEF1
011FFEE0010EFEF1	1FE0FE010EF1FE01	FE0101FEFE0101FE
01E01FFE01F10EFE	1FFE01E00EFE01F1	FE011FE0FE010EF1
FE01E01FFE01F10E	1FFEE0010EFEF101	FE1F01E0FE0E01F1
01E0E00101F1F101	1FFEFE1F0EFEFE0E	FE1FE001FE0EF101

2. Vì là liên quan đến khóa yếu và khóa nửa yếu nên ta brute-force để có thể ra được bản rõ

```python
import codecs
from des import DesKey
from Crypto.Util.number import long_to_bytes
import base64
import binascii

test_key = ["011F011F010E010E", "1F011F010E010E01", "01E001E001F101F1",
"E001E001F101F101", "01FE01FE01FE01FE", "FE01FE01FE01FE01", "1FE01FE00EF10EF1",
"E01FE01FF10EF10E", "1FFE1FFE0EFE0EFE", "FE1FFE1FFE0EFE0E"]

for i in test_key:
    try:
        key = DesKey(bytes.fromhex(i))
        ct = "5NfkoSZz5it2H5UYlK6683xV0Vv6A4jldXR/WgAFFK0="
        pt = key.decrypt(base64.b64decode(ct)).hex()
        print("Key")
        print(i)
        print("Maybe plaintext")
        print(codecs.decode(pt, "hex").decode('utf-8'))
    except:
        print("Error")
```

Sau khi chạy chương trình ta được bản rõ như sau: đáp án bài tập 2 ở đâu

```
PS C:\Users\admin\Downloads\Lab02\Lab02> python .\test.py
Key
011F011F010E010E
Maybe plaintext
Error
Key
1F011F010E010E01
Maybe plaintext
đáp án bài tập 2 ở đâu
Key
```

Task 2:

3.

- Các trường hợp nên sử dụng mode CBC: Khi cần đảm bảo tính toàn vẹn của dữ liệu được mã hóa. Mode CBC sử dụng khối mã hóa trước đó để mã hóa khối tiếp theo, do đó nếu có bất kỳ thay đổi nào trong khối đã mã hóa trước đó thì sẽ ảnh hưởng đến khối tiếp theo. Điều này đảm bảo rằng bất kỳ sự thay đổi nào trong dữ liệu đầu vào đều sẽ được phát hiện. Khi muốn giảm thiểu tác động của lỗi truyền thông. Nếu một khối dữ liệu bị lỗi trong quá trình truyền thông, thì lỗi sẽ ảnh hưởng đến khối tiếp theo, nhưng không ảnh hưởng đến các khối khác.

Ví dụ: Mã hóa dữ liệu trước khi lưu trữ

- Các trường hợp nên sử dụng mode OFB: Khi cần mã hóa dữ liệu trực tiếp, chẳng hạn như mã hóa luồng video hoặc âm thanh. Mode OFB sử dụng một khối mã hóa để tạo ra một khối mã hóa tiếp theo, do đó nó cho phép mã hóa các khối dữ liệu lớn hơn mà không bị giới hạn bởi kích thước khối. Khi cần truyền dữ liệu qua kênh không đáng tin cậy hoặc không ổn định. Mode OFB không yêu cầu tính liên tục trong truyền thông, điều này cho phép việc tái truyền lại các khối dữ liệu bị mất mà không phải thực hiện lại toàn bộ quá trình mã hóa.

Ví dụ: Mã hóa trực tiếp các luồng âm thanh

4.

Task 3:

5. Ta viết chương trình như sau:

Open bai4.py ~/

```python
from Crypto.Util.number import inverse

# Các tham số của khóa công khai RSA
e = 6131
n = 10050256277

# Ciphertext đã cho
ciphertext = 283818407

# Số lần mã hoá
num_encryptions = 1373904

# tính giá trị phi euler của n ta được
phi_euler = 10050055776

# Tính plaintext ban đầu
plaintext = pow(ciphertext, inverse(e, num_encryptions * phi_euler), n)

# Chuyển đổi plaintext thành chuỗi ASCII
plaintext_ascii = bytearray.fromhex(hex(plaintext)[2:]).decode()

# In ra plaintext ban đầu
print("Plaintext ban đầu:", plaintext_ascii)
```

Ta được bản rõ như sau:

```
zeri@zeri:~$ gedit bai4.py
zeri@zeri:~$ python3 bai4.py
Plaintext ban đầu: rsa!
```

Sử dụng công thức giải mã RSA: plaintext = ciphertext^e mod n. Nhưng vì mã hoá 1.373.904 lần có thể mất rất nhiều thời gian và tài nguyên tính toán. Việc trực tiếp giải mã ciphertext ban đầu có thể không khả thi trong thực tế. Việc giải mã nhiều lần như vậy thường được thực hiện thông qua việc sử dụng phép tính modular exponentiation

và các tối ưu hóa hiệu suất. Ở bài trên ta tính được hàm phi euler của n = 10050256277 là 10050055776.

6. Bài này vì số quá lớn nên em dùng tool RsaCryptoTool để giải vì chạy code không được.

```
┌──(kali㉿kali)-[~/RsaCtfTool]
└─$ python RsaCtfTool.py -n 57896403840314051987261759673337301366683734816998649920480488791205037879705769545016401892067274501332754787735709240226460179134533746561124235517179498295171697331129995126899865863088633916897685608908987976914507875189904879873199730484482915025729856457329138265637959640132093788945226056400799524208136486484813222494670203894269801413153000353276965037773484695251691911683336749109096476174639689543664317155369816484713225696210718554897742216981768272441139909371302707957322928484223834102457933570350169163062844917787632800076243675198238619204874867257741542875604775505086216735298641732617297575839454718597707783770392921256680691460681995582837210084434524120116402000585119015392499578887902625052278393991723077343354546659547886750492033460654605387186960699022930017839563525035413906421121388855450067594744410130937403782435622424889028237939378282601331319063299896551441464515996825953246797022412850746228168549415166861527349532615309328883089773489323284100542174271240620552204793183553900405571505763280976616194561791528389644269649277890548552383494952658298809118597295976732683956605251149748326086335896711096041016068164324925760839626711076330252150389071234547023956664129650708853527300567775320176159290286675311463338661354397664415174980323399114107119899872088159938496751422867570247132208074331954604219218750000
```

Sau khi chạy ta được kết quả như sau:

```
Results for /tmp/tmppyxv83s1:

Unciphered data :
HEX : 0x727361207769746820776974686f7574206e2e
INT (big endian) : 25523359160595528645142676341914770184538967 50
INT (little endian) : 10354276744208496283940863155319886947078275 70
utf-8 : rsa with without n.
STR : b'rsa with without n.'
```

7.