

BÁO CÁO THỰC HÀNH

Môn học: Mật mã học

Tên chủ đề: Thi cuối kỳ

GVHD: Tô Trọng Nghĩa

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT219.N21.ANTT

STT	Họ và tên	MSSV	Email
1	Trần Minh Duy	21522010	21522010@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Nội dung	Tình trạng	Trang
1	Bài tập 1	0%	
2	Bài tập 2	100%	
3	Bài tập 3	100%	
4	Bài tập 4	0%	
5	Bài tập 5	100%	
6	Bài tập 6	100%	
7	Bài tập 7	0%	
8	Bài tập 8	100%	
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

Bài tập 1:

FE FE 1F 1F FE FE 0E 0E là khóa yếu - possibly weak keys

Theo <http://websites.umich.edu/~x509/ssleay/des-weak.html> trong quá trình generate 16 subkey thì key trên chỉ tạo ra 4 subkey khác nhau.

⇒ Dùng key này để mã hóa một đoạn text bất kỳ sẽ thỏa mãn yêu cầu đề bài

Bài tập 2: (1đ) Cho chương trình mã hóa DES bên dưới. Kết quả mã hóa được encode base64. Biết DES mã hóa mode ECB và khoá có liên quan đến khoá yếu và khoá nửa yếu. Hãy tìm ra khoá mà mã hóa 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

Danh sách khóa yếu và khóa nửa yếu tìm thấy trong tài liệu NIST đưa ra [Draft SP 800-67 Rev. 2, Recommendation for Triple Data Encryption Algorithm \(TDEA\) Block Cipher \(nist.gov\)](#):

3.3.2 Weak Keys

There are a few keys that are considered weak for the TDEA cryptographic engine. The use of weak keys can reduce the effective security afforded by TDEA and should be avoided. Keys that are considered weak are (in hexadecimal format):

- 01010101 01010101
- FEFEEFEFE FEFEEFEFE
- E0E0E0E0 F1F1F1F1
- 1F1F1F1F 0E0E0E0E

Note that the weak keys listed above and the semi-weak keys and the possibly weak keys listed below are expressed with odd parity, which is indicated in the rightmost bit of each byte.

Some pairs of keys encrypt plaintext to identical ciphertext and also should be avoided. These semi-weak keys are (in hexadecimal format):

- 011F011F010E010E and 1F011F010E010E01
- 01E001E001F101F1 and E001E001F101F101
- 01FE01FE01FE01FE and FE01FE01FE01FE01
- 1FE01FE00EF10EF1 and E01FE01FF10EF10E
- 1FFE1FFE0EFE0EFE and FE1FFE1FFE0EFE0E
- EOFEE0FEF1FEF1FE and FEE0FEE0FEF1FEF1

Thử sử dụng những khóa trên để giải mã cipher text đề đưa ra,... khi đó ta tìm được khóa giải mã là “**E001E001F101F101**”

Kết quả chạy chương trình giải mã DES mode ECB với cipher text ở hình dưới:

```
[2] 1 !pip install pycryptodome
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pycryptodome
  Downloading pycryptodome-3.18.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
  ━━━━━━━━━━━━━━━━ 2.1/2.1 MB 22.2 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.18.0

▶ 1 from Crypto.Cipher import DES
2 import base64
3
4 key = bytes.fromhex('E001E001F101F101') # khóa bí mật
5 mode = DES.MODE_ECB # mode mã hóa DES
6 iv = None # không sử dụng vector khởi tạo (IV) cho ECB mode
7
8 ciphertext_base64 = "6MupHn98v/yhX3jSCMF+LFOVQc7iRLALzTjd5ow34a5vnoPkSmZ1MHG/wU9Elkva"
9 ciphertext = base64.b64decode(ciphertext_base64)
10
11 cipher = DES.new(key, mode)
12 plaintext = cipher.decrypt(ciphertext)
13
14 print(plaintext.decode())
semi weak key des - khoá nữa yếu trong des
```

Có thể thấy plain text được giải mã ra giống với plaintext ban đầu đề đưa ra, nhưng có thêm những ký tự padding ở cuối chuỗi.

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) và Mode OFB (Output Feedback) là hai trong số nhiều chế độ hoạt động khác nhau của thuật toán mã hoá AES (Advanced Encryption Standard). Cả hai chế độ này đều có những ưu điểm và hạn chế riêng, và tùy thuộc vào yêu cầu bảo mật của ứng dụng cụ thể, một trong hai chế độ có thể phù hợp hơn.

1. Sử dụng mode CBC:

- CBC là một trong những chế độ hoạt động phổ biến nhất của AES.
- CBC phù hợp khi cần mã hóa các thông tin có kích thước lớn, ví dụ như tệp tin hoặc dữ liệu đang truyền trong mạng.
- CBC sử dụng việc kết hợp các khối thông tin trước đó để mã hóa khối tiếp theo, tạo ra tính chất đồng bộ hóa và ngăn chặn việc thay đổi dữ liệu trong truyền thông.
- CBC cũng có thể bảo vệ khỏi các cuộc tấn công phá mã khối đơn lẻ, vì việc thay đổi một khối thông tin sẽ ảnh hưởng đến các khối tiếp theo.

- Ví dụ: Một công ty muốn bảo vệ các tệp tin quan trọng của mình trên máy chủ lưu trữ, thì họ có thể sử dụng mode CBC để mã hóa các tệp tin này.

2. Sử dụng mode OFB:

- OFB phù hợp khi cần mã hóa dữ liệu trực tiếp, ví dụ như các luồng dữ liệu trực tuyến, nhưng không cần đồng bộ hóa.
- OFB tạo ra một chuỗi ngẫu nhiên từ khóa và IV, sau đó mã hóa các khối thông tin bằng cách XOR với chuỗi này.
- OFB có thể bảo vệ khỏi các cuộc tấn công phá mã bằng cách sử dụng một sê-ri khóa mật mã hóa dữ liệu thay vì sử dụng khóa mật mã duy nhất.
- Ví dụ: Một ứng dụng web muốn bảo vệ dữ liệu đăng nhập của người dùng khi truyền từ trình duyệt đến máy chủ, thì họ có thể sử dụng mode OFB để mã hóa dữ liệu này.

Tóm lại, CBC và OFB đều có những ưu điểm và hạn chế riêng, và tùy thuộc vào yêu cầu bảo mật của ứng dụng cụ thể, một trong hai chế độ có thể phù hợp hơn.

Bài tập 4:

~~Ta thấy, đối với mã hóa bằng ECB thì cần vector IV, còn giải mã bằng CBC thì không cần vector IV.~~

~~Ta có công thức như sau:~~

~~+ Đối với block đầu tiên: $P_2[1] = \text{Key} \oplus C[1] = \text{Key} \oplus P_1[1] \oplus IV$~~

~~+ Đối với các block tiếp: $P[i] = \text{Key} \oplus C[i] = \text{Key} \oplus P_1[i] \oplus P_1[i-1]$~~

~~Từ đó, ta tính như sau:~~

~~+ $P_1[1] = \text{Key} \oplus P_2[1] \oplus IV$~~

~~+ $P_1[i] = \text{Key} \oplus P_2[i] \oplus P_1[i-1]$~~

Gọi Plaintext, plaintext 2, cipher lần lượt là P, Q, C

$$C[i] = E(P[i] \wedge C[i-1], \text{key}) \quad \text{và} \quad C[1] = E(P[1] \wedge iv, \text{key})$$

$$Q[i] = D(C[i], \text{key}) = P[i] \wedge C[i-1] \quad \text{và} \quad Q[1] = P[1] \wedge iv$$

$$\begin{aligned} Q &= Q[1]Q[2]\dots Q[n] \\ &= (P[1] \wedge iv)(P[2] \wedge C[1])\dots(P[n] \wedge C[n-1]) \\ &= P[1]P[2]\dots P[n] \wedge iv.C[1]C[2]\dots C[n-1] \\ &= P \wedge iv.C[1]C[2]\dots C[n-1] \end{aligned}$$

$$\Rightarrow P = Q \wedge iv.C[1]C[2]\dots C[n-1]$$

```

1 import base64
2 import binascii
3
4 def xor_strings(string1, string2):
5     str1 = int(string1, 16)
6     str2 = int(string2, 16)
7     xor_result = hex(str1 ^ str2)
8     result = str(xor_result)[2:]
9     result = bytes.fromhex(result)
10    print(result.decode("utf-8"))
11
12 def base64_to_hex(base64_string):
13     decoded_bytes = base64.b64decode(base64_string)
14     hex_string = binascii.hexlify(decoded_bytes).decode()
15     return hex_string
16
17 def concat_base64_strings(base64_string1, base64_string2):
18     bytes1 = base64.b64decode(base64_string1)
19     bytes2 = base64.b64decode(base64_string2)
20     concatenated_bytes = bytes1 + bytes2
21     concatenated_base64 = base64.b64encode(concatenated_bytes).decode()
22     return concatenated_base64
23
24 cipher = "XEkdm4AGP6oQK0OfKhN7a/fTBb+XyUMLKOHAKoPw7wUzrcbuJnj40jaCfVViLhmlU1V/dXIltUF8dsAhC8LDBIIBhvKzDucvYRwg1l"
25 plain2 = "hU+6vyyXZhQtXk+dtQ//OyopjjjJkvC8ZDy/Ap7EopMVGuF46GAKUaG4Fv2t88wE86uD53Rm/JYRv47BbQa45D4ER1SXbhXDeKR"
26 iv = "yIwZn0RUXvlw9lxtvhCuQg=="
27
28 string1 = base64_to_hex(iv) + base64_to_hex(cipher)
29 string2 = base64_to_hex(plain2)
30 print(string1)
31 print(string2)
32
33 ciphers = ['c88c199f44545ef970f65c6d5610ae3a5c491d9b80063faa102b439f2a137b6bf7d305bf97c9432528e1c02a83d6ef0533a'
34 plain2s = ['854fbabf2c97ed9850b7193e76d43ffceca8a638e3264bc2f190f2fc0a7b128a4c546b9fe3a180914686e05bf6b7cf3013c'
35
36 for str1, str2 in zip(ciphers, plain2s):
37     xor_strings(str1, str2)
38

```

```

PS C:\Users\ADMIN\Downloads> wsl
duy@LAPTOP-HRE30JQL:/mnt/c/Users/ADMIN/Downloads$ python3 bai4.py
c88c199f44545ef970f65c6d5610ae3a5c491d9b80063faa102b439f2a137b6bf7d305bf97c9432528e1c02a83d6ef05
33adc6ee2678f8d236827d556294798b53557f757225b5417c76c0210bc2c304820186f2b30ee72f611c20d47c17f730
30c6679122af6a16cebe7a2f34b00cd9ec7418b2177e93a713de7ec2512534497ea68eec557a2cb4f6d06919aea256
8a117133b96001f92b96a9aa01fabbb951fd54f9821fa23c3ef7dd0c0fb46c4b755c2e28b521cc48373d5f97c3a14b3bf
7aa0fa47b897803402cdf5b3d57613e545ade6649f6db965c95d73e5deb9e063ca0e2ecb44b0265b5ea18bc5fed5775
854fbabf2c97ed9850b7193e76d43ffceca8a638e3264bc2f190f2fc0a7b128a4c546b9fe3a180914686e05bf6b7cf30
13ceae0f9dd19bf25846fe3b05b41ae390f8111d524976e15c37a44559adb66ae64ae38b9f2eb45a035e59a01964db10
63ae0ef756fd0561bd925a625dc84fb1f2b22ce552579ff9d11dac895c572b34f6841b87aa398ceb01a985fcba89cab7
31b81213d7a482974cb6c769a1839b518e13ff799a5940e39c9c6a6f8b66bc56ef7d92ab968d2538f0f58d14dbaf02dc
5ac893a60310ee1460ac8f39fbe78085cf39fe058f57fb4bd2f745073e89d05
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. 0.0.0
duy@LAPTOP-HRE30JQL:/mnt/c/Users/ADMIN/Downloads$ |

```

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

e = 65537

n =

194326416639702286024546132569291140934764160534429254756332542

456564420846769802418264203845642726291345024973700618220765757

471114631732011485426550453162783980903850682051719504365819551

378695494461263418178347957477525149752756829799458483358080448

```
204640904000722558467129575482132114314019712648893397848979553  
777371943002840069941912337874408764634203036294522740995774928  
359350793610756535979599054763670020913479914224493181176117375  
077425641188006017981734325511332998105838358491786509679417739  
097282284112315123568770467663376955215760444696892551143067709  
09991892251764320328930944573509467143490908803761  
ciphertext =  
855170488574088974430975503298702010186351242562518088813862740  
360716130495382273899339168052034526559264504306866252070816936  
692978169116991838127336217048867739226341396910883292327848223  
658682742972787562894905848727060265815810169422220948937093367  
388625828062022515347678541096638536893106420441094541551906163  
528110226903722244634683086789109616809890122371823675206906129  
120347833372851603600611649241830844379800492872829723685394001  
060700097508405960942046873646269753641742245602862931302228779  
149091138431649442359046234269522834405279144780425770698113288  
2707663376242387425202256065291974952961394903512
```

Hãy tìm lại plaintext ban đầu (ascii). Biết được sau khi mã hóa thêm $2^{1025}-2$ lần thì trả lại về ciphertext.

Vì ciphertext mã hóa thêm $2^{1025}-2$ lần về lại ciphertext, nên để có plaintext ta giảm số lần mã hóa đi một thì sẽ được plaintext, tức là mã hóa $2^{1025}-3$ lần hay nói rõ: Ciphertext mã hóa $2^{1025}-3$ lần được plaintext.

Sau đó ta thực hiện các phép toán và đặt ra nhiều giả sử..., nói chung khá khó, tham khảo 2 bài bên dưới có cách giải tương tự.

Bài writeup:

<https://ur4ndom.dev/posts/2022-07-04-gctf-cycling/>
<https://blog.maple3142.net/2022/07/04/google-ctf-2022-writeups/>

Hình dưới là code cùng kết quả:

```

[37] %pip install gmpy2
      %pip install pycryptodome
      %pip install requests

Requirement already satisfied: gmpy2 in /usr/local/lib/python3.10/dist-packages (2.1.5)
Requirement already satisfied: pycryptodome in /usr/local/lib/python3.10/dist-packages (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2023.7.22)

import requests

url = 'http://factordb.com/api?query=2^1025-2'
response = requests.get(url)
with open('fact.json', 'wb') as file:
    file.write(response.content)

import json
from itertools import product
import gmpy2
from tqdm import tqdm
from Crypto.Util.number import isPrime, long_to_bytes
e = 65537
n = 1943264166397022860245461325692911409347641605344295475633254245656442084676980241826420384564272629134502497370061822076575747
ct = 8551704885740889744309755032987020101863512425625180888138627403607161304953822738993391680520345265592645043068662520708169366
k = 2**1025 - 3

with open("fact.json", "rb") as f:
    # curl 'http://factordb.com/api?query=2^1025-2' -o fact.json
    fact = json.load(f)
prime = [gmpy2.mpz(p) for p, _ in fact["factors"]]
print(prime)
lamda = 1
for filter in tqdm(product([0, 1], repeat=len(prime)), total=2 ** len(prime)):
    multi = 1
    for f, p in zip(filter, prime):
        if f:
            multi *= p
    if isPrime(multi + 1):
        lamda *= multi + 1
d = gmpy2.invert(e, lamda)
pt = gmpy2.powmod(ct, d, n)
print(long_to_bytes(int(pt)).decode())
# https://urandom.dev/posts/2022-07-04-gctf-cycling/

[mpz(2), mpz(3), mpz(5), mpz(17), mpz(257), mpz(641), mpz(65537), mpz(274177), mpz(2424833), mpz(6700417), mpz(67280421310721), mpz(100%|██████████| 131072/131072 [00:21<00:00, 6112.05it/s]
https://urandom.dev/posts/2022-07-04-gctf-cycling/

```

Bài tập 6: (1đ) Luyện tập sử dụng RSA với các thông số sau:

Cho $n =$

435227995514595233481777875128553267789518490019706910041264522
 576167370450264035329257425687482934218288024104078760071937867
 729665165419232294862039037539532541018791091866554026890772575
 542500150368974003244421620283419245914341287587224850385960568
 423695648149349495978867307820806527640140691505076087374609353
 298143855991989318319669872124242201980703435385615498797284322
 677624835318103731393994098910886917562724347695067095669231136
 720031684292021310133061079843385444572080810544948629628974181
 777140491860399469644232450319213975427311559751291837603257905
 925943747171923743698060317530792349350565355167829754335569445
 46479288928119986905095165006275220727008443208488661576814628



312460127436112433665026888094684183767029613461887272581286948
 495000989351862729355214783631666092155446521437980264068221581
 550496355873858463013338063976581480921483114034017598115462382
 106048201140451953055975911628100841560325826387331926991422032
 779045330472217987711669298126827082999696913345514967480677833
 463901712977749379374425621167007520596686819348591909364154970
 947609912147460887228457333276376257510336977082403345074258916
 164127134850307837982021563565563407259490768379096274939487599
 712954658915228217488835923905955229

$e = 13$

cipher text =
 476571097361060712805283430610657498271936961456822634648857175
 380529608803901802760993465949620967958626141563851005820592167
 730085060290419632039841105730979318019479078715201753887446756
 062404554684911770626043180714027744617745344458698435421380020
 840817358604995463436341736272836579615685301783159150858520879
 420080221905331503395347356296036476615978362646961605585534974
 049093004914633532819880643142080759567584600177170720817851983
 702541917483916363432538052549112656589320428839171031224774297
 390078158400291697438394384804913953729256484171251651400462750
 974928602074326662338107961433661604958026091208405652573460412
 991510712318847973519565492089103801447814243199399268480111326
 23919976835350521437383319982906148216551195427258610736723328
 68039866593082940424090185093722845938558671289607841

Ta có: $\text{plaintext}^e \bmod n = \text{ciphertext}$

Quan sát giá trị được đưa ra, giá trị của n rất lớn trong khi đó so với nó giá trị ciphertext có vẻ ngắn hơn rất nhiều, e cũng quá nhỏ, điều này tạo nên khả năng lớn

$\text{plaintext}^e = \text{ciphertext} < n$

Ta thử trường hợp này:

```
[6] 1 !pip install gmpy2
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting gmpy2
  Downloading gmpy2-2.1.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.7 MB)
    1.7/1.7 MB 20.9 MB/s eta 0:00:00
Installing collected packages: gmpy2
Successfully installed gmpy2-2.1.5

1 from gmpy2 import iroot
2 n = 43522799551459523348177787512855326778951849001970691004126452257616737045026403532925742568748293421828802410407876007193786772966516
3 e = 13
4 cipher = 4765710973610607128052834306106574982719369614568226346488571753805296088039018027609934659496209679586261415638510058205921677306
5 print(bytes.fromhex(hex(iroot(cipher, e)[0])[2:]))

b'wao. m^e < n is so bad !!!'
```

wao. $m^e < n$ is so bad !!!

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 kèm nào là đúng với tập tin ban đầu.(có 10 signature cần kiểm tra)

Trong file code cho sẵn có nhiều hàm hữu ích, trong đó có cả hàm VerifyMessage(), ta sẽ sửa lại file code này để sử dụng.

```

Y: 554062904348962958402079106242929185804054838061.

Modulus:
1461501637330902918203684832716283019653785059327.

Coefficient A:
1461501637330902918203684832716283019653785059324.

Coefficient B:
163235791306168110546604919403271579530548345413.

Base Point:
X: 425826231723888350446541592701409065913635568770.
Y: 203520114162904107873991457957346892027982641970.

Subgroup Order:
1461501637330902918203687197606826779884643492439.

Cofactor:
1.
1.png: 0
2.png: 1
3.png: 1
4.png: 1
5.png: 0
6.png: 0
7.png: 0
8.png: 0
9.png: 1
10.png: 0

```

Hình bên trên là kết quả sau quá trình thử nhiều lần.... haizzz nó khá mệt đấy...

Đầu tiên ta dễ thấy signature nằm ở cuối mỗi file ảnh dựa vào file code ban đầu, ta sẽ tách nó ra, nhưng đầu tiên cần viết thêm một hàm loadFile để load file ở dạng nhị phân.

```

void loadFile(const std::string& filename, string& mess) {
    FileSource file(filename.c_str(), true, new StringSink(mess));
    file.PumpAll();
}

```

Vấn đề là signature sẽ chiếm bao nhiêu byte, có lẽ có thể dựa vào modulus của publicKey để suy ra, hoặc trực tiếp hơn là đoán khi so sánh phần tail của các file ảnh.

Lab 04: Thi cuối kỳ

```
PS D:\C++\TestC++1\TestC++1> wsl
duy@LAPTOP-HRE30JQL:/mnt/d/C++/TestC++1/TestC++1$ xxd 1.png | tail
0000b010: 8efb f59b 1e52 904e 9049 539e ce93 8997 ....R.N.IS....
0000b020: a87e 4c6d 8365 e1f5 7a74 5b93 0eba 95cf .~Lm.e..zt[....
0000b030: c4c7 7e93 8e57 96cb 095c bb46 181d 254c ...~.W...\.F..%L
0000b040: 4d59 9103 b35b 3ea3 525c 3f71 cfca f46e MY...[>.R\?q...n
0000b050: 1ab0 51ba 35a5 6e07 bb57 66a2 19d5 779c ..Q.5.n..Wf...w.
0000b060: 9cc1 09a8 db1f 7090 efa0 9efe 05e6 ee45 .....p.....E
0000b070: 9ed7 372f 2000 0000 0049 454e 44ae 4260 ..7/ ....IEND.B'
0000b080: 8200 cd27 786e cae5 0dc9 ab4f 0ff5 1d75 ...'xn....O...u
0000b090: da8d 3848 2dea 00f1 5c5a 7636 9e11 093a ..8H-...\Zv6...:
0000b0a0: efd6 2caa a203 819e cb05 c1 .....S...
duy@LAPTOP-HRE30JQL:/mnt/d/C++/TestC++1/TestC++1$ xxd 2.png | tail
0000b010: 8efb f59b 1e52 904e 9049 539e ce93 8997 ....R.N.IS....
0000b020: a87e 4c6d 8365 e1f5 7a74 5b93 0eba 95cf .~Lm.e..zt[....
0000b030: c4c7 7e93 8e57 96cb 095c bb46 181d 254c ...~.W...\.F..%L
0000b040: 4d59 9103 b35b 3ea3 525c 3f71 cfca f46e MY...[>.R\?q...n
0000b050: 1ab0 51ba 35a5 6e07 bb57 66a2 19d5 779c ..Q.5.n..Wf...w.
0000b060: 9cc1 09a8 db1f 7090 efa0 9efe 05e6 ee45 .....p.....E
0000b070: 9ed7 372f 2000 0000 0049 454e 44ae 4260 ..7/ ....IEND.B'
0000b080: 8200 5510 e7b4 70ca 36b8 e9d2 d978 e8f9 ..U...p.6....x..
0000b090: 70c8 677a 82cf 00b1 5069 afb6 d265 4066 p.gz....Pi...e@f
0000b0a0: 5387 80e7 8bff c1f6 5633 11 S.....V3.
duy@LAPTOP-HRE30JQL:/mnt/d/C++/TestC++1/TestC++1$ xxd 3.png | tail
0000b010: 8efb f59b 1e52 904e 9049 539e ce93 8997 ....R.N.IS....
0000b020: a87e 4c6d 8365 e1f5 7a74 5b93 0eba 95cf .~Lm.e..zt[....
0000b030: c4c7 7e93 8e57 96cb 095c bb46 181d 254c ...~.W...\.F..%L
0000b040: 4d59 9103 b35b 3ea3 525c 3f71 cfca f46e MY...[>.R\?q...n
0000b050: 1ab0 51ba 35a5 6e07 bb57 66a2 19d5 779c ..Q.5.n..Wf...w.
0000b060: 9cc1 09a8 db1f 7090 efa0 9efe 05e6 ee45 .....p.....E
0000b070: 9ed7 372f 2000 0000 0049 454e 44ae 4260 ..7/ ....IEND.B'
0000b080: 8200 4a82 6b2f 85e0 f106 1c1e 8911 3275 ..J.k/.....2u
0000b090: a23f 8ded 19ce 00a4 5c4a 05d2 f21d ca0f ..?.....\J.....
0000b0a0: 5c0e f68f 9ef4 cc95 5c18 30 \.....\.\0
duy@LAPTOP-HRE30JQL:/mnt/d/C++/TestC++1/TestC++1$ |
```

À thì dựa vào hình trên tôi đã đoán signature có 41 byte và tôi không nghĩ nó sai, (nhưng thật ra nó sai , thực chất nó là 42 byte, tốn khá nhiều thời gian để tôi nhận ra điều đó)

Đây là hàm main khi tôi có gắng khôi phục giá trị message cũng như tách đúng giá trị signature lưu vào một biến string

```
int main(int argc, char* argv[])
{
    // Scratch result
    bool result = false;

    // Private and Public keys
    ECDSA<ECP, SHA256>::Publickey publicKey;

    // Load key in PKCS#9 and X.509 format
    LoadPublicKey("ec.public.key", publicKey);
    PrintPublicKey(publicKey);
    PrintDomainParameters(publicKey);

    // Sign and Verify a message
    for (int i = 1; i < 11; ++i)
    {
```

```

        string file_name = std::to_string(i) + ".png";
        string content;
        loadFile(file_name, content);

        string signature = content.substr(content.length() - 42);
        string origin_pic = content.substr(0, content.length() - 42);

        string message;
        StringSource ss(origin_pic, true, new Base64Encoder(new
StringSink(message)));
        ss.PumpAll();
        result = VerifyMessage(publicKey, message, signature);
        cout << file_name << ":" << result << endl;
    }
    return 0;
}

```

Và rồi ta có kết quả là 4/10 hình có signature được verify (sao nhiều vậy nhỉ, tôi nghĩ rằng chỉ nên có 1 thôi, nhưng mà kê, tôi không muốn quan tâm nữa)

Bài tập 8:(2đ) Luyện tập về hàm băm. Tìm giá trị xung đột băm (hash collision) của chương trình trong baitap_8.py

```

def cryptohash(msg):
    initial_state = xor(Y_bytes, Z_bytes)

    msg_padded = pad(msg)
    msg_blocks = blocks(msg_padded)

    for i,b in enumerate(msg_blocks):
        mix_in = scramble_block(b)
        for _ in range(i):
            mix_in = rotate_right(mix_in, i+11)
            mix_in = xor(mix_in, X_bytes)
            mix_in = rotate_left(mix_in, i+6)
        if (i % 2 != 0):
            mix_in = xor(mix_in, rotate_right(mix_in, 12))
        initial_state = xor(initial_state,mix_in)
        initial_state = scramble_block(initial_state)

    return initial_state.hex()

def main():
    m1 = input("Please enter a hex encoded messages m1:\n")
    m2 = input("Enter a hex encoded messages m2:\n")
    print("----")
    print("hash m1:",cryptohash(bytes.fromhex(m1)))
    print("hash m2:",cryptohash(bytes.fromhex(m2)))
    print("----")

    if cryptohash(bytes.fromhex(m1)) == cryptohash(bytes.fromhex(m2)):
        if m1 != m2:
            print(f'Congratulations on finding the collision value {cryptohash(bytes.fromhex(m1))}')

```

Trong chương trình, ta tập trung phân tích hàm **cryptohash**, hàm này sẽ xử lý từng block riêng của msg, từ block tạo **mix_in** sau đó xor **initial_state** với **mix_in**:

block -> scr -> scr2(i) i times -> scr3 if I odd -----> xor init and scramble

scr: hàm scramble()

scr2: chỉ vòng lặp i lần với i là index

scr3: chỉ phép xor dịch phải 12 nếu i là số lẻ

Hàm *scr* có thể đơn giản hóa bằng cách tương đương với 1 lần dịch byte k và xor Y, với k, Y cố định và có thể tính được giá trị cụ thể...

$$scr(b) = b \rightarrow k \wedge Y$$

Hàm *scr2* cũng có thể làm tương tự với điều kiện *i* bằng một giá trị cố định

$$scr2(b_i) = b_i \rightarrow k_i \wedge Y_i$$

Với các block có *i* thỏa điều kiện (*i % 2 != 0*), tức ở vị trí lẻ, hàm sẽ thực hiện thêm một phép xor: **mix_in = xor(mix_in, rotate_right(mix_in, 12))**

Ở đây cho đơn giản ta giả sử tất cả các byte trong một block đều giống nhau, khi đó sẽ xuất hiện một số tính chất đặc biệt...

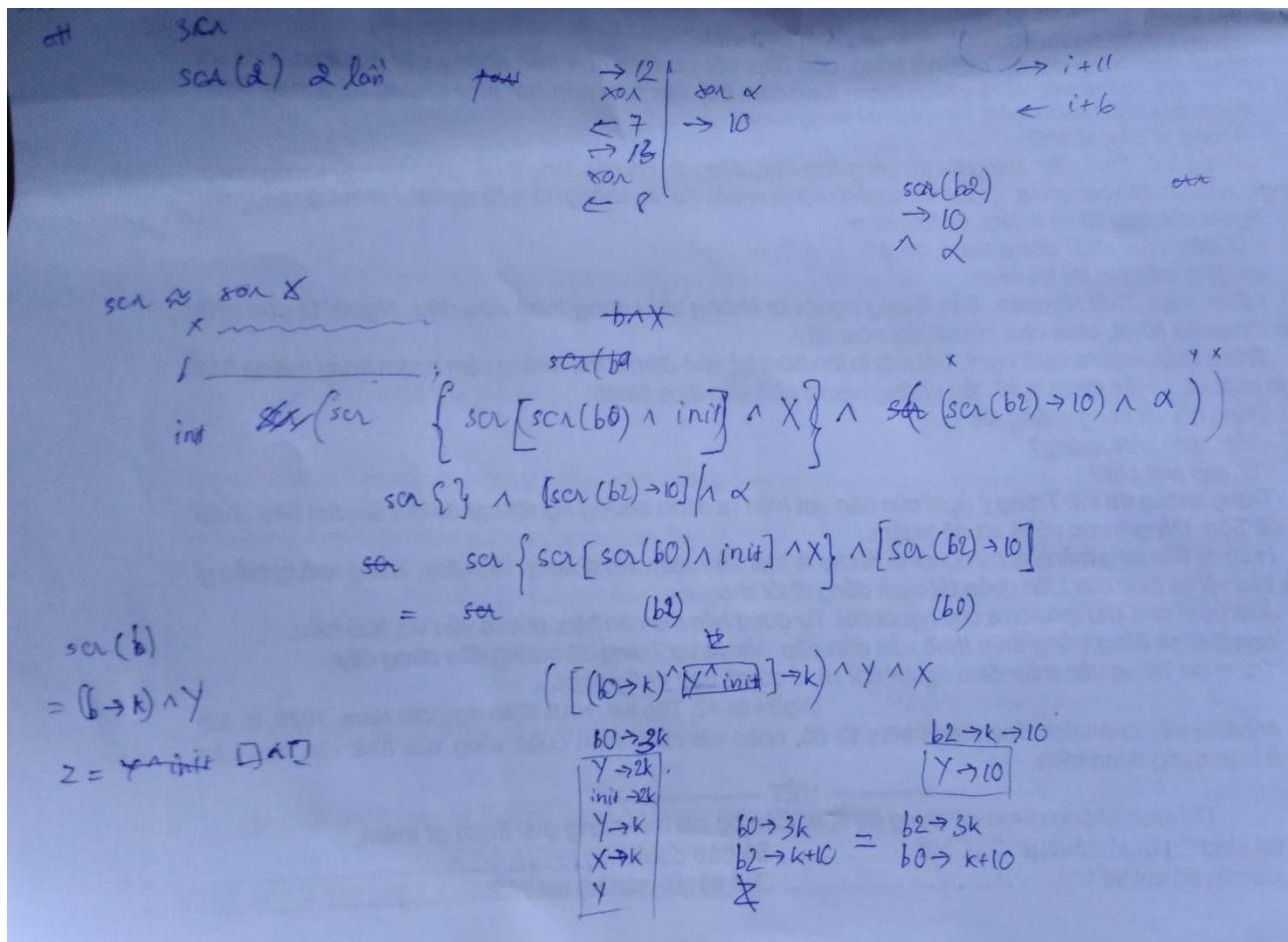
Ví dụ như các block ở vị trí lẻ sẽ trả về **mix_in= xor(mix_in, rotate_right(mix_in, 12))** giống nhau do các byte của block triệt tiêu nhau khi xor phần dịch byte của chính nó (vì tất cả các byte trong block giống nhau nên khi dịch byte bao nhiêu đều như nhau)

Giải thích: mix_in = scr2(scr(b1))

$$\begin{aligned} &= scr2(b1 \rightarrow k \wedge Y) \\ &= (b1 \rightarrow k \wedge Y) \rightarrow k1 \wedge Y1 \\ &= b1 \rightarrow k \rightarrow k1 \wedge Y \rightarrow k1 \wedge Y1 \\ &= b1 \rightarrow n \wedge Z \end{aligned}$$

mix_in = scr3(mix_in) = xor(mix_in, rotate_right(mix_in, 12))

$$\begin{aligned} &= mix_in \wedge mix_in \rightarrow 12 \\ &= b1 \rightarrow n \wedge Z \wedge (b1 \rightarrow n \wedge Z) \rightarrow 12 \\ &= b1 \rightarrow n \wedge Z \wedge b1 \rightarrow n \rightarrow 12 \wedge Z \rightarrow 12 \\ &= b1 \wedge Z \wedge b1 \wedge Z \rightarrow 12 \\ &= Z \wedge Z \rightarrow 12 \\ &= X \end{aligned}$$



Hình 1. Bản nháp của tôi :> chắc chả ai coi hiểu đâu hehe...

Sau đó ta tạo ra trường hợp tổng quát cho message 1, 2, 3, 4, 5... block thì sẽ thấy được kết quả cuối cùng là xor của rất nhiều thành phần, trong đó (vì tính chất đặc biệt của trường hợp giả sử) chỉ có biến số là các block chẵn.

Có thể tạo 2 msg cho kết quả hàm **cryptohash** giống nhau tạo các block có byte giống nhau, byte block lẻ chọn tùy ý, byte block chẵn chọn sao cho xor tất cả lại cho kết quả giống nhau.

```
def main():
    m1 = '04'*32 + '12'*32 + '8f'*32 + '34'*32 + '70'*32
    m2 = '5a'*32 + '56'*32 + 'f5'*32 + '78'*32 + '54'*32

    print(m1)
    print("----")
    print(m2)
    print("----")
    # cryptohash(bytes.fromhex(m1))
    # print("----")
    # cryptohash(bytes.fromhex(m2))
    # print("----")

    if cryptohash(bytes.fromhex(m1)) == cryptohash(bytes.fromhex(m2)):
        if m1 != m2:
            print(f'Congratulations on finding the collision value {cryptohash(bytes.fromhex(m1))}')

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

```
PS C:\Users\ADMIN\OneDrive\Tài liệu\123-OFFICE\MMH\ThiTH> & C:/  
Users/ADMIN/AppData/Local/Programs/Python/Python311/python.exe  
"c:/Users/ADMIN/OneDrive/Tài liệu/123-OFFICE/MMH/ThiTH/dề.2/bai  
tap_8.py"  
04040404040404040404040404040404040404040404040404040404040404040404  
412121212121212121212121212121212121212121212121212121212121212121  
128f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8  
f8f343434343434343434343434343434343434343434343434343434343434343434  
343470707070707070707070707070707070707070707070707070707070707070707  
07070  
----  
5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5  
a5656565656565656565656565656565656565656565656565656565656565656565  
56f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f5f  
5f5f57878787878787878787878787878787878787878787878787878787878787878  
787854545454545454545454545454545454545454545454545454545454545454545  
45454  
----  
Congratulations on finding the collision value 5a9cea92234af8b9  
c6ffb0674887d2a7ed29ca374ef33a1cccaa1bdfe2c8137e  
PS C:\Users\ADMIN\OneDrive\Tài liệu\123-OFFICE\MMH\ThiTH>
```

HẾT