**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**GIẢNG VIÊN: TS.NGUYỄN NGỌC TỰ**

# TRẦN VỸ KHANG – 22520628

# NGUYỄN ĐẶNG NGUYÊN KHANG - 22520617

**BÁO CÁO CUỐI KÌ MẬT MÃ HỌC**

**ỨNG DỤNG CHỮ KÝ SỐ ĐỂ XÁC THỰC GIAO DỊCH TRÊN NỀN TẢNG E-COMMERCE**

**APPLICATION OF DIGITAL SIGNATURE FOR TRANSACTION AUTHENTICATION ON E-COMMERCE PLATFORM**

# 

# 

# 

# NGÀNH AN TOÀN THÔNG TIN

**Mục Lục**

**I . TỔNG QUAN ĐỀ TÀI ..................................................................................... 3.**

Chủ đề & Ngữ cảnh.............................................................................................

Yêu cầu về bảo mật & Giải pháp bảo mật ..........................................................

Các bên liên quan ................................................................................................

Thư viện, các nguồn tham khảo ..........................................................................

**II . SOLUTION....................................................................................................... 5.**

Solution Architecture........................................................................................

Mô tả chi tiết Architecture........................ 4 Module ....................................

Thực thi, vai trò các bên liên quan...................................................................

**III. KIẾN THỨC NỀN TẢNG............................................................................... 13.**

Keygen , complexities, sign , ffSampling ,verify.............................................

**IV. DEMO................................................................................................................ 18.**

Kết quả và Link video ghi lại quá trình thực thi .......................................

**V.WRITE-UP CTF ...........................CryptoHack.................................................. 22.**

Nguyễn Đặng Nguyên Khang ................................................................. **22.**

Trần Vỹ Khang ............................................................................................ **36.**

|  |  |
| --- | --- |
| **PHÂN CHIA CÔNG VIỆC** | |
| **Trần Vỹ Khang** | * **Code chương trình** * **viết phần thuật toán Falcon trong bài Final.** * **Góp ý và sửa đổi final** |
| **Nguyễn Đặng Nguyên Khang** | * **Viết bài final** * **code kết nối database Mongodb(up dữ liệu, load dữ liệu)** * **Góp ý và sửa đổi code** |

**I . Tổng quan đề tài**

**1. Chủ đề triển khai**

Ứng dụng chữ ký số để xác thực giao dịch trên nền tảng E-commerce

**2. Ngữ cảnh ứng dụng**

Trong nền tảng thương mại điện tử các chữ ký số dùng hệ mã hóa bất đối xứng(RSA,DSA,ECDSA,…) không thể chống lại khi kẻ tấn công sử dụng máy tính lượng tử.

* **Reliable Arguments for the Gaps:**

Các chữ ký số tạo ra từ  hệ mã hóa bất đối xứng (RSA ,ECDSA,…) không đáp ứng đủ tiêu chuẩn về độ an toàn của **NIST** :

+ RSA : Máy tính lượng tử sẽ tấn công bằng thuật toán Shor và thuật toán Grover .

+ ECDSA : Bị tấn công bởi thuật toán logarit rời rạc lượng tử.

**3. Motivations**

Những hợp đồng trên nền tảng thương mại điện tử như (đơn hàng, invoice, đơn thanh toán,...) nếu bị attacker dùng máy tính lượng tử làm giả có thể gây thiệt hại cho các bên liên quan trên nền tảng thương mại điện tử. Vì vậy việc tạo chữ ký số bằng thuật toán có thể chống lại máy tính lượng tử là thực sự cần thiết.

* **Desired Security Features:**
* Đảm bảo các hợp đồng trên nền tảng thương mại điện tử không bị chỉnh sửa. ( tính nguyên gốc của dữ liệu )
* Xác định được người đã ký ( nguồn gốc của chữ ký )

**4. Giải pháp bảo mật**

* Thư viện: Dùng thư viện chữ ký số Falcon
* Ngôn ngữ lập trình: C#, C
* Database dùng lưu trữ : MongoDB
* Nơi lưu trữ privateKey : BitLocker ( on local )

**5. Các bên liên quan**

1.Nền tảng thương mại điện tử,

2.Người bán

3.Người mua

4.Ngân hàng

5.Đơn vị vận chuyển

6.Attacker giả mạo và đánh cắp dữ liệu khi dùng máy tính lượng tử

**6. Các nguồn tham khảo:**

Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang. “Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU Specification v1.2”. Post-Quantum Cryptography Standardization Selected Algorithms 2022.

**II. SOLUTION**

1. **Solution Architecture**

**A diagram of a company

Description automatically generated**

**A diagram of a flowchart

Description automatically generated**

**A diagram of a company

Description automatically generated with medium confidence**

1. **Solution Detail Architecture**

**Module 1:**

A white square with black text

Description automatically generated

* Người mua thực hiện mua hàng trên sàn thương mại thì hệ thống bên phía người mua sẽ tạo ra đơn hàng cần thanh toán và sẽ dùng privatekey của người mua ký vào đơn hàng cần thanh toán , thông tin sẽ được cho sàn thương mại điện tử .
* Khi đó, bên phía người bán nhìn thấy thông tin sản phẩm mà người mua được hiển thị trên sàn và sẽ verify lại thông tin người mua bằng publickey người mua .

**Module 2:**

**A diagram of a e-commerce

Description automatically generated**

* Bên sàn thương mại điện tử sau khi người bán verify thành công thì sàn thương mại sẽ tạo ra invoice. Người bán ký bằng privatekey vào invoice, sau đó sàn thương mại điện tử gửi về cho người mua và người mua verify lại bằng publickey của người bán.
* Sau đó người mua sẽ được lựa chọn phương thức thanh toán là **thanh toán online (Module 3)** hoặc **thanh toán khi nhận hàng(Module 4).**

**Module 3 : Thanh toán online**

**A diagram of a flowchart

Description automatically generated**

**A diagram of a company

Description automatically generated with medium confidence**

* Thông tin đơn hàng của người mua sẽ được ký bằng privatekey của người mua và sẽ được gửi đến ngân hàng để ngân hàng verify lại bằng publickey của người mua.
* Khi đã xác nhận thành công , ngân hàng ký bằng privatekey của ngân hàng vào đơn thanh toán rồi gửi đến sàn thương mại điện tử để lưu trữ kèm với số tiền cần thanh toán, khi tất cả các bước xác minh hoàn tất , người bán thấy được thông tin trên sàn và tiến hành tạo vận đơn và ký bằng private gửi cho sàn thương mại , sàn thực hiện verify lại bằng publickey của người bán .
* Khi verify thành công , sàn thương mại ký vào thông tin vận đơn và gửi cho đơn vị vận chuyển , đơn vị vận chuyển tiến hành verify lại chữ ký số của nền tảng, khi xác thực thành công thì đơn vị vận chuyển ký vào **thông tin vận đơn** và chuyển tới cho người mua và người mua verify thành công thì thực hiện ký vào thông tin vận đơn để xác nhận rằng đã nhận hàng thành công , sau đó đơn vị vận chuyển thực hiện ký xác nhận vào thông tin vận đơn để xác nhận đã giao hàng cho người mua thành công .
* Thông tin vận đơn kèm cả chữ ký của người mua và đơn vị vận chuyển sẽ được gửi đến cho sàn thương mại , sàn thương mại tiến hành verify lại thông tin vận đơn đó.
* Sau khi sàn verify thành công thì thông tin vận đơn cùng với thông tin chuyển khoản sẽ được gửi cho người bán và người bán ký vào ,gửi  trung gian thông qua nền tảng đến cho người mua , người mua tiến hành verify , khi xác thực thành công người mua bấm “đã nhận được hàng” trên nền tảng.
* Sau khi người mua bấm “ đã nhận được hàng” ,  nền tảng sẽ tiến hành chuyển số tiền đang nắm giữ đến cho phía người bán. Khi đã nhận được thông báo và tiền hàng từ nền tảng , người bán tạo ra receipt và ký vào receipt đó gửi cho bên nền tảng và nền tảng verify lại chữ ký số của người bán .
* Nền tảng sau đó thực hiện ký vào receipt gửi tới cho người mua và người mua tiếp tiến hành verify lại chữ ký số của bên nền tảng, nếu xác minh thành công sẽ hoàn tất việc mua hàng.

**Module 4: Thanh toán khi nhận hàng**

**A diagram of a company

Description automatically generated**

* Người bán tạo vận đơn và ký bằng privatekey của người bán gửi tới cho sàn. Sàn thương mại sau khi đã verify lại thông tin thành công thì gửi đơn hàng cần giao cho đơn vị vận chuyển.
* Đơn vị vận chuyển ký vào thông tin vận đơn và tiến hành việc chuyển hàng đến cho người mua , khi hàng được chuyển đến thì người mua verify lại và tiến hành thanh toán giá trị đơn hàng cho đơn vị vận chuyển và ký vào **thông tin vận đơn** để xác nhận rằng đã nhận hàng và thanh toán đơn hàng thành công, sau đó đơn vị vận chuyển ký vào để xác nhận đã giao hàng đến người mua thành công .
* Thông tin vận đơn kèm cả chữ ký của người mua và đơn vị vận chuyển sẽ được gửi đến cho sàn thương mại , sàn thương mại tiến hành verify lại thông tin vận đơn đó. Sau khi verify thành công thì vận đơn sẽ được gửi cho người bán và người bán tạo ra receipt và ký bằng private vào receipt đó , gửi tới cho người mua thông qua sàn thương mại điện tử , người mua A tiến hành verify, khi xác thực thành công người mua bấm “đã nhận được hàng” trên nền tảng.

**Detail of Functional Features:**

**+ Generate key**  : tạo ra privatekey, Publickey của người dùng( gồm người mua , người bán, sàn thương mại điện tử, đơn vị vận chuyển )

**+ Sign Function :** thực hiện chức năng mã hóa, dữ liệu sẽ được băm bằng SHAKE256 và sau đó được ký bằng privatekey của bên liên quan.

**+ Verify Function :** thực hiện chức năng xác thực dữ liệu được gửi từ ai bằng publickey của người đó và tính nguyên gốc của dữ liệu , kiểm tra dữ liệu có bị thay đổi hay không .

**+ Encryption Data :** thực hiện chức năng mã hóa dữ liệu bằng thuật toán AES để để bảo dữ liệu trong quá trình truyền đi .

**+ Decryption Data :** thực hiện chức năng giải mã dữ liệu đã nhận từ bên liên quan bằng thuật toán AES để xem được bản rõ .

**+ Selection Payment method :** thực hiện chức năng cho phép người dùng lựa chọn phương thức thanh toán trực tiếp hay thanh toán online. Nếu :

**+ Thanh toán online:** Tiến hành chọn tới Module 3 và tiếp tục thực hiện quá trình. Các bên liên quan lúc này bao gồm ( người bán, người mua , ngân hàng, nền tảng thương mại điện tử )

**+ Thanh toán trực tiếp:** Tiến hành chọn tới Module 4 và tiếp tục thực hiện quá trình . Các bên liên quan lúc này bao gồm ( người bán, người mua , nền tảng thương mại điện tử , đơn vị vận chuyển )

**3. Implementation**

Các quá trình tạo public key, private key, ký và verify chữ ký của các bên được thực hiện bằng thuật toán Falcon thuật toán Lattice dựa trên NTRU theo NIST Selected Algorithms 2022 để chống lại được sự tấn công của máy tính lượng tử.

* **Triển khai vai trò của các bên trong nền tảng E-commerce:**

**+ Người mua:**

Thực hiện tạo thông tin đơn hàng trên nền tảng thương mại điện tử.

- Thực hiện việc ký và verify khi tiến hành các giao dịch giữa các bên liên quan

- Thực hiện chọn phương thức thanh toán trong phần **Selection Payment method.**

Thanh toán online.

Thanh toán khi nhận hàng.

- Tương tác với phía ngân hàng để thực hiện việc thanh toán online ,

- Thanh toán tiền hàng cho đơn vị vận chuyển khi chọn thanh toán khi nhận hàng.

**+ Người bán:**

- Thực hiện việc ký và verify khi tiến hành các giao dịch giữa các bên liên quan

-  Thực hiện tạo thông tin vận đơn thông qua sàn thương mại điện tử.

- Khi người mua bấm “đã nhận được hàng” trên nền tảng thì nền tảng sẽ chuyển tiền về cho người bán , tạo ra receipt .

- Tạo ra receipt khi người mua bấm “đã nhận được hàng” trên nền tảng khi chọn phương thức thanh toán là thanh toán khi nhận hàng (Module 4)

**+ Nền tảng E-commerce:**

Nền tảng thương mại điện tử thực hiện việc tạo publickey và lưu trữ  cho mỗi người dùng bằng thuật toán Identity-based cryptography , sau đó publickey qua hàm băm SHA3-512 sẽ phân phối cho mỗi người dùng làm ID .

- Thực hiện việc ký và verify khi tiến hành các giao dịch giữa các bên liên quan

Quản lý publickey và sign của người dùng.

Đảm đảm an toàn khi tiến hành thanh toán giữa các bên liên quan.

**+ Ngân hàng:**

- Thực hiện việc ký và verify khi tiến hành các giao dịch giữa các bên liên quan

- Thực hiện việc chuyển tiền cho sàn thương mại khi người mua yêu cầu thanh toán.

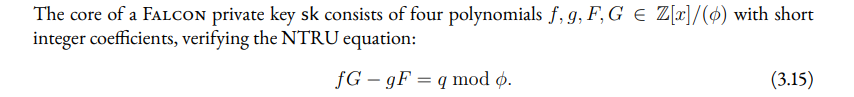
**+ Đơn vị vận chuyển:**

- Thực hiện việc ký và verify khi tiến hành các giao dịch giữa các bên liên quan.

- Thực hiện việc chuyển tiền cho nền tảng khi người mua chọn phương thức thanh toán là thanh toán khi nhận hàng.

**III. Kiến thức nền tảng**

1. **KeyGen**



* **Private key** gồm 4 đa thức f,g,F,G thuộc vành Z[x]/ϕ (với ϕ=Xn+1)(có dạng a0+a1.x1+a2.x2+..+an-1.xn-1) . Với f, g được tạo ngẫu nhiên, q=12289.
* Đa thức f= f0 + f1.x + f2. x2 +…+ fn-1.xn-1 . Hệ số fi được tính bằng 
* Đa thức g= g0+g1.x+g2.x2+…+gn-1.xn-1. Hệ số gi được tính bằng 

A math equations and numbers

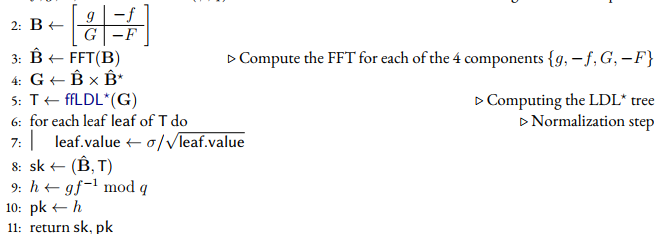
Description automatically generated with medium confidence

F,G được tính dựa trên NTRU equation:

A white paper with black text and numbers

Description automatically generated

* Sử dụng field norm N để ánh xạ f,g tới vành nhỏ hơn Z[x]/(xn/2+ 1), lặp lại cho tới khi được vành Z. Tính toán u,v trên vành Z theo công thức  uf − vg = gcd(f, g). F=v.q,G=u.q.
* Sau đó áp dụng tính chất nhân của field norm để liên tục nâng F,G lên vành Z[x]/(xn+1).
* Sau khi có được 4 đa thức F,G,f,g:



+ Tạo ma trận B theo như hình.

+ Tính B̂ từ hàm **FFT**

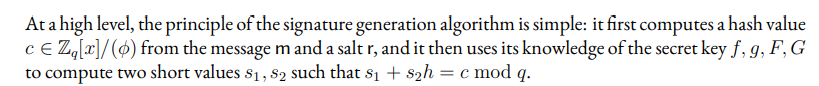
+ Tính G từ B̂. B̂

+Tính cây Falcon T từ hàm **ffLDL**(G) .Với mỗi lá trong cây T: cập nhật value

-> Private key sẽ gồm (B̂ ,T)



**Public key** là đa thức h thuộc vành Z[x]/ϕ(với ϕ=Xn+1) được tính theo công thức bên trên.



**2.Complexities**

Độ khó dựa vào việc khó tìm đa thức nhỏ f ’, g’ sao cho h=g’ . f ’-1 mod q

**3.Sign**

**A screenshot of a computer

Description automatically generated**

1. Sinh ngẫu nhiên một số r có độ dài 320 bit từ tập {0, 1}.
2. Dùng HashToPoint để chuyển (r||m) thành 1 điểm trên vành Zq[x]/(xn+1).
3. Tiền ảnh t (không nhất thiết phải ngắn) của c được tính toán.
4. Dựa vào tiền ảnh t và cây Falcon T dùng **ffSampling** để lấy mẫu vectơ z từ vectơ t theo một phân phối Gaussian.
5. Tính s dựa vào công thức 
6. Lặp lại (4 và 5) sao cho độ dài của s thỏa  với β = τsig · σ√2n, τsig = 1,1.
7. Lặp lại (4,5 và 6) nếu s=⊥.
8. s1 và s2 thỏa công thức s1+s2.h=c bằng c được tính bằng **invFFT(s).**.
9. s2 được mã hóa (nén) thành chuỗi bit s dùng hàm Compress.

=> Sig=(r,s)

**4.ffSampling**

+ Cách thích hợp để tạo (s1, s2) mà không làm rò rỉ khóa riêng là sử dụng trapdoor sampler. Ở Falcon sử dụng trapdoor sample có tên là fast fourier sampling.

+ ffSampling áp dụng làm tròn ngẫu nhiên (theo phân bố Gaussian rời rạc) trên các hệ số của t. Nhưng nó thực hiện điều đó một cách thích ứng, sử dụng thông tin được lưu trữ trong cây Falcon T.

**A screenshot of a computer

Description automatically generated**

**5.Verify**

**A screenshot of a computer

Description automatically generated**

+ Dùng HashToPoint để chuyển (r||m) thành 1 điểm trên vành Zq[x]/(xn+1).

+ Tính s2 bằng cách Decompress s. Nếu s2=⊥ => Lỗi

+ Tính s1 theo công thức 

+ Nếu  (với β = τsig · σ√2n, τsig = 1,1) thỏa =>Verify thành công.

Ngược lại => Lỗi(chiều dài chữ ký quá lớn)

**IV. DEMO**

**Kết quả :**

+ Public Key người bán :

A black screen with a blue line

Description automatically generated

+ Public Key người mua:

A black screen with white text

Description automatically generated

+ Private Key người bán:

A black screen with white lines

Description automatically generated

+ Private Key người mua:

A screenshot of a computer

Description automatically generated

+ File thông tin đơn hàng (pdf):

A screen shot of a computer

Description automatically generated

+ File chữ ký tương ứng:

A screenshot of a computer

Description automatically generated

+ Nội dung các file được load lên Mongodb:

A screenshot of a computer

Description automatically generated

+Thông tin người dùng trên Mongodb:

A close-up of a computer screen

Description automatically generated

**Link video ghi lại quá trình Demo:**

<https://youtu.be/A8_89305KqY>

**V.WRITE-UP CTF**

**Tên : Nguyễn Đặng Nguyên Khang**

**Finding Flags**

Có flag sẵn trong đề bài .

**flag:** crypto{y0ur\_f1rst\_fl4g}.

**Great Snakes**

Tải file [great\_snakes.py](https://cryptohack.org/static/challenges/great_snakes_35381fca29d68d8f3f25c9fa0a9026fb.py) đề cho về sau đó chạy đoạn code sẽ đưa ra flag.



**flag:** crypto{z3n\_0f\_pyth0n}

Bài 3:tải file [telnetlib\_example.py](https://cryptohack.org/static/challenges/telnetlib_example_dbc6ff5dc4dcfac568d7978a801d3ead.py) về sau đó chỉnh "buy": "clothes" về thành "buy": "flag" vì mình cần mua flag sau đó chạy file ta sẽ tìm được flag.

def json\_send(hsh):

request = json.dumps(hsh).encode() tn.write(request)

print(readline())

print(readline())

print(readline())

print(readline())

request = {

}

"buy": "flag"

json\_send(request)

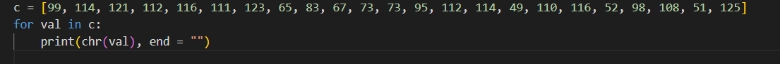
response = json\_recv()

**ASCII5**

Sử dụng hàm chr() trong python để chuyển từ số sang mã ascii tương ứng. Từ đó ta lấy được đoạn flag.

A black background with white text

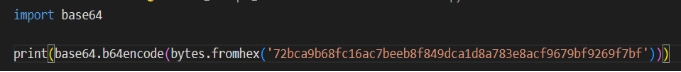
Description automatically generated



## HEX

Dùng hàm bytes.fromhex() có trong python để chuyển từ chuỗi hex sang lại dạng bytes.

**Base64**

Dùng hàm bytes.fromhex() mã hex sang bytes. Sau đó dùng dùng hàm b64encode() trong thư viện base64 để đổi từ bytes sang hệ cơ số 64.

**Bytes and Big Integers**

Dùng hàm long\_to\_bytes() trong thư viện Crypto.Util.number để đổi số lớn sang dạng bytes. Sau đó in ra flag cần tìm.A black background with white numbers

Description automatically generated

**Encoding Challenge**

Sử dụng các kiểu giải mã cơ bản ở những bài trước như bigint, hex, mã ascii, base64 và kết hợp với rot13, để giải mã. Server yêu cầu làm liên tục 100 lần để có được flag muốn tìm. Mỗi lần server gửi một mã và kiểu của mã đó về, nhiệm vụ ta là giải mã đó và gửi lại server. Làm liên tục 100 lần sẽ có được flag.A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

**XOR STARTER**

string = "label"

integer = 13

unicode\_repr = [ord(c) for c in string]

xor\_unicode = [13 ^ i for i in unicode\_repr]

xor\_string = "".join(chr(o) for o in xor\_unicode)

flag = "crypto{" + xor\_string + "}"

print("Flag: ")

print(flag)

Bắt đầu : khởi tạo một chuỗi ký tự có tên là "string" với giá trị là "label", và một số nguyên có tên là "integer" với giá trị là 13.

Dùng vòng lặp for để duyệt qua từng ký tự trong chuỗi "string" và sử dụng hàm ord(c) để lấy mã Unicode của từng ký tự đó. Kết quả sẽ được lưu vào một danh sách có tên là "unicode\_repr".

Tiếp theo, sử dụng phép XOR giữa số nguyên 13 (biểu thị bởi "integer") với từng giá trị mã Unicode trong danh sách "unicode\_repr", kết quả được lưu vào danh sách mới có tên là "xor\_unicode".

Dùng vòng lặp for và hàm chr(o) để chuyển đổi từng giá trị số nguyên trong danh sách "xor\_unicode" về dạng ký tự Unicode tương ứng. Kết quả sẽ được ghép lại thành một chuỗi mới có tên là "xor\_string" bằng cách sử dụng hàm "".join().

Cuối cùng, chuỗi "xor\_string" được ghép vào chuỗi "crypto{" ở đầu và ký tự "}" ở cuối để tạo thành chuỗi flag hoàn chỉnh. Chuỗi flag được in ra màn hình bằng lệnh print(), sau đó được hiển thị trong dòng kế tiếp có tiêu đề "Flag:".  
A black screen with blue text

Description automatically generated

**XOR PROPERTIES**

Cách giải quyết bài toán này chỉ là dựa vào tính chất của cổng XOR .   
A black background with yellow numbers

Description automatically generated

Chỉ cần XOR 3 dòng 1 , dòng 3, dòng 4 với nhau thì ta có được flag và chuyển flag về mã ASCII .

k1 = "a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313"

k2\_k1 = "37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e"

k2\_k3= "c1545756687e7573db23aa1c3452a098b71a7fbf0fddddde5fc1"

flag\_k1\_k3\_k2 = "04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf"

k1\_ord = [o for o in bytes.fromhex(k1)]

k2\_k3\_ord = [o for o in bytes.fromhex(k2\_k3)]

flag\_k1\_k3\_k2\_ord = [o for o in bytes.fromhex(flag\_k1\_k3\_k2)]

flag\_k1\_ord = [o\_f132 ^ o23 for (o\_f132, o23) in zip(flag\_k1\_k3\_k2\_ord, k2\_k3\_ord)

]

flag\_ord = [o\_f1 ^ o1 for (o\_f1, o1) in zip(flag\_k1\_ord, k1\_ord)]

flag = "".join(chr(o) for o in flag\_ord)

print("Flag: ")

print (flag)

A black screen with white text

Description automatically generated

**Quadratic Residues**

**for i in range (29):**

**if (i\*i)%29 == 11 :**

**print(i)**

Không tìm được i sao cho i^2 mod 29 =11 suy ra 11 là *Quadratic Non-Residue*

**for i in range (29):**

**if (i\*i)%29 == 14 :**

**print(i)**

Không tìm được i sao cho i^2 mod 29 =14 suy ra 14 là *Quadratic Non-Residue*

**for i in range (29):**

**if (i\*i)%29 == 6 :**

**print(i)**

Tồn tại 2 giá trị i là 8 và 21 có i^2 mod 29=6 và đề yêu cầu lấy giá trị nhỏ hơn làm flag nên flag là 8

**Legendre Symbol**

**from sympy.ntheory. residue\_ntheory import legendre\_symbol, \_sqrt\_mod\_tonelli\_shanks as mod\_sqrt**

**p = 101524035174539890485408575671085261788758965189060164484385690801466167356667036677932998889725476582421738788500738738503134356158197247473850273565349249573867251280253564698939768700489401960767007716413932851838937641880157263936985954881657889497583485535527613578457628399173971810541670838543309159139**

**ints = [25081841204695904475894082974192007718642931811040324543182130088804239047149283334700530600468528298920930150221871666297194395061462592781551275161695411167049544771049769000895119729307495913024360169904315078028798025169985966732789207320203861858234048872508633514498384390497048416012928086480326832803, 45471765180330439060504647480621449634904192839383897212809808339619841633826534856109999027962620381874878086991125854247108359699799913776917227058286090426484548349388138935504299609200377899052716663351188664096302672712078508601311725863678223874157861163196340391008634419348573975841578359355931590555, 17364140182001694956465593533200623738590196990236340894554145562517924989208719245429557645254953527658049246737589538280332010533027062477684237933221198639948938784244510469138826808187365678322547992099715229218615475923754896960363138890331502811292427146595752813297603265829581292183917027983351121325, 14388109104985808487337749876058284426747816961971581447380608277949200244660381570568531129775053684256071819837294436069133592772543582735985855506250660938574234958754211349215293281645205354069970790155237033436065434572020652955666855773232074749487007626050323967496732359278657193580493324467258802863, 4379499308310772821004090447650785095356643590411706358119239166662089428685562719233435615196994728767593223519226235062647670077854687031681041462632566890129595506430188602238753450337691441293042716909901692570971955078924699306873191983953501093343423248482960643055943413031768521782634679536276233318, 85256449776780591202928235662805033201684571648990042997557084658000067050672130152734911919581661523957075992761662315262685030115255938352540032297113615687815976039390537716707854569980516690246592112936796917504034711418465442893323439490171095447109457355598873230115172636184525449905022174536414781771, 50576597458517451578431293746926099486388286246142012476814190030935689430726042810458344828563913001012415702876199708216875020997112089693759638454900092580746638631062117961876611545851157613835724635005253792316142379239047654392970415343694657580353333217547079551304961116837545648785312490665576832987, 96868738830341112368094632337476840272563704408573054404213766500407517251810212494515862176356916912627172280446141202661640191237336568731069327906100896178776245311689857997012187599140875912026589672629935267844696976980890380730867520071059572350667913710344648377601017758188404474812654737363275994871, 4881261656846638800623549662943393234361061827128610120046315649707078244180313661063004390750821317096754282796876479695558644108492317407662131441224257537276274962372021273583478509416358764706098471849536036184924640593888902859441388472856822541452041181244337124767666161645827145408781917658423571721, 18237936726367556664171427575475596460727369368246286138804284742124256700367133250078608537129877968287885457417957868580553371999414227484737603688992620953200143688061024092623556471053006464123205133894607923801371986027458274343737860395496260538663183193877539815179246700525865152165600985105257601565]**

**print([mod\_sqrt(a, p) for a in ints if legendre\_symbol(a, p)==1][0])**

Thư viện SymPy trong Python để tính toán ký hiệu Legendre và căn bậc hai modulo một số nguyên tố lớn.

Dùng hàm legendre để xác định a có phải là Quadratic Residue

Nếu là Quadratic Residue ta sẽ tính căn bậc 2 bằng hàm mod\_sqrt(a, p)

Kết quả là :93291799125366706806545638475797430512104976066103610269938025709952247020061090804870186195285998727680200979853848718589126765742550855954805290253592144209552123062161458584575060939481368210688629862036958857604707468372384278049741369153506182660264876115428251983455344219194133033177700490981696141526

**Modular Square Root**

from sympy.ntheory. residue\_ntheory import legendre\_symbol, \_sqrt\_mod\_tonelli\_shanks as mod\_sqrt

a = 8479994658316772151941616510097127087554541274812435112009425778595495359700244470400642403747058566807127814165396640215844192327900454116257979487432016769329970767046735091249898678088061634796559556704959846424131820416048436501387617211770124292793308079214153179977624440438616958575058361193975686620046439877308339989295604537867493683872778843921771307305602776398786978353866231661453376056771972069776398999013769588936194859344941268223184197231368887060609212875507518936172060702209557124430477137421847130682601666968691651447236917018634902407704797328509461854842432015009878011354022108661461024768

p = 30531851861994333252675935111487950694414332763909083514133769861350960895076504687261369815735742549428789138300843082086550059082835141454526618160634109969195486322015775943030060449557090064811940139431735209185996454739163555910726493597222646855506445602953689527405362207926990442391705014604777038685880527537489845359101552442292804398472642356609304810680731556542002301547846635101455995732584071355903010856718680732337369128498655255277003643669031694516851390505923416710601212618443109844041514942401969629158975457079026906304328749039997262960301209158175920051890620947063936347307238412281568760161

print(mod\_sqrt(a,p))

Thư viện SymPy trong Python để tính toán ký hiệu Legendre và căn bậc hai modulo một số nguyên tố lớn.

Nếu là Quadratic Residue ta sẽ tính căn bậc 2 bằng hàm mod\_sqrt(a, p)

Kết quả là:

28169512554311284614348161812907461395482195258388583125795498809297226147214152907614055638917789190356917578259717792167302913007927989841763977292434488782635964253677743342038748567333074043589267896292373028724763808006697707070301035339291758998923066001985927788808579330075671953036025191791621915640175242425390397212674797332132801882880223506177201168864920484993546017284338829512010922075018689505381642887042980971582058343875078178836965895987271392081926458392283354971823611423820865651283490761548053384731721391637064349021755899877224522161311561209530712702153163501623531290150340903913036821041

**Chinese Remainder Theorem**

from sympy.ntheory.modular import crt

# Khai báo các đại lượng

moduli = [5, 11, 17]

remainders = [2, 3, 5]

# Giải hệ phương trình đồng dư

x, \_ = crt (moduli, remainders)

# In kết quả

print(x)

Kết quả là 872

**VECTOR**

**Ý tưởng :** chỉ là phép nhân vô hướng , cộng các vector trong không gian 3 chiều, theo những định nghĩa mà đề bài đã đưa về phép tính. “dot” ở dòng 12 là phép nhân vô hướng.

import numpy as np

v = np.array([2, 6, 3])

w = np.array([1, 0, 0])

u = np.array([7, 7, 2])

result1 = 2\*v - w

result2= 3 \* result1

result3 = 2 \* u

ans = np.dot(result2, result3)

print(ans)

Key : 702

**SIZE AND BASIC**

**Ý tưởng :** tính kích thước của vector cho trước bằng cách nhân với chính nó thông qua biến “cal” và sau đó tính căn ( biến ans ), ép kiểu về int .

import numpy as np

# v = np.array( 4, 6, 2, 5)

import math

v = [ 4, 6,2,5]

# tícgh vô hướng với nó lun

cal = sum([a\*\*2 for a in v])

ans = math.sqrt( cal)

print ( int (ans))

Kết quả : 9

**GRAM SCHMIDTDT**

**Ý tưởng :**

bài toán yêu cầu trực giao hóa 4 vector ( mã giả có sẵn ) , kết quả làm tròn tới 5 chữ số thập phân .

import numpy as np

v = [

np.array([4,1,3,-1]),

np.array([2,1,-3,4]),

np.array([1,0,-2,7]),

np.array([6,2,9,-5]),

]

u = [v[0]]

for vi in v[1:]:

mi = [np.dot(vi, uj) / np.dot(uj, uj) for uj in u]

u += [vi - sum([mij \* uj for (mij, uj) in zip(mi,u)])]

# lam tròn 5 chữ số thập phân bẰNG ROUND

print(round(u[3][1], 5))

**Kết quả :**

0.91611

**WHAT’S A LATTICE ?**

**Ý tưởng :** đề thì dài nhưng tóm lại chỉ yêu cầu **tính định thức ,** dùng hàm abs để nguyên dương, round để làm tròn số và ép về int .

**import numpy as np**

**import math as mt**

**v1 = np.array([6, 2, -3])**

**v2 = np.array([5, 1, 4])**

**v3 = np.array([2, 7, 1])**

**matrix = np.vstack((v1, v2, v3))**

**# Tính định thức**

**DET = np.linalg.det(matrix)**

**print(int (round(abs(DET))))modular s**

**Kết quả : 255**

**Gaussian Reduction**

**Ý tưởng :** Nếu độ dài v2 <v1, thì hoán đổi giá trị giữa v1 và v2.

* Nếu m bằng 0, trả về v1 và v2 là cơ sở mới của lưới.
* Cập nhật giá trị của v2 bằng cách trừ m nhân v1 từ v2.
* Tiếp tục lặp lại các bước trên cho đến khi không còn thể thực hiện các phép cập nhật nữa.

Sau khi hoàn thành thuật toán, kết quả là hai vector v1 và v2 là cơ sở mới của lưới, và giá trị của phép nhân chấm giữa v1 và v2 được in ra làm flag cho bài toán.

import numpy as np

ar = np.array

v = ar([846835985, 9834798552], dtype='i8')

u = ar([87502093, 123094980], dtype='i8')

v1, v2 = u, v

if np.linalg.norm(v2) < np.linalg.norm(v1):

print('swap')

v1, v2 = v2, vi

m = int(np.dot (v1, v2) / np.dot(v1, v1))

print(m)

v2 = v2 - m \* v1

print(v2)

if np.linalg.norm(v2) < np.linalg.norm(v1):

print('swap')

v1, v2 = v2, v1

m = int(np.dot (v1, v2) / np.dot(v1, v1))

print(m)

print(v1)

print(v2)

print(np.dot (v1, v2))

**Find the lattices**

**ý tưởng :** 2 cái véc tơ cơ sở là 1, h và 0, q , rút gọn gauss cho 2 véc tơ này là thu được f, g . Rồi xài hàm decrypt có sẵn của bài là xong.

**from Crypto.Util.number import \***

**import numpy as np**

**h = 2163268902194560093843693572170199707501787797497998463462129592239973581462651622978282637513865274199374452805292639586264791317439029535926401109074800**

**q = 7638232120454925879231554234011842347641017888219021175304217358715878636183252433454896490677496516149889316745664606749499241420160898019203925115292257**

**e = 5605696495253720664142881956908624307570671858477482119657436163663663844731169035682344974286379049123733356009125671924280312532755241162267269123486523**

**def decrypt(f, g, e):**

**a = (f\*e) % q**

**m = (a\*inverse(f, g)) % g**

**return m**

**def gauss(u, v):**

**u = np.array(u)**

**v = np.array(v)**

**while True:**

**if u.dot(u) > v.dot(v):**

**u, v = v, u**

**m = u.dot(v) // u.dot(u)**

**if m == 0:**

**return u**

**v -= m \* u**

**# We gotta find (f, g) such that f \* h == g + t \* q**

**(f, g) = gauss((1, h), (0, q))**

**assert f \* f < q and g \* g < q and GCD(f, g) == 1**

**msg = decrypt(f, g, e)**

**print(long\_to\_bytes(msg).decode())**

**# flag : crypto{Gauss\_lattice\_attack!}**

**TÊN : TRẦN VỸ KHANG**

**FAVOURITE BYTE**

Ý tưởng : để giải quyết bài toán này thì ta sử dụng brute force algorithm, ta xor đoạn mã đề cho với từng byte từ [0, 256] . tìm dòng lệnh có fomat ‘crypto’ và in dòng đó ra màn hình.

**string = "73626960647f6b206821204f21254f7d694f7624662065622127234f726927756d"**

**string\_ord = [o for o in bytes.fromhex(string)]**

**for order in range(256):**

**possible\_flag\_ord = [order^ o for o in string\_ord]**

**possible\_flag = "".join(chr(o) for o in possible\_flag\_ord)**

**if possible\_flag.startswith("crypto"):**

**flag = possible\_flag**

**break**

**print("Flag: ")**

**print(flag)**

**Flag:**

**crypto{0x10\_15\_my\_f4v0ur173\_by7e}**

**You either know, XOR you don't**

**encrypted\_msg = "0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e175d0e077e263451150104"**

**encrypted\_msg = bytes.fromhex(encrypted\_msg)**

**flag\_format = b"crypto{"**

**key = [O1 ^ O2**

**for (O1, O2) in zip(encrypted\_msg, flag\_format)] + [ord("y")]**

**flag = []**

**key\_len = len(key)**

**for i in range(len(encrypted\_msg)):**

**flag.append(**

**encrypted\_msg[i] ^ key[i % key\_len])**

**flag = "".join(chr(o) for o in flag)**

**print("Flag:")**

**print (flag)**

**Ý tưởng của bài toán :** giải mã chuỗi encrypted\_msg bằng phép XOR (^) với một danh sách các giá trị byte được tính toán từ encrypted\_msg và flag\_format. Để làm điều này, danh sách key được tạo ra bằng cách thực hiện phép XOR giữa từng cặp byte tương ứng trong encrypted\_msg và flag\_format, sau đó thêm giá trị số nguyên tương ứng với ký tự "y" vào cuối danh sách key.

Sau khi có danh sách key, đoạn mã tiếp tục thực hiện phép XOR (^) giữa từng byte trong encrypted\_msg với byte tương ứng trong key, lặp lại chu kỳ này cho đến hết chuỗi encrypted\_msg. Kết quả được lưu vào danh sách flag dưới dạng các giá trị byte giải mã. Cuối cùng, danh sách flag được chuyển đổi thành chuỗi ký tự (str) bằng cách sử dụng "".join(chr(o) for o in flag), và kết quả giải mã sẽ được in ra màn hình.

**Flag:**

crypto{1f\_y0u\_Kn0w\_En0uGH\_y0u\_Kn0w\_1t\_4ll}

**Lemur XOR**

**Ý TƯỞNG :** Vì RGB có 8 màu , đề yêu cầu cho xor các byte giữa 2 ảnh , cho nên chúng ta sử dụng các thao tác trên 2 hình ảnh đã được cho sẵn , chuyển 2 hình ảnh thành mảng numpy \* 255 vì RGB có giá trị trong khoảng từ [0-255]. Và sau đó sử dụng hàm bitwise XOR của numpy để có được một mảng số nguyên (int), do đó ta sử dụng astype(np.uint8) để chuyển đổi kiểu dữ liệu của mảng n\_image thành kiểu dữ liệu uint8, đại diện cho các số nguyên không dấu 8-bit. Và dùng Image.fromarray(n\_image).save('n.png')

ĐỂ đổi lại thành đuôi png ( tức file ảnh ) .

import numpy as np

from PIL import Image

img1 = Image.open('lemur.png')

img2 = Image.open('flag.png')

n1 = np.array(img1)\*255

n2 = np.array(img2)\*255

# xor đe ra dc hinh anh

n\_image = np.bitwise\_xor(n1, n2).astype(np.uint8)

Image.fromarray(n\_image).save('n.png')

**Được kết quả** : crypto{ X0Rly\_n0t ! }

A close up of a cat

Description automatically generated**Greatest Common Divisor**

* Để tìm ước chung lớn nhất của 2 số a và b, ta tìm số lớn nhất mà cả a và b đều chia hết và xuất ra.

A computer screen shot of a code

Description automatically generated

A black background with white text

Description automatically generated

**Extended GCD**

* Viết hàm euclid mở rộng và xuất số có giá trị nhỏ hơn trong 2 sốx và y

A screen shot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generated

**Modular Arithmetic 1**

* Dùng phép toán modulo và hàm min trong python  
  A number on a black background

  Description automatically generated

A black background with white text

Description automatically generated

**Modular Arithmetic 2**

* Dùng hàm pow() trong python và thêm phép toán modulo. Nhưng không cần thực hiện phép toán modulo vì nếu p là số nguyên tố thì pow(a, p) % p = a % p. Do đó ta có a ^ (p - 1) = 1 (mod p)A black background with white text

  Description automatically generated

* Do định lý fermat nhỏ, ta có a^(p - 1) = 1 (mod p) nên a \* a ^ (p - 2) = 1 (mod p) trong biểu thức thì ta có a ^ (p - 2) là nghịch đảo modulo.

A black background with white letters

Description automatically generated

**Privacy-Enhanced Mail?**

from Crypto. PublicKey import RSA

key=RSA.importKey(open('privacy\_enhanced\_mail\_1f696c053d76a78c2c531bb013a92d4a.pem').read())

Khóa bí mật là ‘15682700288056331364787171045819973654991149949197959929860861228180021707316851924456205543665565810892674190059831330231436970914474774562714945620519144389785158908994181951348846017432506464163564960993784254153395406799101314760033445065193429592512349952020982932218524462341002102063435489318813316464511621736943938440710470694912336237680219746204595128959161800595216366237538296447335375818871952520026993102148328897083547184286493241191505953601668858941129790966909236941127851370202421135897091086763569884760099112291072056970636380417349019579768748054760104838790424708988260443926906673795975104689’

**CERtainly Not**

key=RSA.importKey(open('2048b-rsa-example-cert.der','rb').read())

print(key.n)

Dùng wget trong kali linux để tải file .der về sau đó dùng thư viện crypto trong python để dùng lệnh RSA.importKey để lấy key từ việc đọc file.

Để đọc file ta dùng lệnh

key=RSA.importKey(open('2048b-rsa-example-cert.der','rb').read())

Sau đó để in ra modulo ta print(key.n)

Kết quả là

22825373692019530804306212864609512775374171823993708516509897631547513634635856375624003737068034549047677999310941837454378829351398302382629658264078775456838626207507725494030600516872852306191255492926495965536379271875310457319107936020730050476235278671528265817571433919561175665096171189758406136453987966255236963782666066962654678464950075923060327358691356632908606498231755963567382339010985222623205586923466405809217426670333410014429905146941652293366212903733630083016398810887356019977409467374742266276267137547021576874204809506045914964491063393800499167416471949021995447722415959979785959569497

**SSH Keys**

key=RSA.importKey(open('bruce\_rsa.pub').read())

print(key.n)

dùng wget để tải file .pub sau đódùng thư viện crypto trong python để dùng lệnh RSA.importKey để lấy key từ việc đọc file.

Để đọc file ta dùng lệnh key=RSA.importKey(open('bruce\_rsa\_6e7ecd53b443a97013397b1a1ea30e14.pub').read()).Sau đó lấy kết quả modulo bằng print(key.n)

**Transparency**

import hashlib để sử dụng hàm băm SHA256.

sau đó

lấy khóa công khai key

dùng der = key.exportKey(format='DER') để chuyn đổi key từ định dạng PEM sang định dạng DER

Dùng hashlib.sha256(der) để tính giá trị băm SHA256 của khóa công khai der

dùng sha256\_fingerprint = sha256.hexdigest() để chuyển đổi giá trị băm SHA256 từ dạng byte sang dạng chuỗi hex và lưu vào biến "sha256\_fingerprint".

sau đó print(f"Public Key SHA256: {sha256\_fingerprint}") để in ra giá trị vân tay SHA256 của khóa công khai.

**import hashlib**

**from Crypto. PublicKey import RSA**

**pem = open('transparency.pem', 'r').read()**

**key = RSA.importKey(pem)**

**der = key.exportKey(format='DER')**

**sha256 = hashlib.sha256 (der)**

**sha256\_fingerprint = sha256.hexdigest()**

**print(f"Public Key SHA256: {sha256\_fingerprint}")**

Sau khi lấy được giá trị vân tay SHA256 của khóa công khai ta truy cập vào web <https://search.censys.io/> sau đó nó dán vào thanh tìm kiếm ta sẽ tìm link của trang web

A screenshot of a computer

Description automatically generated

Là <https://thetransparencyflagishere.cryptohack.org/>

Sau đó truy cập vào web ta được flag là :crypto{thx\_redpwn\_for\_inspiration}

A screenshot of a computer

Description automatically generated

**ý tưởng :** sử dụng cuộc tấn công mật mã thấp mật độ để giải quyết bài toán knapsack-cryptosystem. Đầu tiên, người tạo thử thách xây dựng một ma trận đặc biệt dựa trên công thức được mô tả trong một bài báo nghiên cứu. Ma trận này được thiết kế để tận dụng thuật toán LLL để giải quyết bài toán tổng con của hệ mật mã knapsack. Sau đó, người tạo thách thức sử dụng công cụ Sage để áp dụng thuật toán LLL vào ma trận này, từ đó thu được các giá trị khôi phục lại được từ văn bản rõ (plaintext), và cuối cùng là khôi phục được cờ (flag) của thử thách.

* **code : a = [**
* **#public key**
* **]**
* **s = #ciphertext**
* **n = len(a)**
* **N = ceil(sqrt(n) / 2)**
* **b = []**
* **for i in range(n):**
* **vec = [0 for \_ in range(n + 1)]**
* **vec[i] = 1**
* **vec[-1] = N \* a[i]**
* **b.append(vec)**
* **b.append([1 / 2 for \_ in range(n)] + [N \* s])**
* **BB = matrix(QQ, b)**
* **l\_sol = BB.LLL()**
* **for e in l\_sol:**
* **if e[-1] == 0:**
* **msg = 0**
* **isValidMsg = True**
* **for i in range(len(e) - 1):**
* **ei = 1 - (e[i] + (1 / 2))**
* **if ei != 1 and ei != 0:**
* **isValidMsg = False**
* **break**
* **msg |= int(ei) << i**
* **if isValidMsg:**
* **print('[\*] Got flag:', long\_to\_bytes(msg))**
* **break**

**Successive Powers**

* Ta có hệ phương trình như sau:

s[0] \* x mod p = s[1]

s[1] \* x mod p = s[2]

……….

s[n - 1] \* x mod p = s[n]

Hệ phương trình trên tương đương:

x = s[0]^(-1) \*s[1] mod p

x = s[1]^(-1) \*s[2] mod p

……….

x = s[n - 1]^(-1) \*s[n] mod p

Ta biết được p là một số nguyên tố có 3 chữ số nên ta chỉ cần duyệt trâu p và sau đó kiểm tra liệu các phương trình có cho ra cùng 1 số giống nhau là được.A computer screen with numbers and symbols

Description automatically generatedA black background with white text

Description automatically generated

**Adrien's Signs**

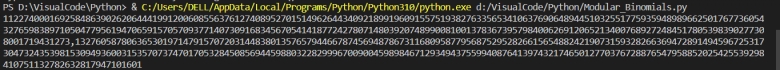
* Dùng Legendre Symbol vì nếu là quadratic residue thì bit có thể là 1 hoặc ngược lại.A computer screen with white text

  Description automatically generatedA black background with white text

  Description automatically generated

**Modular Binomials**

* Dùng factordb có sẵn để phân tích số n ra 2 thừa số nguyên tố p, q. Sau đó chỉ việc xuất ra.A black background with white lines

  Description automatically generated

**Broken RSA**

* Vì n là số nguyên tố, e là một số lũy thừa của 2, chúng ta có thể lấy các căn bậc hai liên tiếp (nếu có thể) để tìm căn bậc thứ e.A computer screen with text

  Description automatically generated

**No Way Back Home**

* Mục đích của ta là tìm ra v và sau đó giải mã v theo mã hóa AES. Để tìm v, ta làm như sau:

v \* b \* b^(-1) mod n = v \* a \* a^(-1) mod n

v \* a \* b \* v mod n = v \* (a \* b) \* v mod n

v = va \* vb \* (vab)^(-1) mod n

Sau đó giải mã theo AES sẽ được cờ. A computer screen with text

Description automatically generated

**Ellipse Curve Cryptography**

**from Crypto.Cipher import AES**

**from Crypto.Util.Padding import pad, unpad**

**from hashlib import sha1**

**shared\_secret = 83201481069630956436480435779471169630605662777874697301601848920266492**

**key = sha1(str(shared\_secret).encode('ascii')).digest()[:16]**

**iv = bytes.fromhex('64bc75c8b38017e1397c46f85d4e332b')**

**encrypted\_flag = bytes.fromhex('13e4d200708b786d8f7c3bd2dc5de0201f0d7879192e6603d7c5d6b963e1df2943e3ff75f7fda9c30a92171bbbc5acbf')**

**cipher = AES.new(key, AES.MODE\_CBC, iv)**

**print(unpad(cipher.decrypt(encrypted\_flag), 16).decode())**

Sử dụng thư viện Crypto để giải mã một chuỗi ký tự được mã hóa bằng AES-CBC với một khóa và vector khởi tạo (IV) nhất định. Khóa được tính toán bằng cách áp dụng hàm băm SHA-1 lên một số nguyên được chia sẻ trước đó và lấy 16 byte đầu tiên của giá trị băm. Sau khi giải mã, chuỗi ký tự được bỏ đi phần đệm và được chuyển đổi thành một chuỗi văn bản thông thường.

**Flag:** crypto{c0n1c\_s3ct10n5\_4r3\_f1n1t3\_gr0up5}

**Roll your Own**

**from pwn import remote**

**from json import loads, dumps**

**io = remote('socket.cryptohack.org', 13403)**

**io.readuntil(b'Prime generated: "')**

**q = int(io.readline()[:-2], 16)**

**io.sendline(dumps({"g":hex(q+1), "n":hex(q\*\*2)}).encode())**

**io.readuntil(b'Generated my public key: "')**

**pub = int(io.readline()[:-2], 16)**

**io.sendline(dumps({"x":hex((pub-1)//q)}).encode())**

**io.readuntil(b'What is my private key: ')**

**print(loads(io.readline().decode())['flag'])**

Thực hiện kết nối tới một máy chủ ở địa chỉ 'socket.cryptohack.org' trên cổng 13403. Sau đó, gửi một thông điệp JSON qua đường truyền để tạo một khóa công khai cho một hệ mã hóa khóa công khai được xác định bởi một số nguyên tố q và một số nguyên n. Sau đó, Gửi một thông điệp JSON khác để tính toán khóa riêng tư cho khóa công khai được tạo trước đó và cuối cùng hiển thị thông tin cờ được trả về bởi máy chủ.

**Flag:** crypto{Grabbing\_Flags\_with\_Pascal\_Paillier}

**Unencryptable**

**N = 0x7fe8cafec59886e9318830f33747cafd200588406e7c42741859e15994ab62410438991ab5d9fc94f386219e3c27d6ffc73754f791e7b2c565611f8fe5054dd132b8c4f3eadcf1180cd8f2a3cc756b06996f2d5b67c390adcba9d444697b13d12b2badfc3c7d5459df16a047ca25f4d18570cd6fa727aed46394576cfdb56b41**

**e = 0x10001**

**c = 0x5233da71cc1dc1c5f21039f51eb51c80657e1af217d563aa25a8104a4e84a42379040ecdfdd5afa191156ccb40b6f188f4ad96c58922428c4c0bc17fd5384456853e139afde40c3f95988879629297f48d0efa6b335716a4c24bfee36f714d34a4e810a9689e93a0af8502528844ae578100b0188a2790518c695c095c9d677b**

**p = 8239835397208516111720362847949425401045672365829937602117480449316694558226622200110057535873802132963548914201468383545676262090246827792522994758916609**

**q = 10900824353334471830007307529937357926160386461967884446160315218630687793341471079170750548554707926611542019859296605188535413447791710067186432371970369**

**d = pow(e, -1, (p-1)\*(q-1))**

**print(bytes.fromhex(hex(pow(c, d, N))[2:]).decode())**

Thực hiện giải mã RSA trên một ciphertext c cho trước bằng cách sử dụng khóa bí mật (N, d) và in ra plaintext tương ứng.

Các giá trị N, e, c, p và q là các thành phần tiêu chuẩn của RSA:

N là modulus RSA, là tích của hai số nguyên tố lớn p và q.

e là số mũ công khai, là một số lẻ nhỏ tương đối nguyên tố cùng nhau với (p-1)(q-1).

c là ciphertext, là kết quả của việc mã hóa plaintext sử dụng khóa công khai (N, e).

p và q là những yếu tố nguyên tố của N.

Để giải mã ciphertext, tính toán số mũ bí mật d bằng cách sử dụng modular inverse của e modulo (p-1)(q-1). Sau đó, sử dụng khóa bí mật (N, d) để giải mã ciphertext c bằng cách sử dụng tính chất mũ mod với hàm pow(). Cuối cùng, chuyển đổi plaintext kết quả từ định dạng hex sang ASCII bằng cách sử dụng các phương thức bytes.fromhex() và decode()

**Flag:** crypto{R3m3mb3r!\_F1x3d\_P0iNts\_aR3\_s3crE7s\_t00}