

**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 ĐẠNG NGUYỄN 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

|  |            |
|--|------------|
| <b>I . TỔNG QUAN ĐỀ TÀI .....</b>                      | <b>3.</b>  |
| Chủ đề & Ngữ cảnh.....                                 |            |
| Yêu cầu về bảo mật & Giải pháp bảo mật .....           |            |
| Các bên liên quan .....                                |            |
| Thư viện, các nguồn tham khảo .....                    |            |
| <b>II . SOLUTION.....</b>                              | <b>5.</b>  |
| Solution Architecture.....                             |            |
| Mô tả chi tiết Architecture..... 4 Module .....        |            |
| Thực thi, vai trò các bên liên quan.....               |            |
| <b>III. KIẾN THỨC NỀN TẢNG.....</b>                    | <b>13.</b> |
| Keygen , complexities, sign , ffSampling ,verify.....  |            |
| <b>IV. DEMO.....</b>                                   | <b>18.</b> |
| Kết quả và Link video ghi lại quá trình thực thi ..... |            |
| <b>V.WRITE-UP CTF .....</b>                            | <b>22.</b> |
| <b>CryptoHack.....</b>                                 |            |
| Nguyễn Đặng Nguyên Khang .....                         | 22.        |
| Trần Võ Khang .....                                    | 36.        |

## PHÂN CHIA CÔNG VIỆC

|                                 |  |
|---------------------------------|--|
| <b>Trần Võ Khang</b>            | <ul style="list-style-type: none"><li>- Code chương trình</li><li>- viết phần thuật toán Falcon trong bài Final.</li><li>- Góp ý và sửa đổi final</li></ul>        |
| <b>Nguyễn Đặng Nguyên Khang</b> | <ul style="list-style-type: none"><li>- Viết bài final</li><li>- code kết nối database MongoDB(up dữ liệu, load dữ liệu)</li><li>- Góp ý và sửa đổi code</li></ul> |

# 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

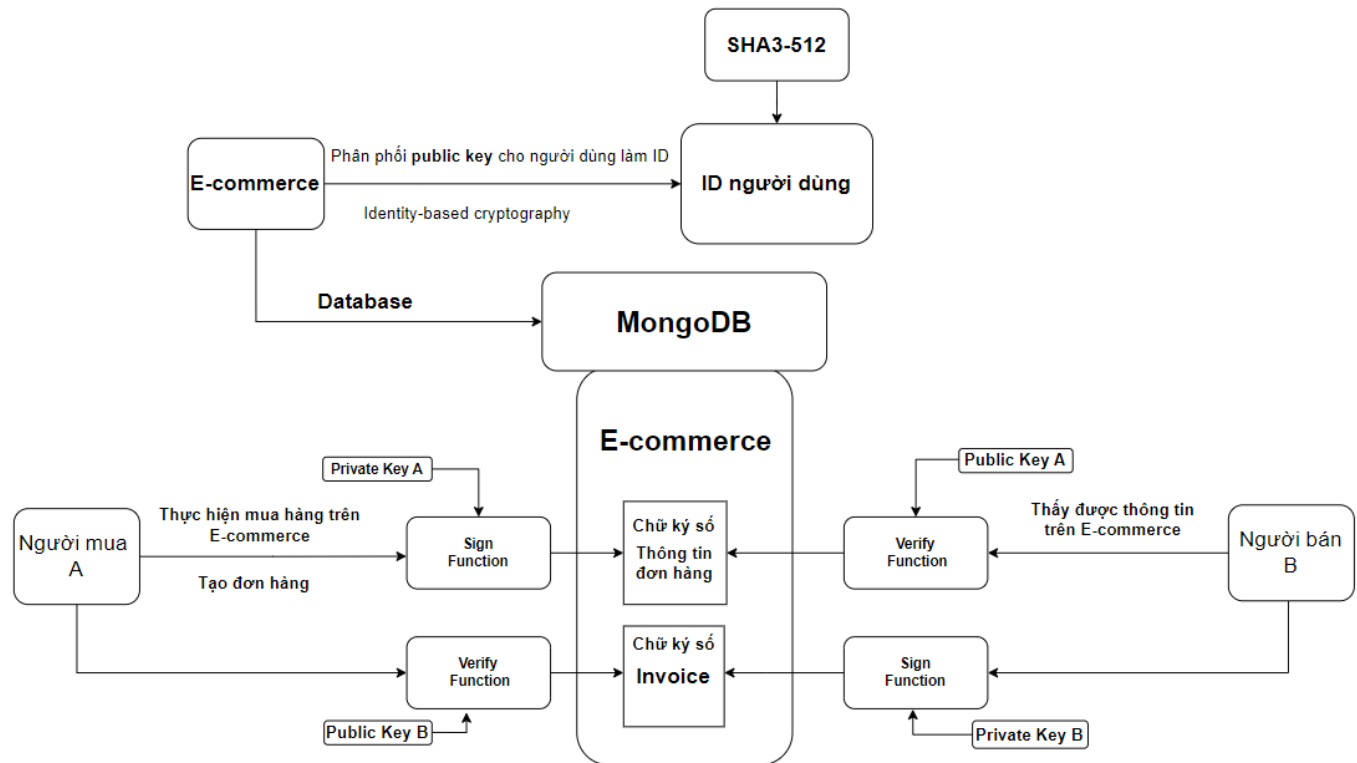
- 1.Nền tảng thương mại điện tử,
- 2.Người bán
- 3.Người mua
- 4.Ngân hàng
- 5.Đơn vị vận chuyển
- 6.Attacker giả mạo và đánh cắp dữ liệu khi dùng máy tính lượng tử

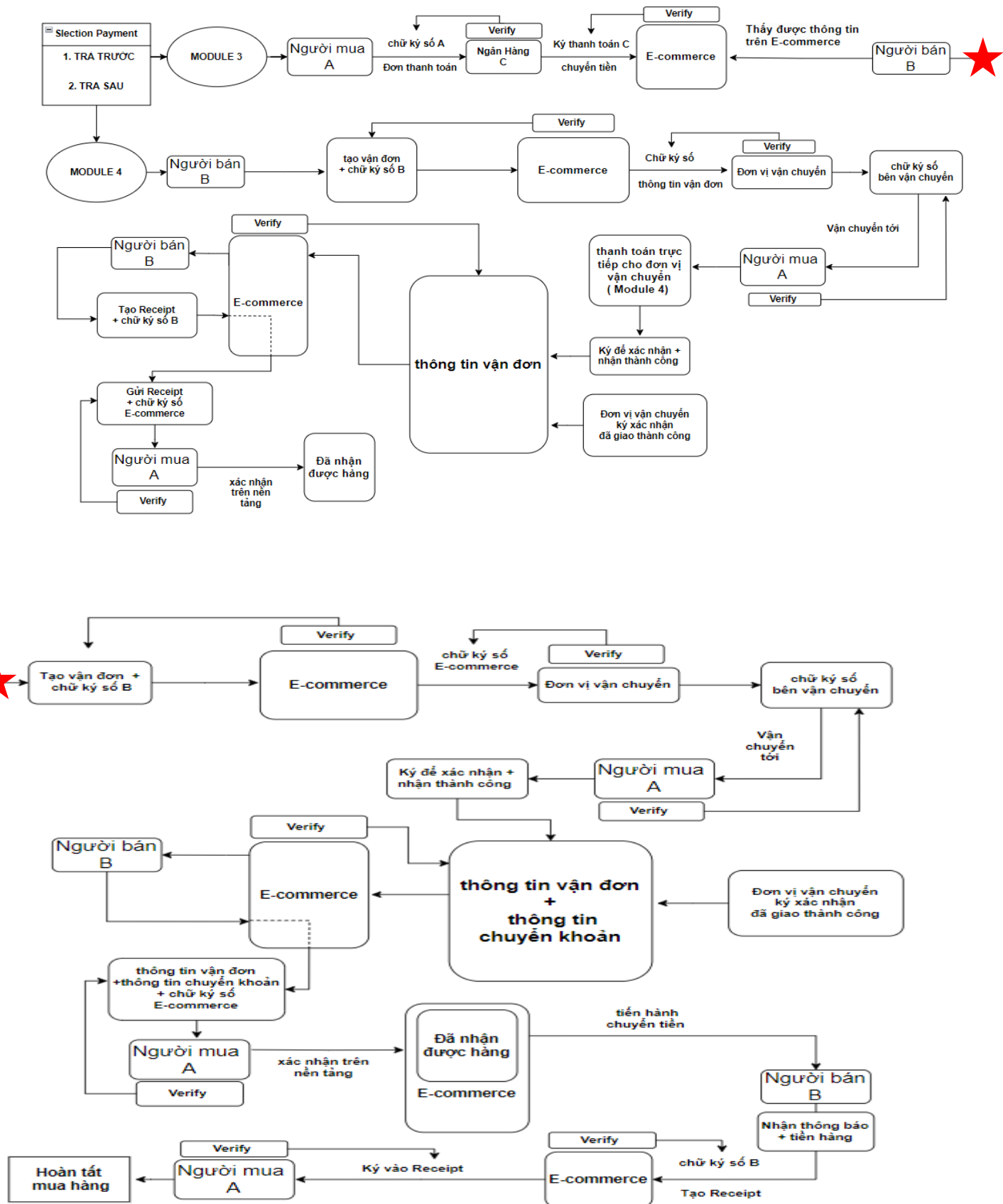
## 6. Các nguồn tham khảo:

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





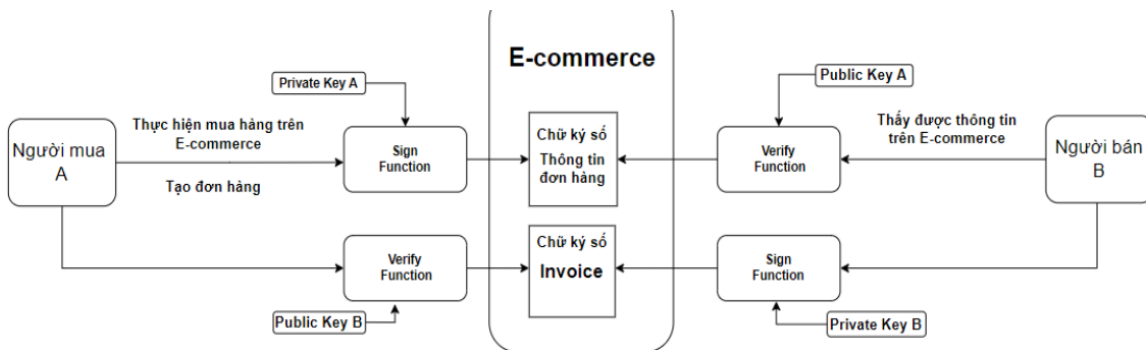
## 2. Solution Detail Architecture

### Module 1:



- 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.

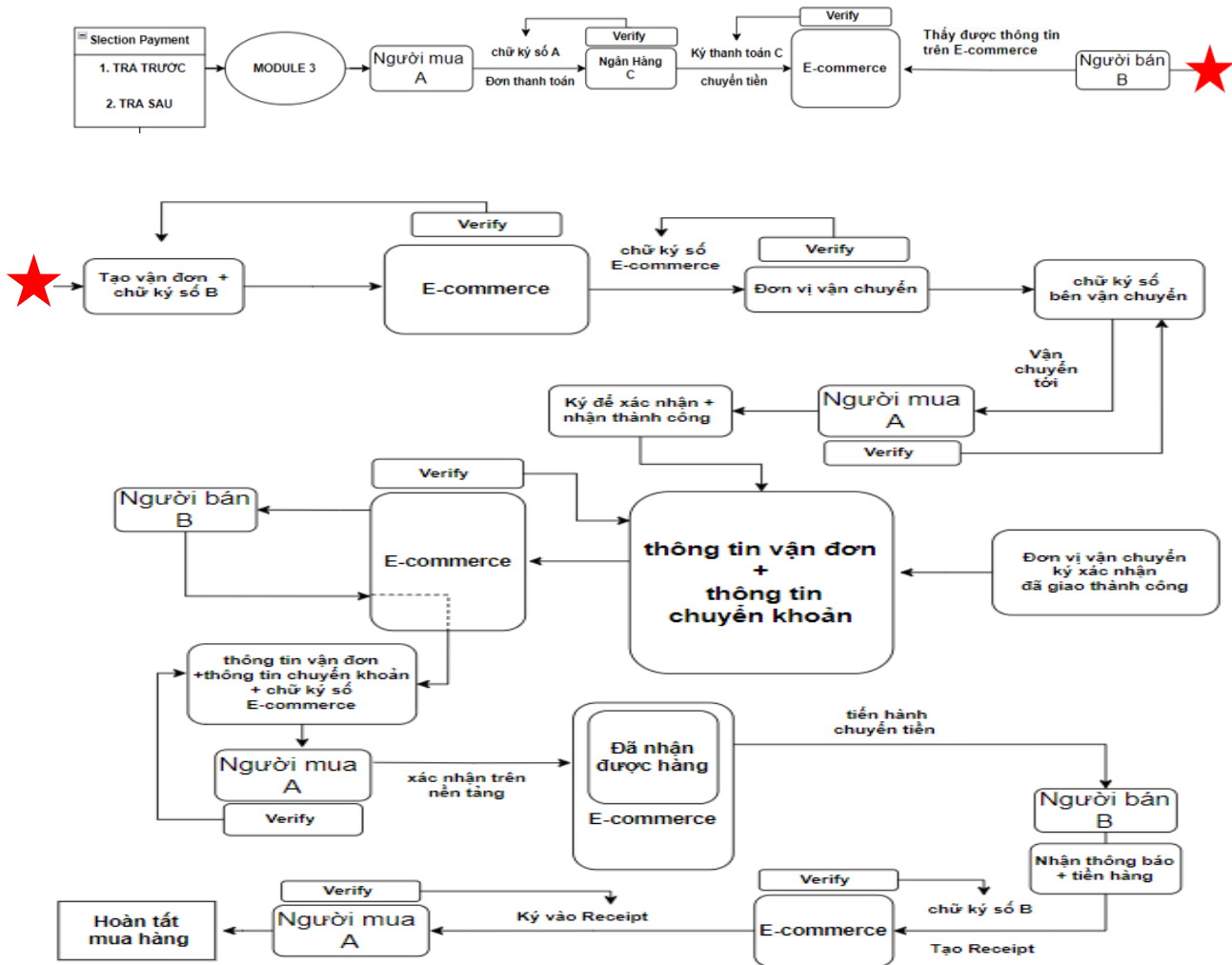
### Module 2:



- 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



- 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

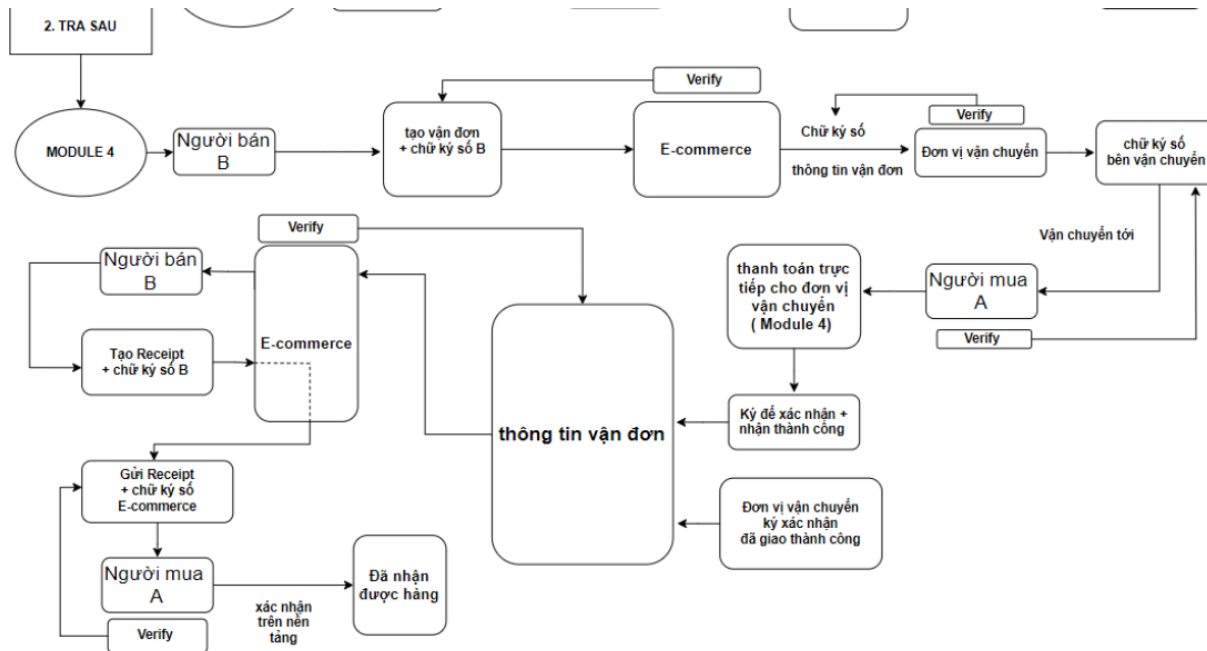


được thông tin trên sàn và tiến hành tạo vận đơn và ký bằng private gửi cho sàn thương mại , sàn thực hiện verify lại bằng publickey của người bán .

- 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



- 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 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 bảo 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 sản 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**

The core of a FALCON private key  $sk$  consists of four polynomials  $f, g, F, G \in \mathbb{Z}[x]/(\phi)$  with short integer coefficients, verifying the NTRU equation:

$$fG - gF = q \bmod \phi. \quad (3.15)$$

- **Private key** gồm 4 đa thức  $f, g, F, G$  thuộc vành  $\mathbb{Z}[x]/\phi$  (với  $\phi = X^n + 1$ ) (có dạng  $a_0 + a_1.x^1 + a_2.x^2 + \dots + a_{n-1}.x^{n-1}$ ) . Với  $f, g$  được tạo ngẫu nhiên,  $q=12289$ .
- Đa thức  $f = f_0 + f_1.x + f_2.x^2 + \dots + f_{n-1}.x^{n-1}$  . Hệ số  $f_i$  được tính bằng  $D_{\mathbb{Z}, \sigma_{\{f, g\}}, 0}$

- Đa thức  $g = g_0 + g_1.x + g_2.x^2 + \dots + g_{n-1}.x^{n-1}$ . Hệ số  $g_i$  được tính bằng  $D_{\mathbb{Z}, \sigma_{\{f,g\}}, 0}$

One way to sample  $z \leftarrow D_{\sigma_{\{f,g\}}}$  (lines 5 and 6) is to perform:

$$z = \sum_{i=1}^{4096/n} z_i, \quad \text{where } \begin{cases} z_i \leftarrow \text{SamplerZ}(0, \sigma^*), \\ \sigma^* = 1.17 \cdot \sqrt{\frac{q}{8192}} \approx 1.43300980528773 \end{cases}$$

F, G được tính dựa trên NTRU equation:

---

**Algorithm 6**  $\text{NTRUSolve}_{n,q}(f, g)$

---

Require:  $f, g \in \mathbb{Z}[x]/(x^n + 1)$  with  $n$  a power of two

Ensure: Polynomials  $F, G$  such that (3.15) is verified

```

1: if  $n = 1$  then
2:   Compute  $u, v \in \mathbb{Z}$  such that  $uf - vg = \gcd(f, g)$  ▷ Using the extended GCD
3:   if  $\gcd(f, g) \neq 1$  then
4:     abort and return  $\perp$ 
5:    $(F, G) \leftarrow (vq, uq)$ 
6:   return  $(F, G)$ 
7: else
8:    $f' \leftarrow N(f)$  ▷  $f', g', F', G' \in \mathbb{Z}[x]/(x^{n/2} + 1)$ 
9:    $g' \leftarrow N(g)$  ▷  $N$  as defined in either (3.25) or (3.26)
10:   $(F', G') \leftarrow \text{NTRUSolve}_{n/2,q}(f', g')$  ▷ Recursive call
11:   $F \leftarrow F'(x^2)g(-x)$  ▷  $F, G \in \mathbb{Z}[x]/(x^n + 1)$ 
12:   $G \leftarrow G'(x^2)f(-x)$ 
13:  Reduce $(f, g, F, G)$  ▷  $(F, G)$  is reduced with respect to  $(f, g)$ 
14: return  $(F, G)$ 

```

- Sử dụng field norm  $N$  để ánh xạ  $f, g$  tới vành nhỏ hơn  $\mathbb{Z}[x]/(x^{n/2} + 1)$ , lặp lại cho tới khi được vành  $\mathbb{Z}$ . Tính toán  $u, v$  trên vành  $\mathbb{Z}$  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  $\mathbb{Z}[x]/(x^n + 1)$ .
- Sau khi có được 4 đa thức  $F, G, f, g$ :

```

2:  $\mathbf{B} \leftarrow \left[ \begin{array}{c|c} g & -f \\ \hline G & -F \end{array} \right]$ 
3:  $\hat{\mathbf{B}} \leftarrow \text{FFT}(\mathbf{B})$  ▷ Compute the FFT for each of the 4 components  $\{g, -f, G, -F\}$ 
4:  $\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^*$ 
5:  $\mathbf{T} \leftarrow \text{ffLDL}^*(\mathbf{G})$  ▷ Computing the LDL* tree
6: for each leaf  $\text{leaf}$  of  $\mathbf{T}$  do ▷ Normalization step
7: |  $\text{leaf.value} \leftarrow \sigma / \sqrt{\text{leaf.value}}$ 
8:  $\text{sk} \leftarrow (\hat{\mathbf{B}}, \mathbf{T})$ 
9:  $h \leftarrow gf^{-1} \bmod q$ 
10:  $\text{pk} \leftarrow h$ 
11: return  $\text{sk}, \text{pk}$ 

```

+ Tạo ma trận B theo như hình.

+ Tính  $\hat{\mathbf{B}}$  từ hàm **FFT**

+ Tính G từ  $\hat{\mathbf{B}}$ .

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

nhật value  $\text{leaf.value} \leftarrow \sigma / \sqrt{\text{leaf.value}}$

-> Private key sẽ gồm  $(\hat{\mathbf{B}}, \mathbf{T})$

The FALCON public key  $\text{pk}$  corresponding to the private key  $\text{sk} = (f, g, F, G)$  is a polynomial  $h \in \mathbb{Z}_q[x]/(\phi)$  such that:

$$h = gf^{-1} \bmod (\phi, q). \quad (3.17)$$

**Public key** là đa thức  $h$  thuộc vành  $\mathbb{Z}[x]/\phi$  (với  $\phi = X^n + 1$ ) được tính theo công thức bên trên.

At a high level, the principle of the signature generation algorithm is simple: it first computes a hash value  $c \in \mathbb{Z}_q[x]/(\phi)$  from the message  $m$  and a salt  $r$ , and it then uses its knowledge of the secret key  $f, g, F, G$  to compute two short values  $s_1, s_2$  such that  $s_1 + s_2 h = c \bmod q$ .

## 2.Complexities

Độ khó dựa vào việc khó tìm đa thức nhỏ  $f', g'$  sao cho  $h = g' \cdot f'^{-1} \bmod q$

## 3.Sign

---

**Algorithm 10** **Sign** ( $m, sk, \lfloor \beta^2 \rfloor$ )

---

Require: A message  $m$ , a secret key  $sk$ , a bound  $\lfloor \beta^2 \rfloor$

Ensure: A signature  $\text{sig}$  of  $m$

```
1:  $r \leftarrow \{0, 1\}^{320}$  uniformly
2:  $c \leftarrow \text{HashToPoint}(r \| m, q, n)$ 
3:  $\mathbf{t} \leftarrow \left( -\frac{1}{q} \text{FFT}(c) \odot \text{FFT}(F), \frac{1}{q} \text{FFT}(c) \odot \text{FFT}(f) \right) \quad \triangleright \mathbf{t} = (\text{FFT}(c), \text{FFT}(0)) \cdot \hat{\mathbf{B}}^{-1}$ 
4: do
5:   do
6:      $\mathbf{z} \leftarrow \text{ffSampling}_n(\mathbf{t}, T)$ 
7:      $\mathbf{s} = (\mathbf{t} - \mathbf{z})\hat{\mathbf{B}} \quad \triangleright \text{At this point, } \mathbf{s} \text{ follows a Gaussian distribution: } \mathbf{s} \sim D_{(c,0)+\Lambda(\mathbf{B}),\sigma,0}$ 
8:     while  $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor \quad \triangleright \text{Since } \mathbf{s} \text{ is in FFT representation, one may use (3.8) to compute } \|\mathbf{s}\|^2$ 
9:        $(s_1, s_2) \leftarrow \text{invFFT}(\mathbf{s}) \quad \triangleright s_1 + s_2 h = c \bmod (\phi, q)$ 
10:     $\mathbf{s} \leftarrow \text{Compress}(s_2, 8 \cdot \text{sbytelen} - 328) \quad \triangleright \text{Remove 1 byte for the header, and 40 bytes for } r$ 
11:  while  $(\mathbf{s} = \perp)$ 
12: return  $\text{sig} = (r, \mathbf{s})$ 
```

---

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  $\mathbb{Z}_q[x]/(x^n+1)$ .
3. Tiền ảnh  $\mathbf{t}$  (không nhất thiết phải ngắn) của  $c$  được tính toán.
4. Dựa vào tiền ảnh  $\mathbf{t}$  và cây Falcon  $T$  dùng `ffSampling` để lấy mẫu vector  $\mathbf{z}$  từ vector  $\mathbf{t}$  theo một phân phối Gaussian.

5. Tính  $\mathbf{s}$  dựa vào công thức  $\mathbf{s} = (\mathbf{t} - \mathbf{z})\mathbf{B}$

6. Lặp lại (4 và 5) sao cho độ dài của  $\mathbf{s}$  thỏa  $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$  với  $\beta = \tau_{\text{sig}} \cdot \sigma \sqrt{2n}$ ,  $\tau_{\text{sig}} = 1,1$ .
7. Lặp lại (4,5 và 6) nếu  $\mathbf{s} = \perp$ .
8.  $s_1$  và  $s_2$  thỏa công thức  $s_1 + s_2 \cdot h = c$  bằng  $c$  được tính bằng `invFFT(s)`.
9.  $s_2$  được mã hóa (nén) thành chuỗi bit  $s$  dùng hàm `Compress`.

=>  $\text{Sig} = (r, s)$

## 4.ffSampling

+ Cách thích hợp để tạo  $(s_1, s_2)$  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  $\mathbf{t}$ . Nhưng nó thực hiện điều đó một cách thích ứng, sử dụng thông tin được lưu trữ trong cây Falcon  $T$ .



---

**Algorithm 11** `ffSamplingn`(**t**, **T**)

---

Require: **t** = ( $t_0, t_1$ )  $\in \text{FFT}(\mathbb{Q}[x]/(x^n + 1))^2$ , a **FALCON** tree **T**

Ensure: **z** = ( $z_0, z_1$ )  $\in \text{FFT}(\mathbb{Z}[x]/(x^n + 1))^2$

Format: All polynomials are in FFT representation.

```
1: if  $n = 1$  then
2:    $\sigma' \leftarrow \mathbf{T.value}$  ▷ It is always the case that  $\sigma' \in [\sigma_{\min}, \sigma_{\max}]$ 
3:    $z_0 \leftarrow \text{SamplerZ}(t_0, \sigma')$  ▷ Since  $n = 1$ ,  $t_i = \text{invFFT}(t_i) \in \mathbb{Q}$  and  $z_i = \text{invFFT}(z_i) \in \mathbb{Z}$ 
4:    $z_1 \leftarrow \text{SamplerZ}(t_1, \sigma')$ 
5:   return z = ( $z_0, z_1$ )
6: ( $\ell, \mathbf{T}_0, \mathbf{T}_1$ )  $\leftarrow$  (T.value, T.leftchild, T.rightchild)
7:  $\mathbf{t}_1 \leftarrow \text{splitfft}(t_1)$  ▷  $\mathbf{t}_0, \mathbf{t}_1 \in \text{FFT}(\mathbb{Q}[x]/(x^{n/2} + 1))^2$ 
8:  $\mathbf{z}_1 \leftarrow \text{ffSampling}_{n/2}(\mathbf{t}_1, \mathbf{T}_1)$  ▷ First recursive call to ffSamplingn/2
9:  $z_1 \leftarrow \text{mergefft}(\mathbf{z}_1)$  ▷  $\mathbf{z}_0, \mathbf{z}_1 \in \text{FFT}(\mathbb{Z}[x]/(x^{n/2} + 1))^2$ 
10:  $t'_0 \leftarrow t_0 + (t_1 - z_1) \odot \ell$ 
11:  $\mathbf{t}_0 \leftarrow \text{splitfft}(t'_0)$ 
12:  $\mathbf{z}_0 \leftarrow \text{ffSampling}_{n/2}(\mathbf{t}_0, \mathbf{T}_0)$  ▷ Second recursive call to ffSamplingn/2
13:  $z_0 \leftarrow \text{mergefft}(\mathbf{z}_0)$ 
14: return z = ( $z_0, z_1$ )
```

---

## 5. Verify

---

**Algorithm 16 Verify** ( $m, \text{sig}, \text{pk}, \lfloor \beta^2 \rfloor$ )

---

Require: A message  $m$ , a signature  $\text{sig} = (r, s)$ , a public key  $\text{pk} = h \in \mathbb{Z}_q[x]/(\phi)$ , a bound  $\lfloor \beta^2 \rfloor$

Ensure: Accept or reject

- 1:  $c \leftarrow \text{HashToPoint}(r \| m, q, n)$
- 2:  $s_2 \leftarrow \text{Decompress}(s, 8 \cdot \text{sbytelen} - 328)$
- 3: if  $(s_2 = \perp)$  then
- 4: | reject ▷ Reject invalid encodings
- 5:  $s_1 \leftarrow c - s_2 h \bmod q$  ▷  $s_1$  should be normalized between  $\left[-\frac{q}{2}\right]$  and  $\left[\frac{q}{2}\right]$
- 6: if  $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$  then
- 7: | accept
- 8: else
- 9: | reject ▷ Reject signatures that are too long

+ Dùng HashToPoint để chuyển  $(r \| m)$  thành 1 điểm trên vành  $\mathbb{Z}_q[x]/(x^n+1)$ .

+ Tính  $s_2$  bằng cách Decompress  $s$ . Nếu  $s_2 = \perp \Rightarrow$  Lỗi

+ Tính  $s_1$  theo công thức  $s_1 \leftarrow c - s_2 h \bmod q$

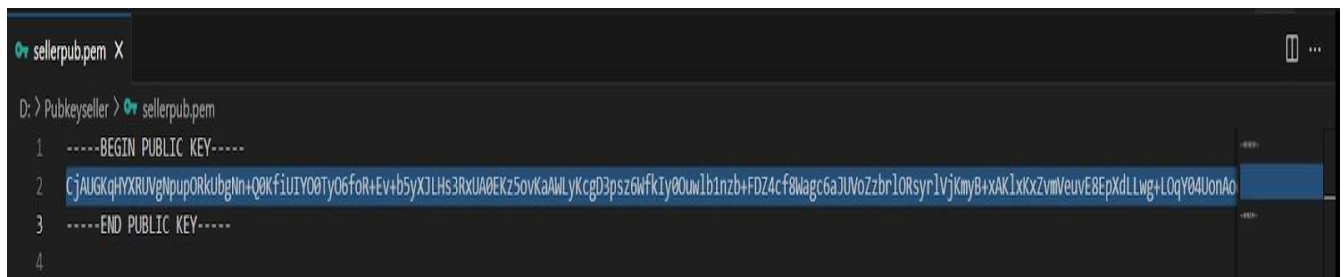
+ Nếu  $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$  (với  $\beta = \tau_{\text{sig}} \cdot \sigma \sqrt{2n}$ ,  $\tau_{\text{sig}} = 1,1$ ) thỏa  $\Rightarrow$  Verify thành công.

Ngược lại  $\Rightarrow$  Lỗi (chiều dài chữ ký quá lớn)

## IV. DEMO

### Kết quả :

+ Public Key người bán :



```
sellerpub.pem X
D:\Pubkeyseller > cat sellerpub.pem
1 -----BEGIN PUBLIC KEY-----
2 CjAUGKqHYXRUVgltupORkubgn+Q8Kf1UIY00Ty06for+Ev+bsyX3LHs3RXUA0EKz5ovKaAWlyKcgD3psz6WfkIy00uwlbnzb+FDZ4cf8Wagc6aJUVoZzbr1ORsyrlVjKmyB+xAkLxKxZvmVeuvE8EpXdLLwg+LQqY04UonAo
3 -----END PUBLIC KEY-----
4
```

+ Public Key người mua:

```

D: > Pubkeybuyer > buyerpub.pem
1  -----BEGIN PUBLIC KEY-----
2  ClDqzqjdV/qxsNcIw3jcCw74MEpjKVVV18AtzfZDwFoIP1VaJZgwK9ptnSLVXzu1lGcboKPFkItng9ysFHTSPFGuJpBUASRATj1405lRV2ITxi4IUgSG5IFXEdhbpV6oI1NGPLQ9u7wFZQcc6neT2f188qh0vkz5IqKaVh3Fi4i
3  -----END PUBLIC KEY-----
4

```

+ Private Key người bán:

```

K: > sellerpri > sellerpri.pem
1  -----BEGIN PRIVATE KEY-----
2  WgeCMYgiAWH/eGH4v/38Rh994fgB6TfRg4L/vgAHwPh6AHvC90PvaJshP+90WSdGAhCff/3e/8AQN98MoxcB73weGAHyC7v3e+H/e+9D4YhgD4YCd6HgAh+EYOiB8Qv1H4YQDEH3Q8Fw3QD6AwBAEH/AeILfxABw3PC6b/+iB8
3  -----END PRIVATE KEY-----
4

```

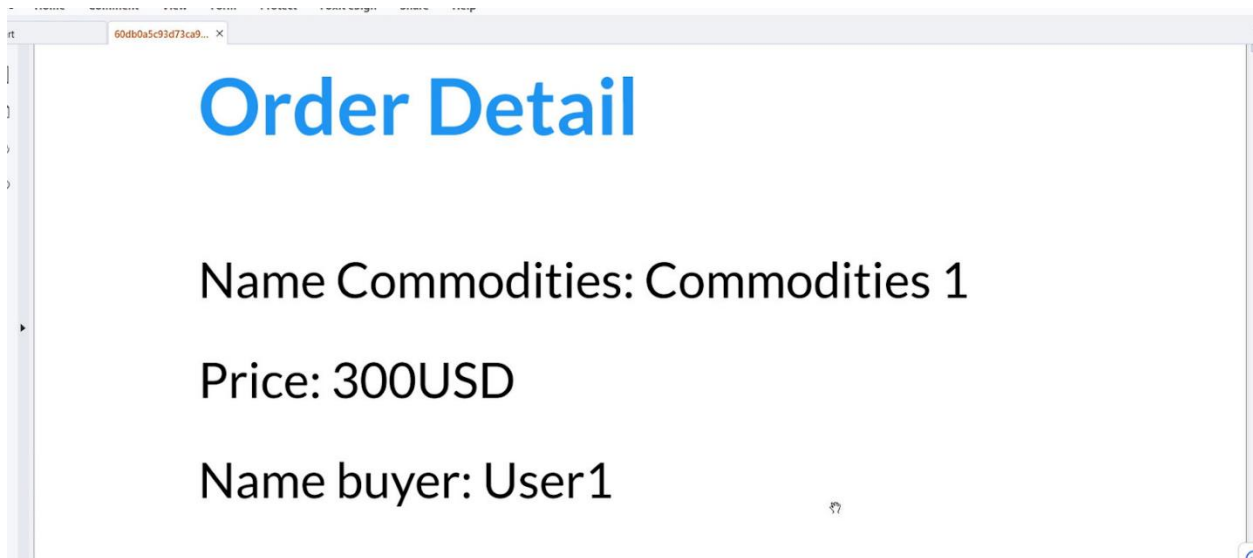
+ Private Key người mua:

```

K: > buyerpri > buyerpri.pem
1  -----BEGIN PRIVATE KEY-----
2  WgAEDgAj0Hp0iEL3w/IHoMmB4Wu1B/4ff+IACA7v+f4T/+af8fgAB/gQfH8X8eALvQe8L2Te2T5dBCMIukHr49h//IRfALovm+AH/D8LhAC/sA+fAEXff+AhRfKAgxCEP3+jAEPw+6DgygLsg8+98H/AEQ3+hCAYheCAHeE98
3  -----END PRIVATE KEY-----
4

```

+ File thông tin đơn hàng (pdf):



+ File chữ ký tương ứng:



The screenshot shows a web browser window with a single tab titled "60dbf". The browser's address bar is empty. The page content consists of a single line of text, which is a long Base64-encoded string. The string begins with "pGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" and ends with "aGbkY+T1q1tjnYws1wdiXKo7WjGzhtY1Fycan5ecfBCl9Qxn". The browser's status bar at the bottom indicates the current position is "Ln 1, Col 1", the zoom level is "100%", the encoding is "Windows (CRLF)", and the character set is "UTF-8".

```
pGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAADxRFC4DV1zskN5nchDBbiJP5NFGPrCyPLavmepL3s
y+We7eOm3tFoconk2Cu5oxETquVnFsTJYHgbBI/PzSt4jvJK
U4tFWtkulUq0EZSQeGP/tNYZ0z9evd26kM0tGvIH2C1Xm1mN
qpvrD2TnvY8vMGpbaFSzRMip1dV7CJ07GDbXjpRMmDS1X0tG
nEsSHUnSCsDhYw8K0tqd4zdMq1IitihnNxuTraFrNuRCd8ZH
1x84/IMsyOKSBiUTuz9arIc7nbVLXCSNRRcplLc4JsgJZd+p
HuKvqDjP9jf3H2Z+ma1bTxNJ09sVe7MfXxjsluuNKqefdAdH
AyCv1kHZSajlsijDveiRE0Iq+
7zjTPC3324TfRBCcg/nJZJ71Pu2ruk5kynPGbbzPPTUzVVbU
UzLYvxHWlew2CN4F/WU2nHzcct03YMZNsFSnGO0UphIOMhMl
ielceowumZpkEIgJJzU554om73nR/Ay2QkC2kOK3DZ9v1Ynk
Z2Led7Gtc3keQvuEtTQj1HuUFZfsQtkMi36rdVpJI9Egaoi6
Gsqu9oZ3iRSPRzobFOTgNFmelbMoTQ//uXrco+w70E5Hbcpw
qvAlgjVEuTR91186aHFU6awciBarNJboZlUKSn5nvrkxcLl/
lPzdYYcMSJl9o69ZQkvcevr7G/NlFSybEMYJFaVwhtrYDcpc
sU/bDBotw0iLLJcv7IClyMmDx5cGBfK57HISh56pc3Fu3Drb
CGuaDlc6z82OV6CNe2jw//Qcc3WGW6ZQ9VZsJo9U6YR56lRJ
QUL0Iq5ueuae0bf7pOKnLWGsONT/63XTwvRKF+pTBxp94edc
182/VlsjQOWs8xbnHTldTeS2/6VHtwKsNRdGXv8UgqIS9pLP
MILDiihiZmXRkRS1VqJzeIutuNzBqMUaG0EQzq17tcLmOava
YJO43VbP5FNJQ2UtXwfNZD0RSFz7ijhTHXJmm9G+jgFlMIY3
wNtevP2U9cG805PMPPrHVmmTKfLnTXlMO7cR555AYJPkdafJ+
HfXrSN6XBpVtJfzrgVGGVFrKbL26mN2h80i6c0d5Z9FDVS1h
Hb68RlyrPJieggiYOzeU3l3pn5NTY7L6w268lEubjeKgA7Je
aGbkY+T1q1tjnYws1wdiXKo7WjGzhtY1Fycan5ecfBCl9Qxn
```

+ Nội dung các file được load lên MongoDB:

## DDH.File

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 78.86KB TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 4KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes

INSERT DOCUMENT

Filter

Type a query: { field: 'value' }

Reset

Apply

Options

QUERY RESULTS: 1-1 OF 1

```
_id: null
FilePdfName: "60db0a5c93d73ca9330485cb1376f4c9cde58711873e30cb3a97d02af2e873b5bd4c79..."
ContentPdf: Binary.createFromBase64('JVBER10xLjQKJdPr6eEKMSAwIG9iago8PC9DcmVhdGlvbkrRhdGUGKEQ6MjAyNDAAxMDYyMDQ2NDcrMDcnMDAn...
FileSigName: "60db0a5c93d73ca9330485cb1376f4c9cde58711873e30cb3a97d02af2e873b5bd4c79..."
ContentSig: "OgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADxRFC4DV1zskN5nc..."
FilePubName: "60db0a5c93d73ca9330485cb1376f4c9cde58711873e30cb3a97d02af2e873b5bd4c79..."
ContentPub: "-----BEGIN PUBLIC KEY-----
CldQzqjdV/qxsNcIwJjCcW74WEpjKVVV10AtzfZDWfo..."
IdSeller: "47e42fc753fea72ccb437bf00679b189b3db85c6cccf6cc8ae01b3551e5e1ade6632c0..."
```

+Thông tin người dùng trên MongoDB:

QUERY RESULTS: 1-2 OF 2

```
_id: "47e42fc753fea72ccb437bf00679b189b3db85c6cccf6cc8ae01b3551e5e1ade6632c0..."
FirstName: "Khang"
LastName: "Tran"
UserName: "248b86037e5597c75e4157bf58726c92921575f86df14f942bac641a7a5936b833a164..."
password: "248b86037e5597c75e4157bf58726c92921575f86df14f942bac641a7a5936b833a164..."
pubkeyname: "248b86037e5597c75e4157bf58726c92921575f86df14f942bac641a7a5936b833a164..."
pubkeycontent: "-----BEGIN PUBLIC KEY-----
CjAUGKqHYXRUVgNpupORkUbgNn+Q0KfiUIY00TyO6fo..."
```

```
id: "768c422c2c98b2b5ddac52ca92859f46dcd20446b51160c55f074bf18860a370a0b1 "
```

Link video ghi lại quá trình Demo:

[https://youtu.be/A8\\_89305KqY](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](#) về cho về sau đó chạy đoạn code sẽ đưa ra flag.

```
...code\extensions\ms_python\python-2923.6.0\python11es\110\python\debuggy\adapter\...\debuggy\launcher 56661 C:\Users\Khang\Downloads\g
reat_snakes_35381fca29d68df3f25c9fa0a9026fb (5).py'
Here is your flag:
crypto{z3n_of_pyth0n}
PS C:\Users\Khang\Downloads> █
```

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

Bài 3:tải file [telnetlib\\_example.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.

```
}
PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe crypto{ASCII_print4b13}

c = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]
for val in c:
    print(chr(val), end = "")
```

## HEX

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

```
print(bytes.fromhex('6372797074667b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f737472696e67735f615f6c6f747d'))

PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe crypto{You_will_be_working_with_hex_strings_a_lot}
```

## 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.

```
import base64

print(base64.b64encode(bytes.fromhex('72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf')))

PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe crypto/Base+64+Encoding+is+Web+Safe/
PS C:\Users\DELL>
```

## 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.

```

from Crypto.Util.number import *

a = 11515195063862318899931685488813747395775516287289682636499965282714637259206269
c = long to bytes(a)
print(c)

PS C:\Users\DELL> & C:/Users/DELL/App
b'crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}'

```

## 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.

```

pwntools_example_f93ca6ccef2def755aa8f6d9aa6e9c5b.py 2 • telnetlib_example_dbc6ff5dc4dcfac568d7978a801d3ead (1).py
C: > Users > DELL > Downloads > telnetlib_example_5b11a835055df17c7c8f8f2a08782c44.py > ...

22
23 def json_send(hsh):
24     request = json.dumps(hsh).encode()
25     tn.write(request)
26
27 for i in range(0, 101):
28     received = json_rcv()
29     if "flag" in received:
30         print(received["flag"])
31         break
32     code = received["encoded"]
33     t = received["type"]
34
35     res = None
36     if t == "flag":
37         print(code)
38         break
39     if t == "base64":
40         res = base64.b64decode(code).decode('utf8')
41     elif t == "hex":
42         res = (unhexlify(code)).decode('utf8')
43     elif t == "rot13":
44         res = codecs.decode(code, 'rot_13')
45     elif t == "bigint":
46         res = unhexlify(code.replace("0x", "")).decode('utf8')
47     elif t == "utf-8":
48         res = ""
49         for c in code:
50             res += chr(c)
51     print("code : " + res)

code : accurate_expert_editorials
code : rob_rotation_int
code : surge_shanghai_often
crypto{3nc0d3_d3c0d3_3nc0d3}
PS C:\Users\DELL>

```



## 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:".

```
ugpy\adapter/../../debugpy\launcher' '49886' '--' 'D:\PYTHON\python_tech28\less1.py'
Flag:
crypto{aloha}
PS D:\PYTHON>
```

## XOR PROPERTIES

Cách giải quyết bài toán này chỉ là dựa vào tính chất của cổng XOR .

```
KEY1 = a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313
KEY2 ^ KEY1 = 37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e
KEY2 ^ KEY3 = c1545756687e7573db23aa1c3452a098b71a7fbf0fdddddde5fc1
FLAG ^ KEY1 ^ KEY3 ^ KEY2 = 04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf
```

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 = "c1545756687e7573db23aa1c3452a098b71a7fbf0fdddddde5fc1"

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)
```

```
Flag:
crypto{x0r_i5_ass0c1at1v3}
PS D:\PYTHON>
```

## Quadratic Residues

```
for i in range (29):  
    if (i*i)%29 == 11 :  
        print(i)
```

Không tìm được i sao cho  $i^2 \bmod 29 = 11$  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  $i^2 \bmod 29 = 14$  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  $i^2 \bmod 29 = 6$  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 =  
101524035174539890485408575671085261788758965189060164484385690801  
466167356667036677932998889725476582421738788500738738503134356158  
197247473850273565349249573867251280253564698939768700489401960767  
007716413932851838937641880157263936985954881657889497583485535527  
613578457628399173971810541670838543309159139  
  
ints =  
[250818412046959044758940829741920077186429318110403245431821300888  
042390471492833347005306004685282989209301502218716662971943950614  
625927815512751616954111670495447710497690008951197293074959130243  
601699043150780287980251699859667327892073202038618582340488725086  
33514498384390497048416012928086480326832803,  
454717651803304390605046474806214496349041928393838972128098083396
```

198416338265348561099990279626203818748780869911258542471083596997  
999137769172270582860904264845483493881389355042996092003778990527  
166633511886640963026727120785086013117258636782238741578611631963  
40391008634419348573975841578359355931590555,  
173641401820016949564655935332006237385901969902363408945541455625  
179249892087192454295576452549535276580492467375895382803320105330  
270624776842379332211986399489387842445104691388268081873656783225  
479920997152292186154759237548969603631388903315028112924271465957  
52813297603265829581292183917027983351121325,  
143881091049858084873377498760582844267478169619715814473806082779  
492002446603815705685311297750536842560718198372944360691335927725  
435827359858555062506609385742349587542113492152932816452053540699  
707901552370334360654345720206529556668557732320747494870076260503  
23967496732359278657193580493324467258802863,  
437949930831077282100409044765078509535664359041170635811923916666  
208942868556271923343561519699472876759322351922623506264767007785  
468703168104146263256689012959550643018860223875345033769144129304  
271690990169257097195507892469930687319198395350109334342324848296  
0643055943413031768521782634679536276233318,  
852564497767805912029282356628050332016845716489900429975570846580  
000670506721301527349119195816615239570759927616623152626850301152  
559383525400322971136156878159760393905377167078545699805166902465  
921129367969175040347114184654428933234394901710954471094573555988  
73230115172636184525449905022174536414781771,  
505765974585174515784312937469260994863882862461420124768141900309  
356894307260428104583448285639130010124157028761997082168750209971  
120896937596384549000925807466386310621179618766115458511576138357  
246350052537923161423792390476543929704153436946575803533332175470  
79551304961116837545648785312490665576832987,  
968687388303411123680946323374768402725637044085730544042137665004

```

075172518102124945158621763569169126271722804461412026616401912373
365687310693279061008961787762453116898579970121875991408759120265
896726299352678446969769808903807308675200710595723506679137103446
48377601017758188404474812654737363275994871,
488126165684663880062354966294339323436106182712861012004631564970
707824418031366106300439075082131709675428279687647969555864410849
231740766213144122425753727627496237202127358347850941635876470609
847184953603618492464059388890285944138847285682254145204118124433
7124767666161645827145408781917658423571721,
182379367263675566641714275754755964607273693682462861388042847421
242567003671332500786085371298779682878854574179578685805533719994
142274847376036889926209532001436880610240926235564710530064641232
051338946079238013719860274582743437378603954962605386631831938775
39815179246700525865152165600985105257601565]
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

```

:932917991253667068065456384757974305121049760661036102699380257099
522470200610908048701861952859987276802009798538487185891267657425
508559548052902535921442095521230621614585845750609394813682106886
298620369588576047074683723842780497413691535061826602648761154282
51983455344219194133033177700490981696141526

```

### Modular Square Root

```

from sympy.ntheory.residue_ntheory import legendre_symbol,
_sqrt_mod_tonelli_shanks as mod_sqrt
a =

```

```
847999465831677215194161651009712708755454127481243511200942577859
549535970024447040064240374705856680712781416539664021584419232790
045411625797948743201676932997076704673509124989867808806163479655
955670495984642413182041604843650138761721177012429279330807921415
317997762444043861695857505836119397568662004643987730833998929560
453786749368387277884392177130730560277639878697835386623166145337
605677197206977639899901376958893619485934494126822318419723136888
706060921287550751893617206070220955712443047713742184713068260166
696869165144723691701863490240770479732850946185484243201500987801
1354022108661461024768
```

p =

```
305318518619943332526759351114879506944143327639090835141337698613
509608950765046872613698157357425494287891383008430820865500590828
351414545266181606341099691954863220157759430300604495570900648119
401394317352091859964547391635559107264935972226468555064456029536
895274053622079269904423917050146047770386858805275374898453591015
524422928043984726423566093048106807315565420023015478466351014559
957325840713559030108567186807323373691284986552552770036436690316
945168513905059234167106012126184431098440415149424019696291589754
570790269063043287490399972629603012091581759200518906209470639363
47307238412281568760161
```

```
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:

```
281695125543112846143481618129074613954821952583885831257954988092
972261472141529076140556389177891903569175782597177921673029130079
279898417639772924344887826359642536777433420387485673330740435892
678962923730287247638080066977070703010353392917589989230660019859
```

277888085793300756719530360251917916219156401752424253903972126747  
973321328018828802235061772011688649204849935460172843388295120109  
220750186895053816428870429809715820583438750781788369658959872713  
920819264583922833549718236114238208656512834907615480533847317213  
916370643490217558998772245221613115612095307127021531635016235312  
90150340903913036821041

### 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]
# tích 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, v1
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 =
216326890219456009384369357217019970750178779749799846346212959223
997358146265162297828263751386527419937445280529263958626479131743
9029535926401109074800

```

```

q =
763823212045492587923155423401184234764101788821902117530421735871
587863618325243345489649067749651614988931674566460674949924142016
0898019203925115292257

e =
560569649525372066414288195690862430757067185847748211965743616366
366384473116903568234497428637904912373335600912567192428031253275
5241162267269123486523

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 format ‘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 = [01 ^ 02
for (01, 02) 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 để ra dc hình ảnh
n_image = np.bitwise_xor(n1, n2).astype(np.uint8)

Image.fromarray(n_image).save('n.png')
```

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



## 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.

```
PS C:\Users\user> python3 -i
a = 66528
b = 52920
res = 0
for i in range(1, b):
    if a%i == 0 and b%i == 0:
        res = max(res, i)
print(res);
```

```
PS C:\Users\user>
1512
```

## Extended GCD

- Viết hàm euclid mở rộng và xuất số có giá trị nhỏ hơn trong 2 số  $x$  và  $y$

```
def extendedEuclid(a, b):
    global temp, d, x, y
    if b == 0:
        d = a
        x = 1
        y = 0
    else:
        extendedEuclid(b, a%b)
        temp = x
        x = y
        y = temp - int(a / b) * y

extendedEuclid(26513, 32321)
print(min(x, y))
```

```
PS C:\Users\user>
-8404
```

## Modular Arithmetic 1

- Dùng phép toán modulo và hàm min trong python

```
print(min(11%6, 8146798528947%17))
```

```
PS C:\Users\DELL> & C:  
4
```

## 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  $\text{pow}(a, p) \% p = a \% p$ . Do đó ta có  $a^{(p-1)} = 1 \pmod{p}$

```
print(pow(273246787654,65536)%65537) 1
```

```
PS C:\Users\DELL> & C:/U
```

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

```
print(pow(3,11)%13)
```

```
PS C:\U  
9  
PS C:\U
```

## Privacy-Enhanced Mail?

```
from Crypto.PublicKey import RSA  
key=RSA.importKey(open('privacy_enhanced_mail_1f696c053d76a78c2c531bb013a92d4a.pem').read(  
))
```

Khóa bí mật là

‘1568270028805633136478717104581997365499114994919795992986086  
12281800217073168519244562055436655658108926741900598313302314  
36970914474774562714945620519144389785158908994181951348846017  
43250646416356496099378425415339540679910131476003344506519342  
95925123499520209829322185244623410021020634354893188133164645  
11621736943938440710470694912336237680219746204595128959161800  
59521636623753829644733537581887195252002699310214832889708354  
71842864932411915059536016688589411297909669092369411278513702



02421135897091086763569884760099112291072056970636380417349019  
579768748054760104838790424708988260443926906673795975104689'

## 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

22825373692019530804306212864609512775374171823993708516509897  
63154751363463585637562400373706803454904767799931094183745437  
88293513983023826296582640787754568386262075077254940306005168  
72852306191255492926495965536379271875310457319107936020730050  
47623527867152826581757143391956117566509617118975840613645398  
79662552369637826660669626546784649500759230603273586913566329  
08606498231755963567382339010985222623205586923466405809217426  
67033341001442990514694165229336621290373363008301639881088735  
60199774094673747422662762671375470215768742048095060459149644  
91063393800499167416471949021995447722415959979785959569497

## SSH Keys

```
key=RSA.importKey(open('bruce_rsa.pub').read())  
print(key.n)
```

dùng wget để tải file .pub 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('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 chuyển 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

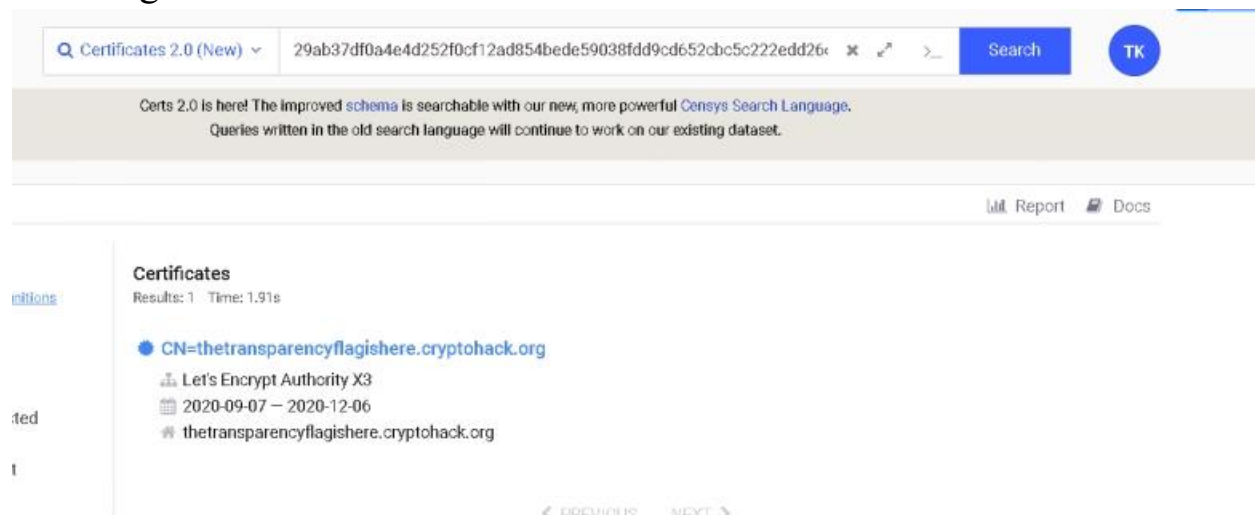
dùng `sha256_fingerprint = sha256.hexdigest()` để chuyển đổi giá trị băm SHA256 từ dạng byte sang dạng chuỗi hex và lưu vào biến

"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 dẫn vào thanh tìm kiếm ta sẽ tìm link của trang web



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

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



**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:

$$s[0] * x \bmod p = s[1]$$

$$s[1] * x \bmod p = s[2]$$

.....

$$s[n - 1] * x \bmod p = s[n]$$

Hệ phương trình trên tương đương:

$$x = s[0]^{-1} * s[1] \bmod p$$

$$x = s[1]^{-1} * s[2] \bmod p$$

.....

$$x = s[n - 1]^{-1} * s[n] \bmod p$$

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.

```

from Crypto.Util.number import inverse

s = [588, 665, 216, 113, 642, 4, 836, 114, 851, 492, 819, 237]

pmn = max(s) + 1

for p in range(pmn, 1000):
    a = pow(s[0], p - 2) * s[1] % p
    check = True
    firstVal = a
    for i in range(0, len(s) - 1):
        a = pow(s[i], p - 2) * s[i + 1] % p
        if firstVal != a:
            check = False
            break
    if check:
        print(p, firstVal)
        break

```

PS D:\VisualCode\Python> &

919 209

## Adrien's Signs

- Dùng Legendre Symbol vì nếu là quadratic residue thì bit có thể là 1 hoặc ngược lại.

```

arr = [67594220461269, 501237540280788, 718316769824518, 296304224247167, 48290626940198, 30829701196032, 521453693392074, 84098532438379]
haha = 288260533169915
base = 1007621497415251

cipher = ""
for i in arr:
    if (pow(i, (base - 1) // 2, base) == 1):
        cipher += "1"
    else:
        cipher += "0"

for i in range(0, 224, 8):
    binary = cipher[i:i + 8]
    print(chr(int(binary, 2)), end="")

```

PS D:\VisualCode\Python> & C:/Us

crypto{p4tterns\_1n\_re5idu3s}

## Modular Binomials

- Dùng factordb có sẵn để phân tích số n ra 2 thừa số nguyên tố p, q. Sau đó

```
from factordb.factordb import FactorDB

N = 1490556225784271405793272412957500282540539350265086976711594260640860034338032786625898240244799256498846658830517427167465784435245
e1 = 128866576673896608007807964629705049101939289928885189782000298269759786247186277992155647000960078499248666271549873650595243150976
e2 = 121105866739917884157803551396355790579209268648871103083432292560468682421794454448977901713513025751886071170815801214882535402157
c1 = 14010729418703228234352465883041270611113735889838753433295478495763409056136734155612156934673988344882629541204985909650438192052
c2 = 143869971386379788607482789869450986485071428645841111242025803651037931658116669876648512102300093752673989579794940668802964180133

f = FactorDB(N)
f.connect()
a = f.get_factor_list()

p = int(a[0])
q = int(a[1])
print(p, end = ",")
print(q)
```

PS D:\VisualCode\Python> & C:\Users\DELL\AppData\Local\Programs\Python\Python310\python.exe d:\VisualCode\Python\Modular\_Binomials.py  
1122740001692584863902620644419912006085563761274089527015149626443409218991960915575193827633565341063769064894451032551775935948989662501767736054  
3276598389710504779561947065915705709377140730916834567054141877242780714803920748990081001378367395798400626912065213400768927248451780539839027730  
8001719431273, 13276058780636530197147915707203144838013576579446678745694878673116809587795687529528266156548824219073159328266369472891494596725317  
3047324353981530949360031535707374701705328450856944598803228299967009004598984671293494375599408764139743217465012770376728876547958852025425539298  
410751132782632817947101601

## 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.

```
from sympy.ntheory.residue_ntheory import sqrt_mod

n = 2777285740987525752941599091121421197584430718443024145189940783875050302432336789554098160658670998598000343508211699588801773142663
e = 16
ct = 113031747618944311467356975694891347472349751441621721624016745672730348313919369163972340683461154591346024439636040636793792859193

for a in sqrt_mod(ct, n, all_roots=True):
    for b in sqrt_mod(a, n, all_roots=True):
        for c in sqrt_mod(b, n, all_roots=True):
            for d in sqrt_mod(c, n, all_roots=True):
                try:
                    print(bytes.fromhex(hex(d)[2:]).decode())
                except:
                    continue
```

PS D:\VisualCode\Python> & C:\Users\DELL\AppData\Local\Programs\Python\Python310\python.exe d:\VisualCode\Python\Broken\_RSA.py  
Hey, if you are reading this maybe I didn't mess up my code too much. Phew. I really should play more CryptoHack before rushing to code stuff from scratch again. Here's the flag: crypto{m0dular\_sqrt3\_root}

## 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:

$$v * b * b^{(-1)} \bmod n = v * a * a^{(-1)} \bmod n$$

$$v * a * b * v \bmod n = v * (a * b) * v \bmod n$$

$$v = va * vb * (vab)^{(-1)} \bmod n$$

Sau đó giải mã theo AES sẽ được cờ.



```

from Crypto.Cipher import AES
from hashlib import sha256
from Crypto.Util.number import long_to_bytes

p, q = (106999406481964110281707134307265594704271136897212028033926384579207714394528970322298383173216395995062838705859248070899415105
n = p*q
va = 124641741967121300068241280971408306625050636261192655845274494695382484894973990899018981438824398885984003880665335336872849819983
vavb = 1147782451840916775761340467246098682047711511114464578705248434143568974794737396272125524954133119854098295237009196035026166673
vb = 656889784012771314738234583279864566711023716801133564063044000658392310250365927310489958482763796192142867733518062042165471200051
c = 'fef29e5ff72f28160027959474fc462e2a9e0b2d84b1508f7bd0e270bc98fac942e1402aa12db6e6a36fb380e7b53323'

v = (va * vb * pow(vavb, -1, q)) % n

key = sha256(long_to_bytes(v)).digest()
print(AES.new(key, AES.MODE_ECB).decrypt(bytes.fromhex(c)))

```

PS D:\VisualCode\Python> & C:/Users/DELL/AppData/Local/Programs/Python/Python31  
b'crypto{1nv3rt1bl3\_k3y\_3xch4ng3\_pr0t0c0l}\x08\x08\x08\x08\x08\x08\x08\x08'

PS D:\VisualCode\Python> & C:/Users/DELL/AppData/Local/Programs/Python/Python31

## 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('13e4d200708b786d8f7c3bd2dc5de0201f0d7879192e6603d7c5d6b963e1df2943e3ff75f7fda
9c30a92171bbbc5acbf')
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 =
0x7fe8cafec59886e9318830f33747cafd200588406e7c42741859e15994ab62410438991ab5d9fc94f386219e3c
27d6ffc73754f791e7b2c565611f8fe5054dd132b8c4f3eadcf1180cd8f2a3cc756b06996f2d5b67c390adcba9d4
44697b13d12b2badfc3c7d5459df16a047ca25f4d18570cd6fa727aed46394576cfdb56b41
e = 0x10001
c =
0x5233da71cc1dc1c5f21039f51eb51c80657e1af217d563aa25a8104a4e84a42379040ecdffd5afa191156ccb4
0b6f188f4ad96c58922428c4c0bc17fd5384456853e139afde40c3f95988879629297f48d0efa6b335716a4c24b
fee36f714d34a4e810a9689e93a0af8502528844ae578100b0188a2790518c695c095c9d677b

p =
82398353972085161117203628479494254010456723658299376021174804493166945582266222001100575
35873802132963548914201468383545676262090246827792522994758916609
q =
10900824353334471830007307529937357926160386461967884446160315218630687793341471079170750
548554707926611542019859296605188535413447791710067186432371970369
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}`