**UNIVERSITY OF INFORMATION TECHNOLOGY**

**FALCUTY OF COMPUTER NETWORKS AND COMMUNICATIONS**

Ảnh có chứa Đồ họa, biểu tượng, thiết kế đồ họa, Phông chữ

Mô tả được tạo tự động**----------🙢🟔🙠----------**

**BÁO CÁO CUỐI KỲ**

**MÃ HÓA DỮ LIỆU PERSONAL HEALTH RECORD (PHR) BẰNG CP-ABE**

Giảng viên hướng dẫn : **TS. Nguyễn Ngọc Tự**

Thành viên trong nhóm:

**Hoàng Anh Khoa – 21522220**

**Đào Võ Hữu Hiệp – 21522065**

**Lê Thanh Lâm – 21521052**

Môn : **Mật Mã Học**

Lớp : **NT219.N21.ANTT**

**PHỤ LỤC**

[**Mở đầu** 3](#_Toc138529403)

[**Tổng quan** 3](#_Toc138529404)

[**1. Vấn đề cần giải quyết** 3](#_Toc138529405)

[**2. Ngữ cảnh** 4](#_Toc138529406)

[**3. Các bên liên quan** 4](#_Toc138529407)

[**4. Phân tích hệ thống** 5](#_Toc138529408)

[**5. Nhu cầu bảo mật** 5](#_Toc138529409)

[**Giải pháp** 5](#_Toc138529410)

[**Tài liệu tham khảo** 6](#_Toc138529411)

[**Đánh giá** 7](#_Toc138529412)

[**Triển khai** 7](#_Toc138529413)

[**Chạy mẫu** 19](#_Toc138529414)

# **Mở đầu**

PHR (Personal Health Record) là một công cụ quản lý thông tin sức khỏe cá nhân, tuy nhiên, cũng như các dịch vụ hiện thời, PHR cũng đang đối mặt với nhiều thách thức về bảo mật và quyền riêng tư.

Để giải quyết các vấn đề này, việc triển khai các cơ chế bảo mật và kiểm soát truy cập là rất quan trọng. Một trong những cơ chế đó là việc sử dụng các kỹ thuật mã hóa và kiểm soát truy cập tiên tiến để bảo vệ dữ liệu trong PHR.

Mã hóa có thể được sử dụng để bảo vệ dữ liệu trong PHR, từ quá trình thu thập, truyền tải và lưu trữ. Bằng cách mã hóa dữ liệu, thông tin sức khỏe cá nhân có thể được bảo vệ khỏi những kẻ tấn công tiềm năng. Ngoài ra, mã hóa dữ liệu trong PHR khi nó được lưu trữ có thể cung cấp một lớp bảo vệ bổ sung chống lại việc truy cập trái phép vào dữ liệu.

Kiểm soát truy cập là một khía cạnh quan trọng khác của việc bảo vệ dữ liệu trong PHR. Bằng cách thực hiện các chính sách kiểm soát truy cập chi tiết, có thể đảm bảo rằng chỉ có người dùng hoặc ứng dụng được ủy quyền mới có thể truy cập hoặc sửa đổi dữ liệu trong PHR. Điều này có thể bao gồm các kỹ thuật như kiểm soát truy cập dựa trên thuộc tính (ABAC), kiểm soát truy cập dựa trên vai trò (RBAC) hoặc các mô hình kiểm soát truy cập khác.

Tổng thể, việc sử dụng các cơ chế bảo mật tiên tiến như mã hóa và kiểm soát truy cập có thể cung cấp mức độ bảo vệ và kiểm soát cao hơn đối với dữ liệu trong PHR, giúp giải quyết các vấn đề liên quan đến bảo mật và quyền riêng tư. Khi PHR trở nên phổ biến hơn, việc ưu tiên bảo mật và triển khai các biện pháp bảo mật mạnh mẽ để bảo vệ thông tin sức khỏe cá nhân là rất cần thiết cho người dùng và các nhà cung cấp dịch vụ.

# **Tổng quan**

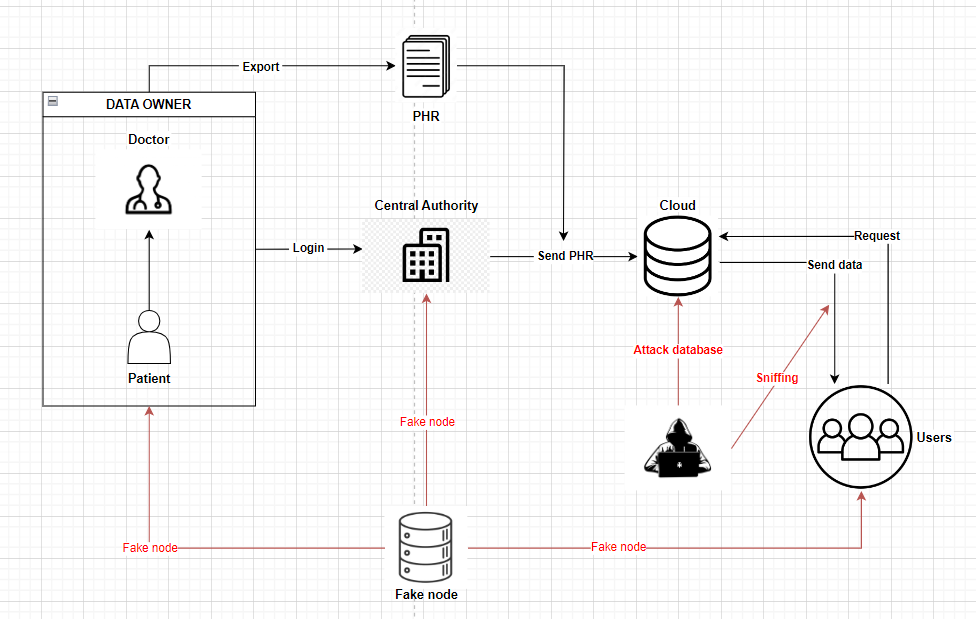
## **1. Vấn đề cần giải quyết**

Hiện nay, việc sử dụng PHR để ghi chép bệnh án và trao đổi bệnh án thông qua nền tảng số ngày càng phát triển và lan rộng. Và các lợi ích tiện lợi và khả năng mở rộng của nó đi kèm với nhu cầu về bảo mật dữ liệu cần được khắc phục và cải thiện.

Ví dụ: việc lưu trữ các thông tin nhạy cảm như bệnh án, kết quả xét nghiệm và các thông tin khác trong đám mây đặc biệt dễ bị phá hoại bởi các mối đe dọa về bảo mật như việc xâm nhập dữ liệu và mối đe dọa từ bên trong.

## **2. Ngữ cảnh**

Ngữ cảnh ở đây, một bệnh viện lưu trữ thông tin PHR của bệnh nhân và họ phải đối diện với những vấn đề về bảo mật thông tin từ bên ngoài và nội bộ



Hình 1: Hệ thống ban đầu

## **3. Các bên liên quan**

* **Data owner :** Data owner chịu trách nhiệm đảm bảo rằng dữ liệu của bệnh nhân được lưu trữ an toàn. Điều này bao gồm thực hiện các biện pháp thích hợp để mã hóa dữ liệu, kiểm soát quyền truy cập vào dữ liệu và giám sát truy cập trái phép hoặc vi phạm dữ liệu
* **Data user :** Trách nhiệm của người dùng dữ liệu là đảm bảo rằng họ truy cập và sử dụng dữ liệu tuân thủ các chính sách truy cập của chủ sở hữu dữ liệu, đồng thời thực hiện các biện pháp thích hợp để bảo đảm tính bảo mật và tính toàn vẹn của dữ liệu.
* **Database :** có trách nhiệm lưu trữ dữ liệu
* **Adversary :** Bất kỳ cá nhân, tổ chức nào tìm cách xâm phạm tính bảo mật của dữ liệu có trách nhiệm chấm dứt mọi hoạt động có thể làm tổn hại đến tính bảo mật của dữ liệu. Họ cũng phải chịu trách nhiệm về mọi thiệt hại hoặc tổn hại do hành động của mình gây ra và chịu trách nhiệm pháp lý nếu hành động của họ được cho là bất hợp pháp.

## **4. Phân tích hệ thống**

Hệ thống trên mô tả cách thức hoạt động của việc lưu trữ thông tin (PHR) ra bên thứ 3 lưu trữ (Database - Cloud). Việc đó dẫn đến 1 số rủi ro như sau:

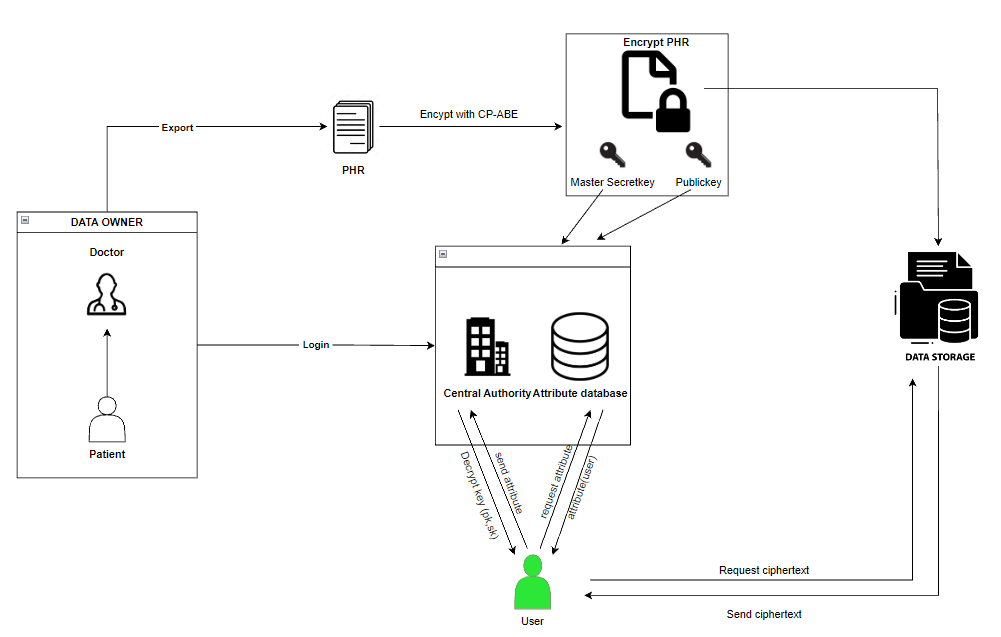
* Cloud (Semi-Trust): Là một nguồn không chắc tin cậy. Bên Cloud có thể bán hay làm rò rỉ thông tin của mình ra bên ngoài.
* Attacker có thể tấn công vào nơi lưu trữ hoặc đường truyền để lấy bản rõ.
* Sẽ có những node mạo danh Data-Owner để đăng dữ liệu không phù hợp, hay là mạo danh user để lấy PHR khi không có quyền truy cập,...

## **5. Nhu cầu bảo mật**

* **Tính bảo mật :** Đảm bảo rằng dữ liệu được lưu tại database được bảo vệ khỏi những truy cập và tiết lộ trái phép.
* **Tính toàn vẹn :** Đảm bảo rằng dữ liệu được lưu sẽ không bị can thiệp hoặc sửa đổi trái phép
* **Ủy quyền :** Đảm bảo rằng người dùng truy cập dữ liệu trên database có mức độ quyền và đặc quyền truy cập phù hợp

# **Giải pháp**

Dựa trên các vấn đề đã đặt ra ở trên nhóm em đề xuất 1 mô hình lưu trữ như sau:



Hình 2: Hệ thống cải tiến

* DataOwner sẽ là bên mã hoá PHR và đẩy lên Cloud. Master secret key (msk) và public key (pk) được tạo ra trong quá trình mã hoá sẽ được lưu trữ tại Central Authority (CA).
* Khi người dùng (user) yêu cầu bản PHR nào, phía Cloud sẽ gửi PHR dạng ciphertext cho user, đồng thời CA xác định xem các thuộc tính (attributes) của user có thoả mãn chính sách mã hoá hay không, nếu thoả, CA sẽ gửi Decrypt key để user có thể giải mã và xem được PHR ở dạng bản rõ.

⇒ Từ đó, mô hình lưu trữ mà nhóm em đưa ra đã giải quyết được các vấn đề bảo mật còn tồn đọng trong hệ thống cũ như sau:

* Vì PHR đã được mã hoá bằng CP-ABE ngay từ phía DataOwner và bản mã được truyền lên Cloud lưu trữ, từ đó ngăn Attacker tấn công vào nơi lưu trữ hoặc can thiệp vào đường truyền để lấy bản rõ.
* Như đã nói ở trên, Cloud(Semi-Trust) lưu trữ PHR nhưng ở dạng ciphertext, đồng thời Cloud cũng không lưu trữ Decrypt key, nên dữ liệu lộ ra từ bên phía Cloud không thể đọc được ở dạng bản rõ, từ đó ngăn được việc Cloud có thể bán hay làm rò rỉ thông tin PHR ra bên ngoài.
* Chỉ có những user có tập thuộc tính (attributes) thoả chính sách truy cập mới có thể có được PHR bản rõ, do đó ngăn được việc có những node mạo danh user để lấy PHR khi không có quyền truy cập.
* Việc thiết lập kết nối TLS giúp ngăn ngừa các cuộc tấn công mạo danh bằng cách sử dụng certificate và xác thực hai bên (mutual authentication). Trong quá trình trao đổi này, DataOwner và user ngoài sẽ sử dụng certificate để xác định danh tính của mình và của CA. Nếu certificate không hợp lệ hoặc không khớp với danh tính được khai báo, kết nối sẽ bị từ chối, và từ đó có thể ngăn được các node mạo danh DataOwner để đăng dữ liệu không phù hợp ; và đồng thời việc này giúp user có thể xác thực danh tính CA.

# **Tài liệu tham khảo**

[1] Zhang, L., Ye, Y., & Mu, Y. (2020). Multiauthority access control with anonymous authentication for personal health record. IEEE Internet of Things Journal, 8(1), 156-167.

[2] Agrawal, S., & Chase, M. (2017, October). FAME: fast attribute-based message encryption. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 665-682).

[3] Bethencourt, J., Sahai, A., & Waters, B. (2007, May). Ciphertext-policy attribute-based encryption. In 2007 IEEE symposium on security and privacy (SP'07) (pp. 321-334). IEEE.

# **Đánh giá**

"CP-ABE là một hệ mật mã công khai, nơi mỗi người dùng được liên kết với một tập hợp các thuộc tính. Một thông điệp được mã hóa bằng cách sử dụng các thuộc tính của người dùng như là một chính sách để xác định ai có thể giải mã thông điệp. Một người dùng có thể giải mã thông điệp nếu tất cả các thuộc tính của người dùng đó khớp với chính sách được sử dụng để mã hóa thông điệp."[2]

Trong đồ án môn học này, nhóm em sử dụng scheme AC17 trong CP-ABE. Scheme này so với các scheme ra đời trước như BSW07, waters11,.. có một số cải tiến, chẳng hạn như: không hạn chế kích thước của các chính sách hoặc bộ thuộc tính; cho phép bất kỳ chuỗi tùy ý nào được sử dụng làm thuộc tính; thời gian chạy của các thuật toán mã hoá và giải mã nhanh hơn,…Cụ thể trong scheme AC17 thời gian giải mã chỉ là 0,06 giây ngay cả khi có tới 100 thuộc tính tham gia, trong khi BSW07 mất hơn 2 giây. Bản mã và khóa được tạo từ các thuật toán trong scheme AC17 cũng nhỏ hơn 25% so với BSW07.

# **Triển khai**

**Môi trường thực hiện:** Linux

**Ngôn ngữ lập trình:** Python

**Thư viện sử dụng chủ yếu:** Charm-Crypto

**Chọn scheme phù hợp:** sử dụng scheme AC17 CP-ABE

**Thiết lập kết nối** bằng TLS

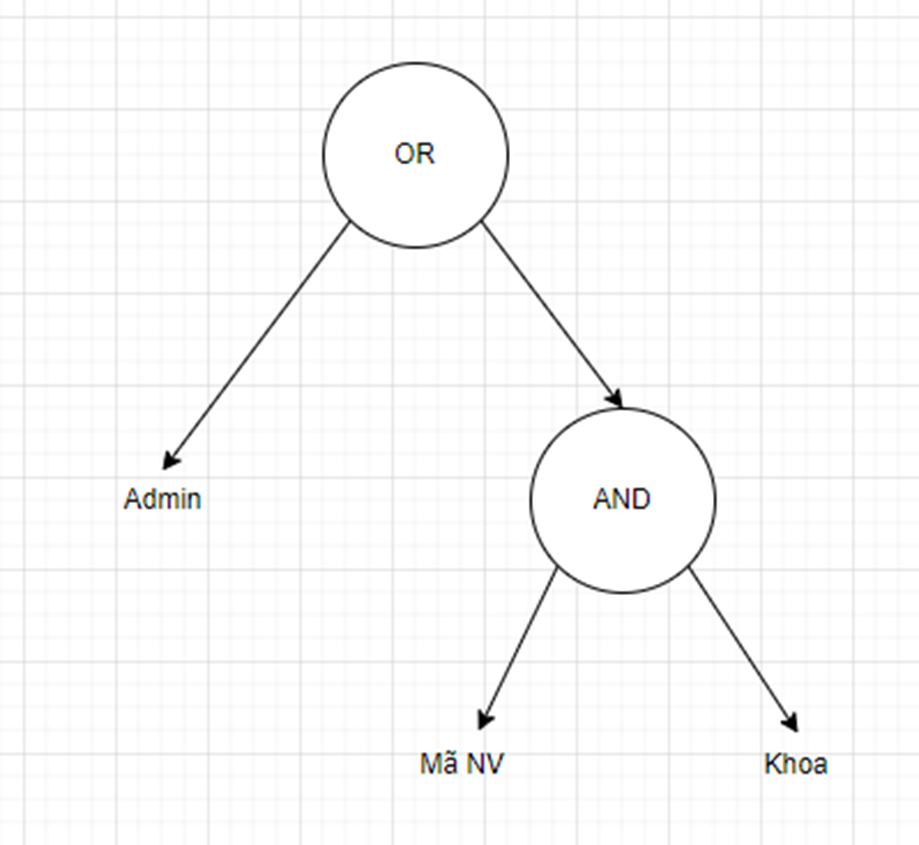
**Triển khai certificate trong kết nối TLS:** Certificate được ký bằng thuật toán ECDSA

**Data Storage** (chứa ciphertext): Google Firebase (Cloud Firestore)

**Attribute Database** (chứa thông tin login): Google Firebase (Realtime Database)

**PHR:** dạng cấu trúc JSON

**Chính sách truy cập (Policy) :**



**Giải thích thuật toán CP-ABE:** (Giả sử kích thước giả định tuyến tính (assump\_size) =2):

**Bước 1:** Setup

A picture containing text, font, screenshot, line

Description automatically generated

* Tạo các giá trị *a*và *d* (trong đoạn code dưới thì *d* là *k*:

|  |
| --- |
| A = []  B = []  **for** i **in** range(self.assump\_size):  A.append(self.group.random(ZR))  B.append(self.group.random(ZR)) # note that A, B are vectors here  # vector  k = []  **for** i **in** range(self.assump\_size + **1**):  k.append(self.group.random(ZR)) |

* Tính toán  và  hay chính là các thành phần trong public key (pk):

|  |
| --- |
| # pick a random element from the two source groups and pair them  g = self.group.random(G1)  h = self.group.random(G2)  e\_gh = pair(g, h)  # compute the [A]\_2 term  h\_A = []  **for** i **in** range(self.assump\_size):  h\_A.append(h \*\* A[i])  h\_A.append(h)  e\_gh\_kA = []  **for** i **in** range(self.assump\_size):  e\_gh\_kA.append(e\_gh \*\* (k[i] \* A[i] + k[self.assump\_size]))  # the public key  pk = {'h\_A': h\_A, 'e\_gh\_kA': e\_gh\_kA} |

* Tính toán msk gồm: 

|  |
| --- |
| # compute the e([k]\_1, [A]\_2) term  g\_k = []  **for** i **in** range(self.assump\_size + **1**):  g\_k.append(g \*\* k[i])  # the master secret key  msk = {'g': g, 'h': h, 'g\_k': g\_k, 'A': A, 'B': B} |

**Bước 2:** keygen

A picture containing text, font, screenshot, line

Description automatically generated

* Tính toán:  :

|  |
| --- |
| # pick randomness  r = []  sum = **0**  **for** i **in** range(self.assump\_size):  rand = self.group.random(ZR)  r.append(rand)  sum += rand  # compute the [Br]\_2 term  # first compute just Br as it will be used later too  Br = []  **for** i **in** range(self.assump\_size):  Br.append(msk['B'][i] \* r[i])  Br.append(sum)  # now compute [Br]\_2  K\_0 = []  **for** i **in** range(self.assump\_size + **1**):  K\_0.append(msk['h'] \*\* Br[i]) |

* Tính toán: 

|  |
| --- |
| # compute [W\_1 Br]\_1, ...  K = {}  A = msk['A']  g = msk['g']  **for** attr **in** attr\_list:  key = []  sigma\_attr = self.group.random(ZR)  **for** t **in** range(self.assump\_size):  prod = **1**  a\_t = A[t]  **for** l **in** range(self.assump\_size + **1**):  input\_for\_hash = attr + str(l) + str(t)  prod \*= (self.group.hash(input\_for\_hash, G1) \*\* (Br[l]/a\_t))  prod \*= (g \*\* (sigma\_attr/a\_t))  key.append(prod)  key.append(g \*\* (-sigma\_attr))  K[attr] = key |

* Tính toán: 

|  |
| --- |
| # compute [k + VBr]\_1  Kp = []  g\_k = msk['g\_k']  sigma = self.group.random(ZR)  **for** t **in** range(self.assump\_size):  prod = g\_k[t]  a\_t = A[t]  **for** l **in** range(self.assump\_size + **1**):  input\_for\_hash = '01' + str(l) + str(t)  prod \*= (self.group.hash(input\_for\_hash, G1) \*\* (Br[l] / a\_t))  prod \*= (g \*\* (sigma / a\_t))  Kp.append(prod)  Kp.append(g\_k[self.assump\_size] \* (g \*\* (-sigma))) |

* Output: 

|  |
| --- |
| **return** {'attr\_list': attr\_list, 'K\_0': K\_0, 'K': K, 'Kp': Kp} |

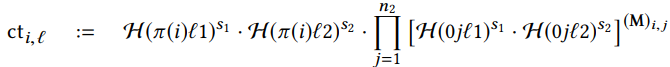
**Bước 3:** Encrypt

A picture containing text, font, screenshot, line

Description automatically generated

* Tính toán  :

|  |
| --- |
| policy = self.util.createPolicy(policy\_str)  mono\_span\_prog = self.util.convert\_policy\_to\_msp(policy)  num\_cols = self.util.len\_longest\_row  # pick randomness  s = []  sum = **0**  **for** i **in** range(self.assump\_size):  rand = self.group.random(ZR)  s.append(rand)  sum += rand  # compute the [As]\_2 term  C\_0 = []  h\_A = pk['h\_A']  **for** i **in** range(self.assump\_size):  C\_0.append(h\_A[i] \*\* s[i])  C\_0.append(h\_A[self.assump\_size] \*\* sum) |

* Tính toán:

|  |
| --- |
| # compute the [(V^T As||U^T\_2 As||...) M^T\_i + W^T\_i As]\_1 terms  # pre-compute hashes  hash\_table = []  **for** j **in** range(num\_cols):  x = []  input\_for\_hash1 = '0' + str(j + **1**)  **for** l **in** range(self.assump\_size + **1**):  y = []  input\_for\_hash2 = input\_for\_hash1 + str(l)  **for** t **in** range(self.assump\_size):  input\_for\_hash3 = input\_for\_hash2 + str(t)  hashed\_value = self.group.hash(input\_for\_hash3, G1)  y.append(hashed\_value)  # if debug: print ('Hash of', i+2, ',', j2, ',', j1, 'is', hashed\_value)  x.append(y)  hash\_table.append(x)  C = {}  **for** attr, row **in** mono\_span\_prog.items():  ct = []  attr\_stripped = self.util.strip\_index(attr) # no need, re-use not allowed  **for** l **in** range(self.assump\_size + **1**):  prod = **1**  cols = len(row)  **for** t **in** range(self.assump\_size):  input\_for\_hash = attr\_stripped + str(l) + str(t)  prod1 = self.group.hash(input\_for\_hash, G1)  **for** j **in** range(cols):  # input\_for\_hash = '0' + str(j+1) + str(l) + str(t)  prod1 \*= (hash\_table[j][l][t] \*\* row[j])  prod \*= (prod1 \*\* s[t])  ct.append(prod)  C[attr] = ct |

* Tính toán: 

|  |
| --- |
| # compute the e(g, h)^(k^T As) . m term  Cp = **1**  **for** i **in** range(self.assump\_size):  Cp = Cp \* (pk['e\_gh\_kA'][i] \*\* s[i])  Cp = Cp \* msg |

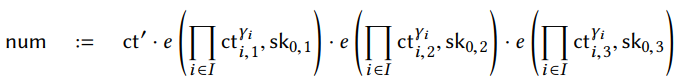
* Output ciphertext: 

|  |
| --- |
| **return** {'policy': policy, 'C\_0': C\_0, 'C': C, 'Cp': Cp} |

**Bước 4:** Decrypt

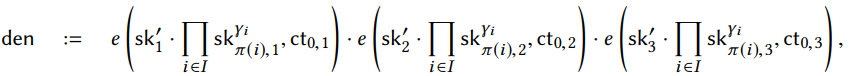
A picture containing text, font, screenshot, line

Description automatically generated

* Tính toán 

|  |
| --- |
| nodes = self.util.prune(ctxt['policy'], key['attr\_list'])  **if** **not** nodes:  print ("Policy not satisfied.")  **return** **None**  prod1\_GT = **1**  prod2\_GT = **1**  **for** i **in** range(self.assump\_size + **1**):  prod\_H = **1**  prod\_G = **1**  **for** node **in** nodes:  attr = node.getAttributeAndIndex()  attr\_stripped = self.util.strip\_index(attr) # no need, re-use not allowed  # prod\_H \*= key['K'][attr\_stripped][i] \*\* coeff[attr]  # prod\_G \*= ctxt['C'][attr][i] \*\* coeff[attr]  prod\_H \*= key['K'][attr\_stripped][i]  prod\_G \*= ctxt['C'][attr][i]  prod1\_GT \*= pair(key['Kp'][i] \* prod\_H, ctxt['C\_0'][i]) |

và



|  |
| --- |
| prod2\_GT \*= pair(prod\_G, key['K\_0'][i]) |

* Output: 

|  |
| --- |
| **return** ctxt['Cp'] \* prod2\_GT / prod1\_GT |

**Dữ liệu mẫu:**

|  |
| --- |
| {  "ID": "BN001",  "ten": "Nguyen Van Dat",  "ngay\_sinh": "01/01/2003",  "gioi\_tinh": "Nam",  "nhom\_mau": "AB",  "chieu\_cao": **175**,  "can\_nang": **70**,  "NGUOIPHUTRACH": [  {  "ID": "BS001",  "khoa": "Pharmacy"  },  {  "ID": "BS005",  "khoa": "Respiratory"  }  ],  "lich\_su\_benh\_an": [  {  "thoi\_gian": "2010",  "mo\_ta": "Phau thuat ruot thua"  },  {  "thoi\_gian": "2018",  "mo\_ta": "Dieu tri viem xoang"  }  ],  "thuoc\_dang\_dung": [  {  "ten\_thuoc": "Amlodipine",  "lieu\_luong": "5mg",  "lieu\_lan": "1 lan/ngay",  "ngay\_bat\_dau": "01/01/2022",  "ngay\_ket\_thuc": ""  },  {  "ten\_thuoc": "Losartan",  "lieu\_luong": "50mg",  "lieu\_lan": "1 lan/ngay",  "ngay\_bat\_dau": "01/01/2022",  "ngay\_ket\_thuc": ""  },  {  "ten\_thuoc": "Metformin",  "lieu\_luong": "500mg",  "lieu\_lan": "2 lan/ngay",  "ngay\_bat\_dau": "01/01/2022",  "ngay\_ket\_thuc": ""  },  {  "ten\_thuoc": "Insulin",  "lieu\_luong": "10 units",  "lieu\_lan": "2 lan/ngay",  "ngay\_bat\_dau": "01/01/2022",  "ngay\_ket\_thuc": ""  }  ],  "tiem\_phong": [  {  "ten\_tiem\_phong": "Phong benh cum",  "ngay\_tiem": "01/01/2022",  "lieu\_luong": "1 lieu",  "noi\_tiem": "Benh vien ABC"  }  ],  "thong\_tin\_lien\_he": {  "dia\_chi": "So 123, duong ABC, phuong DEF, quan GHI, TP. HCM",  "so\_dien\_thoai": "09xxxxxxxxxx",  "email": "nguyenvandat@gmail.com"  }  } |

**Coding phần mã hóa:**

|  |
| --- |
| # Hàm mã hoá msg  **def ABEencryption**(self, filename, pk, policy):  # Mở file chứa PHR cần mã hoá  msg = **open**(filename,"rb").read()  serialize\_encoder = **ac17**.**Serialize**()  # Tạo key AES dùng dùng để mã hoá msg  key = self.groupObj.random(GT)  # Mã hoá key AES bằng CP-ABE  encrypt\_key = self.cpabe.encrypt(pk, key, policy)  #Đóng gói key  encrypt\_key\_byte = serialize\_encoder.jsonify\_ctxt(encrypt\_key)  encrypt\_key\_byte = **base64**.b64encode(encrypt\_key\_byte.encode())  encrypt\_key\_size = len(encrypt\_key\_byte)  stream = struct.pack('Q',encrypt\_key\_size)  # Mã hoá msg bằng aes với key vừa tạo  aes\_key = **hashlib**.sha256(**str**(key).encode()).digest()  iv = **os**.urandom(**16**)  encryptor = **AES**.new(aes\_key,AES.MODE\_CFB,iv)  encrypted\_data = encryptor.encrypt(msg)  # Xuất ra output để gửi đi  output = stream + iv + encrypt\_key\_byte + encrypted\_data  **return** output |

Đoạn code trên sử dụng CP-ABE để mã hóa key ngẫu nhiên được sinh ra. Dùng key băm ra để làm key cho AES mã hóa file dữ liệu chính. Output của hàm trên gồm có 1 chuỗi đã được đóng gói bao gồm: stream(Đánh dấu) + iv (Vector khởi tạo) + encrypt\_key\_byte (bản mã của key được sinh ngẫu nhiên đã mã hóa bằng CP-ABE) + encrypted\_data (bản mã của file ban đầu đã mã hóa bằng AES).

**Coding phần giải mã:**

|  |
| --- |
| # Hàm giải mã ciphertext  **def ABEdecryption**(self, filename, pk, sk):  serialize\_encoder = **ac17**.**Serialize**()  # Mở file chứa input gồm (stream + iv + encrypt\_key\_byte + encrypted\_data) và tách các trường  ciphertext\_stream = **open**(filename,"rb")  encrypt\_key\_size = **struct**.unpack('Q',ciphertext\_stream.read(struct.calcsize('Q')))[**0**]  ciphertext\_stream.close()  ciphertext = **open**(filename,"rb").read()  iv = ciphertext[**8**:**24**]  encrypt\_key\_byte = ciphertext[**24**:encrypt\_key\_size+**24**]  encrypt\_key\_byte = **base64**.b64decode(encrypt\_key\_byte)  encrypt\_key = serialize\_encoder.unjsonify\_ctxt(encrypt\_key\_byte)  # Giải mã key AES  key = self.cpabe.decrypt(pk,encrypt\_key,sk)  # Giải mã ciphertext được mã hoá bằng AES từ key đã giải mã trên  **if**(key):  aes\_key = **hashlib**.sha256(str(key).encode()).digest()  encryptor = **AES**.new(aes\_key,AES.MODE\_CFB,iv)  decrypted\_data = encryptor.decrypt(ciphertext[**8**+**16**+encrypt\_key\_size:])  **return** decrypted\_data  **else**:  **return** None |

Đoạn code trên sẽ nhận ciphertext (dữ liệu mà đã được đóng gói ở trên) vào để giải mã. Đầu tiên sẽ đọc stream để phân tách các trường trong đó ra. Sau đó giải mã key bằng public key và secret key để lấy lại key. Key này băm ra để giải mã ciphertext được mã hóa bằng AES.

**Coding Central Authority**

* Lắng nghe user (Port 62345)

|  |
| --- |
| **def** **handle\_user\_62345**(conn, addr):  **try**:  **print**(f'**\n**---Connected by user---')  # Nhận dữ liệu từ client (Attr + request)  json\_str = conn.recv(**1024**)  json\_data = **json**.loads(json\_str)  **print**("Received data!")  #Khởi tạo tên filename  mkName = "./Center\_Autho/msk" + json\_data["request"] + ".pem"  pkName = "./Center\_Autho/pk" + json\_data["request"] + ".pem"    #Tiến hành tạo secret key (private key)  **print**("Preparing the encryption key...")  abe = cp\_abe.CP\_ABE()  key = **Serialize**.**Serialize**()  attr\_list = [json\_data['ID'].upper(), json\_data["Faculty"].upper()]  mk = key.load\_file\_mk(mkName)  pk = key.load\_file\_pk(pkName)  sk = abe.PrivateKeyGen(pk, mk, attr\_list)  sk\_bytes = key.jsonify\_sk(sk)  sk\_bytes = **base64**.b64encode(sk\_bytes.encode())  pk\_bytes = key.jsonify\_pk(pk)  pk\_bytes = **base64**.b64encode(pk\_bytes.encode())  # Gửi pk+sk  conn.sendall(pk\_bytes+sk\_bytes)  **print**("Sent the key")  conn.close()  **print**("Connection closed")  **except**:  **print**("ERROR") |

Đoạn code trên sẽ tiến hành nhận dữ liệu từ user, sau đó đọc các attributes của user và tiến hành tạo secret key (sk) từ master key (mk), public key (pk) và attributes, sau đó gửi pk và sk đến user. Không cần kiểm tra thuộc tính vì khi thuộc tính sai đã vi phạm policy và không thể truy cập vào PHR được

* Lắng nghe Data Owner (Port 8888)

|  |
| --- |
| **def** **handle\_data\_owner\_8888**(conn, addr):  **try**:  **print**(f'**\n**---Connected by data owner---')  # Nhận dữ liệu từ client  index = conn.recv(**1024**)  **print**(index)  **print**("Received index!")  #Khởi tạo tên filename  mkName = "./Center\_Autho/msk" + index.decode('utf-8') + ".pem"  pkName = "./Center\_Autho/pk" + index.decode('utf-8') + ".pem"  # Đợi phản hồi từ server  key = **Serialize**.**Serialize**()  # Nhận phản hồi  response = ''  **while** **True**:  data = conn.recv(**1024**)  response += data.decode('utf-8')  **if** **len**(data) < **1024**:  **break**  #Tách public key và master key  response1 = response[:**880**]  response2 = response[**880**:]  print("Received key!")  pk\_bytes = **base64**.b64decode(response1)  pk = key.unjsonify\_pk(pk\_bytes)  mk\_bytes = **base64**.b64decode(response2)  mk = key.unjsonify\_mk(mk\_bytes)  key.save\_file\_pk(pk, pkName)  key.save\_file\_mk(mk, mkName)  **print**('Finished')  **except**:  **print**("ERROR") |

Đoạn code trên sẽ tiến hành nhận dữ liệu từ data owner đến Central Authority. Nhận dữ liệu sẽ gồm index (Là số thứ tự của bản PHR muốn lưu trữ) và 1 đoạn byte (pk và msk) sau đó tiến hành tách ra thành pk và msk. Tiến hành lưu trữ các dữ liệu vừa nhận được.

**Coding kết nối giữa User tới Central Authority**

|  |
| --- |
| **def connect\_returnPlt**(self, json\_data, request, ciphertextName):  **try**:  HOST = '127.0.0.1'  PORT =**62345**  context = **ssl**.create\_default\_context()  context.check\_hostname = **False**  context.load\_verify\_locations('server.crt')  #Thiết lập kết nối  **with** **socket**.create\_connection((HOST, PORT)) **as** sock:  **with** context.wrap\_socket(sock, server\_hostname=HOST) **as** client\_socket:  #Tạo dữ liệu và gửi đi  json\_data["request"] = request  json\_str = **json**.dumps(json\_data)  client\_socket.sendall(json\_str.encode('utf-8'))  key = **Serialize**.**Serialize**()  # Nhận phản hồi  response = ''  **while** **True**:  **try**:  data = client\_socket.recv(**1024**)  response += data.decode('utf-8')  **if** **len**(data) < **1024**:  **break**  **except** socket.timeout:  **print**('Timeout occurred')  **break**    #Tách bytes  response1 = response[:**880**]  response2 = response[**880**:]  #Lấy pk  pk\_bytes = **base64**.b64decode(response1)  pk = key.unjsonify\_pk(pk\_bytes)  #Lấy sk  sk\_bytes = **base64**.b64decode(response2)  sk = key.unjsonify\_sk(sk\_bytes)  # Đóng kết nối  client\_socket.close()  #Giải mã  **print**('Decrypting file...')  abe = **cp\_abe**.**CP\_ABE**()  plt = abe.ABEdecryption(ciphertextName, pk, sk)  **return** plt  **except**:  **print**("ERROR")  **return** None |

Đoạn code trên, user sẽ mở kết nối TLS gửi lên server các attribute dưới dạng json sau đó nhận dữ liệu (gồm pk, sk) sau đó tách byte của dữ liệu nhận được thành pk ,sk để decrypt ciphertext

**Coding Data Owner**

|  |
| --- |
| HOST = 'localhost'  PORT = **8888**  context = **ssl**.create\_default\_context()  context.check\_hostname = **False**  context.load\_verify\_locations('server.crt')  **with** **socket**.create\_connection((HOST, PORT)) **as** sock:  **with** context.wrap\_socket(sock, server\_hostname=HOST) **as** owner\_socket:  #Nhập tên file và lấy index  fileName = **input**("Enter name of PHR file (in JSON format): ")  **if** **not** fileName.endswith('.json'):  **print**("Invalid input file")  exit(**1**)  collection\_ref = self.db.collection('Ciphertext')  docs = collection\_ref.get()  count = **len**(docs)  index = "" + **str**(count+**1**)  owner\_socket.sendall(index.encode('utf-8'))  #Đọc file  sourcefile = **open**(fileName, 'rb')  msg = sourcefile.read()  sourcefile.close()  msg\_dict = **json**.loads(msg)  #Lấy thuộc tính và policy  policy = '((' + msg\_dict["ID"] + ') or ('  **for** item **in** msg\_dict['NGUOIPHUTRACH']:  **if** msg\_dict['NGUOIPHUTRACH'][-**1**] != item:  policy += "(" + item['ID'] + ' and ' + item['khoa'].upper() + ")" + " or "  **else**:  policy += "(" + item['ID'] + ' and ' + item['khoa'].upper() + ")" + '))'  #Chuẩn bị gửi khóa đến Center Authority  print("Sent to server....")  #Tạo pk, msk và ciphertext  abe = **cp\_abe**.**CP\_ABE**()  key = **Serialize**.**Serialize**()  pk, mk = abe.KeyGen()  cipher = abe.ABEencryption(fileName, pk, policy)  #Gửi pk và msk đến Center Authority  pk\_bytes = key.jsonify\_pk(pk)  pk\_bytes = base64.b64encode(pk\_bytes.encode())  mk\_bytes = key.jsonify\_mk(mk)  mk\_bytes = **base64**.b64encode(mk\_bytes.encode())  owner\_socket.sendall(pk\_bytes+mk\_bytes)  #Gửi ciphertext lên cloud  cipherName = 'phr'+ index +'.json.crypt'  doc\_ref = self.db.collection(u'Ciphertext').document(cipherName)  doc\_ref.set({  u'Data': cipher  })  **print**("Add successfully!")  owner\_socket.close()  **return** **True** |

Đoạn code trên sẽ tiến hành khởi tạo kết nối TLS giữa DataOwner và Central Authority, nếu kết nối thành công, DataOwner sẽ tiến hành mã hoá file PHR được chọn, sau đó gửi pk và msk đến Central Authority, đồng thời đưa ciphertext lên cloud.

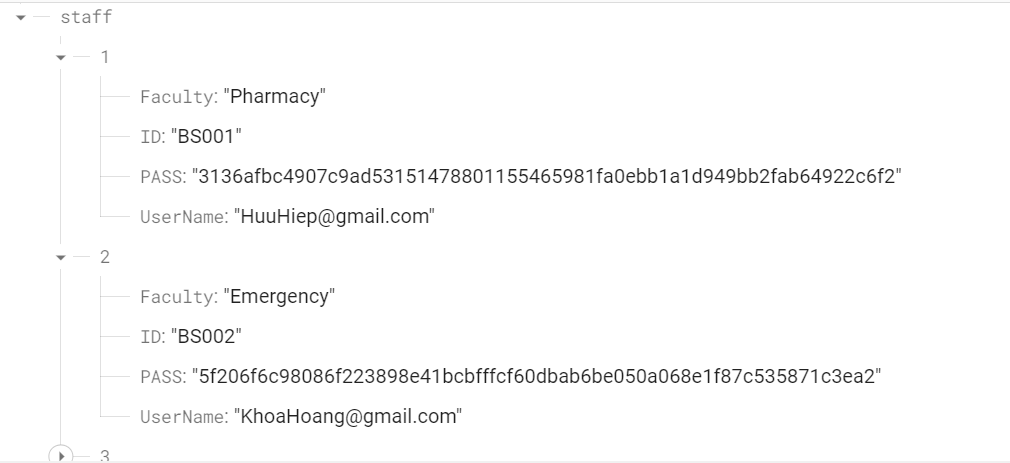
**Coding sinh certificate**

|  |
| --- |
| **def Self\_signed\_certificate**(self):  # Tạo một cặp khóa ECC với đường cong prime256v1  ec\_key = **ec**.generate\_private\_key(ec.SECP256R1(), openssl.backend)  # Chuyển đổi khóa ECC sang định dạng OpenSSL  key = **crypto**.PKey.from\_cryptography\_key(ec\_key)  # Tạo chứng chỉ x509 tự ký  cert = **crypto**.X509()  cert.get\_subject().CN = '127.0.0.1'  cert.get\_subject().countryName = 'VN'  cert.get\_subject().O = 'UIT'  cert.set\_serial\_number(**1000**)  cert.gmtime\_adj\_notBefore(**0**)  cert.gmtime\_adj\_notAfter(**365** \* **24** \* **60** \* **60**)  cert.set\_issuer(cert.get\_subject())  cert.set\_pubkey(key)  cert.sign(key, 'sha256')  # Giả định phân phát cert cho user và phân phối cert và key cho Center Authority  **with** **open**("./Center\_Autho/server.key", "wb") **as** key\_file:  key\_file.write(crypto.dump\_privatekey(**crypto**.FILETYPE\_PEM, key))    **with open**("./Center\_Autho/server.crt", "wb") **as** cert\_file:  cert\_file.write(crypto.dump\_certificate(**crypto**.FILETYPE\_PEM, cert))    **with open**("server.crt", "wb") **as** cert\_file:  cert\_file.write(crypto.dump\_certificate(**crypto**.FILETYPE\_PEM, cert)) |

Sử dụng curve prime256v1, random tạo ra một cặp khóa ECC sau đó chuyển sang định dạng OpenSSL, tạo chứng chỉ x509 xong phân phối cert cho user và phân phối key cho CA

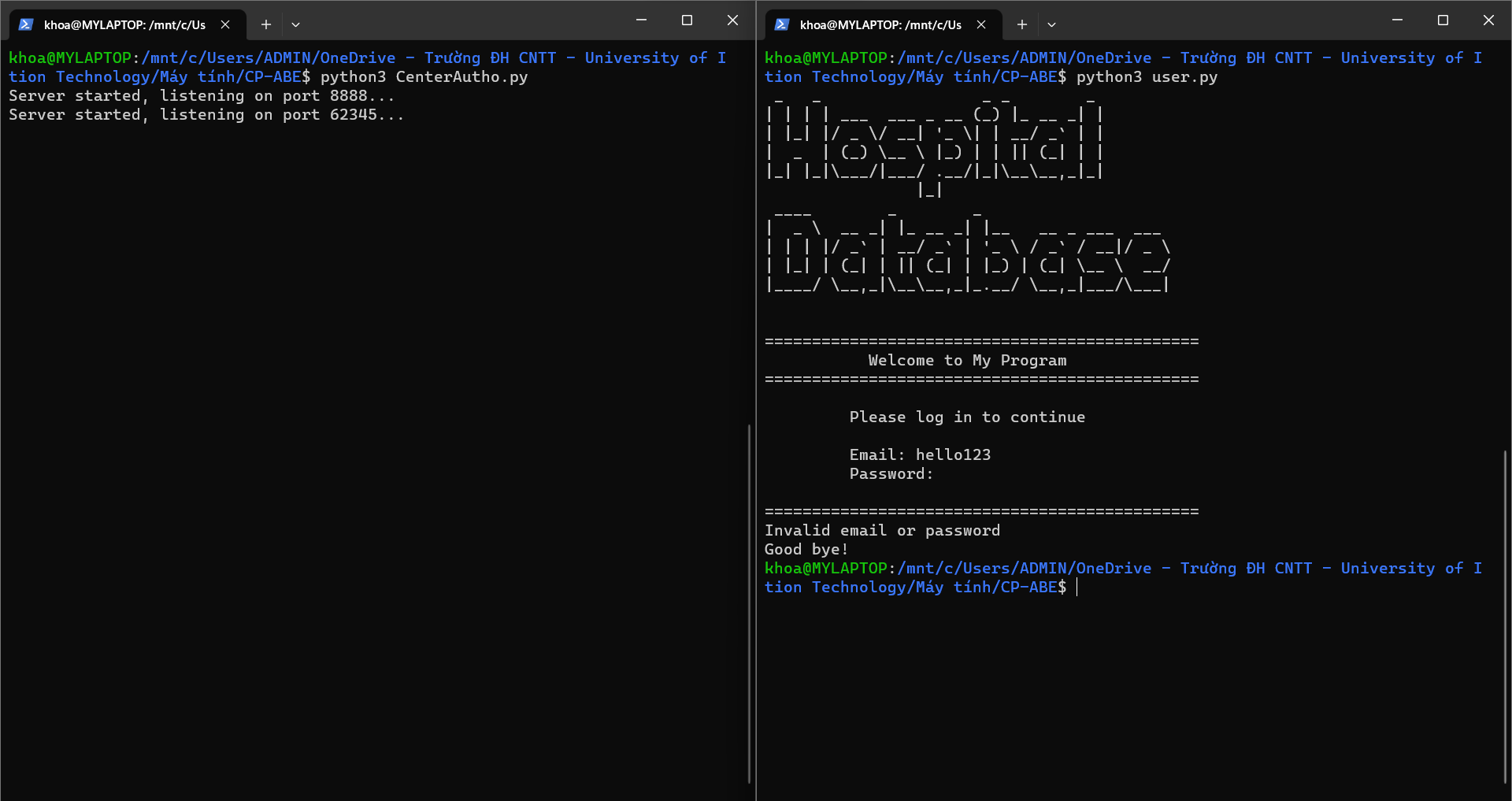
# **Chạy mẫu**

User mẫu:

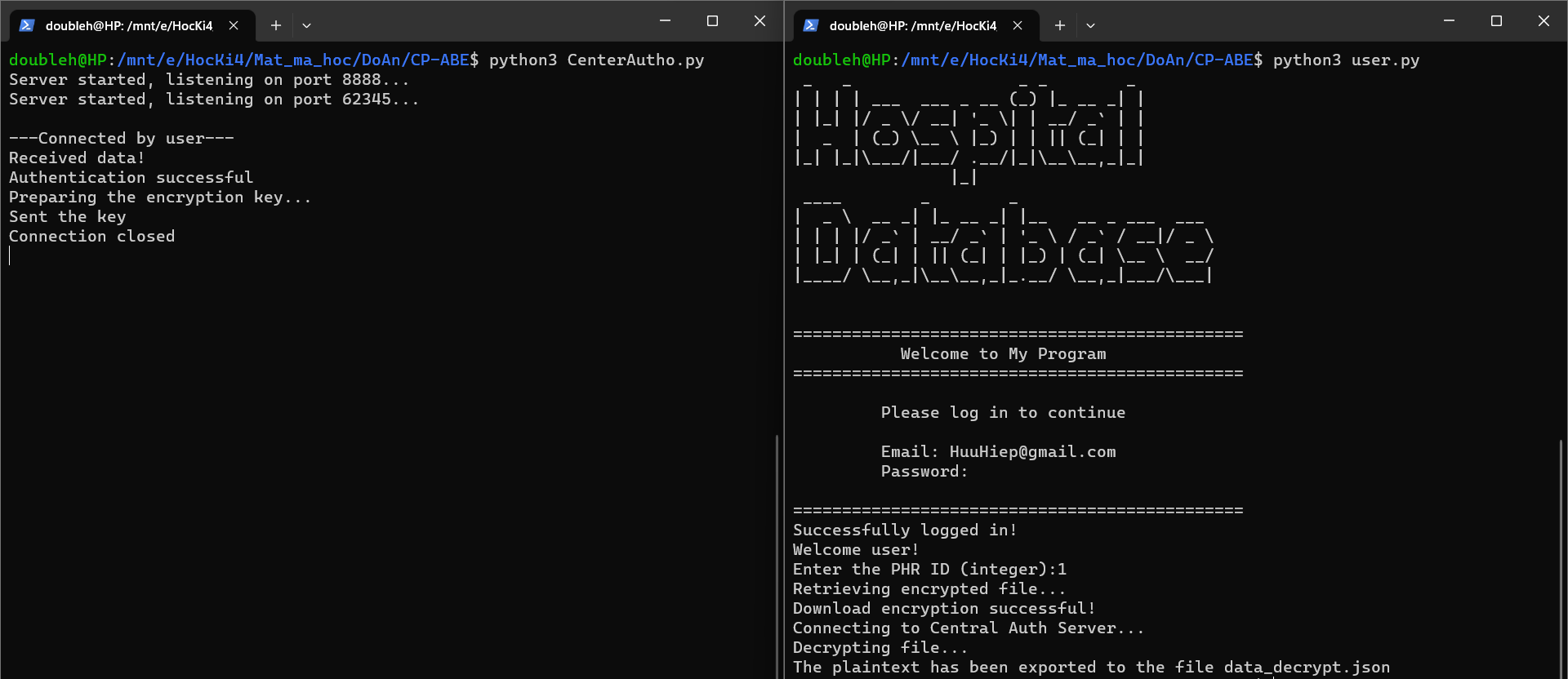


|  |  |
| --- | --- |
| **Người yêu cầu quyền truy cập** | **Quyền và thuộc tính** |
| data owner | Thêm dữ liệu trên database  UserName: “[admin@gmail.com](mailto:admin@gmail.com)”  Password: ”root123” |
| user1 | Được phép truy cập đến các PHR thuộc quyền truy cập của user đó.  UserName: ”[HuuHiep@gmail.com](mailto:HuuHiep@gmail.com)”  Password: ”HuuHiep123”  Có quyền truy cập tới: PHR1, 3, 6, 7, 8 |
| user2 | Được phép truy cập đến các PHR thuộc quyền truy cập của user đó.  UserName: ”[KhoaHoang@gmail.com](mailto:KhoaHoang@gmail.com)”  Password: ”KhoaHoang123”  Có quyền truy cập tới: PHR4, 6 |
| userN | … |

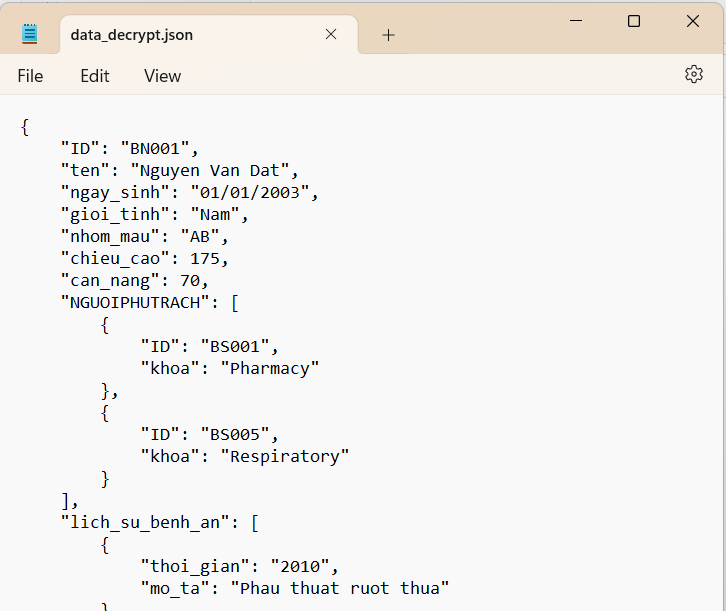
1. **User (Đăng nhập không thành công)**

****

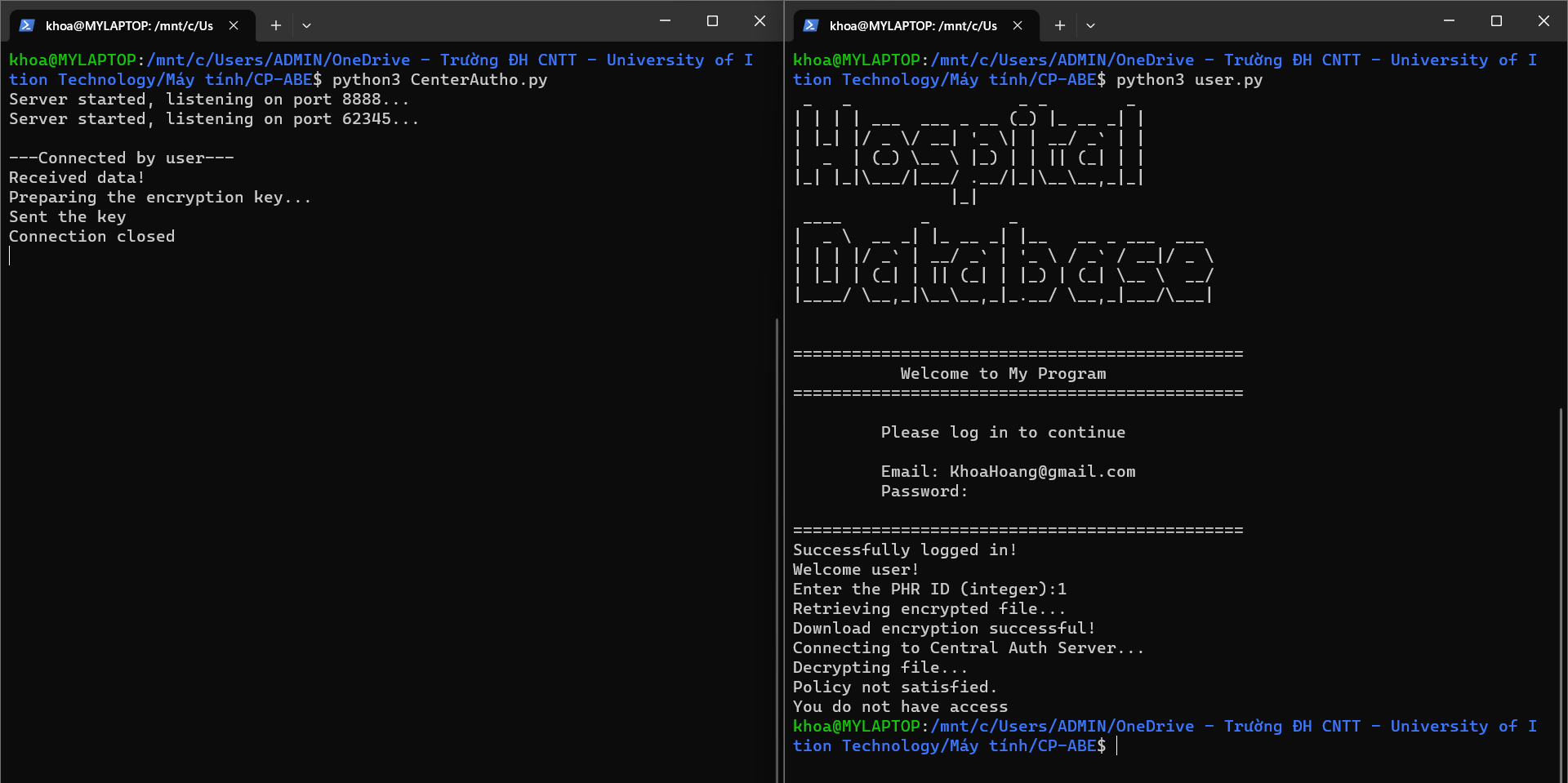
1. **User (Đăng nhập thành công và có quyền truy cập)**

****

Data nhận về:

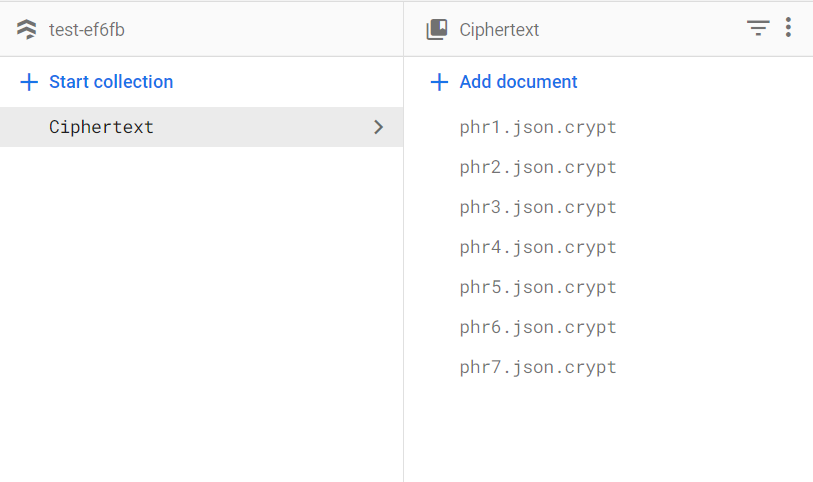


1. **User (Đăng nhập thành công nhưng không có quyền truy cập)**

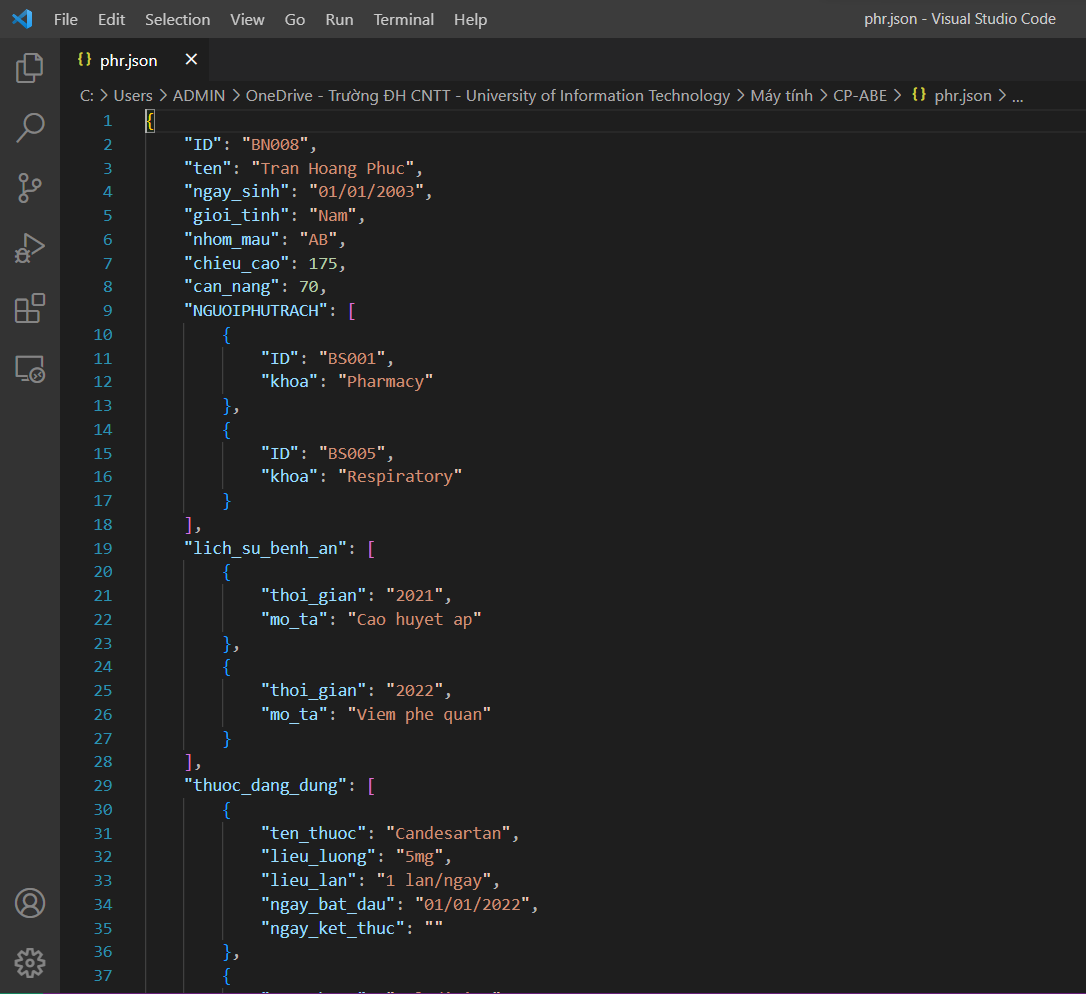
****

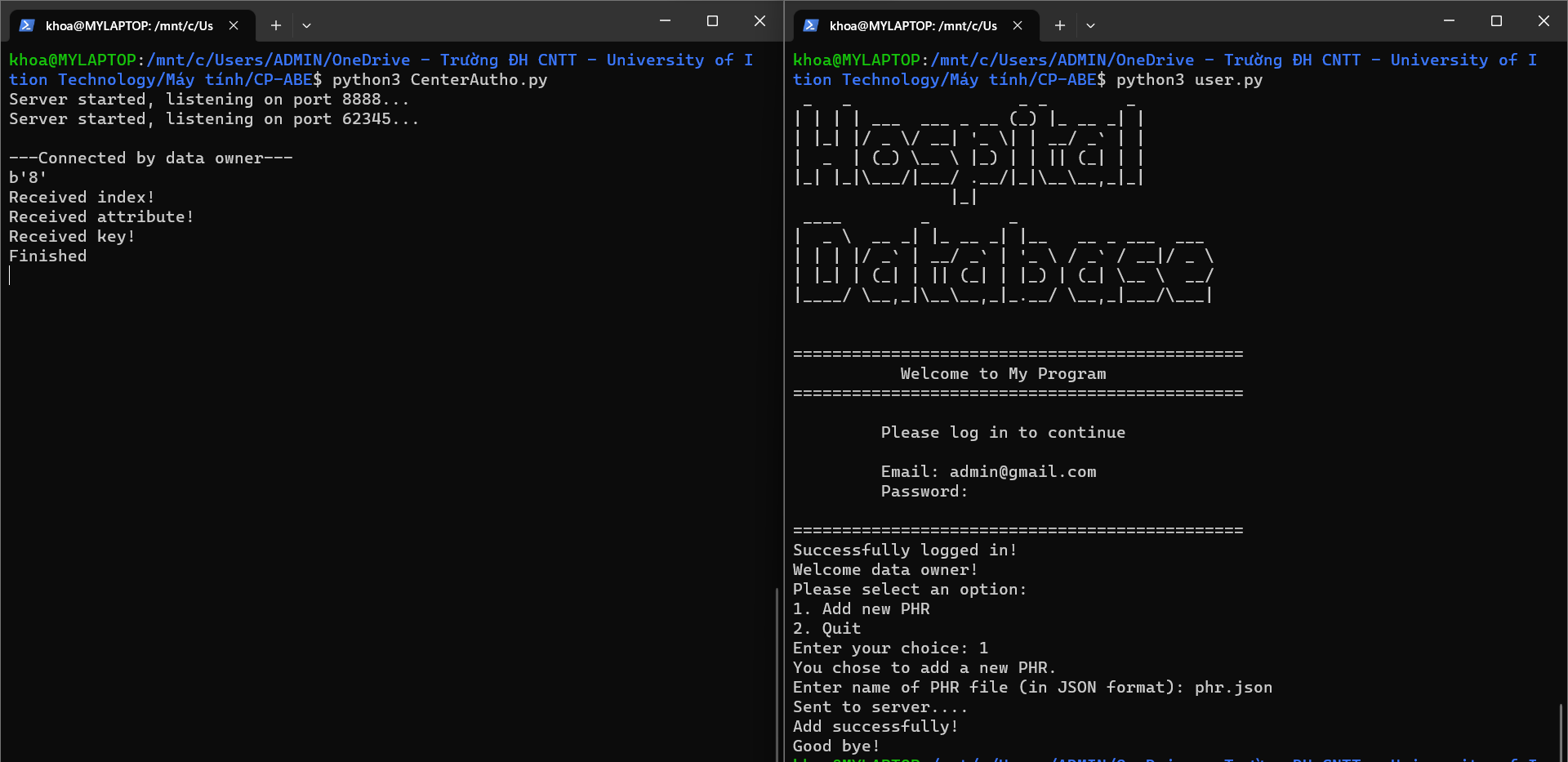
1. **Data Owner đưa dữ liệu lên**

Dữ liệu ban đầu trên Cloud:



Tiến hành đưa dữ liệu như sau lên:





Cloud sau khi mã hoá và đưa bản ciphertext của phr.json lên:

