

**UNIVERSITY OF INFORMATION TECHNOLOGY
FALCUTY OF COMPUTER NETWORKS AND COMMUNICATIONS**



UIT

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
Tổng quan.....	3
1. Vấn đề cần giải quyết.....	3
2. Ngữ cảnh	4
3. Các bên liên quan	4
4. Phân tích hệ thống.....	5
5. Nhu cầu bảo mật	5
Giải pháp	5
Tài liệu tham khảo	6
Đánh giá.....	7
Triển khai	7
Chạy mẫu.....	19

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

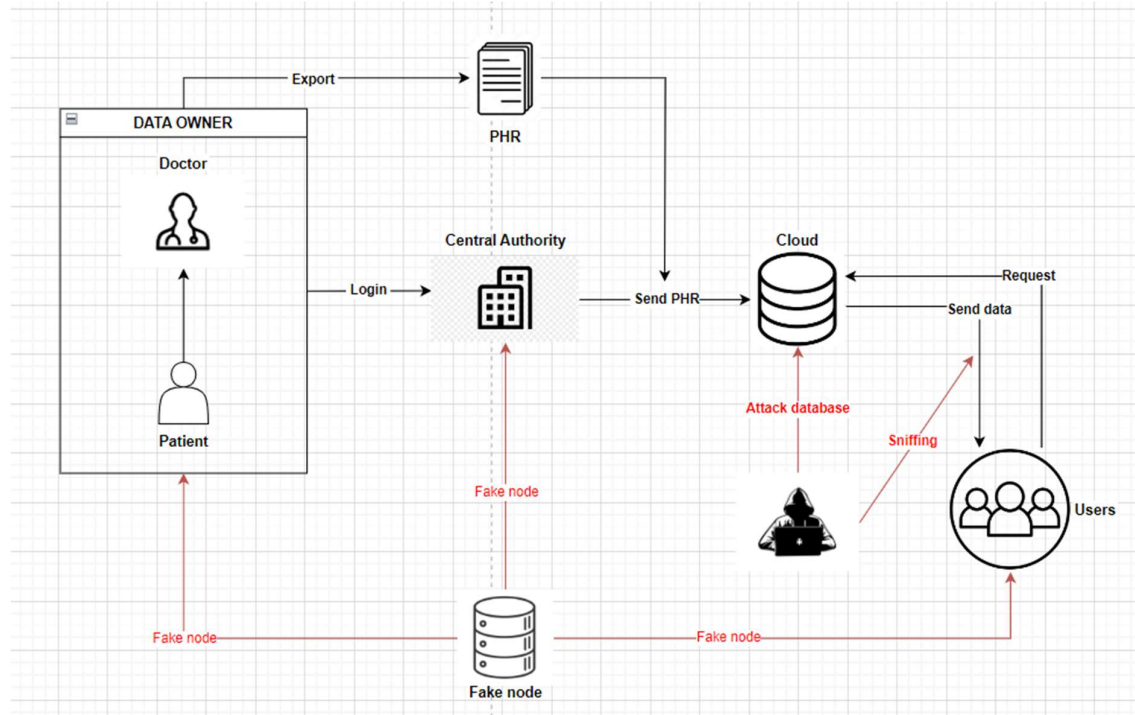
1. Vấn đề cần giải quyết

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:

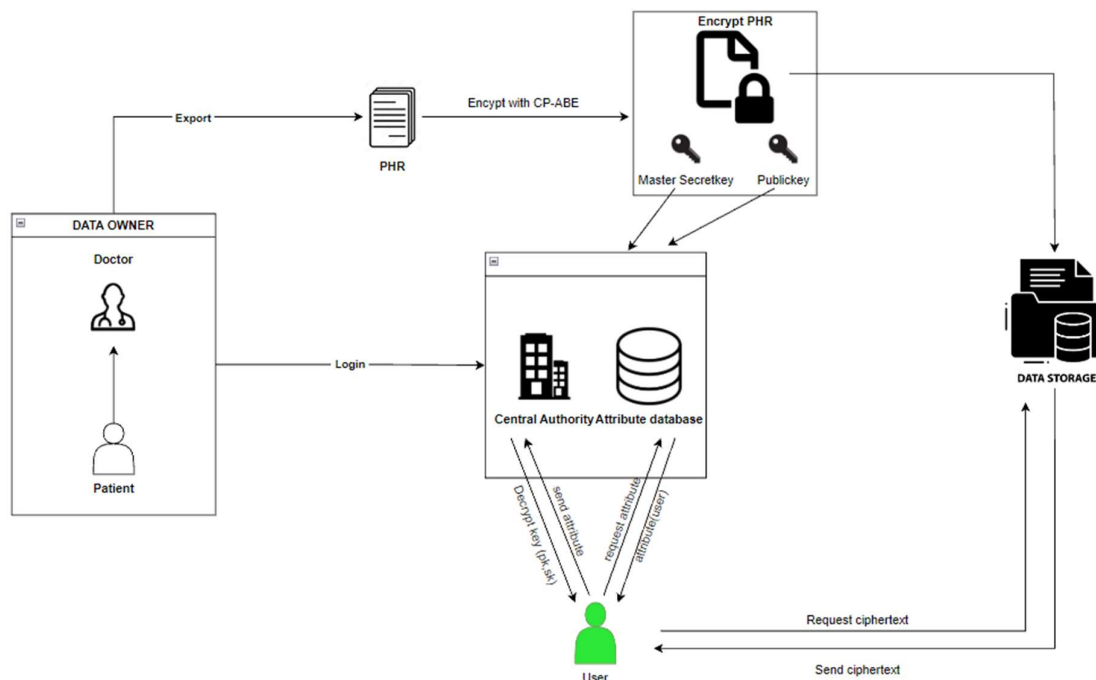
- Cloud (Semi-Trust): Là một nguồn không chắc tin cậy. Bên Cloud có thể bán hay làm rò rỉ thông tin của mình ra bên ngoài.
- Attacker có thể tấn công vào nơi lưu trữ hoặc đường truyền để lấy bản rõ.
- Sẽ có những node mạo danh Data-Owner để đăng dữ liệu không phù hợp, hay là mạo danh user để lấy PHR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

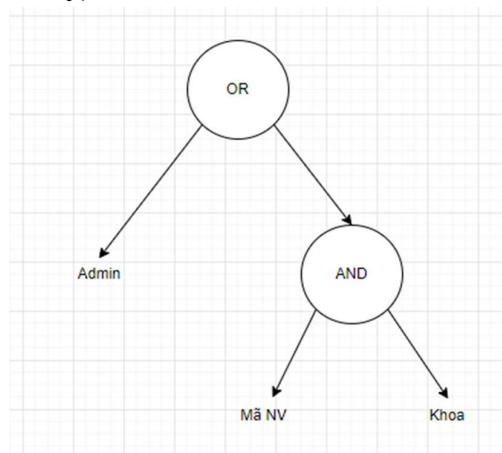
Triển khai certificate trong kết nối TLS: Certificate được ký bằng thuật toán ECDSA

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

- Setup(1^λ) Run GroupGen(1^λ) to obtain $(p, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e, g, h)$. Pick $a_1, a_2 \leftarrow_R \mathbb{Z}_p^*$ and $d_1, d_2, d_3 \leftarrow_R \mathbb{Z}_p$. Output

$$(h, H_1 := h^{a_1}, H_2 := h^{a_2}, T_1 := e(g, h)^{d_1 a_1 + d_3}, T_2 := e(g, h)^{d_2 a_2 + d_3})$$

as the public key pk. Also, pick $b_1, b_2 \leftarrow_R \mathbb{Z}_p^*$ and output

$$(g, h, a_1, a_2, b_1, b_2, g^{d_1}, g^{d_2}, g^{d_3})$$

as the master secret key msk.

- 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 $h, H_1 := h^{a_1}, H_2 := h^{a_2}$ và $T_1 := e(g, h)^{d_1 a_1 + d_3}, T_2 := e(g, h)^{d_2 a_2 + d_3}$ 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: $(g, h, a_1, a_2, b_1, b_2, g^{d_1}, g^{d_2}, g^{d_3})$

```

# 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

- KeyGen(msk, S) Pick $r_1, r_2 \leftarrow_R \mathbb{Z}_p$ and compute

$$sk_0 := (h^{b_1 r_1}, h^{b_2 r_2}, h^{r_1 + r_2})$$

using h, b_1, b_2 from msk. For all $y \in S$ and $t = 1, 2$, compute

$$sk_{y,t} := \mathcal{H}(y1t)^{\frac{b_1 r_1}{a_t}} \cdot \mathcal{H}(y2t)^{\frac{b_2 r_2}{a_t}} \cdot \mathcal{H}(y3t)^{\frac{r_1 + r_2}{a_t}} \cdot g^{\frac{\sigma_y}{a_t}},$$

where $\sigma_y \leftarrow_R \mathbb{Z}_p$. Set $sk_y := (sk_{y,1}, sk_{y,2}, g^{-\sigma_y})$. Also, compute

$$sk'_t := g^{d_t} \cdot \mathcal{H}(011t)^{\frac{b_1 r_1}{a_t}} \cdot \mathcal{H}(012t)^{\frac{b_2 r_2}{a_t}} \cdot \mathcal{H}(013t)^{\frac{r_1 + r_2}{a_t}} \cdot g^{\frac{\sigma'}{a_t}}$$

for $t = 1, 2$, where $\sigma' \leftarrow_R \mathbb{Z}_p$. Set $sk' = (sk'_1, sk'_2, g^{d_3} \cdot g^{-\sigma'})$. Output $(sk_0, \{sk_y\}_{y \in S}, sk')$ as the key.

- Tính toán: $sk_0 := (h^{b_1 r_1}, h^{b_2 r_2}, h^{r_1 + r_2})$:

```
# 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: $sk_{y,t} := \mathcal{H}(y1t)^{\frac{b_1 r_1}{a_t}} \cdot \mathcal{H}(y2t)^{\frac{b_2 r_2}{a_t}} \cdot \mathcal{H}(y3t)^{\frac{r_1 + r_2}{a_t}} \cdot g^{\frac{\sigma_y}{a_t}}$

```
# 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: $sk'_t := g^{d_t} \cdot \mathcal{H}(011t)^{\frac{b_1 r_1}{a_t}} \cdot \mathcal{H}(012t)^{\frac{b_2 r_2}{a_t}} \cdot \mathcal{H}(013t)^{\frac{r_1+r_2}{a_t}} \cdot g^{\frac{\sigma'}{a_t}}$

```
# 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: $(sk_0, \{sk_y\}_{y \in S}, sk')$

```
return {'attr_list': attr_list, 'K_0': K_0, 'K': K, 'Kp': Kp}
```

Bước 3: Encrypt

- Encrypt(pk, (M, π), msg) Pick $s_1, s_2 \leftarrow_R \mathbb{Z}_p$. Compute

$$ct_0 := (H_1^{s_1}, H_2^{s_2}, h^{s_1+s_2})$$

using pk. Suppose **M** has n_1 rows and n_2 columns. Then, for $i = 1, \dots, n_1$ and $\ell = 1, 2, 3$, compute

$$ct_{i,\ell} := \mathcal{H}(\pi(i)\ell 1)^{s_1} \cdot \mathcal{H}(\pi(i)\ell 2)^{s_2} \cdot \prod_{j=1}^{n_2} [\mathcal{H}(0j\ell 1)^{s_1} \cdot \mathcal{H}(0j\ell 2)^{s_2}]^{(M)_{i,j}},$$

where, recall that, $(M)_{i,j}$ denotes the (i, j) th element of **M**. Set $ct_i := (ct_{i,1}, ct_{i,2}, ct_{i,3})$. Also, compute

$$ct' := T_1^{s_1} \cdot T_2^{s_2} \cdot \text{msg}.$$

Output $(ct_0, ct_1, \dots, ct_{n_1}, ct')$ as the ciphertext.

- Tính toán $ct_0 := (H_1^{s_1}, H_2^{s_2}, h^{s_1+s_2})$:

```
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)
```

$$ct_{i,\ell} := \mathcal{H}(\pi(i)\ell 1)^{s_1} \cdot \mathcal{H}(\pi(i)\ell 2)^{s_2} \cdot \prod_{j=1}^{n_2} [\mathcal{H}(0j\ell 1)^{s_1} \cdot \mathcal{H}(0j\ell 2)^{s_2}]^{(M)_{i,j}}$$

- 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: $ct' := T_1^{s_1} \cdot T_2^{s_2} \cdot msg$

```
# 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: $(ct_0, ct_1, \dots, ct_{n_1}, ct')$

```
return {'policy': policy, 'C_0': C_0, 'C': C, 'Cp': Cp}
```

Bước 4: Decrypt

- **Decrypt(pk, ct, sk)** Recall that if the set of attributes S in sk satisfies the MSP (M, π) in ct , then there exists constants $\{y_i\}_{i \in I}$ that satisfy (2.1). Now, compute

$$\begin{aligned} \text{num} &:= ct' \cdot e\left(\prod_{i \in I} ct_{i,1}^{y_i}, sk_{0,1}\right) \cdot e\left(\prod_{i \in I} ct_{i,2}^{y_i}, sk_{0,2}\right) \cdot e\left(\prod_{i \in I} ct_{i,3}^{y_i}, sk_{0,3}\right), \\ \text{den} &:= e\left(sk'_1 \cdot \prod_{i \in I} sk_{\pi(i),1}^{y_i}, ct_{0,1}\right) \cdot e\left(sk'_2 \cdot \prod_{i \in I} sk_{\pi(i),2}^{y_i}, ct_{0,2}\right) \cdot e\left(sk'_3 \cdot \prod_{i \in I} sk_{\pi(i),3}^{y_i}, ct_{0,3}\right), \end{aligned}$$

and output num/den . Here $sk_{0,1}, sk_{0,2}, sk_{0,3}$ denote the first, second and third elements of sk_0 ; the same for ct_0 .

- Tính toán

$$\text{num} := ct' \cdot e\left(\prod_{i \in I} ct_{i,1}^{y_i}, sk_{0,1}\right) \cdot e\left(\prod_{i \in I} ct_{i,2}^{y_i}, sk_{0,2}\right) \cdot e\left(\prod_{i \in I} ct_{i,3}^{y_i}, sk_{0,3}\right)$$

```
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

$$\text{den} := e\left(sk'_1 \cdot \prod_{i \in I} sk_{\pi(i),1}^{y_i}, ct_{0,1}\right) \cdot e\left(sk'_2 \cdot \prod_{i \in I} sk_{\pi(i),2}^{y_i}, ct_{0,2}\right) \cdot e\left(sk'_3 \cdot \prod_{i \in I} sk_{\pi(i),3}^{y_i}, ct_{0,3}\right),$$

```
prod2_GT *= pair(prod_G, key['K_0'][i])
```

- Output: num/den .

```
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,
  "NGUOI PHU TRACH": [
    {
      "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 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))

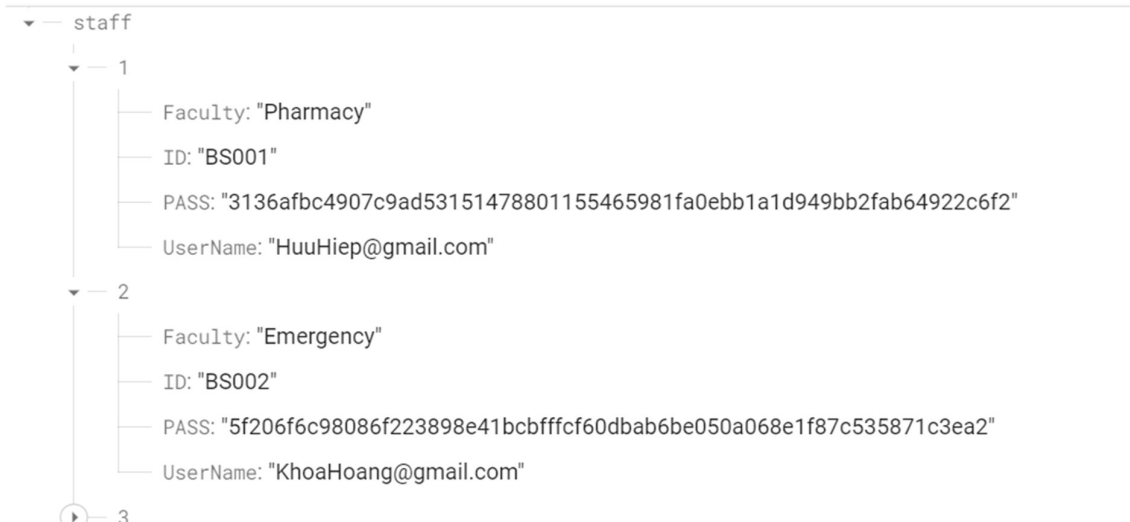
    with open("server.crt", "wb") as cert_file:
        cert_file.write(crypto.dump_certificate(crypto.FILETYPE_PEM,
cert))

```

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 " Password: "root123"
user1	Được phép truy cập đến các PHR thuộc quyền truy cập của user đó. UserName: " 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 " Password: "KhoaHoang123" Có quyền truy cập tới: PHR4, 6
userN	...

1. User (Đăng nhập không thành công)

```
khoa@MYLAPTOP: /mnt/c/Us x + v
khoa@MYLAPTOP: /mnt/c/Users/ADMIN/OneDrive - Trường ĐH CNTT - University of I
tion Technology/Máy tính/CP-ABE$ python3 CenterAuth.py
Server started, listening on port 8888...
Server started, listening on port 62345...

khoa@MYLAPTOP: /mnt/c/Us x + v
khoa@MYLAPTOP: /mnt/c/Users/ADMIN/OneDrive - Trường ĐH CNTT - University of I
tion Technology/Máy tính/CP-ABE$ python3 user.py

=====
Welcome to My Program
=====

Please log in to continue

Email: hello123
Password:

=====
Invalid email or password
Good bye!
khoa@MYLAPTOP: /mnt/c/Users/ADMIN/OneDrive - Trường ĐH CNTT - University of I
tion Technology/Máy tính/CP-ABE$
```

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

```
doubleh@HP: /mnt/e/HocKi4 x + v
doubleh@HP: /mnt/e/HocKi4/Mat_ma_hoc/DoAn/CP-ABE$ python3 CenterAuth.py
Server started, listening on port 8888...
Server started, listening on port 62345...

---Connected by user---
Received data!
Authentication successful
Preparing the encryption key...
Sent the key
Connection closed

doubleh@HP: /mnt/e/HocKi4 x + v
doubleh@HP: /mnt/e/HocKi4/Mat_ma_hoc/DoAn/CP-ABE$ python3 user.py

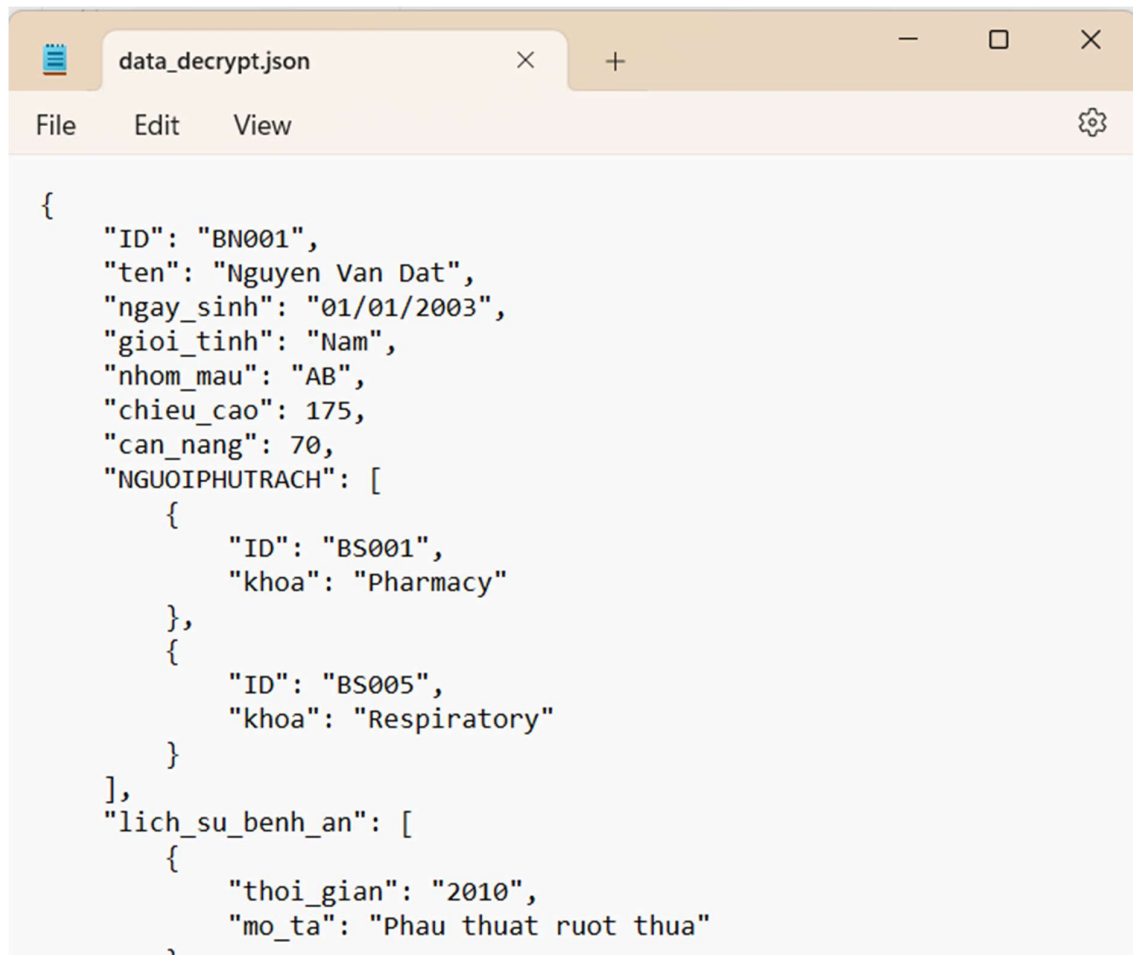
=====
Welcome to My Program
=====

Please log in to continue

Email: HuuHiep@gmail.com
Password:

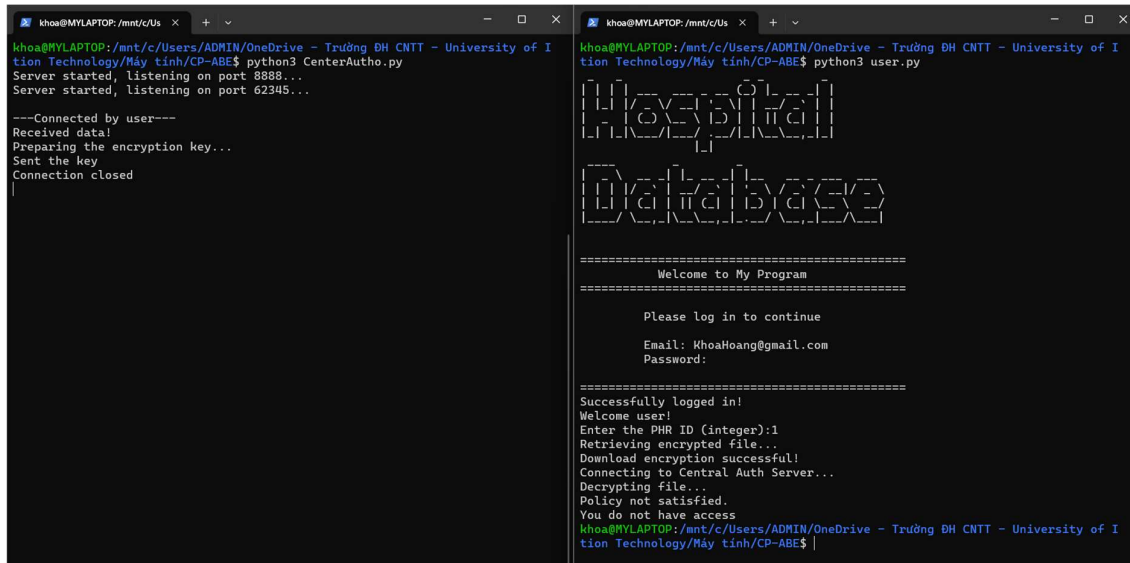
=====
Successfully logged in!
Welcome user!
Enter the PHR ID (integer):1
Retrieving encrypted file...
Download encryption successful!
Connecting to Central Auth Server...
Decrypting file...
The plaintext has been exported to the file data_decrypt.json
```

Data nhận về:



```
{
  "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"
    }
  ]
}
```

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



```
khoa@MYLAPTOP: /mnt/c/Users/ADMIN/OneDrive - Trường ĐH CNTT - University of I
tion Technology/Máy tính/CP-ABE$ python3 CenterAuth.py
Server started, listening on port 8888...
Server started, listening on port 62345...

---Connected by user---
Received data!
Preparing the encryption key...
Sent the key
Connection closed

khoa@MYLAPTOP: /mnt/c/Users/ADMIN/OneDrive - Trường ĐH CNTT - University of I
tion Technology/Máy tính/CP-ABE$ python3 user.py

=====
Welcome to My Program
=====

Please log in to continue

Email: KhoaHoang@gmail.com
Password:





=====
Successfully logged in!
Welcome user!
Enter the PHR ID (integer):1
Retrieving encrypted file...
Download encryption successful!
Connecting to Central Auth Server...
Decrypting file...
Policy not satisfied.
You do not have access
khoa@MYLAPTOP: /mnt/c/Users/ADMIN/OneDrive - Trường ĐH CNTT - University of I
tion Technology/Máy tính/CP-ABE$
```

4. Data Owner đưa dữ liệu lên

Dữ liệu ban đầu trên Cloud:

test-ef6fb	Ciphertext
+ Start collection	+ Add document
Ciphertext >	<p>phr1.json.crypt</p> <p>phr2.json.crypt</p> <p>phr3.json.crypt</p> <p>phr4.json.crypt</p> <p>phr5.json.crypt</p> <p>phr6.json.crypt</p> <p>phr7.json.crypt</p>

Tiến hành đưa dữ liệu như sau lên:

 test-ef6fb	 Ciphertext  
+ Start collection	+ Add document
Ciphertext >	<div>phr1.json.crypt</div> <div>phr2.json.crypt</div> <div>phr3.json.crypt</div> <div>phr4.json.crypt</div> <div>phr5.json.crypt</div> <div>phr6.json.crypt</div> <div>phr7.json.crypt</div> <div>phr8.json.crypt</div>