

University of Information Technology

Faculty of Computer Network and Communications



UIT

FINAL REPORT

Subject : Cryptography

Class : NT219.N22.ATCL

Lecturer : Nguyen Ngoc Tu

Ciphertext Policy - Attributes Based Encryption for Cloud-Based Data Protection

Nguyen Tran Anh Duc – 21521964

Nguyen Huu Tien – 21520479

Le Thanh Duan – 19521370

Table of Contents

Table of Contents

1. Introduction.....	3
1.1 Overview	3
1.2 Problem Statement.....	4
1.2.1 Scenario	4
1.2.2 Related-Parites :.....	4
1.2.3 Security Objectives :	4
2. Solution.....	5
2.1 Encryption Scheme	5
2.1.1 FAME: Fast Attribute-based Message Encryption	5
2.2 Reasons to Choose CP-ABE for Data Encryption in Distributed Systems.....	6
3. Implementation.....	8
3.1 Tools and resources	8
3.2 Design and Code	8
3.2.1 System Design.....	8
3.2.2 CA Server	9
3.2.3 Client	14
4. Testing.....	19
4.1 Testing	19
4.1.1 Uploading and Downloading.....	19
4.1.2 Searching	22
5. SUMMARY	24
5.1 Results	24
5.2 Tasks Chart	25
5.3 Final Words	25

1. Introduction

1.1 Overview

Cloud computing has revolutionized the way we store and share data, offering low-cost storage and easy access through cloud services like Dropbox and Google Drive.

However, as these services become more popular, concerns around data security and privacy have also grown.

To address these concerns, it is important to implement security mechanisms that offer a higher level of protection and access control. One such mechanism is the use of advanced encryption and access control techniques to secure data stored in the cloud.

Encryption can be used to protect data both while it is in transit and at rest. By encrypting data before it leaves the user's device, it can be securely transmitted to the cloud service without exposing sensitive information to potential attackers. Additionally, encrypting data at rest can provide an additional layer of protection against unauthorized access to stored data.

Access control is another key aspect of securing data in the cloud. By implementing fine-grained access control policies, it is possible to ensure that only authorized users or applications can access or modify data stored in the cloud. This can include techniques such as attribute-based access control (ABAC), role-based access control (RBAC), or other access control models.

Overall, the use of advanced security mechanisms such as encryption and access control can provide a higher level of protection and control over data stored in the cloud, helping to address concerns around data security and privacy. As cloud services continue to grow in popularity, it is essential for users and organizations to prioritize security and implement robust security measures to protect their sensitive data.

1.2 Problem Statement

Nowadays the use of cloud services has become increasingly popular for storing sensitive and confidential data. However, the benefits of convenience and scalability come with significant security risks that must be addressed. For example, a company that stores private digital assets such as contracts, internal documentation, and other sensitive information in the cloud is particularly vulnerable to security threats such as data breaches and insider threats.

1.2.1 Scenario

In this scenario, a finance company is using a cloud service to store their private digital assets (contracts, internal documentation, etc). Therefore, they will be facing several security issues like data breach, insider threat.

1.2.2 Related-Parities :

- **Data owner** : The data owner is responsible for ensuring that the data they store in the cloud is secure. This includes taking appropriate measures to encrypt the data, controlling access to it, and monitoring for unauthorized access or data breaches.
- **Cloud Service Provider** : The cloud service provider is responsible for ensuring the security and availability of the data stored in their cloud infrastructure.
- **Data users** : The data users responsibility are to ensure that they access and use the data in compliance with the data owner's access policies and also take appropriate measures to protect the confidentiality and integrity of the data.
- **Adversary** : The adversary, or any entity or individual that seeks to compromise the security of the data stored in the cloud, has a responsibility to cease any activities that may compromise the security of the data. They are also responsible for any damages or harm caused by their actions, and may be subject to legal consequences if their actions are deemed illegal or malicious.

1.2.3 Security Objectives :

1. **Confidentiality** - Ensuring that sensitive data stored in the cloud is protected from unauthorized access or disclosure.
2. **Integrity** - Ensuring that data stored in the cloud is not tampered with or modified in an unauthorized manner.
3. **Authorization** - Ensuring that users accessing the cloud-based data have the appropriate level of access permissions and privileges.

2. Solution

2.1 Encryption Scheme

To fulfill all the security objectives of the problem we mentioned above, our solution is to use Ciphertext-Policy Attributes-Based Encryption (CP-ABE) scheme.

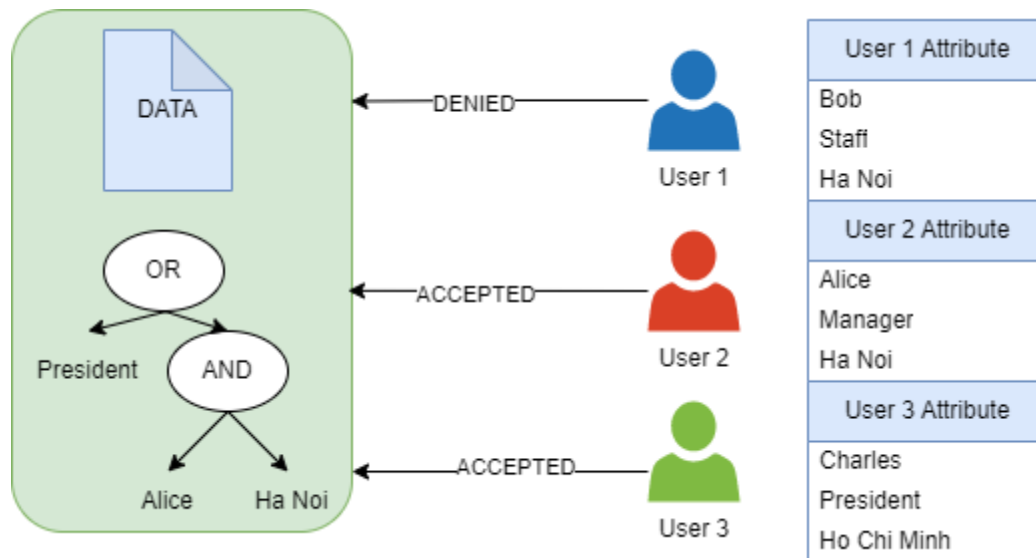


Figure 2.1 Ciphertext Policy – Attributes-Based Encryption

However, CP-ABE is not designed to directly encryption data. Therefore we have to combine CP-ABE with AES256-GCM to encrypt data. Details will be explained in Section 3.

2.1.1 FAME: Fast Attribute-based Message Encryption

Fast Attribute-based Message Encryption, or FAME, is a method of encryption that enables safe communication between two parties based on qualities rather than particular identities. It is an attribute-based encryption (ABE) technique that is intended to be effective and scalable, enabling quick message encryption and decryption even in large-scale systems with numerous users.

FAME provides a number of benefits over other ABE systems, which makes it a popular option for secure communication.

- First off, it employs hybrid encryption, which offers quick encryption and decryption times, making it appropriate for usage in contexts with limited resources, such mobile devices and the Internet of Things.

- Second, it has a low computational overhead, which means that encryption and decryption processes use less computer resources.
- Thirdly, it has a flexible access control mechanism that allows for fine-grained access control policies based on a wide range of attributes. Finally, it's designed to be secure against a range of attacks, including collusion attacks.

Overall, FAME is a powerful and efficient encryption scheme that provides flexible access control and robust security. This makes it a popular choice for a wide range of applications, including secure messaging, access control, and data sharing.

Chase, M., & Shashank, A. (2017). FAME (Fast Attribute-based Message Encryption). *CCS '17 Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* .

Shashank Agrawal, M. C. (2017). FAME: Fast Attribute-based Message Encryption.

2.2 Reasons to Choose CP-ABE for Data Encryption in Distributed Systems

CP-ABE has several advantages over other encryption techniques. It enables data owners to encrypt their data in a way that allows for fine-grained access control, reducing the risk

of unauthorized access and data breaches. It also allows for the secure sharing of data across distributed systems, without the need for a centralized authority to manage access control. Additionally, CP-ABE can provide a high degree of flexibility, allowing data owners to adjust access policies and attributes as needed.

However, CP-ABE also has some limitations. It can be computationally expensive, especially when dealing with large amounts of data and complex access policies. Additionally, it can be difficult to manage and maintain access policies as the number of attributes and users increases.

Overall, CP-ABE is a powerful encryption technique that can provide fine-grained access control and secure data sharing in distributed systems. Its advantages and limitations should be considered carefully when implementing a data security strategy.

3. Implementation

3.1 Tools and resources

- **Programming Language : Python**
Python provides a variety of packages to support this scheme. In this project, our group utilizes Charm-Crypto and PyCryptodome for cryptographic development.
- **Database : MongoDB**
MongoDB is a powerful and flexible database management system that is well-suited to our project because is designed to be scalable, allowing developers to easily add or remove nodes to handle increased data loads.

3.2 Design and Code

3.2.1 System Design

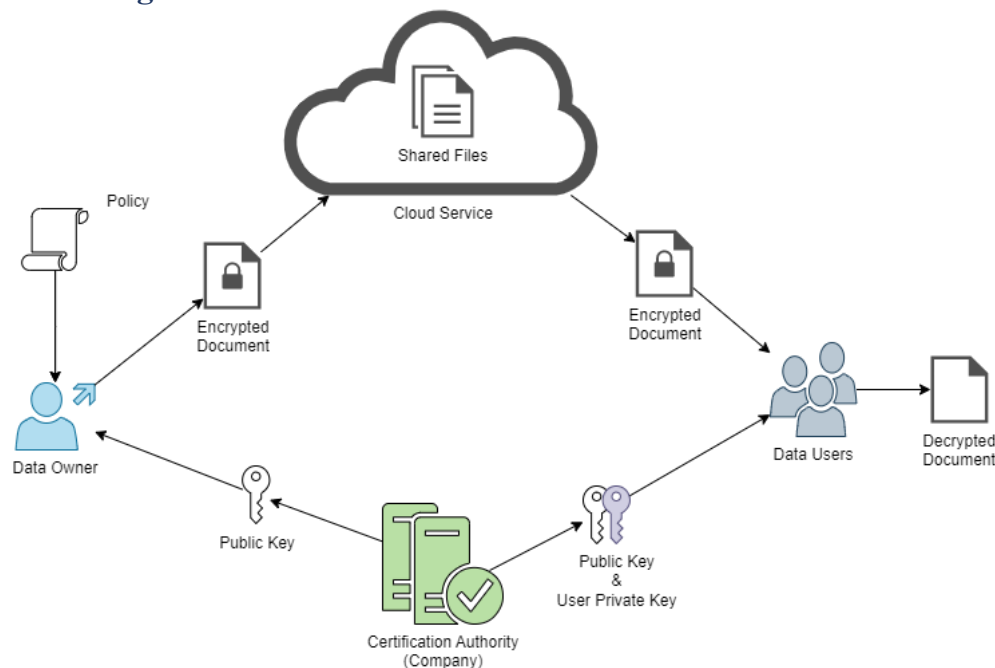


Figure 3.1 System Model

Our design contains 3 main nodes : Users (Data Owner, Data Users), Cloud Service and Certification Authority (CA).

- **Users** : Users are company employees, their personal data (Name, Date of birth, National ID) will be stored in CA's database. Sensitive data like Nation ID will be stored as SHA256 hashes. They will have to verify themselves with CA in order to get their attributes and private keys.
- **Cloud Service** : Only for storage and file management.
- **Certification Authority** : Verify users and provide private keys.

3.2.2 CA Server

3.2.2.1 CA Secrets

CA Server have to secure its API key and Master Key. In order to accomplish this, we utilized TPM 2.0 (Trusted Platform Module 2.0) - a hardware-based security module that provides a secure environment for cryptographic operations and storage of sensitive data.

```
tpm2_createprimary -C o -g sha256 -G ecc -c primary.ctx
```

MEANING:

- C o tells us to use the owner hierarchy
- G ecc tell us that we wish to generate an ECC key
- g sha256 tell us that we wish to use the SHA256 hashing algorithm
- c primary.ctx means to generate a TPM context object (called primary.ctx)
- a ... gives the attributes of the key - these will be explained

```
tpm2_flushcontext -t
```

MEANING:

Typically, when using the TPM, the interactions occur through a resource manager, like tpm2-abrmd(8). When the process exits, transient object handles are flushed. Thus, flushing transient objects through the **command** line is not required. However, when interacting with the TPM directly, this scenario is possible. The below example assumes direct TPM access not brokered by a resource manager. Specifically we will use the simulator.

```
tpm2_pcrread -o srtm.pcrvalues sha256:0,1,2,3
```

MEANING:

we **select** the PCRs and save these in a file.
To output a list of PCR banks (**sha256=**)
and ids (0, 1, 2) specify a PCR selection list as the argument as specified via section "**PCR Bank Specifiers**".

```
tpm2_createpolicy --policy-pcr -l sha256:0,1,2,3 -f srtm.pcrvalues -L srtm.policy
```

MEANING:

- policy-pcr specifies that we are creating a policy from PCRs
- l sha256:0,1,2,3 specifies which PCRs
- f srtm.pcrvalues spcified the file in which we stored the PCR values from earlier
- L strm.policy specifies the output file **for** the policy

```
tpm2_flushcontext -t
```

```
echo '9A1C95B959B9B67EEF032BA0FD0ABC22' | tpm2_create -C primary.ctx -L srtm.policy -i- -u seal.pub -r seal.priv -c seal.ctx
```

MEANING:

Create a child object. The object can either be a key or a sealing object.

A sealing object allows to seal user data to the TPM, with a maximum size of 128 bytes. Additionally it will load the created object **if** the **-c** is specified.

```
tpm2_unseal -c seal.ctx -p pcr:sha256:0,1,2,3
```

MEANING:

Returns a data blob in a loaded TPM object. The data blob is returned in clear.

The data is sealed at the **time** of the object creation using the `tpm2_create` tool.

Such an object intended **for** sealing data has to be of the **type** `TPM_ALG_KEYEDHASH`.

Sealing is the process of encrypting data using a secret key that is stored inside the TPM. This key is generated based on a set of platform-specific characteristics, such as the state of the system's hardware and firmware, and is only accessible when those characteristics are the same as when the key was created. This means that the sealed data can only be decrypted and accessed on the same system configuration that was used to create the key.

The seal operation in TPM 2.0 takes a plaintext data input and encrypts it using the TPM's sealing key. The sealed data output can then be stored securely, for example, on disk or in the cloud, knowing that the data can only be decrypted on the same system configuration that was used to create the sealing key.

Unsealing, on the other hand, is the process of decrypting the sealed data using the same key that was used to seal it. The unseal operation in TPM 2.0 takes the sealed data and the corresponding sealing key as inputs and outputs the original plaintext data.

The seal and unseal operations provide a way to securely store and transmit data, knowing that it can only be accessed by authorized parties on a specific system configuration. This makes TPM 2.0 a powerful tool for protecting sensitive data and ensuring the integrity of computing systems.

3.2.2.2 CA Role

The primary role of a Certificate Authority (CA) is to authenticate users by comparing the information they provide with the information stored in a database. The CA verifies the

user's identity by checking the accuracy of the information provided and matching it with the relevant data in their database.

```
def verify(fullname, dob, cccd):
    action = url + "findOne"
    payload = json.dumps({
        "collection": "Employees",
        "database": "CompanyData",
        "dataSource": "CA",
        "filter" : {"cccd":cccd, "name":fullname, "dob":dob},
        "projection":{
            "name":1,
            "dob":1,
            "cccd":1,
        }
    })

    response = requests.request("POST", action, headers=headers,
data=payload)
    result = json.loads(response.text) ['document']
    if result:
        return True
    return False
```

verify() function take 3 arguments :

1. Fullname
2. Date of birth
3. National ID

We considered that National ID is a private parameter.

```
def isVerified(username):
    action = url + "findOne"
    payload = json.dumps({
        "collection": "Employee Accounts",
        "database": "CompanyData",
        "dataSource": "CA",
        "filter" : {"username":username},
        "projection":{
            "verified":1
        }
    })

    response = requests.request("POST", action, headers=headers,
data=payload)
    result = json.loads(response.text) ['document'] ['verified']
    if result:
        return True
    return False

def updateVerifiedProfile(username, fullname):
```

```

action = url + "updateOne"
payload = json.dumps({
    "collection": "Employees",
    "database": "CompanyData",
    "dataSource": "CA",
    "filter": {"name": fullname},
    "update": {
        "$set": {
            "registered" : True,
            "username": username
        }
    }
})
r = requests.request("POST", action, headers=headers, data=payload)
payload = json.dumps({
    "collection": "Employee Accounts",
    "database": "CompanyData",
    "dataSource": "CA",
    "filter": {"username": username},
    "update": {
        "$set": {
            "verified" : True
        }
    }
})
r = requests.request("POST", action, headers=headers, data=payload)
return

```

After verifying user, CA will update their profile in the database.

Other function of CA is provide users their private key based on their attributes.

```

def GetAttributes(username):
    action = url + "findOne"
    payload = json.dumps({
        "collection": "Employees",
        "database": "CompanyData",
        "dataSource": "CA",
        "filter": {"username": username},
        "projection": {
            "name": 1,
            "role": 1,
            "gender": 1,
            "location": 1
        }
    })
    r = requests.request("POST", action, headers=headers, data=payload)
    return json.loads(r.text)['document']

```

Employees attributes contain :

- Name
- Role
- Gender
- Location

Example Data :

```

• { "_id": {"$oid": "648952d74a19aa03c4ed2a63"},
• "name": "alice",
• "dob": "24/04/2003",
• "cccd": "551639db7847d33d01f25aeefb5c7e6c1b959067a039b0fa997235cbbe92c71",
• "bhyt": "76561435",
• "role": "manager",
• "location": "hcm",
• "dow": "10/04/2023",
• "gender": "female",
• "registered": true,
• "username": "hello" }

```

Based on these attributes, server will generate a private key and send it to the user.

```

def PrivateKeyGen(pk, username):
    mk = GetKey("master key")
    ctx = bytes.fromhex(mk)
    iv = ctx[:16]
    ctx = ctx[16:]
    aes = AES.new(key, AES.MODE_CBC, iv=iv)
    recover = aes.decrypt(ctx)
    mk = Padding.unpad(recover, AES.block_size)
    mkb = cpabe.bytesToObject(mk, cpabe.groupObj)
    attributes = []
    attr = GetAttributes(username)
    for i in attr.keys():
        attributes.append(attr[i].upper())
    sk = cpabe.PrivateKeyGen(pk, mkb, attribute=attributes)
    sk = cpabe.objectToBytes(sk, cpabe.groupObj)
    return binascii.hexlify(sk).decode()

```

First we have to decrypt the Master Key with the secret sealed with TPM as mentioned above.

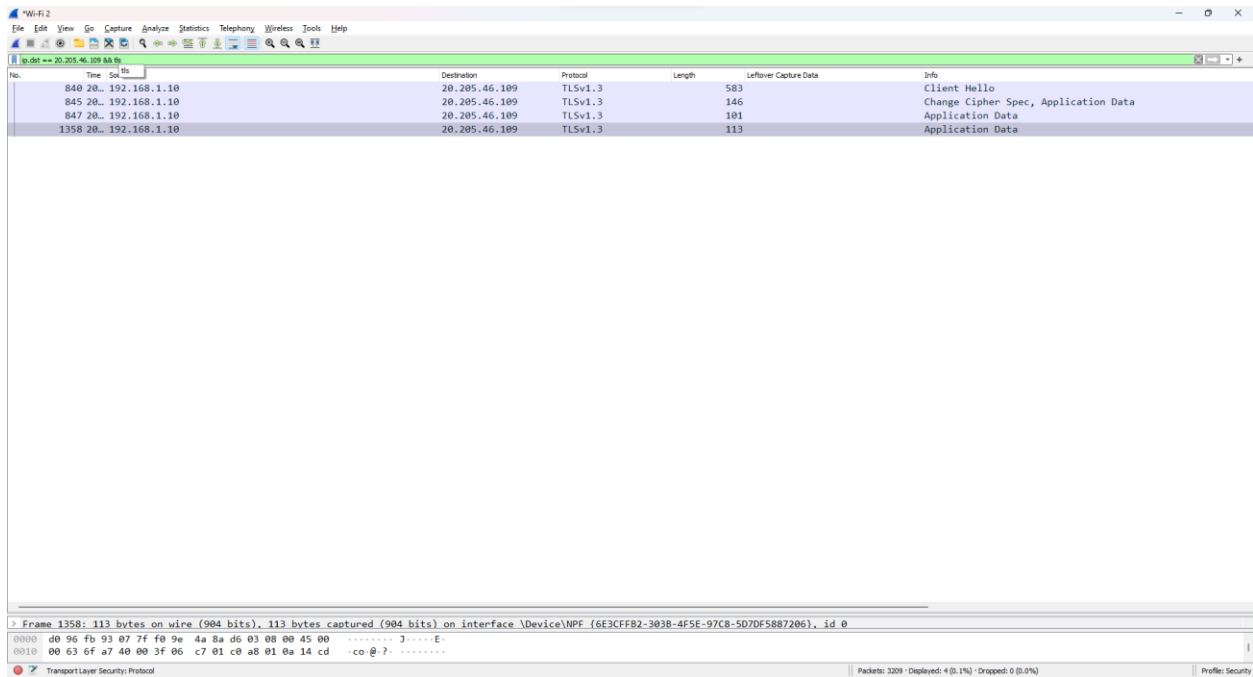
```

cmd = "./unseal.sh"
key = subprocess.check_output(cmd, shell=True)
key = bytes.fromhex(key.decode().strip())

```

3.2.2.3 Secure communication between CA and Clients

The Certificate Authority (CA) Server is operating on IP address 20.205.46.109 and Port 1337. To protect against network sniffing attacks, we are utilizing TLSv1.3 to encrypt all communications between the Clients and the Server. This ensures that sensitive information exchanged between the two parties is secure and cannot be intercepted by unauthorized third parties.



The image shows a Wireshark capture of network traffic. The top pane displays a list of packets, with the selected packet (No. 1358) expanded in the middle pane. The bottom pane shows the raw packet data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
840	20...	192.168.1.10	20.205.46.109	TLSv1.3	583	Client Hello
845	20...	192.168.1.10	20.205.46.109	TLSv1.3	146	Change Cipher Spec, Application Data
847	20...	192.168.1.10	20.205.46.109	TLSv1.3	101	Application Data
1358	20...	192.168.1.10	20.205.46.109	TLSv1.3	113	Application Data

Frame 1358: 113 bytes on wire (904 bits), 113 bytes captured (904 bits) on interface \Device\NPF {6E3CFFB2-303B-4F5E-97C8-5D7DF5887206}, id 0

0000 d0 96 fb 93 07 7f f0 9e 4a 8a d6 03 00 00 45 00J-----E-

0010 00 63 6f a7 40 00 3f 06 c7 01 c0 a8 01 0a 14 cd ..co@?.....

Transport Layer Security Protocol

Packets: 3209 · Displayed: 4 (0.1%) · Dropped: 0 (0.0%)

Profile: Security

Figure 3.2 Network Sniffing traffic between Client and Server

3.2.3 Client

3.2.3.1 Client Utilities

```
===== HELP MENU =====
/register                : register
/login                   : login
/verify                  : verify your account
/key                     : get your private key
```

```

/upload                               : choose file to upload
/download                             : download files
/search [option : -d,-o,-n]          : search files
-d : search by upload date
-o : search by file owner name
-e : search by file type
=====

```

Here is all the available commands that we provided for users.

3.2.3.1.1 Register / Login

```

elif message.startswith('/register'):
    username = input("[+] Enter username : ")
    password = getpass("[+] Enter password : ")
    if(len(password) < 8):
        print("[ERROR] : Password must be longer than 8")
        return None
    conf_password = getpass("[+] Confirm password : ")
    if(conf_password == password):
        return f"/register {username} {ObfuscateAndHash(password)}".encode()
    else:
        print("[ERROR] : Password did not match!")
        return None
elif message.startswith('/login'):
    username = input("[+] Enter username : ")
    password = getpass("[+] Enter password : ")
    USERNAME = username
    return f"/login {username} {ObfuscateAndHash(password)}".encode()

```

When registering and logging, the password will be obfuscated and hashed by ObfuscateAndHash() function.

```

def ObfuscateAndHash(password):
    length = len(password)
    res = ""
    for i in range(length):
        res += printable[ord(password[i]) % 94]
    obj = hashlib.sha256(res.encode()).digest()
    return binascii.hexlify(obj).decode()

```

This ensures that users password can not be exposed by dictationary attack.

Example :

```

Original Password : iloveyou123456
Obfuscated Password : beho7rhnNOPQRS
Hashed Password :
312724c91d5c9fc5b35b2e781b6a438f90d56136bdb8527c78b6c4279395ddfa

```

3.2.3.1.2 Verify

```
if(message.startswith("/verify")):
    fullname = input("[+] Enter fullname : ")
    birth = input("[+] Enter day of birth : ")
    cccd = input("[+] Enter cccd : ")
    return f"/verify {fullname} {birth} {ObfuscateAndHash(cccd)}".encode()
```

Same as password, National ID will be obfuscated and hashed before sending to CA for verification.

3.2.3.1.3 Key

Users can request for public and private keys only if they are verified. Private key are only available in session.

3.2.3.1.4 Upload / Download

Using public/private keys pair, now users can perform file encryption / decryption using CP-ABE and AES.

Let's take a look at ABEencryption() and ABEdecryption() functions.

```
def ABEencryption(filename,pk,policy):
    msg = open(filename,"rb").read()
    """
    Create ABE key then encrypt with CP-ABE
    Encode ABE key and attach to file
    """

    serialize_encoder = ac17.mySerializeAPI()

    abe_key = groupObj.random(GT)
    abe_key_ctxt = cpabe.encrypt(pk,abe_key,policy)

    abe_key_ctxt_b = serialize_encoder.jsonify_ctxt(abe_key_ctxt)
    abe_key_ctxt_b = base64.b64encode(abe_key_ctxt_b.encode())
    abe_key_size = len(abe_key_ctxt_b)
    stream = struct.pack('Q',abe_key_size)
    namesplit = filename.split('/')
    outname = f"{namesplit[len(namesplit)-1]}.scd"

    """
    Use AES to encrypt the file then attach needed component
    """

    aes_key = hashlib.sha256(str(abe_key).encode()).digest()
    iv = os.urandom(16)
```



```

encryptor = AES.new(aes_key,AES.MODE_GCM)
encrypted_data,authTag = encryptor.encrypt_and_digest(msg)
nonce = encryptor.nonce
output = stream + authTag + nonce + abe_key_ctxt_b + encrypted_data

return output

```

Encryption :

1. Random abe_key
2. Encrypt abe_key with CP-ABE , this produced abe_key_ctxt
3. Serialize abe_key_ctxt then attach to the output file
4. Pack the length of serialized abe_key_ctxt and write to the first 8 bytes of output
5. Hash the abe_key to make aes_key
6. Encrypt the file with AES256-GCM then write the encrypted data to the output

Encrypted file structure :

Packed abe key size	authTag	nonce	ABE key	Data
8 bytes	16 bytes	16 bytes	N bytes	N bytes

```

def ABEdecryption(filecontent,pk,sk):
    serialize_encoder = ac17.mySerializeAPI()
    ciphertext_stream = bytes.fromhex(filecontent)
    abe_key_size = struct.unpack('Q',ciphertext_stream[:8])[0]
    ciphertext = bytes.fromhex(filecontent)
    autTag = ciphertext[8:24]
    nonce = ciphertext[24:40]
    abe_key_ctxt_b = ciphertext[40:abe_key_size+40]
    abe_key_ctxt_b = base64.b64decode(abe_key_ctxt_b)
    abe_key_ctxt = serialize_encoder.unjsonify_ctxt(abe_key_ctxt_b)
    abe_key = cpabe.decrypt(pk,abe_key_ctxt,sk)
    if(abe_key):
        aes_key = hashlib.sha256(str(abe_key).encode()).digest()
        decryptor = AES.new(aes_key,AES.MODE_GCM,nonce)
        decrypted_data =
    decryptor.decrypt_and_verify(ciphertext[40+abe_key_size:],autTag)
    return decrypted_data
    else:
    return None

```

Decryption :

1. Extract the abe_key_size , authTag and nonce
2. Recover abe_key_ctxt_b = ciphertext[40:abe_key_len+40]
3. Deserialized abe_key_ctxt_b then decrypt it

4. If policy satisfied to decrypt the `abe_key_ctxt_b`, we hash the `abe_key` to retrieve the `aes_key`
5. Decrypt the file with `aes_key`

3.2.3.1.5 Search

```

_id: ObjectId('649287e6f102e35a1339941a')
filename: "BaoCao.docx.scd"
owner: "alice123"
extension: "docx"
upload_date: "2023-06-21"
sha256: "a974f936ee08cb8c4585b39345c3571deb6db10812ec7425324d99a254f4c695"
chunkSize: 261120
length: 17726508
uploadDate: 2023-06-21T05:17:29.485+00:00

```

Figure 3.3 File Data on database

Whenever a file is uploaded, its name, owner name, file type, data, upload date, and sha256 hash are stored in the database. To enable users to easily find their files, we have provided a search feature with three options:

- Search by owner name
- Search by upload date
- Search by file type

Users can use any combination of these options to search for the files they need.

```

def HandleSearch(message):
    option = ['-d', '-o', '-e']
    msg = '/search'
    for i in message.split(' '):
        if i in option:
            if i == '-d':
                print("Upload date format DMY\nexample : 2023-06-01 ")
                upload_date = input('Enter upload date : ')
                msg += f' -d {upload_date}'
            if i == '-o':
                owner_name = input('Enter owner username : ')
                msg += f' -o {owner_name}'
            if i == '-e':
                filename = input('Enter file extension : ')
                msg += f' -e {filename}'
    return msg

```

This function will handle the command before performing the Search feature.

```
def Search(message):
    option = ['-d', '-o', '-e']
    message = message.split(' ')
    op = [0]*3
    for msg in message:
        if msg in option:
            if(msg == '-d'):
                op[0] = message[message.index(msg) + 1]
            if(msg == '-e'):
                op[1] = message[message.index(msg) + 1]
            if(msg == '-o'):
                op[2] = message[message.index(msg) + 1]
    filter = {}
    if(op[0]):
        filter['upload_date'] = op[0]
    if(op[1]):
        filter['extension'] = op[1]
    if(op[2]):
        filter['owner'] = op[2]
    result = file_collection.find(filter)
    to_print = []
    for i, doc in enumerate(list(result), start=0):
        to_print.append([doc['_id'],
            doc['filename'], doc['owner'], doc['upload_date'], doc['sha256']])
    if(len(to_print) > 0):
        print(tabulate(to_print, headers=['ID', 'File Name', 'Owner', 'Upload
Date', 'SHA256'], tablefmt='grid'))
    else:
        print("[!] Found 0 file")
```

4. Testing

4.1 Testing

4.1.1 Uploading and Downloading

```

>> /help

===== HELP MENU =====
/register                : register
/login                  : login
/verify                 : verify your account
/key                   : get your private key
/upload                : choose file to upload
/download              : download files
/search [option : -d,-o,-e] : search files
-d : search by upload date
-o : search by file owner name
-n : search by file extension
=====

>> /register
[+] Enter username : alice123
[+] Enter password :
[+] Confirm password :
>> [NOTI] Registered! Please login and verify to continue...
Press Enter to continue...

>> /login
[+] Enter username : alice123
[+] Enter password :
>> [NOTI] Logged in.
Press Enter to continue...

>> /key
[!] : You must verified first
>> /verify
[+] Enter fullname : alice
[+] Enter day of birth : 24/04/2003
[+] Enter cccd : 089203000123
>> [NOTI] You are Verified!
Press Enter to continue...

>> /key
>> [NOTI] Key received
Press Enter to continue...

>> /upload
[+] Path to File : /mnt/f/Cryptography/Example/BaoCao.docx
[+] Policy : ((STAFF AND HCM) OR MANAGER)
[NOTI] File Uploaded
>> █

```

Figure 4.1

Here we can see that alice registered an account with username **alice123**, then she tried to get the key immediately but failed. Therefore, she had to verify her identity with 3 information: name, dob and national id.

After verifying, she can request her keys pair and performed upload action. Alice uploaded a .docx file with policy : “((STAFF AND HCM) OR MANAGER)”

```

_id: ObjectId('648953fb4a19aa03c4ed2a64')
name: "bob"
dob: "23/05/2003"
cccd: "922b46de817c490254c479cc06d093cb54fb4e374a932f5b8dbb90c287b2d428"
bhyt: "63749183"
role: "staff"
location: "hcm"
dow: "02/06/2023"
gender: "male"
registered: true
username: "bobby"

_id: ObjectId('6489c213e6a32a77664d0e64')
name: "charlie"
dob: "13/02/2003"
cccd: "7dbbefe524ca0afec2de9d5816cae189db8f893db67e7eceae384da8df747b13"
bhyt: "72930192"
role: "staff"
location: "hn"
dow: "10/06/2023"
gender: "female"
registerd: false

_id: ObjectId('6489c243e6a32a77664d0e65')
name: "david"
dob: "30/11/2003"
cccd: "ec6bb7800432303ad84de9ded21c8d625175be587469ea6c71327e014c90122c"
bhyt: "98726417"
role: "manager"
location: "hn"
dow: "10/06/2023"
gender: "male"
registered: false
username: ""

```

Figure 4.2 Other employees information

As shown in Figure 4.2, we expect that David and Bob can download the file because their attributes match the policy of Alice. Let's test that out.

```

>> /login
[+] Enter username : bobby
[+] Enter password :
>> [NOTI] Logged in.
Press Enter to continue...

>> /key
>> [NOTI] Key received
Press Enter to continue...

>> /search -o
Enter owner username : alice123
+-----+-----+-----+-----+-----+
| ID | File Name | Owner | Upload Date | SHA256 |
+-----+-----+-----+-----+-----+
| 64928d802643406338ab3fe9 | BaoCao.docx.scd | alice123 | 2023-06-21 | 0737b2c249e3dc0533a50a4bb05b904eff26603ab10262c7dbcdc48c7f3d5ccf |
+-----+-----+-----+-----+-----+
>> /download
Enter file name : BaoCao.docx.scd
Enter path to save file : /mnt/f/Cryptography/Downloads
[NOTI] : File downloaded at /mnt/f/Cryptography/Downloads
>>

```

Figure 4.3 Bob downloaded the file

```

>> /login
[+] Enter username : charlie123
[+] Enter password :
>> [NOTI] Logged in.
Press Enter to continue...

>> /key
>> [NOTI] Key received
Press Enter to continue...

>> /search -d -o
Upload date format DMY
example : 2023-06-01
Enter upload date : 2023-06-21
Enter owner username : alice123
+-----+-----+-----+-----+-----+
| ID | File Name | Owner | Upload Date | SHA256 |
+-----+-----+-----+-----+-----+
| 64928d802643406338ab3fe9 | BaoCao.docx.scd | alice123 | 2023-06-21 | 0737b2c249e3dc0533a50a4bb05b904eff26603ab10262c7dbcdc48c7f3d5ccf |
+-----+-----+-----+-----+-----+
>> /download
Enter file name : BaoCao.docx.scd
Enter path to save file : /mnt/f/Cryptography/Downloads
Policy not satisfied.
[NOTI] : You are not allowed to download this file!
>>

```

Figure 4.4 Charlie downloaded the file

As expected, Charlie download request was denied because her attributes did not satisfied the policy.

4.1.2 Searching

In section 3.2.3.1.5 we mentioned our searching mechanism, users can filter the files by 3 supported fields :

- Upload Date
- File Type
- Owner Name

```
>> /search -o
Enter owner username : bobby
+-----+-----+-----+-----+-----+
| ID | File Name | Owner | Upload Date | SHA256 |
+-----+-----+-----+-----+-----+
| 649291fc34eeef19f4fba4be | Presentation.pptx.scd | bobby | 2023-06-21 | 6614ca3c1526155dd5a6de5b7ce78a297547cf100ab2d7dda67929da89223eae |
+-----+-----+-----+-----+-----+

>> /search -d
Upload date format DMY
example : 2023-06-01
Enter upload date : 2023-06-17
+-----+-----+-----+-----+-----+
| ID | File Name | Owner | Upload Date | SHA256 |
+-----+-----+-----+-----+-----+
| 649291c342039cd829907177 | report.pdf.scd | charlie123 | 2023-06-17 | e5d98378be13bef33c11a1aef47f8df0f412fbd9c9b1e02a945ad9298b05319b |
+-----+-----+-----+-----+-----+

>> /search -e
Enter file extension :xlsx
+-----+-----+-----+-----+-----+
| ID | File Name | Owner | Upload Date | SHA256 |
+-----+-----+-----+-----+-----+
| 6492919fe37a1abc80e3a9a6 | Salary.xlsx.scd | david123 | 2023-06-21 | fccabece618bf80caa48da407796cf2a0f96320645be892ed804be8de2e50b13 |
+-----+-----+-----+-----+-----+

>> /search -e -d
Enter file extension : docx
Upload date format DMY
example : 2023-06-01
Enter upload date : 2023-06-21
+-----+-----+-----+-----+-----+
| ID | File Name | Owner | Upload Date | SHA256 |
+-----+-----+-----+-----+-----+
| 64928d802643406338ab3fe9 | BaoCao.docx.scd | alice123 | 2023-06-21 | 0737b2c249e3dc0533a50a4bb05b904eff26603ab10262c7dbcdc48c7f3d5ccf |
+-----+-----+-----+-----+-----+

>> /search -o -e
Enter owner username : bobby
Enter file extension : docx
[!] Found 0 file
>>
```

5. Summary

5.1 Results

There are 3 objectives we mentioned in section 1.2.3

1. **Confidentiality** - Ensuring that sensitive data stored in the cloud is protected from unauthorized access or disclosure.
2. **Integrity** - Ensuring that data stored in the cloud is not tampered with or modified in an unauthorized manner.
3. **Authorization** - Ensuring that users accessing the cloud-based data have the appropriate level of access permissions and privileges.

Certainly. Our solution has successfully achieved the three primary objectives that were set out. Firstly, we have implemented a robust user authentication system using a Certificate Authority (CA). This approach ensures that only authenticated users can access the system, adding an extra layer of security to protect against unauthorized access and potential breaches.

Secondly, we have incorporated the Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) to maintain data integrity. This encryption mode provides authentication and confidentiality, ensuring that the data is not tampered with during transmission or storage. Additionally, our solution provides file hashing, which allows users to check the integrity of files and detect any unauthorized modifications.

Finally, we have implemented a versatile access control mechanism using the Ciphertext Policy Attribute-Based Encryption (CP-ABE) technique. This approach enables us to provide fine-grained access control to users in a way that is both flexible and efficient. Users can be granted access to specific data based on their attributes, such as their role or department, without having to individually manage access permissions for each user.

Overall, our solution provides a comprehensive and effective security system that addresses the critical issues of user authentication, data integrity, and access control. By utilizing these advanced techniques, we have developed a robust solution that can provide secure access to sensitive data, while also protecting against potential threats and breaches.

5.2 Tasks Chart

Task	Anh Đức	Hữu Tiến	Thanh Đuẩn
Client Side Coding	X		
Server Side Coding		X	X
Applied Cryptography	X		
System Design	X	X	
Testing	X	X	X

5.3 Final Words

Dear PhD Nguyen Ngoc Tu,

As the semester comes to a close, we would like to take this opportunity to thank you for your wonderful lectures and advice during this project. I am very grateful for the knowledge and abilities you have taught to us; your knowledge and passion have been a tremendous benefit to our learning process.

Your enthusiasm for the topic and your ability to concisely and clearly explain complicated ideas have positively motivated each and every one of us. We are very lucky to have had you as our instructor for this project because of your dedication to supporting our academic progress and development.

We appreciate the time and effort that you have invested in us, and we are confident that the knowledge and skills that I have gained through your instruction will serve me well in my future endeavors. Once again, thank you for your support and guidance, and we wish you all the best in your future endeavors.

Sincerely,

Nguyen Tran Anh Duc

Nguyen Huu Tien

Le Thanh Duan

END.