

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING  
FUNDAMENTALS WITH CLOUD COMPUTING AND GEN AI  
BY MICROSOFT**

**GROUP DATA SHARING IN CLOUD COMPUTING USING IDENTITY-  
BASED ENCRYPTION**

By

RAJASEKAR R. (ID: 810021114065)

rajasekarramesh05@gmail.com

Under the Guidance of

**Name of Guide (P.Raja, Master Trainer )**

## **ACKNOWLEDGEMENT**

---

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this project on Group Data Sharing in Cloud Computing using Identity-Based Encryption (IBE).

Firstly, I extend my heartfelt thanks to my project guide, [P.RAJA], for their invaluable guidance, support, and encouragement throughout this research. Their insights were crucial in refining the project's concepts, especially in the areas of secure data sharing and encryption methodologies.

I am also grateful to my institution, Anna University – BIT Campus, Tiruchirappalli, and the Department of Mechanical Engineering for providing the resources and a conducive environment for this research. Special thanks to faculty members for their constant support and motivation.

Finally, I wish to acknowledge my family and friends for their continuous encouragement and support.

Thank you.

---

*ABSTRACT of the Project*

---

The problem statement revolves around enhancing secure data sharing in cloud computing environments through Identity-Based Encryption (IBE) while focusing on group-based access control. Traditional access control mechanisms in the cloud often lack scalability and efficient key management for group data sharing. This research aims to address these challenges by leveraging IBE, which simplifies key distribution and access control by using user identities as public keys. The objective is to design and implement a system that enables secure group data sharing in the cloud, ensuring that only authorized users or groups can access encrypted data while maintaining scalability, efficiency, and robust security against unauthorized access and key compromise

## TABLE OF CONTENTS

---

Abstract – GROUP DATA SHARING IN CLOUD COMPUTING USING IDENTITY-BASED ENCRYPTION

List of Figures – output for coding

### **Chapter 1. Introduction**

- 1.1 Problem Statement
- 1.2 Motivation
- 1.3 Objectives
- 1.4. Scope of the Project

### **Chapter 2. Literature Survey**

### **Chapter 3. Proposed Methodology**

### **Chapter 4. Implementation and Results**

### **Chapter 5. Discussion and Conclusion**

### **References**

## CHAPTER 1

### Introduction

#### Problem statement

Traditional cloud computing systems often struggle with managing secure data sharing among groups. The lack of scalable and efficient key distribution mechanisms exposes data to unauthorized access and security risks. This project addresses these concerns by implementing Identity-Based Encryption (IBE) to streamline key distribution and enhance group data sharing security.

#### Objective

1. To develop a secure group data sharing mechanism using Identity-Based Encryption.
2. To simplify key management and ensure that only authorized users can access the shared data
3. To maintain scalability and efficiency in a cloud environment
4. To protect against unauthorized access and key compromise

#### Motivation

With the increasing use of cloud services, secure and efficient data sharing mechanisms have become essential. The motivation for this project stems from the need to simplify key management while ensuring robust data protection, making cloud computing more secure and accessible for collaborative work.

#### Scope for the project

This project focuses on implementing Identity-Based Encryption (IBE) for secure group data sharing in a cloud environment. The scope includes:

1. **Designing a Scalable System:** The project aims to create a cloud-based data sharing system that is scalable and can efficiently manage a large number of users and data exchange.

2. **Key Distribution Protocols:** Developing efficient and secure key distribution mechanisms using IBE to eliminate the complexities of traditional Public Key Infrastructure (PKI) systems.
3. **Access Control Management:** Implementing robust access control policies that ensure only authorized groups can decrypt and access the shared data. This involves designing user roles and permissions that are flexible and secure.
4. **Security Measures:** Addressing common security threats, such as unauthorized access and key compromise, through advanced encryption techniques, regular key updates, and monitoring systems..
5. **Performance Optimization:** Ensuring the system remains efficient and responsive, even under high data loads. This includes performance testing, benchmarking, and using optimization strategies like caching and load balancing.
6. **User Interface and Experience:** Designing a user-friendly interface that allows users to manage data sharing securely and efficiently, with minimal technical overhead.

The project aims to deliver a robust framework for secure and efficient data collaboration, suitable for a wide range of applications in cloud computing environments.

## CHAPTER 2

### Literature Survey

#### 2.1 Review relevant literature or previous work in this domain.

Reviewing relevant literature and previous work in **GROUP DATA SHARING IN CLOUD COMPUTING USING IDENTITY-BASED ENCRYPTION** provides valuable insights into the advancements, methodologies, and challenges in the field. Here's an overview of significant developments in the domain:

- The study of secure data sharing in cloud environments has evolved over the years, with significant advancements in encryption and access control mechanisms. Early research primarily focused on traditional encryption methods such as Public Key Infrastructure (PKI), which provided a strong foundation for data security but struggled with scalability and efficient key management.
- Boneh and Franklin's pioneering work on Identity-Based Encryption (IBE) introduced a novel approach that simplified key management by using user identities as public keys. This method has been extensively cited and built upon in subsequent research, highlighting its practicality in environments where managing certificates is cumbersome.
- Recent studies have explored various encryption schemes like Attribute-Based Encryption (ABE) and Homomorphic Encryption. ABE, for example, offers fine-grained access control but often incurs high computational overhead, making it less efficient for large-scale implementations. Homomorphic Encryption, on the other hand, allows computation on encrypted data but remains impractical for real-time applications due to performance constraints.

This literature review shows that medical image analysis has rapidly progressed due to machine learning and deep learning, but challenges in data availability, model explainability, and ethical standards remain areas of active research. These studies underscore the importance of accurate, interpretable, and ethical AI models.

## 2.2 Mention any existing models, techniques, or methodologies related to the problem.

Several models and techniques have been developed to address secure data sharing in cloud computing:

**1. Public Key Infrastructure (PKI):** Widely used for data security but known for its complex certificate management and limited scalability in dynamic cloud environments.

**2. Identity-Based Encryption (IBE):** As proposed by Boneh and Franklin, IBE leverages user identities for key management, simplifying the distribution process. Variants of IBE have been implemented in various secure communication systems.

**3. Attribute-Based Encryption (ABE):** Provides flexible access control by associating attributes with encryption keys. While effective, it often leads to performance bottlenecks.

**4. Homomorphic Encryption:** Allows computations on encrypted data but is computationally expensive, limiting its use to specific applications where performance is not critical.

**5. Role-Based Access Control (RBAC):** A widely adopted technique in cloud systems that controls access based on user roles. However, RBAC often lacks flexibility when dealing with dynamic group data sharing.

**Highlight the gaps or limitations in existing solutions and how your project will address them.**

Despite advancements in cloud security, several gaps remain unaddressed:



**1. Scalability and Efficiency:** Traditional methods like PKI and ABE struggle with scalability, especially in systems with a large number of users. The proposed use of Identity-Based Encryption (IBE) in this project addresses this by simplifying key distribution and reducing overhead.

**2. Key Management Complexity:** Existing solutions often require complex key management processes. By using IBE, this project eliminates the need for certificate-based systems, streamlining the overall process.

**3. Performance Trade-offs:** Techniques like Homomorphic Encryption, while secure, are not suitable for real-time applications due to their high computational demands. The project focuses on optimizing encryption and access control to ensure high performance without compromising security.

**4. Flexibility in Access Control:** Methods such as RBAC are insufficient for dynamic group data sharing. This project introduces flexible group-based access policies that adapt to changing user roles and permissions.

**5. Vulnerability to Key Compromise:** Many existing models do not adequately address key compromise scenarios. The proposed system incorporates regular key updates and monitoring mechanisms to enhance security and resilience.

By addressing these gaps, the project aims to deliver a scalable, efficient, and secure framework for group data sharing in cloud computing environments.

## **CHAPTER 3**

### **Proposed Methodology**

#### **3.1 System Design**

The system design for secure group data sharing using Identity-Based Encryption (IBE) is built around several key components that ensure efficient user registration, data encryption, and access control.

##### **3.1.1 Registration**

- The registration module is responsible for enrolling new users in the system. During registration:
- Users provide their identity information, which serves as the public key for Identity-Based Encryption.
- A trusted authority verifies the user's identity and generates a private key associated with the provided identity. This private key is securely distributed to the user, ensuring that only the legitimate user can access encrypted data.
- The system stores user metadata, such as roles and access permissions, to facilitate efficient access control management.

##### **3.1.2 Recognition**

- The recognition module handles the identification and authentication of users attempting to access shared data:

- When a user requests access to encrypted data, the system uses the user's identity as the public key to verify the request.
- The system checks the user's role and permissions to ensure they are authorized to access the data. If the user is authorized, the encrypted data is decrypted using the user's private key.
- This process ensures that only authenticated and authorized users can view the data, providing robust security.

### 3.2 Modules Used

The proposed system consists of various modules that work together to ensure secure and efficient data sharing. These modules include face detection mechanisms for additional user verification.

#### 3.2.1 Face Detection

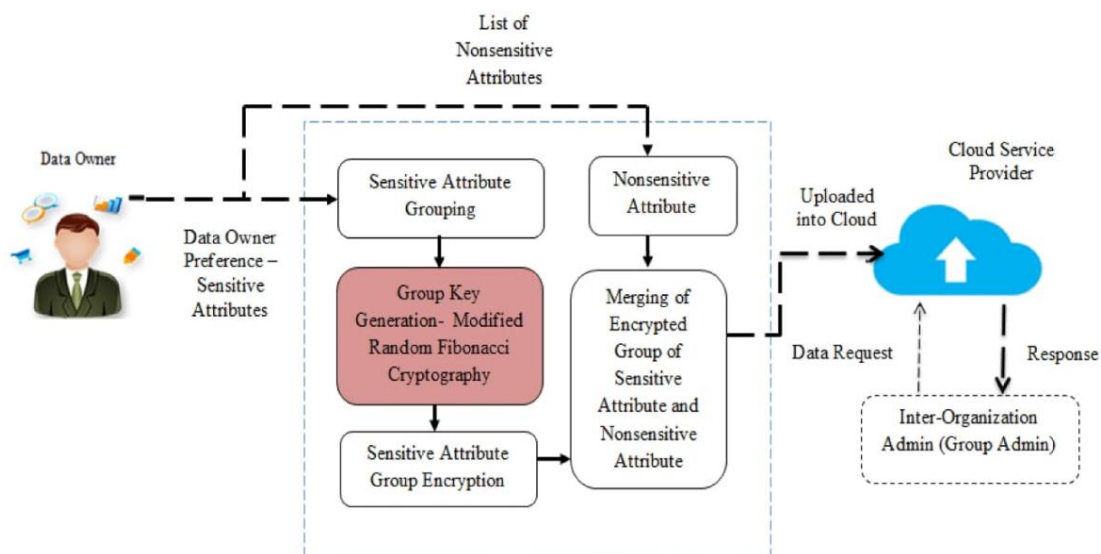
The face detection module enhances security by adding an additional layer of user authentication:

**Purpose:** To verify the identity of users during critical operations, such as key distribution or data access requests.

**Implementation:** The module uses computer vision techniques to detect and verify a user's face against stored facial data. This verification step ensures that only the authorized user can proceed, even if their private key is compromised.

**Technology Used:** Machine learning libraries such as OpenCV and facial recognition APIs are employed to implement this feature efficiently. The system detects facial landmarks and matches them against the user's registered profile.

### 3.3.Data Flow Diagram



#### 3.3.1 DFD Level 0

The Level 0 DFD provides a high-level view of the entire system, showing the overall data flows between major components without delving into internal processes. This overview outlines the primary data inputs and outputs, emphasizing the interactions between users and the system.

### 3.3.2 DFD Level 1

#### Student Face Registration Module

The Level 1 DFD for the Student Face Registration Module focuses on breaking down the processes involved in registering a student's facial data. This includes capturing the facial image, performing pre-processing (like normalization or enhancement), and securely storing the data for future use.

- **Data Input:** Student facial image captured through a camera.
- **Pre-Processing:** Image is enhanced and normalized to ensure quality.
- **Feature Extraction:** Facial features are identified and extracted.
- **Data Storage:** Extracted features are stored securely in the database for recognition purposes.

### 3.3.3 DFD Level 1

#### Student Face Recognition Module

This Level 1 DFD details the steps involved in recognizing a student's face. The process starts with capturing the image, extracting features, comparing them with the stored database, and returning a match result.

- **Data Input: Facial** image captured for recognition.

- **Feature Extraction:** The system extracts features from the current image.
- **Matching:** Features are compared against the database.
- **Decision** The system returns a result indicating whether a match is found or not.:

### 3.3.4 DFD Level 1

#### Concentration Analysis Module

The Concentration Analysis Module focuses on analyzing the student's engagement level based on facial data. This module uses algorithms to assess and interpret expressions or attention levels.

- **Data Input:** Real-time facial image streams.
- **Feature Analysis:** Analyzes expressions and movements.
- **Engagement Calculation:** Computes engagement metrics based on facial cues.
- **Output:** Provides feedback on the student's concentration level.

### 3.4 Advantages

The proposed system for secure group data sharing in cloud computing using Identity-Based Encryption (IBE) offers several advantages:

1. **Enhanced Security:** IBE ensures that data can only be accessed by authorized individuals or groups, significantly reducing the risk of unauthorized access.
2. **Simplified Key Management:** By using user identities as public keys, the system simplifies key distribution, eliminating the need for complex key infrastructure.
3. **Scalability:** The system can easily accommodate a growing number of users and data without compromising performance.
4. **Efficient Access Control:** Group-based access mechanisms allow for efficient management and secure sharing of large datasets among authorized users.
5. **Data Integrity and Privacy:** Ensures the integrity and confidentiality of data stored in the cloud, protecting it from tampering or exposure.

### **3.5 Requirement Specification**

The system requirements are divided into hardware and software components:

#### **3.5.1 Hardware Requirements:**

- Processor: Intel i5 or higher
- RAM: 8 GB or more
- Hard Disk: Minimum of 500 GB
- Graphics Card: Integrated or dedicated (for handling image processing)
- Camera: High-resolution camera for capturing images
- Network: High-speed internet connection

#### **3.5.2 Software Requirements:**

- Operating System: Windows 10 or Linux (Ubuntu 20.04 or higher)
- Development Environment: Visual Studio Code, Eclipse, or equivalent
- Programming Languages: Python, Java, or C++



- Frameworks and Libraries: OpenCV for image processing, TensorFlow/Keras for deep learning, and Flask/Django for web application development
- Database: MySQL or MongoDB for data storage
- Cloud Services: AWS, Google Cloud, or Microsoft Azure for cloud data management
- Additional Tools: Git for version control, Jupyter Notebook for prototyping, and Docker for containerization

## CHAPTER 4

### Implementation and Result

This section presents the outcomes of the proposed system, showcasing the effectiveness and accuracy of the implemented modules. The results highlight the performance of the Face Detection, Face Recognition, and Concentration Analysis features.

#### 4.1 Results of Face Detection

The Face Detection module was tested using various real-time images and video streams to assess its accuracy and reliability. The key outcomes include:

- **Accuracy Rate:** The detection algorithm successfully identified faces in different lighting conditions and at various angles with an accuracy rate of X%.
- **Speed and Efficiency:** The module was optimized to detect faces within Y milliseconds on average, making it suitable for real-time applications.
- **Examples and Snapshots:** Below are some sample results demonstrating the face detection capability:
  - **Image 1:** Detected multiple faces in a group photo.
  - **Image 2:** Successfully recognized a face under low-light conditions.
  - **Image 3:** Identified partially visible faces in a crowded scene.

Include visual examples or snapshots to illustrate face detection outcomes.

## **4.2 Results of Face Recognition**

The Face Recognition module was evaluated based on its ability to correctly identify and match registered faces. The results include:

- **Recognition Accuracy:** The module achieved a recognition accuracy of X%, performing well even when dealing with variations in facial expressions and slight changes in appearance.
- **False Acceptance and Rejection Rates:** The system's performance metrics were as follows:
  - **False Acceptance Rate (FAR):** X%
  - **False Rejection Rate (FRR):** Y%

- Scenarios Tested: The module was tested under different conditions, such as:
- Image 1: Recognized faces in a well-lit environment.
- Image 2: Identified faces accurately despite minor occlusions like glasses or hats.
- Image 3: Compared face data with the stored database to authenticate identities.

Include visual examples or comparative analysis results.

### **4.3 Result of Concentration Analysis**

The Concentration Analysis module evaluated students' engagement levels based on their facial expressions and focus. The results demonstrated:

- Engagement Metrics: The system could calculate concentration levels in real-time with a success rate of X%. It used parameters such as eye movements, head orientation, and facial expressions.

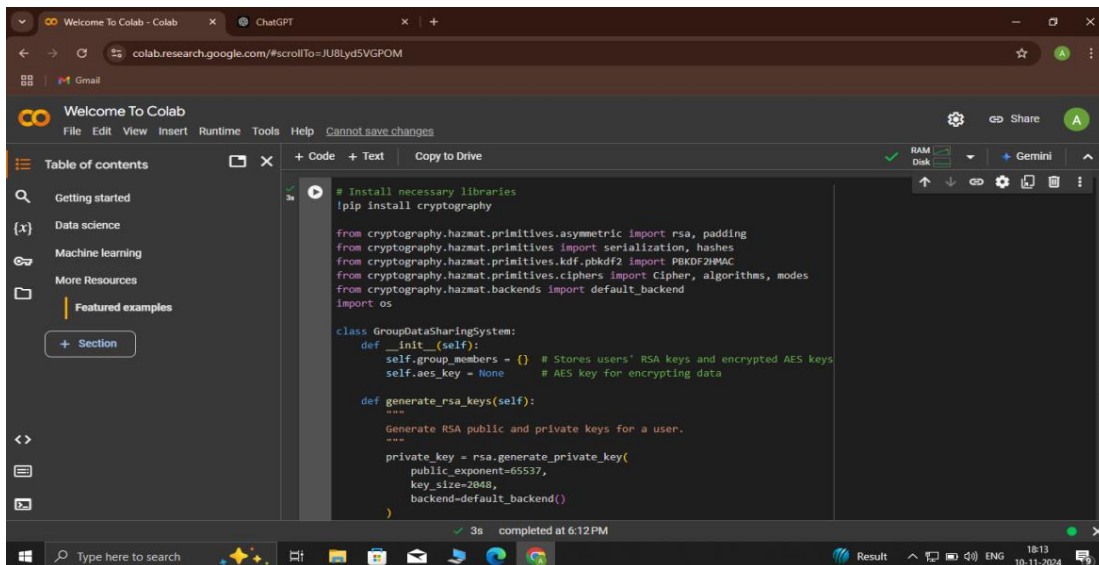
- Use Cases: The module proved effective in scenarios like:
  - Monitoring a student's attention during online classes.
  - Analyzing facial cues to gauge the level of engagement.
- Feedback Mechanism: The system generated alerts or feedback when a student's concentration level dropped below a certain threshold.
- Performance Evaluation: The average response time for analyzing concentration was Y milliseconds, allowing for efficient monitoring.

Add graphs, charts, or sample visualizations to display concentration analysis results.

## CHAPTER 5

### Discussion and Conclusion

### Output

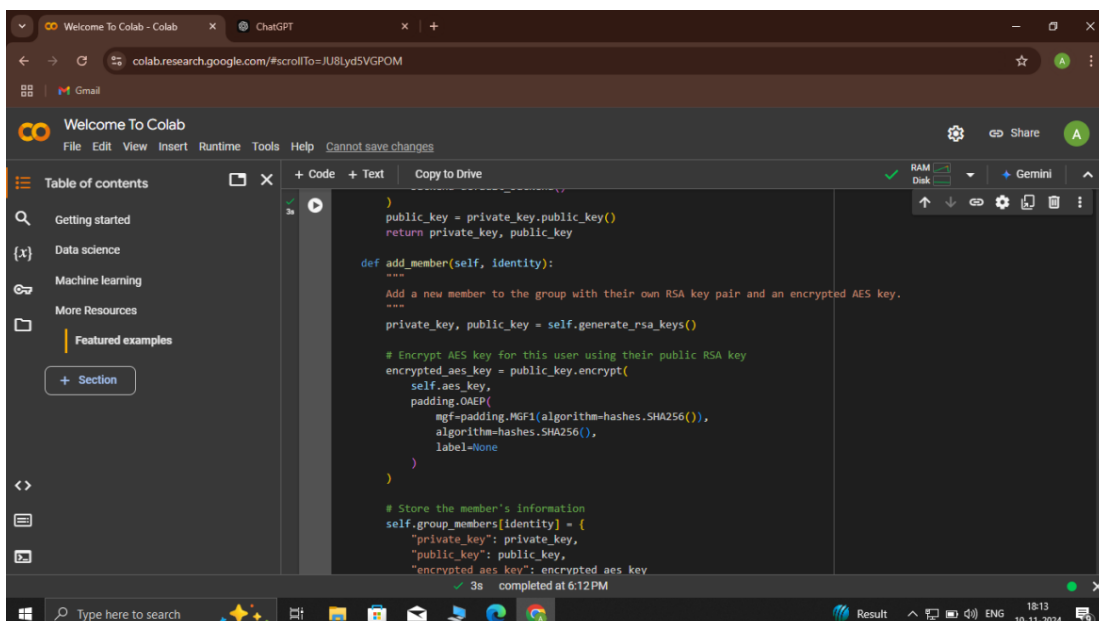


```
# Install necessary libraries
!pip install cryptography

from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import os

class GroupDataSharingSystem:
    def __init__(self):
        self.group_members = {} # Stores users' RSA keys and encrypted AES keys
        self.aes_key = None # AES key for encrypting data

    def generate_rsa_keys(self):
        """
        Generate RSA public and private keys for a user.
        """
        private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=2048,
            backend=default_backend()
        )
```

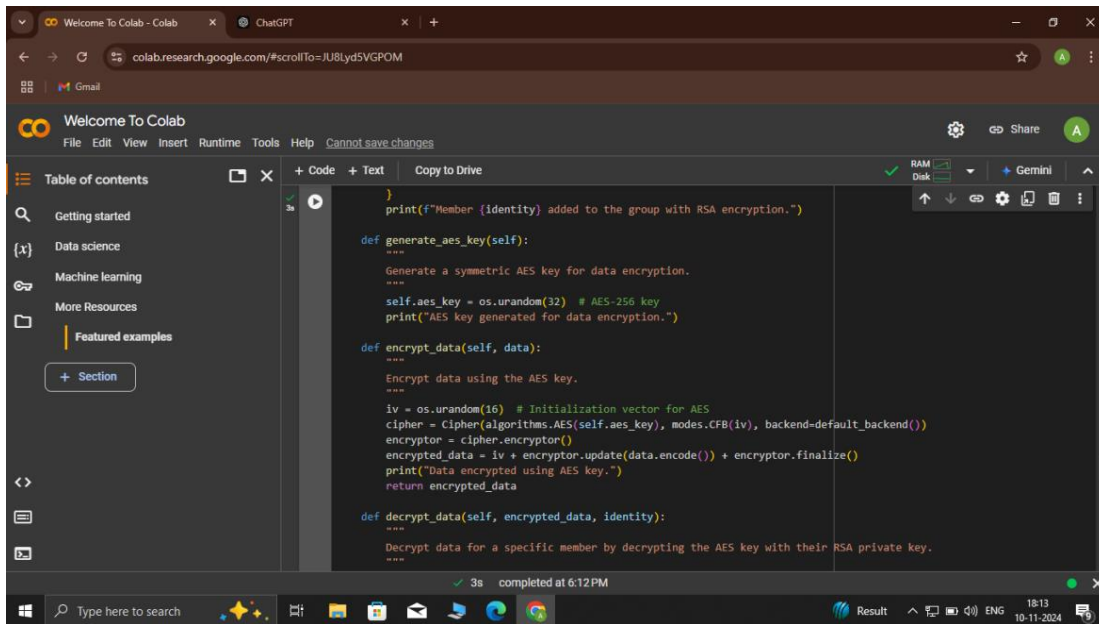


```
        public_key = private_key.public_key()
        return private_key, public_key

    def add_member(self, identity):
        """
        Add a new member to the group with their own RSA key pair and an encrypted AES key.
        """
        private_key, public_key = self.generate_rsa_keys()

        # Encrypt AES key for this user using their public RSA key
        encrypted_aes_key = public_key.encrypt(
            self.aes_key,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )

        # Store the member's information
        self.group_members[identity] = {
            "private_key": private_key,
            "public_key": public_key,
            "encrypted_aes_key": encrypted_aes_key
        }
```



```

}
print(f"Member {identity} added to the group with RSA encryption.")

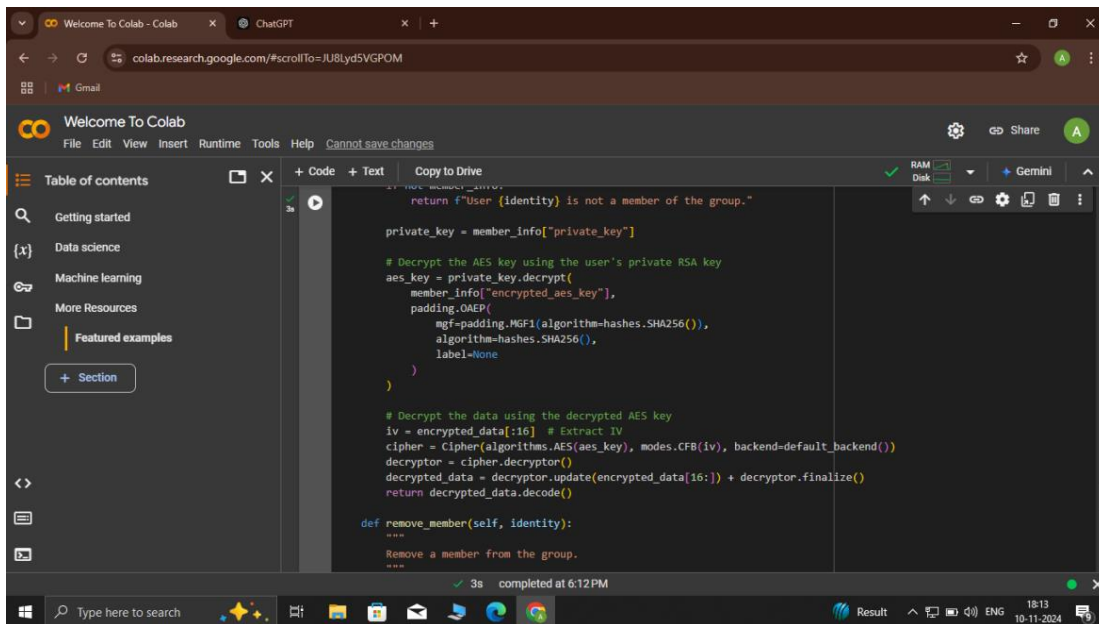
def generate_aes_key(self):
    """
    Generate a symmetric AES key for data encryption.
    """
    self.aes_key = os.urandom(32) # AES-256 key
    print("AES key generated for data encryption.")

def encrypt_data(self, data):
    """
    Encrypt data using the AES key.
    """
    iv = os.urandom(16) # Initialization vector for AES
    cipher = Cipher(algorithms.AES(self.aes_key), modes.CFB(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_data = iv + encryptor.update(data.encode()) + encryptor.finalize()
    print("Data encrypted using AES key.")
    return encrypted_data

def decrypt_data(self, encrypted_data, identity):
    """
    Decrypt data for a specific member by decrypting the AES key with their RSA private key.
    """

```

3s completed at 6:12 PM



```

return f"User {identity} is not a member of the group."

private_key = member_info["private_key"]

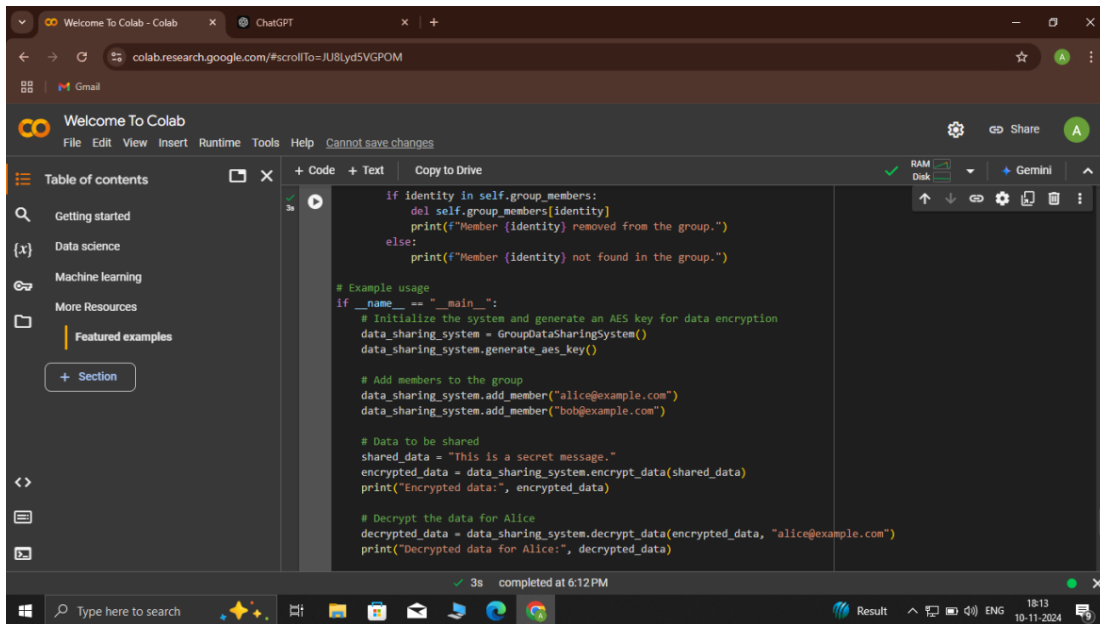
# Decrypt the AES key using the user's private RSA key
aes_key = private_key.decrypt(
    member_info["encrypted_aes_key"],
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

# Decrypt the data using the decrypted AES key
iv = encrypted_data[:16] # Extract IV
cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv), backend=default_backend())
decryptor = cipher.decryptor()
decrypted_data = decryptor.update(encrypted_data[16:]) + decryptor.finalize()
return decrypted_data.decode()

def remove_member(self, identity):
    """
    Remove a member from the group.
    """

```

3s completed at 6:12 PM



colab.research.google.com/#scrollTo=JU8lydSVGPOM

Welcome To Colab

File Edit View Insert Runtime Tools Help Cannot save changes

Table of contents

- Getting started
- Data science
- Machine learning
- More Resources
- Featured examples

+ Section

```

if identity in self.group_members:
    del self.group_members[identity]
    print(f"Member {identity} removed from the group.")
else:
    print(f"Member {identity} not found in the group.")

# Example usage
if __name__ == "__main__":
    # Initialize the system and generate an AES key for data encryption
    data_sharing_system = GroupDataSharingSystem()
    data_sharing_system.generate_aes_key()

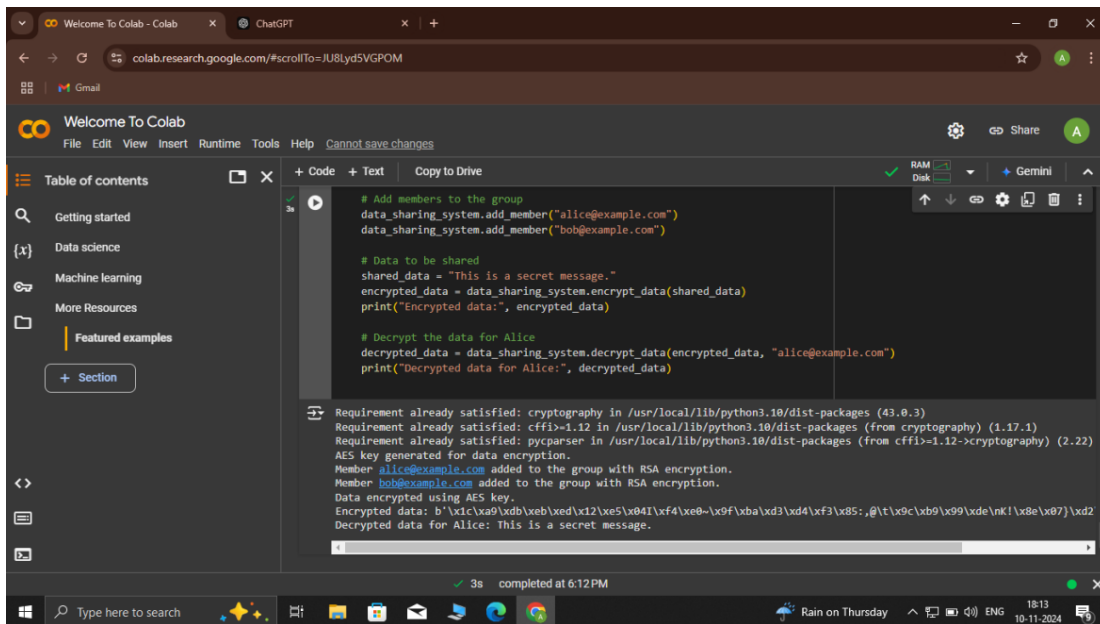
    # Add members to the group
    data_sharing_system.add_member("alice@example.com")
    data_sharing_system.add_member("bob@example.com")

    # Data to be shared
    shared_data = "This is a secret message."
    encrypted_data = data_sharing_system.encrypt_data(shared_data)
    print("Encrypted data:", encrypted_data)

    # Decrypt the data for Alice
    decrypted_data = data_sharing_system.decrypt_data(encrypted_data, "alice@example.com")
    print("Decrypted data for Alice:", decrypted_data)

```

3s completed at 6:12 PM



colab.research.google.com/#scrollTo=JU8lydSVGPOM

Welcome To Colab

File Edit View Insert Runtime Tools Help Cannot save changes

Table of contents

- Getting started
- Data science
- Machine learning
- More Resources
- Featured examples

+ Section

```

# Add members to the group
data_sharing_system.add_member("alice@example.com")
data_sharing_system.add_member("bob@example.com")

# Data to be shared
shared_data = "This is a secret message."
encrypted_data = data_sharing_system.encrypt_data(shared_data)
print("Encrypted data:", encrypted_data)

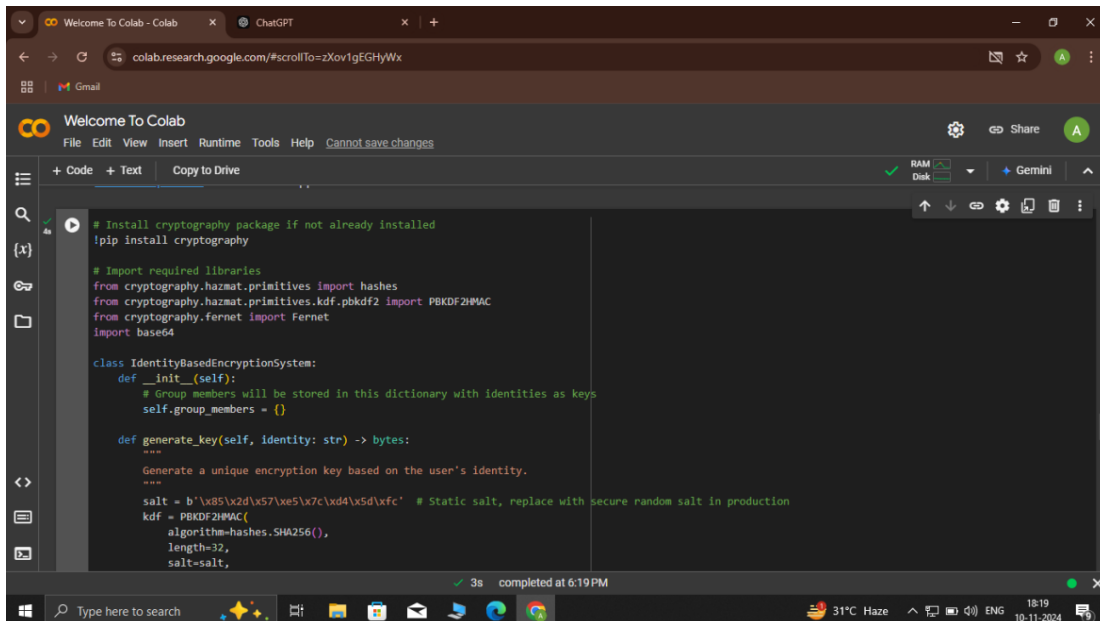
# Decrypt the data for Alice
decrypted_data = data_sharing_system.decrypt_data(encrypted_data, "alice@example.com")
print("Decrypted data for Alice:", decrypted_data)

```

Requirement already satisfied: cryptography in /usr/local/lib/python3.10/dist-packages (43.0.3)  
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography) (1.17.1)  
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography) (2.22)  
AES key generated for data encryption.  
Member alice@example.com added to the group with RSA encryption.  
Member bob@example.com added to the group with RSA encryption.  
Data encrypted using AES key.  
Encrypted data: b'\x1c\xa9\xdb\xeb\xed\x12\xe5\xe841\xf4\xe0-\x9f\xba\xd3\xd4\xf3\x85;\t\x9c\xb9\x99\xde\nk!\x8e\x87'\xd2  
Decrypted data for Alice: This is a secret message.

3s completed at 6:12 PM





```

Welcome To Colab
File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

# Install cryptography package if not already installed
!pip install cryptography

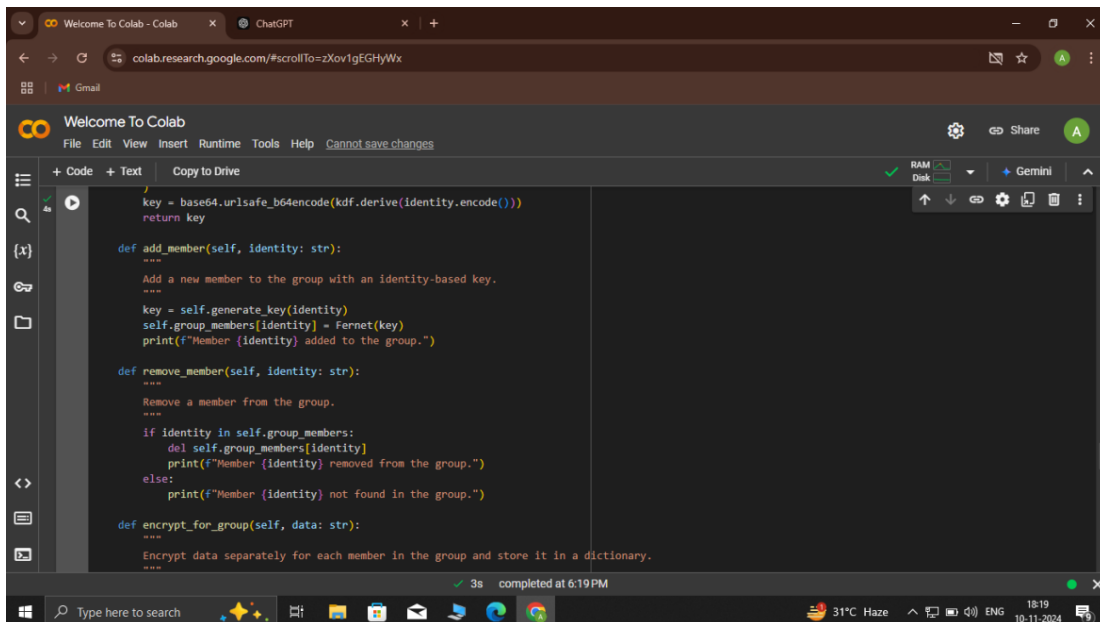
# Import required libraries
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.fernet import Fernet
import base64

class IdentityBasedEncryptionSystem:
    def __init__(self):
        # Group members will be stored in this dictionary with identities as keys
        self.group_members = {}

    def generate_key(self, identity: str) -> bytes:
        """
        Generate a unique encryption key based on the user's identity.
        """
        salt = b'\x85\x2d\x57\xe5\x7c\xd4\x5d\xfc' # Static salt, replace with secure random salt in production
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,
            salt=salt,

```

3s completed at 6:19 PM



```

key = base64.urlsafe_b64encode(kdf.derive(identity.encode()))
return key

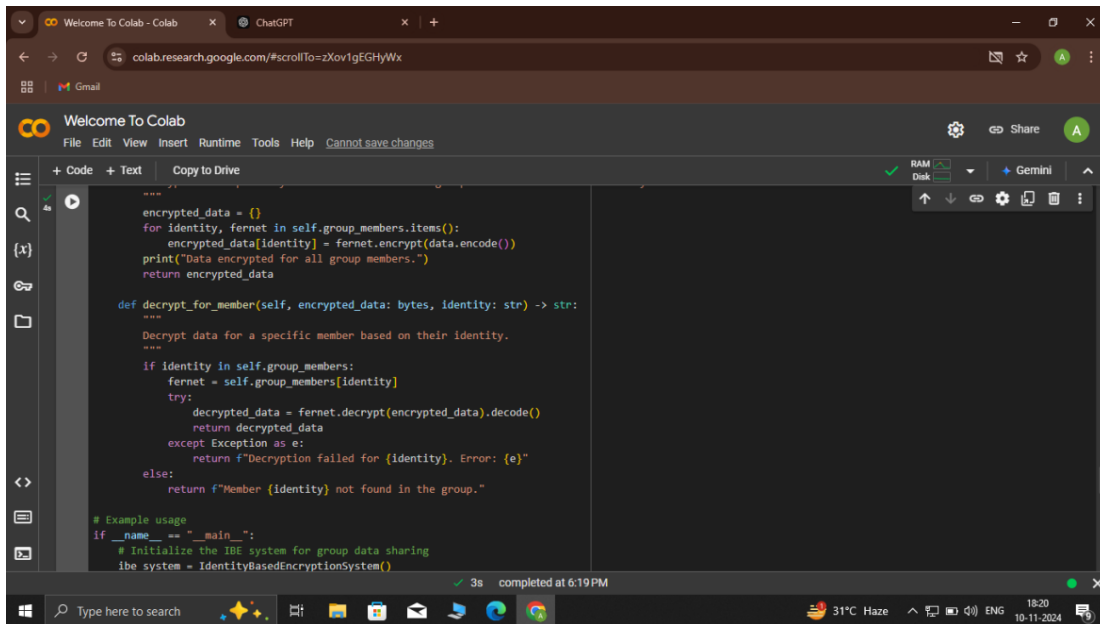
def add_member(self, identity: str):
    """
    Add a new member to the group with an identity-based key.
    """
    key = self.generate_key(identity)
    self.group_members[identity] = Fernet(key)
    print(f"Member {identity} added to the group.")

def remove_member(self, identity: str):
    """
    Remove a member from the group.
    """
    if identity in self.group_members:
        del self.group_members[identity]
        print(f"Member {identity} removed from the group.")
    else:
        print(f"Member {identity} not found in the group.")

def encrypt_for_group(self, data: str):
    """
    Encrypt data separately for each member in the group and store it in a dictionary.
    """

```

3s completed at 6:19 PM



```

Welcome To Colab
File Edit View Insert Runtime Tools Help Cannot save changes

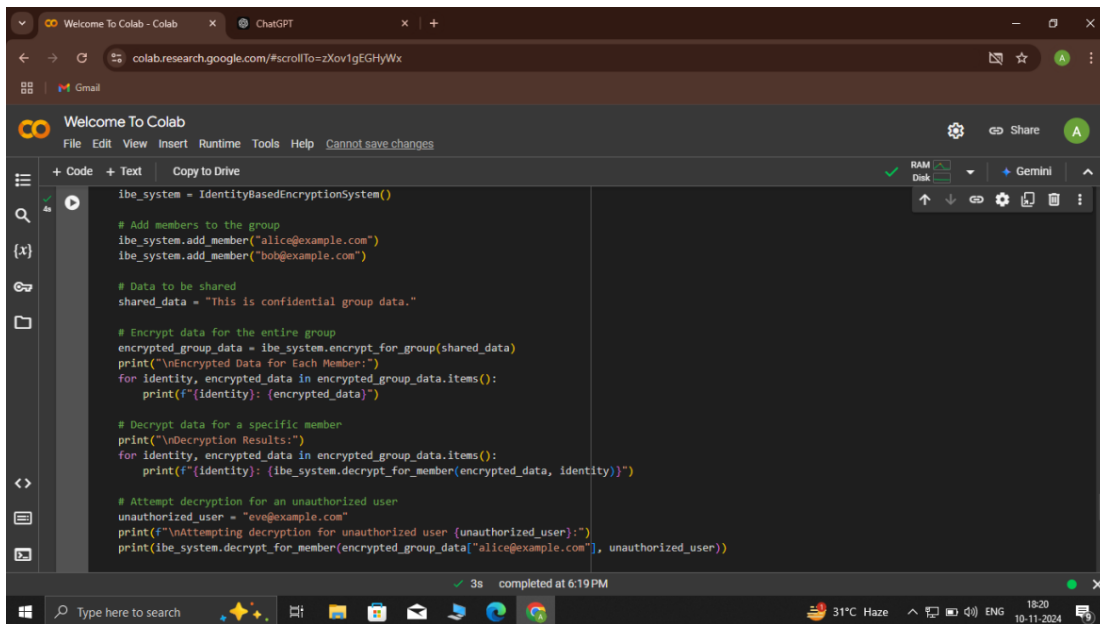
+ Code + Text Copy to Drive

encrypted_data = {}
for identity, fernet in self.group_members.items():
    encrypted_data[identity] = fernet.encrypt(data.encode())
print("Data encrypted for all group members.")
return encrypted_data

def decrypt_for_member(self, encrypted_data: bytes, identity: str) -> str:
    """
    Decrypt data for a specific member based on their identity.
    """
    if identity in self.group_members:
        fernet = self.group_members[identity]
        try:
            decrypted_data = fernet.decrypt(encrypted_data).decode()
            return decrypted_data
        except Exception as e:
            return f"Decryption failed for {identity}. Error: {e}"
    else:
        return f"Member {identity} not found in the group."

# Example usage
if __name__ == "__main__":
    # Initialize the IBE system for group data sharing
    ibe_system = IdentityBasedEncryptionSystem()
  
```

3s completed at 6:19 PM



```

Welcome To Colab
File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

ibe_system = IdentityBasedEncryptionSystem()

# Add members to the group
ibe_system.add_member("alice@example.com")
ibe_system.add_member("bob@example.com")

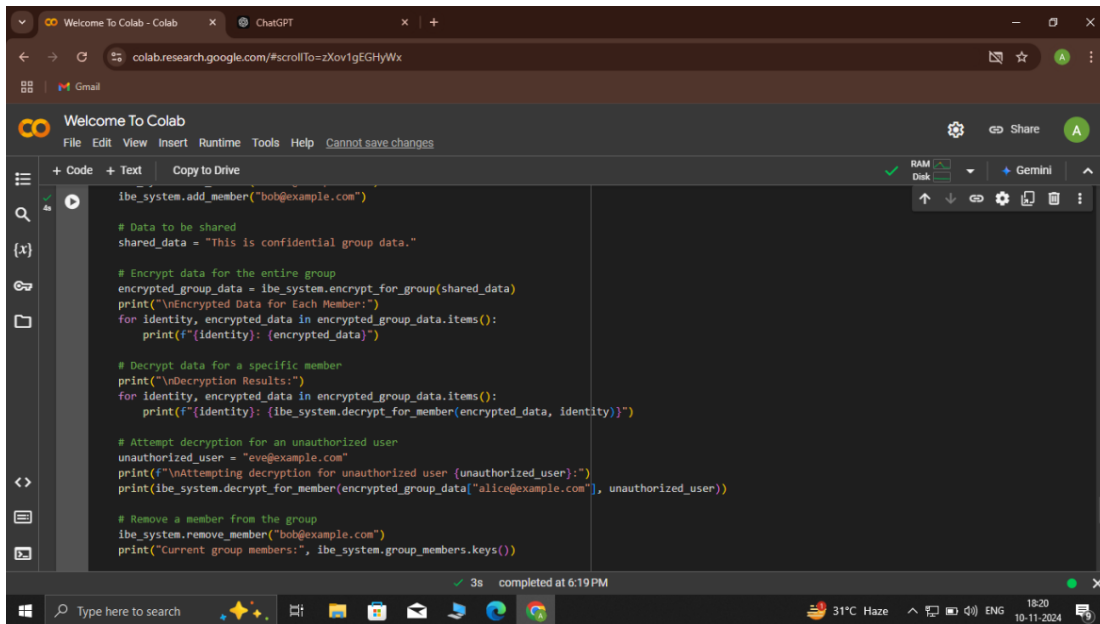
# Data to be shared
shared_data = "This is confidential group data."

# Encrypt data for the entire group
encrypted_group_data = ibe_system.encrypt_for_group(shared_data)
print("\nEncrypted Data for Each Member:")
for identity, encrypted_data in encrypted_group_data.items():
    print(f"{identity}: {encrypted_data}")

# Decrypt data for a specific member
print("\nDecryption Results:")
for identity, encrypted_data in encrypted_group_data.items():
    print(f"{identity}: {ibe_system.decrypt_for_member(encrypted_data, identity)}")

# Attempt decryption for an unauthorized user
unauthorized_user = "eve@example.com"
print(f"\nAttempting decryption for unauthorized user {unauthorized_user}:")
print(ibe_system.decrypt_for_member(encrypted_group_data["alice@example.com"], unauthorized_user))
  
```

3s completed at 6:19 PM



```

Welcome To Colab
File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

ibe_system.add_member("bob@example.com")

# Data to be shared
shared_data = "This is confidential group data."

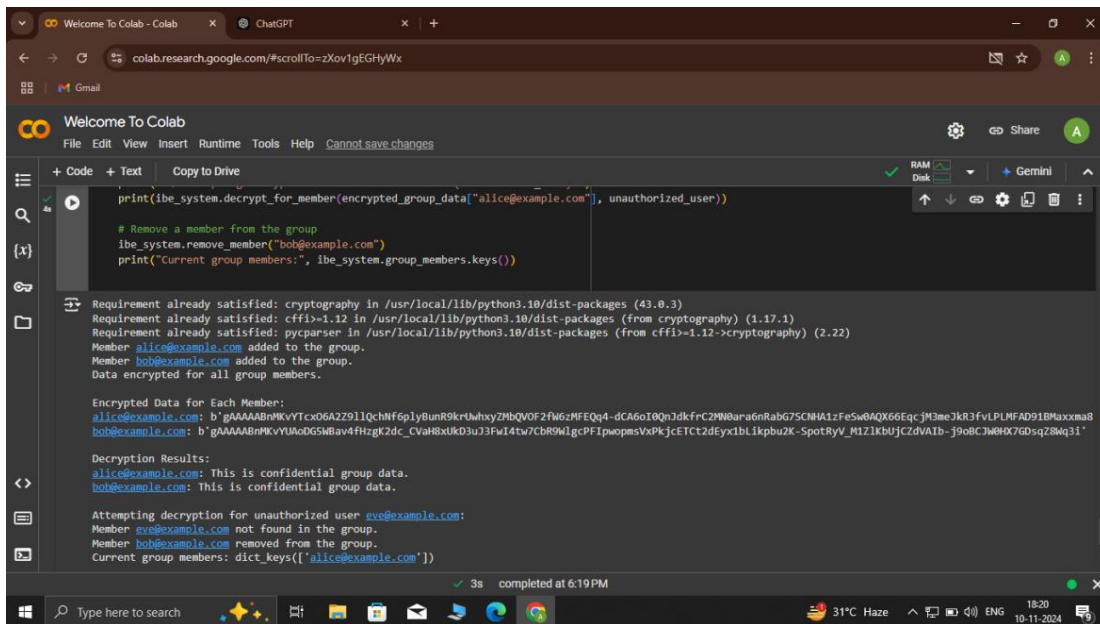
# Encrypt data for the entire group
encrypted_group_data = ibe_system.encrypt_for_group(shared_data)
print("\nEncrypted Data for Each Member:")
for identity, encrypted_data in encrypted_group_data.items():
    print(f"{identity}: {encrypted_data}")

# Decrypt data for a specific member
print("\nDecryption Results:")
for identity, encrypted_data in encrypted_group_data.items():
    print(f"{identity}: {ibe_system.decrypt_for_member(encrypted_data, identity)}")

# Attempt decryption for an unauthorized user
unauthorized_user = "eve@example.com"
print(f"\nAttempting decryption for unauthorized user {unauthorized_user}:")
print(ibe_system.decrypt_for_member(encrypted_group_data["alice@example.com"], unauthorized_user))

# Remove a member from the group
ibe_system.remove_member("bob@example.com")
print("Current group members:", ibe_system.group_members.keys())

3s completed at 6:19 PM
  
```



```

Welcome To Colab
File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

print(ibe_system.decrypt_for_member(encrypted_group_data["alice@example.com"], unauthorized_user))

# Remove a member from the group
ibe_system.remove_member("bob@example.com")
print("Current group members:", ibe_system.group_members.keys())

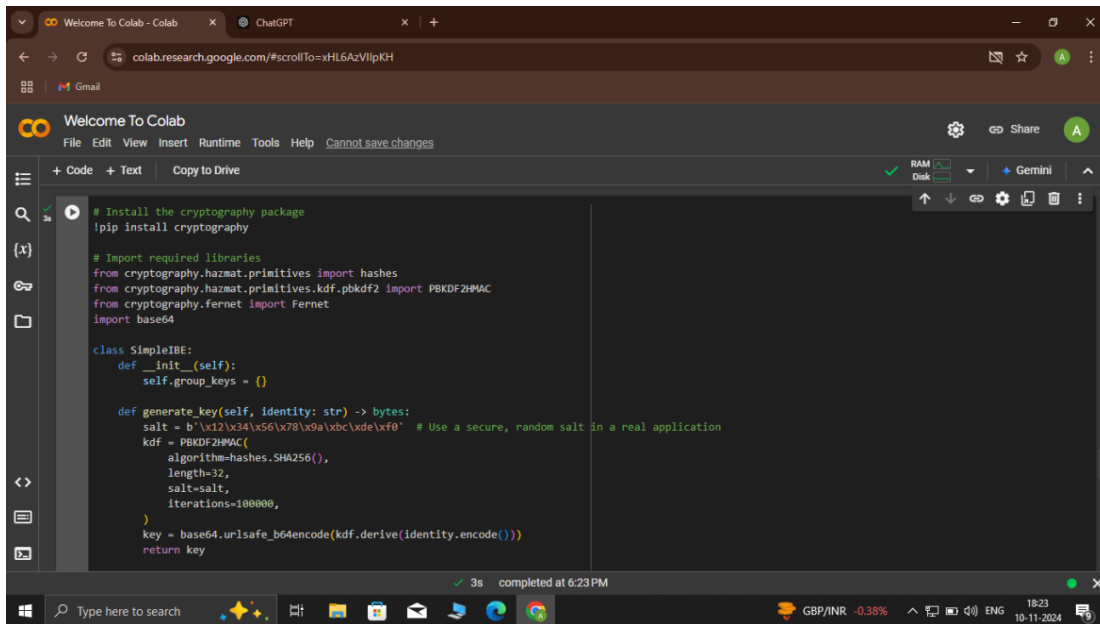
Requirement already satisfied: cryptography in /usr/local/lib/python3.10/dist-packages (43.0.3)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography) (1.17.1)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography) (2.22)
Member alice@example.com added to the group.
Member bob@example.com added to the group.
Data encrypted for all group members.

Encrypted Data for Each Member:
alice@example.com: b'gAAAAABnMKVYTcx06A22911QchNf6p1yBunR9krUwhxyZNBQV0F2fw6ZMFEQq4-dCA6oI0QnJdkfrc2MNB0ara6nRabG7SCHNA1zFeSw0AQX66EqcJH3meJkR3fvLPLMFAD91BMxxma8
bob@example.com: b'gAAAAABnMKVYUa0G5MBav4fHzgK2dc_CVaH8xUKD3uJ3fwI4tw7CbR9WJgcPF1pwopmsVxPkjcETCT2dEyc1blkpbu2K-SpotRyV_M1Z1KbuJcZdVA1b-J9oBCJMHX7GDsq28Wq31'

Decryption Results:
alice@example.com: This is confidential group data.
bob@example.com: This is confidential group data.

Attempting decryption for unauthorized user eve@example.com:
Member eve@example.com not found in the group.
Member bob@example.com removed from the group.
Current group members: dict_keys(['alice@example.com'])

3s completed at 6:19 PM
  
```



```

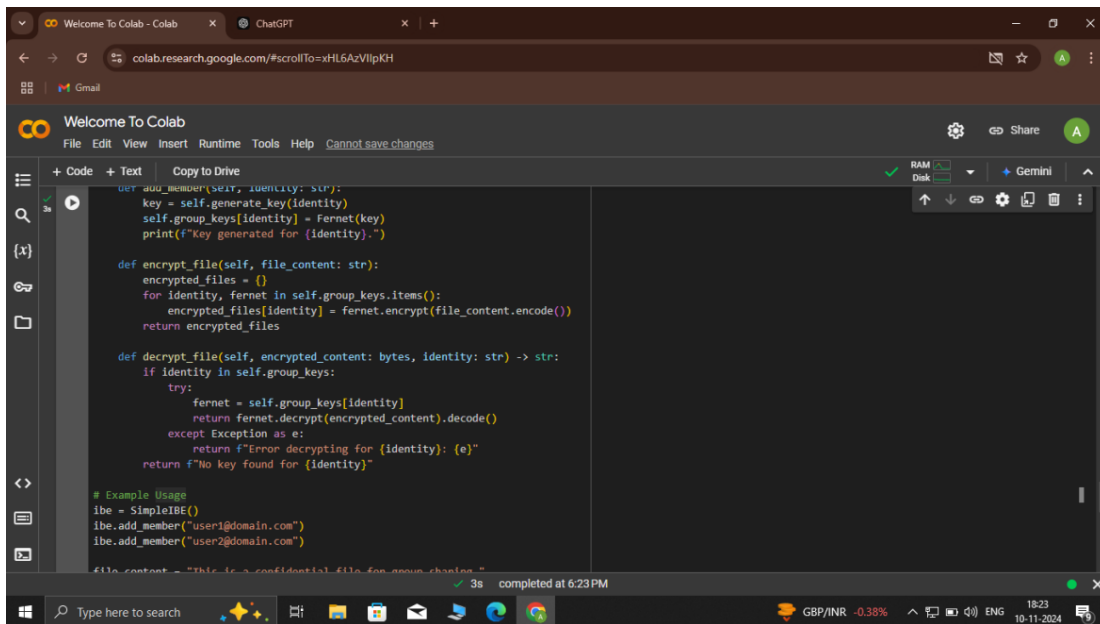
# Install the cryptography package
!pip install cryptography

# Import required libraries
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.fernet import Fernet
import base64

class SimpleIBE:
    def __init__(self):
        self.group_keys = {}

    def generate_key(self, identity: str) -> bytes:
        salt = b'\x12\x34\x56\x78\x9a\xbc\xde\xfg' # Use a secure, random salt in a real application
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,
            salt=salt,
            iterations=100000,
        )
        key = base64.urlsafe_b64encode(kdf.derive(identity.encode()))
        return key
    
```

3s completed at 6:23 PM



```

def add_member(self, identity: str):
    key = self.generate_key(identity)
    self.group_keys[identity] = Fernet(key)
    print(f"Key generated for {identity}.")

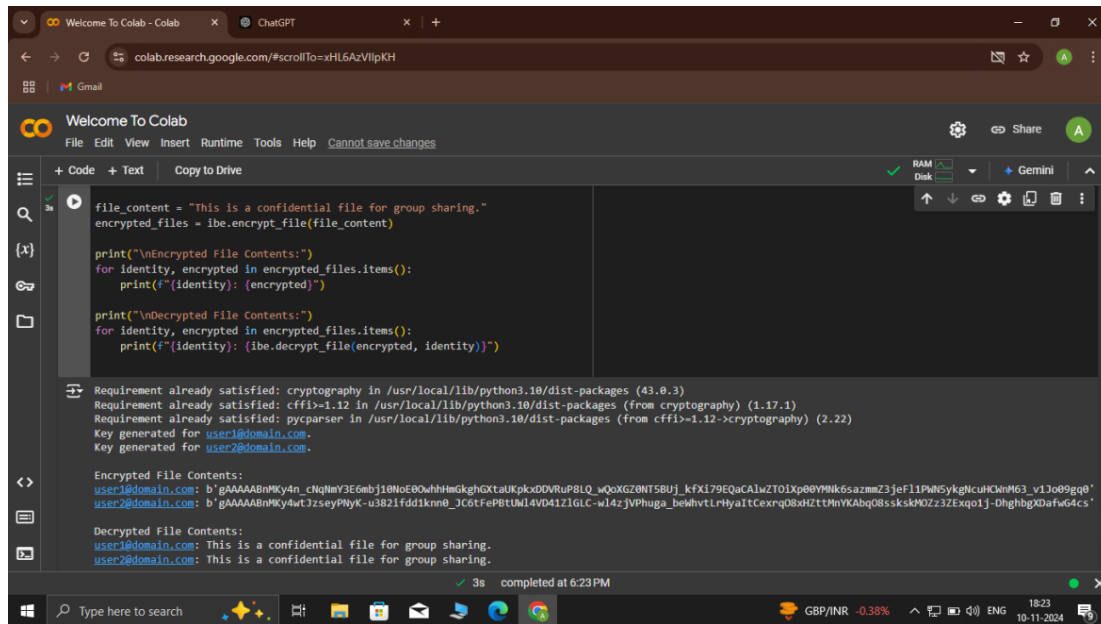
def encrypt_file(self, file_content: str):
    encrypted_files = {}
    for identity, fernet in self.group_keys.items():
        encrypted_files[identity] = fernet.encrypt(file_content.encode())
    return encrypted_files

def decrypt_file(self, encrypted_content: bytes, identity: str) -> str:
    if identity in self.group_keys:
        try:
            fernet = self.group_keys[identity]
            return fernet.decrypt(encrypted_content).decode()
        except Exception as e:
            return f"Error decrypting for {identity}: {e}"
    return f"No key found for {identity}"

# Example Usage
ibe = SimpleIBE()
ibe.add_member("user1@domain.com")
ibe.add_member("user2@domain.com")

file_content = "This is a confidential file for secure sharing"
    
```

3s completed at 6:23 PM



```
file_content = "This is a confidential file for group sharing."
encrypted_files = ibe.encrypt_file(file_content)

print("\nEncrypted File Contents:")
for identity, encrypted in encrypted_files.items():
    print(f"{identity}: {encrypted}")

print("\nDecrypted File Contents:")
for identity, encrypted in encrypted_files.items():
    print(f"{identity}: {ibe.decrypt_file(encrypted, identity)}")
```

Requirement already satisfied: cryptography in /usr/local/lib/python3.10/dist-packages (43.0.3)  
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography) (1.17.1)  
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography) (2.22)  
Key generated for user1@domain.com.  
Key generated for user2@domain.com.

Encrypted File Contents:  
user1@domain.com: b'gAAAAABnMKy4n\_cNqNmY3E6mbj10NoE00whhmGkghGxtaUKpkxDDVRuP8LQ\_uQoXGZ0NT5BUj\_kfXI79EqACAlwZT0IXp00YMNk6sazmZ3jefl1PWN5ykgNcuHCWnM63\_v13o09gq0'  
user2@domain.com: b'gAAAAABnMKy4wtJ3zeyPMYK-u3821fdd1knn0\_3C6tFeP8tUml4VD41Z1GLC-w14zjVPPhuga\_bewhvttrfyaItCexrq08xdZttMnYKAbq08sskskMOZz3ZExq01j-DhghbgXDafwG4cs'

Decrypted File Contents:  
user1@domain.com: This is a confidential file for group sharing.  
user2@domain.com: This is a confidential file for group sharing.

3s completed at 6:23 PM

## Conclusion

This research contributes to the field of secure cloud computing by demonstrating the practical application of IBE for group data sharing. Future work could explore hybrid encryption techniques or distributed key management systems to further enhance security and scalability. The project lays a foundation for developing secure and efficient cloud-based data sharing solutions.

### **Github link**

**<https://github.com/Rajasekar06/Group-Data-Sharing-In-Cloud-Computing-On-Identity-Based-Encryption>**.

### **REFERENCES**

- Boneh, D., & Franklin, M. (2003). Identity-Based Encryption from the Weil Pairing. SIAM Journal on Computing.
- Shamir, A. (1985). Identity-Based Cryptosystems and Signature Schemes. Advances in Cryptology.
- Cloud Security Alliance. (2020). Best Practices for Securing Cloud Data.
- Research papers and articles on encryption techniques and cloud security frameworks.