A

Project Report

on

Secure Data Sharing With Efficient Key Update for Industrial Cloud-Based Multi Attribute Access Control

A report submitted in partial fulfillment of the requirements for the award of the B.Tech degree

By

Gaddam Poojasri(20EG105313)

Vunnam Nandini Krishna(20EG105326)

Achugatla Sindhura(20EG105732)

Under the guidance of

MRS. ASMA TAHSEEN

Assistant Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANURAG UNIVERSITY
VENKATAPUR– 500088
TELANGANA
YEAR 2023-2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERITIFICATE

This is to certify that the Report/dissertation entitled "**Secure Data Sharing With Efficient Key Update for Industrial Cloud-Based Multi Attribute Access Control**" that is being submitted by G Poojasri(20EG105313), V Nandini Krishna(20EG105326), A Sindhura(20EG105732) in partial fulfillment for the award of B.Tech in Computer Science and Engineering to the Anurag University is a record of bonafide work carried out by us under my guidance and supervision. The results embodied in this Report have not been submitted to any other University or Institute for the award of any degree or diploma

Signature of Supervisor

MRS. ASMA TAHSEEN
Assistant professor

Signature of Dean

Dr. G Vishnu Murthy
M.Tech., Ph.D
Professor, CSE

External Examiner

DECLARATION

I hereby declare that the Report entitled"**Secure Data Sharing With Efficient Key Update for Industrial Cloud-Based Multi Attribute Access Control"** submitted for the award of Bachelor of technology Degree is my original work and the Report has not formed the basis for the award of any degree, diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

Place: Anurag University, Hyderabad

Gaddam Poojasri(20EG105313)
Vunnam Nandini Krishna(20EG105326)
Achugatla Sindhura(20EG105732)

Date:

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **MRS. ASMA TAHSEEN** for her constant encouragement and inspiring guidance without which this project could not have been completed. Her critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. Her patience, guidance and encouragement made this project possible.

We would like to express my special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for their encouragement and timely support in our B.Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy** , Dean, Dept. of CSE ,Anurag University. We also express my deep sense of gratitude to **Dr. V V S S S Balaram** , Academic coordinator. **Dr. T SHYAM PRASAD** Project Coordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated me during the crucial stage of our project work.

Gaddam Poojasri
(20EG105313)

Vunnam Nandini Krishna
(20EG105326)

Achugatla Sindhura
(20EG105732)

ABSTRACT

The secure and flexible management of shared data in industrial cloud environments necessitates enabling users to access authorized data portions from smart edge devices using integrated secure protocols for interaction with cloud storage. Ciphertext-policy attribute-based encryption (CP-ABE) is acknowledged as a viable cryptographic solution for achieving fine-grained and secure data access control. Nevertheless, the drawback of CP-ABE lies in attribute revocation, incurring subsequent costs like ciphertext re-encryption, user key regeneration, and key redistribution. Existing revocable CP-ABE models have concentrated on minimizing communication and computation costs for ciphertext re-encryption, neglecting the key update issue inherent in traditional CP-ABE. This key update becomes critical, especially in scenarios with a high number of users accessing shared cloud data. This paper introduces a lightweight access control model featuring an efficient key updating scheme. A performance evaluation is conducted to illustrate the effectiveness of the proposed scheme.

Keywords: CP-ABE, key update, revocation, access control

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF GRAPHS

1.INTRODUCTION

Cloud computing has revolutionized data storage and sharing practices, providing flexibility and accessibility. However, it also poses significant security problems such as data leaks, bandwidth issues, and data tampering risks, which can compromise the integrity and confidentiality of business data. To address these issues, an effective access control system is necessary, which should provide both convenience and functionality in data access and sharing processes while ensuring data security. This requires the development of new solutions that can combine encryption, key distribution, and process management, and implement effective access control policies based on multiple attributes while reducing the overhead associated with key updates.

To address these needs, this study proposes solutions for the security and quality management of information in environmental conditions. The proposed system offers a new access control system with a lightweight and scalable design and significant updates. The system aims to improve data security, integrity, and availability by leveraging Password Policy Attribute-Based Encryption (CP-ABE) capabilities and integrated authentication. Through careful analysis and practical application, this study focuses on the effectiveness and efficiency of problem-solving strategies in addressing complex problems related to job copy management data in the cloud environment. The emergence of cloud computing in today's business world has revolutionized data storage and sharing practices by providing unprecedented flexibility and accessibility.

However, this simplicity brings with it numerous security problems, especially the understanding of business data. Data leaks, bandwidth issues, and data tampering risks have become significant and pose a serious threat to the integrity and confidentiality of storing and sharing business data in the cloud.

To solve these problems, it is necessary to have an effective access control system that will provide convenience and functionality in data access and sharing processes as well as data security.

This requires the development of new solutions that can combine encryption, key distribution, and process management, reducing the overhead associated with key updates and implementing effective access control policies based on multiple attributes. In response to these needs, this study aims to propose solutions for the security and quality management of information in environmental conditions. The proposed system offers a new access control system with a lightweight and scalable design and significant updates. The system is designed to improve data security, integrity, and availability by leveraging Password Policy Attribute-Based Encryption (CP-ABE) capabilities and integrated authentication. Through careful analysis and practical application, this study focuses on the effectiveness and efficiency of problem-solving strategies in solving complex problems from job copy management data in the cloud environment.

2.LITERATURE SURVEY

2.1 Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption

Authors, Year : M. Li, S. Yu, Y. Zheng, K. Ren and W. Lou, Jan. 2013.

Their work addressed the critical need for secure management of personal health records (PHRs) in cloud environments. By leveraging attribute-based encryption (ABE), their model provided a scalable and secure solution for sharing PHRs in the cloud. This approach allowed for fine-grained access control over sensitive healthcare data, ensuring that only authorized users with the necessary attributes could access specific records. The contributions of Li, Yu, Zheng, and Ren have had a significant impact on healthcare data management, enabling efficient sharing and collaboration while maintaining the privacy and security of sensitive information.

2.2 Ciphertext-policy attribute-based encryption

Authors, Year : J. Bethencourt, A. Sahai, and B. Waters , 2007.

Their work introduced the concept of attribute-based encryption (ABE) and its variant, ciphertext-policy attribute-based encryption (CP-ABE). In CP-ABE, ciphertexts are associated with sets of attributes, and users' secret keys are constructed based on access structures defined by the data owner. This innovative approach decouples data encryption from access policies, enabling fine-grained access control in cloud computing and other distributed systems.

The contributions of Bethencourt, Sahai, and Waters have had a profound impact on data security and privacy, paving the way for more flexible and scalable access control mechanisms in modern computing environments.

2.3 HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing

Authors, Year :Z. Wan, J. Liu, and R. H. Deng, 2012.

Their work addressed the challenges of access control in cloud computing environments by proposing a hierarchical attribute-based encryption scheme. HASBE organizes attributes in a hierarchical structure, allowing for efficient management of access policies and delegation of access rights. This hierarchical approach enhances scalability and flexibility in access control, making it suitable for complex organizational structures and diverse access requirements in cloud environments. The contributions of Wan, Liu, and Deng have significantly advanced access control mechanisms in cloud computing, providing a practical solution for managing access to sensitive data in distributed systems.

2.4 DAC-MACS: Effective data access control for multi authority cloud storage systems

Authors, Year :K. Yang, X. Jia, and K. Ren, 2013.

Their research addressed the challenges associated with access control in multi-authority cloud storage systems. DAC-MACS utilized techniques from ciphertext-policy attribute-based encryption (CP-ABE) to improve the efficiency of the decryption process and resolve the revocation problem. Their model introduced novel decryption token and key update algorithms, enhancing the security and usability of multi-authority cloud storage systems.

The contributions of Yang, Jia, and Ren have advanced access control mechanisms in cloud storage environments, providing effective solutions for managing access to sensitive data across multiple authorities.

2.5 Proxy cryptosystems: Delegation of the power to decrypt ciphertexts

Authors, Year :M. Mambo and E. Okamoto, 1997.

Their research introduced the concept of proxy cryptosystems, which enable the delegation of decryption rights from the original data owner to a proxy entity. This delegation allows the proxy to decrypt ciphertexts on behalf of the data owner without compromising the security of the encrypted data. Mambo and Okamoto's work laid the foundation for the development of proxy re-encryption schemes, which have applications in scenarios where users need to delegate decryption capabilities while maintaining control over their encrypted data. Their contributions have had a lasting impact on the field of cryptography, providing new avenues for secure data sharing and delegation in distributed systems.

Table 1 : Comparison between methods

Author	**Method**	**Advantages**	**Disadvantages**
●M. Li ●S. Yu ● Y. Zheng ●K. Ren ●W. Lou	Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption	-Scalable and secure sharing of sensitive data in the cloud -Fine-grained access control over personal health records -Reduced risk of unauthorized access through attribute-based access policies	- Potential overhead in managing attributes and access policies -Complexity in defining and enforcing access policies for diverse user groups - Increased computational overhead for policy evaluation
●J. Bethencourt ●A. Sahai ● B. Waters	Ciphertext-policy attribute-based encryption	- Fine-grained access control - Decouples data encryption from access policies. - Flexibility in defining complex access structures	- Complexity of managing attributes and policies - Potential scalability issues as the number of attributes and policies grow - Increased computational overhead for policy evaluation

●Z. Wan ●J. Liu ● R. H. Deng	HASBE: Hierarchical attribute-based encryption scheme for access control in cloud computing	-Hierarchical organization of attributes for efficient access control - Scalability and flexibility in managing access policies - Reduced overhead in access policy evaluation	-Implementation complexity -Potential overhead in managing hierarchical structures -Increased computational overhead for policy evaluation -
●K. Yang ●X. Jia ●K. Ren	DAC-MACS: Data Access Control for Multi-Authority Cloud Storage	- Enhanced efficiency of decryption process - Resolves revocation issues associated with ABE - Flexibility in managing access control policies and decryption tokens	- Complexity in implementing and managing decryption tokens and key update algorithms - Potential overhead in key management and distribution -Increased computational overhead for policy evaluation

● M. Mambo ● E. Okamoto	Proxy cryptosystems for delegation of decryption rights	- Enables delegation of decryption rights to proxy entities - Maintains security of encrypted data during delegation process - Flexibility in managing access control through proxy entities	- Complexity in implementing and managing proxy re-encryption schemes - Potential overhead in managing proxy entities and associated decryption keys - Increased risk of security breaches through compromised proxy entities

3. SYSTEM SPECIFICATIONS

3.1 System Requirements

- **Hardware Requirements:**

System : Pentium IV 2.4 GHz

Hard Disk : 200 GB.

Monitor : 15 VGA Color

Input Devices : Keyboard, Mouse

Ram : 1 GB

- **Software Requirements:**

Operating system : Windows 10.

Coding Language : Python

Tool : anaconda

Database: MYSQL

4. SYSTEM METHODOLOGY

4.1 System Analysis

4.1.1 Existing System

The existing method introduced a privacy-aware access control model, PRSX-AC, specifically tailored to safeguard electronic health records (EHRs) within a hybrid cloud environment. This model, an extension of the standard XACML-ABAC, integrates a semantic relationship-based access control (rel-BAC) approach. In rel-BAC, access rights are defined through relationships between users and objects.

Under Joshi's scheme, all processes—request handling, decision evaluation, document retrieval, as well as encryption and decryption—take place within the organization's trusted domain. This strategy capitalizes on the intricate ontology schema's flexibility, allowing for the expression and enforcement of complex policies. These policies leverage contextual and environmental attributes, enabling comprehensive protection of sensitive health records.

Disadvantages

- ➢ This application was developed in health care environment were data access is limited to few users
- ➢ When coming to multiple user access control policy in industrial environment this method cant give access permission to all users effectively.

Traditional access control models like Mandatory Access Control (MAC), Discretionary Access Control (DAC), and Role-Based Access Control (RBAC) are inadequate for data outsourcing scenarios, where resources are managed by cloud providers rather than housed within organizations. Despite the generally trustworthy nature of cloud providers, there's a risk of compromising data privacy, emphasizing the need for encryption to maintain confidentiality. However, encryption techniques pose challenges, such as the generation of multiple copies of encrypted data in public key encryption and maintenance costs in symmetric encryption.

These challenges, coupled with the incompatibility of traditional models with outsourcing environments, underscore the necessity for a new approach to access control that aligns with the dynamics of data outsourcing.

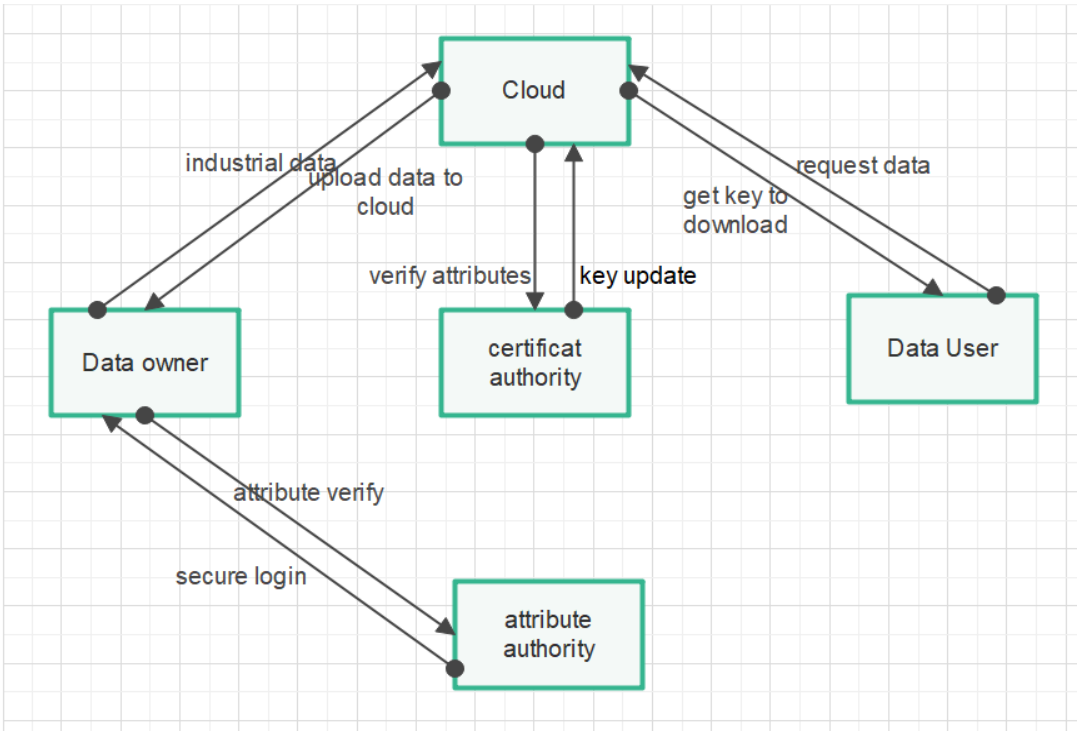
CP-ABE relies on computationally expensive operations like pairing operations and exponentiation, especially challenging for resource-constrained devices. Additionally, CP-ABE naturally supports read-only data sharing without specific privileges for different users, which may not be suitable for certain industries requiring users to update and encrypt data. Attribute or user revocation in CP-ABE incurs significant costs, including policy updates, ciphertext re-encryption, and key distribution, which can impede practical implementation attack.

4.1.2 Proposed System

We introduce a lightweight and scalable access control model integrated with an efficient key update mechanism. Our proposed system aims to enhance data access and security within industrial settings by employing encryption, key distribution, and access control measures. Industrial data serves as the input, undergoing secure AES encryption, and an extension involving multi-attribute authorization is applied to bolster access control policies. In this setup, users can download data only when both access policies set by the owner are satisfied. Following the download, a central authority promptly updates the key to prevent unauthorized sharing of the file among other users. This approach ensures robust protection and controlled access to sensitive industrial data throughout its lifecycle.

We have developed a web application to demonstrate the efficacy of our key update scheme in mitigating risks associated with access control in cloud environments. This application serves as a practical showcase of how our system enhances security by efficiently managing key updates, thereby reducing the likelihood of unauthorized access to cloud-stored data.

Through this demonstration, we aim to illustrate the real-world applicability and benefits of our approach in strengthening access control mechanisms within cloud environments.



The image illustrates the process of uploading data to a cloud-based system. Here's a detailed breakdown of each step:

1. **Data Upload**: Initially, the data owner uploads their data to the cloud platform. This can encompass various types of data, ranging from documents to multimedia files.

2. Secure Login: Following data upload, the data owner securely logs in to the cloud system using authentication methods such as usernames and passwords.

3. Attribute Verification: Once logged in, the system verifies the attributes associated with the uploaded data.

4. **Cloud Storage**: If the attributes meet the criteria, the data is stored securely in the cloud storage infrastructure.

5. Key Issuance: Subsequently, the data owner receives a unique key that enables them to request access to their stored data in the future.

6. **Data Access Request**: At a later point, a data user, distinct from the data owner, may request access to the stored data.

7. Attribute Verification: The request from the data user is routed to an attribute authority, which verifies the user's attributes to ascertain whether they possess the necessary permissions to access the data.

8. Key Update (if required): Upon successful attribute verification, the data user may be granted a key to access the data. In certain scenarios, the attribute authority might update the data owner's key to reflect any new access permissions granted.

9. Data Download: Equipped with the appropriate key, the data user can then proceed to download the desired data from the cloud storage system.

This cloud-based system offers a secure mechanism for data owners to upload and manage their data while controlling access permissions. Data users can request access, and the system ensures access is granted only to those with the requisite permissions, thereby enhancing data security and privacy.

4.1.3 Advantages of Proposed System :

Lightweight and Scalable Access Control Model: The system proposes an access control model that is both lightweight and scalable. This means that it should be efficient in terms of computational resources and capable of handling a large number of users and data without significant performance degradation.

Efficient Key Update Scheme: The system features an efficient key update scheme. This scheme ensures that keys used for encryption are regularly updated to enhance security. Efficient key updates prevent unauthorized access to data by ensuring that only authorized users have access to the latest encryption keys.

Improving Data Access and Security: The primary goal of the proposed system is to improve both data access and security. This is achieved through a combination of encryption, key distribution, and access control mechanisms.

AES Encryption: Advanced Encryption Standard (AES) is used for encrypting the industrial data. AES is a widely adopted encryption algorithm known for its security and efficiency. By encrypting the data, the system ensures that even if unauthorized users gain access to the data, they cannot read it without the proper decryption key

Multi-Attribute Extension for Access Control: The system enhances access control by incorporating a multi-attribute extension. This means that access control policies are based on multiple attributes, such as user roles, time of access, and location. By considering multiple attributes, the system can enforce more fine-grained access control policies.

Two-Step Access Policy Verification: The access control mechanism involves a two-step verification process. Users can only download data if they satisfy two access policies set by the owner. This adds an extra layer of security by ensuring that only users who meet specific criteria can access the data.

Key Update after File Download: After a user successfully downloads a file, the central authority updates the encryption key associated with that file. This prevents the user from sharing the file with unauthorized users since the previously used key is no longer valid. It also ensures that even if the file is shared, only users with the updated key can decrypt and access its contents.

4.2 System Design

4.2.1 Introduction

The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces.

4.2.2 Modules

Owner Module:

The Owner Module facilitates the management of files within the system. Owners can register, login using email and password credentials, upload files with specified attributes such as "scientist" or "researcher", view uploaded files, encrypt files for security, and logout when their session is complete.

User Module:

The User Module caters to individuals utilizing the system to access files. Users can register with attributes like "scientist" or "researcher", login using their email and password, view available files, request access keys for encrypted files, decrypt files for viewing or download, and logout to secure their session.

AA Module:

The AA Module, standing for Attribute Authentication, serves the purpose of verifying user attributes and ownership status. It allows users to log in, verifies their assigned roles, checks for owner privileges, and facilitates secure logout procedures.

Cloud Module:

The Cloud Module functions as the storage and access hub for files within the system. It enables users to log in, view encrypted files stored in the cloud, upload new files, handle requests for file access, send encryption keys to authorized users, and securely logout when tasks are completed.

4.3 Parameters & Formulas

Secure key sharing:

To initiate the process, a user through the on-premise key management system denoted OKM S generates the master key k. Executed by an application on behalf of the key management system on premise, it takes k. The protocol outputs the finite field Fp, the public shares [s]I of kalong with its MAC (γ) and the public shares of the MAC γ0ifor i= 1,· · · , n..

Sharing:

This protocol is executed with Set Up and Reconstruction protocols. It takes shares of the master key [si] key and MAC γ along with the value of the MAC γ0i. The protocol delivers/retrieves the shares [s]i, γj, for i, j ∈ {1,· · · , n}to/from the servers. The value of the MAC γis stored by the trusted appliance

Key Generation:

The key generation algorithm will take as input a set of attributes S and output a key that identifies with that set. The algorithm first chooses a random r ∈ Zp, and then random rj ∈ Zp for each attribute j ∈ S. Then it computes the key as

SK = D = g (α+r)/β , ∀j ∈ S : Dj = g r · H(j) rj ,D′ j = g rj

Attribute based access control:

(Access Structure [4]) Let {P1, P2, . . . , Pn} be a set of parties. A collection A ⊆ 2 {P1,P2,...,Pn} is monotone if ∀B, C : if B ∈ A and B ⊆ C then C ∈ A. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) A of non-empty subsets of {P1, P2, . . . , Pn}, i.e., A ⊆ 2 {P1,P2,...,Pn}\{∅}. The sets in A are called the authorized sets, and the sets not in A are called the unauthorized sets.

Encryption:

This is a randomized algorithm that takes as input a message m, a set of attributes γ, and the public parameters PK. It outputs the ciphertext E

Decryption with Key:

The advantage of an adversary A in this access control is defined as Pr[b ′ = b] − 1 2 . We note that the model can easily be extended to handle chosen-ciphertext attacks by allowing for decryption queries

5. Software Environment

5.1 The Python Programming Language

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It has gained immense popularity in various fields due to its extensive libraries, easy syntax, and broad community support. Here are some key aspects of Python programming language and its application in technology:

- **Simplicity and Readability**: Python's syntax is designed to be easy to read and understand, making it suitable for both beginners and experienced programmers. Its clean and concise code structure enhances productivity and reduces development time.
- **Extensive Libraries:** Python boasts a rich collection of libraries and frameworks for various purposes, including data science (NumPy, Pandas, Matplotlib), machine learning (scikit-learn, TensorFlow, PyTorch), web development (Django, Flask), and more. These libraries provide pre-built functions and tools that simplify complex tasks and accelerate development.
- **Versatility:** Python is a multipurpose language that supports a wide range of applications, from web development and desktop GUIs to scientific computing and automation. Its versatility makes it a preferred choice for developers working on diverse projects across different domains.
- **Community Support:** Python has a vibrant and active community of developers, contributors, and enthusiasts. This community-driven ecosystem fosters collaboration, knowledge sharing, and continuous improvement of the language and its associated tools and libraries.
- **Scalability:** While Python is often criticized for its performance compared to lower-level languages like C++ or Java, it offers scalability through various means. Techniques such as code optimization, using compiled extensions (e.g.,

Cython), and asynchronous programming (with libraries like asyncio) help improve performance and scalability for large-scale applications.

- **Integration with Other Technologies:** Python seamlessly integrates with other technologies and platforms, allowing developers to leverage existing systems and services. It can interact with databases, web servers, cloud services, and APIs, facilitating interoperability and system integration.
- **Rapid Prototyping and Development:** Python's simplicity and rich ecosystem enable rapid prototyping and development of software applications. Developers can quickly build prototypes, iterate on ideas, and test concepts, which is particularly advantageous in agile development environments.
- **Support for Emerging Technologies:** Python remains at the forefront of emerging technologies such as artificial intelligence, machine learning, and data science. Its libraries and frameworks provide powerful tools for developing and deploying advanced solutions in these rapidly evolving fields.

5.2 Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine Learning applications, Large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. It is developed and maintained by Anaconda, Inc. The distribution includes data-science packages suitable for Windows, Linux, and macOS. Packaged versions are required and are managed by the package management system anaconda. This package manager was spun out as a separate open-source package as it ended up being useful on its own and for other things than Python. There is also a small, bootstrap version of Anaconda called Miniconda, which includes only conda, Python, the packages they depend on, and a small number of other packages.

.

6.IMPLEMENTATION

6.1 Packages

import os : The os module in Python serves as a versatile tool for interacting with the underlying operating system, offering functions to manage files, directories, and environment variables. It simplifies tasks related to file manipulation and system navigation, making it indispensable for various applications involving operating system interaction.

import datetime: The datetime module facilitates efficient manipulation and handling of dates and times within Python programs. It provides classes and functions to work with timestamps, durations, and calendar dates, enabling developers to perform operations such as date arithmetic, formatting, and parsing with ease..

import hashlib: hashlib is a crucial module for generating secure hash functions and message digests in Python. It supports a variety of cryptographic hash algorithms like MD5 and SHA-256, making it invaluable for tasks requiring data integrity verification, password hashing, and digital signatures

import Crypto.Cipher: The Crypto.Cipher module, a part of the Crypto package, equips Python programmers with powerful encryption and decryption capabilities. It enables the implementation of symmetric and asymmetric encryption schemes, including popular algorithms like AES and RSA. This module is instrumental in safeguarding sensitive data, ensuring secure communication channels, and preserving data privacy.

Flask: The Crypto.Cipher module, a part of the Crypto package, equips Python programmers with powerful encryption and decryption capabilities. It enables the implementation of symmetric and asymmetric encryption schemes, including popular algorithms like AES and RSA. This module is instrumental in safeguarding sensitive data, ensuring secure communication channels, and preserving data privacy.

from AES import AESCipher

The AES (Advanced Encryption Standard) package, accessed via from AES import AESCipher, empowers developers to implement robust encryption mechanisms using the AES algorithm for data confidentiality. This package provides essential tools for securing sensitive information in Python applications, ensuring that data remains protected from unauthorized access.

from des import des

The DES (Data Encryption Standard) package, imported using from des import des, equips Python applications with functions for encrypting and decrypting data using the DES symmetric encryption algorithm. Developers can leverage this package to enhance the security of their applications by encoding sensitive data before storage or transmission.

import random

The random module, brought in with import random, plays a vital role in generating pseudorandom numbers and sequences essential for various applications requiring randomness. This module provides developers with a reliable source of randomness, facilitating tasks such as simulations, cryptography, and game development.

6.2 Coding

```
import os

import datetime

import hashlib

import Crypto.Cipher

from flask import Flask,
session, url_for, redirect,
render_template, request,
abort, flash

from database import
db_connect,user_reg,owner_re
g,owner_login,upload_file,ow
ner_viewfiles,upload_clouddat
a,user_request,owner_request,
user_lastdownload

from database import
onwer_viewdata,user_loginact,
user_viewfile,user_viewfiledat
a,user_down,verify_user,verif
y_user2,user_finaldown,owner
_update,user_down1

from werkzeug.utils import
secure_filename

import cv2

from AES import AESCipher
```

```
from des import des

import random

import base64

import cv2

from stegano import lsb

from RSA import
encrypt,decrypt,generate

from cloud import
uploadFile,downloadFile,close

from sendmail import
sendmail

app = Flask(__name__)

app.secret_key =
os.urandom(24)



#redirect to index page

    @app.route("/")
    def FUN_root():
        return render_template("index.html")


    # redirects to owner home page
    @app.route("/owner")
    def FUN_admin():
        return render_template("owner.html")


    #verify owner login details call owner_login function to verify data from database.
    #if verification is success redirect to ownerhome.html or if not owner.html
```

```
@app.route("/ownerlogact",methods = ['GET','POST'])
def owner_logact():
  if request.method == 'POST':
    status=owner_login(request.form['username'],request.form['password'])
    if status == True:
      session['username'] = request.form['username']
      return render_template("ownerhome.html",m1="sucess")
    else:
      return render_template("owner.html",m1="sucess")

# Redirect to user home page
@app.route("/user/")
def FUN_student():

   return render_template("user.html")

# redirect to user registration page
@app.route("/userreg/")
def FUN_userreg():
   return render_template("userreg.html")

# User registration details are passed to user_reg function to store data in user table.
@app.route("/userregact", methods = ['GET','POST'])
def user_regact():
  if request.method == 'POST':
    status =
user_reg(request.form['username'],request.form['password'],request.form['dob'],req
uest.form['email'],request.form['city'],request.form['contactno'])
    if status == True:
     return render_template("user.html",m1="Success")
else:
      return render_template("user.html",m1="Login failed")
```

```
#user login details are verified by passing details to user loginact function and
verify in database
@app.route("/userlogact",methods = ['GET','POST'])
def user_logact():
  if request.method == 'POST':
    status=user_loginact(request.form['email'],request.form['password'])
    if status == True:
      session['email'] = request.form['email']
      return render_template("userhome.html",m1="success")
    else:
      return render_template("user.html",m1="success")

#redirect to user home page
@app.route("/userhome")
def user_home():
    return render_template("userhome.html")

# get datafrom database to show to user
@app.route("/vf/")
def user_vf():
     viewfile = user_viewfile(session['email'])
     return render_template("vf.html", viewfiledata = viewfile)

# insert data in to request table when user requests data from owner
@app.route("/vf1/", methods = ['GET', 'POST'])
def user_vf1():
      fname = request.args.get('filename')
      owner = request.args.get('owner')
      data = request.args.get('data')
      print(fname,owner,data,session['email'])
      check = user_viewfiledata(fname,owner,data,session['email'])
      if check == True:
```

```
        return render_template("vf.html",m1="Request_Sucess")
       else:
        return render_template("vf.html",m1="Request_failed")


# get data from user_down table to show to user for file download files with request is validated
@app.route("/download/")
def user_download():
    downloaddata = user_down(session['email'])
    return render_template("download.html", downloads = downloaddata)
# user details are called for specific file download option is called
@app.route("/downloadact/" , methods = ['GET', 'POST'])
def user_downloadact():
    fname = request.args.get('fname')
    downloaddata = user_down1(session['email'],fname)
    return render_template("downloadact.html", downloadview = downloaddata)


# redirects to owner registration page
@app.route("/ownerreg/")
def FUN_ownerreg():
    return render_template("ownerreg.html")


# redirects to owner registration page
@app.route("/ownerregact", methods = ['GET','POST'])
def FUN_ownerregact():
   if request.method == 'POST':
    status = owner_reg(request.form['username'],request.form['password'],request.form['dob'],request.form['email'],request.form['city'],request.form['contactno'])
    if status == True:
     return render_template("ownerhome.html",m1="Login sucess")
    else:
     return render_template("owner.html",m1="Login failed")
```

```
# redirects to owner home page
@app.route("/ownerhome")
def FUN_ownerhome():
    return render_template("ownerhome.html")


# owner uploaded details are sotred in database by calling upload _ file function
and store in clouddata table
@app.route("/Upload", methods = ['GET','POST'])
def owner_upload():
   if request.method == 'POST':
      file = request.files['inputfile']
      check = upload_file(request.form['fname'],file,session['username'],"No","No")
      if check == True:
         return render_template("fileupload.html",m1="success")
      else:
         return render_template("fileupload.html",m1="Failed")


# redirects to file upload page
@app.route("/fileupload")
def FUN_fileupload():
         return render_template("fileupload.html")


# owner uploaded files are retirved from db using owner_viefiles function and
displayed to owner
@app.route("/ownerviewfiles")
def FUN_ownerviewfiles():
    viewdata = owner_viewfiles(session['username'])
    notes_table = zip([x[0] for x in viewdata],\
                      [x[1] for x in viewdata],\
                      [x[2] for x in viewdata],\
                      [x[3] for x in viewdata],\
                      [x[4] for x in viewdata])
```

```
    return render_template("ownerviewfiles.html",showdata = notes_table)


# click on split option by owner to divide data to three equal parts by calculating size of file

@app.route("/split/", methods = ['GET' , 'POST'])

def owner_split():

    fname = request.args.get('fname')

    owner = request.args.get('owner')

    data = request.args.get('data')

    # Length of file is calculated and size of each part is calculated

    size = len(data)

    l=len(data)

    leng=len(data)//3

    length=len(data)//2

    s1 = leng+leng

    k = s1+leng

    le = 1

    halfString=data[0:leng]

    second = data[leng:s1]

    third = data[s1:l]


   #   DES PART input of first part is passed to des algorihtm by generation random key encoded securiy key is generated and both

   #parameters are called by object d and encrypt function is called which returns encrypted data. data is stored in des.txt and uploaded to cloud

    DESkey_16 = os.urandom(8)

    d = des()

    encodedkey1 = base64.b64encode(DESkey_16)

    strkey = str(encodedkey1, 'utf-8')

    desencrypted = d.encrypt(DESkey_16,halfString,padding=True)

    print("Ciphered: %s" % desencrypted)

    print("encodedDESkey_16: %s" % strkey)
```

```
fileDES= open("C:/Users/Desktop/input/DES.txt", "w")

fileDES.write(str(desencrypted.encode("utf-8")))

fileDES.close()

uploadFile('DES.txt',"C:/Users/Desktop/input/DES.txt")
```

```
# AES PART input of second part is passed to AES algorihtm by generation random key encoded securiy key is generated and both

#parameters are called by object aescipher and encrypt function is called which returns encrypted data. data is stored in aes.txt and uploaded to cloud

AESkey_16 = os.urandom(16)

aescipher = AESCipher(AESkey_16)

encodedkey = base64.b64encode(AESkey_16)

strkey1 = str(encodedkey, 'utf-8')

aesencrypted = aescipher.encrypt(second)

print('aesEncrypted: %s' % aesencrypted)

print("encodedAESkey_16: %s" % strkey1)

print("AESkey_16: %s" % AESkey_16)


fileAES= open("C:/Users/Mrida/Desktop/input/AES.txt", "w")

fileAES.write(str(aesencrypted))

fileAES.close()

uploadFile('AES.txt',"C:/Users/Mrida/Desktop/input/AES.txt")
```

```
#RSA ORIGHINAL input of third part is passed to RSA algorihtm by generation random keypair public and private securiy key is generated and both

#parameters are called by object aescipher and encrypt function is called which returns encrypted data. data is stored in rsa.txt and uploaded to cloud

key_pair = generate(8)

print('Generated Key pairs')

public_key = key_pair["public"]

private_key = key_pair["private"]

print(key_pair["public"])
```

```
    print(key_pair["private"])
     #Now encrypt
    ciphertext = encrypt(public_key, third)
    txt = ', '.join(map(lambda x: str(x), ciphertext))
    print ("Ciphertext is: ")
    print (txt)
    print(ciphertext)
    fileRSA= open("C:/Users/Desktop/input/RSA.txt", "w")
    fileRSA.write(txt)
    fileRSA.close()
    uploadFile('RSA.txt',"C:/Users/Desktop/input/RSA.txt")
    val = ', '.join(map(lambda x: str(x), private_key))

    close()
    # Details are stored in cloud data table
    status =
upload_clouddata(data,fname.rstrip(),owner,desencrypted,strkey,aesencrypted,strke
y1,str(txt),str(val))
    return render_template("ownerviewfiles.html",m1="sucess")

#redirects to view split data
@app.route("/viewencfiles")
def owner_viewencfiles():
    splitdata = onwer_viewdata(session['username'])
    return render_template("viewencfiles.html",spliinfo = splitdata)

# redirects to view user request data
@app.route("/vuserreq")
def owner_vuserreq():
    userrequest = user_request(session['username'])
    return render_template("vuserreq.html",userreqdata = userrequest)

#redircts to index page and session closed
```

```
@app.route("/logout")
def FUN_logout():
    return render_template("index.html")


#owner responds to requests received from user
@app.route("/response", methods = ['GET','POST'])
def owner_response():
        fname1 = request.args.get('filename')
        data1 = request.args.get('data')
        owner1 = request.args.get('owner')
        email1 = request.args.get('email')
        result = owner_request(fname1,owner1,email1)
        rdata = result
        status = owner_update(rdata,fname1,owner1,email1)
        # keys are written in jonit key text file which is stored in given path using lsb
hide keys are hidden inside bt.jpg file
        #and new file bt_sec is created in same path
        if status == True:
          for skey,skey1,skey2,f1,f2,f3 in rdata:
              print(skey,skey1,skey2)
              sendmail(skey,skey1,skey2,email1)
              filen= open("C:/Users/Desktop/input/jointkey.txt", "w")
              filen.close()
              filen=open("C:/Users/Desktop/input/jointkey.txt",'a') #text file where all 3
keys are stored
              filen.write(skey)
              filen.write(skey1)
              filen.write(skey2)
              filen.close
              # kyes are written in to image by giving specific sting values between
each key
              stringkeys = 'fi'+skey+'se'+skey1+'co'+skey2+'th'
          with open("C:/Users/Desktop/input/jointkey.txt",'r')as f:
```

```
            #stringkeys = f.readlines()
            print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
            print(stringkeys)
            msg= lsb.hide("C:/Users/Desktop/input/bt.jpg",str(stringkeys)) #hide key in image
            msg.save("C:/Users/Mrida/Desktop/input/bt_sec.png") #save the embedded image

            return redirect(url_for('owner_vuserreq'))
        else:
            return render_template("vuserreq.html",m1="false")

# user verify key 1 and redirects to next page for second key
@app.route("/verify1", methods = ['GET','POST'])
def user_verfiy():
    filename =request.form['filename']
    dkey = request.form['dkey']
    vdat=verify_user(filename,dkey)
    if vdat:
      return render_template("download2act.html",vdatavsend = vdat)
    else:
      return render_template("download.html",m1="no data in table")

# redirects to thrid key page after verifying with second key
@app.route("/verify2", methods = ['GET','POST'])
def user_verfiy2():
    filename =request.form['filename']
    key = request.form['dkey']
    dat=verify_user2(filename,key)
    if dat:
      return render_template("download3act.html",datavsend = dat)
    else:
      return render_template("download.html",m1="no data in table")
```

```
# verify all three keys with keys hidden inside image by revial dat from bt_sec.png file using lsb function

# decryption of each part is peformed and final data is stored in myfile.txt

@app.route("/verify3", methods = ['GET','POST'])

def user_verfiy3():

    filename =request.form['filename']

    keys = request.form['dkey']

    dats=user_finaldown(filename,keys)

    if dats:

       for filename,owner in dats:

          fdata = user_lastdownload(filename,owner)

          if fdata:

           for skey,skey1,skey2,f1,f2,f3 in fdata:

            clear_msg = lsb.reveal("C:/Users/Desktop/input/bt_sec.png") # take out keys from image

            print(clear_msg) # show the keys

            left = 'fi'

            right = 'se'

            left1= 'se'

            right1 = 'co'

            left2 = 'co'

            right2 = 'th'

            # retive data from total message by finding each key which is stored between strings

            key1 = clear_msg[clear_msg.index(left)+len(left):clear_msg.index(right)]

            key2 = clear_msg[clear_msg.index(left1)+len(left1):clear_msg.index(right1)]

            key3 = clear_msg[clear_msg.index(left2)+len(left2):clear_msg.index(right2)]




          #   DES PART  first key from image with encypted data is given input to des decryption to
```

```
            # decrypt first part of data
             decodedkey = base64.b64decode(key1)
             descipherdec = des()
             desdecrypted = descipherdec.decrypt(decodedkey,f1,padding=True)
             print('desdecrypted: %s' % desdecrypted)

          #   AES PART second key from image with encypted data is given input to
aes decryption to
             # decrypt second part of data
             decodedkey1 = base64.b64decode(key2)
             aescipherdec = AESCipher(decodedkey1)
             aesdecrypted = aescipherdec.decrypt(f2)
             print('aesDecrypted: %s' % aesdecrypted)


          #   RSA PART third key from image with encypted data is given input to rsa
decryption to
             # decrypt third part of data
            rsaencrypteddata = list(f3.split(", "))
            print(rsaencrypteddata)
            rsapublickey = tuple(key3.split(", "))
            print(rsapublickey)
            #decrypt
            rsadeciphertext = decrypt(rsapublickey, rsaencrypteddata)
            print(rsadeciphertext)
            # final data combined to total data
            totaldata = desdecrypted+aesdecrypted+rsadeciphertext
            print('totaldata: %s' % totaldata)
          # final data stored in myfile.txt file and redirectd to download .html page
            file1 = open("C:/Users/Desktop/input/myfile.txt","w")
            file1.write(totaldata)
            return render_template("download.html",m2="downlaod sucessfull")
      else:
```

```
            return render_template("download.html",m2="downlaod failed")


# redicted to index page
@app.route("/logout")
def admin_logout():
    return render_template("index.html")
if __name__ == "__main__":
   app.run(debug=True,host='127.0.0.1', port=5000)
```

7. RESULTS

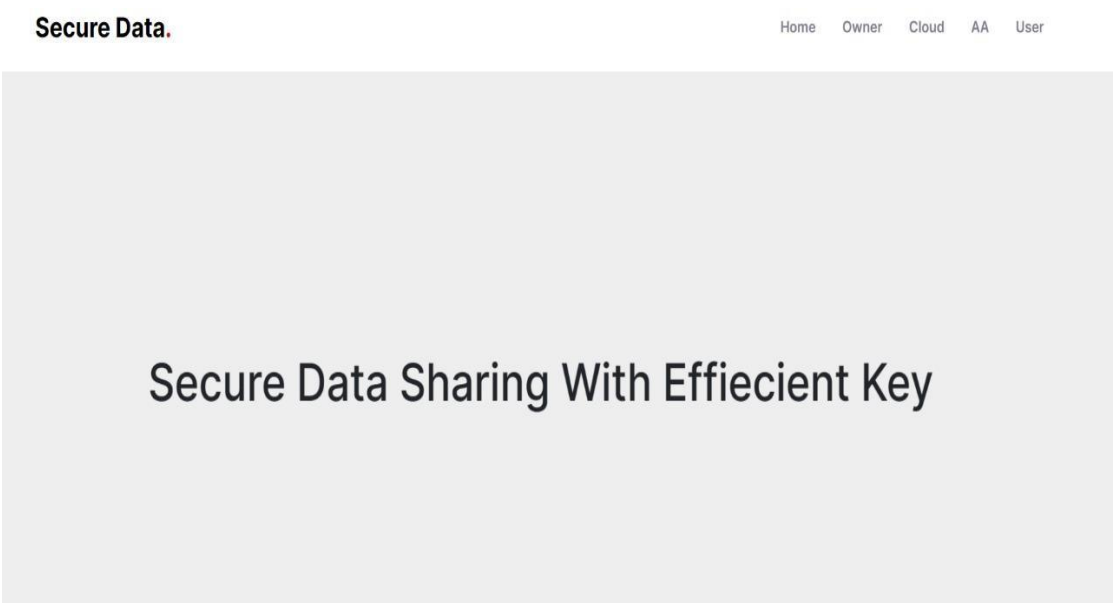


Fig 7.1 Home Page

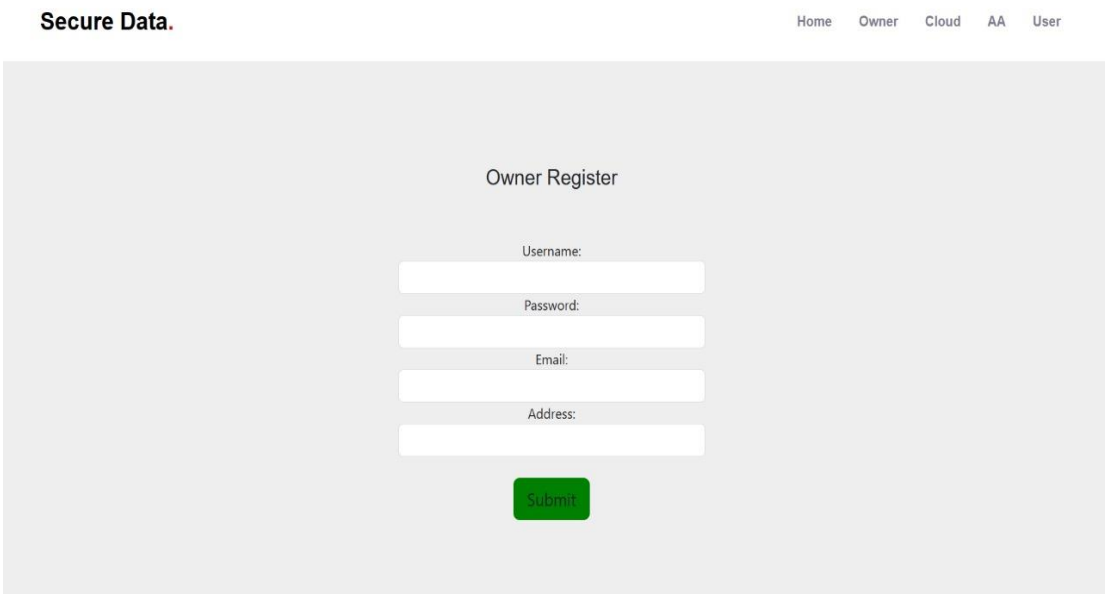


Fig 7.2 Owner Register

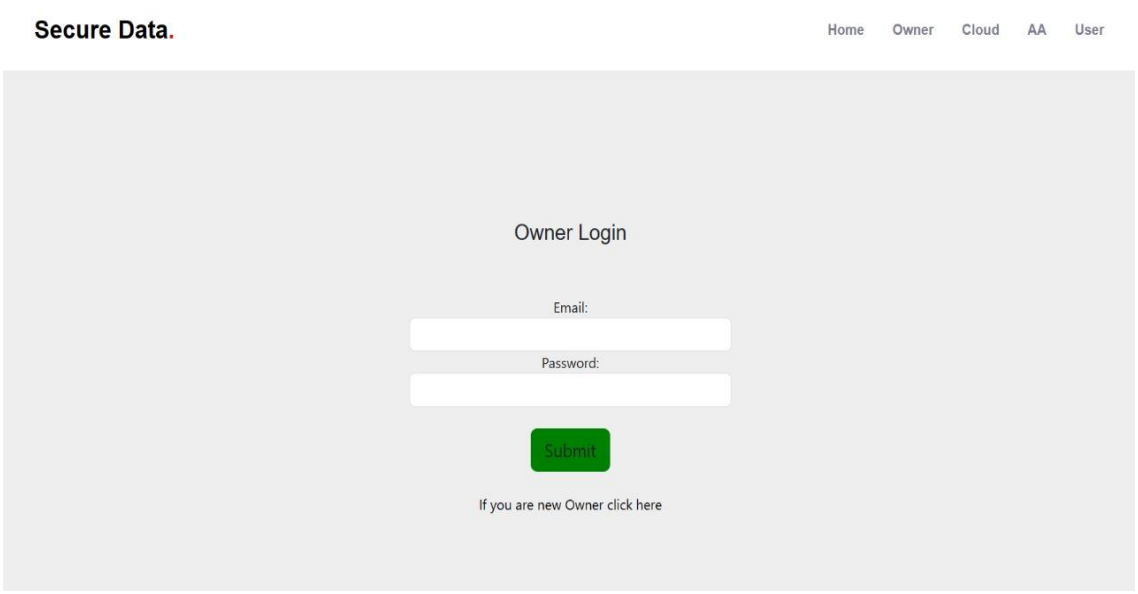


Fig 7.3 Owner Login

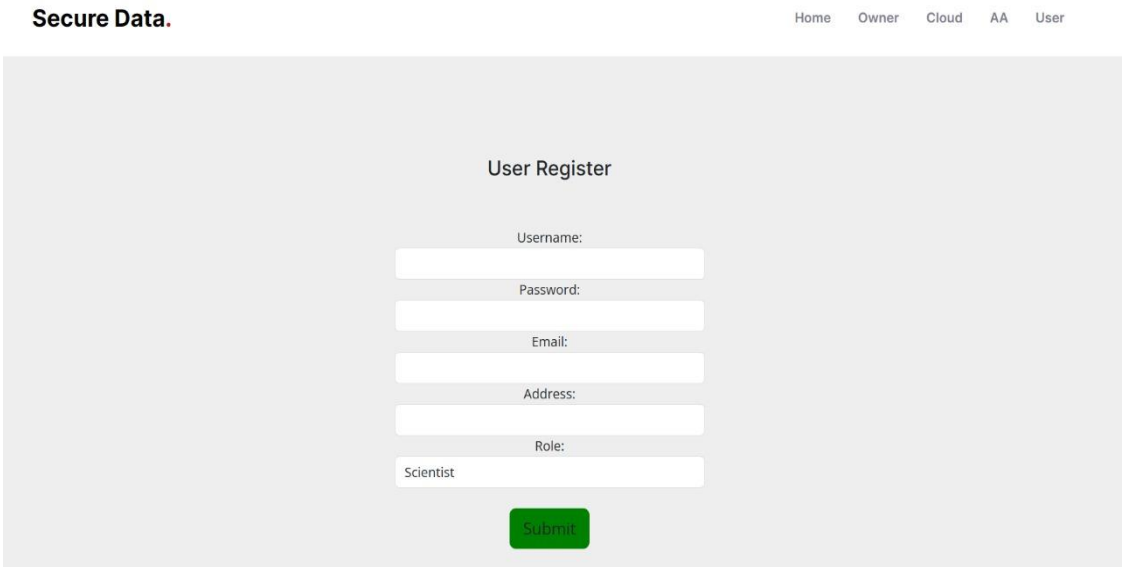


Fig 7.4 User Register

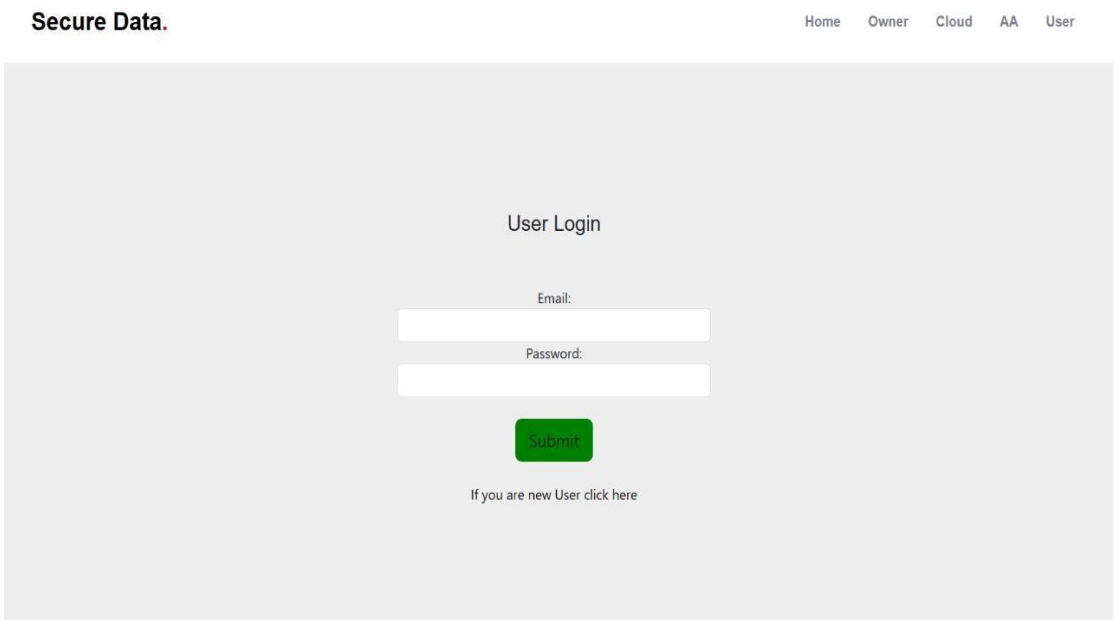


Fig 7.5 User Login

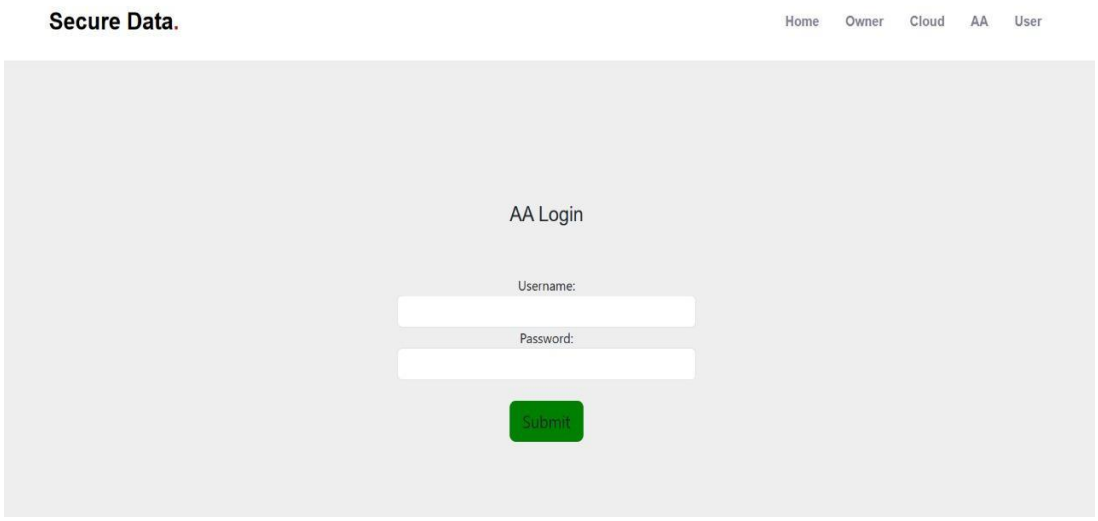


Fig 7.6 AA Login

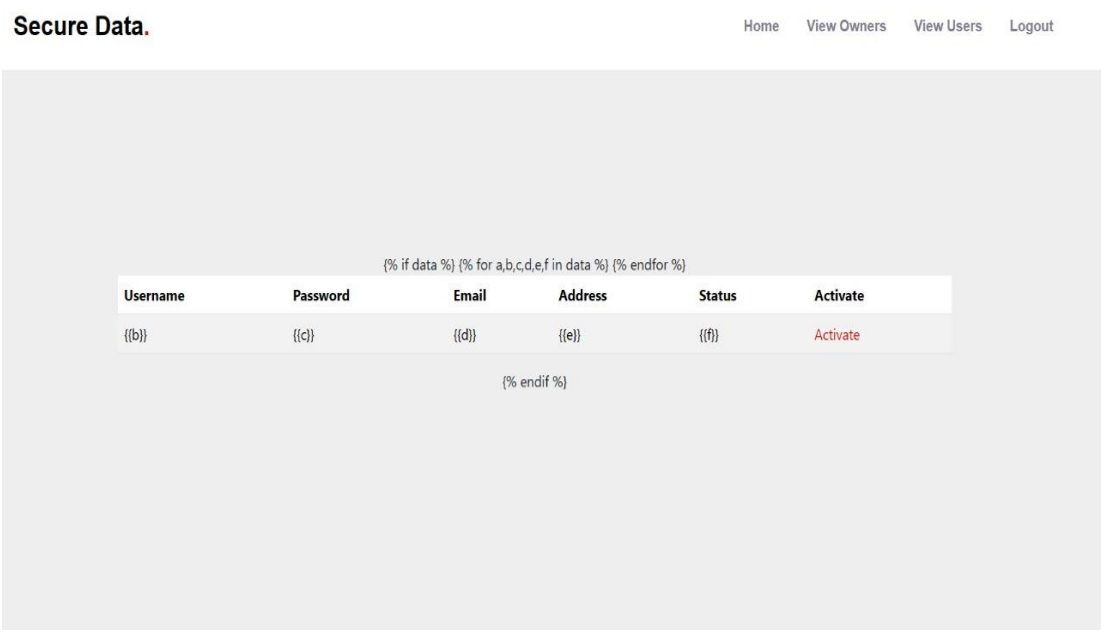


Fig 7.7 Owner & User Activation

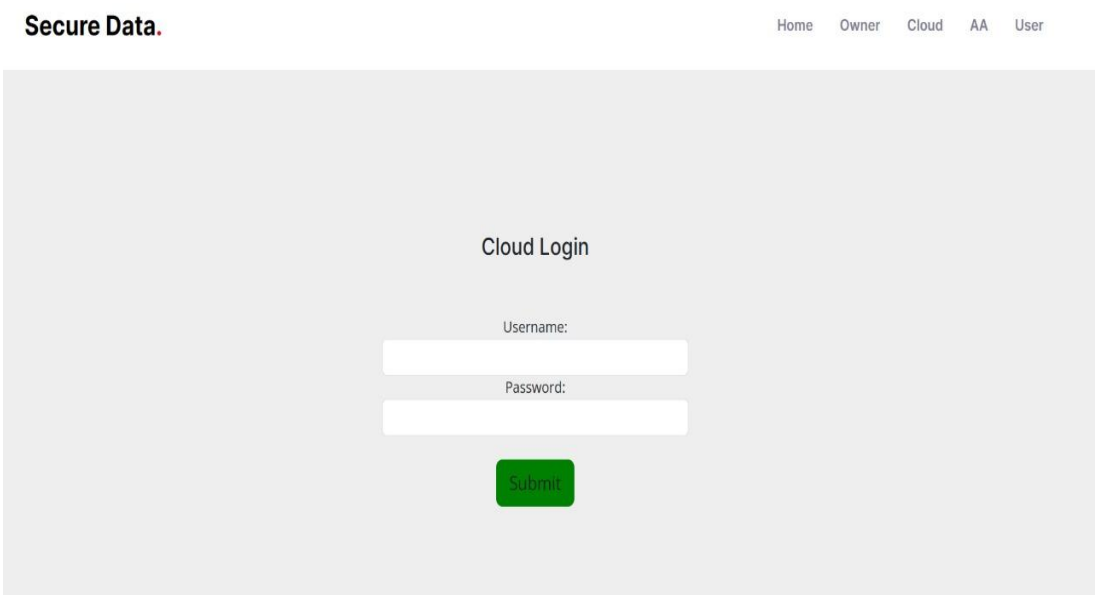


Fig 7.8 Cloud Login

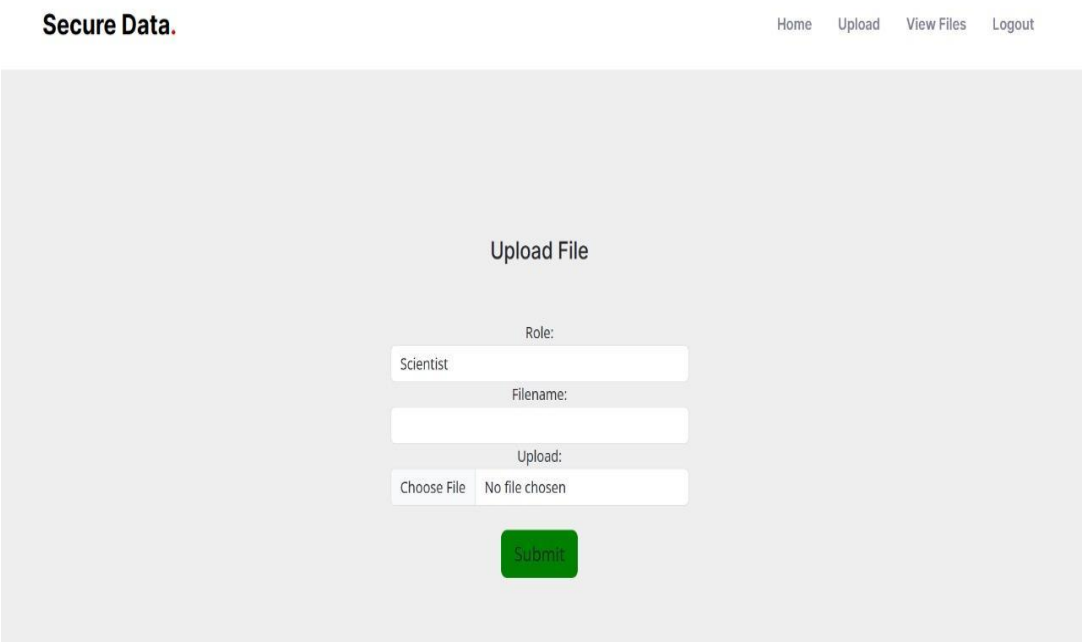


Fig 7.9 Upload File

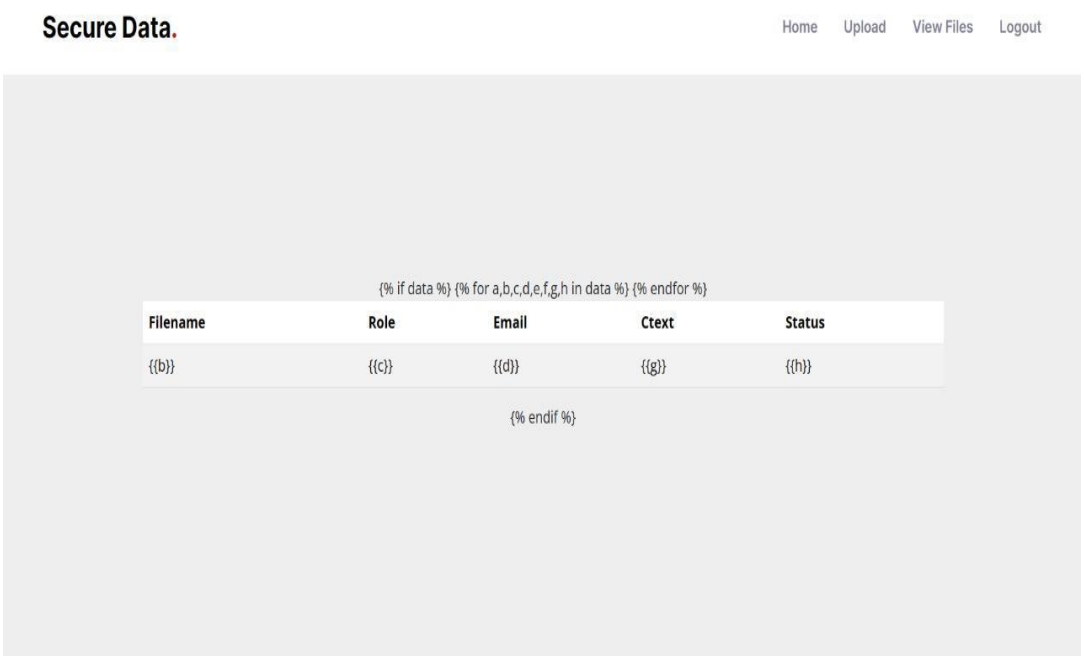


Fig 7.10 Upload Status

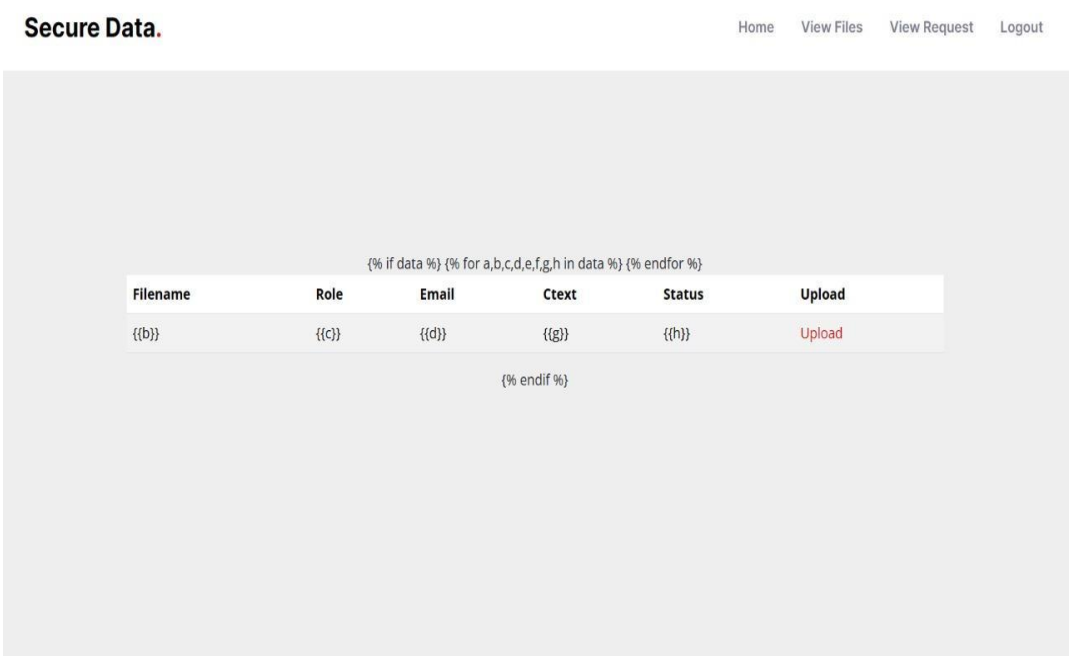


Fig 7.11 Uploaded Status

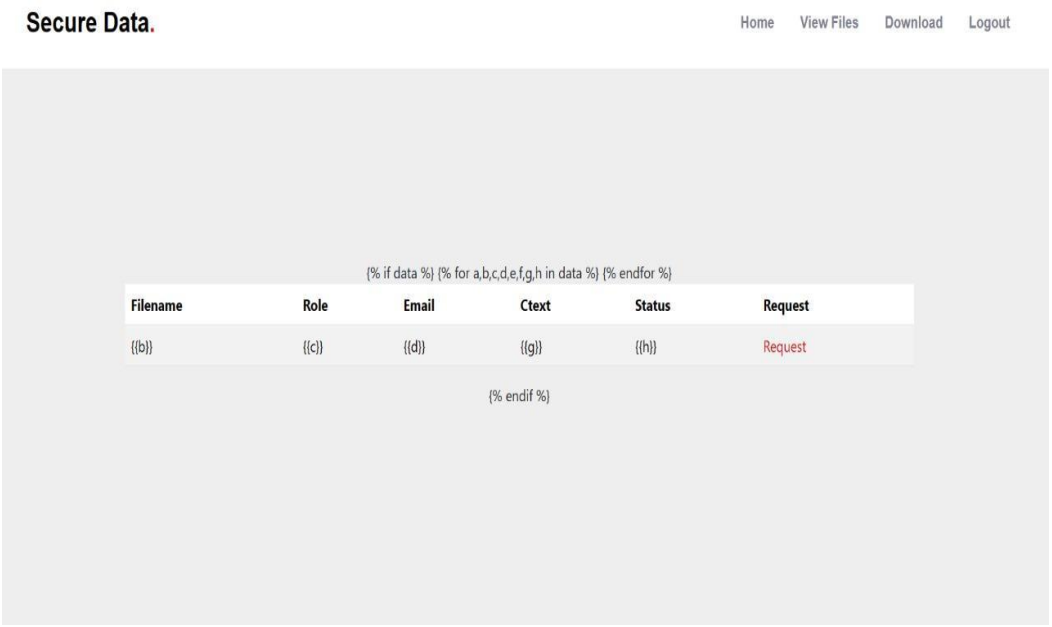


Fig 7.12 Request File

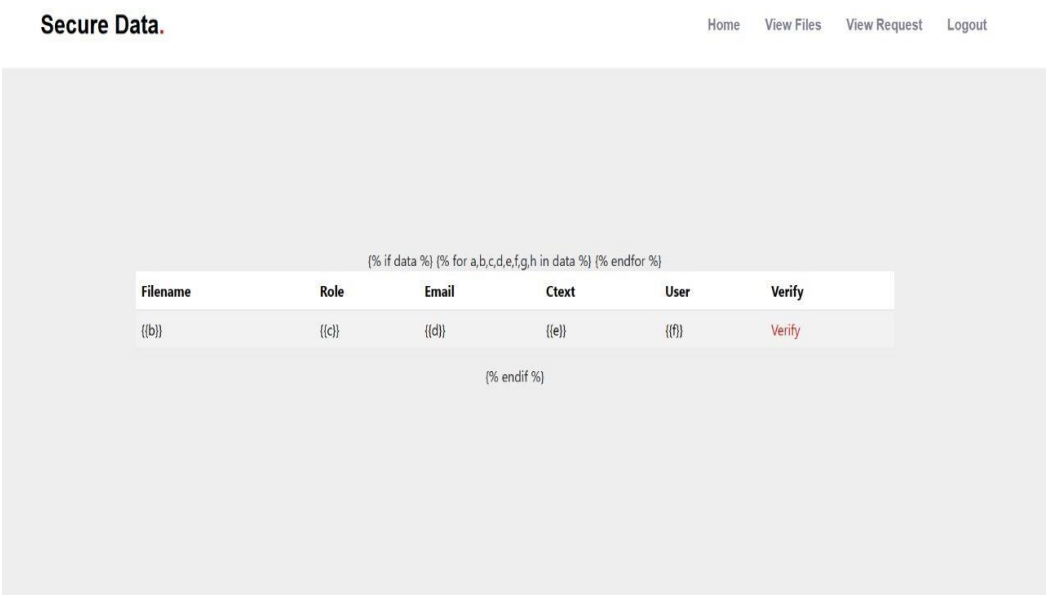


Fig 7.13 Verify User

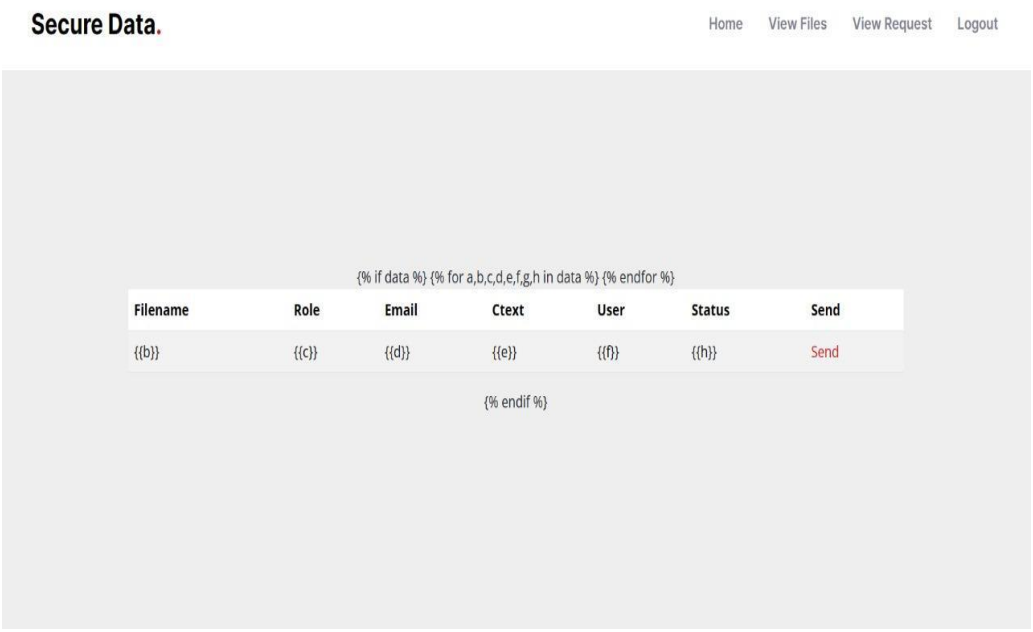


Fig 7.14 Send File To User

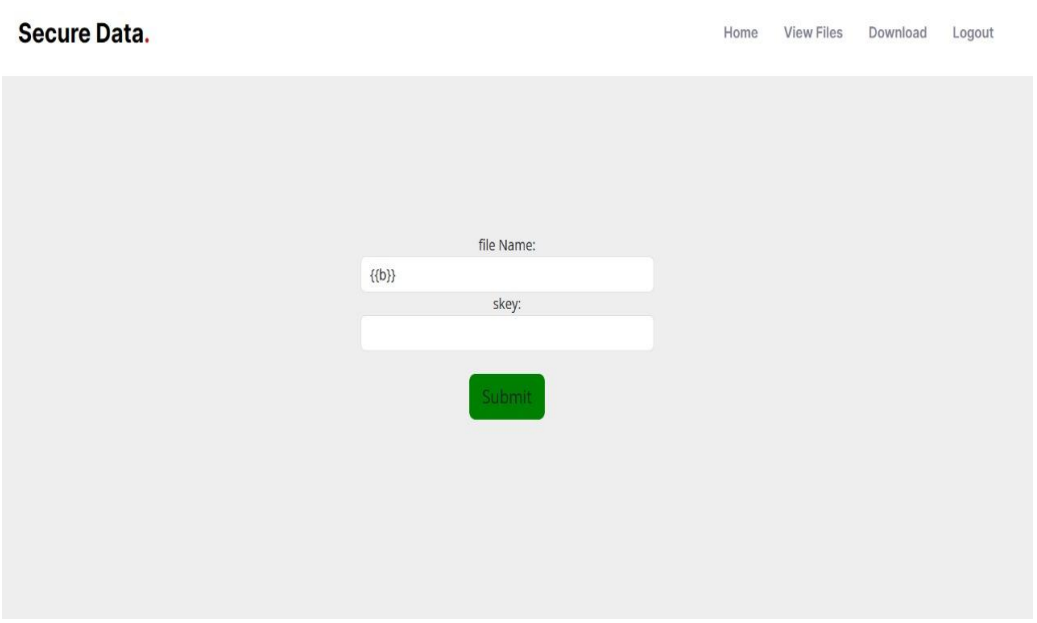


Fig 7.15 User Secret Key

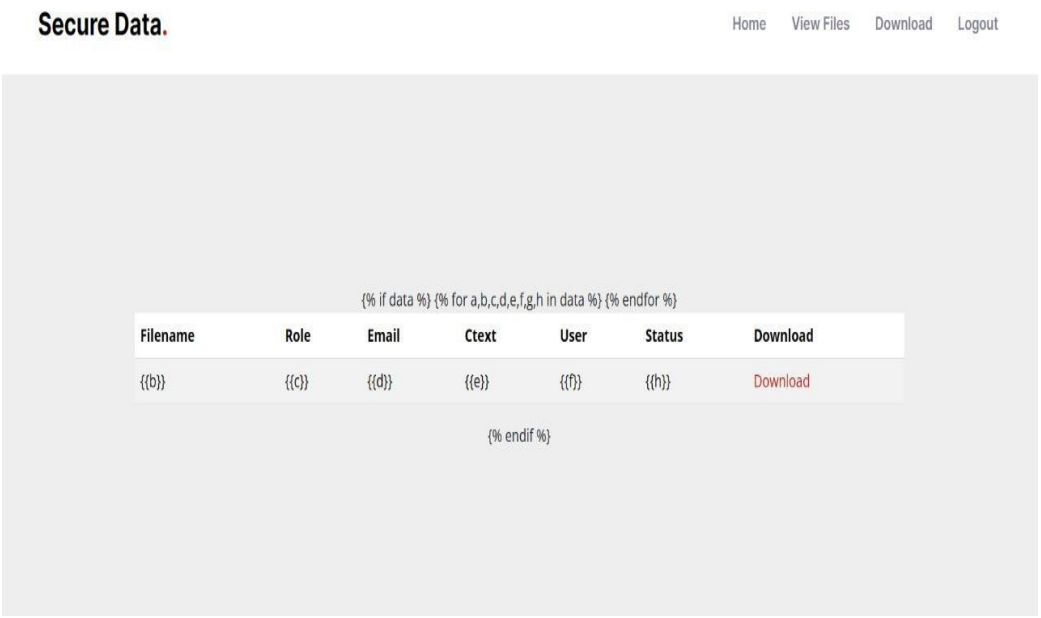


Fig 7.16 Download File

8. CONCLUSION

Our proposed method aims to make it easier for people to control who can access data in the cloud. We mix different methods like CP-ABE, RBAC, and PMI to create a system that can handle lots of users and rules without problems.

A big part of our plan is using attribute certificates (AC) to help decide who can access what within CP-ABE. This helps us manage user info better and makes it simpler to update keys when needed, especially if someone's access is revoked. It also cuts down on the work needed to manage all this info.

We've even built a web app to demonstrate how our key update system works. It shows how our approach can reduce risks in cloud access control. This real-world example proves our plan isn't just theory—it's practical and useful.

In short, our approach combines different techniques to solve access control challenges in cloud storage. By adding features like attribute certificates and smart key updates to the mix, we're making data sharing in the cloud safer, easier, and more flexible for everyone involved.

8.1 Future Enhancements

In our upcoming work, we're focusing on two key areas: ensuring secure auditing of data access control and key management, as well as testing our proposed scheme in a real cloud setting.

Firstly, we'll delve into creating solutions that meet the stringent requirements of regulatory bodies concerning security compliance. This involves developing methods for securely auditing data access control and managing keys. These efforts will ensure that our system meets the standards set by regulatory bodies.

Secondly, we plan to take our proposed scheme out of the lab and into a real cloud environment. By doing this, we can assess its performance on a larger scale, using bigger datasets with more users and attributes. This real-world testing will help us confirm that our AC-CP-ARBE model is both scalable and efficient.

Furthermore, we'll be focusing on revocation, which involves managing situations where access rights need to be taken away. We'll develop strategies to handle this effectively, ensuring that revoked access rights are properly managed within our system.

To illustrate our findings and progress, we'll provide visual representations such as graphs. These graphs will show metrics like the number of access rights revoked over time, helping us understand how our system performs in handling access changes. Overall, our goal is to continue improving our system and ensuring its effectiveness in real-world cloud environments.

9. REFERENCES

[1] M. Li, S. Yu, Y. Zheng, K. Ren and W. Lou, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption", IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 1, pp. 131-143, Jan. 2013.

[2] J. Bethencourt, A. Sahai and B. Waters, "Ciphertext-policy attribute-based encryption", Proc. IEEE Symp. Secur. Privacy, pp. 321-334, 2007.

[3] Z. Wan, J. Liu and R. H. Deng, "HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing", IEEE Trans. Inf. Forensics Secur., vol. 7, no. 2, pp. 743-754, Apr. 2012.

[4] K. Yang, X. Jia, K. Ren, B. Zhang and R. Xie, "DAC-MACS: Effective data access control for multiauthority cloud storage systems", IEEE Trans. Inf. Forensics Secur., vol. 8, no. 11, pp. 1790-1801, Nov. 2013.

[5] M. Mambo and E. Okamoto, "Proxy cryptosystems: Delegation of the power to decrypt ciphertexts", IEICE Trans. Fundamentals Electron. Commun. Comput. Sci., vol. E80-A, no. 1, pp. 54-63, 1997.

[6] V. Goyal, O. Pandey, A. Sahai and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data", Proc. 13th ACM Conf. Comput. Commun. Secur., pp. 89-98, 2006