

# **DataCrypt: Client Side File Encryption for Secure Cloud Storage**

A PROJECT REPORT  
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE AWARD OF THE DEGREE  
OF  
BACHELOR OF TECHNOLOGY  
IN  
INFORMATION TECHNOLOGY

Submitted by:

**Rajnish Kumar (2K21 / IT / 139)**

**Ritik (2K21 / IT / 145)**

**Rugung Daimary (2K21 / IT / 151)**

Under the supervision of

**Dr. Bindu Verma**



**DEPARTMENT OF INFORMATION TECHNOLOGY**

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-1100

## **CANDIDATE'S DECLARATION**

We, Rajnish Kumar (2K21/IT/139), Ritik (2K21/IT/145) and Rugung Daimary (2K21/IT/151) students of B. Tech. (Information Technology), hereby declare that the project Dissertation titled “DataCrypt: Client Side File Encryption for Secure Cloud Storage” which is submitted by us to the Department of Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is original and not copied from any source without proper citation. His work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

**Rajnish Kumar (2K21/IT/139)**

Date: 27 / 05 / 2025

**Ritik (2K21/IT/145)**

**Rugung Daimary (2K21/IT/151)**

## **CERTIFICATE**

I hereby certify that the Project Dissertation titled “DataCrypt: Client Side File Encryption for Secure Cloud Storage” which is submitted by Rajnish Kumar (2K21/IT/139), Ritik (2K21/IT/145) and Rugung Daimary (2K21/IT/151) Department of Information Technology, Delhi Technological University, Delhi in the partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

**Dr. Bindu Verma**

Date: 27 / 05 / 2025

**Supervisor**

## **ABSTRACT**

If Alice, the administrator in Company X, uses a secure cloud storage system and provides access to authorized employees in the company. The system follows a Zero-Knowledge encryption mechanism, such that only the intended recipients have the ability to decrypt the data.

Here we present DataCrypt, a client-side encryption solution that uses AES-256 to encrypt data and ECC to provide secure key exchange.

The admin can securely share files by setting multiple recipients with their public keys to derive an encryption key through an elliptic curve key exchange algorithm. Supabase is used to store the encrypted file and the encrypted AES keys in a cloud storage system.

Only the authorized recipients, such as the administrator are able to decrypt the file with their private keys. In contrast to classical cloud storage systems, our solution guarantees that encryption happens locally on the client device. The cloud provider receives no plaintext data. In addition, the admin must encrypt the file just once, limiting computational cost and minimizing storage needs. Users and cloud storage system communication is protected with TLS and authenticated with JWT tokens.

As far as we know DataCrypt represents the inaugural cloud storage software which unifies AES-256 encryption with ECC protocols within a Zero-Knowledge framework to maintain file sharing performance and top security standards.

## **ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to **Dr. Bindu Verma**, Assistant Professor, Department of Information Technology, Delhi Technological University, for her invaluable guidance, constructive feedback, and unwavering support throughout the duration of this project. Her insightful suggestions and encouragement have been pivotal to the successful completion of this work.

We also extend our heartfelt thanks to our friends and peers for their patience, assistance, and constant motivation during this endeavour. Their encouragement and support was crucial in helping us navigate challenges and achieve our objectives.

Additionally, we wish to acknowledge the contributions of researchers such as **Qin Liu**, **Guojun Wang**, **Uma Somani**, **Kanika Lakhani**, and **Manish Mundra**, whose significant research papers have provided us with essential references and a deeper understanding of the cloud security domain.

Finally, we are grateful to the faculty and staff of the Department of Information Technology at **Delhi Technological University** for fostering an environment conducive to learning, research, and innovation.

## **CONTENTS**

Cover Page	i
Candidate's Declaration	ii
Certificate	iii
Acknowledgement	iv
Abstract	v
Contents	vi
List of Figures	vii
List of Tables	viii
List of Symbols, Abbreviations and Nomenclature	ix
Chapter 1: Introduction	10-13
Chapter 2: System Design and Implementation	14-34
Chapter 3: Result, Discussion and Conclusion	35-39
References	40-42

## **LIST OF TABLES**

<b>Table No.</b>	<b>Title</b>	<b>Page no.</b>
1.	DES vs AES comparison.....	19
2.	RSA vs ECC comparison .....	20

## **LIST OF FIGURES**

<b>Table No.</b>	<b>Title</b>	<b>Page no</b>
1.	System Architecture Diagram of Local Interface.....	14
2.	System Architecture Diagram of Remote Server.....	15
3.	AES vs DES cpu comparison.....	19
4.	AES vs DES speed comparison.....	19
5.	DES vs ECC cpu comparison.....	20
6.	DES vs ECC speed comparison.....	20
7.	Flowchart.....	21
8.	Local Interface.....	21
9.	Generation of key pair.....	22
10.	Public key of the user.....	22
11.	Encryption Flow Diagram.....	23
12.	Need of recipient's public key for file encryption.....	24
13.	File encryption successful.....	24
14.	Remote Server Diagram.....	25
15.	Sign up page for remote server.....	26
16.	Authentication Flow.....	26
17.	Sign in page for remote server.....	27
18.	Searching for the receiver.....	27
19.	Invitation sent to sign up to the remote server.....	28
20.	Invitation mail to sign up.....	28
21.	Receiver found to send files encrypted file.....	29
22.	Uploading the encrypted file and encrypted aes file.....	30
23.	Decryption Flow Diagram.....	30
24.	Encrypted files received by receiver.....	31
25.	Decrypting the file using public key of sender.....	32
26.	File Decryption Successful.....	33



## **LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE**

AES: Advanced Encryption Standard

ECC: Elliptic Curve Cryptography

ECDH: Elliptic-Curve Diffie–Hellman

RSA: Rivest, Shamir, Adleman

DES: Data Encryption Standard

AWS: Amazon Web Server

JWT: JSON Web Token

TLS: Transport Layer Security

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 BACKGROUND**

In today's digital age, storing files in the cloud has become extremely common. Whether it's photos, documents, or sensitive company data, people rely heavily on cloud services like Google Drive, Dropbox, and AWS to keep their files safe and accessible. However, the security of these files is still a big concern. Most cloud platforms encrypt files on their own servers, which means users have to trust that these services won't misuse their data or fall victim to security breaches.

That's where client-side encryption comes into play. Instead of relying on the cloud provider to secure the data, the user encrypts the file before it even leaves their device. This way, even if the data gets intercepted or the cloud server is hacked, the encrypted files remain unreadable without the correct decryption key. This project, Datacrypt, is based on this principle — securing files at the user's end using strong encryption techniques before uploading them to the cloud.

#### **1.2 MOTIVATION**

The idea for this project came from the growing need to protect personal and confidential data in a world where cyber threats are increasing every day. Data breaches are happening more frequently, and many of them are due to weak security practices on cloud platforms.

We wanted to create something that gives users full control over the privacy of their files. Imagine a situation where someone wants to share a sensitive document with a colleague. Instead of trusting the cloud to handle the security, what if the user could encrypt that document locally and then send it — already protected — to the receiver? That's the motivation behind DataCrypt: a system that empowers users to keep their files private and share them securely, without depending too much on external services.

### **1.3 PROBLEM STATEMENT**

The main problem we set out to solve is the lack of end-to-end control users have over their data when they use cloud storage services. Most platforms encrypt data only after it's uploaded, and they manage the encryption keys themselves. This means users have to trust them not just with their files, but also with the keys needed to decrypt them.

DataCrypt addresses this issue by allowing users to encrypt their files on their own devices before uploading. It also ensures that only the intended receiver — and no one else — can decrypt and view the file. The project also provides a smooth way to share encrypted files, making it practical for everyday use.

### **1.4 OBJECTIVES**

The main goals of this project are:

- To build a local application where users can encrypt their files using a combination of AES (for file content) and ECC (for key encryption).

- To allow users to generate their own public and private keys securely on their own device.
- To implement a secure sign-up and login system where users can register with their email, password, and OTP verification, and upload their public key to a central server.
- To enable users to search for a recipient's email and get their public key to encrypt the file for them.
- To make file sharing secure by uploading the encrypted file and encrypted AES key to a cloud server (AWS), and generating a download link.
- To allow the receiver to download and decrypt the file locally using their private key and the sender's public key.

## **1.5 SCOPE OF THE PROJECT**

Datacrypt is designed for anyone who wants to take their file security into their own hands — from students and professionals to businesses dealing with sensitive information. The system includes:

A local interface to encrypt and decrypt files.

- A secure backend for account management and public key storage using MongoDB.
- A cloud component that stores encrypted files on AWS and generates download links.
- A complete workflow for secure file sharing using public-key encryption.

While this project is a prototype, it lays the foundation for a much larger and scalable system that can be integrated into real-world cloud storage platforms.

## **CHAPTER 2**

### **SYSTEM DESIGN AND IMPLEMENTATION**

#### **2.1 OVERVIEW OF THE SYSTEM**

The Datacrypt system is built around a simple yet powerful idea: encrypt files before they are uploaded to the cloud, and make sure only the intended receiver can decrypt them. The system is split into two main parts:

- Local Interface (Client Side) – This is the part the user interacts with directly. It runs on the user's system and allows them to generate keys, encrypt and decrypt files securely.

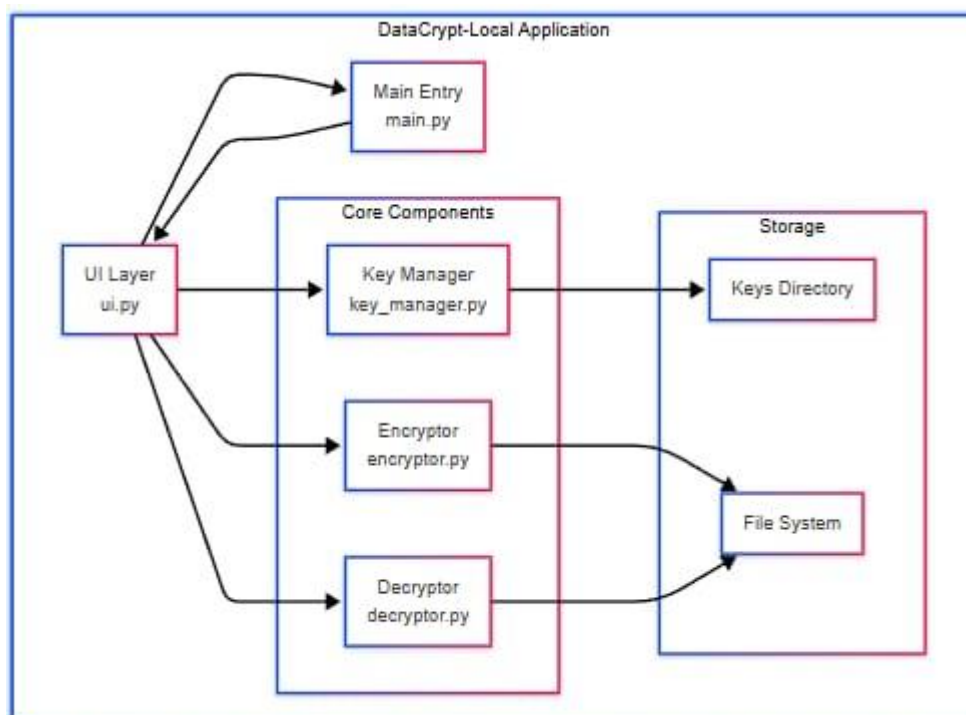


Fig 1: System Architecture Diagram of Local Interface

- Remote Server (Cloud Backend) – This part handles user authentication, stores public keys, and provides the cloud storage feature for encrypted files. It acts as a bridge between users for secure file sharing.

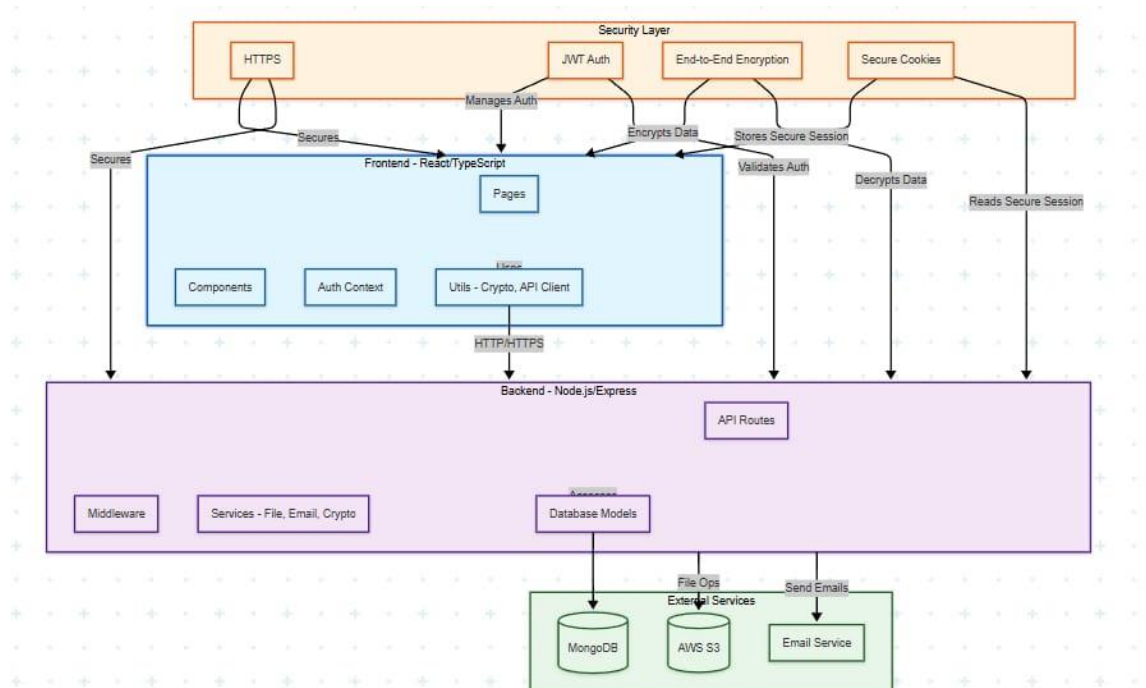


Fig 2: System Architecture Diagram of Remote Server

Together, these components make file encryption, upload, retrieval, and decryption seamless and secure.

## 2.2 TECHNOLOGIES USED

**Python with PyQt6** – For building the local interface that users run on their computers.

**AES (Advanced Encryption Standard)** – Used for encrypting/decrypting the contents of the file.

**ECC (Elliptic Curve Cryptography)** – Used for encrypting the AES key so that only the receiver can decrypt it.

**Node.js** – Backend framework used for building the server.

**MongoDB** – Database used to store user details and public keys.

**AWS S3** – Cloud storage used to store the encrypted files and encrypted AES keys.

**JWT** – Used for managing secure user sessions.

**TLS (HTTPS)** – Ensures secure communication between the client and server.

## 2.3 CRYPTOGRAPHIC TECHNIQUES USED

The DataCrypt project employs a hybrid cryptographic model that integrates both symmetric encryption (AES-256) and asymmetric encryption (ECC). This combination ensures strong security while maintaining efficiency in cloud environments.



### Elliptic Curve Cryptography (ECC):

- ECC is used for secure key exchange and user authentication.
- ECC was first proposed in 1985 by Neal Koblitz and Victor Miller. In ECC, an elliptic curve over a field  $K$  has a set of points  $(X_i, Y_i)$  in a plane and this set is finite, called  $E$ . ECC is known for being one of the most secure cryptographic algorithms and is especially useful in cloud environments because it's efficient.

### Mathematical Formulation:

The standard form of ECC is given by the equation:

$$y^2 = x^3 + ax + b$$

where  $a$  and  $b$  are fixed parameters. ECC operates with a Group Operator, denoted by the symbol '+'. The Group Operator is used to find  $P$ , a point on the curve, and proceed with the computation as:

$$P + P, P + P + P, \dots$$

This series of operations makes it computationally difficult for attackers to reverse-engineer the encryption, thereby making the data highly secure.

### Key Agreement Using ECDH Algorithm:

In Elliptic Curve Diffie-Hellman (ECDH), both parties (e.g., Cloud A and Cloud B) agree on a shared session key that is used for secure data encryption and decryption.

Key Generation:

- Cloud A selects a private key  $X_A$  and generates the public key:

$$Y_A = X_A * P$$

- Similarly, Cloud B selects a private key  $X_B$  and generates its public key:

$$Y_B = X_B * P$$

### Key Exchange:

Once both parties have generated their public keys, they exchange these keys:

Cloud A computes the shared secret  $K_A$  as:

$$K_A = X_A * Y_B = k_1 * k_2 * P$$

Cloud B computes the same shared secret  $K_B$  as:

$$K_B = X_B * Y_A = k_2 * k_1 * P$$

Both parties arrive at the same shared session key  $K$ , ensuring that the encryption process is secure. The session key will be used to encrypt and decrypt data transmitted between the two parties. This process ensures that private keys are never exposed and only the shared session key is used for encryption.

### Advanced Encryption Standard (AES-256):

AES-256 ensures high-level security when encrypting files locally on the user's device. The encrypted data is then uploaded to the cloud for secure storage.

Encryption Process:

- When Sender A wants to send data to Receiver B, Sender A encrypts the message using the AES encryption key.
- The AES decryption process makes sure that only Receiver B can decrypt the message, maintaining confidentiality.

## 2.4 EXPERIMENTAL RESULTS

### 2.4.1 SYMMETRIC ALGORITHM COMPARISON (DES vs AES)

Symmetric Algorithms							
Algorithms	Key Size (bits)	Encrypting Time (ms)	Decrypting Time (ms)	Total Time taken(ms)	Throughput (byte/ms)	CPU Speed (%)	Memory Requirement (%)
DES	56-bits	525	548	1073	13.33	5.4	7.7
AES	128-bits	408	525	933	78.4	5.0	7.6

Table 1: DES vs AES comparison

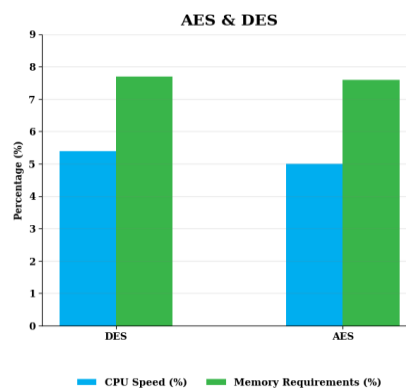


Fig 3: AES vs DES cpu comparison

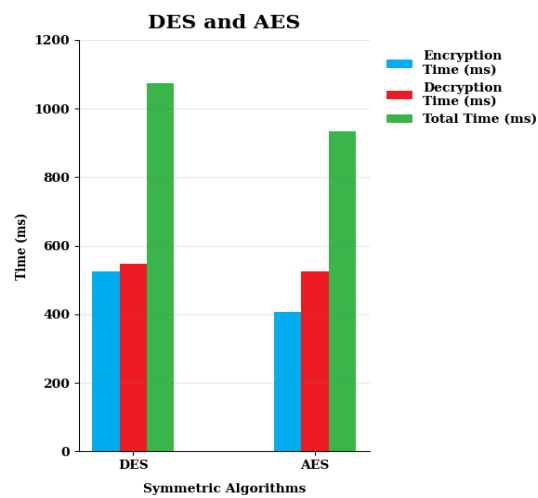


Fig 4: AES vs DES speed comparison

## 2.4.2 ASYMMETRIC ALGORITHM COMPARISON (RSA vs ECC)

Asymmetric Algorithms						
Algorithms	Key-Size (bits)	Encryption Key Time (s)	Decryption Key Time (s)	Total Time (s)	CPU Speed (%)	Memory Requirement (%)
RSA	256-bits	939	608	1.547	6.5	7.8
ECC	256-bits	550	542	1.092	2.4	7.5

Table 2: RSA vs ECC comparison

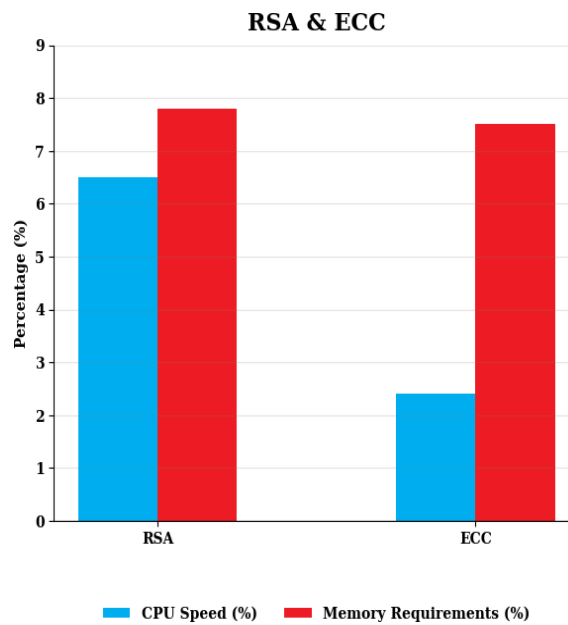


Fig 5: RSA vs ECC cpu comparison

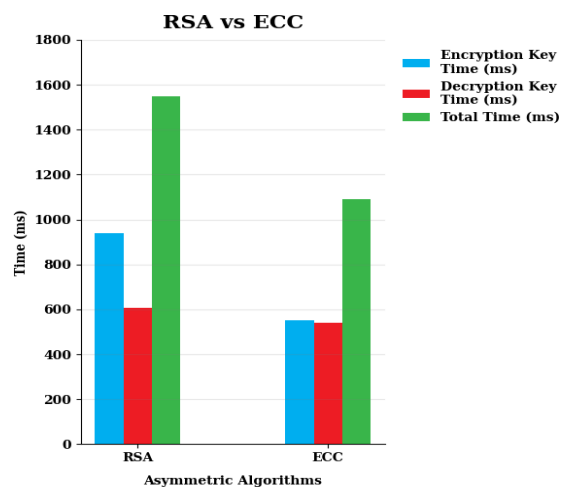


Fig 6: RSA vs ECC speed comparison

## 2.5 WORKING OF THE LOCAL INTERFACE

### 2.5.1 KEY GENERATION

The first step is for the user to generate a pair of cryptographic keys: a public key and a private key. This is done locally using ECC. The private key stays with the user and is never shared, while the public key can be uploaded to the server for others to use when encrypting files meant for this user.

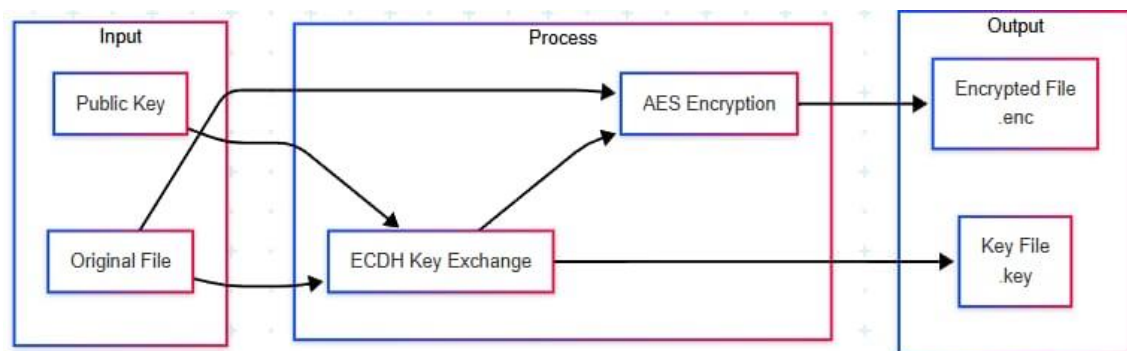


Fig 7: Flowchart

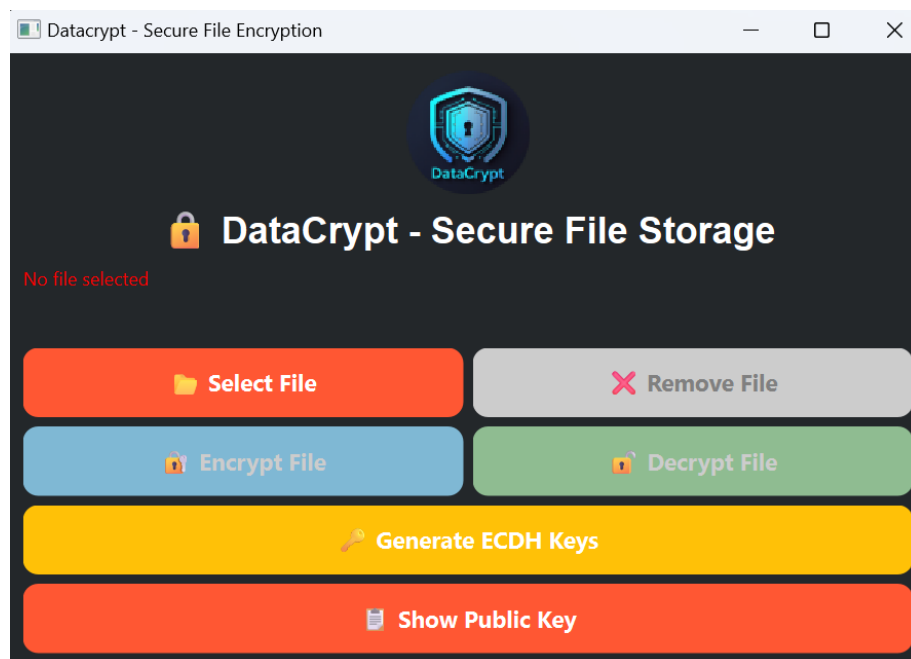


Fig 8: Local Interface

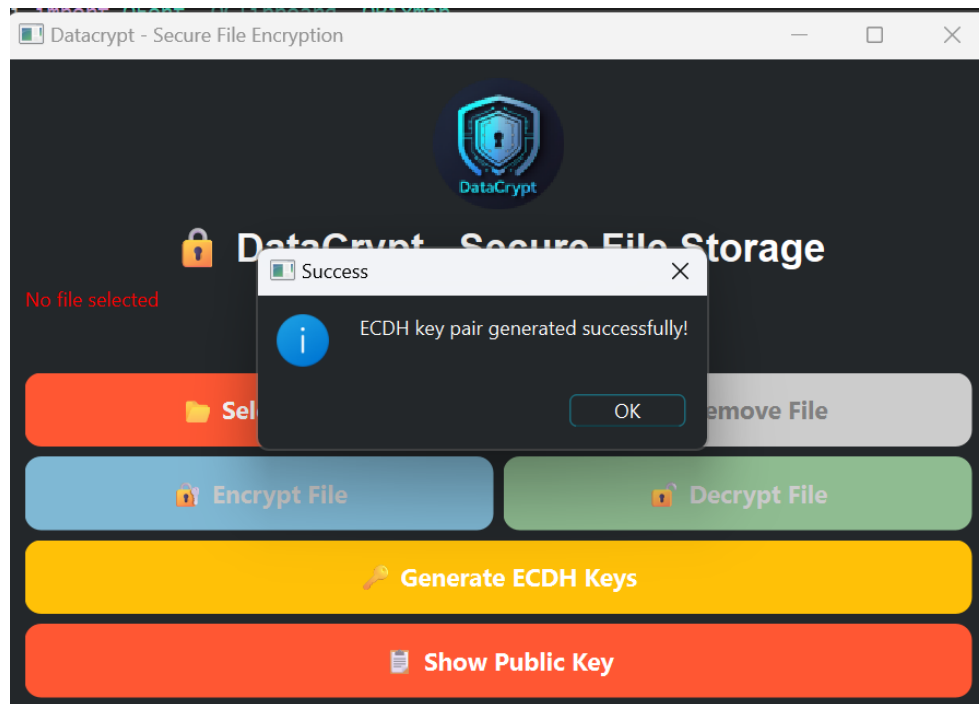


Fig 9: Generation of key pair

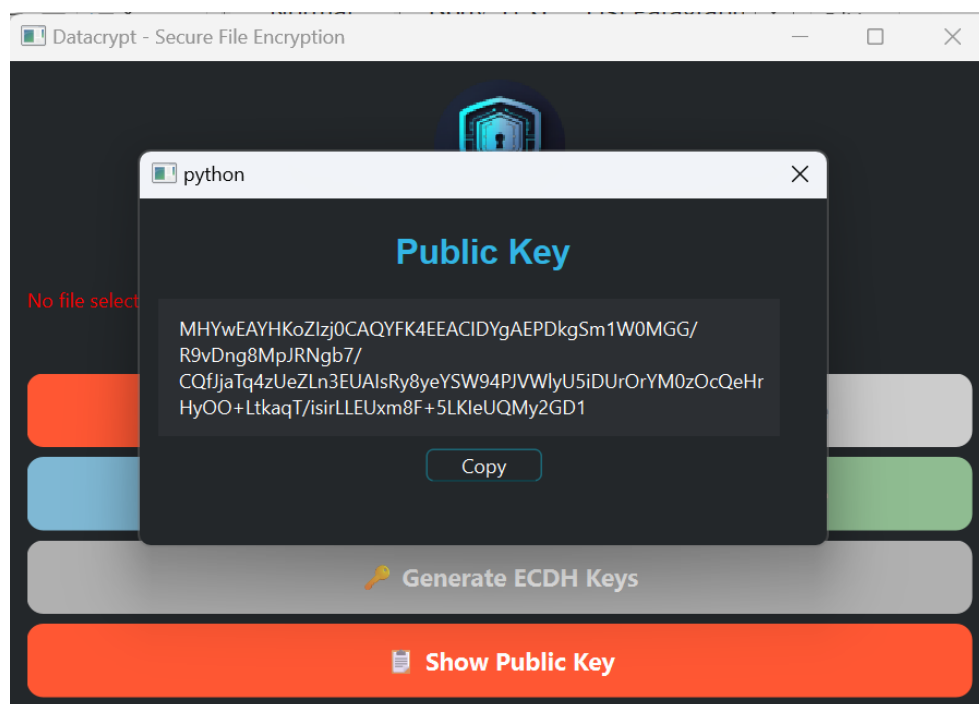


Fig 10: Public Key of the user

## 2.5.2 FILE ENCRYPTION

When a user wants to send a file:

- The file is first encrypted using AES, which is fast and efficient for large files.
- The AES key itself is then encrypted using the receiver's public ECC key.

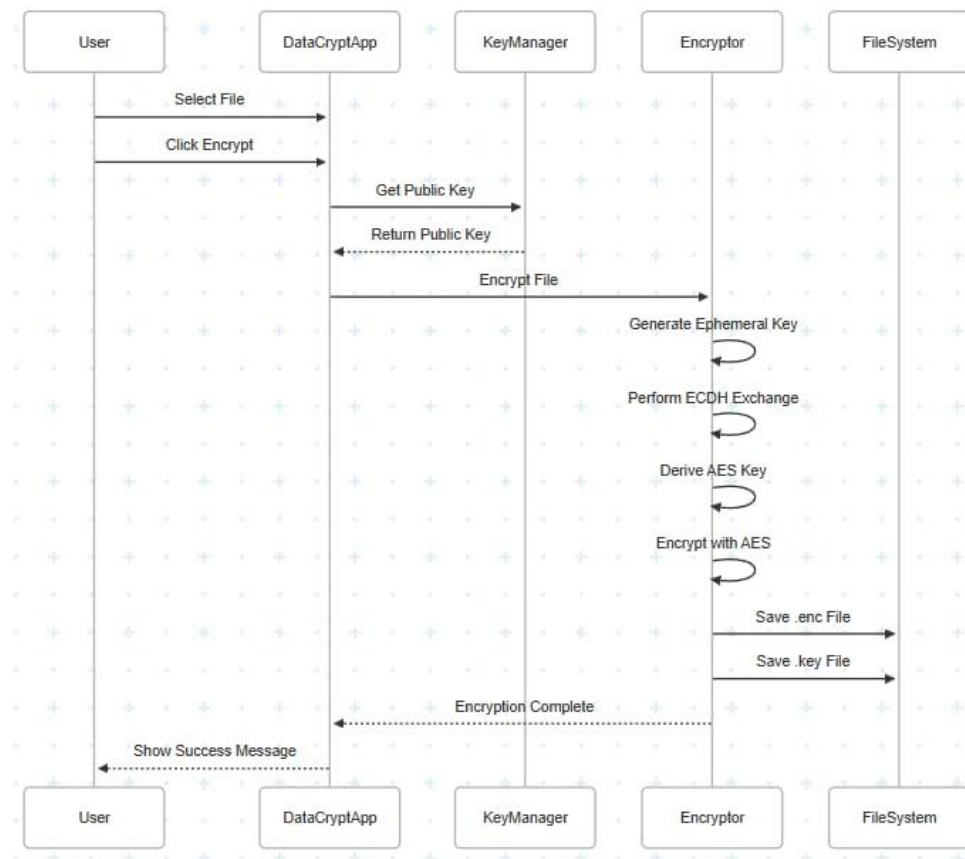


Fig 11: Encryption Flow Diagram

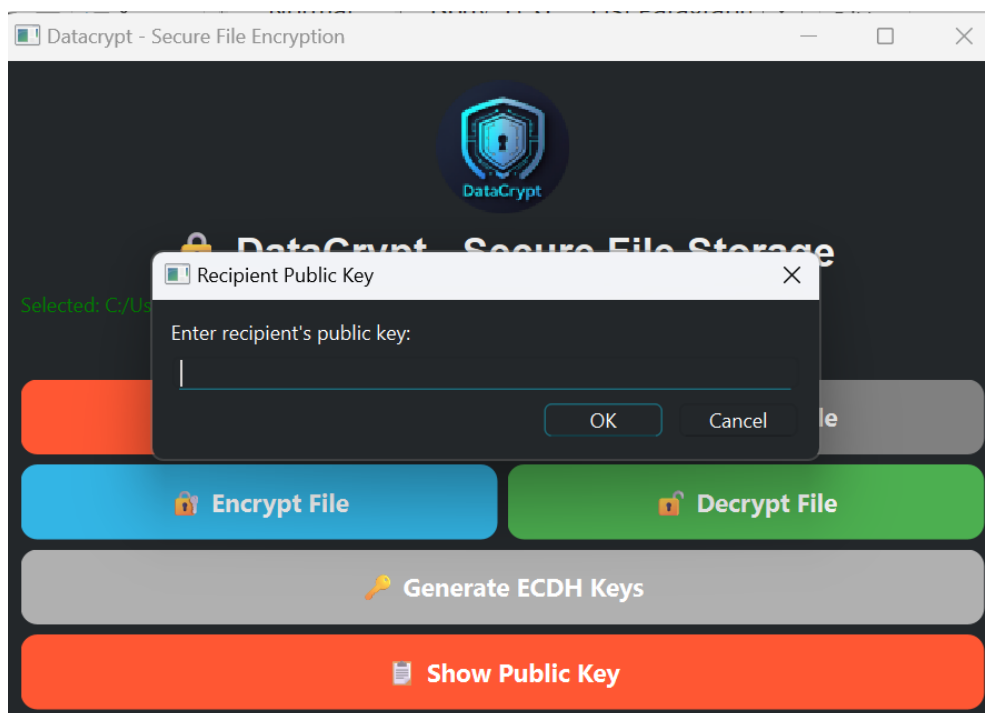


Fig 12: Need of recipient's public key for file encryption

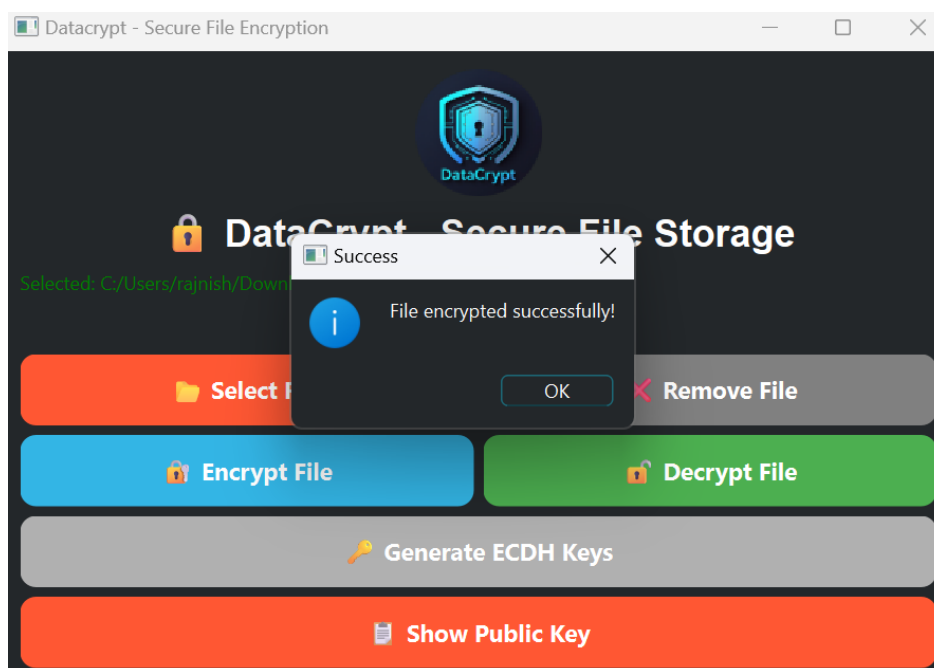


Fig 13: File Encryption Successful



This results in two files:

- An encrypted version of the original file.
- An encrypted AES key.

These are saved locally and are ready to be uploaded to the cloud.

## 2.6 BACKEND AND CLOUD INTEGRATION

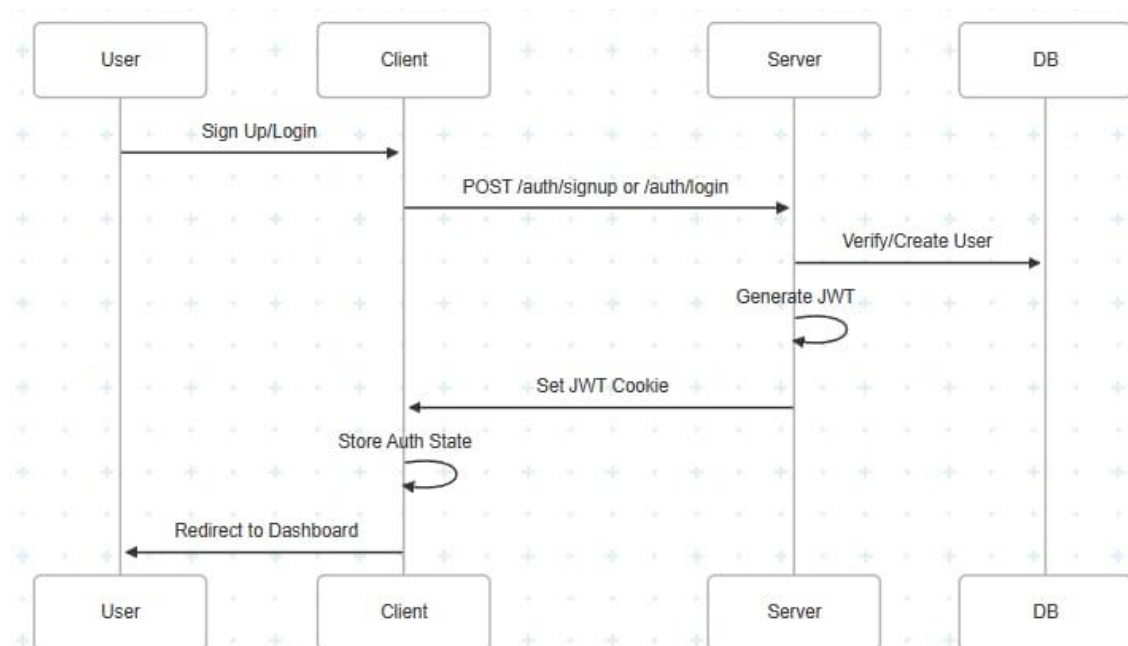


Fig 14: Remote Service Diagram

## 2.6.1 USER SIGN-UP AND SIGN-IN

The remote server allows new users to sign up with their email, password, and an OTP verification step to ensure security. During sign-up, the user also uploads their public key. All this data is securely stored in MongoDB.

**Create Your Secure Account**

Full Name  
John Doe

Email Address  
you@example.com

Password  
\*\*\*\*\*

Confirm Password  
\*\*\*\*\*

Public Key  
Paste your public key here

**Continue**

[Already have an account? Sign in](#)

Fig 15: Sign Up Page of the remote server

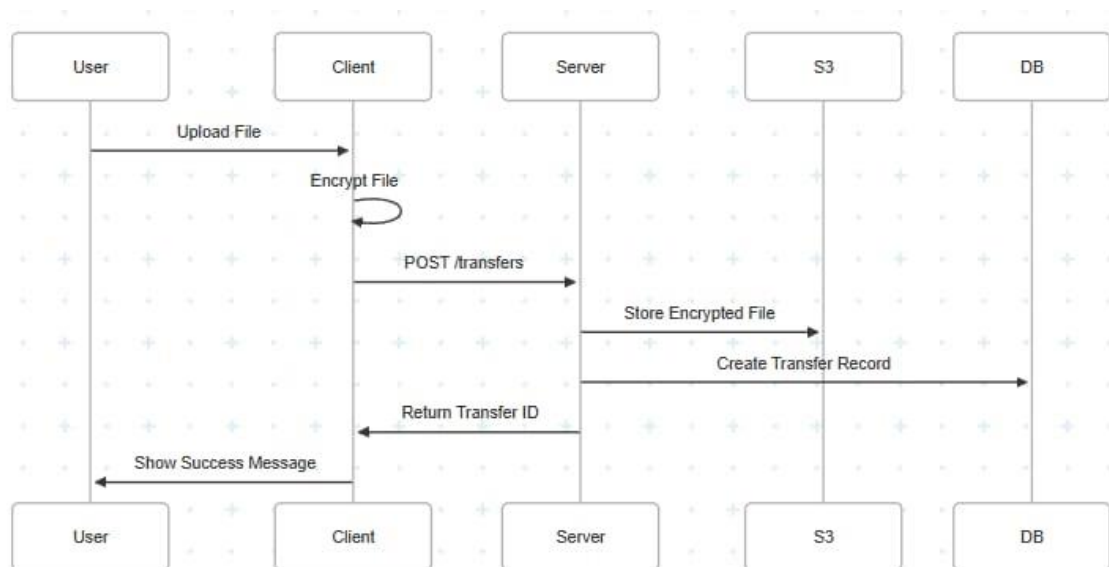
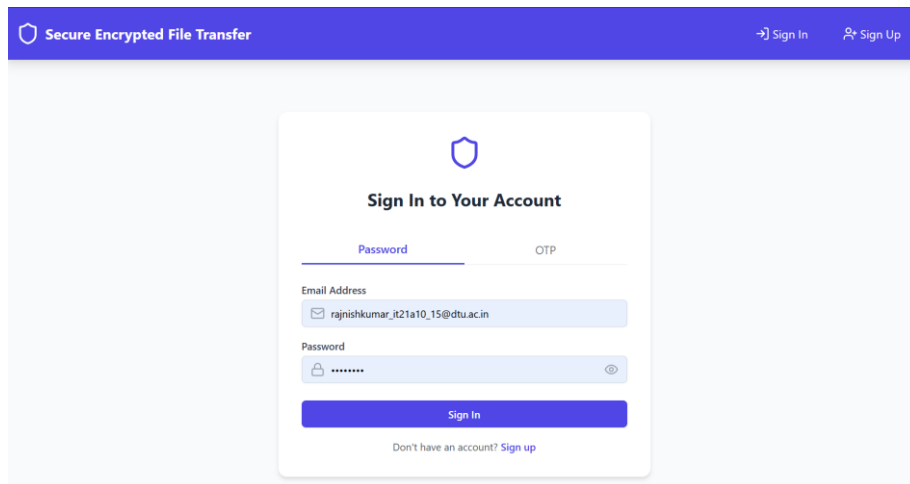


Fig 16: Authentication Flow

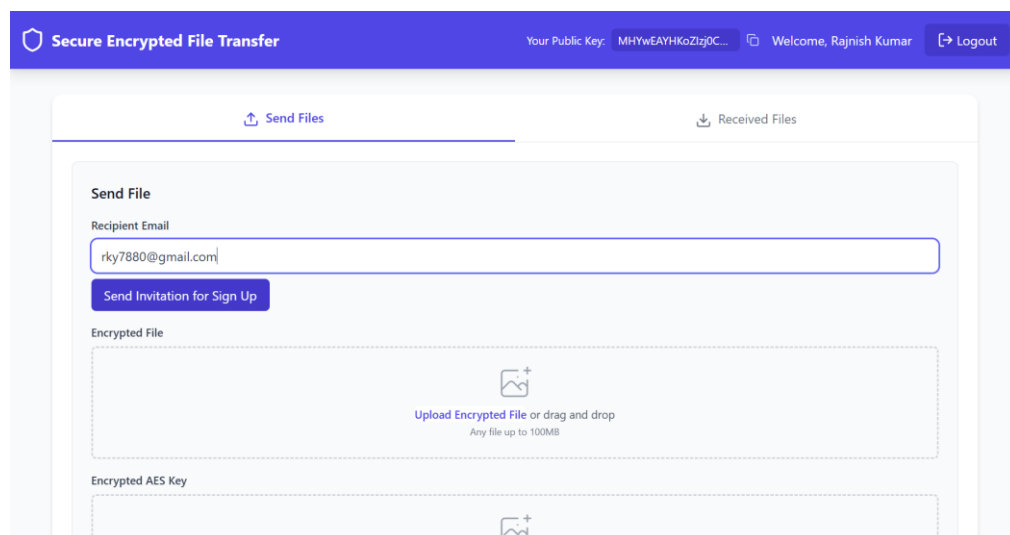


The image shows the 'Sign In to Your Account' page of the 'Secure Encrypted File Transfer' application. The page has a blue header with the application name and links for 'Sign In' and 'Sign Up'. The main content area is white and contains a shield icon, the title 'Sign In to Your Account', and two tabs: 'Password' (selected) and 'OTP'. Below the tabs are input fields for 'Email Address' (containing 'rajnishkumar\_it21a10\_15@dtu.ac.in') and 'Password' (masked with dots). A blue 'Sign In' button is at the bottom, with a link 'Don't have an account? Sign up' below it.

Fig 17: Sign In Page of the remote server

## 2.6.2 SEARCHING FOR A RECEIVER

To send a file to someone, the sender searches for the recipient's email address in the system. If the user is registered, their public key is retrieved. If they are not yet registered, the system can send an auto-generated email asking them to sign up and upload their public key.



The image shows the 'Send File' page of the 'Secure Encrypted File Transfer' application. The page has a blue header with the application name, the user's public key 'MHYwEAYHkoZlj0C...', a welcome message 'Welcome, Rajnish Kumar', and a 'Logout' button. The main content area is white and contains two tabs: 'Send Files' (selected) and 'Received Files'. Below the tabs is a 'Send File' section with a 'Recipient Email' input field (containing 'rky7880@gmail.com') and a blue 'Send Invitation for Sign Up' button. Below this is an 'Encrypted File' section with a dashed box and a plus icon, with the text 'Upload Encrypted File or drag and drop' and 'Any file up to 100MB'. At the bottom is an 'Encrypted AES Key' section with a dashed box and a plus icon.

Fig 18: Searching for the receiver

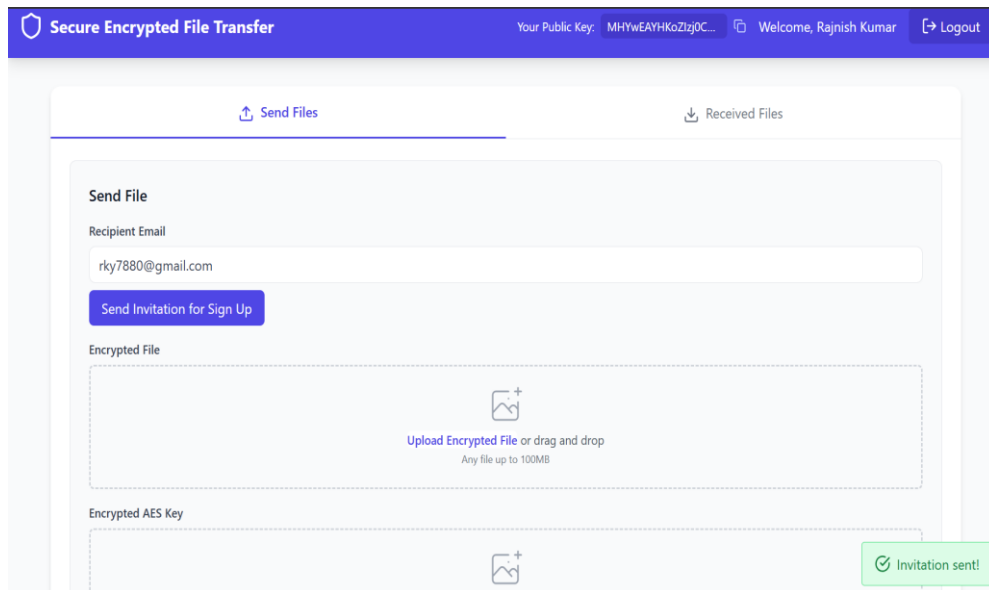


Fig 19: Invitation Sent to sign up to receive the file



Fig 20: Invitation Mail to sign up

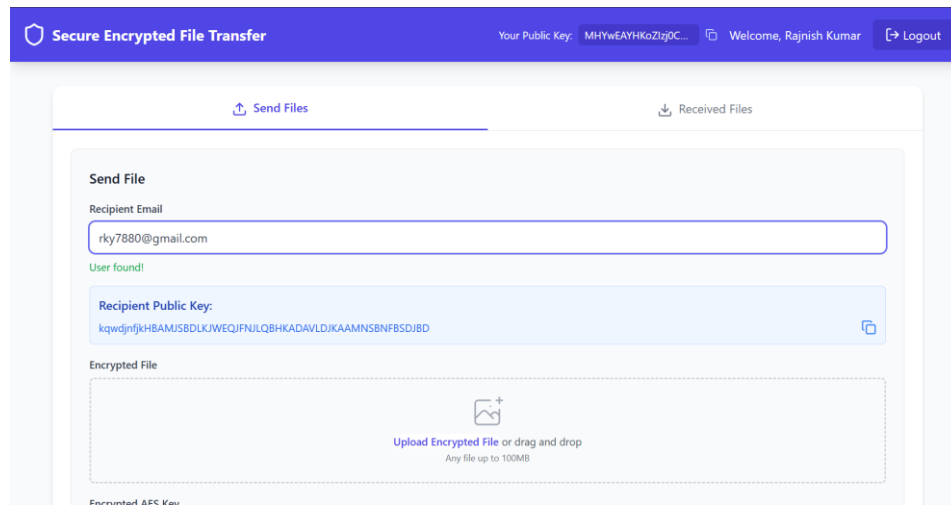


Fig 21: Receiver found to send the encrypted file

### 2.6.3 UPLOADING ENCRYPTED FILES

Once the file is encrypted, the user uploads both:

- The encrypted file.
- The encrypted AES key.

These are stored in AWS S3 securely. A unique download link is generated, which the sender can then share with the recipient.

**SEND FILE**

Recipient Email  
rky7880@gmail.com

User found!

**Recipient Public Key:**  
kqwdfnfjkbAMJSBDLKJWEQ/FNJLQBHKADAVLDJKAAMNSBNFBSDBD

**Encrypted File**

Upload Encrypted File or drag and drop  
Any file up to 100MB

Selected file: Rajnish\_DataCrypt (2).docx.enc (0.14 MB)

**Encrypted AES Key**

Upload Encrypted AES Key or drag and drop  
.key, .bin, or .txt (up to 10KB)

Selected key: Rajnish\_DataCrypt (2).docx.key (0.21 KB)

Fig 22: Uploading the encrypted file and encrypted AES key file

## 2.7 FILE DECRYPTION BY THE RECEIVER

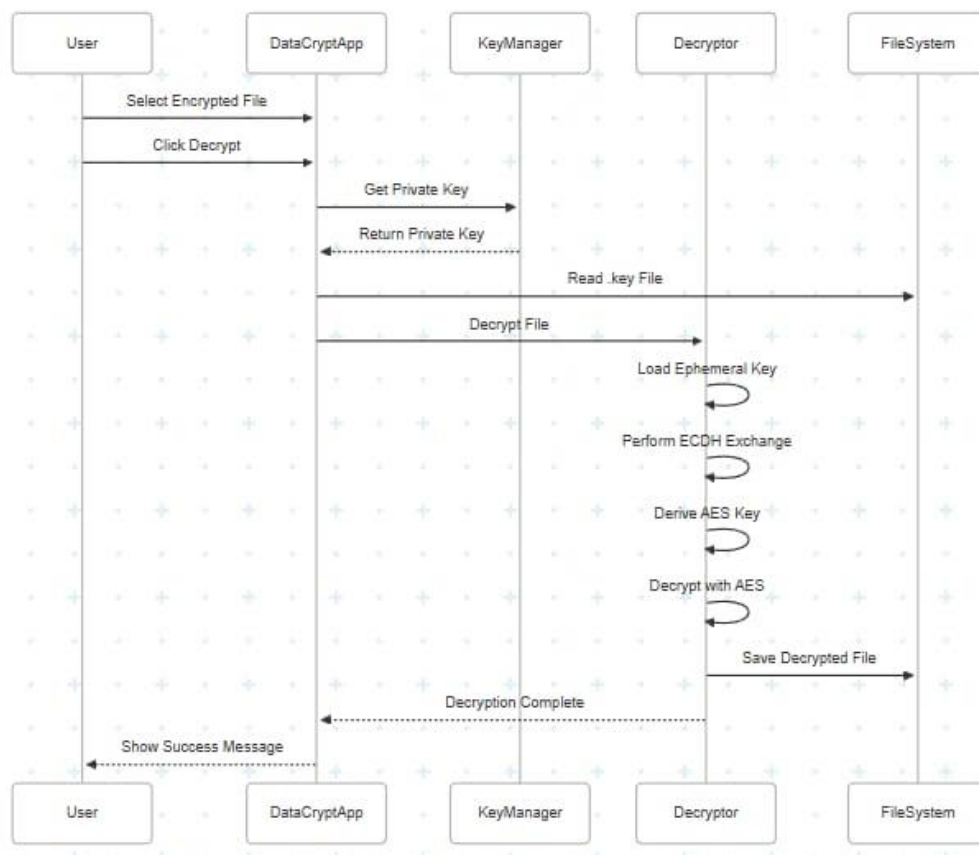


Fig 23: Decryption Flow Diagram

When the receiver gets the download link:

- They download both the encrypted file and the encrypted AES key.

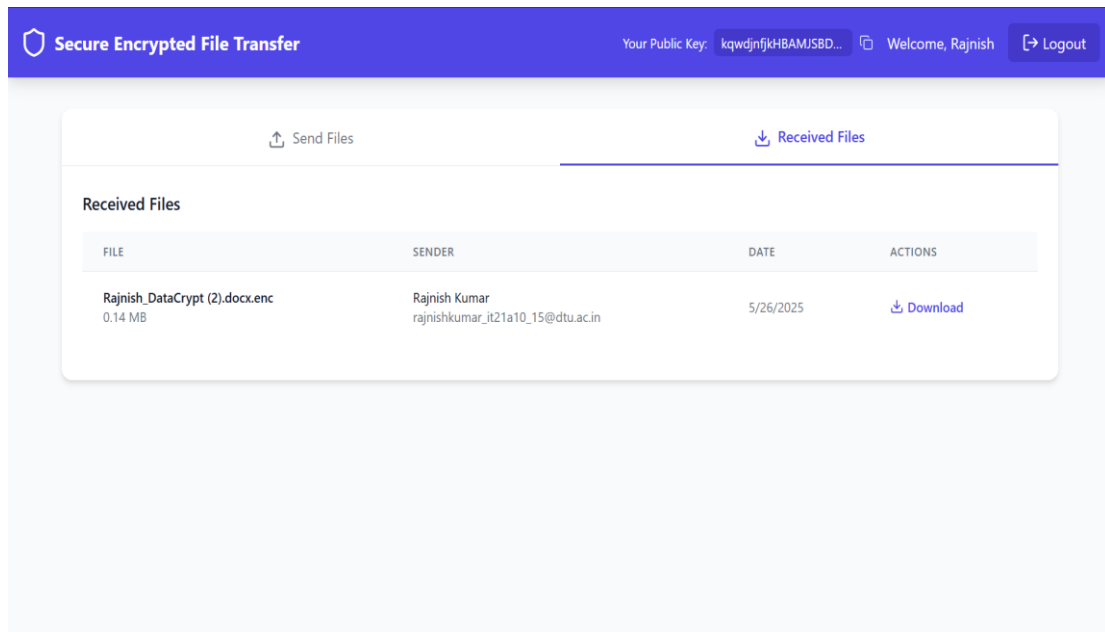


Fig 24: Encrypted file received by the receiver

- The local interface (on their system) uses the sender's public key and their own private key to decrypt the AES key using ECC.

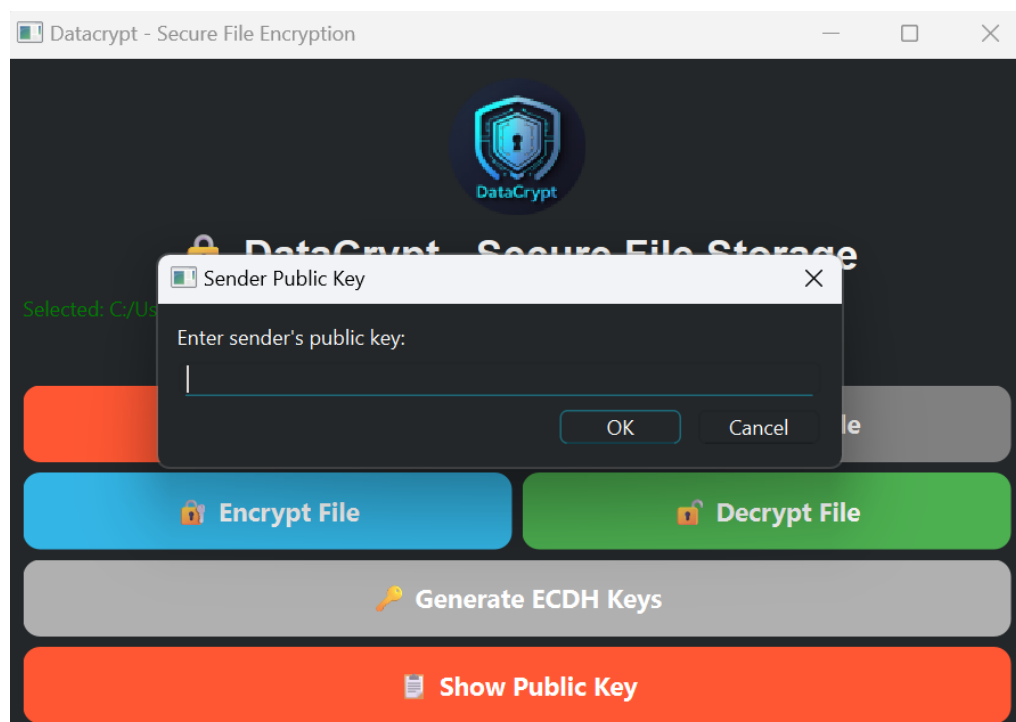


Fig 25: Decrypting the file using the public key of the sender



- With the decrypted AES key, they can then decrypt the original file.



Fig 26: File decryption successful

- At no point is the private key shared or exposed, ensuring that only the receiver can access the contents of the file.

## 2.8 SECURITY HIGHLIGHTS

**End-to-End Encryption:** Files are encrypted before upload and decrypted only after download, keeping cloud providers completely out of the loop.

**Key Isolation:** Users generate and store their private keys locally. Only public keys are shared via the backend.

**Mutual Trust:** Encryption requires both sender and receiver to exchange keys securely, eliminating the risk of impersonation or eavesdropping.

**Tamper-Proof Communication:** All communication with the server is protected using HTTPS and token-based authentication.

## CHAPTER 3

### RESULTS, DISCUSSION AND CONCLUSION

#### 3.1 RESULTS AND OBSERVATIONS

After the development and integration of both the local encryption interface and the remote backend system, we carried out several tests to evaluate the functionality and security of the Datacrypt system. The key observations are as follows:

**Key Generation:** The ECC key pair generation was fast and consistent, taking less than a second on most modern systems. Keys were securely stored locally with no server exposure.

**File Encryption and Decryption:**

- A 5MB file took about 1–2 seconds to encrypt using AES-256.
- The AES key encryption with ECC was near-instantaneous.
- Decryption using the correct keys worked smoothly and reliably, even after several file transfers.

**User Authentication and Key Retrieval:**

- The sign-up process with OTP verification ensured that only legitimate users could register.
- Public key retrieval through email search was quick, and the fallback email invitation mechanism worked as intended.

**File Sharing:**

- Encrypted files and AES keys uploaded to AWS S3 were accessible only via a unique, time-limited link.
- Download and decryption on the receiver's end was successful when correct keys were used. Incorrect keys resulted in expected decryption failures, validating security.

These results confirm that the system performs well for its intended use — secure file sharing without exposing private data to third parties.

**3.2 DISCUSSION**

While building and testing Datacrypt, we explored several technical and practical aspects of encryption-based file sharing. A few key points stood out:

**Ease of Use vs. Security:** One challenge was balancing security with user experience. For instance, forcing users to manage private keys locally adds security but may be unfamiliar to non-technical users. A simple and intuitive interface helped reduce this barrier.

**Trust Model:** The system ensures that only users with the correct private keys can access shared files. The trust is placed in public key distribution, which is managed via our secure backend — a critical part of ensuring end-to-end confidentiality.

**Scalability:** The backend is designed to support multiple users, but the current system is a prototype. A full-fledged application would require enhancements like key expiration, multiple key management, and more advanced access control.

**Error Handling:** We made sure to handle issues like missing keys, invalid emails, or unauthorized access attempts gracefully, giving proper feedback to users.

### 3.3 CONCLUSION

Datacrypt successfully demonstrates how client-side encryption can be combined with secure backend infrastructure to protect user data in cloud environments. The system ensures that:

- Files are encrypted before upload, keeping even the cloud provider from accessing them.
- Only the intended recipient can decrypt the file, using their private key.

- A simple, user-friendly interface enables even non-technical users to securely share files.
- All keys are managed securely, with the backend handling only public keys and user credentials.

Through this project, we've shown that privacy-first cloud storage is not only possible but also practical. The system is modular and can be extended or integrated into existing platforms for wider use.

### **3.4 FUTURE WORK**

While Datacrypt provides a solid foundation, there's still room to grow. Some possible future improvements include:

- Adding support for multiple file types and batch encryption.
- Implementing key rotation and automatic key backup mechanisms.
- Introducing mobile app support for cross-platform accessibility.
- Allowing temporary shared access with expiration controls and file activity logs.

- Enhancing the UI/UX to support drag-and-drop encryption and visual feedback.

## REFERENCES

- [1]. Qin Liu, Guojun Wang, and Jie Wu, "Efficient Sharing of Secure Cloud Storage Services," *10th IEEE International Conference on Computer and Information Technology (CIT 2010)*, 2010.
- [2]. Uma Somani, Kanika Lakhani, and Manish Mundra, "Implementing Digital Signature with RSA Encryption Algorithm to Enhance the Data Security of Cloud in Cloud Computing," *IEEE 1st International Conference on Parallel, Distributed, and Grid Computing (PDGC 2010)*, 2010.
- [3]. Ashutosh Kumar Dubey, Animesh Kumar Dubey, Mayank Namdev, and Shiv Shakti Shrivastava, "Cloud-User Security Based on RSA and MD5 Algorithm for Resource Attestation and Sharing in Java Environment," *2011*.
- [4]. Arthur Rahumed, Henry C. H. Chen, Yang Tang, Patrick P. C. Lee, and John C. S. Lui, "A Secure Cloud Backup System with Assured Deletion and Version Control," *2011 International Conference on Parallel Processing Workshops*.
- [5]. Eman M. Mohamed and Sherif El-Etriby, "Randomness Testing of Modern Encryption Techniques in Cloud Environment," *2008*.
- [6]. Komal, Naveen Kumar, Sandeep Kumar, Ashok Kumar Kashyap, and Ritesh Rana, "Performance Evaluation of Cryptography Algorithms: AES, DES, RSA, and ECC," *Journal of Emerging Technologies and Innovative Research (JETIR)*, Vol. 10, Issue 1, January 2023.
- [7]. Rohan Jathanna and Dhanamma Jagli, "Cloud Computing and Security Issues," *International Journal of Engineering Research and Applications*, Vol. 7, Issue 6, pp. 31-38, June 2017.



[8]. Ahmed Albugmi, Madini O. Alassafi, Robert Walters, and Gary Wills, "Data Security in Cloud Computing," 2016 Fifth International Conference on Future Generation Communication Technologies (FGCT), IEEE, August 2016.

[9]. S. Nagendrudu, P. Ishaq Alam, B. Srinivasulu, S. Sai Nanda Kishore, U. Giri Babu, and S. Shahid Basha, "Security Using Elliptic Curve Cryptography (ECC) in Cloud," International Journal of Creative Research Thoughts (IJCRT), Vol. 12, April 2024.

[10]. Sarmad Mahmood Ahmed and Baban Ahmed Mahmood, "Cloud Computing Security: Assured Deletion," Informatica, Vol. 48, pp. 485-496, 2024.

[11]. Wejdan Alsuwat and Hatim Alsuwat, "A Survey on Cloud Storage System Security via Encryption Mechanisms," International Journal of Computer Science and Network Security (IJCSNS), Vol. 22, No. 6, June 2022.

[12]. Yuke Liu, Junwei Zhang, and Qi Gao, "A Blockchain-Based Secure Cloud Files Sharing Scheme with Fine-Grained Access Control," NeurIPS Workshop, January 2018.

[13]. "Hybrid AES-Modified ECC Algorithm for Improved Data Security over Cloud Storage," Journal of Advanced Research in Applied Sciences and Engineering Technology, Vol. 32, No. 1, pp. 46-56, August 2023.

[14]. "A Hybrid Approach Using AES-RSA Encryption for Cloud Data Security," International Journal of Intelligent Systems and Applications in Engineering, Vol. 12, No. 21s, pp. 62-69, March 2024.

[15]. Arpitha K and Sethupathi T.R, "Cloud Data Encryption (The Review Paper)," International Journal of Advance Research, Ideas and Innovations in Technology (IJARIIE), Vol. 10, Issue 4, 2024.

[16]. Arunima Raj et al., "A Hybrid Approach Of ECC (Elliptic Curve Cryptography) and AES (Advanced Encryption Standard) Cryptography in a File Transfer System Combined with Cloud Computing," International Journal of Production, Research, Engineering, Management and Sciences (IJPREMS), January 2024.

[17]. "A Survey on Public-Key Based Searchable Encryption for Secure Big Data Management on Cloud," SSRN Electronic Journal, 34 pages, March 2023.

[18]. "Hybrid Approach to Cloud Storage Security Using ECC-AES Encryption and Key Management Techniques," International Journal of Engineering Trends and Technology (IJETT), Vol. 72, Issue 4, April 2024.

[19]. "Enhancement of Big Data Security in Cloud Computing Using RSA Algorithm," University of West London Repository, 2024.