PUBLIC – KEY AUTHENTICATED ENCRYPTION WITH CIPHERTEXT UPDATE AND KEYWORD SEARCH

A PROJECT REPORT

Submitted by

AKSHAYA V (422420104004)

KAVIYA V (422420104702)

SINDHUJA R (422420104032)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING TINDIVANAM

ANNA UNIVERSITY: CHENNAI 600025

MAY 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**PUBLIC-KEY AUTHENTICATED ENCRYPTION WITH CIPHERTEXT UPDATE AND KEYWORD SEARCH"** is the bonafide work of **"AKSHAYA V(422420104004), KAVIYA V (422420104702), SINDHUJA R (422420104032)"** who carried out the project work under my supervision.

SIGNATURE

Dr. L. Jegatha Deborah., M.E, Ph.D.,

HEAD OF THE DEPARTMENT,

Assistant Professor,

Department of CSE,

University College of Engineering

Tindivanam,

Melpakkam – 604 001.

SIGNATURE

Ms. P. Sathya, M.Tech.,

TEACHING FACULTY,

Department of CSE,

University College of Engineering

Tindivanam,

Melpakkam – 604 001.

Submitted for the University Examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

At the outset we like to express our gratitude to the God almighty. We express our thanks to our respected Dean **Dr. P. THAMIZHAZHAGAN, M.E., Ph.D.,** for his constant inspiration and encouragement throughout the project.

We pay our grateful acknowledgement and extend our sincere gratitude to our Head of the Department **Dr. L. JEGATHA DEBORAH, M.E., Ph.D.,** for extending facilities of department towards our project and for unstinting support.

We express our thanks to our guide, **Ms. P. SATHYA, M.Tech.,** for guiding us every phase of the project. We appreciate her thoroughness, tolerance and ability to share her Knowledge with us. We thank her for being easily approachable and quite thoughtful. Without her immense support through every step of the way, we could never have it to this extent.

We extend our thanks to the project coordinator **Ms. P. SATHYA, M.Tech.,** for her continual support during the entire project schedule.

We thank all our teaching and non-teaching faculty members of the department and also our fellow friends for helping us in providing valuable suggestions and timely ideas for the successful completion of the project. Last but not least we extend our thanks to our family member who have been a great source of inspiration and strength to us during the course of this project work. We sincerely thank all of them.

ABSTRACT

In hospitals, electronic medical records (EMRs) are crucial for improving patient care and reducing errors. However, managing these records can be challenging due to the increasing volume of data which bring heavy storage burden to the hospitals. Cloud based solutions offer a way to store and manage EMRs efficiently which can save hospitals local storage, but many hospitals are hesitant to use them due to concerns about patient privacy. To address this issue, we propose a new method called PAUKS, which stands for Public-Key Authenticated Encryption with Ciphertext Update and Keyword Search. PAUKS enables hospitals to securely store and share EMRs in the cloud while protecting patient privacy. With PAUKS, hospitals can encrypt EMRs and still search them without needing to decrypt the data. This means that researchers and healthcare providers can access necessary information without compromising patient privacy. PAUKS also defends against insider attacks, ensuring that sensitive data remains safe. Compared to existing methods like PAEKS, PAUKS offers lower computational and communication overheads, The required number of trapdoors per query is constant in PAUKS scheme, instead of linearly expanding as the number of senders increases in PEAKS. Maintaining a constant number of trapdoors per query in PAUKS, making it more efficient as the number of users increases. Furthermore, PAUKS supports the creation of an inverted index, which speeds up the search process. Our experiments show that PAUKS performs similarly in terms of running costs compared to other methods but offers better query efficiency, particularly after updating EMR records. Overall, PAUKS provides a secure and efficient solution for hospitals to manage and share electronic medical records in the cloud.

KEYWORDS: Searchable encryption, keyword guessing attacks, electronic medical record, light overhead, fast search.

திட்டப் பணிச்சுருக்கம்

மருத்துவமனைகளில், நோயாளியின் பராமரிப்பை மேம்படுத்துவதற்கும் பிழைகளைக் குறைப்பதற்கும் மின்னணு மருத்துவப் பதிவுகள் (EMRs) முக்கியமானவை. இருப்பினும், அதிகரித்து வரும் தரவுகளின் அளவு காரணமாக இந்தப் பதிவுகளை நிர்வகிப்பது சவாலானதாக இருக்கலாம். கிளவுட்-அடிப்படையிலான தீர்வுகள் EMRகளை திறம்படச் சேமித்து நிர்வகிப்பதற்கான வழியை வழங்குகின்றன, ஆனால் நோயாளிகளின் தனியுரிமை குறித்த கவலைகள் காரணமாக பல மருத்துவமனைகள் அவற்றைப் பயன்படுத்தத் தயங்குகின்றன. இந்தச் சிக்கலைத் தீர்க்க, PAUKS எனப்படும் புதிய முறையை நாங்கள் முன்மொழிகிறோம். சைபர்டெக்ஸ்ட் புதுப்பிப்பு மற்றும் முக்கிய வார்த்தை தேடலுடன். PAUKS ஆனது, நோயாளியின் தனியுரிமையைப் பாதுகாக்கும் அதே வேளையில், EMRகளை மேகக்கணியில் பாதுகாப்பாகச் சேமித்து, பகிர்ந்துகொள்ள மருத்துவமனைகளுக்கு உதவுகிறது. PAUKS மூலம், மருத்துவமனைகள் EMRகளை என்க்ரிப்ட் செய்து தரவை மறைகுறியாக்கத் தேடலாம். இதன் பொருள் ஆராய்ச்சியாளர்கள் மற்றும் சுகாதார வழங்குநர்கள் நோயாளியின் தனியுரிமையை சமரசம் செய்யாமல் தேவையான தகவல்களை அணுக முடியும். தற்போதுள்ள PAEKS போன்ற முறைகளுடன் ஒப்பிடும்போது, PAUKS குறைந்த கணக்கீடு மற்றும் தகவல்தொடர்பு மேல்நிலைகளை வழங்குகிறது.

முக்கிய வார்த்தைகள்: தேடக்கூடிய குறியாக்கம், முக்கிய வார்த்தைகளை யூகிக்கும் தாக்குதல்கள், மின்னணு மருத்துவ பதிவு, லேசான மேல்நிலை, வேகமான தேடல்.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S.NO	ABBREVIATIONS	DEFINITIONS
1.	JPBC	Java Pairing Based Cryptography Library
2.	PBC	Pairing Based Cryptography
3.	JVM	Java Virtual Machine
4.	EMR	Electronic Medical Record
5.	PAUKS	Public key Authenticated Encryption with ciphertext Update and Keyword Search
6.	PAEKS	Public Authenticated Encryption with Keyword Search
7.	PP	Public Parameter
8.	UML	Unified Modeling Language

CHAPTER 1

INTRODUCTION

Electronic medical records (EMRs) are crucial components of hospital information systems, managing patient data like demographics, medical history, and diagnoses. They facilitate communication between patients, families, and researchers, and improve service efficiency while reducing staff workload and medical risks. However, challenges like limited storage and security issues have emerged. Insufficient storage space arises due to the accumulation of records, especially in renowned hospitals. Various schemes like Dual-server based PEKS and Secure channel-free PEKS [20] have been proposed, but they either incur high costs or lack full security against internal threats.[4] Public key authenticated encryption with keyword search (PAEKS) [24] is a promising solution, but it faces scalability issues, with trapdoor generation linearly increasing with the number of users. Moreover, ensuring information security is critical, as network threats could lead to data breaches, posing serious privacy risks to patients. To address these concerns, leveraging advanced cloud computing and storage technologies is proposed. Hospitals can encrypt EMRs and outsource them to the cloud under the supervision of a data administrator. However, using traditional public key encryption may not suffice for frequent queries from medical staff and researchers. Public key encryption with keyword search (PEKS), initially introduced by Boneh et al., seems suitable as it allows users to generate trapdoors for specific keywords, facilitating efficient data retrieval. Nonetheless, some PEKS schemes are vulnerable to offline keyword guessing attacks (KGAs), where adversaries can exploit small keyword spaces to infer queried keywords.[26]

1.1 PUBLIC KEY CRYPTOGRAPHY

In public key cryptography, since the same key is used for both encryption and decryption, the two parties must take great care in exchanging the key so that an eavesdropper does not obtain it. The problem of key exchange is one of the most

difficult in symmetric cryptosystems. Symmetric cryptosystem creates problems of trust, as the same key is held by both users in any sender/receiver pair using the PPT.

These requirements are impossible to achieve in classical one-key cryptosystems. The breakthrough came in 1976 with the invention of public key cryptography by Stanford University Professor Martin Hellman and graduate student W. Diffie. By definition, a public key cryptosystem has the property that the enciphering function f : P → C is easy to compute once the enciphering key, KE, is known, but it is very hard in practice to compute the inverse function f −1: C → P, that is, the function f is not invertible (without some additional information - the deciphering key KD). Such a function f is called a trapdoor function.

1.2 BILINEAR MAPPING

The bilinear mapping is used to manage cryptographic keys, perform secure key exchanges, enforce access control policies, and support secure communication between different entities in the system. Bilinear maps enable the generation, manipulation, and verification of cryptographic keys that are essential for secure data sharing and search operations.

Bilinear maps offer scalability and flexibility in designing cryptographic protocols for multi-user scenarios. By leveraging the properties of bilinear pairings, the scheme can support multiple users, enable efficient search operations on encrypted data, and ensure traceability and revocation of malicious users while preserving the forward security of the system.

Function bilinear map (G1, G2, GT, g, h):

r1 = random number in Zq

r2 = random number in Zq

GT element = e(g^{r1}, h^{r2})

return GT element

1.3 KEY GENERATION

Key generation in public key authenticated encryption with constant trapdoor generation and fast search is a process that efficiently creates public-private key pairs for encryption. It also involves generating constant trapdoors to enable rapid key retrieval for search operations. Authentication of public keys ensures their integrity and prevents unauthorized access. Data encryption relies on the public key, while decryption utilizes the corresponding private key for security. Additionally, constant trapdoors facilitate swift search operations. Secure storage of keys is crucial to prevent unauthorized access or compromise. Overall, this process ensures the confidentiality, integrity, and efficiency of cryptographic operations in a public key authenticated encryption system.

1.4 CONSTANT TRAPDOOR GENERATION

Trapdoor is a secret key that allows the user to search for specific keyword in the encrypted message. Trapdoor is generated based on the keyword and the public key and is specific to that particulate keyword. Constant trapdoor generation involves creating fixed-size cryptographic objects used for efficient search operations, typically in cryptographic systems like searchable encryption or public key authenticated encryption. These trapdoors are generated deterministically based on specific parameters and keys. They enable fast retrieval of encrypted data corresponding to specific search queries without compromising security. Constant trapdoors are designed to be computationally efficient to generate and utilize. They ensure that search operations remain fast and scalable even as the dataset grows. Trapdoor generation algorithms typically leverage cryptographic primitives and techniques to ensure security and reliability. Overall, constant trapdoor generation is crucial for enabling efficient and secure search functionality in various cryptographic applications.

CHAPTER 2

LITERATURE SURVEY

D. Boneh et al. proposed Public key encryption with keyword search, in this cryptographic technique allows someone to search through encrypted documents for a particular keyword without the need to decrypt them. Essentially, PEKS enables a user to generate a special kind of token, called a "trapdoor", for a keyword and give it to the server. The server can then use this trapdoor to identify encrypted documents that contain the keyword without actually learning the keyword or the contents of the documents.

PEKS allows for searching over encrypted data, which is crucial in scenarios where privacy is a concern, such as cloud storage or secure communication. PEKS algorithms are designed to be efficient, allowing for practical implementations even with large datasets and numerous users. The security of PEKS schemes relies heavily on the security of the underlying cryptographic primitives. If there are vulnerabilities in these primitives or in the implementation of the scheme itself, it could compromise the security of the encrypted data. PEKS schemes can be complex to implement and require careful consideration of various security and efficiency trade-offs. This complexity can make them more prone to implementation errors and vulnerabilities. [24]

H.Li et al. proposed Designated server identity-based authenticated encryption with keyword search for encrypted emails allows a user to send an encrypted email using the recipient's identity (like their email address) as the public key. The email is not only encrypted but also contains a secure proof of its origin. When the recipient or an authorized third party wants to search through these encrypted emails for certain keywords, they can do so using a "trapdoor" (a special kind of search token) that preserves the privacy of the email content. The designated server assists in this search process, although it does not have the ability to decrypt the emails or know the search keywords. This ensures that email

communication is confidential, authentic, and searchable, all while being convenient for users in terms of key management and searchability.

Identity-based encryption simplifies key management by using users' identities as public keys, eliminating the need for a public key infrastructure (PKI) or certificate authorities. The scheme provides end-to-end encryption and authentication for emails, ensuring privacy and integrity of communication. The designated server represents a single point of failure and potential target for attacks, requiring robust security measures and redundancy to mitigate risks. The security of the scheme relies on trusting the designated server and the cryptographic primitives used, which may not always hold in practice. [15]

Y.Lu et al. proposed Secure channel free certificate-based searchable encryption withstanding outside and inside keyword guessing attacks. It is a sophisticated cryptographic system designed to secure data storage and retrieval in a manner that is both user-friendly and highly secure against various types of cyber threats. In such a system, data is encrypted before being stored, and it can be searched using encrypted search queries, known as trapdoors, without the need for a secure communication channel.

It focus on withstanding both outside and inside keyword guessing attacks suggests it proposes methods to significantly enhance the security of searchable encryption schemes. By eliminating the need for a secure channel for certificate distribution, the system can be more easily implemented and scaled across different platforms and applications, reducing the operational complexity and potentially lowering costs. This could provide a more robust defense against various types of cybersecurity threats. With added security features and the certificate-based mechanism, there might be an increase in computational complexity, which could affect the system's efficiency and performance, especially in real-time applications. Any novel encryption scheme or cryptographic approach might introduce

unforeseen vulnerabilities or be susceptible to future cryptographic advances that could undermine its security premises. [4]

R. Chen et al. proposed Dual-server public-key encryption with keyword search for secure cloud storage. It is a cryptographic method designed to enhance the security of data storage, particularly in cloud environments. This approach divides responsibilities between two separate servers, each playing a unique role in the encryption and keyword search process. In a traditional public-key system, data is encrypted with the public key of a receiver, ensuring that only the holder of the corresponding private key can decrypt and read the data.

The introduction of a dual-server architecture aims to bolster security against internal threats, particularly inside keyword guessing attacks, by distributing trust and operational roles across two servers. Utilizing linear and homomorphic SPHFs contributes to the robustness of the encryption scheme, providing a more secure framework for cloud storage services. The dual-server model may introduce additional complexity in system deployment and management, possibly affecting scalability and ease of integration with existing cloud storage infrastructures. The added security measures, while beneficial for privacy and security, might incur performance penalties, affecting the efficiency of keyword search operations and overall system responsiveness. [20]

L. Xu et al. proposed Delegatable searchable encryption with specified keywords for EHR systems. It is a security protocol that allows for the storage of encrypted EHRs in such a way that they can be searched for specific keywords by authorized parties. This type of system is particularly relevant in healthcare settings, where patient data is sensitive and privacy must be rigorously maintained, yet the information needs to be readily accessible to authorized healthcare providers. In such a system, EHRs are encrypted before they are stored. However, despite being encrypted, these records can still be searched through the use of a secure searchable encryption technique.

DSE techniques typically provide strong security guarantees, such as confidentiality of the data and the search queries, as well as resistance against various attacks.DSE techniques often introduce computational overhead due to the need for cryptographic operations, which can impact search performance, especially in large-scale systems. Implementing DSE schemes correctly can be complex and may require expertise in both cryptography and system design, potentially making them challenging to deploy and maintain. [11]

R. Behnia et al. proposed limited proxy re-encryption with keyword search for data access control in cloud computing offers a robust framework for ensuring data security and efficient access control. While it brings significant advantages in terms of security and functional capabilities, the complexity, management requirements, and potential performance impacts must be carefully considered. Organizations looking to implement this technology should weigh its benefits against its challenges and consider their specific security needs, data access requirements, and technical capabilities. [9]

In this setup, a data owner can securely outsource the storage of their encrypted data to a cloud service, which acts as the designated server. By enabling keyword search capabilities in a proxy re-encryption scheme, the paper likely proposes a method to streamline the sharing of encrypted data. This can significantly improve the efficiency of accessing relevant encrypted data in large datasets. The combination of designated server proxy re-encryption with keyword search could offer robust security measures for big data storage and access, safeguarding against unauthorized data breaches and ensuring data privacy. [19]

The implementation of efficient proxy re-encryption systems, especially those designed for big data environments, can be complex. This complexity might pose challenges in real-world applications, from deployment to maintenance. While enhancing security and adding keyword search functionality, there might be

trade-offs in terms of system performance, such as increased latency or computational overhead, especially when dealing with very large datasets.

Z.-Y.Liu et al. proposed an identity-based searchable encryption framework supported by an identity-certifying authority, aimed at enhancing data security in cloud systems. The framework integrates identity-based encryption (IBE) with searchable encryption (SE), which could enhance security by simplifying key management and allowing encrypted data to be searched by authorized users. The framework likely supports scalability in cloud environments, which is essential as the volume of data and number of users can grow rapidly. The integration of IBE with a searchable encryption mechanism can be complex to implement correctly without introducing vulnerabilities. [22]

The additional layers of security, especially in handling searchable encryption queries and identity verification, could introduce latency and computational overhead, which might affect the system's performance. The dependence on an identity-certifying authority introduces a potential single point of failure. If the authority's system is compromised, the security of the entire system can be jeopardized. Depending on the jurisdiction, the involvement of an identity-certifying authority and the way data is handled and accessed can raise regulatory and privacy concerns, particularly related to data sovereignty and access controls. An efficient public-key searchable encryption scheme designed to withstand inside keyword guessing attacks, where adversaries attempt to deduce search queries from encrypted data. Despite its security advantages, the scheme may incur computational overhead due to complex cryptographic operations and require careful key management. Nonetheless, it represents a significant advancement in public-key searchable encryption, offering a balance between security and efficiency crucial for practical deployment in various scenarios. Public-key cryptography often involves more complex key management compared to symmetric-key schemes, potentially leading to challenges in key distribution, storage, and revocation. [18]

CHAPTER 3

PROBLEM STATEMENT

- Number of trapdoor per query is relatively high which causes inefficient searching and communication overhead.
- Existing public key encryption schemes typically do not allow for searching encrypted data that may require revealing sensitive information to the search server.

3.1 OBJECTIVES

- To generate a constant size trapdoor which reduces communication overhead, storage cost and improves efficient searching.
- To allow authorized users to search over encrypted data using specific keywords without decrypting the data.

3.2 SCOPE

The system can be employed in cloud storage environments where users want to store their data securely and perform keyword searches on the encrypted data without compromising its confidentiality. Financial institutions dealing with sensitive financial data could use this system to securely store and retrieve information related to transactions, account details, or other financial records. In healthcare, where patient records need to be stored securely, the system can be applied to ensure the confidentiality of patient data. Medical professionals could perform keyword searches to retrieve relevant information while maintaining data privacy. Individuals concerned about the privacy of their personal information stored in various applications could use the system to encrypt and search for specific details without exposing the entire dataset.

CHAPTER 4

REQUIREMENT ANALYSIS

Requirements are the basic needs or constraints which are required to develop a system. Requirements are broadly classified as user requirements and the system requirements. These requirements can be collected while designing the system. The two major classifications of requirements are software and the hardware requirements.

1.Functional Requirements

2.Non-Functional Requirements

4.1 FUNCTIONAL REQUIREMENTS

The requirements specification is a technical description of the software's requirements. It is the initial phase in the process of requirements analysis. It contains a list of a software system's needs, including functional, performance, and security requirements. User, and administrative usage scenarios are also included in the requirements. The goal of a software requirements specification is to give a complete overview of the software project, its parameters, and its objectives. This section outlines the project's intended audience, as well as the project's user interface, hardware, and software needs. It establishes the client's, team's, and audience's perceptions of the project's functionality.

4.1.1 HARDWARE REQUIREMENTS

- Disk space : 64GB SSD
- RAM : 8GB
- Processor : Intel® Core™ i5-8250U CPU @1.60GHz 1.80GHz

4.1.2 SOFTWARE REQUIREMENTS

- IDE : Intellij IDEA Community
- OS : Windows 10 Home

- Language : Java Software
- Library : JPBC

4.2 NON-FUNCTIONAL REQUIREMENTS

Describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Non-Functional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands) or accuracy (i.e., how precise are the systems numerical answers).

4.2.1 PERFORMANCE REQUIREMENTS

- Response Time: The system should respond to user queries within 2 seconds under normal load conditions.
- Throughput: The system should be capable of handling a minimum of 100 concurrent user requests without significant degradation in performance.
- Scalability: The system should be designed to scale horizontally to accommodate an increase in the volume of EMRs and user queries over time.

4.2.2 USABILITY REQUIREMENTS

- User Interface: The system should have an intuitive and user-friendly interface that allows healthcare professionals to easily search for and retrieve EMRs.
- Documentation: Comprehensive documentation, including user manuals and troubleshooting guides, should be provided to assist users in understanding and navigating the system.

4.2.3 SECURITY REQUIREMENTS

- Data Encryption: All EMRs stored in the system should be encrypted using strong cryptographic algorithms to prevent unauthorized access.
- Intrusion detection and prevention: The system should be equipped with intrusion detection and prevention mechanisms to detect and respond to unauthorized attempts to access or tamper with EMRs.

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

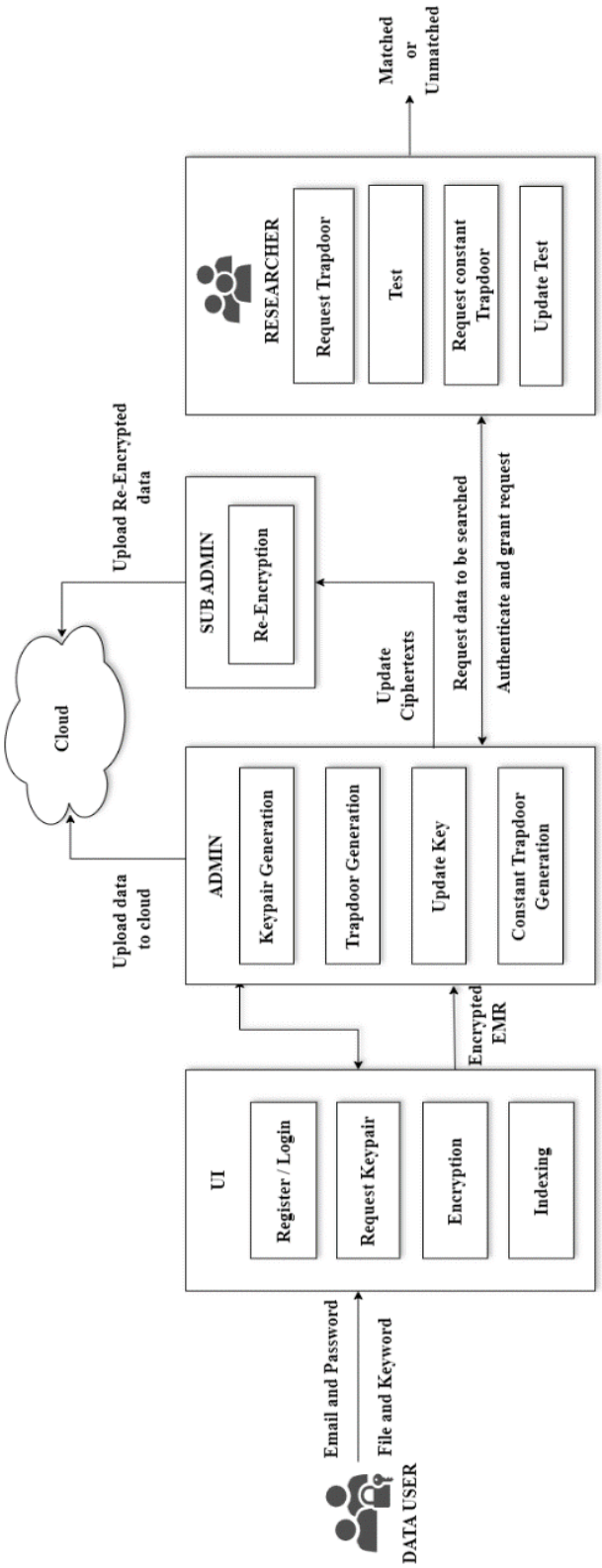


Fig. 5.1 System Architecture

5.2 MODULE DESCRIPTION

5.2.1. DATA USER

Data user verifies user credentials, manages sessions, and incorporates security measures to protect against threats. Features include a user-friendly interface for login, options for password recovery/reset, rate limiting to prevent brute force attacks, and optional multi-factor authentication for enhanced security. It supports secure data transmission and adheres to privacy regulations.

5.2.2. KEY GENERATION

Public key authentication and encryption involve a process where systems can securely authenticate their identity or encrypt messages using a pair of cryptographic keys. The process ensures confidentiality, integrity, and authenticity of communications. A public key that is openly distributed and used for encryption or verification. It can encrypt data or verify signatures but cannot decrypt data encrypted with it. A private key kept confidential by the owner and used for decryption or signing. It can decrypt data encrypted with its corresponding public key.

KEY GENERATION ALGORITHM

Input: File and Keyword

Output: Generate the secret Key and public key for the user

1.Randomly choose the secret key x1, x2, x3, x4 and y.

2.Generate the public key for sender and receiver,

$-pK_{R_1}=g^{x_1}, pK_{R_2}=g^{x_2}, pK_{R_3}=g^{x_3}.$ (1)

$-sK_{R_1} = x_1,\ sK_{R_2} = x_2,\ sK_{R_3} = x_3,\ sK_{R_4} = x_4.$ (2)

$-pK_S=g^y$. (3)

sK_S = y. (4)

5.2.3. ENCRYPTION

Encryption process uses sender's secret key, receiver's public key and keyword for encryption and produce ciphertext as output. Encryption is involved in the public key authentication process primarily for security reasons, enhancing the confidentiality, integrity, and authenticity of the communication between parties. Encryption is a security process that transforms plaintext, or readable data, into ciphertext, an unreadable form, using an encryption algorithm and an encryption key. This cryptographic technique ensures that sensitive information remains confidential and secure by making it inaccessible to unauthorized users or systems. The encryption process involves applying a mathematical algorithm to the plaintext along with an encryption key, resulting in ciphertext that appears random and unintelligible without the corresponding decryption key. To decrypt and access the original plaintext, the ciphertext is reversed using the decryption key.

ENCRYPTION ALGORITHM

Input: pk_R, sk_S

Output: Ciphertext

1. Generate four secure Hash functions

H_1:G × {0,1}*→ Z_p, H_2: G →Z_p, H_3: G→Z_p, H_4:{0,1}*→G

2. Encrypts the given text.

- $C_1 = \left(pK_{R_2}^{H1\left(pK_{R_1}^{sK_S}, w\right)} . pK_{R_3} \right)^{r1}$ (5)
- $C_2 = g^{r1}$ (6)
- $C_3 = \left(pK_{R_2}^{H2\left(pK_{R_1}^{sK_S}\right)} . pK_{R_3} \right)^{r2} . g^{H3\left(pK_{R_1}^{sK_S}\right).r1}$ (7)
- $C_4 = H4(w)^{r2}$ (8)
- $C_5 = H(C_1, C_2, C_3, C_4)^{r1}$ (9)

5.2.4. INDEXING

Indexing in the context of searchable encryption, such as the Public-Key Authenticated Encryption with Keyword Search (PAUKS) scheme, involves creating data structures that map keywords to encrypted data without the need for decryption. This indexing mechanism accelerates the search process by allowing users to quickly retrieve relevant encrypted information based on specific keywords. By utilizing indexing, users can efficiently search and access encrypted data without compromising data security and privacy.

INSERT INDEX ALGORITHM

Input: Public parameters, C, H

Output: H

1. Set the ptr = H and iseql=0

2. Compute while loop for the equality test and set the if condition,

    2.1. Iseql = equality Test (PP, ptr.LabelCipher, C) is true, return H.

    2.2. Else move to ptr.nextnode

3. Insert the C into ptr.Pointer.

4. Stored in cloud.

5.2.5. TRAPDOOR GENERATION

The Trapdoor Generation Module generates trapdoors for keyword search queries without revealing the actual content of the encrypted data. It allows users, such as administrators, senders, and researchers, to generate trapdoors for specific keywords to search for relevant information. The module ensures constant trapdoor generation to support efficient and secure keyword searches while maintaining data privacy.

5.2.6 TEST

The test functionality plays a pivotal role in enabling users to efficiently retrieve relevant encrypted information based on specific keywords while maintaining data security and privacy. Through the indexing mechanism established during data insertion, which maps keywords to encrypted data without the need for decryption, the search process becomes expedited. Users can input search queries consisting of keywords, which are then matched against the indexed data structures. This matching process allows for rapid identification of encrypted data associated with the specified keywords, without compromising the confidentiality of the underlying information. By leveraging cryptographic techniques and efficient data structures, the search functionality ensures that users can access encrypted data securely and conveniently, enhancing the overall usability and effectiveness of the PAUKS system.

5.2.7. UPDATE KEY

The Update Key Module facilitates the transformation of ciphertexts from various senders into a standardized format to streamline the search process. It ensures that ciphertexts can be securely updated and integrated into the system for efficient keyword-based searches. The Update Key Module ensures that ciphertexts from different senders are converted into a unified format, enabling consistent and standardized processing during keyword searches. This transformation simplifies the search process and enhances the efficiency of retrieving relevant encrypted data.

5.2.8 RE-ENCRYPTION

Re-Encryption within a Public-Key Authenticated Encryption with Keyword Search (PEKS) system plays a critical role in enhancing the system's flexibility and security by allowing encrypted data to be securely transformed from one cryptographic key to another without the need to decrypt it first. This module is particularly useful in multi-user environments, cloud storage systems, and other

contexts where data might need to be accessed by different users or transferred securely between different domains. This process, known as proxy re-encryption, allows a proxy server to perform re-encryption operations without gaining access to the plaintext contents of the data, thus maintaining confidentiality throughout the transfer process.

5.2.9 CONSTANT TRAPDOOR GENERATION

Constant trapdoor generation in Public-Key Authenticated Encryption with Keyword Search (PAUKS) is a vital aspect of maintaining the system's efficiency and security. It ensures that trapdoors, which are cryptographic tokens used to search for specific keywords without exposing the encrypted data, are continually generated and available. This constant generation of trapdoors enables users to perform keyword searches swiftly and securely, without compromising the confidentiality of the underlying encrypted information. It's akin to having a reliable mechanism always ready to facilitate searches while safeguarding sensitive data from unauthorized access.

5.2.10 UPDATE TEST

Update test is a process within a cryptographic system that focuses on modifying and improving the search functionality without altering the underlying data structure or compromising security. Instead of changing the fundamental architecture, update search aims to refine search algorithms, enhance indexing mechanisms, or optimize keyword retrieval methods. This ensures that the system remains efficient and effective in retrieving relevant information while maintaining data privacy and integrity. Update search activities might involve fine-tuning search algorithms to improve search speed, updating indexing mechanisms to handle larger datasets efficiently, or implementing new techniques to enhance search accuracy. By continuously refining and updating the search functionality, the system can adapt to evolving user needs and technological advancements without compromising its security or reliability.

TRAPDOOR GENERATION AND TESTING ALGORITHM

Input: C, Trapdoors, keyword

Output: Determine the Boolean value where the pairing of ciphertext and trapdoor are equals or not.

1.Generate the Trapdoor Tw1 and Tw2 ,

$$\mathrm{T}_{\mathrm{w},1} = g^{\frac{r3}{sK_{R_3}.H_1\left(pK_S^{sK_{R_1}},w'\right)+sK_{R_3}}}, \mathrm{T}_{\mathrm{w},2} = g^{r3} \quad (10)$$

2.If $\hat{e}(C_1, T_{w,1}) = = \hat{e}(C_2, T_{w,2})$ hold the Boolean value 1.

3. Otherwise, Return 0;

4. End If

5. Updated Key uks1 and uks2,

$$\mathrm{Uks1} = H_3\left(pK_S^{sK_{R_1}}\right), \mathrm{Uks2} = \frac{sK_{R_4}}{sK_{R_2}.H_2\left(pK_S^{sK_{R_1}}\right)+sK_{R_3}} \quad (11)$$

6. Re- Encrypted the Ciphertext $C_6 = g^{r_2.sk_{R_1}}$ and return $\hat{\mathrm{C}}$= (C, C_6).

7. Generate the Constant Trapdoor by using PP, sk_R, w' as input and output is

$\hat{\mathrm{T}}_{\mathrm{w},1}$ and $\hat{\mathrm{T}}_{\mathrm{w},2}$

$$\hat{\mathrm{T}}_{\mathrm{w},1} = g^{sK_{R_4}.r}, \hat{\mathrm{T}}_{\mathrm{w},2} = H_4\,.(w')^r \quad (12)$$

8. Now, Test the updated ciphertext and Trapdoor pair , (C_4, $T_{w,1}$) and (C_6, $T_{w,2}$).

9. If $(\mathrm{C}_4, \hat{\mathrm{T}}_{\mathrm{w},1}) == (\mathrm{C}_6, \hat{\mathrm{T}}_{\mathrm{w},2})$ hold the Boolean value 1. Otherwise Return 0;

CHAPTER 6

UML DIAGRAMS

6.1. USE CASE DIAGRAM

Users and Researchers are primarily engaged in generating trapdoors for specific keywords and using these trapdoors to search encrypted data securely, with Researchers potentially employing advanced search functionalities. Admins are responsible for system management tasks, managing user profiles and permissions, and conducting system audits to ensure security and compliance. Sub-admins assist in user management but with more limited or specialized responsibilities, as determined by the main Admin.

Fig. 6.1 Use Case Diagram

6.2. CLASS DIAGRAM

The "Register" and "Login" classes manage user authentication and registration processes, while "Key Generation" handles cryptographic key generation. "Encryption" and "Re-encryption" classes implement encryption algorithms for data security, with "Update Key" managing key updates. ."Trapdoor" and "Constant Trapdoor" classes facilitate efficient keyword search operations, "Test" and "Update Test" classes search for the file. These classes interact to provide a comprehensive system ensuring secure encryption, authentication, efficient keyword search, and key management.

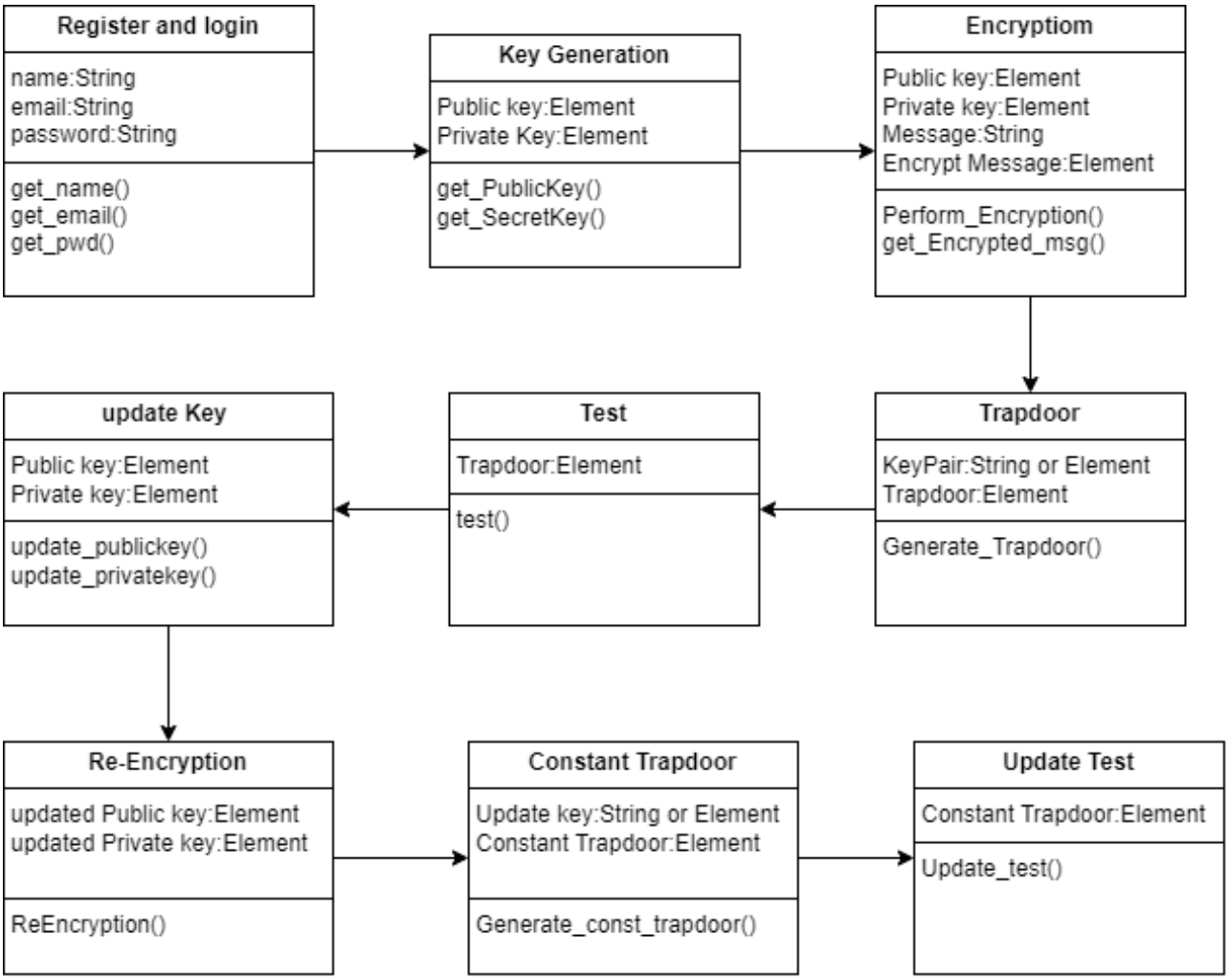


Fig. 6.2 Class Diagram

6.3. SEQUENCE DIAGRAM

The process begins with a user initiating a request, which is then received by the admin. The admin, assisted by the sub-admin, manages the request and interacts with the cloud component where encrypted data is stored. The researcher, representing another entity in the system, can query the encrypted data in the cloud using specific keywords. Throughout the process, the system ensures security by employing constant trapdoor generation techniques. This sequence diagram provides a visual representation of the system's functionality, demonstrating how users can securely search and retrieve encrypted data while maintaining confidentiality.

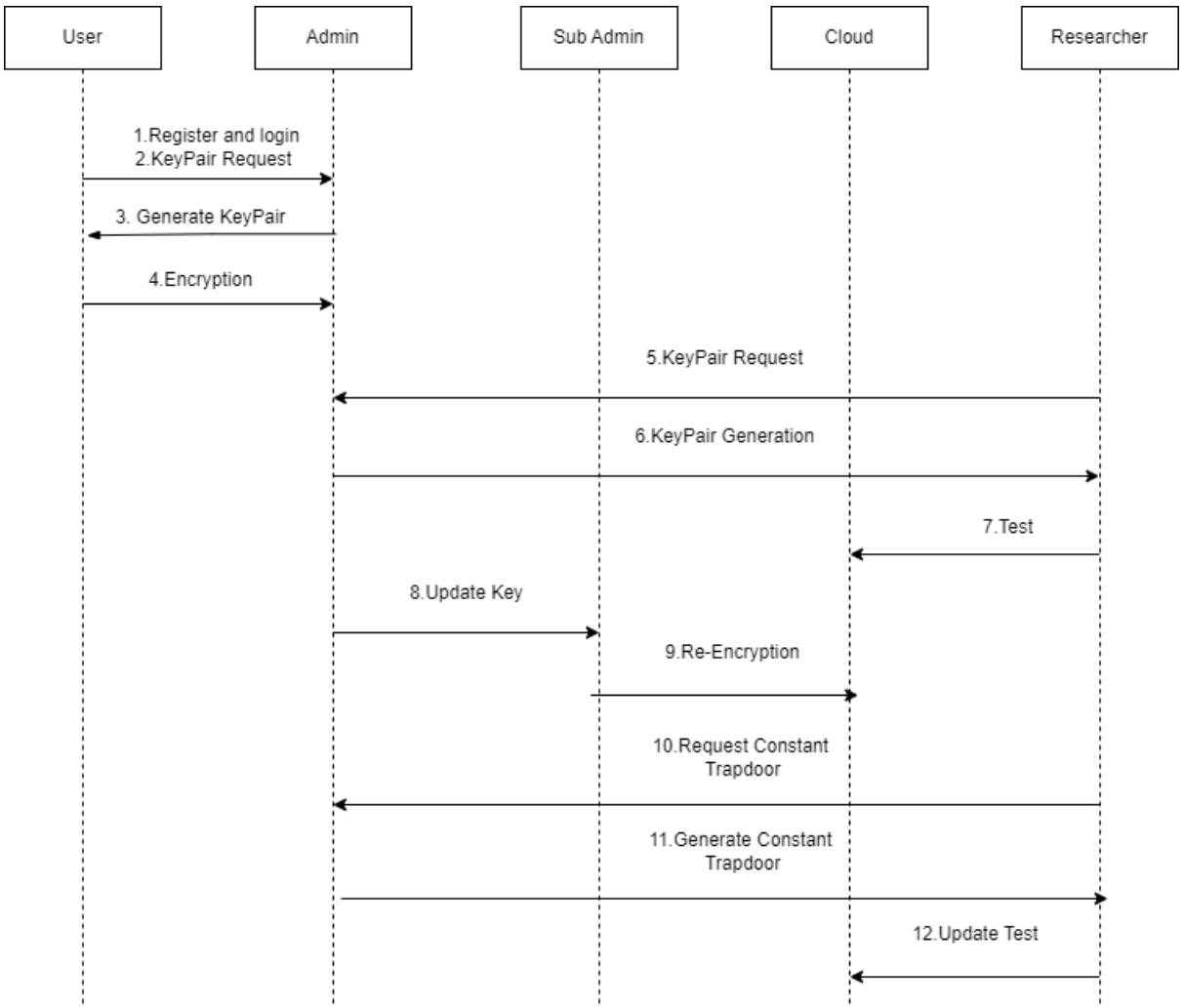


Fig. 6.3 Sequence Diagram

6.4. COLLABORATION DIAGRAM

The user initiates search or encryption tasks and interacts with the admin for access and permissions. The admin manages user rights and configures the system, coordinating with the cloud for data storage and retrieval. The cloud handles encrypted data storage and search operations. This collaborative framework ensures secure and efficient data search and encryption processes for users while maintaining system integrity and performance.

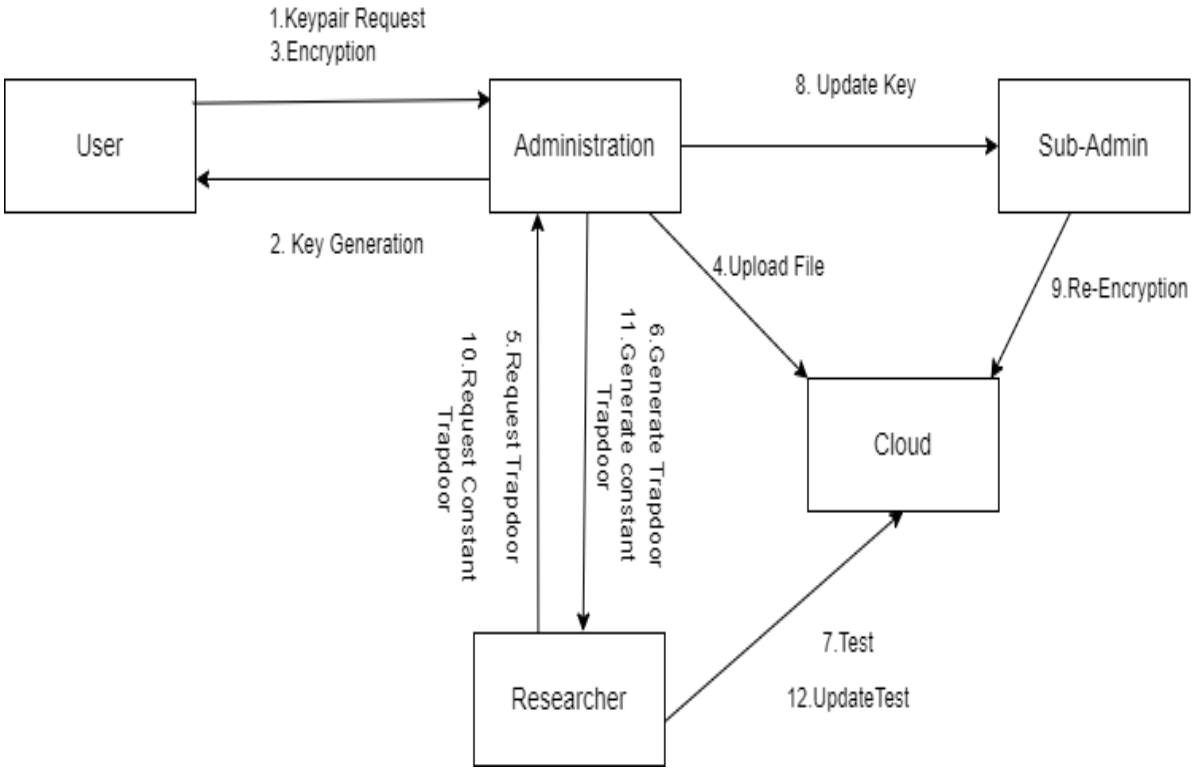


Fig. 6.4 Collaboration Diagram

6.5. STATE CHART DIAGRAM

Users first register and log in to access the system's features. Upon request, the system generates a unique keypair for each user, facilitating encryption and decryption. Users encrypt their data using their public key and generate trapdoors for keywords they wish to search within their encrypted data. The system then tests these trapdoors against the encrypted data for matches, enabling fast and efficient

searches. Updates to the keypair trigger corresponding updates to the testing process to ensure consistency and accuracy. This coherent workflow ensures secure, efficient, and user-friendly encryption with robust keyword search capabilities.

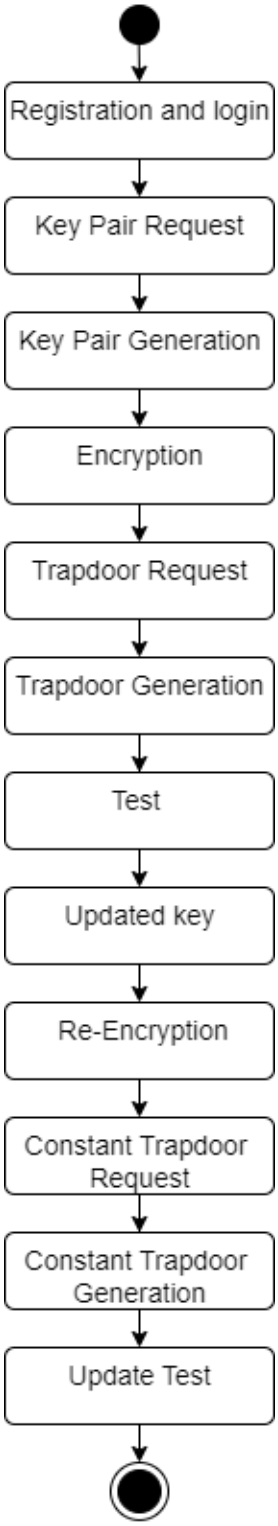


Fig. 6.5 State chart Diagram

6.6. ACTIVITY DIAGRAM

Initially, during registration, users provide necessary information, following which the system generates public and private keys for them, ensuring secure storage of this data. Upon login, users enter their credentials, which are then verified by the system. Upon successful authentication, the system generates a symmetric key for encryption and decryption purposes, securely associating it with the user. Subsequently, users can encrypt the file. The encrypted content is then securely stored in cloud. Finally, the system undergoes testing, which may involve the search functionality.

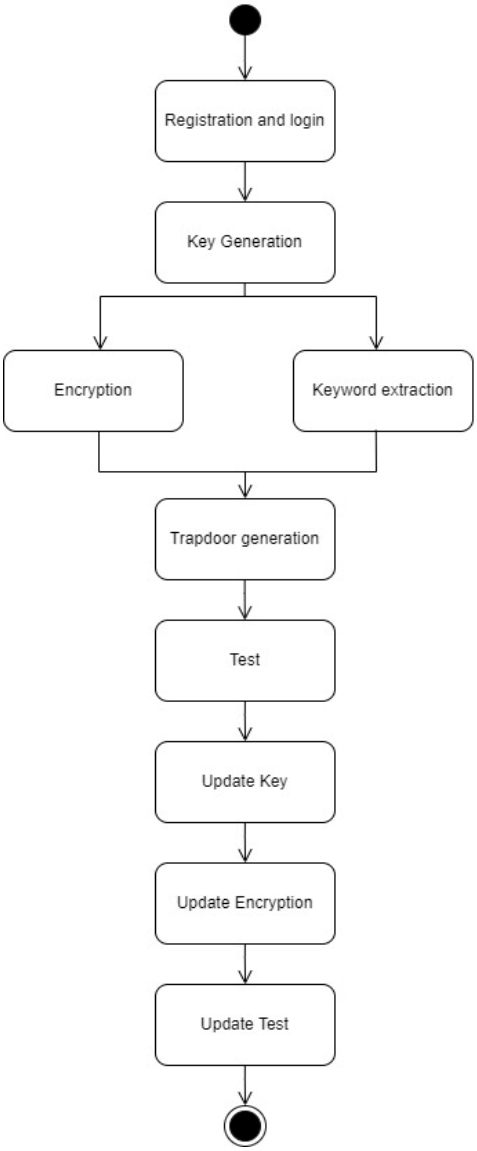


Fig. 6.6 Activity Diagram

6.7. COMPONENT DIAGRAM

Users, including regular users, administrators, and sub-administrators, interact with the system for various tasks such as data encryption and access management. The cloud component represents the infrastructure where the system is hosted, encompassing servers, storage, and networking resources. Researchers, with specialized access permissions, utilize the system for data analysis and research purposes. Each component collaborates to enable features like constant trapdoor generation and fast keyword search, facilitating secure and efficient data management within the system.

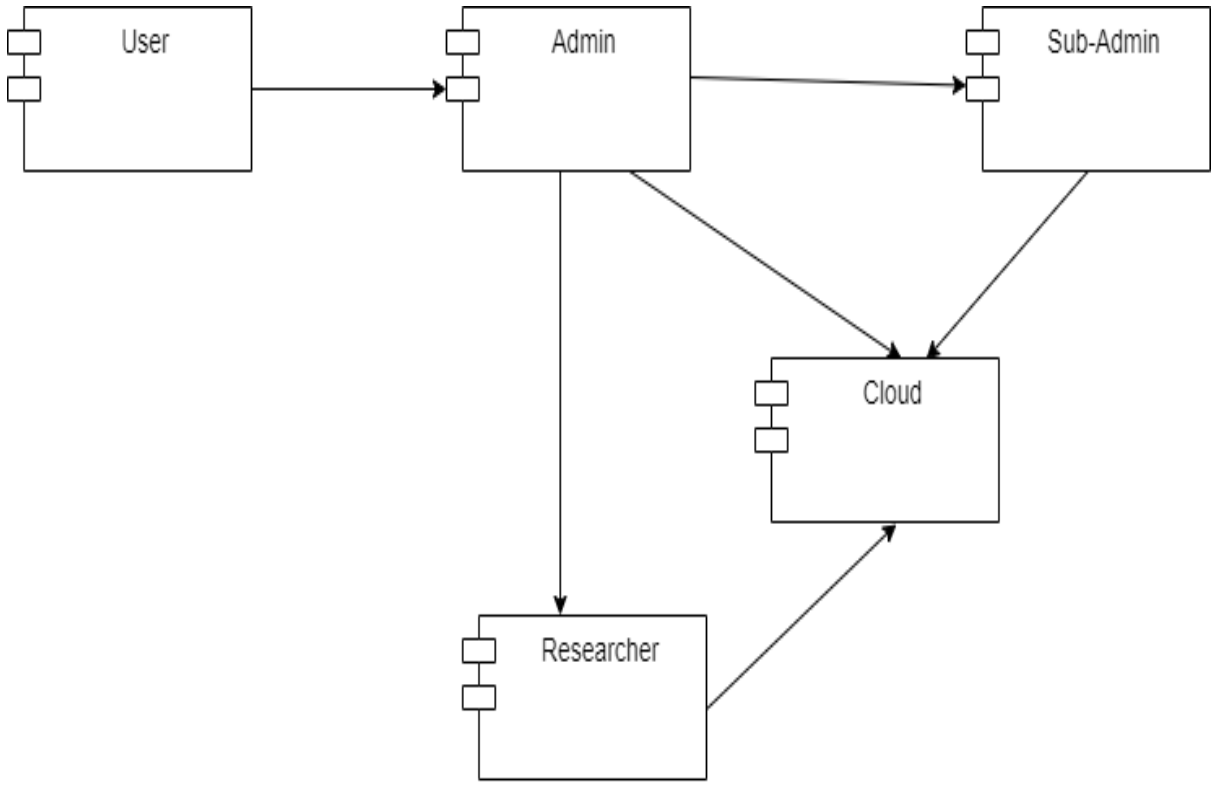


Fig. 6.7 Component Diagram

6.8. DEPLOYMENT DIAGRAM

Key generation is responsible for creating public/private key pairs, encryption encrypts data using the public key, and trapdoor generation generates the trapdoor for keyword search. The test component verifies the keyword search process, while the update test ensures the correctness of updated data. Constant

trapdoor management maintains a consistent trapdoor for efficient keyword search, and re-encryption allows for secure data updates. The update key component handles the updating of encryption keys when necessary. These components collectively enable a secure and efficient system for authenticated encryption with keyword search while supporting constant trapdoor generation.

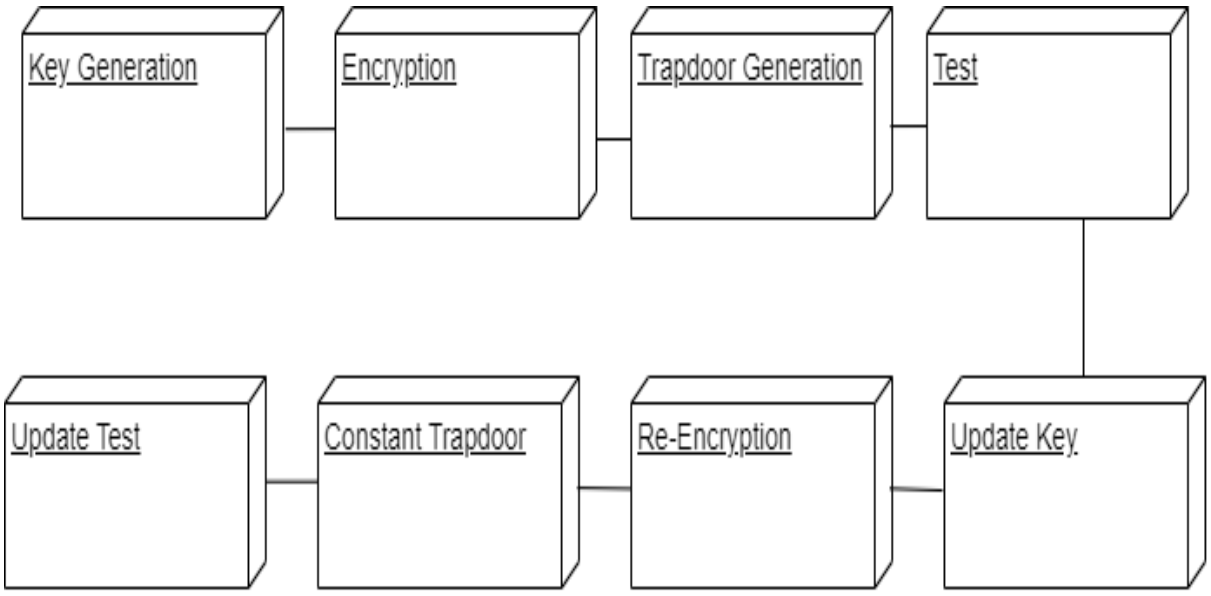


Fig. 6.8 Deployment Diagram

6.9 PACKAGE DIAGRAM

Key Generation creates the necessary public and private keys. These keys are then used by the Encryption component to encrypt plaintext data. The Trapdoor Generation component creates trapdoors for efficient keyword searches, while the Test component verifies if a given trapdoor matches the encrypted keywords. Additionally, the Update Test ensures that the test process remains up-to-date with any changes. The system also includes Constant Trapdoor generation to maintain efficiency regardless of the number of keywords. Re-Encryption allows for ciphertexts to be re-encrypted under different public keys, and Update Key ensures the encryption keys remain secure and up-to-date.

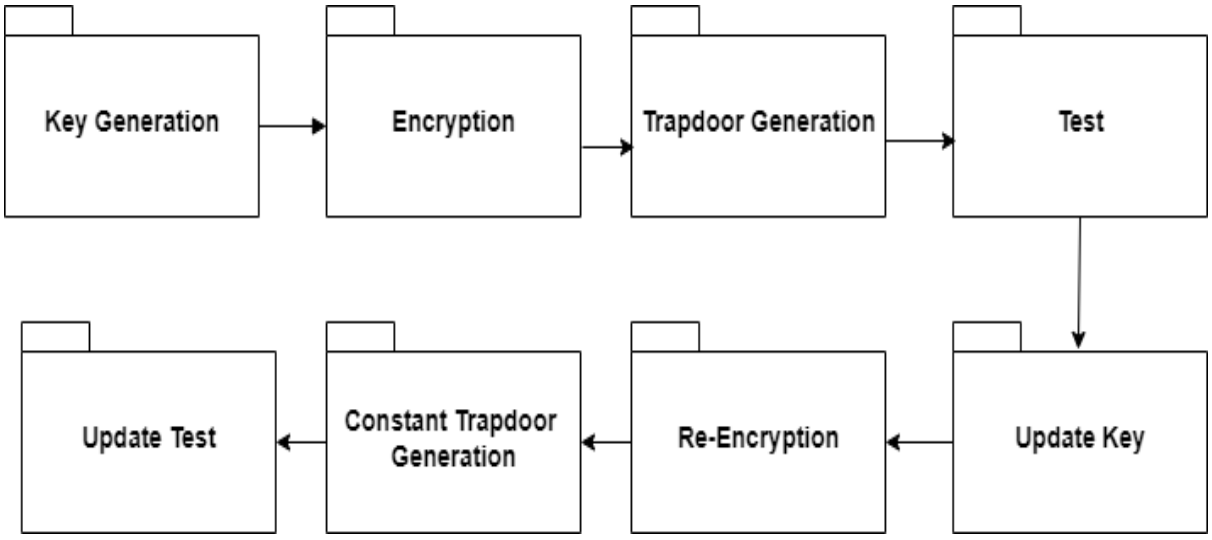


Fig. 6.9 Package Diagram

6.10 DATA FLOW DIAGRAM (DFD)

A data flow diagram (DFD) is a graphical representation of the 'flow' of a data through an information system. It differs form the flowchart as it shows the data flow instead of the control flow of the program. A data flow diagram can also be used for the visualization of data processing.

6.10.1 DFD LEVEL 0

It begins with the Levels Key Pair Generation (LKPG), responsible for generating the necessary key pair for encryption. Following this, the Encryption Trapdoor Generation (ETG) process produces trapdoors essential for efficient keyword searches within encrypted data. The system also incorporates an Update Test (UT) process to ensure ongoing adaptability and integrity. Together, these processes enable secure data storage and retrieval, maintaining both data security and search efficiency as key objectives of the system.

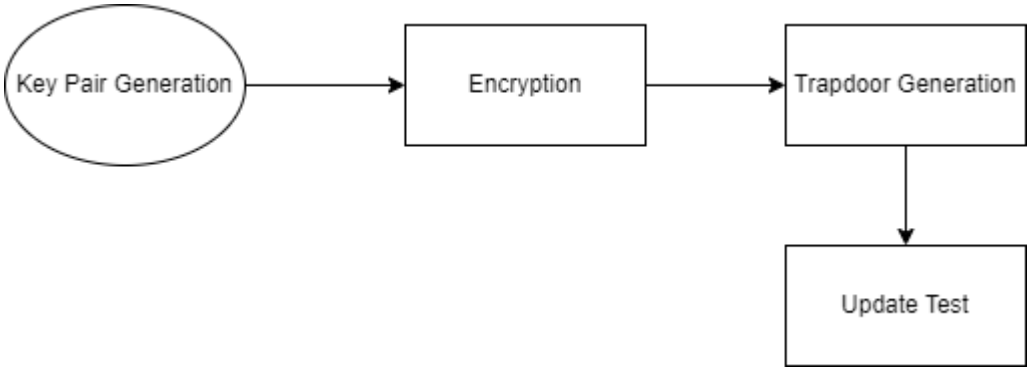


Fig. 6.10.1 DFD Level 0

6.10.2 DFD LEVEL 1

It starts with key generation, where cryptographic keys are generated for encryption and authentication purposes. These keys are then used for encryption. The system also generates trapdoors, enabling efficient keyword search on the encrypted data. Additionally, it includes processes for testing which search the file. The system supports constant trapdoor generation, providing consistent and reliable search capabilities. Overall, the Level 1 DFD provides a comprehensive overview of the encryption, authentication, and search functionalities supported by the system.

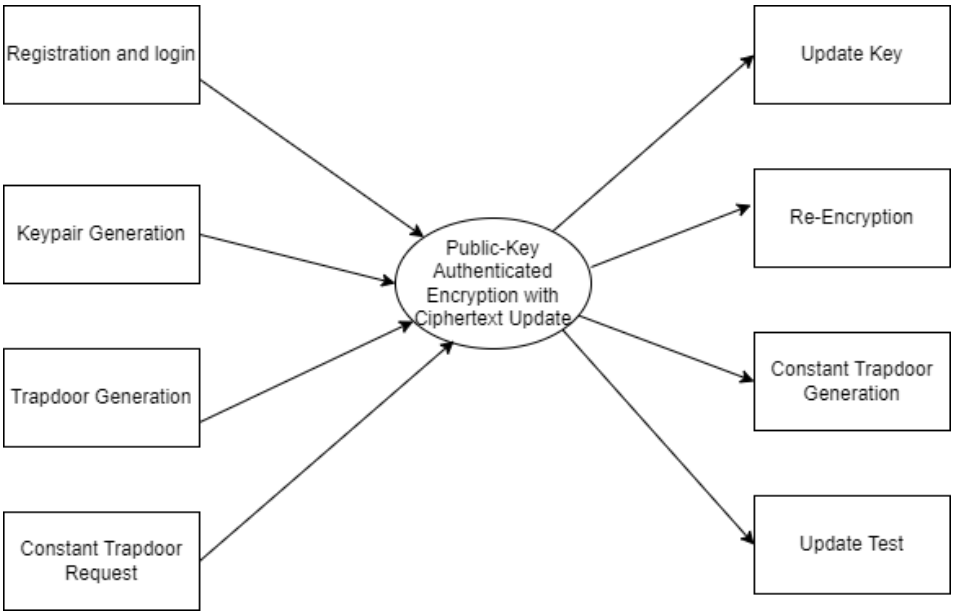


Fig. 6.10.2 DFD Level 1

6.10.3 DFD LEVEL 2

The system includes components for user registration and login, key generation, encryption, indexing, insertion of indexes, fast search, and constant trapdoor generation. Upon registration and login, users interact with the system to generate cryptographic keys. These keys are then utilized for encrypting and decrypting data. The system also facilitates the indexing of encrypted data for efficient retrieval through fast search operations. Additionally, it supports the generation of constant trapdoors, enhancing the security and efficiency of keyword searches while maintaining user privacy.

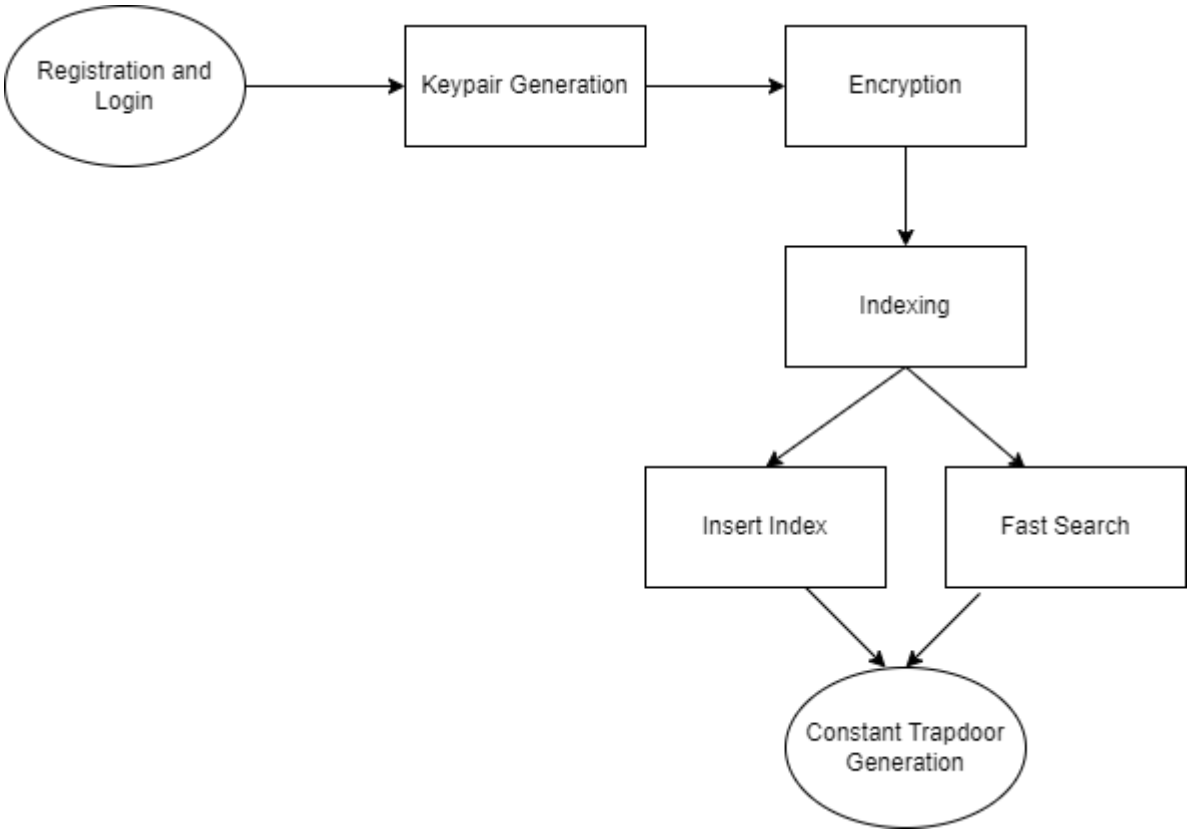


Fig. 6.10.3 DFD Level 2

CHAPTER 7

IMPLEMENTATION

7.1 JAVA

Java is a widely-used programming language known for its platform independence, simplicity, and robustness. It allows developers to write code that can run on any device with a Java Virtual Machine (JVM). Java is highly versatile and is used for web development, mobile apps, enterprise systems, and more. Its key features include object-oriented programming, built-in error handling, and support for multithreading.

7.2 STRUCTURE

7.2.1 KEY GENERATION

Implement a key generation algorithm to generate public and private keys for encryption and decryption operations. Ensure that keys are securely generated and stored.

7.2.2 ENCRYPTION AND DECRYPTION

Implement functions for encrypting and decrypting EMRs using the chosen cryptographic algorithms. Ensure that the encryption process preserves data integrity and confidentiality.

7.2.3 TRAPDOOR GENERATION

Implement a trapdoor generation algorithm to generate trapdoors for keywords. This algorithm should securely transform keywords into trapdoors that can be used for efficient search operations.

7.3 SNAPSHOTS

HOME PAGE

We present the graphical environment of our system using java. The home page contains register button and login button. The user can register or login using their credentials.

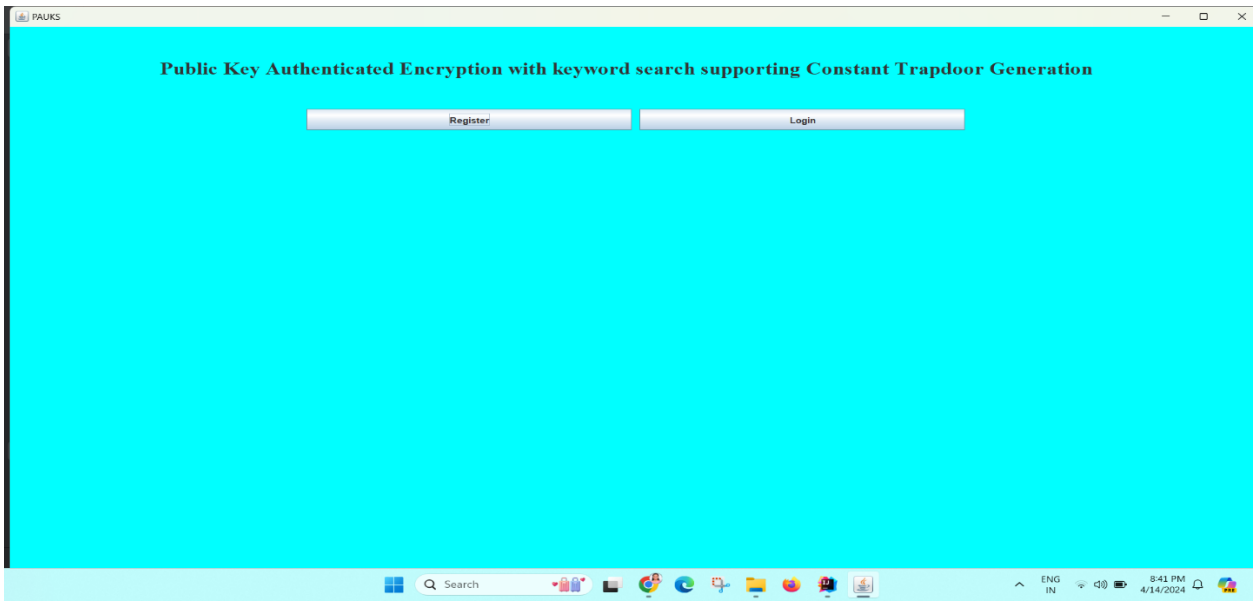
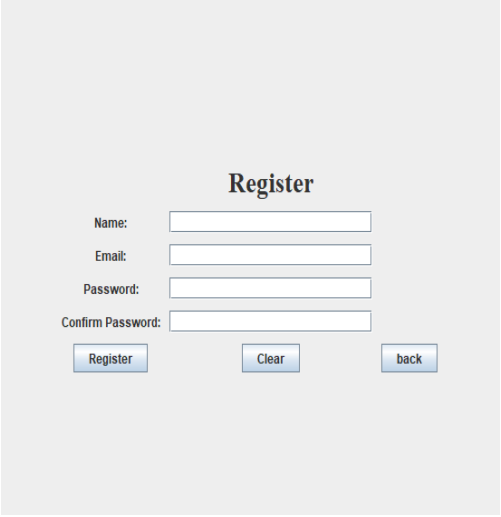


Fig. 7.1 Home page

REGISTER AND LOGIN

The registration and login process serves as the primary means of authenticating users. Returning users can securely access their accounts by logging in with their email and password, while new users can easily register by providing their email and setting up a password.



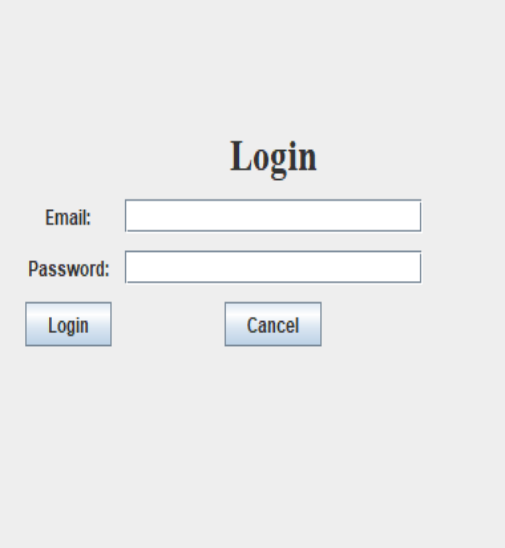


Fig. 7.2 Register and login

KEY GENERATION

Once the message is generated, it remains static until clicked. Similarly, the key pair is created upon clicking the key generation button. The key pair contains public key and secret key.

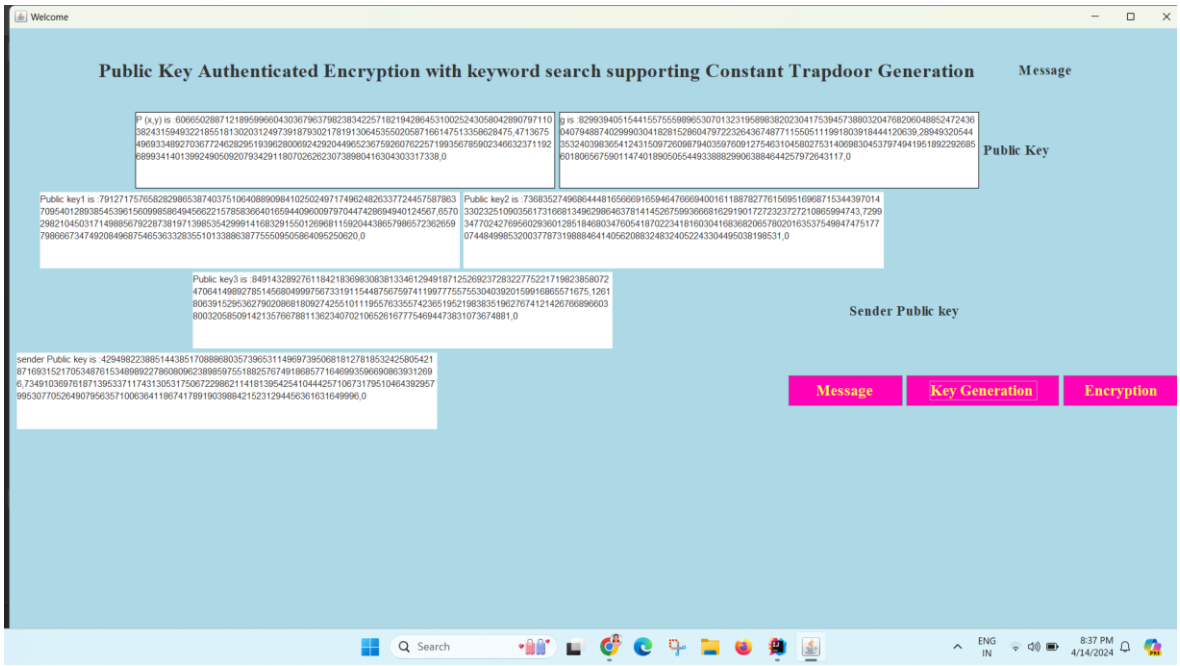


Fig. 7.3 Key Generation

ENCRYPTION

Utilizing both the message and the generated key pair, encryption is performed. This process involves transforming the message into an encoded format using the generated key pair, ensuring secure transmission and storage of sensitive information. The encryption process involves applying a mathematical algorithm to the plaintext along with an encryption key, resulting in ciphertext that appears random and unintelligible without the corresponding decryption key.

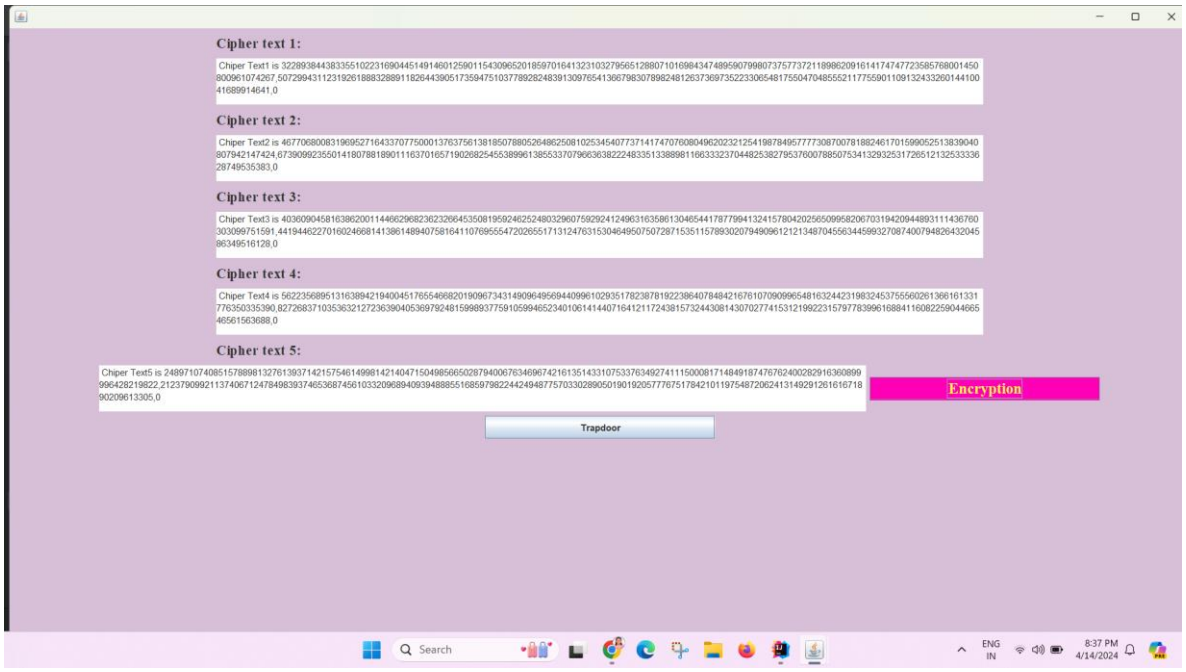


Fig. 7.4 Encryption

TRAPDOOR GENERATION AND TESTING

Upon requesting a trapdoor, it is generated utilizing the provided keyword. Subsequently, the "test" function will execute a search for the file, returning either a match or non-match as output. Following this, a constant trapdoor is generated and applied, and the updated test function is executed, producing the corresponding output. Constant trapdoor generation contributes to faster and more accurate search results. This security feature protects sensitive information from unauthorized access.

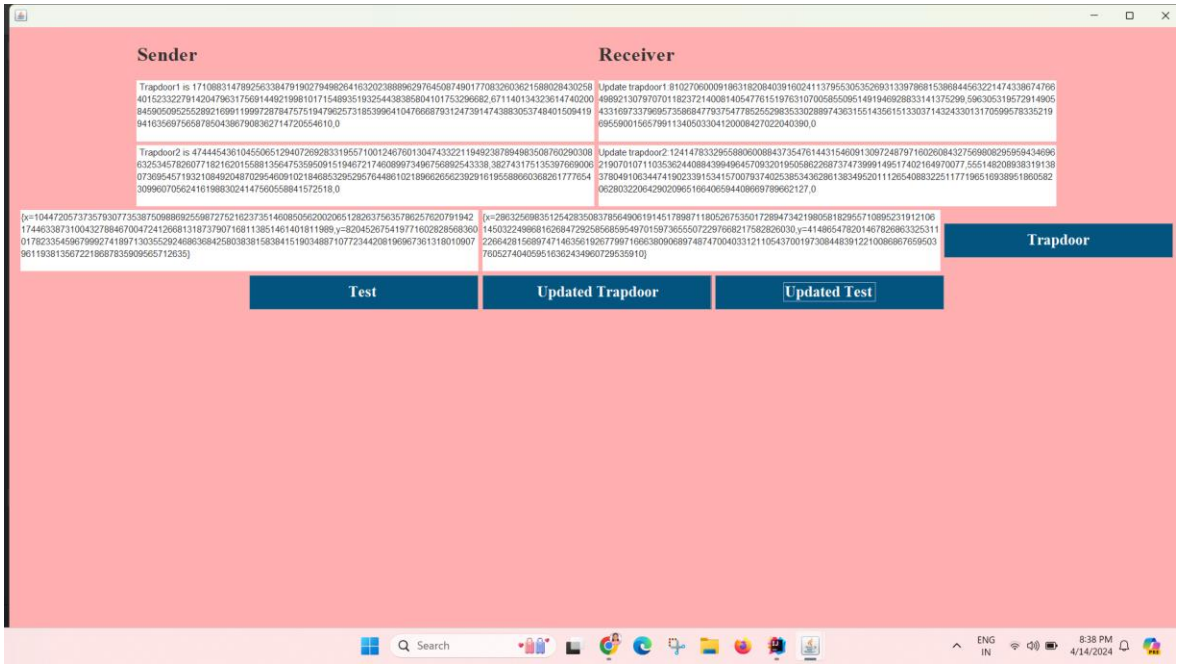


Fig. 7.5 Trapdoor generation and testing

RESULT

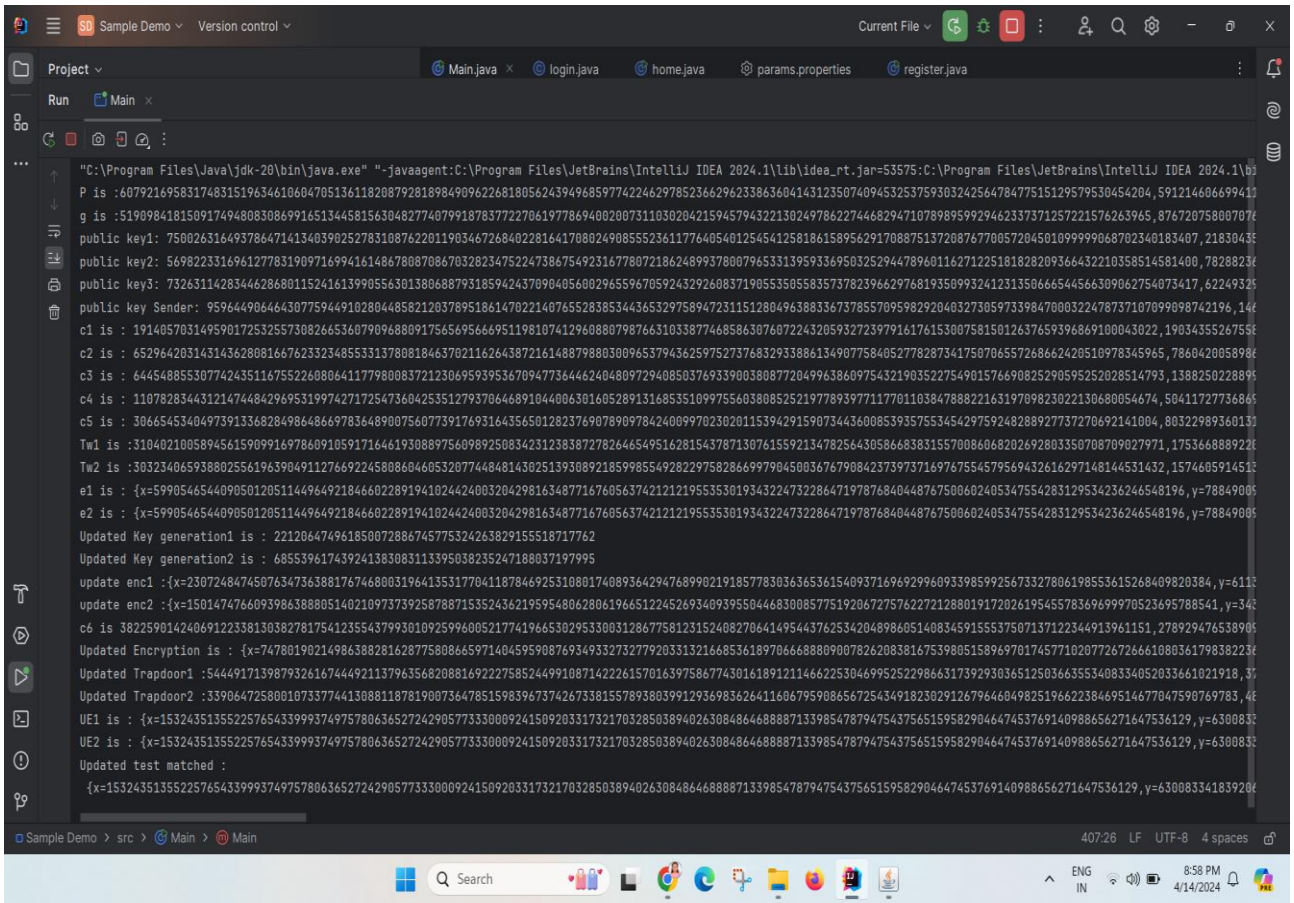


Fig. 7.6 Result

CHAPTER 8

TESTING

8.1 SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test.

8.1.1 TYPES OF TESTING

8.1.1.1 UNIT TESTING

For testing the key generation functions for correctness and ensure they generate secure and unique keys every time. Ensure that the encryption and decryption modules work accurately so that only the intended recipient with the correct key can decrypt the message. Verify that the trapdoor generation is constant in time as advertised, regardless of the input size or system load. Test the search function to ensure it correctly identifies and retrieves all instances of data associated with the keyword represented by the trapdoor.

8.1.1.2 INTEGRATION TESTING

Test the complete flow from key generation, encryption, trapdoor generation, data uploading, searching, and decryption to ensure that all parts work together seamlessly. Evaluate the interaction between different modules (e.g., the encryption module and the search module) to detect any integration issues. It provides systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

8.1.1.3 FUNCTIONAL TESTING

To verify that the system returns the correct search results without any false positives or negatives. Test with a variety of keywords to ensure consistent

performance regardless of keyword length or complexity.

The following items are the focus of functional testing.

1. Valid Input: identified classes of valid input must be accepted.
2. Invalid Input: identified classes of invalid input must be rejected.
3. Functions: identified functions must be exercised.
4. Output: identified classes of application outputs must be exercised.
5. Systems/Procedures: interfacing systems or procedures must be invoked.

8.1.1.4 SYSTEM TESTING

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

8.1.1.5 WHITE BOX TESTING

White box testing involves an in-depth examination of the internal structures or workings of the application, which is possible due to the tester's knowledge of the software's source code, architecture, and implementation. Test the logical paths through the code for the key generation, encryption/decryption processes, trapdoor generation, and searching algorithms. Evaluate the code for potential buffer overflows, memory leaks, and other security vulnerabilities that could be exploited. Ensure that all independent paths within a module have been exercised through the tests, including loop bounds and internal data structure boundaries.

8.1.1.6 BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. The test provides inputs and responds to outputs without considering how the software works. Test the system's response to various input conditions in encryption, decryption, trapdoor generation, and searching. This includes testing with valid and invalid input data to see if the system behaves as expected.

CHAPTER 9

RESULT AND DISCUSSION

PERFORMANCE ANALYSIS

This section evaluates the performance of our scheme that communication and computation overhead of PAUKS scheme compared with PAEKS scheme.

9.1 COMMUNICATION OVERHEAD

To evaluate the computation overhead, we construct bilinear pairings e:G × G → GT and terms of ciphertext and the trapdoors

Scheme	Ciphertext	Trapdoor	Updated ciphertext	Updated trapdoor
PAEKS	2|G|	m.$\|G_T\|$	-	-
PAUKS	5|G|	m.(2$\|G_T\|$)	6|G|	2|G|

Table 9.1 Communication Overhead Evaluation

In our approach, the PAUKS scheme has a higher but still comparable communication overhead in terms of the ciphertext and the trapdoor, after ciphertext update, the trapdoor communication overhead of our PAUKS scheme reduces to constant complexity and is significantly lower in comparison with when the number of senders increases. Our PAUKS scheme has slightly larger computation overhead than PAEKS scheme in terms of Encryption, Trapdoor and Test algorithms, in enjoys a lower communication overhead when using Constant Trapdoor and Updated Test algorithms to realize ciphertext retrieval.

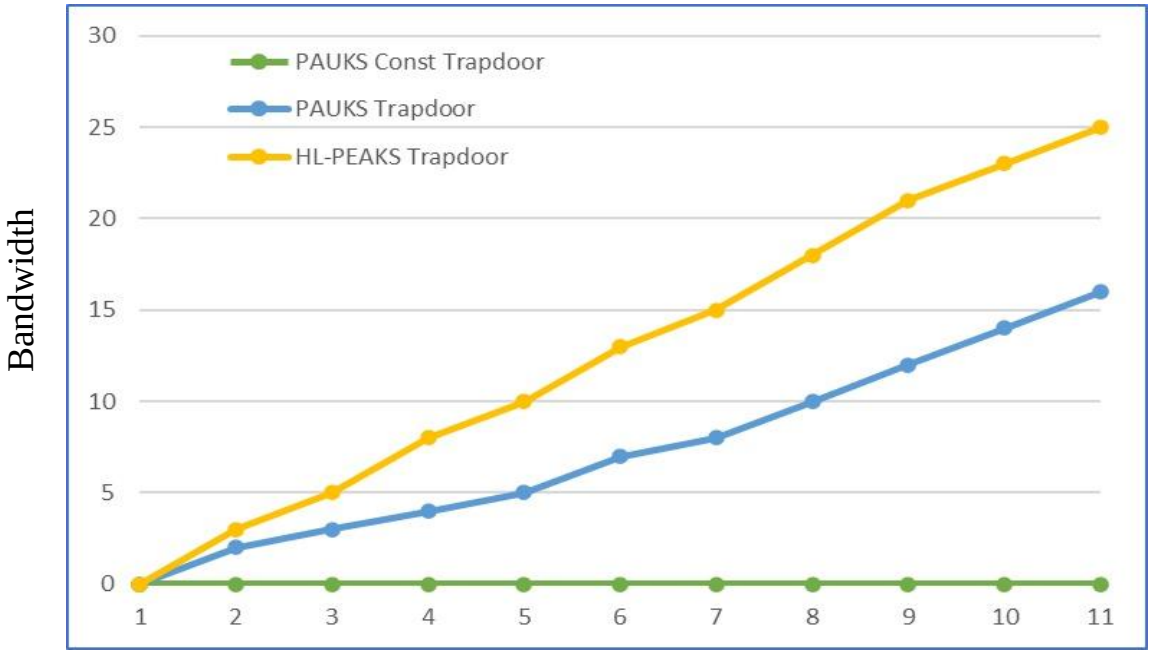


Fig. 9.1 Trapdoor bandwidth cost comparison.

9.2 STORAGE COST

In the scheme, the storage cost of each key, ciphertext and trapdoor. The storage cost of the sender's keys in both PAEKS and our PAUKS scheme needs large storage space that PAEKS in terms of the receiver's key, ciphertext and trapdoors shown on Figure 9.1, PAUKS supports the constant trapdoor transmission after the updated ciphertext and needs much less the bandwidth when the number of senders is large.

Scheme	Public key		Secret key		Updated Key	Ciphertext		Trapdoor	
	Sender	Receiver	Sender	Receiver		Sender	Receiver	Sender	Receiver
PAEKS	128	128	20	20	-	256	-	128	-
PAUKS	128	384	20	80	40	640	768	256	256

Table 9.2 Storage Cost of Keys, Ciphertext and Trapdoor

The storage cost of each key, ciphertext and trapdoor. The storage cost of the sender's keys in both HL-PAEKS and our PAUKS are the same. Due to extensional

functions, our PAUKS scheme needs larger storage space than HL-PAEKS scheme in terms of the receiver's keys, ciphertexts and trapdoors. PAUKS scheme supports constant trapdoor transmission after ciphertext update and needs much less bandwidth when the number of senders is large. The storage requirements for the sender's keys are identical between HL-PAEKS and PAUKS. This similarity suggests that both schemes utilize a similar cryptographic foundation for key generation and storage on the sender's side.

The PAUKS scheme, despite its higher storage demands on certain components, strategically reduces the operational bandwidth for scenarios involving numerous senders and frequent ciphertext updates. This makes it an attractive option for large-scale implementations where bandwidth conservation is crucial. The trade-off between increased local storage and reduced bandwidth usage is a common theme in systems design, reflecting a tailored approach to optimize resource usage based on specific operational priorities and constraints. PAUKS exemplifies how cryptographic designs can evolve to meet the demands of more dynamic and complex network environments.

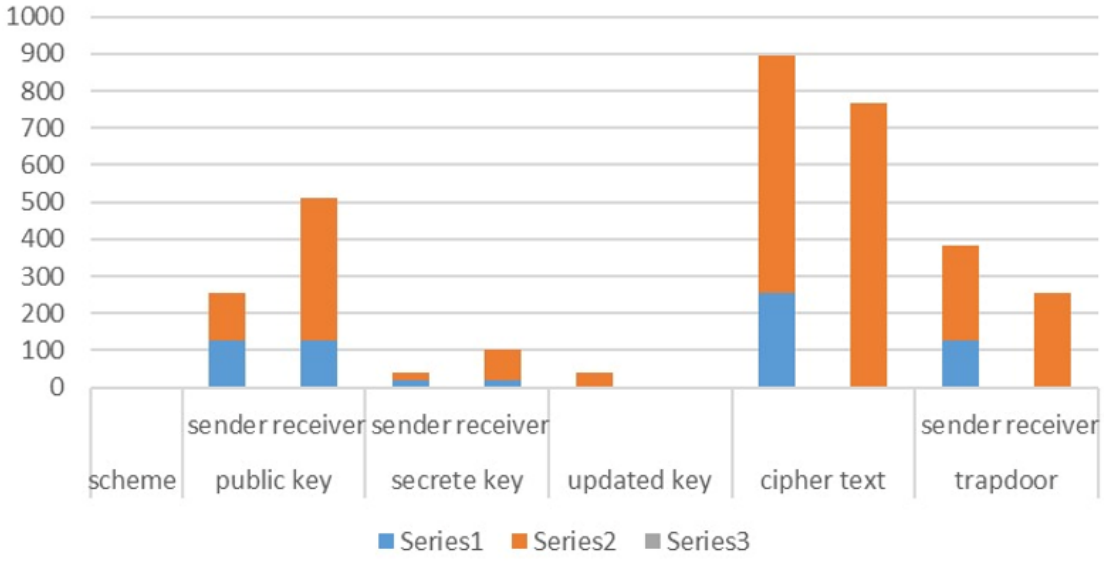


Fig. 9.2 Storage Cost.

CHAPTER 10

CONCLUSION AND FUTURE WORK

10.1 CONCLUSION

Our project proposes an EMR system by introducing a public key authenticated encryption with ciphertext update and keyword search (PAUKS) scheme. Different from preceding PAEKS schemes, the proposed PAUKS scheme reduces trapdoor communication overhead from linear to sub-linear with the help of a non-collusion proxy. As a side result, the proposed PAUKS scheme is compatible with a secure inverted index, which achieves a searching efficiency. The experimental results demonstrate that the PAUKS scheme is a promising solution for secure and efficient storage and sharing of EMRs in cloud environments. The scheme offers comparable running overhead and higher query efficiency after updating ciphertexts, making it suitable for scenarios where EMRs are frequently modified or updated. Further research can focus on evaluating the scheme's performance under different scenarios and improving its security against external attacks.

10.2 FUTURE WORK

Future work would concentrate on enhancing the proposed scheme for storing and sharing electronic medical records in cloud servers. To involve exploring optimizations to further reduce computation and communication overheads, improving the scheme's scalability to handle larger volumes of EMRs, and conducting thorough security evaluations to identify and address potential vulnerabilities. Additionally, research can be conducted to investigate the integration of emerging technologies such as blockchain and homomorphic encryption in the context of EMR storage and sharing.

REFERENCES

[1] Hongbo Li, Qiong Huang, Jianye Huang, Willy Susilo," Public-Key Authenticated Encryption with Keyword Search Supporting Constant Trapdoor Generation and Fast Search," IEEE Transaction on Information Forensics and Security, vol. 18, June 2023.

[2] Y. Lu and J. Li, "Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices," IEEE Transaction on Mobile Computing., vol. 21, no. 12, pp. 4397–4409, Dec. 2022.

[3] Z.-Y. Liu, Y.-F. Tseng, R. Tso, Y.-C. Chen, and M. Mambo, "Identity-certifying authority-aided identity-based searchable encryption framework in cloud systems," IEEE System, vol. 16, no. 3, pp. 4629–4640, Sep. 2022.

[4] Y. Lu, J. Li, and Y. Zhang, "Secure channel free certificate-based searchable encryption withstanding outside and inside keyword guessing attacks," IEEE Transactions on Services Computing., vol. 14, no. 6, pp. 2041–2054, Nov. 2021.

[5] X. Liu, K. He, G. Yang, W. Susilo, J. Tonien, and Q. Huang, "Broadcast authenticated encryption with keyword search," in Proceedings of the Australasian Conference on Information Security and Privacy. Cham, Switzerland: Springer, pp. 193–213, Sep. 2021.

[6] B. Qin, H. Cui, X. Zheng, and D. Zheng, "Improved security model for public-key authenticated encryption with keyword search," in Proceedings of the

International Conference on Provable Practical Security. Cham, Switzerland: Springer, pp. 19–38, July 2021.

[7] X. Pan and F. Li, “Public-key authenticated encryption with keyword search achieving both multi-ciphertext and multi trapdoor indistinguishability,” Journal of Systems Architecture., vol. 115, Art. no. 102075 May 2021.

[8] Y. Lu, J. Li, and F. Wang, “Pairing-free certificate-based searchable encryption supporting privacy-preserving keyword search function for IIoTs,” IEEE Transactions on Industrial Informatics. vol. 17, no. 4, pp. 2696–2706, Apr. 2021.

[9] R. Behnia, M. O. Ozmen, and A. A. Yavuz, “Lattice-based public key searchable encryption from experimental perspectives,” IEEE Transaction on Dependable Secure Computing, vol. 17, no. 6, pp. 1269–1282, Nov. 2020.

[10] N. Pakniat, D. Shiraly, and Z. Eslami, “Certificateless authenticated encryption with keyword search: Enhanced security model and a concrete construction for industrial IoT,” Journal of Information Security and Applications, vol. 53, Art. no. 102525, Aug. 2020.

[11] L. Xu, Z. Sun, W. Li, and H. Yan, “Delegatable searchable encryption with specified keywords for EHR systems,” Wireless Network, pp. 1–13, Jul. 2020.

[12] L. Xu, Z. Sun, W. Li, and H. Yan, “Delegatable searchable encryption with specified keywords for EHR systems,” Wireless Network, pp. 1–13, Jul. 2020.

[13] B. Wu, C. Wang, and H. Yao, “Security analysis and secure channel free certificateless searchable public key authenticated encryption for a cloud-based Internet of Things,” Public Library of Science ONE, vol. 15, no. 4, Art. no. e0230722, Apr. 2020.

[14] B. Qin, Y. Chen, Q. Huang, X. Liu, and D. Zheng, “Public key authenticated encryption with keyword search revisited: Security model and constructions,” Information Science, vol. 516, pp. 515–528, Apr. 2020.

[15] H. Li, Q. Huang, J. Shen, G. Yang, and W. Susilo, “Designated server identity-based authenticated encryption with keyword search for encrypted emails,” Information Science., vol. 481, pp. 330–343, May 2019.

[16] X. Liu, H. Li, G. Yang, W. Susilo, J. Tonien, and Q. Huang, “Towards enhanced security for certificateless public-key authenticated encryption with keyword search,” in Provable Security (Lecture Notes in Computer Science), R. Steinfeld and T. H. Yuen, Eds. Cham, Switzerland: Springer, pp. 113–129, Apr. 2019.

[17] T. Hoang, M. O. Ozmen, Y. Jang, and A. A. Yavuz, “Hardware supported ORAM in effect: Practical oblivious search and update on very large dataset,” Proceedings on Privacy Enhancing Technologies, vol. 2019, no. 1, pp. 172–191, Jan. 2019.

[18] Q. Huang and H. Li, “An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks,” Information Science, vols. 403–404, pp. 1–14, Sep. 2017.

[19] R. Chen et al., “Server-aided public key encryption with keyword search,” IEEE Transaction on Information Forensics Security, vol. 11, no. 12, pp. 2833–2842, Dec. 2016.

[20] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, “Dual-server public-key encryption with keyword search for secure cloud storage,” IEEE Transactions on Information Forensics and Security, vol. 11, no. 4, pp. 789–798, Apr. 2016.

[21] Z. Chen, S. Li, Y. Guo, Y. Wang, and Y. Chu, “A limited proxy re-encryption with keyword search for data access control in cloud computing,” in

Proceedings of the International Conference on Network Systems Security. Cham, Switzerland: Springer, pp. 82–95, 2015.

[22] L. Fang, W. Susilo, C. Ge, and J. Wang, “Public key encryption with keyword search secure against keyword guessing attacks without random Oracle,” Information Science, vol. 238, pp. 221–241, Jul. 2013.

[23] J. Shao, Z. Cao, X. Liang, and H. Lin, “Proxy re-encryption with keyword search,” Information Science, vol. 180, no. 13, pp. 2576–2587, Jul. 2010.

[24] J. Baek, R. Safavi-Naini, and W. Susilo, “Public key encryption with keyword search revisited,” in Proceedings International Conference on Computer Science Applications Perugia, Italy, pp. 1249–1259, 2008.

[25] X. Boyen, “The uber-assumption family,” in Proceedings International Conference Pairing Based Cryptography Cham, Switzerland: Springer, pp. 39–56, 2008.

[26] J. W. Byun, H. S. Rhee, H. A. Park, and D. H. Lee, “Off-line keyword guessing attacks on recent keyword search schemes over encrypted data,” in Secure Data Management, 3rd ed. Seoul, (South) Korea: SDM, Sep. 2006.

[27] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in Advances in Cryptology—EUROCRYPT, C. Cachin and J. L. Camenisch, Eds. Berlin, Germany: Springer, pp. 506–522, 2004.

[28] F. Bao, R. H. Deng, and H. Zhu, “Variations of Diffie-Hellman problem,” in Proceedings International Conference on Information Communication Security Cham, Switzerland: Springer, pp. 301–312, 2003.

APPENDICES

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.*;
import javax.swing.*;
import java.awt.*;
import java.awt.Color;
import javax.swing.border.EmptyBorder;
import java.math.BigInteger;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class Main extends JFrame{
    private JTextArea  m1, m2, m31, m32, m33 ;
    private JLabel m, p31, p32;
    private JButton b1, b2;
public Main(){
    setTitle("Welcome");
    setSize(getMaximumSize());
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JLabel w1 = new JLabel("Public Key Authenticated Encryption with keyword
    search supporting Constant Trapdoor Generation");
    w1.setFont(new Font("Times New Roman",Font.BOLD,26));
    w1.setPreferredSize(new Dimension(1200,100));
```

```
JLabel m=new JLabel("Message ");
m.setPreferredSize(new Dimension(100, 50));
m.setFont(new Font("Times New Roman",Font.BOLD,18));
JTextArea m1=new JTextArea();
m1.setPreferredSize(new Dimension(550, 100));
m1.setBorder(BorderFactory.createLineBorder(Color.BLACK,1));
JTextArea m2 = new JTextArea();
m2.setPreferredSize(new Dimension(550, 100));
m2.setBorder(BorderFactory.createLineBorder(Color.BLACK,1));
JLabel k1=new JLabel("Public Key ");
k1.setPreferredSize(new Dimension(100, 50));
k1.setFont(new Font("Times New Roman",Font.BOLD,18));
JTextArea m31 = new JTextArea();
m31.setPreferredSize(new Dimension(550, 100));
JTextArea m32 = new JTextArea();
m32.setPreferredSize(new Dimension(550, 100));
JLabel k2=new JLabel("     ");
k2.setPreferredSize(new Dimension(350, 50));
JLabel k3=new JLabel("     ");
k3.setPreferredSize(new Dimension(450, 50));
JTextArea m33 = new JTextArea();
m33.setPreferredSize(new Dimension(550, 100));
JTextArea m34 = new JTextArea();
m34.setPreferredSize(new Dimension(550,100));
JLabel p31 = new JLabel();
p31.setPreferredSize(new Dimension(300,100));
JLabel p32 = new JLabel("Sender Public key ");
```

```
p32.setPreferredSize(new Dimension(200, 100));
p32.setFont(new Font("Times New Roman",Font.BOLD,18));
JButton b1 = new JButton("Message ");
b1.setPreferredSize(new Dimension(150,40));
JButton b2 = new JButton("Key Generation ");
b2.setPreferredSize(new Dimension(150,40));
JButton b3 = new JButton("Encryption");
b3.setPreferredSize(new Dimension(150,40));
JPanel panel=new JPanel();
panel.setBackground(new Color(173, 216, 230));
panel.add(w1);
panel.add(m);
panel.add(m1);
panel.add(m2);
panel.add(k1);
panel.add(m31);
panel.add(m32);
panel.add(k2);
panel.add(m33);
panel.add(p31);
panel.add(p32);
panel.add(m34);
panel.add(k3);
panel.add(b1);
panel.add(b2);
panel.add(b3);
Pairing pairing = PairingFactory.getPairing("params.properties");
```

```
PairingFactory.getInstance().setUsePBCWhenPossible(true);
// initiate p
Element P = pairing.getG1().newRandomElement().getImmutable();
System.out.println("P is :" + P);
// initiate g
Element g = pairing.getG1().newRandomElement().getImmutable();
System.out.println("g is :"+g);
Element x1 = pairing.getZr().newRandomElement().getImmutable();
Element x2 = pairing.getZr().newRandomElement().getImmutable();
Element x3 = pairing.getZr().newRandomElement().getImmutable();
Element x4 = pairing.getZr().newRandomElement().getImmutable();
```

```
/* public key generation  Element pr- public key of Receiver and ps- public key of sender */
```

```
Element pr1=g.powZn(x1);
System.out.println("public key1: "+pr1);
Element pr2=g.powZn(x2);
System.out.println("public key2: "+pr2);
Element pr3=g.powZn(x3);
System.out.println("public key3: "+pr3);
// Skr1= x1,skr2=x2,skr3=x3 and skr4=x4 are the serect keys
Element skr1=x1;
Element skr2=x2;
Element skr3=x3;
Element skr4=x4;
// Key Generation for receiver  - ends here.
// Key Generation for Sender  - Starts here.
// y is the Secret key of sender.
Element y = pairing.getZr().newRandomElement().getImmutable();
```

```
// public key of Sender is pks
Element pks=g.powZn(y);
System.out.println("public key Sender: "+pks);
// Key Generation for Sender  - Ends here.
// Keyword Encryption Starts here
/* Random number for powers */
Element r1 = pairing.getZr().newRandomElement().getImmutable();
Element r2 = pairing.getZr().newRandomElement().getImmutable();
/* Encryption*/
Element h1_hash_int = pairing.getZr().newRandomElement().getImmutable();
Element c1 = (pr2.powZn(h1_hash_int).add(pr3)).powZn(r1);
Element c2 = g.powZn(r1);
Element h2_hash_int = pairing.getZr().newRandomElement().getImmutable();
Element h3_hash_int = pairing.getZr().newRandomElement().getImmutable();
Element c3 = ((pr2.powZn(h2_hash_int).add(pr3)).powZn(r2)).
add(g.powZn(h3_hash_int.mul(r1)));
Element c4_hash_point = pairing.getG1().newRandomElement().
getImmutable();
Element c4 = c4_hash_point.powZn(r2);
Element c5_hash_point = pairing.getG1().newRandomElement().
getImmutable();
Element c5 = c5_hash_point.powZn(r1);
System.out.println("c1 is : " + c1);
System.out.println("c2 is : " + c2);
System.out.println("c3 is : " + c3);
System.out.println("c4 is : " + c4);
System.out.println("c5 is : " + c5);
```

```
//action
b1.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      m1.setText("P (x,y) is :"+P.toString());
      m1.setLineWrap(true);
      // initiate g
      Element g = pairing.getG1().newRandomElement().getImmutable();
      System.out.println("g is :"+g);
      m2.setText("g is :"+g.toString());
      m2.setLineWrap(true);
}
});
b2.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                       // Key Generation for receiver  - Starts here
                       m31.setText("Public key1 is :"+pr1.toString());
                       m31.setLineWrap(true);
                       m32.setText("Public key2 is :"+pr2.toString());
                       m32.setLineWrap(true);
                       m33.setText("Public key3 is :"+pr3.toString());
                       m33.setLineWrap(true);
                       m34.setText("sender Public key is :"+pks.toString());
                       m34.setLineWrap(true);
                       // Key Generation for Sender  - Ends here.
                    }
```

```
                    });
    b3.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
        message msg = new message();
        msg.setVisible(true);
        setVisible(false);
        }
    });

    add(panel);
    setVisible(true);
        //updated test ends here
    }
    public static void main(String[] args) {
        new Main();
}
}
```

Homepage.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Image;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class home extends JFrame{
```

```
// form components
private JLabel w1, w2, image, email, pass;
private JTextField t1;
private JPasswordField pt1;
private JButton reg, log;
public home() {
    setTitle("PAUKS");
    setSize(getMaximumSize());
    setLocationRelativeTo(null);
    JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));
    panel.setBackground(Color.CYAN);
    JLabel w1 = new JLabel("Public Key Authenticated Encryption with keyword
    search supporting Constant Trapdoor Generation");
    w1.setFont(new Font("Times New Roman",Font.BOLD,26));
    w1.setPreferredSize(new Dimension(1150,100));
    JButton reg = new JButton("Register");
    reg.setPreferredSize(new Dimension(400,30));
    reg.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            register reg = new register();
            reg.setVisible(true);
            setVisible(false);
        }
    });
    JButton log = new JButton("Login");
    log.setPreferredSize(new Dimension(400,30));
```

```
        panel.add(w1);
        panel.add(Box.createRigidArea(new Dimension(10, 0)));
        log.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                    login log1 = new login();
                    log1.setVisible(true);
                    setVisible(false);
            }
        });
        //add(image);
        //set text field to panel
        panel.add(reg);
        panel.add(log);
        //set border to panel
        // add(newPanel, BorderLayout.CENTER);
        add(panel);
        setVisible(true);
    }
    public static void main(String args[]){
        new home();
    }
    }
```

Register.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
import java.io.*;
    public class register extends JFrame implements ActionListener {
        // form components
        private JLabel reg, nameLabel, emailLabel, passwordLabel, confirmLabel;
        private JTextField nameField, emailField;
        private JPasswordField passwordField, confirmField;
        private JButton registerButton, clearButton, cancelButtuon;
        public register() {
            setTitle("Registration Form");
            setBackground(new Color(124,230,12));
            // create components
            reg = new JLabel("Register");
            reg.setFont(new Font("Times New Roman",Font.BOLD,26));
            nameLabel = new JLabel("Name:");
            emailLabel = new JLabel("Email:");
            passwordLabel = new JLabel("Password:");
            confirmLabel = new JLabel("Confirm Password:");
            nameField = new JTextField(20);
            emailField = new JTextField(20);
            passwordField = new JPasswordField(20);
            confirmField = new JPasswordField(20);
            registerButton = new JButton("Register");
            clearButton = new JButton("Clear");
            cancelButtuon = new JButton("back");
            // set layout
            setLayout(new GridBagLayout());
            setBackground(Color.MAGENTA);
```

```
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets(5, 5, 5, 5);
gbc.gridx = 1;
gbc.gridy = 0;
add(reg, gbc);
gbc.gridx = 0;
gbc.gridy = 1;
add(nameLabel, gbc);
gbc.gridx = 1;
gbc.gridy = 1;
add(nameField, gbc);
gbc.gridx = 0;
gbc.gridy = 2;
add(emailLabel, gbc);
gbc.gridx = 1;
gbc.gridy = 2;
add(emailField, gbc);
gbc.gridx = 0;
gbc.gridy = 3;
add(passwordLabel, gbc);
gbc.gridx = 1;
gbc.gridy = 3;
add(passwordField, gbc);
gbc.gridx = 0;
gbc.gridy = 4;
add(confirmLabel, gbc);
gbc.gridx = 1;
```

```
        gbc.gridy = 4;
        add(confirmField, gbc);
        gbc.gridx = 0;
        gbc.gridy = 5;
        add(registerButton, gbc);
        gbc.gridx = 1;
        gbc.gridy = 5;
        add(clearButton, gbc);
        gbc.gridx = 2;
        gbc.gridy = 5;
        add(cancelButtuon, gbc);
        // add action listeners
        registerButton.addActionListener(this);
        clearButton.addActionListener(this);
        cancelButtuon.addActionListener(this);
        // set size and visibility
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == registerButton) {
            // validate form input
            String name = nameField.getText().trim();
            String email = emailField.getText().trim();
            char[] password = passwordField.getPassword();
            char[] confirm = confirmField.getPassword();
```

```
        if (name.isEmpty() || email.isEmpty() || password.length == 0 ||
   confirm.length == 0) {
        JOptionPane.showMessageDialog(this, "Please fill out all fields.");
        } else if (!String.valueOf(password).equals(String.valueOf(confirm))) {
           JOptionPane.showMessageDialog(this, "Passwords do not match.");
        } else {
           // write data to CSV file
           try {
              BufferedWriter writer = new BufferedWriter(new
  FileWriter("users.csv", true));
              writer.write(name + "," + email + "," + String.valueOf(password));
              writer.newLine();
              writer.close();
           } catch (IOException ex) {
              JOptionPane.showMessageDialog(this, "Error writing to file: " +
ex.getMessage());
           }
           JOptionPane.showMessageDialog(this, "Registration successful.");
           nameField.setText("");
           emailField.setText("");
           passwordField.setText("");
           confirmField.setText("");
        }
      } else if (e.getSource() == clearButton) {
        // clear form input
        nameField.setText("");
        emailField.setText("");
```

```
            passwordField.setText("");
        }else if (e.getSource()==cancelButtuon){
            JOptionPane.showMessageDialog(this, "Registration cancelled");
            home hm = new home();
            hm.setVisible(true);
            setVisible(false);
        }
        }
    public static void main(String args[]){
        new register();
    }
}
```

Login.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
public class login extends JFrame implements ActionListener {
   // form components
   private JLabel emailLabel, passwordLabel, image, login;
   private JTextField emailField;
   private JPasswordField passwordField;
   private JButton loginButton, clearButton;
   public login() {
      setTitle("Login Form");
      // create components
      login = new JLabel("Login");
```

```
login.setFont(new Font("Times New Roman",Font.BOLD,26));
emailLabel = new JLabel("Email:");
passwordLabel = new JLabel("Password:");
emailField = new JTextField(20);
passwordField = new JPasswordField(20);
loginButton = new JButton("Login");
clearButton = new JButton("Cancel");
// set layout
setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets(5, 5, 5, 5);
gbc.gridx = 1;
gbc.gridy = 0;
add(login, gbc);
gbc.gridx = 0;
gbc.gridy = 1;
add(emailLabel, gbc);
gbc.gridx = 1;
gbc.gridy = 1;
add(emailField, gbc);
gbc.gridx = 0;
gbc.gridy = 2;
add(passwordLabel, gbc);
gbc.gridx = 1;
gbc.gridy = 2;
add(passwordField, gbc);
gbc.gridx = 0;
```

```
    gbc.gridy = 3;
    add(loginButton, gbc);
    gbc.gridx = 1;
    gbc.gridy = 3;
    add(clearButton,gbc);
    // add action listeners
    loginButton.addActionListener(this);
    clearButton.addActionListener(this);
    // set size and visibility
    setSize(600, 550);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == loginButton) {
        // validate form input
        String email = emailField.getText().trim();
        char[] password = passwordField.getPassword();
        if (email.isEmpty() || password.length == 0) {
            JOptionPane.showMessageDialog(this, "Please enter your email and
  password.");
        } else {
            // check user credentials against CSV file
            boolean found = false;
            try {
                FileReader reader = new FileReader("users.csv");
                BufferedReader bufferedReader = new BufferedReader(reader);
```

```
                String line;
                while ((line = bufferedReader.readLine()) != null) {
                    String[] parts = line.split(",");
                    if (parts[1].equals(email) && parts[2].equals(String.valueOf
(password))) {
                        found = true;
                        break;
                    }
                }
                bufferedReader.close();
                reader.close();
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
            }
            if (found) {
                JOptionPane.showMessageDialog(this, "Login Successfully");
                Main M1 = new Main();
                M1.setVisible(true);
                setVisible(false);
            } else {
                JOptionPane.showMessageDialog(this, "Invalid email or password.");
            }
        }
    } else if (e.getSource() == clearButton) {
        // clear form input
        emailField.setText("");
        passwordField.setText("");
```

```
            home h = new home();
            h.setVisible(true);
            setVisible(false);
        }
    }
}
```