ISSN 2277-2685
IJESR/Jan-Mar. 2026/ Vol-16/Issue-1/377-385
G.Harshith *et. al.*, /International Journal of Engineering & Science Research

Full Length Article

EBMD: Efficient Based Medical Data Share In Database

Gajam.Shekar[1],G.Harshith[2],G.Mohith[3],B.Johnwesly[4]
[1]Assistant Professor ; Department Of Information Technology , Guru Nanak Institutions Technical Campus, Hyderabad, India.
[2,3,4]B.Tech Students; Department Of Information Technology , Guru Nanak Institutions Technical Campus, Hyderabad, India.
Mail Id; godamalaharshith6@gmail.com[2]
Article Received 25-02-2026, Accepted 24-03-2026

ABSTRACT
Cloud computing has significantly reshaped the management of medical data by enabling healthcare organizations to outsource storage and processing to third-party service providers. Although this paradigm offers advantages such as scalability, flexibility, and reduced infrastructure costs, it also raises critical concerns regarding data security and patient privacy, especially when sensitive information is stored on untrusted servers. Existing cryptographic approaches, including searchable encryption, attempt to mitigate these risks but often face challenges such as information leakage, computational inefficiency, and limited scalability in large datasets.
*To overcome these limitations, this paper introduces **EBMD**, a novel secure outsourcing framework that combines an ordered additive secret sharing mechanism with an innovative index permutation strategy. The proposed method ensures both data confidentiality and query privacy by effectively hiding data contents as well as access patterns from potential adversaries. Furthermore, EBMD is designed to be computationally efficient and scalable, making it suitable for large-scale healthcare systems.*
Experimental results demonstrate that EBMD outperforms existing techniques in terms of efficiency, security, and scalability, while maintaining minimal storage overhead. These findings highlight the potential of the proposed approach for secure and practical deployment in cloud-based medical data management systems.

Keywords: *Cloud Computing, Medical Data Security, Patient Privacy, Secure Data Outsourcing, EBMD Framework, Secret Sharing, Additive Secret Sharing, Index Permutation, Searchable Encryption, Data Confidentiality, Query Privacy, Cryptography, Information Leakage Prevention, Scalable Systems, Healthcare Data Management, Secure Cloud Storage, Data Protection, Privacy-Preserving Techniques, Distributed Systems, Secure Query Processing*

Introduction

The digital transformation of healthcare has led to the widespread adoption of Electronic Health Records (EHRs), which play a crucial role in clinical diagnosis, treatment, and patient care. As hospitals and healthcare institutions generate vast volumes of data, many are outsourcing storage and processing to cloud-based servers. While cloud outsourcing reduces infrastructure costs and enhances accessibility, it also introduces significant privacy and security concerns. Unauthorized access to sensitive medical information can result in identity theft, financial loss, and reputational damage. Data breaches may occur due to vulnerabilities in external servers, inadequate security protocols, or sophisticated cyberattacks. Moreover, outsourcing reduces the direct control healthcare providers have over data management, increasing the risk of exposure.

To address these challenges, researchers have proposed several encryption-based solutions. Techniques such as searchable encryption allow secure outsourcing while enabling query functionalities. Attribute-Based Encryption (ABE) enforces fine-grained access control, whereas Proxy Re-Encryption (PRE) facilitates sharing among multiple untrusted domains. Despite their strengths, these approaches face critical limitations: centralized architectures create single points of failure, and access patterns during storage and retrieval can inadvertently leak sensitive information. Countermeasures like oblivious access reduce leakage but often incur high computational costs, limiting practical applicability. With the rise of quantum computing, even encrypted data may become vulnerable to brute-force attacks.

These challenges form a “security trilemma” in medical data outsourcing:

Research Motivation

The exponential growth of digital healthcare and cloud computing has led to massive volumes of

medical data. Outsourcing data storage provides scalability and cost efficiency, but conventional security mechanisms are often inadequate. Existing schemes may leak access patterns, impose high computational overhead, and struggle with large datasets. Additionally, untrusted servers may attempt to infer confidential information from queries or stored data. This motivates the development of a secure, efficient, and scalable outsourcing protocol that preserves privacy while supporting fast data retrieval.

Problem Statement

While cloud computing offers scalable storage for healthcare data, outsourcing sensitive information introduces significant privacy and security challenges. Traditional encryption methods, including searchable encryption, often fail to fully conceal access patterns and suffer from high computational costs and poor scalability. Consequently, there is a pressing need for protocols that ensure confidentiality, hide access patterns, and support efficient retrieval in large-scale medical datasets.

Research Gap

Existing solutions primarily focus on encrypting data content but provide limited protection against access pattern leakage. Inference attacks exploiting query frequency and access sequences remain a significant risk. Additionally, many cryptographic methods impose high computational and communication overhead, limiting their applicability in large databases. Indexing mechanisms are often inefficient and not designed for dynamic data operations. Furthermore, most approaches assume partially trusted servers, leaving fully untrusted environments insufficiently addressed. The integration of secret sharing with efficient index permutation to enhance confidentiality and hide access patterns remains underexplored.

Objectives

This study aims to design and implement a secure medical data outsourcing protocol that:

- Protects sensitive patient data by concealing content and access patterns.
- Reduces computational overhead while maintaining retrieval efficiency.
- Enhances scalability for large-scale medical databases.
- Evaluates the proposed protocol's performance and compares it with existing approaches.

Scope and Limitations

The system focuses on secure outsourcing and retrieval of medical records in cloud environments. Real-time clinical decision systems, interoperability across multiple cloud providers, and hospital system integration are outside the scope. The study primarily targets research and experimental evaluation of secure and scalable cloud-based medical data management.

Significance

The ECMO protocol advances privacy-preserving cloud storage for healthcare by:

- Strengthening patient privacy and regulatory compliance.
- Overcoming limitations of conventional searchable encryption.
- Enabling scalable and efficient retrieval for large medical datasets.
- Providing a foundation for secure cloud-based applications in healthcare and other sensitive domains.

PROJECT DESCRIPTION

Protecting medical records stored in cloud environments has become increasingly important due to the sensitive nature of healthcare information. A straightforward method to ensure confidentiality is to encrypt the data before outsourcing it to cloud servers. However, relying solely on encryption introduces vulnerabilities such as offline brute-force attacks, where an adversary attempts to recover plaintext by trying multiple key combinations. Additionally, the rapid evolution of quantum computing may weaken traditional cryptographic algorithms in the future, further threatening data security. To address these limitations, the proposed system integrates encryption with secret sharing techniques. Instead of storing encrypted data in a single location, the ciphertext is divided into multiple shares and distributed across different servers. This distributed approach ensures that even if one or more servers are compromised, the original plaintext remains protected. The system adopts additive secret sharing, which splits a secret into multiple components that can only be reconstructed when combined. By distributing encrypted shares across multiple storage nodes, the system improves security, eliminates single-point failures, and enhances resilience against unauthorized access.

Methodology

The proposed system follows a structured methodology to ensure secure outsourcing, efficient retrieval, and privacy preservation of medical data in cloud environments. Initially, medical records are collected and preprocessed to ensure uniform formatting and proper organization. Each data item is encrypted before being outsourced, thereby protecting sensitive patient information from unauthorized access. After encryption, an ordered additive secret sharing algorithm is applied to divide the ciphertext into multiple shares. These shares are then distributed among different servers so that no

single entity can reconstruct the original data independently. To improve retrieval efficiency, an indexing mechanism combined with a unique index permutation technique is employed. This method hides the original ordering of stored data and prevents adversaries from learning access patterns during search operations. When an authorized user submits a query, a secure trapdoor is generated and transmitted to the cloud server. The server processes the trapdoor using the permuted index structure and retrieves the corresponding encrypted shares. The shares are returned to the user, who combines them and performs decryption to reconstruct the original medical records. Performance evaluation of the system is conducted by measuring storage efficiency, retrieval time, and scalability under varying dataset sizes. In addition, security analysis is performed to verify resistance against data leakage, brute-force attacks, and access pattern inference. This methodology ensures secure medical data outsourcing while maintaining efficient access and strong privacy guarantees.

Modules Description

The proposed system is divided into several functional modules, including the Health Centre module, ECMO module, Database Service Provider module, Admin module, and ID Generation and Data Retrieval module. These modules collectively ensure efficient management and secure access to healthcare data.

The Health Centre module consists of hospitals, doctors, nurses, and researchers. The hospital acts as the central authority responsible for registration and authentication of users. Doctors must first register and obtain approval from the hospital before accessing the system. Once approved, physicians can treat patients, prescribe medications, and share treatment information with researchers. Nurses are added by the hospital and assist doctors by updating patient records and supporting treatment activities. Researchers register separately, analyze medications, and provide feedback to physicians, facilitating collaborative decision-making in patient care The ECMO module manages patient information and treatment details. Patients register by providing personal and medical data and receive unique login credentials. After authentication, patients can search for hospitals that provide ECMO facilities, view hospital details, and receive prescribed medications from physicians. All treatment-related information is securely stored in the system to ensure continuity of care and accurate documentation.

Input and Output

Each module processes specific inputs and produces corresponding outputs. In the Health Centre module, inputs include hospital login credentials, doctor registration details, nurse addition information, and treatment data. The output includes successful authentication, doctor approval, nurse assignment, and secure storage of treatment updates. In the ECMO module, patient registration, login credentials, hospital search requests, and physician prescriptions serve as inputs, while outputs include patient account creation, hospital listing, and medication updates. The Database Service Provider module receives login credentials, disease information, remedies, and performance metrics as input, and produces outputs such as stored records, analytical reports, and performance evaluations. The Admin module takes login credentials, hospital details, and user information as input, and outputs system configuration updates, user management actions, and access control enforcement.

REQUIREMENTS ENGINEERING

Requirements engineering plays a crucial role in defining the functional and non-functional needs of the proposed system. Although the integration of encryption with additive secret sharing reduces the risk of offline brute-force attacks, information leakage may still occur through access pattern exposure. For instance, if multiple Database Service Providers (DSPs) are compromised and receive shares containing identical identifiers, adversaries may infer that those shares belong to the same medical record. By analyzing the ordering of these shares, attackers may attempt to reconstruct the original information. Such inference attacks can weaken the security guarantees of the secret sharing mechanism and expose sensitive medical data. Therefore, it is essential to design the system requirements carefully to prevent access pattern leakage and ensure robust security.

Hardware Requirements

The hardware requirements specify the minimum infrastructure necessary to develop, deploy, and operate the proposed system efficiently. These requirements ensure that the application runs smoothly under realistic working conditions and supports secure healthcare data processing. The system requires a processor equivalent to Intel Core i3 or higher to handle computation and server operations. A minimum of 4 GB DDR4 RAM is recommended to support application execution and database processing. A 15.6-inch LED monitor is sufficient for user interaction and interface visualization. The storage requirement includes at least 100 GB of hard disk space to accommodate application files, databases, and backups. Additionally, standard input devices such as a keyboard and mouse are required, and a webcam may be included for optional features such as facial authentication or QR-based access.

Software Requirements

The software requirements define the development environment and platform dependencies required to implement the proposed system. The application is

designed using Java EE technologies, including JSP and Servlets, for the front-end interface and server-side logic. The backend database management system used is MySQL version 5.5 or above, which stores medical records and system data. The system operates on Windows 10 or Windows 11 operating systems. Apache Tomcat 7.0 is used as the web server to deploy and run the Java-based application. The system supports modern browsers such as Google Chrome and Mozilla Firefox for user access. Development is carried out using integrated development environments such as Eclipse IDE or IntelliJ IDEA, which provide debugging, deployment, and project management support.

DESIGN ENGINEERING

The design engineering phase defines the structural and behavioral aspects of the proposed ECMO-based healthcare management system. The primary objective of the design is to safeguard medical data outsourced to third-party databases from unauthorized access and potential leakage. Although encryption and additive secret sharing enhance confidentiality, adversaries may still attempt to infer sensitive information through access pattern analysis. Therefore, the system design incorporates multiple layers of security and modular architecture to minimize information exposure. To clearly represent system functionality, structure, and interaction, Unified Modeling Language (UML) diagrams are used. These diagrams provide both static and dynamic views of the system and support implementation and future scalability. The design documentation includes use case diagrams, class diagrams, sequence diagrams, activity diagrams, component diagrams, deployment diagrams, object diagrams, and state diagrams. Each diagram highlights specific aspects such as functional requirements, structural relationships, behavioral flow, and physical deployment.

Use Case Diagram

The use case diagram presents a high-level functional view of the system by identifying key actors and their interactions. The primary actors include Hospital, Doctor, Nurse, Researcher, Patient, Database Service Provider, and Admin. Each actor interacts with the system to perform specific tasks such as registration, login, approval, treatment management, hospital search, medication sharing, data analysis, and user monitoring. The hospital actor manages doctor approvals and nurse assignments, while doctors handle patient treatment and prescription updates. Researchers analyze medication data and provide feedback. Patients search for hospitals and view prescribed treatments. DSP users manage disease records and performance metrics, whereas the admin supervises system configuration and user management. This diagram helps define responsibilities and provides an overview of system functionality.

Class Diagram

The class diagram represents the static structure of the system by identifying major classes, their attributes, and operations. The primary classes include Hospital, Doctor, Patient, Nurse, Researcher, DSP, and Admin. Each class contains attributes such as user ID, password, personal details, and role-specific information. Methods such as login(), approveDoctor(), prescribeMedicine(), updatePatientData(), and generateReports() are defined within the respective classes. Relationships such as associations, dependencies, and aggregations illustrate interactions among classes. For example, a hospital is associated with multiple doctors and nurses, while doctors interact with patients and researchers. This diagram assists in designing database schemas and application logic.

Object Diagram

The object diagram provides a snapshot of system instances at a specific moment. It illustrates how real objects derived from classes interact in real-time scenarios. For instance, a doctor object treating a patient object or a researcher object evaluating medication demonstrates the dynamic relationships between system components. This diagram helps visualize runtime behavior and validates class relationships.

State Chart Diagram

The state chart diagram models the dynamic behavior of objects by representing state transitions. It describes how an entity changes its state in response to events. For example, a doctor progresses through states such as registered, approved, and active, while a patient moves from registered to logged-in and then to treated. This diagram helps in understanding workflow transitions and lifecycle management.

Sequence Diagram

The sequence diagram illustrates interactions among system components in chronological order. It shows message exchanges between objects for processes such as doctor approval, patient login, and medication sharing. The diagram emphasizes the sequence of operations and communication flow, providing clarity on system execution logic.

Collaboration Diagram

The collaboration diagram highlights how system objects cooperate to accomplish tasks. It focuses on communication links between entities such as doctors, researchers, and patients. For example, during medication evaluation, doctors share prescriptions with researchers, who analyze and provide recommendations. This diagram complements the sequence diagram by emphasizing relationships rather than time ordering.

Activity Diagram

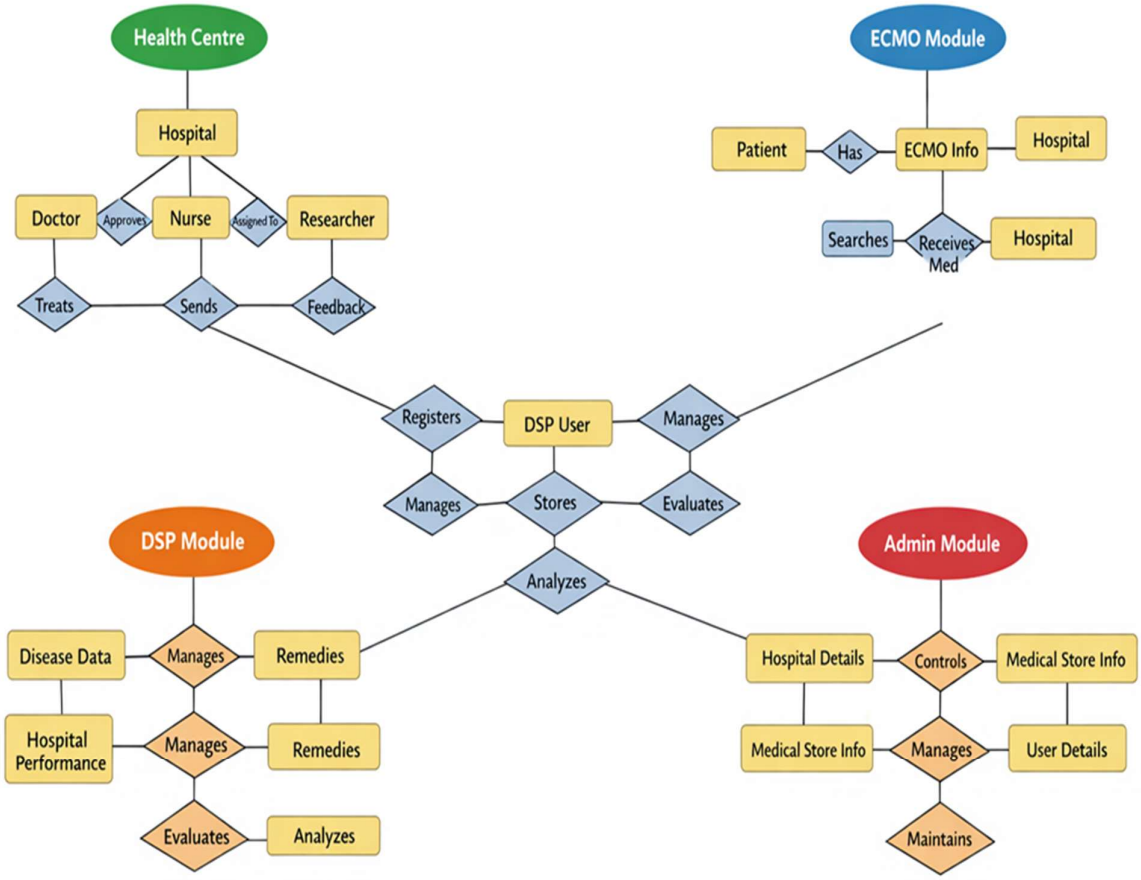
The activity diagram represents the workflow of system processes. It illustrates steps such as patient registration, hospital search, doctor approval, treatment assignment, and medication delivery.

Decision points and parallel processes are included to depict dynamic control flow. This diagram helps visualize operational procedures and improves process understanding.

Component Diagram

The component diagram describes the high-level modular architecture of the healthcare system. The main components include the Health Centre module, ECMO module, Database Service Provider module, Admin module, and Central Database. Each component encapsulates specific functionalities and communicates through defined interfaces. The Health Centre component manages hospital operations, doctor approvals, and treatment workflows. The ECMO component handles patient registration, hospital search, and medication updates. The DSP component manages disease data and performance analytics. The Admin component oversees system configuration and user management. The central database stores all persistent data. This modular structure enhances maintainability and scalability.

Entity Relationship Diagram



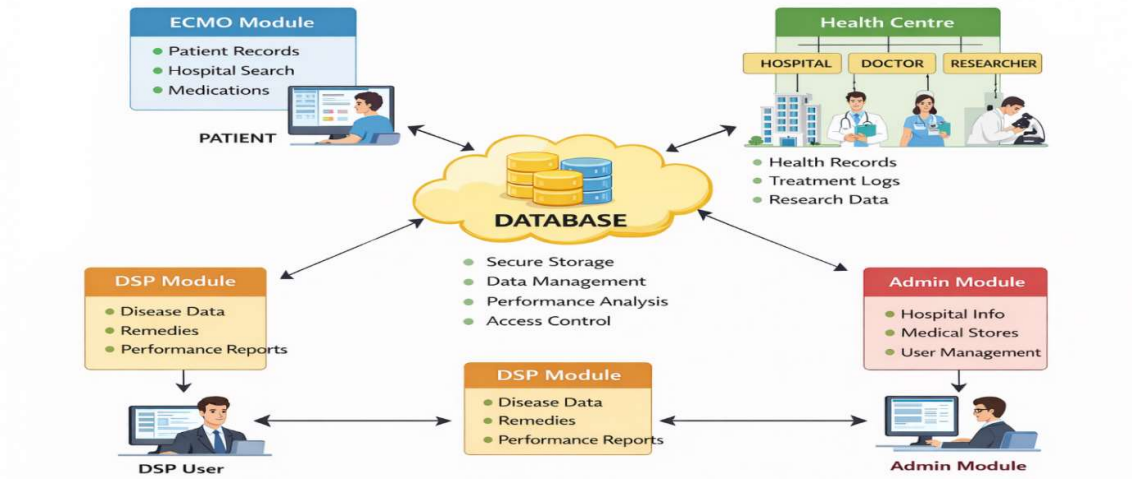
The Entity Relationship (ER) diagram illustrates the logical data structure of the system. Entities include Hospital, Doctor, Nurse, Researcher, Patient, DSP, and Admin. Each entity contains attributes such as identification details, contact information, and role-specific parameters. Relationships define interactions, such as hospitals approving doctors, doctors treating patients, and researchers analyzing medications. The DSP entity stores disease and performance data, while the admin manages all user accounts. This diagram supports database normalization, reduces redundancy, and establishes data dependencies.

Deployment Diagram

The deployment diagram represents the physical architecture of the system. It includes client devices, application server, database server, and cloud storage server. Client devices are used by hospitals, doctors, patients, researchers, DSP users, and admin to access the system. The application server hosts business logic and functional modules. The database server stores persistent data such as user records and

medical information. Cloud storage may be used for large datasets and backups. Secure communication links connect all nodes, ensuring reliable operation and scalability.

System Architecture



The proposed system follows a multi-tier client-server architecture. The client layer consists of end-user devices such as hospital systems, doctor terminals, and patient interfaces. Users access the system through secure login credentials. The application layer contains core modules including Health Centre, ECMO, DSP, and Admin. This layer handles authentication, treatment management, hospital search, and data analytics. The data layer includes the central database and optional cloud storage, where all persistent information is stored securely. Communication between layers is protected using authentication and encryption mechanisms. This layered architecture improves modularity, scalability, and maintainability while ensuring secure handling of healthcare data.

DEVELOPMENT TOOLS

This chapter presents the software technologies and tools employed in the development of the proposed system. The implementation platform selected for this project is Java due to its portability, scalability, and robust support for enterprise applications. The primary technologies used include Java, J2EE, and J2ME. Among these, J2EE was chosen as the core development framework because it provides extensive libraries and components for building distributed, web-based, and database-driven applications. The use of J2EE facilitates modular development, improves maintainability, and supports multi-tier architecture, making it suitable for enterprise-level healthcare management systems.

Features of Java

The Java Framework

Java is a high-level programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as part of the Java platform. The syntax of Java is influenced by C and C++, but it simplifies memory management and provides a cleaner object-oriented model. Java programs are compiled into bytecode, which can be executed on any system equipped with a Java Virtual Machine (JVM), ensuring platform independence. The language is designed to be general-purpose, class-based, concurrent, and object-oriented, while minimizing hardware and operating system dependencies. This design philosophy supports the concept of "write once, run anywhere," which allows developers to deploy applications across different environments without modification. Java technology is widely used across various domains, including desktop applications, web services, enterprise solutions, and mobile platforms. Its reliability, portability, and security make it an ideal choice for network-based applications and distributed computing systems.

Objectives of Java

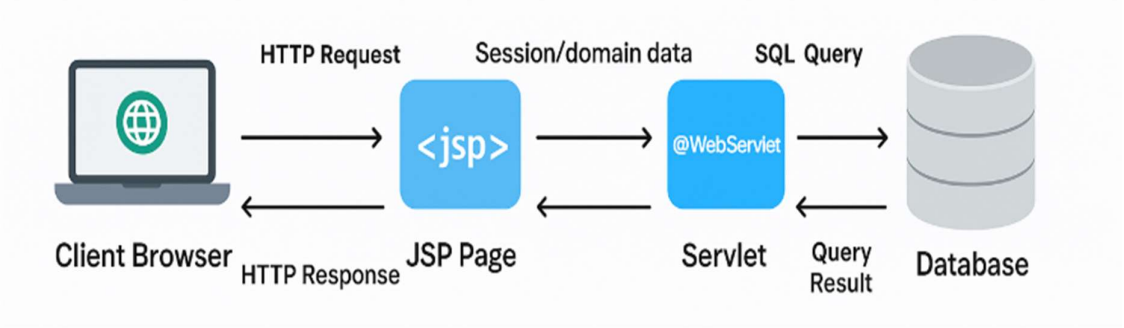
Java has gained widespread acceptance among developers due to its flexibility, portability, and performance. It enables software to be developed on one platform and executed on multiple platforms without requiring recompilation. Java supports the creation of applications that run within web browsers, server-side systems, and distributed enterprise environments. Developers can use Java to build scalable applications for online services, e-commerce platforms, and data processing systems. Additionally, Java allows integration of different services and components to form customized solutions. Its support for embedded systems and mobile devices further extends its applicability.

Java is based on object-oriented programming

Advanced Java Technologies

Advanced Java technologies extend core Java capabilities for developing enterprise-level applications. Servlets are server-side programs that process HTTP requests and generate responses. Java Server Pages (JSP) allow embedding Java code within HTML to create dynamic web pages. Java Database Connectivity (JDBC) provides an interface for connecting Java applications with relational databases. JavaBeans serve as reusable components for encapsulating business logic. The Model-View-Controller (MVC) architecture separates presentation, business logic, and control flow to improve modularity. Filters and listeners manage request processing and application events, while session management techniques maintain user data across multiple interactions.

Apache Tomcat is used as the application server for deploying Java web applications. It supports servlets, JSP, and HTTP protocols, making it suitable for lightweight enterprise deployment. Tomcat simplifies application hosting and integrates with development environments such as Eclipse and IntelliJ IDEA.



SOFTWARE TESTING

Software testing is a critical phase in system development aimed at identifying errors and verifying that the application performs according to specified requirements. The primary objective of testing is to detect faults, weaknesses, and inconsistencies within the software product before deployment. Testing involves executing the software system under controlled conditions to ensure that individual components, integrated modules, and the complete application function correctly. It also verifies that the system satisfies user expectations and operates reliably without unacceptable failures. Various testing approaches are applied, each designed to evaluate specific aspects of system functionality, performance, and reliability.

Types of Tests

Unit Testing

Unit testing focuses on validating individual modules or components of the software. In this stage, test cases are designed to ensure that internal logic and control flow operate correctly. Each unit is tested independently after development and before integration with other modules. This type of testing verifies that inputs produce expected outputs and that decision branches function properly. Unit testing is structural in nature, requiring knowledge of program design and implementation. By examining each component separately, developers can identify and resolve defects at an early stage, improving overall software quality.

Functional Testing

Functional testing evaluates whether system features operate according to business requirements and technical specifications. This testing approach validates input handling, functional operations, and output accuracy. It ensures that valid inputs are accepted, invalid inputs are rejected, and all defined system functions are executed correctly. Additionally, functional testing verifies that interfaces between procedures and subsystems operate as expected. The primary objective is to confirm that the application performs the intended tasks in accordance with user requirements.

System Testing

System testing is performed after integrating all modules to validate the behavior of the complete application. This testing ensures that the fully integrated system satisfies functional and non-functional requirements. It focuses on verifying workflows, data processing, and communication between components. System testing also checks configuration settings and ensures predictable results under various operational conditions. This phase helps confirm that the application is ready for deployment in a real-world environment.

Conclusion

This work introduced ECMO, an efficient and secure protocol designed for confidential outsourcing of medical data. The proposed approach integrates symmetric encryption, ordered additive secret sharing, and a permutation-based mechanism to address major challenges related to healthcare data security. By combining these techniques, ECMO ensures both confidentiality of sensitive

medical information and protection against leakage through access pattern analysis.

A formal security evaluation demonstrates that the protocol maintains strong protection even when an adversary controls up to n−1n - 1n−1 database service providers. This guarantees that confidential information remains secure while preventing unauthorized inference of access patterns. Furthermore, the incorporation of a proxy-based enhancement improves resistance against time-based brute force attacks and strengthens overall system robustness.

Extensive experimental analysis confirms that ECMO performs efficiently under realistic conditions. The results indicate that the protocol scales effectively with increasing data size and maintains acceptable computational overhead. The evaluation also highlights the system's resilience against strong adversarial models and demonstrates its suitability for practical deployment in healthcare environments. Overall, the proposed ECMO framework offers a secure, scalable, and efficient solution for outsourced medical data management.

References

[1] A. Hoerbst and E. Ammenwerth, "Electronic health records," *Methods of Information in Medicine*, vol. 49, no. 4, pp. 320–336, 2010.

[2] A. K. Jha et al., "Use of electronic health records in US hospitals," *New England Journal of Medicine*, vol. 360, no. 16, pp. 1628–1638, 2009.

[3] R. S. Evans, "Electronic health records: Then, now, and in the future," *Yearbook of Medical Informatics*, vol. 25, no. S1, pp. S48–S61, 2016.

[4] B. Brumen et al., "Outsourcing medical data analyses: Can technology overcome legal, privacy, and confidentiality issues," *Journal of Medical Internet Research*, vol. 15, no. 12, 2013.

[5] E. Bertino, B. C. Ooi, Y. Yang, and R. H. Deng, "Privacy and ownership preserving of outsourced medical data," in *Proc. IEEE International Conference on Data Engineering*, 2005, pp. 521–532.

[6] W. Wang, L. Chen, and Q. Zhang, "Outsourcing high-dimensional healthcare data to cloud with personalized privacy preservation," *Computer Networks*, vol. 88, pp. 136–148, 2015.

[7] M. Wang et al., "MedShare: A privacy-preserving medical data sharing system using blockchain," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 438–451, 2023.

[8] K. Fan et al., "MedBlock: Efficient and secure medical data sharing via blockchain," *Journal of Medical Systems*, vol. 42, 2018.

[9] H. Li et al., "A fine-grained privacy protection data aggregation scheme for outsourcing smart grid," *Frontiers of Computer Science*, vol. 17, no. 3, 2023.

[10] C. Zhang et al., "Privacy-preserving multi-server collaborative search in smart healthcare," *Future Generation Computer Systems*, vol. 143, pp. 265–276, 2023.

[11] D. DeFord, "Sustainable digital health demands cybersecurity transformation," *Frontiers of Health Services Management*, vol. 38, no. 3, pp. 31–38, 2022.

[12] H. Li et al., "Secure and efficient dynamic searchable symmetric encryption over medical cloud data," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 484–494, 2020.

[13] L. Xu et al., "Authorized encrypted search for multi-authority medical databases," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 534–546, 2021.

[14] Q. Wang et al., "Searchable encryption with autonomous path delegation function in healthcare cloud," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 879–896, 2023.

[15] J. Liu et al., "Secure sharing of personal health records in cloud computing," *Future Generation Computer Systems*, vol. 52, pp. 67–76, 2015.

[16] C. Guo et al., "Fine-grained database field search using attribute-based encryption for e-healthcare clouds," *Journal of Medical Systems*, vol. 40, 2016.

[17] Y.-H. Park et al., "Secure outsourced blockchain-based medical data sharing system," *Applied Sciences*, vol. 11, no. 20, 2021.

[18] T. Bhatia et al., "Secure incremental proxy re-encryption for e-healthcare data sharing," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, 2020.

[19] W. Zhang et al., "Inference attack-resistant e-healthcare cloud system," *IEEE Transactions on Services Computing*, vol. 14, no. 1, pp. 167–178, 2021.

[20] M. S. Islam et al., "Inference attack against encrypted range queries," in *Proc. ACM Conference on Data and Application Security*, 2014, pp. 235–246.

[21] E. Stefanov et al., "Path ORAM: An extremely simple oblivious RAM protocol," *Journal of the ACM*, vol. 65, no. 4, 2018.

[22] P. Mishra et al., "Oblix: An efficient oblivious search index," in *IEEE Symposium on Security and Privacy*, 2018, pp. 279–296.

[23] A. Ekert and R. Jozsa, "Quantum computation and Shor's factoring algorithm," *Reviews of Modern Physics*, vol. 68, no. 3, 1996.

[24] A. Beimel, "Secret-sharing schemes: A survey," in *International Conference on Coding and Cryptology*, 2011, pp. 11–46.

[25] L. Harn et al., "Threshold changeable secret sharing scheme," *Frontiers of Computer Science*, vol. 16, 2022.

[26] D. Cash et al., "Highly-scalable searchable symmetric encryption," in *Annual Cryptology Conference*, 2013, pp. 353–373.

[27] B. Fuller et al., "Cryptographically protected database search," in *IEEE Symposium on Security and Privacy*, 2017, pp. 172–191.

[28] P. Grubbs et al., "Pancake: Frequency smoothing for encrypted data stores," in *USENIX Security Symposium*, 2020, pp. 2451–2468.