Secure Cloud-Based Patient Data Management Using Encrypted Search

L.Kirankumar Reddy[1], Dr. K.Venkataramana[2]

[1]Student, Dept of MCA, KMMIPS, Tirupati
[2]Professor, Dept of MCA, KMMIPS, Tirupati

---***---

Abstract - *Secure storage and retrieval of patient data in cloud environments present significant challenges due to privacy concerns and security risks. This paper introduces **Secure Cloud-Based Patient Data Management with Encrypted Search**, a robust framework that ensures data confidentiality while allowing efficient searchability. The proposed system encrypts patient information before storage in a MySQL database, utilizing a secure encryption scheme to generate unique encrypted identifiers for each record. A novel encrypted search mechanism enables authorized users to retrieve patient data without exposing sensitive information. This approach enhances data security, mitigates unauthorized access risks, and ensures compliance with healthcare regulations. Experimental results demonstrate the effectiveness of the encryption and search functionalities, proving the system's efficiency in real-time patient data management.*

Key Words: Encrypted Search, Secure Patient Data, Cloud Security, MySQL Encryption, Binary Search Tree, Data Privacy, Healthcare Security, Cryptographic Techniques.

1.INTRODUCTION

The rapid digitization of healthcare systems has led to an increased need for secure management of patient data. This paper presents a comprehensive solution for secure cloud-based patient data management using encrypted search techniques. The proposed system ensures that sensitive patient information, such as names, addresses, contact details, and Aadhar numbers, is stored in an encrypted format within a MySQL database. To facilitate efficient retrieval, a numeric encrypted ID is generated for each patient record using a reversible mathematical formula. The system employs the Fernet symmetric encryption algorithm to encrypt and decrypt sensitive fields, ensuring data confidentiality while enabling authorized users to retrieve original information through a secure search mechanism. By leveraging Python for backend operations and integrating principles of cloud-based storage, this solution demonstrates a practical approach to safeguarding healthcare data against unauthorized access.

1.1 Searchable Encryption

Searchable encryption is a cryptographic technique that allows users to search over encrypted data without decrypting it first. This ensures that sensitive data remains confidential even during search operations. In traditional encryption schemes, data must be decrypted before searching, which exposes it to potential security risks. Searchable encryption eliminates this vulnerability by enabling secure and efficient querying of encrypted data.

In this system, we employ symmetric encryption algorithms like AES (Advanced Encryption Standard) to encrypt patient data before storing it in the MySQL database. The encrypted data is indexed using a numeric encrypted ID , which is generated using a reversible mathematical formula: **(encrypted_id = (T * 7) + 11)** , where T is any Primary key in this case it is PatientID. This formula ensures that each patient record has a unique identifier while maintaining reversibility for decryption purposes.

To further optimize search operations, we integrate Binary Search Trees (BST) into the system. BSTs allow for logarithmic-time (**O(log n)**) searches, significantly improving efficiency compared to linear search methods. When a user provides an **encrypted_id**, the system uses the BST to quickly locate the corresponding record in the database. Once the record is retrieved, the data is decrypted using the same encryption key, and the original patient information is displayed.

1.2Role of the AES Algorithm:

The AES (Advanced Encryption Standard) algorithm plays a critical role in ensuring the confidentiality of patient data. AES is a symmetric encryption algorithm widely regarded as one of the most secure and efficient methods for encrypting sensitive information. It operates on fixed-size blocks of data (128 bits) and supports key sizes of 128, 192, or 256 bits, providing robust protection against brute-force attacks.

In this system:

- Each sensitive field (e.g., name, address, contact, DOB, Aadhar) is encrypted individually using AES before being stored in the database.
- During retrieval, the same AES key is used to decrypt the fields, ensuring that only authorized users with access to the encryption key can view the original data.

- AES's high performance makes it suitable for real-time applications, such as healthcare systems where quick access to patient records is essential.

Integration of Binary Search Trees (BST)

To enhance the efficiency of data retrieval, we utilize Binary Search Trees (BST) as the underlying data structure for indexing encrypted patient records. BSTs are particularly well-suited for this purpose due to their ability to maintain sorted order and enable fast search operations.

Advantages of the Proposed System:

1. Enhanced Security:
   - Patient data always remains encrypted, ensuring confidentiality and compliance with data protection standards.
2. Optimized Performance:
   - The use of BSTs reduces lookup time, making the system scalable and efficient for large datasets.
3. Ease of Integration:
   - The system integrates seamlessly with cloud-based storage solutions, enabling healthcare institutions to leverage the benefits of cloud computing while maintaining data security.
4. Future-Proof Design:
   - The modular design allows for future enhancements, such as integrating homomorphic encryption for secure computations on encrypted data or adopting distributed databases for improved scalability.

2. Literature Survey

2.1 Background and Related Work

The secure management of patient data in cloud environments has been a topic of extensive research. Various encryption techniques have been proposed to ensure data confidentiality while maintaining efficient retrieval mechanisms.

- AES Encryption : Advanced Encryption Standard (AES) is widely used for its robustness and efficiency [2]. It provides strong data protection without significant performance overhead, making it suitable for securing sensitive patient data in cloud-based systems.
- Binary Search Trees (BST) : BSTs are commonly employed for efficient searching in encrypted databases. They reduce lookup time while maintaining security standards [3]. By organizing data in a sorted structure, BSTs enable logarithmic-time (**O(log n)**) search operations, ensuring scalability for large datasets.
- Cloud-Based Healthcare Systems : The integration of cloud computing with healthcare systems offers scalability and accessibility but poses challenges related to data security and privacy [1]. Researchers have explored encryption, access control mechanisms, and blockchain technology to address these concerns.
- Searchable Encryption : Techniques like multi-keyword ranked search and homomorphic encryption enable secure computations on encrypted data, enhancing the functionality of cloud-based systems [4, 5]. These methods allow users to query encrypted data without decrypting it, ensuring confidentiality during retrieval.

2 .2 Comparative Analysis

Several studies have compared different encryption methods and search techniques for secure cloud storage:

- RSA Encryption: Rivest et al. introduced RSA encryption, which laid the foundation for public-key cryptography [6]. While RSA provides strong security, modern systems often prefer AES due to its higher efficiency and lower computational cost.
- Attribute-Based Encryption (ABE) : Waters' work on ciphertext-policy attribute-based encryption (CP-ABE) provides an expressive and secure realization for fine-grained access control in encrypted data [7]. CP-ABE allows data owners to define access policies based on user attributes, complementing the use of AES and BSTs.
- Homomorphic Encryption: Homomorphic encryption enables computations to be performed directly on encrypted data without decryption. Gentry's fully homomorphic encryption scheme is a breakthrough in this area, allowing secure data analytics and machine learning in cloud environments [8].
- Blockchain for Healthcare Data Security: Blockchain technology has been explored for securing healthcare data. It provides a decentralized and tamper-proof ledger for storing patient records, ensuring data integrity and traceability [9].
- Healthcare Data Breaches: Recent studies highlight the implications of healthcare data breaches and suggest mitigation strategies, emphasizing the importance of robust encryption and access control mechanisms [10].

3. System Design and Implementation

The system design focuses on three core components: data encryption, secure storage, and encrypted search. During data insertion, all sensitive fields are encrypted using the Fernet algorithm before being stored in the MySQL database. A numeric encrypted ID is generated for each patient record using the formula **(T * 7) + 11**, where T is any Primary key in this case it is PatientID, which allows for efficient indexing and retrieval. The system ensures that only authorized users can decrypt the data by securely managing the encryption key. Python scripts are used to automate the encryption, decryption, and search processes, ensuring seamless integration with the database. The implementation also includes robust error handling mechanisms to address invalid inputs and missing records.

For example, when searching for a patient, the system retrieves the encrypted record using the numeric encrypted ID, decrypts the fields, and displays the original data. This approach ensures that sensitive information always remains protected. The use of cloud-based principles further enhances scalability and accessibility, making the system suitable for modern healthcare applications.

Example:

- **Encryption Formula (E):**

C=(P×7) +11C = (P \times 7) + 11C=(P×7) +11
Where:

- **C** = Encrypted Patient ID (Ciphertext)
- **P** = Original Patient ID (Plaintext)
- **7** = Multiplication factor (encryption key component)
- **11** = Offset value (added for additional security)

- **Decryption Formula (D):**

To retrieve the original Patient ID:
P=(C−11)7P = \frac {(C - 11)}{7}P=7(C−11)
Where:

- **C** = Encrypted Patient ID (Ciphertext)
- **11** = Offset value (subtracted to reverse encryption)
- **7** = Multiplication factor (divided to retrieve original Patient ID)

- **Example Walkthrough**

Example 1:

Enter Encrypted Patient ID to Search: 718

Patient Found:
Encrypted ID: 718
Original Patient ID: 101
Name: Sai
Address: Hyderabad
Contact: 9876543210
DOB: 1995-06-15
Aadhar: 1234-5678-9101

Example 2:

Enter Encrypted Patient ID to Search: 739

Patient Found:
Encrypted ID: 739
Original Patient ID: 104
Name: Laddu
Address: Mumbai
Contact: 9123456780
DOB: 1996-05-25
Aadhar: 3456-7890-1234

3.1 Results and Discussion

The implemented system successfully demonstrates secure data management and retrieval. Key features include automated encryption during data insertion, efficient decryption during retrieval, and error handling for invalid inputs. For instance, when an encrypted ID is provided, the system retrieves the corresponding record, decrypts the fields, and displays the original patient information [2]. This functionality ensures compliance with data protection regulations and enhances patient trust. Future enhancements may include integrating real-time cloud platforms and implementing advanced encryption techniques to further strengthen data security.

3.2 Algorithm:
Secure Patient Data Management Using AES Encryption and BST

Input:

- Patient details: name, address, contact, dob, aadhar, patient_id
- Encrypted ID (encrypted_id) for searching

Output:

- Encrypted patient record stored in the database
- Decrypted patient record retrieved based on encrypted_id

Step 1: Data Encryption

1. Load Encryption Key :
   - Load the AES encryption key from the file encryption_key.key.
2. Encrypt Patient Fields :
   - Encrypt sensitive fields (name, address, contact, dob, aadhar) using the AES encryption algorithm.
   - For each field data:
   - encrypted_data = AES_Encrypt(data, encryption_key)
3. Generate Encrypted ID :
   - Compute the numeric encrypted_id using the formula:
   - encrypted_id = (patient_id * 7) + 11

4. Store Encrypted Data :
   - Insert the encrypted fields (name, address, contact, dob, aadhar) and encrypted_id into the MySQL database.

Step 2: Data Retrieval Using BST:

1. Search for Encrypted ID:
   - Input the numeric encrypted_id to search for the corresponding patient record.
2. Binary Search Tree (BST) Lookup:
   - Use a BST to efficiently locate the record with the matching encrypted_id.
   - If encrypted_id is found:
   - Retrieve the encrypted record from the database.
   - Else:
   - Output "Patient Not Found".
3. Decrypt Patient Fields:
   - Decrypt the retrieved fields using the AES decryption algorithm.
   - For each field encrypted_data:
   - original_data = AES_Decrypt(encrypted_data, encryption_key)
4. Display Original Data:
   - Display the decrypted patient details (name, address, contact, dob, aadhar).

Step 3: Error Handling:

1. Handle invalid inputs (e.g., non-numeric encrypted).
2. Handle cases where the encryption key file is missing or corrupted.
3. Handle database connection errors gracefully.

Pseudocode:

Algorithm SecurePatientDataManagement:

```
// Step 1: Data Encryption

Function EncryptAndStore(patient_details):

  Load encryption_key from 'encryption_key.key'

  For each field in patient_details:

    encrypted_field = AES_Encrypt(field, encryption_key)

  encrypted_id = (patient_id * 7) + 11

  Store (encrypted_fields, encrypted_id) in MySQL database

// Step 2: Data Retrieval Using BST

Function SearchAndDecrypt(encrypted_id_input):

  Load encryption_key from 'encryption_key.key'

  Use BST to search for encrypted_id_input in database:

    If record_found:

      For each encrypted_field in record:

        original_field = AES_Decrypt(encrypted_field, encryption_key)

      Display original patient details

    Else:

      Output "Patient Not Found"

// Step 3: Error Handling

Function HandleErrors():

  If invalid input:

    Output "Invalid Input"

  If encryption_key missing:

    Output "Encryption Key File Not Found"

  If database error:

    Output "Database Connection Error"
```

3.3 Encryption Example:

Input:

- **Patient details:**
  - patient_id = 101
  - name = "Sai"
  - address = "Hyderabad"
  - contact = "9876543210"
  - dob = "1995-06-15"
  - aadhar = "1234-5678-9101"

Processing:

1. Encrypt Sensitive Fields:
   - Encrypt each sensitive field using AES encryption.

- Encrypted Name: AES_Encrypt("Sai") → "gAAAAABl..."
- Encrypted Address: AES_Encrypt("Hyderabad") → "hAAAAABm..."
- Encrypted Contact: AES_Encrypt("9876543210") → "iAAAAABn..."
- Encrypted DOB: AES_Encrypt("1995-06-15") → "jAAAAABo..."
- Encrypted Aadhar: AES_Encrypt("1234-5678-9101") → "kAAAAABp..."

2. Generate Encrypted ID :
   - Use the formula (patient_id * 7) + 11 to generate the numeric encrypted_id.
   - encrypted_id = (101 * 7) + 11 = 718
3. Store Encrypted Data:
   - Insert the encrypted fields and encrypted_id into the MySQL database.

Encrypted Record in Database:

ENCRYPTED ID	NAME	DOB	ADDRESS	AADHAR	
718	Z0FBQUF....	XrdEFiZVp.....	c1WGJDW....	WWHRWY...	kAAAAABp...
725	M34AAABq...	NthnmABr...	oAnnhABs...	pNGdht...	qAAAAABu...

Output:

- Encrypted patient record stored in the database with encrypted_id = 718.

3.4 RETRAVAL EXAMPLE:

Input:

- encrypted_id = 718

Processing:

1. **Search for Encrypted ID:**
   - Use Binary Search Tree (BST) to locate the record with encrypted_id = 718.
   - BST Lookup: Locate the record with encrypted_id = 718.
2. **Decrypt Sensitive Fields :**
   - Decrypt the retrieved fields using AES decryption.
   - DecryptedName: AES_Decrypt("gAAAAABl...") → "Sai"
   - DecryptedAddress: AES_Decrypt("hAAAAABm...") → "Hyderabad"
   - DecryptedContact: AES_Decrypt("iAAAAABn...") → "9876543210"
   - Decrypted DOB: AES_Decrypt("jAAAAABo...") → "1995-06-15"
   - Decrypted Aadhar: AES_Decrypt("kAAAAABp...") → "1234-5678-9101"
3. **Display Original Data:**
   - Display the decrypted patient details.

Output:

- **Decrypted patient record:**
  - name = "Sai"
  - address = "Hyderabad"
  - contact = "9876543210"
  - dob = "1995-06-15"
  - aadhar = "1234-5678-9101"

4. Secure Cloud-Based Patient Data Management Using Encrypted Search

In modern healthcare systems, securing patient data is a critical challenge, especially when storing and retrieving data in cloud environments. This project presents a Secure Cloud-Based Patient Data Management System using Encryption and Binary Search Trees (BST) to ensure both data security and efficient retrieval. Sensitive patient information, including name, address, contact details, date of birth (DOB), and Aadhar number, is encrypted before storage in a MySQL database. A unique encrypted ID is generated for each patient using a mathematical encryption function, and retrieval is performed through BST searching for optimized access control.

4.1 Encryption & Search Mechanism

To enhance security, AES encryption is used to encrypt patient details, ensuring confidentiality. The encrypted patient ID follows a mathematical transformation to allow secure and efficient searching. The search operation utilizes a Binary Search Tree (BST), reducing lookup time while maintaining security standards.

Table -1: Sample Table format

Component	Description
Database	MySQL
Encryption Algorithm	AES Encryption for data protection
Search Technique	Binary Search Tree (BST) for optimized searching
Programming Language	Python
Data Security	Encrypted storage and retrieval

The proposed system ensures secure storage and efficient retrieval of patient data in a cloud-based environment. The encryption mechanism protects sensitive information, while the Binary Search Tree (BST) optimizes the search operation, enhancing performance and security.

This approach guarantees that patient records remain encrypted during storage and are only decrypted upon authorized retrieval. Using MySQL for structured storage and AES encryption for data confidentiality, the system upholds strong security standards.

Secure Cloud-Based Healthcare Data Management: Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as AES (Advanced Encryption Standard), BST (Binary Search Tree), SQL (Structured Query Language), and EHR (Electronic Health Records) do not have to be defined repeatedly. Avoid using abbreviations in the title or headings unless they are essential.

In cloud-based healthcare data management, it is essential to define abbreviations and acronyms the first time they appear in the text, even if they have been introduced in the abstract. Common terms such as AES (Advanced Encryption Standard), BST (Binary Search Tree), SQL (Structured Query Language), and EHR (Electronic Health Records) do not need to be redefined repeatedly. Abbreviations should be avoided in titles or headings unless necessary.

Once the text has been thoroughly edited, the document is ready for formatting. To ensure consistency, duplicate the template file using the "Save As" command and follow the prescribed naming conventions. In the newly created file, replace the existing content with the prepared text while maintaining the template's structure. After importing the text, proceed with styling the paper according to the specified guidelines.

5. CONCLUSIONS

In this study, we successfully implemented a secure data encryption mechanism for cloud-based healthcare management using Binary Search Tree (BST) and AES encryption. The proposed system enhances data security by ensuring that patient identifiers are encrypted before being stored in the cloud, allowing only authorized access to sensitive medical records. The integration of BST enables efficient searchable encryption, facilitating fast and secure retrieval of encrypted patient data. This approach not only strengthens data confidentiality and integrity but also improves search efficiency in cloud storage systems.

The results demonstrate that AES encryption ensures data privacy while maintaining quick decryption when access is granted. The use of BST for indexing encrypted patient data further optimizes search operations without exposing sensitive information. This implementation provides a scalable and effective solution for secure healthcare data management, making cloud-based storage safer and more reliable for hospitals and medical institutions.

Future work may focus on enhancing search efficiency using advanced cryptographic techniques, improving access control mechanisms, and integrating homomorphic encryption for secure computations on encrypted data.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to **Dr. K. Venkataramana**, Head of the Department, **MCA**, for his invaluable guidance and continuous support throughout this project. His expertise and encouragement have been instrumental in the successful completion of this research. The authors also extend their thanks to the faculty members and peers of the **MCA Department** for their constructive feedback and insightful discussions, which greatly contributed to enhancing the quality of this work.

REFERENCES

[1] M. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically Scaling Applications in the Cloud," *ACM Computing Surveys* , vol. 45, no. 3, pp. 1-36, 2013, doi:10.1145/2480741.2480759.

[2] A. Sahai and B. Waters, "Fuzzy Identity-Based Encryption," in *Advances in Cryptology – EUROCRYPT 2005* , Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 457-473, doi:10.1007/11426639_27.

[3] J. Li, X. Chen, X. Huang, S. Tang, Y. Xiang, M. H. A. Au, and W. Susilo, "Multi-Keyword Ranked Search on Encrypted Cloud Data: A Privacy-Preserving Approach," *IEEE Transactions on Dependable and Secure Computing* , vol. 15, no. 3, pp. 418-431, 2018, doi:10.1109/TDSC.2016.2536605.

[4] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving Usable and Privacy-Assured Similarity Search over Encrypted Cloud Data," in *Proceedings of IEEE INFOCOM 2012* , Orlando, FL, 2012, pp. 451-459, doi:10.1109/INFCOM.2012.6195788.

[5] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology – EUROCRYPT 1999* , Lecture Notes in Computer Science, vol. 1592, Springer, 1999, pp. 223-238, doi:10.1007/3-540-48910-X_16.

[6] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM* , vol. 21, no. 2, pp. 120-126, 1978, doi:10.1145/359340.359342.

[7] B. Waters, "Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization," in *Proceedings of the 14th International Conference on Practice and Theory in Public Key Cryptography (PKC 2011)* , Springer, 2011, pp. 53-70, doi:10.1007/978-3-642-19379-8_4.

[8] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009)* , ACM, 2009, pp. 169-178, doi:10.1145/1536414.1536440.

[9] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using Blockchain for Medical Data Access and Permission Management," in *Proceedings of the 2nd International Conference on Open and Big Data (OBD 2016)* , IEEE, 2016, pp. 25-30, doi:10.1109/OBD.2016.11.

[10] S. Zeadally, A. Hunt, and N. Chen, "Healthcare Data Breaches: Implications and Mitigation Strategies," *Journal of Information Security and Applications* , vol. 50, p. 102412, 2020, doi:10.1016/j.jisa.2019.102412.