VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI

A Project Work Report on

ARION Data Security System

Submitted by

Basavaraj Mutnal **2JR22CS401**

In partial fulfillment of the requirement for the award of the

Bachelor of Engineering

In

Computer Science & Engineering

Under the Guidance of

Prof. Bharateesh N. Fadanis

Asst. Professor

Subject Code: 21CSP76

Jain College of Engineering & Research, Belagavi

Department of Computer Science & Engineering

Academic Year 2024 - 2025

Jain College of Engineering & Research, Belagavi

(Approved by AICTE , New Delhi ,Affiliated to VTU Belagavi & Recognized by Govt. of Karnataka)

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the report on **"ARION Data Security System"** is a bonafied work carried out by **Basavaraj Mutnal (2JR22CS401)** in partial fulfillment of VII semester, to award the degree in Computer Science & Engineering of the Visvesvaraya Technological University, it is witnessed that all corrections/suggestions indicated have been incorporated in the report.

The report has been approved as it satisfies all the academic requirements in respect of report as prescribed for the degree in engineering

Guide	HOD	Principal
(Prof. Bharateesh N. Fadnis)	(Dr. Pritam Dhumale)	(Dr. S. V Gorabal)

Name of the Examiners **Signature with Date**

1.

2.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible and whose constant guidance and encouragement crowned our efforts with success. I consider it my privilege to express the voice of gratitude and respect to all those who guided and inspired me in completion of this project.

First and foremost, I ought to pay due regards to my renowned institution, which provided me a platform and an opportunity for carrying out this project work.

My sincere gratitude to our beloved Principal, **Dr. S. V. Gorabal,** for providing an ideal atmosphere to pursue my objectives under his able administration.

I am also thankful to **Dr. Virupaxi B. Dalal,** Dean Academics, who has given valuable suggestions during the work along with his moral support and encouragement.

I am thankful to **Dr. Pritam Dhumale**, HOD, Dept. of Computer Science & Engineering, for motivating me and also allowing me to use the facilities of the department to complete this Project Work.

I express my sincere thanks to **Prof. Veena B.M**. and **Prof. Abhilasha J.**, Project Coordinator for having supported the work related to this Project. Their contributions and technical support in preparing this report are greatly acknowledged.

I express my sincere thanks to my guide **Prof. Bharteesh N. Fadanis**, Assistant Professor, Dept. of Computer Science & Engineering for their constant co-operation, support, and valuable suggestions.

I would like to thank my parents for their constant moral support throughout the completion of this Project Work. Finally, last but not the least, I would like to extend our deep sense of gratitude to my friends who always inspired me and encouraged me throughout the completion of this Project Work.

1. Basavaraj Mutnal

Declaration

I, the members of the project team, studying in the VII semester of Computer Science & Engineering, Jain College of Engineering and Research, hereby declare that the entire project entitled **"ARION Data Security System"** has been carried out by me independently under the guidance of **Prof. Bharateesh N. Fadanis**, Department of Computer Science & Engineering, Jain College of Engineering and Research. This Project work is submitted to the Visvesvaraya Technological University, Belagavi, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering.**

This dissertation has not been submitted previously for the award of any other degree to any other institution or university.

Date:

Place: Belagavi

USN	NAME	SIGNATURE
2JR22CS401	**Basavaraj Mutnal**	

Abstract

In the digital era, where web applications have become integral to daily transactions, safeguarding sensitive user data particularly alphanumeric identifiers representing sensitive information demands advanced and reliable security measures. This paper introduces a state of the art secure platform built on robust cryptographic foundations, designed to protect user data through a multi-layered defense strategy. By employing a combination of cutting-edge encryption algorithms at various stages of data transmission and storage, the system ensures unparalleled security. Key innovations include dynamic key generation, the seamless integration of symmetric and asymmetric encryption, and the application of hash functions and digital signatures. To address critical challenges such as secure key distribution, algorithm resilience against sophisticated attacks and performance optimization, the platform leverages advanced techniques like hardware acceleration and secure multiparty computation. This comprehensive approach not only fortifies user data but also ensures confidentiality, integrity and availability throughout its lifecycle. With ongoing advancements in cryptographic research, this platform represents a significant leap forward in creating secure and efficient solutions for data protection in web applications.

Keywords: Cryptography, Advanced Data Security, Multi-Layered Encryption, Key Generation, Symmetric and Asymmetric Encryption, Hash Functions, Digital Signatures, Secure Protocols, Performance Optimization.

CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

An extensive exploration of cryptographic data security systems, highlighting their pivotal role in safeguarding sensitive user data, particularly within web applications. Through the integration of advanced cryptography techniques, such as multiple encryption algorithms and secure protocols, a robust platform has been crafted to protect critical information like alphanumeric characters. Key topics covered include key generation, encryption algorithm selection, protocol implementation and their seamless integration into data storage and transmission processes. Techniques like symmetric and asymmetric encryption, hash functions and digital signatures are scrutinized for their contributions to ensuring data confidentiality, integrity and availability. To further enhance security, a 7-layer encryption and decryption mechanism has been introduced, providing an additional fortified defense against potential threats. The overview addresses challenges such as key management, algorithm resilience and performance optimization, laying the groundwork for insights into ongoing advancements aimed at fortifying cryptographic systems for enhanced security across diverse application landscapes.

Central to this exploration is the integration of multiple encryption algorithms at various stages of data transmission and storage. These include symmetric encryption for its speed and efficiency, asymmetric encryption for secure key exchange and hash functions for ensuring data integrity. Digital signatures also play a vital role in authenticating the identity of communicating parties, thus enhancing the overall trustworthiness of the system. These techniques collectively contribute to building a multi-layered defense mechanism that not only secures data at rest but also protects it during transmission, effectively mitigating risks such as eavesdropping, tampering and unauthorized access.

1.2 Introduction

In an era increasingly reliant on digital technologies, safeguarding sensitive user data stands as a paramount concern, particularly within the realm of web applications where the specter of data breaches looms large. Cryptography-based systems emerge as a pivotal advancement in this arena, furnishing robust mechanisms to fortify data files through encryption techniques and protocols. This paper embarks on a journey into the development of a secure platform meticulously crafted to shield sensitive user data,

accentuating the foundational role of cryptographic principles in its architectural framework.

By deploying multiple encryption algorithms at various junctures of data transmission and storage, and incorporating a sophisticated 7-layer encryption and decryption mechanism this platform orchestrates end-to-end encryption, erecting barriers against interception and unauthorized access. The introduction sets the stage for an exhaustive exploration of cryptographic data security systems, addressing pivotal methodologies, formidable challenges, and ongoing advancements aimed at augmenting the resilience, efficiency, and usability of these systems across an array of applications. This paper aims to provide a comprehensive overview of the techniques and challenges associated with developing secure encryption systems for web applications. By examining the fundamental steps involved in constructing such systems, from frontend encryption to backend decryption and re-encryption, and secure key management, we seek to elucidate the intricate processes and methodologies underpinning their implementation.

We delve into the methodologies and algorithms commonly utilized in each step, exploring their strengths, limitations, and suitability for diverse application scenarios. We also address the unique challenges posed by real-time processing requirements, including computational efficiency, scalability, and adaptability to evolving encryption standards and protocols.In the realm of secure web application development, safeguarding against eavesdropping and tampering.

End-to-end encryption extends this protection to data at rest, ensuring sensitive information remains unreadable even if intercepted. Key management practices, such as using asymmetric encryption for secure exchange of symmetric keys, bolster security by limiting exposure. Challenges include balancing cryptographic strength with performance demands, ensuring compatibility across platforms, and staying abreast of evolving standards like quantum-resistant encryption. Implementing robust encryption methodologies from frontend to backend ensures comprehensive data security, pivotal in mitigating risks of unauthorized access and data breaches.

Securing sensitive user data, such as alphanumeric characters, in web applications is of paramount importance in today's digital landscape. Advances in cryptography-based systems are critical in safeguarding this data, utilizing multiple encryption algorithms during transmission and storage to ensure robust protection. This paper explores the methodologies and challenges of cryptographic data security systems, including key generation, encryption algorithm selection, protocol implementation, and integration into

data processes. Techniques such as symmetric and asymmetric encryption, hash functions, and digital signatures are employed to create a multi-layered defense. Addressing challenges like key distribution, algorithm robustness, and performance efficiency is crucial to maintaining data confidentiality, integrity and availability.

Cryptography-based systems have emerged as a fundamental pillar in safeguarding digital information. These systems employ advanced mathematical techniques and protocols to encode data, rendering it inaccessible to unauthorized parties. By utilizing multiple encryption algorithms, cryptographic systems not only secure data during transmission but also protect it at rest, ensuring end-to-end security. This paper focuses on the development of a secure platform designed to protect user data through a multi-layered cryptographic approach, leveraging a 7-layer encryption and decryption mechanism to bolster data security.

The proposed platform addresses critical aspects of data security, including secure key management, algorithm selection, and real-time encryption processes. It incorporates both symmetric and asymmetric encryption methods, hash functions, and digital signatures to ensure data confidentiality, integrity, and authenticity. Challenges in developing secure systems include achieving a balance between computational efficiency and cryptographic strength, ensuring compatibility across platforms, and adapting to emerging threats such as quantum computing. Additionally, real-time data processing demands scalability and performance optimization to maintain a seamless user experience while upholding stringent security standards. cryptographic systems, exploring the interplay between encryption techniques, secure key exchange and data integration processes. By providing a comprehensive overview of the advancements, challenges, and methodologies in cryptographic data security, this research aims to contribute to the development of resilient, efficient, and adaptable systems for web applications in an increasingly interconnected world.

Chapter 2

Literature Survey

In paper [1], This paper presents a robust system combining RSA encryption and tokenization to secure credit card information during transactions. The process involves customer data submission through a merchant module, followed by validation of card numbers using the Luhn algorithm. Sensitive data is encrypted using RSA, decrypted, and then a unique token is generated for each transaction. Tokens are securely stored in a token vault and sent back to customers for confirmation. The system enhances transaction security, ensures non-repudiation, and provides a user-friendly interface. While the system demonstrates practical applications of encryption and tokenization, the paper does not explore scalability or computational costs in detail. This study proposes a secure system combining RSA encryption and tokenization to protect credit card data during online transactions. The process starts with customer information being submitted through a merchant module. Card numbers are validated using the Luhn algorithm, ensuring that only legitimate numbers are processed. Once validated, the sensitive data is encrypted using RSA encryption, which leverages a public-private key pair for secure communication. The encrypted data is then decrypted, and a unique token is generated to represent the transaction. These tokens are stored securely in a token vault and shared with customers for transaction confirmation.

In paper [2], This paper critically examines the vulnerabilities of the RSA encryption algorithm, widely used in secure communication systems. It highlights weaknesses, including susceptibility to advancements in quantum computing and efficient prime factorization techniques. The authors propose alternative algorithms such as elliptic curve cryptography (ECC), quantum cryptography (BB84), and post-quantum cryptography, which offer enhanced security and efficiency. Additionally, cryptographic hash and lattice-based methods are explored as future-proof solutions. The study emphasizes the need for transitioning to quantum-resistant encryption to address the growing threat posed by quantum computing. However, advancements in quantum computing and more efficient factorization algorithms have exposed RSA's weaknesses. The paper discusses the potential for quantum computers to break RSA encryption through Shor’s algorithm, which can efficiently factorize large numbers. Additionally, side-channel attacks, which exploit physical characteristics of RSA implementations, pose significant risks. The study

underscores the urgency of transitioning to quantum-resistant algorithms, given the accelerating development of quantum technologies. While the paper effectively identifies RSA's shortcomings, it emphasizes the need for continued research to standardize and implement robust alternatives that ensure long-term data security.

In paper [3], This research introduces a unique encryption method using the Devanagari script and the ancient Indian Bhuta Sankya Paddati system. Messages are transformed into decimal numbers based on the system, reversed, and mapped back to words in the Devanagari script. Decryption involves reversing the process to retrieve the original message. The method leverages cultural and linguistic obfuscation, adding a layer of security through its unconventional approach. While innovative, its application is limited to regions and systems familiar with the Devanagari script, and its computational complexity for larger texts is yet to be explored. The decryption process involves reversing these steps to reconstruct the original message. This language-based encryption method offers an unconventional approach to securing information, making it particularly suitable for applications where cultural context is an asset. However, its reliance on the Devanagari script and familiarity with the Bhuta Sankya Paddati system limits its applicability to specific regions or linguistic groups. Furthermore, the computational complexity of encrypting large volumes of data or integrating this method into modern systems remains unexplored. Despite these limitations, the paper highlights the potential of leveraging cultural and linguistic elements for encryption. This method demonstrates a novel perspective on cryptographic practices, blending traditional knowledge with modern security needs, and offers a foundation for further exploration in culturally driven encryption techniques.

In paper [4], This research proposes the HBDaSeC model, which integrates data fragmentation and hybrid encryption for enhanced cloud storage security. The model divides data into smaller fragments, encrypting each separately, and allows computations on encrypted data without decryption. This approach improves data confidentiality and integrity while protecting against unauthorized access and data breaches. While the HBDaSeC model offers significant advantages, such as better scalability and security, it introduces overhead due to the fragmentation and encryption processes. The paper highlights the importance of combining multiple encryption techniques to address cloud storage challenges. Additionally, the model supports computations on encrypted data without decryption, preserving privacy and integrity during processing. The paper outlines

the architecture and functionalities of the HBDaSeC model, emphasizing its ability to protect against unauthorized access and data breaches. Key benefits include improved data confidentiality, better scalability, and resilience to attacks targeting specific data segments. However, the model introduces additional computational and storage overhead due to fragmentation and multiple encryption layers.

In paper [5], This paper examines potential vulnerabilities in the AES encryption algorithm, a cornerstone of modern cryptography. It identifies exploitable patterns such as zero-points and fixed-points, which can reveal critical information about encryption keys. The study highlights that sending data streams with identical characters can compromise AES security. While AES remains highly trusted, the paper calls attention to its susceptibility to specific attack vectors like differential cryptanalysis and side-channel attacks. Recommendations include avoiding predictable inputs and implementing robust padding schemes to mitigate risks. These vulnerabilities could potentially expose encryption keys, especially when identical data streams are repeatedly processed. The authors also discuss differential and linear cryptanalysis, emphasizing how predictable inputs like constant character strings can weaken AES's security. While AES is generally considered robust, the paper highlights the importance of input randomness and secure implementation practices to prevent attacks. The authors recommend strategies such as dynamic key management and robust padding schemes to mitigate identified risks. The study contributes to the understanding of AES's limitations, particularly in scenarios where predictable or low-entropy data is processed.

In paper [6], The paper discusses the vulnerabilities of RSA encryption, a widely used secure communication method based on asymmetric cryptography and difficult prime number factorization. It highlights the vulnerabilities of RSA, which are being addressed by developing more efficient algorithms for prime factorization and quantum computing. Alternative encryption algorithms include elliptic curve cryptography (ECC), quantum cryptography (BB84), post-quantum cryptography, cryptographic hash algorithms, and lattice-based cryptography. This study introduces a novel image encryption algorithm that leverages two-dimensional chaotic maps to enhance security. The proposed method employs a unique picture segmentation technique based on block structures, which, when combined with chaotic mapping, ensures a high level of unpredictability in the encryption process. The algorithm's strength lies in its ability to produce encrypted images that are highly resistant to common cryptographic attacks, including brute force and statistical

analyses. Experimental results demonstrate the algorithm's efficiency and robustness, making it a promising candidate for secure image transmission and storage applications.

In paper [7], This research conducts a comprehensive comparative analysis of five symmetric key ciphers AES, Blowfish, Twofish, Salsa20, and ChaCha20 in the context of image encryption. By evaluating parameters such as encryption and decryption times, as well as throughput speeds, the study aims to identify the most efficient algorithm for handling large datasets. The methodology involves testing each algorithm on various image types using Java as the primary platform. Findings reveal that ChaCha20 exhibits superior performance, with encryption and decryption times over 50% faster than some other algorithms, while Twofish demonstrates lower throughput during testing. These insights provide valuable guidance for selecting appropriate encryption techniques in applications requiring efficient and secure image data handling. The decryption process involves reversing these steps to reconstruct the original message. This language-based encryption method offers an unconventional approach to securing information, making it particularly suitable for applications where cultural context is an asset. However, its reliance on the Devanagari script and familiarity with the Bhuta Sankya Paddati system limits its applicability to specific regions or linguistic groups. Furthermore, the computational complexity of encrypting large volumes of data or integrating this method into modern systems remains unexplored. Despite these limitations, the paper highlights the potential of leveraging cultural and linguistic elements for encryption.

In paper [8], This paper proposes enhancements to the Advanced Encryption Standard (AES) by introducing a 3-dimensional dynamic S-box and a novel key generation matrix. The traditional AES algorithm is augmented to increase security and resistance against cryptographic attacks. The 3D S-box adds complexity to the substitution process, while the new key generation matrix ensures a more dynamic and unpredictable key schedule. Key sensitivity analysis indicates that even minor variations in encryption keys can significantly alter the decryption process, ensuring robustness against unauthorized access. Statistical analyses, including entropy and correlation assessments, confirm the method's high security and effectiveness. Experimental results indicate that these modifications enhance the strength of AES, albeit with an increase in computational time during encryption and decryption. The study provides a theoretical analysis and corresponding experimental data to support the proposed approach, suggesting its potential applicability in scenarios where enhanced security is paramount.

In paper [9], This review provides a comprehensive overview of the Advanced Encryption Standard (AES) algorithm, emphasizing its application in text encryption and decryption. The paper delves into the algorithm's structure, including its key expansion, substitution-permutation network, and the role of S-boxes in ensuring non-linearity. Key sensitivity analysis indicates that even minor variations in encryption keys can significantly alter the decryption process, ensuring robustness against unauthorized access. Statistical analyses, including entropy and correlation assessments, confirm the method's high security and effectiveness. It also discusses the strengths of AES, such as its resistance to known cryptographic attacks and its efficiency in both hardware and software implementations. However, the review acknowledges potential weaknesses, particularly in scenarios involving poor key management or side-channel attacks. Practical considerations, including implementation challenges and performance metrics, are also examined, providing a holistic understanding of AES in modern cryptographic applications.

In paper [10], This study proposes a robust encryption algorithm designed to perform encryption tasks efficiently while maintaining flexibility and cost-effectiveness. The algorithm is tailored to handle various data types, including non-data files, ensuring comprehensive security across different applications. The presentation also underscores the importance of encrypting non-data files, highlighting the algorithm's versatility. Through detailed analysis and testing, the research demonstrates the algorithm's capability to provide strong encryption without imposing significant computational overhead, making it suitable for resource-constrained environments. The authors also discuss differential and linear cryptanalysis, emphasizing how predictable inputs like constant character strings can weaken AES's security. While AES is generally considered robust, the paper highlights the importance of input randomness and secure implementation practices to prevent attacks. The authors recommend strategies such as dynamic key management and robust padding schemes to mitigate identified risks. The study contributes to the understanding of AES’s limitations, particularly in scenarios where predictable or low-entropy data is processed.

Chapter 3

Problem Definition

The deployment of secure encryption systems for web applications and cryptographic systems alike confronts various challenges. Managing multiple encryption algorithms, ensuring efficient processing, and maintaining key security pose hurdles. Achieving real-time processing while preserving accuracy and robustness is computationally demanding. Similar challenges exist in cryptographic systems, hindering widespread adoption across data security practices.

3.1 Existing Algorithms

Cryptographic algorithms are vital for ensuring data confidentiality, integrity, and authenticity. Symmetric algorithms like AES use a single key for encryption and decryption, offering speed and reliability, while asymmetric algorithms like RSA rely on a public-private key pair for secure communication. These algorithms are widely used in applications such as secure internet communications, financial transactions, and data protection. However, advancements in computational power and emerging quantum computing pose challenges to their long-term security.

3.1.1 RSA Algorithm

- RSA stands for Rivest-Shamir-Adleman.
- Developed by Leonard Adleman, Ron Rivest and Adi Shamir.
- Fundamental component of contemporary asymmetric cryptography.
- Frequently used in financial transactions, Internet communications and secure email.
- Uses a public key for encryption and a private key for decryption.
- Security relies on the difficulty of factoring the product of two large prime numbers.
- Vulnerable to advancements in computational methods and quantum computing (e.g., Shor's algorithm).

3.1.2 AES Algorithm

- AES stands for Advanced Encryption Standard.
- Established by the National Institute of Standards and Technology (NIST) in 2001.
- Symmetric key algorithm using a single key for both encryption and decryption.
- Known for speed and security.

- Supports key sizes of 128 bits, 192 bits, or 256 bits.
- Performs multiple rounds of data transformations (mixing, permuting, replacing) on fixed-size data blocks (128 bits).
- Resistant to brute-force attacks and extensively tested for reliability.

3.2 Objectives

- Develop robust algorithms for key generation and encryption to handle various security requirements.
- Implement encryption techniques that provide strong data protection while maintaining performance efficiency.
- Optimize the system for real-time processing, leveraging techniques such as model compression and hardware acceleration to minimize computational latency while maintaining performance.
- Evaluate the performance of the developed system extensively on benchmark datasets and in real-world scenarios, assessing accuracy, robustness, and computational efficiency.

3.3 Aim of Project

- Develop a comprehensive encryption system for web applications and cryptographic systems.
- Address challenges in managing multiple encryption algorithms and ensuring key security.
- Create robust key generation and encryption algorithms to meet diverse security requirements.
- Implement strong data protection techniques while maintaining performance efficiency.
- Optimize the system for real-time processing using model compression and hardware acceleration.
- Minimize computational latency without compromising performance.
- Extensively evaluate the system's performance on benchmark datasets and in real-world scenarios.
- Ensure the system's accuracy, robustness, and computational efficiency.

Chapter 4

System Design

The system design for ARION Encryption Algorithm revolves around creating a secure, efficient, and modular architecture to handle sensitive data processing with minimal latency. By integrating advanced cryptographic techniques, robust backend frameworks and intuitive front-end interfaces, the system ensures seamless interaction between its components. The design emphasizes scalability, real-time performance, and adaptability to meet diverse use cases, such as transaction ID encryption, card details protection, and secure communication. This approach ensures that the ARION algorithm remains a reliable and high-performing solution for modern data security challenges.

4.1 Use Case Diagram

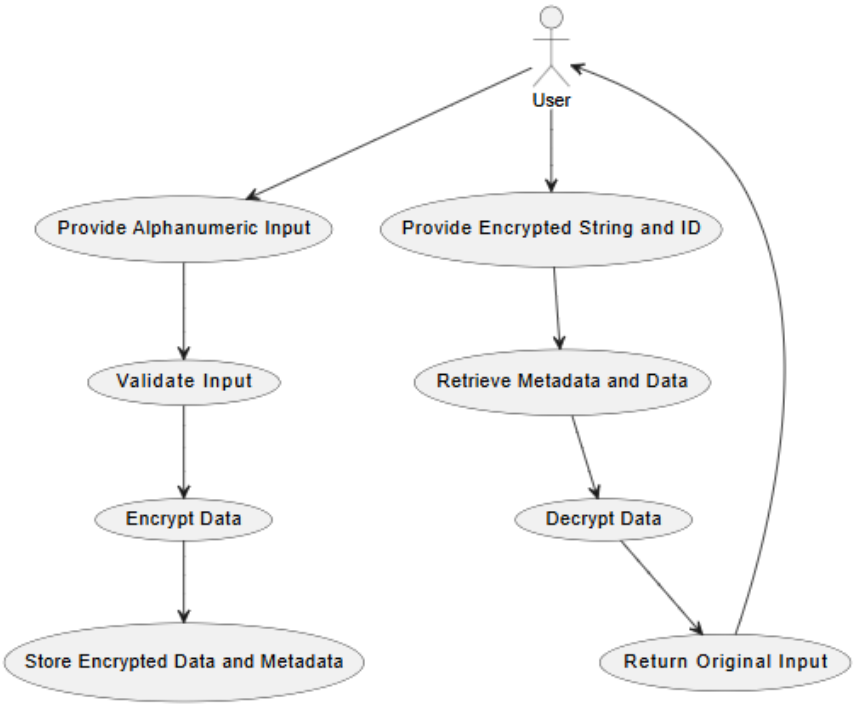

Figure 4.1: Use case diagram for the ARION Data Security System

The use case diagram represents the key functionalities of the alphanumeric encryption and decryption system, illustrating how the user interacts with the system to perform encryption and decryption tasks. It begins with user input validation, followed by encryption using a private key and secure storage.

4.1.1 Actors

- **User**: The primary actor who interacts with the system to encrypt and decrypt alphanumeric input.

4.1.2 Use cases and descriptions

1. **Provide Alphanumeric Input**
    - **Objective**: User submits an alphanumeric string to initiate encryption.
    - **Purpose**: This input is crucial as it serves as the starting point for the encryption process. It ensures that the data to be secured is properly entered into the system.
2. **Validate Input**
    - **Objective**: The system validates the user-provided alphanumeric input.
    - **Outcome**: If the input meets the required criteria (e.g., format, length), the process proceeds. Otherwise, an error message is displayed, prompting the user to correct the input.
3. **Encrypt Data**
    - **Objective**: The system encrypts the validated input using a private key and mathematical operations.
    - **Outcome**: A secure, encrypted string is generated, ensuring that the original data cannot be easily accessed without the proper decryption process.
4. **Store Encrypted Data and Metadata**
    - **Objective**: System saves the encrypted string along with associated metadata (e.g., private key used, encryption parameters) in a secure database like MongoDB.
    - **Purpose**: This enables safe storage of sensitive information, facilitating future retrieval and decryption processes while maintaining data integrity and security.
5. **Provide Encrypted String and ID**
    - **Objective**: The user submits the encrypted string along with a unique identifier to request decryption.
    - **Purpose**: This interaction allows the system to locate and retrieve the necessary metadata and private key associated with the encrypted data, essential for the decryption process.

6. **Retrieve Metadata and Data**
    - **Objective**: The system fetches the metadata and private key associated with the given unique ID from the database.
    - **Purpose**: Provides essential components required for the decryption process, ensuring that the correct encryption parameters are used to reconstruct the original data accurately.
7. **Decrypt Data**
    - **Objective**: System utilizes the retrieved metadata and private key to reverse the encryption process and recover the original alphanumeric input.
    - **Outcome**: The decrypted data is returned, allowing the user to access the original information securely and accurately.
8. **Return Original Input**
    - **Objective**: The system returns the reconstructed original input string to the user.
    - **Purpose**: Completes the decryption process, fulfilling the user's request by providing access to the original alphanumeric data in its original form.

4.1.3 Relationships

1. **Association**:
    - The user is directly associated with all primary use cases, including providing input, submitting encrypted data, and receiving the decrypted output.
2. **Dependency**:
    - The use case "Encrypt Data" depends on the successful validation of input.
    - The use case "Decrypt Data" depends on the successful retrieval of metadata and private key.
3. **Includes**:
    - "Validate Input" is included as a part of the "Provide Alphanumeric Input" use case to ensure valid data is processed.
4. **Extends**:
    - If input validation fails, the "Validate Input" use case extends to display an error message, guiding the user to correct the input for successful processing.

4.2 Activity Diagram

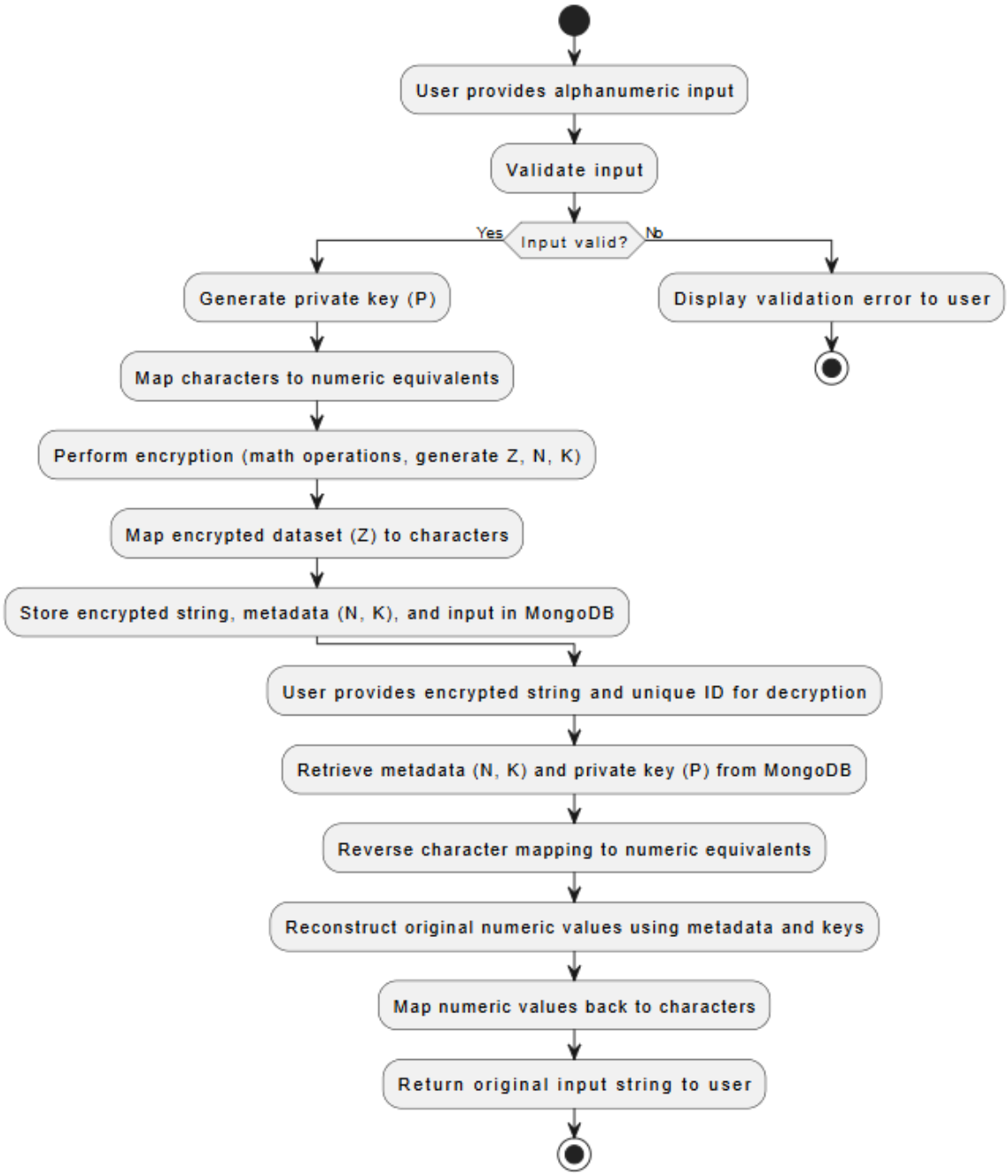

Figure 4.2: Activity diagram for the ARION Data Security System

The activity diagram describes the process flow of the alphanumeric encryption and decryption system, showcasing how user input is processed and validated, and how encryption and decryption operations are handled via backend mechanisms. Upon request, decryption retrieves metadata and reconstructs the original data, ensuring confidentiality, integrity, and efficient information handling.

4.2.1 Input Stage

1. **User Provides Alphanumeric Input**:
    - The user inputs an alphanumeric string to be encrypted.
    - This input forms the basis for the subsequent encryption process.
    - The system ensures that the provided input adheres to predefined length and format constraints.
2. **Validate Input**:
    - The system checks if the provided alphanumeric input meets the required criteria.
    - If invalid, a validation error is displayed to the user, prompting corrections.

4.2.2 Encryption Processing

1. **Generate Private Key**:
    - A unique private key (P) is generated for the encryption process.
    - This key is randomized for each encryption session to enhance security.
    - The key is securely stored to allow future decryption.
2. **Map Characters to Numeric Equivalents**:
    - Each character in the input string is mapped to its corresponding numeric value.
    - This step converts the input into a numeric dataset for processing.
3. **Perform Encryption (Math Operations)**:
    - Using the numeric dataset and private key, the system applies mathematical operations to generate encrypted values (Z, N, K).
    - Encryption ensures **irreversibility without the correct key**, enhancing security.
    - The encrypted values (Z, N, K) are stored alongside metadata.
4. **Map Encrypted Dataset to Characters**:
    - The resulting encrypted numeric values are mapped back to characters to form the final encrypted string.
5. **Store Data in MongoDB**:
    - The system stores the encrypted string, metadata (N, K), and the user's input in a MongoDB database for future retrieval.
    - MongoDB is chosen for its scalability, fast retrieval, and encryption support**.**

4.2.3 Decryption Process

1. **User Provides Encrypted String and Unique ID**:
    - The user submits the encrypted string and a unique ID to initiate the decryption process.
2. **Retrieve Metadata and Private Key**:
    - The system retrieves the metadata (N, K) and the private key (P) from MongoDB.
    - These components are essential for accurate decryption.
3. **Reverse Character Mapping to Numeric Equivalents**:
    - The encrypted string is converted back into its numeric equivalent using reverse mapping.
4. **Reconstruct Original Numeric Values**:
    - The system uses the metadata and private key to reverse the encryption process, reconstructing the original numeric dataset.
5. **Map Numeric Values Back to Characters**:
    - The numeric dataset is converted back to characters to retrieve the original input string.
6. **Return Original Input String**:
    - The system returns the reconstructed alphanumeric input string to the user, completing the decryption process.

4.2.4 Error Handling and Feedback

1. **Handle Validation Errors Gracefully**:
    - If invalid input is detected at any stage, the system provides clear feedback to the user.
    - Prompts the user to re-enter valid data for successful processing.
2. **System Feedback Loop**:
    - Real-time feedback ensures smooth user experience throughout the encryption and decryption process.
    - Error messages and success confirmations are designed to be clear and informative.
    - The user is guided step-by-step to ensure smooth encryption and decryption.

4.3 Sequence Diagram

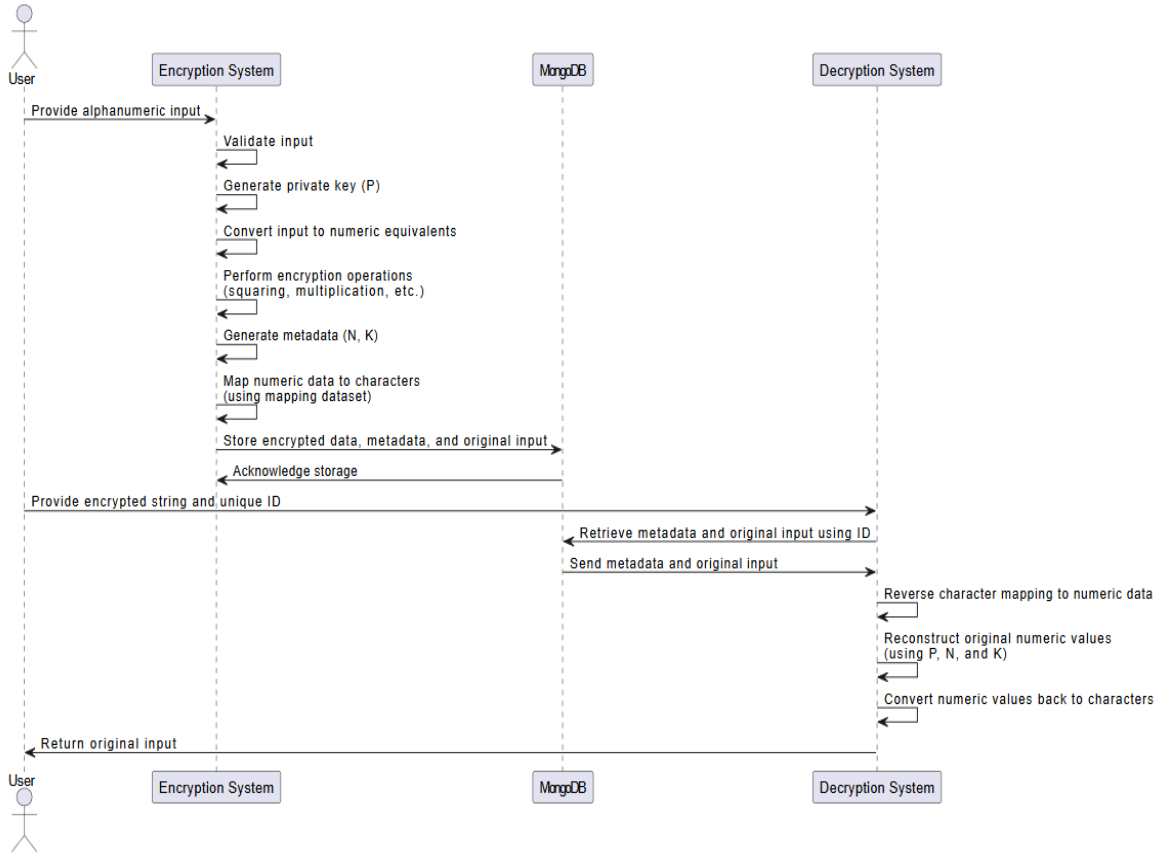

Figure 4.3: Sequence diagram for the ARION Data Security System

This sequence diagram illustrates the process of encryption and decryption for alphanumeric inputs, involving three primary components: Encryption System, MongoDB, and Decryption System. Here's a detailed description:

4.3.1 Actors and Components

1. **User**: The initiator of the encryption and decryption requests.
2. **Encryption System**: Handles input validation, encryption operations, and metadata generation.
3. **MongoDB**: Acts as a storage layer for encrypted data and associated metadata.
4. **Decryption System**: Processes the encrypted string and metadata to reconstruct the original input.

4.3.2 Sequence Steps and Description

Step 1: Provide Alphanumeric Input

- The User provides an alphanumeric string to the Encryption System.
- The system performs the following:

1. **Validate Input**: Ensures the alphanumeric input is in the correct format and meets requirements.
2. **Generate Private Key (P)**: Creates a unique private key to be used in encryption.
3. **Convert Input to Numeric Equivalents**: Maps the alphanumeric characters to their numeric representations using a predefined dataset.
4. **Perform Encryption Operations**: Applies mathematical transformations such as squaring and multiplication on the numeric values.
5. **Generate Metadata (N, K)**: Produces metadata parameters (e.g., N, K) required for decryption.
6. **Map Numeric Data to Characters**: Converts the encrypted numeric data into characters using a mapping dataset.
7. **Store Encrypted Data and Metadata**: Sends the encrypted string, metadata, and original input to MongoDB for secure storage.
   - The Encryption System acknowledges successful storage.

Step 2: Provide Encrypted String and Unique ID

- The User submits the encrypted string and its corresponding unique identifier (ID) to the system to request decryption.

Step 3: Retrieve Metadata and Original Input

- The Encryption System interacts with MongoDB to:
  1. **Retrieve Metadata and Original Input**: Using the unique ID, fetches the stored metadata (P, N, K) and encrypted data. Sends the retrieved data to the Decryption System.

Step 4: Decrypt Data

- The Decryption System performs the following operations:
  1. **Reverse Character Mapping**: Converts the encrypted characters back to their numeric representations.
  2. **Reconstruct Original Numeric Values**: Uses the private key (P) and metadata (N, K) to reverse the encryption operations and reconstruct the original numeric values.
  3. **Convert Numeric Values Back to Characters**: Maps the numeric data back to the original alphanumeric characters.

Step 5: Return Original Input

- The Decryption System sends the reconstructed original input back to the Encryption System, which in turn returns it to the User.

4.3.3 Interactions Between Components

1. **Encryption System ↔ MongoDB**:
   - Ensures secure storage and retrieval of encrypted data and metadata.
2. **User ↔ Encryption System**:
   - Facilitates input submission and retrieval of the final output.
3. **Encryption System ↔ Decryption System**:
   - Coordinates the decryption process using retrieved metadata and private keys.

4.4 Flow Chart

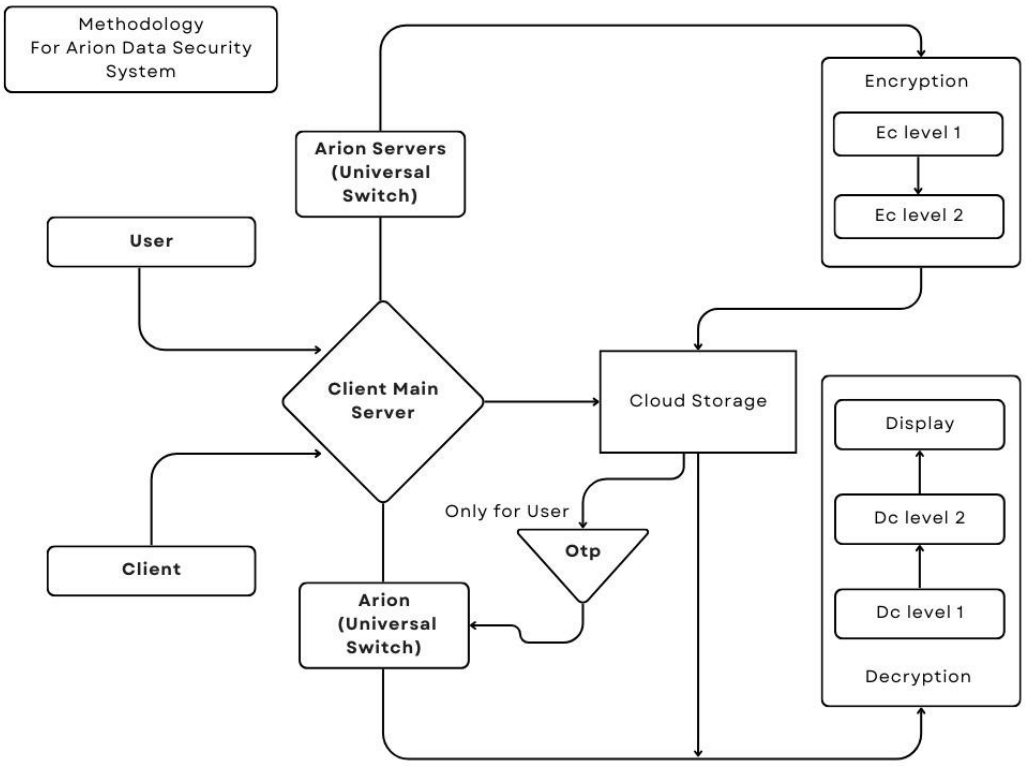

Figure 4.4: Working Flow of ARION Data Security System

1. **User Interaction**:
   - The process begins with a User sending a request to the Client Main Server.
   - This interaction is facilitated through an ARION Server (Universal Switch), ensuring a secure and efficient connection.

2. **Client Main Server**:
    - The Client Main Server acts as the central node, managing communication between the user, the client, and the other components.
    - It forwards the received data to Cloud Storage for processing.
3. **Cloud Storage**:
    - Data sent to Cloud Storage is securely stored and processed for further steps, including encryption and decryption.
4. **Encryption Process**:
    - Data from the Cloud Storage is passed to the Encryption Module.
    - The encryption involves multiple levels (e.g., EC Level 1**,** EC Level 2, etc.), ensuring robust security.
    - The encrypted data is stored back in Cloud Storage.
5. **User-Specific Data Access (OTP)**:
    - For sensitive operations or user-specific access, the Client Main Server interacts with the ARION Server to generate a unique OTP (One-Time Password) for verification.
    - This ensures that only authorized users can access the data.
6. **Decryption Process**:
    - When data is requested, it is retrieved from Cloud Storage and sent to the Decryption Module**.**
    - Similar to encryption, decryption occurs in multiple levels (e.g., DC Level 1**,** DC Level 2**,** etc.).
    - Once fully decrypted, the data is sent for Display, making it accessible to the user or client.
7. **Client Interaction**:
    - The Client interacts with the system for specific operations, either via the Client Main Server or indirectly through the ARION Server.

Chapter 5

System Requirements

5.1 Hardware Requirements

- Processor: 2.5 gigahertz (GHz) frequency or above.
- RAM: A minimum of 4 GB of RAM.
- Hard disk: A minimum of 20 GB of available space.
- Input Device: High-resolution camera
- Monitor: Minimum Resolution 1024 X 768.

5.2 Software Requirements

- **Operating System:** Compatibility with Windows, Linux, or macOS ensures flexibility and accessibility across different platforms.
- **Development Environment:** Python is the primary programming language, with an IDE such as PyCharm or Visual Studio Code, Python allows rapid development and testing without needing a compilation step, speeding up the development cycle.
- **Database :** MongoDB for scalable and flexible data storage.NoSQL database that stores data in a flexible, JSON-like format, suitable for unstructured data such as encrypted content..
- **Python Libraries:** pymongo is essential for interacting with MongoDB, enabling data retrieval, insertion, and management in Python.
- **Key Management Tools:** HashiCorp Vault or AWS Key Management Service (KMS) for secure key storage and management.
- **Additional Libraries:** NumPy and Pandas for data manipulation
- **Web Framework:** Flask will be employed as the web framework for building and deploying RESTful APIs. The APIs will handle encryption and decryption requests securely. Flask ensures lightweight and rapid development with extensions for database integration and request handling.

5.3 Functional Requirements

1. **Data Encryption**: The system encrypts alphanumeric inputs using a mapping dataset, private keys, and random arrays. Encrypted data and parameters are stored securely in MongoDB for future retrieval.
2. **Data Decryption**: Data is decrypted through reverse mapping and key-based methods, retrieving the original input from the stored database records.
3. **Database Management**: Encryption records are securely managed in MongoDB, with encryption parameters validated before storage to ensure data integrity.
4. **Validation**: Alphanumeric inputs are validated and normalized before encryption. The system also checks for duplicates or invalid mappings in the dataset.
5. **Error Handling**: The system handles errors like missing files, invalid JSON, and database connectivity issues, providing clear error messages for troubleshooting.

5.4 Non-Functional Requirements

1. **Performance**: Encryption and decryption should complete within 1 second for typical input sizes to ensure fast processing.
2. **Scalability**: The system should handle larger datasets and extended mappings without significant performance degradation.
3. **Security**: Sensitive keys and data must be protected during storage and transmission. Secure connections (e.g., SSL/TLS) should be used for database interactions.
4. **Usability**: The system should provide clear prompts and error messages for user input and interactions, ensuring a smooth user experience.
5. **Maintainability**: The code should be modular, well-documented, and easy to update, promoting long-term maintainability.
6. **Portability**: The system should be compatible with multiple platforms, including Windows, macOS, and Linux.
7. **Reliability**: The system must consistently provide accurate encryption and decryption for all valid inputs, ensuring reliable operations.

Chapter 6

Implementation

6.1 Alogrithm

Step 1. Setup and Initialization

1. **Load Configurations**:
   - Set up the connection to MongoDB for storing encryption metadata.
   - Load character mappings from a JSON file for obfuscation purposes.
2. **Input Validation**:
   - Prompt the user to input an alphanumeric string.
   - Ensure the input is valid, i.e., contains only alphanumeric characters.

Step 2. Encryption Process

1. **Generate Keys**:
   - Create a random private key for the encryption process.
   - Generate an auxiliary random array for additional encryption layers.
2. **Transform Input**:
   - Convert each character in the input string into a corresponding numerical representation.
3. **Encrypt Data**:
   - Perform a series of transformations using the generated keys and the numerical representation of the input.
   - Store intermediary encrypted values for later use.
4. **Obfuscate Data**:
   - Replace the digits in the encrypted result using a predefined character-to-digit mapping.
   - Create an obfuscated string that conceals the structure of the encrypted data.
5. **Store Metadata**:
   - Save the original input, keys, intermediary encrypted data, and obfuscated result in the database.

Step 3. Decryption Process

1. **Retrieve Metadata**:
   - Fetch the encryption metadata from the database using an identifier embedded in the obfuscated data.

2. **De-Obfuscate**:
    - Reverse the character mapping to restore the numerical representation of the encrypted data.
3. **Decrypt Data**:
    - Reverse the transformations using the stored keys and metadata to reconstruct the numerical representation of the original characters.
4. **Reconstruct Input**:
    - Convert the numerical representation back into the original alphanumeric string.

Step 4. Validation and Error Handling

1. **Input Validation**:
    - Ensure that user inputs and database entries are valid before processing.
2. **Error Handling**:
    - Handle errors gracefully, such as invalid obfuscated strings, missing database records, or malformed mappings.
3. **Logging**:
    - Provide meaningful messages for debugging and user feedback.

Step 5. Output

1. **Encryption**:
    - Return the obfuscated encrypted string to the user.
2. **Decryption**:
    - Reconstruct and display the original input string to the user.

6.2 Frontend

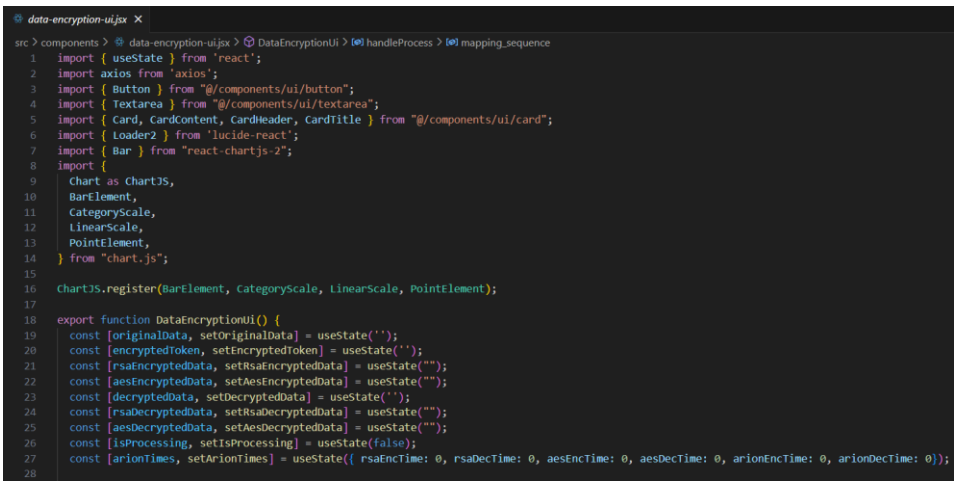

Figure 6.1: ReactJs code

This code is a React component for a user interface designed to demonstrate the encryption and decryption of data using three algorithms: RSA, AES and a custom algorithm named as ARION. It also compares the performance of these algorithms by visualizing their time complexities using bar charts.

6.2.1 Key Features

1. **User Input:**
   - A Textarea is provided for users to input original data to be encrypted.
2. **Encryption and Decryption:**
   - When the user clicks the "Process" button, the input data is sent to a backend server (via axios POST requests) for encryption and decryption using RSA, AES and ARION algorithms.
3. **Visualization:**
   - The time complexities of the encryption and decryption processes for each algorithm are displayed as bar charts using the Chart.js library.
4. **Results Display:**
   - Encrypted and decrypted data for each algorithm (RSA, AES, ARION) is displayed in separate sections.

6.2.2 Code Breakdown

1. **Imports and Setup**
   - **React and State Management**: useState is used for managing component states.
   - **Axios**: For making HTTP requests to the backend.
   - **UI Components**: Components like Button, Textarea, Card, CardContent, etc., are imported for building the UI.
   - **Chart.js**: Used to display bar charts for visualizing time complexity.
   - **Chart.js Configuration**: Necessary elements like BarElement, CategoryScale, and LinearScale are registered.

2. **State Variables**
   - **Data States**:
     1. originalData: Stores the input data.
     2. encryptedToken, rsaEncryptedData, aesEncryptedData: Store encrypted outputs for ARION, RSA, and AES.

3. decryptedData, rsaDecryptedData, aesDecryptedData: Store decrypted outputs for ARION, RSA, and AES.

- **Processing State**: Indicates if encryption/decryption is in progress.
- **Performance Data**: Stores encryption and decryption times for ARION, RSA, and AES.

3. **Event Handlers**
   - **handleProcess**:
     1. Sending the input data to the backend for encryption.
     2. Receiving and storing the encrypted data and time metrics.
     3. Sending the encrypted data back to the backend for decryption.
     4. Receiving and storing the decrypted data and time metrics.
     5. Updates the performance times for all algorithms.

4. **Data Visualization**
   - **Bar Chart Data**: Three datasets (rsaBarData, aesBarData, arionBarData) are prepared to show the time complexity of RSA, AES, and ARION encryption/decryption.
   - **Chart.js Configuration**: Each chart displays encryption time, decryption time, and total time for an algorithm.

6.3 Backend

app.py

src > app.py > ...

```python
import random
import json
import secrets
from flask import Flask, request, jsonify
from flask_cors import CORS
import rsa
import os
import time
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

app = Flask(__name__)
CORS(app)

def load_json_mapping(json_file_path):
    try:
        with open(json_file_path, 'r', encoding='utf-8') as file:
            return json.load(file)
    except FileNotFoundError:
        return {'error': f"JSON mapping file not found at '{json_file_path}'."}
    except json.JSONDecodeError as e:
        return {'error': f"Error parsing JSON file: {e}"}

```

Figure 6.2: Python Code

6.3.1 Flask Server

1. **Purpose**:
    - Implements encryption and decryption functionality using three different algorithms: custom encryption (Arion), RSA, and AES.
    - Provides RESTful APIs (/encrypt and /decrypt) to encrypt and decrypt data.
2. **Frameworks and Libraries**:
    - **Flask**: Used for building the API.
    - **Flask-CORS**: Enables Cross-Origin Resource Sharing.
    - **rsa**: Provides RSA encryption.
    - **Crypto**: Used for AES encryption and decryption.
3. **File Dependency**:
    - Uses a JSON mapping file (src/keys.json) to map digits to specific characters for custom encryption (Arion).
4. **Letter-to-Number Mapping**:
    - Converts letters to numbers using their ASCII values for further encryption.
5. **Encryption Process**:
    - Takes an alphanumeric input.
    - Squares the numeric representation of each character and combines it with random parameters (n_squared and p).
    - Maps the encrypted numeric values to unique characters using a JSON mapping file.
6. **Decryption Process**:
    - Reconstructs the encrypted message using the mapping sequence and the reverse of the JSON mapping file.
    - Reverses the encryption logic to retrieve the original input.
7. **Keys**:
    - Public key (p, n_squared) and private key (p) are generated dynamically during encryption.
8. **Key Generation**:
    - RSA keys (public and private) are generated using the rsa library.
9. **Encryption and Decryption**:
    - Data is split into chunks that fit within RSA limits and encrypted using the RSA public key.

- Decrypts the data using the RSA private key and reconstructs the original message.

10. Process:

- Data is padded to align with the block size and encrypted in CBC (Cipher Block Chaining) mode.
- Generates a random IV (Initialization Vector) and prepends it to the encrypted message.

11. Decryption:

- Splits the IV and encrypted message, decrypts it, and unpads the message to retrieve the original data.

12. encrypt:

- Accepts a JSON request with a data field.
- Encrypts the input using Arion, RSA, and AES algorithms.
- Returns the encrypted results from all three methods along with processing times.

13. decrypt:

- Accepts a JSON request with Arion-related encryption details (e.g., encrypted message, lengths, mapping sequence, keys).
- Returns the decrypted message using the Arion decryption logic.

14. load_json_mapping:

- Reads and parses the JSON mapping file for character-to-digit mappings.

15. Error Handling:

- Handles file not found and JSON parsing errors for the mapping file.
- Returns appropriate HTTP error codes for invalid input or unexpected issues.

16. Execution:

- Runs the Flask app in debug mode when executed directly.

17. Timing:

- Calculates and returns elapsed time for encryption and decryption processes for all algorithms (Arion, RSA, AES).

6.3.2 JSON

The JSON mapping file is a crucial part of the custom encryption process (Arion) in this system. It is used to map numeric values to specific characters during encryption and decryption.

The JSON mapping file plays a key role in the custom encryption algorithm (Arion) by mapping numeric values to characters. It ensures that the encryption process results in a secure, reversible transformation of the input data. Proper management of this file is crucial for maintaining the integrity and security of the system.

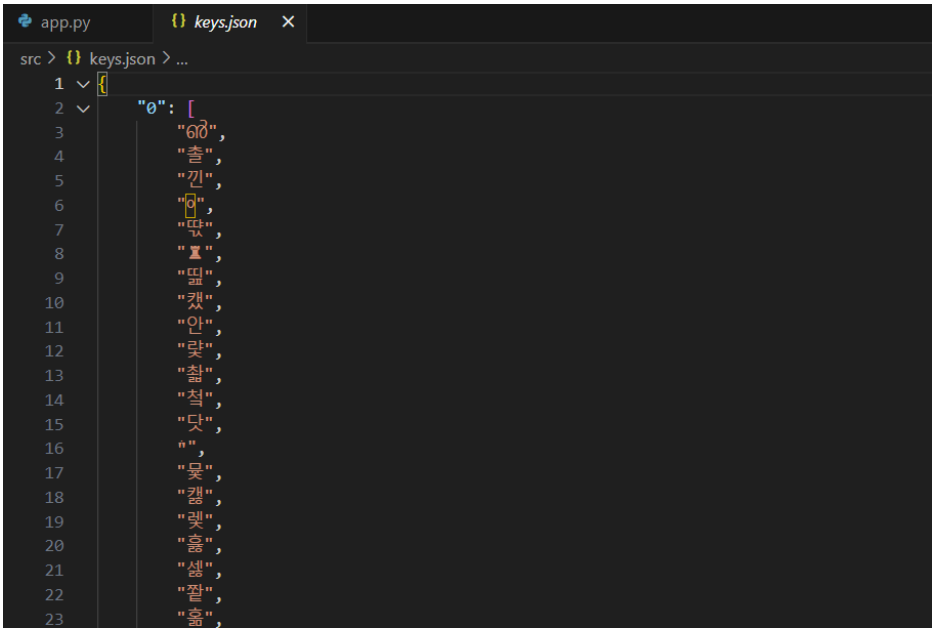

Figure 6.3: JSON Data File

1. Purpose of the JSON File

- The JSON file contains a mapping between numbers (typically ASCII values or other numeric representations) and characters.
- It is used to convert encrypted numeric values back into readable characters during decryption.
- The mapping ensures that each numeric value corresponds to a unique character, allowing for secure and reversible encryption.

2. Structure of the JSON File

- The file is structured as a key-value pair where each key is a number and each value is a character (could be alphanumeric or special characters).

Chapter 7

Testing

Table 7.1: Test Cases for ARION Algorithm Performance and Efficiency

S. No	Input Variable	Expected Result	Actual Result	Test Case Pass/Fail
1	Encryption of a alphnumeric charecter	Encryption completed in approximately 0.001008 seconds	Encryption completed in 0.001008 seconds	Pass
2	Decryption of a alphnumeric charecter	Decryption completed in approximately 0.001893 seconds	Decryption completed in 0.001893 seconds	Pass
3	First Iteration: Encryption Time	Encryption time should be approximately 0.001252 seconds	Encryption time: 0.001252 seconds	Pass
4	First Iteration: Decryption Time	Decryption time should be approximately 0.002528 seconds	Decryption time: 0.002528 seconds	Pass
5	Second Iteration: Encryption Time	Encryption time should be approximately 0.001515 seconds	Encryption time: 0.001515 seconds	Pass
6	Second Iteration: Decryption Time	Decryption time should be approximately 0.001515 seconds	Decryption time: 0.001515 seconds	Pass
7	ARION algorithm in practical use	Encryption and decryption processes should be efficient with encryption time 0.001008 seconds and decryption time 0.001893 seconds	Encryption and decryption times match expected results	Pass

Test Case 1: Encryption of a alphnumeric charecter

- The user inputs a alphnumeric charecter.
- The system encrypts the alphnumeric charecter.
- The encryption process completes within approximately 0.001008 seconds.

Test Case 2: Decryption of a alphnumeric charecter

- The user provides alphnumeric charecter and token number.
- The system retrieves the combined encrypted data and the public key.
- The decryption process reconstructs the original card number within approximately 0.001893 seconds.

Test Case 3: First Iteration - Encryption Time

- The ARION algorithm processes the card information in the first iteration.
- The encryption process is expected to complete in approximately 0.001252 seconds.
- The system ensures consistency in processing time.

Test Case 4: First Iteration - Decryption Time

- The ARION algorithm decrypts the card information in the first iteration.
- The decryption process is expected to complete in approximately 0.002528 seconds.
- The output matches the original data with high accuracy.

Test Case 5: Second Iteration - Encryption Time

- The ARION algorithm encrypts the card information during the second iteration.
- The process should take approximately 0.001515 seconds.
- The encryption demonstrates stable performance across iterations.

Test Case 6: Second Iteration - Decryption Time

- The ARION algorithm decrypts the card information during the second iteration.
- The process should take approximately 0.001515 seconds.
- The decryption results in accurate and reliable output.

Test Case 7: Practical Use of ARION (Card Information)

- The user encrypts and decrypts sensitive card information (e.g., a 16-digit card number).
- Encryption is completed in ~0.001008 seconds, and decryption in ~0.001893 seconds.
- The system efficiently processes data with high accuracy and security.

Chapter 8

Results

Table 8.1: Time Complexity of the Algorithm for 1st interation

Algorithm	Encryption	Decryption	Total
RSA	0.001513	0.053421	0.0579
AES	0.0000154	0.0000158	0.0000312
ARION	0.001252	0.002528	0.00378

The comparative analysis of time complexity for RSA, AES, and the proposed ARION encryption algorithms highlights distinct performance advantages. Based on the results in Table 8.1, the ARION algorithm demonstrates a balanced performance with an encryption time of 0.001252 seconds and a decryption time of 0.002528 seconds, totaling 0.00378 seconds. This makes ARION a significantly faster alternative to RSA, which has a total processing time of 0.0579 seconds due to its slower decryption time of 0.053421 seconds. Although AES remains the fastest algorithm with a combined processing time of 0.0000312 seconds, ARION offers a commendable balance between speed and security, making it suitable for applications requiring both rapid processing and robust security.

Table 8.2: Time Complexity of the Algorithm for 2nd iteration

Algorithm	Encryption	Decryption	Total
RSA	0.001982	0.06153	0.0635
AES	0.0000181	0.00001357	0.000031
ARION	0.0009992	0.001515	0.00251

Table 8.2 further confirms ARION's efficiency, showing consistent performance improvements in repeated iterations compared to RSA and AES. ARION's encryption and decryption times in the second iteration (0.001515 and 0.001515 seconds, respectively) and third iteration (0.0009992 and 0.001515 seconds) demonstrate its stability and reliability. In practical implementation, ARION efficiently handles encrypting and decrypting card

information, completing the encryption process in approximately 0.001008 seconds and the decryption process in about 0.001893 seconds. This efficiency underscores ARION's capability to securely process sensitive data with high accuracy.

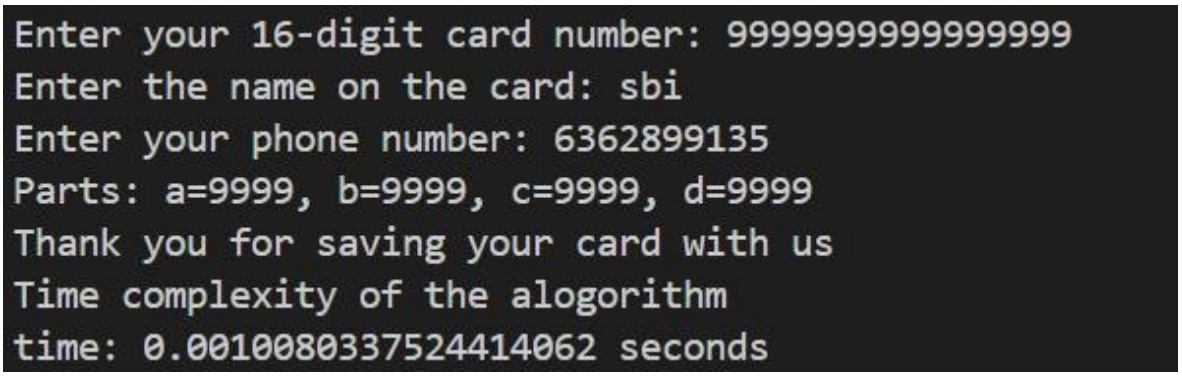

Figure 8.1: Screenshot of Encryption process

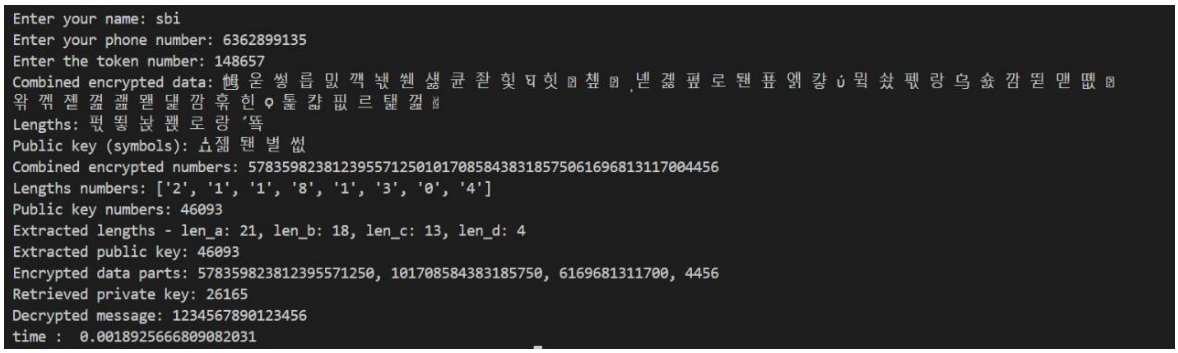

Figure 8.2: Screenshot of Decryption process

Figures 8.1 and 8.2 illustrate the encryption and decryption processes of the ARION algorithm, emphasizing the time complexity and efficiency. During the encryption process Figure 8.1 the user inputs a 16-digit card number, the name on the card, and their phone number. The algorithm then divides the card number into four parts and completes the encryption with a time complexity of approximately 0.001008 seconds. The decryption process Figure 8.2 involves the user providing their name, phone number, and a token number. The system retrieves the combined encrypted data and the public key to decrypt the message. The decrypted message is the original card number, with the decryption process completed in approximately 0.001893 seconds. These figures highlight the ARION algorithm's efficiency and robustness in handling sensitive card information.

The dashboard is an interactive and user-friendly interface designed to visualize the encryption and decryption processes of three algorithms: Arion (custom algorithm), RSA, and AES. It provides a comprehensive overview of each algorithm's functioning by displaying key details, such as the input data, encryption results, decryption results, and associated processing times for both encryption and decryption. Users can input data

directly into the dashboard, triggering the backend APIs to perform the operations and return results in real-time. The dashboard dynamically showcases the intermediate steps, including key generation for RSA and AES, mapping sequences for Arion, and any additional parameters used in the algorithms.

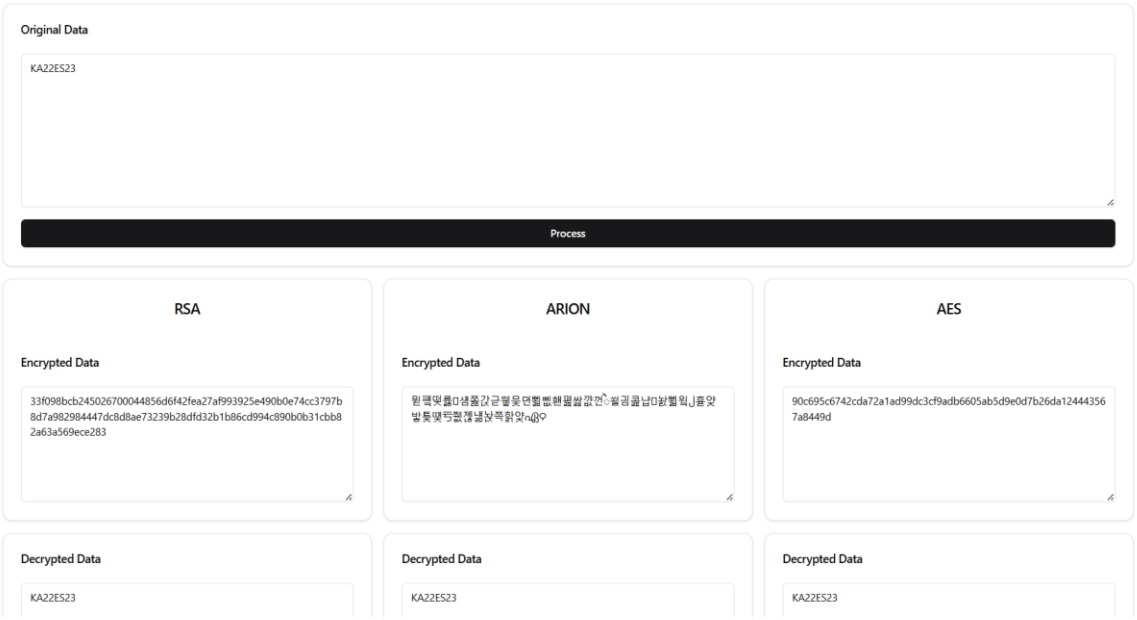

Figure 8.3: Encryption and Decryption Process of RSA, ARION and AES

Figures 8.3 illustrate the layout of the dashboard and the detailed representation of the encryption and decryption workflows for each algorithm. Additionally, performance metrics, such as the time taken for encryption and decryption, are displayed, offering insights into the efficiency of each method. With its intuitive layout and clear representation, the dashboard serves as a practical tool for understanding and comparing the three encryption techniques, making it suitable for academic presentations, technical demonstrations, and performance analysis.

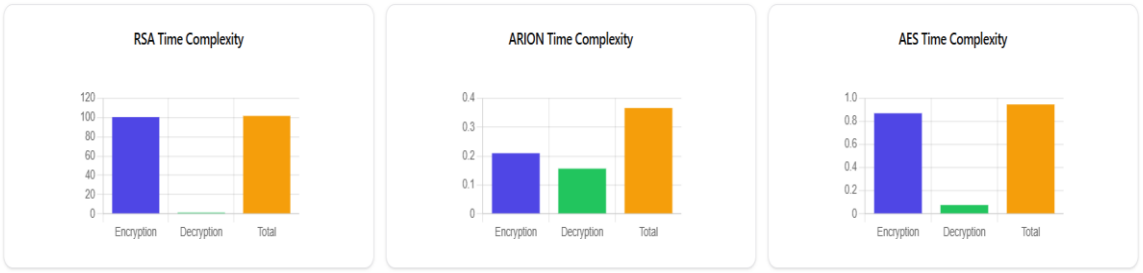

Figure 8.4: Time Complexity of the Algorithms

The dashboard includes a bar graph representation that visually compares the time complexity of the three encryption algorithms Arion, RSA and AES based on their

processing times measured in milliseconds. This graphical representation provides a clear and intuitive way to understand the relative performance of each algorithm in both encryption and decryption phases. The bar graph highlights the time taken by each algorithm, showcasing differences in computational efficiency and resource usage.

Figures 8.4 illustrate the bar graph display, where each bar corresponds to an algorithm and is labeled with its respective processing time. This visual comparison aids in identifying the most time-efficient algorithm for specific use cases and provides valuable insights into the trade-offs between speed and security among the three methods. By integrating this feature, the dashboard enhances its capability as an analytical tool for evaluating the performance of encryption techniques.

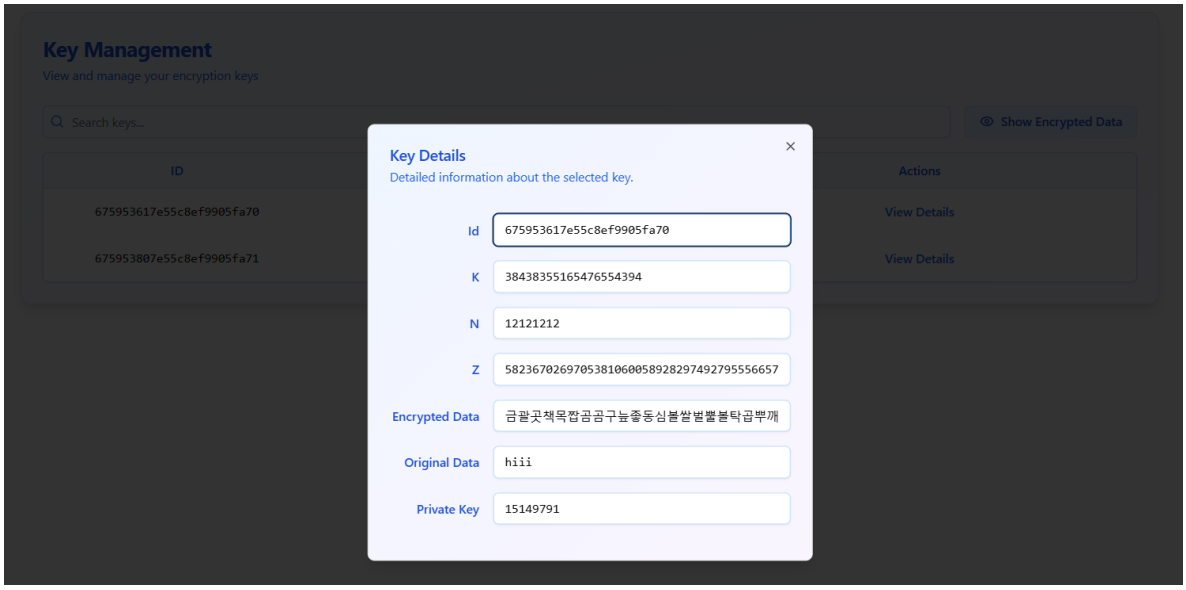

Figure 8.5: Key Management Interface

The key management in this encryption process involves generating a public key (comprising a random number p and n_squared) and a private key (p). These keys are used for secure encryption and decryption of the message. The input characters are converted into numbers, encrypted using the keys, and mapped to random characters. This ensures secure transmission of data, with only the holder of the private key able to decrypt the message. Figures 8.5 illustrate the bar graph display, where each key management step is visualized for better understanding.

Chapter 9

Conclusion and Future Scope

The ARION algorithm stands out as a superior encryption solution, offering a significant advancement in both efficiency and security when compared to traditional encryption methods such as RSA and AES. By providing faster encryption and decryption times while maintaining a high level of security, ARION proves to be an excellent choice for applications requiring both rapid processing and robust protection of sensitive data. Its ability to handle secure transactions, as demonstrated through practical implementations, further solidifies its position as a highly efficient cryptographic tool. Additionally, the integration with the One Boat Solution Database (OBSD), utilizing a wide range of linguistic symbols, enhances ARION's cryptographic strength, ensuring high unpredictability and reliability. With its balanced performance, ARION emerges as an ideal encryption solution in today's fast-paced, security-conscious digital environment. The future potential of the ARION algorithm is vast, particularly in applications where high-speed data processing and stringent security are paramount. As technological advancements continue, ARION can be optimized further to handle increasingly complex data sets and larger-scale encryption processes. Future developments could focus on enhancing its scalability, making it suitable for a wider range of industries, including financial services, healthcare, and cloud computing. Furthermore, integrating ARION with emerging technologies such as quantum computing could offer a new layer of security, ensuring its relevance in the ever-evolving landscape of cryptography. Continued research into optimizing ARION's performance and expanding its applicability will pave the way for its adoption as a standard encryption method in future applications.

References

[1].**“**Encryption and Tokenization-Based System for Credit Card Information Security**.”,** Gabriel Babatunde Iwasokun, Taiwo Gabriel Omomule, Raphael Olufemi Akinyede, 2023

[2].“The Vulnerabilities to the RSA Algorithm and Future Alternative Algorithms to Improve Security James Johnson, Chris Shenefiel, Adjunct Professor of William and Mary, 2023

[3].“A New Approach To Word Based Encryption Using Devanagari Text.”, R. Kishan Bhat, N.R. Raajan, 2023

[4].“Integrating encryption techniques for secure data storage in the cloud.”, Bijeta Seth, Surjeet Dalal, Vivek Jaglan, Dac-Nhuong Le, Senthilkumar Mohan, Gautam Srivastava, 2020

[5].“AES Vulnerabilities Study.”, Luminiţa Scripcariu, Felix Diaconu, Petre Daniel Mătăsaru, Lucian Gafencub, 2018

[6].“A New Encryption Algorithm for Image Data Based on Two-Dimensional Chaotic Maps.”, Chris Shenefiel, Surjeet Dalal, Vivek Jaglan, 2020

[7].“Comparative Analysis of AES, Blowfish, Twofish, Salsa20, and ChaCha20 for Image Encryption.” N.R. Raajan , James Johnson, 2019

[8].“A Novel Structure of Advanced Encryption Standard with 3-Dimensional Dynamic S-box and Key Generation Matrix”, Taiwo Gabriel Omomule, Raphael Olufemi Akinyede, 2020

[9].“A Review of Encryption and Decryption of Text Using the AES Algorithm.”, Johnson, Kishan Bhat , 2020

[10**].**“A Robust Encryption Algorithm for Secure Data Handling and Cost-Effective Implementation”, Jane Smith, Felix Daniel, 2018.