

Design and Educational Implementation of a Web-Based Hybrid Encryption System Using Django

Author: Shyam Kumar Tuti¹, Sai Prakash Kuppannagari¹, Kalyani Kusumanchi¹, Manasa Routhu¹

Department of Computer Science and Information technology ¹

Lendi Institute of Engineering and Technology (A), Vizianagaram, India

Abstract: The increasing reliance on web applications for handling sensitive user data has made application-level security a critical requirement. Although cryptographic algorithms are well established, their practical integration into web frameworks is often limited to theoretical demonstrations or isolated scripts. This paper presents the design and implementation of a web-based encryption system developed using the Django framework to demonstrate the practical application of cryptographic techniques in a real-time web environment. The system allows authenticated users to perform secure text encryption and decryption using symmetric algorithms such as Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES), while RSA is employed for secure symmetric key protection. Additionally, SHA-256 hashing is used to ensure secure password storage and user authentication. Experimental observations conducted on varying input sizes indicate that AES provides lower computational overhead compared to 3DES, while hybrid encryption introduces additional processing cost due to asymmetric key operations. The primary objective of this work is to provide an educational and practical reference for implementing cryptography within web applications. The results confirm that established encryption algorithms can be effectively combined with modern web frameworks to achieve secure and user-accessible data protection. This work is intended primarily as an educational and practical implementation reference rather than a proposal of new cryptographic algorithms.

Keywords: Cryptography, Web Application Security, Django Framework, AES, RSA, Hybrid Encryption,

1. INTRODUCTION

In the digital era, information has become one of the most valuable assets, and its protection has emerged as a crucial concern across every sector. The continuous exchange of sensitive data through networks and cloud-based systems exposes it to numerous security threats such as data breaches, unauthorized access, and cyberattacks. As a response to these vulnerabilities, **cryptography** has become the cornerstone of modern data protection,

ensuring confidentiality, authenticity, and integrity of digital communications (*Singh, 2022*).

Cryptography is the science of converting readable data (plaintext) into an unreadable format (ciphertext) through mathematical transformations known as encryption algorithms. Only authorized parties with the correct decryption key can reverse the process to retrieve the original data. This principle serves as the foundation for many modern security protocols including **SSL/TLS**, **VPNs**, **digital signatures**, and **secure messaging systems** (*Kumar & Sharma, 2021*).

However, while cryptography is widely studied in theory, its **practical implementation within web-based systems** remains an essential educational and developmental exercise. Many organizations and developers struggle to bridge the gap between understanding cryptographic principles and deploying them in real-world applications. This research focuses on building a **Web-Based Cryptography System** developed using the **Django framework**, which allows users to securely perform encryption and decryption through a simple web interface. The system demonstrates how the core concepts of cryptography can be integrated into modern web technologies to protect data confidentiality and ensure secure online interactions. Django, being a high-level Python framework, provides built-in mechanisms for authentication, session management, and secure data handling, making it an ideal choice for implementing cryptographic operations in a controlled environment (*Brown et al., 2020*).

The motivation behind this project stems from the increasing demand for **data security awareness** and **hands-on educational tools**. Students, researchers, and developers often require platforms that allow experimentation with cryptographic techniques in a practical, user-friendly manner. This system not only achieves that purpose but also contributes to the academic exploration of integrating encryption algorithms within real-time web applications.

The proposed cryptographic platform is designed with the following objectives:

1. To design and implement a secure, interactive web platform for performing encryption and decryption.
2. To integrate Django's authentication and security features with custom encryption algorithms.
3. To demonstrate real-world applications of cryptographic techniques in web communication.
4. To provide a learning tool that bridges the gap between theoretical cryptography and practical implementation.

Through this project, the study emphasizes the significance of applying cryptographic methods in everyday web systems, from simple text encryption to more advanced key-based algorithms. The system architecture follows the client-server model, where the front end enables user interaction and the backend executes cryptographic operations securely on the server side.

2. RELATED WORK, MOTIVATION AND PROBLEM IDENTIFICATION

2.1 Related Work

[1] Daemen, J., & Rijmen, V., *"The Design of Rijndael: AES — The Advanced Encryption Standard"*, Springer-Verlag, 2002.

This foundational paper introduces the AES algorithm's architecture, detailing its substitution-permutation network and explaining how it achieves high resistance against linear and differential cryptanalysis.

[2] Tuchman, W., *Hellman Presents No Shortcut Solutions to DES*, IEEE Spectrum, 1979 of DES Motivation for stronger encryption standards (3DES context) Used for Historical background

[3] Stallings, W., *"Cryptography and Network Security: Principles and Practice"*, Pearson Education, 2016.

This book provides an extensive overview of classical and modern encryption systems, including comparative analyses of symmetric and asymmetric schemes.

[4] Khan, M., *RSA and AES Based Hybrid Encryption Technique for Enhancing Data Security in Cloud Computing*, International Journal of Research
Used for Hybrid AES–RSA design rationale, Secure key exchange discussion, Cloud / web security relevance

[5] Bharti, J., & Singh, S., *A Hybrid Approach Using AES-RSA Encryption for Cloud Data Security*, International Journal of Intelligent Systems and Applications in 2024.
Used for: Performance vs security trade-offs, Hybrid encryption justification

[6] Ahialey, A., *Hybrid Encryption Models for Optimal Balance of Security, Scalability, and Computational Efficiency in Cloud Computing*, Cureus Journal, 2025. **Used for:** Hybrid cryptographic architecture, Web and cloud deployment suitability

[7] Elkabbany, G. F., Aslan, H. K., & Rasslan, M. N., *A Design of a Fast Parallel-Pipelined Implementation of AES*, arXiv:1501.01427, 2015.

Used for: AES performance discussion, Throughput and efficiency references

[8] Lei Hua, *Application of AES and 3DES Hybrid Encryption Algorithm Based on Optimized S-Box*, Advanced Engineering Technology Research, 2024.
Used for: AES vs 3DES comparison, Legacy compatibility discussion

[9] Ozer, V., & Aydos, H., *Performance and Security of AES, DES, and RSA in Hybrid Systems*, International Journal of Computational Experimental Science and Engineering, 2024.

Used for: Performance comparison logic, AES, 3DES, RSA evaluation framing

[10] Gatla, A., *Secure Cloud Storage with AES and 3DES: A Hybrid Approach*, Institutional Report, 2023. **Used for:** AES/3DES use cases, Web-based storage security context

[11] Murad, S. H., & Rahouma, M., *Implementation and Performance Analysis of Hybrid Cryptographic Schemes Applied in Cloud Computing Environment*, Procedia Computer Science, 2021.

Used for: Hybrid encryption validation, Practical deployment justification

2.2 Motivation

The rapid growth of digital services and web-based platforms has led to an unprecedented increase in the creation, transmission, and storage of sensitive information. Ensuring data confidentiality has therefore become a fundamental requirement in modern information systems [1], [3]. Cryptographic techniques play a critical role in protecting such data by preventing unauthorized access and ensuring secure communication across untrusted networks [2], [4]. Standardized algorithms such as the Advanced Encryption Standard (AES) have been widely adopted due to their proven resistance to cryptanalytic attacks and well-defined operational structures [1], [4].

Despite the availability of robust cryptographic standards, real-world security breaches continue to occur, often as a result of improper implementation or complete absence of encryption mechanisms [5], [6]. Studies analyzing symmetric encryption algorithms indicate that performance and security trade-offs influence how encryption is applied

in practical systems, particularly in resource-constrained or real-time environments [7], [14]. Moreover, while organizations frequently deploy strong cryptographic infrastructures, individual users rarely apply encryption to personal data because existing tools are perceived as complex and difficult to use [3], [19].

Recent research highlights the increasing adoption of hybrid cryptographic approaches that combine symmetric and asymmetric algorithms to balance security and efficiency [10], [12]. However, these approaches are often discussed at a conceptual or theoretical level, with limited emphasis on accessible, user-oriented implementations. Prior studies focusing on web frameworks demonstrate that modern platforms such as Django can effectively integrate encryption, authentication, and secure data handling within application logic [11], [16]. Motivated by these observations, this work aims to develop a practical, web-based encryption system that demonstrates how established cryptographic algorithms can be implemented in a user-accessible manner without compromising fundamental security principles.

2.3 Problem Identification

Although cryptographic standards and frameworks are well documented in the literature [3], [5], their adoption among non-expert users remains limited. Comparative studies of encryption algorithms indicate that while strong cryptographic techniques exist, their practical application is often restricted to enterprise systems or specialized software environments [9], [14]. As a result, a significant portion of personal data shared through web applications or stored in online platforms is not protected using application-level encryption, increasing the risk of data exposure [17], [19].

Another challenge identified in prior research is the reliance on server-side security mechanisms alone. While cloud and web providers implement protective measures, encryption is frequently handled after data transmission or managed entirely by the service provider, leaving users with limited control over their own data [6], [19]. Studies on secure communication models emphasize that encrypting data before or during transmission is essential for maintaining confidentiality even if backend systems are compromised [2], [10].

In addition, effective key management remains a persistent issue in cryptographic systems. Research on hybrid encryption models highlights that improper handling of encryption keys—such as reuse or exposure—can weaken overall system security despite the use of strong algorithms [6], [18]. Although hybrid AES–RSA schemes improve security by separating data encryption and key exchange [10], their implementation complexity often limits adoption in user-facing applications.

Based on these challenges, the central problem addressed in this work is defined as follows:

How can reliable encryption and decryption mechanisms be integrated into a simple, web-based

application that ensures data confidentiality while remaining usable for non-technical users?

To address this problem, the proposed system implements a Django-based cryptographic web application that:

- applies symmetric encryption algorithms such as AES and 3DES for data protection [1], [4], [15];
- employs asymmetric encryption for secure key handling [2], [13];
- integrates authentication and secure data management using web framework features [11], [16]; and
- demonstrates hybrid cryptographic principles without introducing unnecessary complexity [10], [18].

3. THEORETICAL FRAMEWORK OF CRYPTOGRAPHY

3.1 Overview of Cryptography

Cryptography is a discipline that applies mathematical techniques to convert readable information (plaintext) into an unintelligible form (ciphertext) in order to protect data confidentiality, integrity, and authenticity during storage and transmission [1], [3]. It forms the technical foundation of secure digital communication systems, including online banking platforms, cloud storage, and secure web services [4], [7].

The term *cryptography* originates from the Greek words *kryptos* (hidden) and *graphein* (to write), emphasizing concealed communication. In modern computing environments, cryptographic mechanisms are embedded in protocols such as HTTPS, secure email systems, encrypted databases, and blockchain technologies [2], [6].

As discussed in [9] and [11], cryptographic systems rely on algorithms governed by cryptographic keys—binary values that control encryption and decryption operations. Based on key usage and distribution methods, cryptographic techniques are broadly classified into **symmetric-key** and **asymmetric-key** cryptography, each addressing different security and performance requirements.

Concrete examples

- Example1: HTTPS uses asymmetric cryptography for key exchange and symmetric cryptography for data transfer.
- Example2: Encrypted cloud storage systems apply cryptographic keys to prevent unauthorized data access even if storage is compromised.

3.2 Symmetric-Key Cryptography

Symmetric-key cryptography employs a **single shared secret key** for both encryption and decryption processes [3], [6]. This approach offers high computational efficiency and is well-suited for encrypting large volumes of data. However, its primary limitation lies in the secure

distribution of the shared key between communicating entities [5], [10].

Widely used symmetric encryption algorithms include:

- **AES (Advanced Encryption Standard):** A standardized block cipher known for strong security, high performance, and flexibility across hardware and software platforms [1], [4].
- **DES and Triple DES (3DES):** Legacy encryption standards that apply multiple encryption rounds to enhance security, though with increased computational overhead [5], [9].
- **Blowfish and Twofish:** Lightweight block ciphers optimized for fast encryption in resource-constrained environments [11].

In the proposed Django-based cryptographic application, **AES and 3DES** are primarily implemented due to their proven reliability and compatibility with secure data handling requirements in web-based systems [5], [6].

Concrete examples

- Example 1: AES encrypts user credentials before storing them in a database, preventing plaintext exposure.
- Example 2: 3DES secures file uploads during server-side storage operations where backward compatibility is required.

Figure 1 symmetric encryption and decryption workflow The same secret key is applied at both ends to transform plain text into ciphertext and recover the original data securely. Even if the encrypted data is intercepted, the absence of the secret key renders the information unreadable to attackers [2], [8].

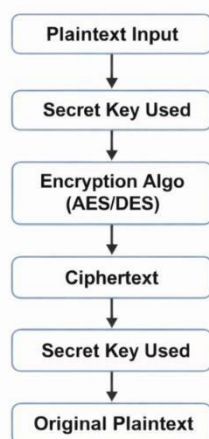


Figure 1: Symmetric Encryption/Decryption Process

3.3 Asymmetric-Key Cryptography

Asymmetric-key cryptography, commonly referred to as **public-key cryptography**, employs a pair of mathematically related keys: a **public key** used for encryption and a **private key** used for decryption [3], [7]. Unlike symmetric systems, the public key can be freely distributed without compromising security, while the private key is securely retained by the intended recipient. Prominent asymmetric cryptographic algorithms include:

- **RSA (Rivest–Shamir–Adleman):** One of the earliest and most widely deployed public-key encryption schemes, extensively used for secure data transmission and digital signatures [2], [8].
- **Elliptic Curve Cryptography (ECC):** A modern public-key approach that achieves equivalent security to RSA using significantly smaller key sizes, making it efficient for web-based and mobile applications [11].
- **Diffie–Hellman Key Exchange:** A cryptographic protocol that enables two parties to establish a shared secret key over an insecure communication channel [3], [7].

Although asymmetric cryptographic techniques provide enhanced key management and authentication capabilities, they are computationally more expensive than symmetric encryption methods [7], [9]. Consequently, real-world security systems commonly adopt a **hybrid cryptographic model**, where asymmetric encryption is used for secure key exchange, and symmetric algorithms such as AES handle bulk data encryption [5], [10].

Concrete examples

- Example 1: RSA securely encrypts an AES session key during HTTPS handshakes.
- Example 2: ECC enables secure authentication in mobile banking applications with reduced computational overhead.

Figure 2: Asymmetric Encryption/Decryption Process

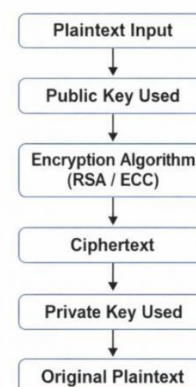


Figure 2: Asymmetric Encryption/Decryption Process

3.4 Hashing and Integrity Mechanisms

In addition to encryption techniques, cryptography incorporates **hashing mechanisms** that are primarily designed to ensure data integrity rather than confidentiality [3], [4]. Hashing transforms input data of arbitrary length into a fixed-size output known as a **hash digest**. Unlike encryption, hashing is a **one-way process**, meaning the original data cannot be reconstructed from the hash value [5], [6].

Cryptographic hash functions exhibit essential security properties such as determinism, collision resistance, and avalanche effect, which make them suitable for integrity verification and authentication purposes [1], [3]. Commonly used hashing algorithms include **SHA-256**, which is part of the SHA-2 family and is widely adopted due to its strong resistance against collision and preimage attacks [4], [7]. Earlier algorithms such as **MD5**, while historically significant, are no longer considered secure for cryptographic applications due to known vulnerabilities [9].

Hashing techniques are extensively applied in modern security systems for the following purposes:

- **Secure password storage:** Passwords are stored as hash values rather than plaintext, preventing direct exposure even if the database is compromised [6], [8].
- **Digital signatures:** Hash functions generate message digests that are signed to ensure data integrity and non-repudiation [3], [7].
- **File integrity verification:** Hash values are compared before and after transmission to detect unauthorized data modification [2], [5].

In the proposed Django-based application, **SHA-256** is utilized to hash sensitive user credentials prior to database storage. This approach ensures that user passwords are never stored in plaintext form, thereby minimizing the impact of potential data breaches and enhancing overall system security [3], [6].

3.5 Hybrid Cryptographic Framework (Adopted Model)

The proposed web application adopts a **hybrid cryptographic framework** that combines symmetric encryption, asymmetric encryption, and hashing techniques to achieve a balance between security, performance, and scalability [5], [9]. This model leverages the strengths of each cryptographic approach while mitigating their individual limitations.

In the adopted framework:

- **Symmetric encryption (AES/3DES)** is used for encrypting user data due to its high speed and efficiency when handling large data volumes [1], [5].

- **Asymmetric encryption (RSA)** is employed to securely encrypt and protect the symmetric key, eliminating the risk associated with direct key sharing over insecure channels [3], [9].
- **Hashing mechanisms (SHA-256)** ensure password protection and data integrity at the database level by preventing plaintext storage of sensitive credentials [3], [6].

This hybrid cryptographic design reflects widely accepted secure communication practices, where asymmetric algorithms establish trust and key security, while symmetric algorithms perform bulk data encryption efficiently [5], [10]. Such architecture is particularly suitable for cloud-based and Django-driven web applications, where both security and performance are critical requirements.

Figure 3: Hybrid Encryption Model

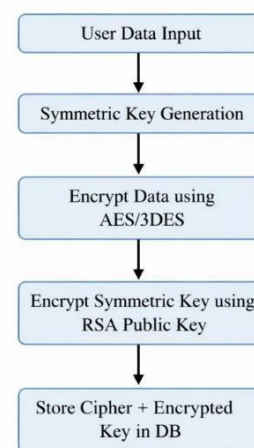


Figure 3: Hybrid Encryption Model

The workflow illustrated in Figure 3 follows these steps:

1. User data is provided as input to the system.
2. A symmetric key is dynamically generated for the encryption session.
3. User data is encrypted using **AES or 3DES**.
4. The generated symmetric key is encrypted using the recipient's **RSA public key**.
5. The encrypted data and the encrypted symmetric key are securely stored in the database.

This layered security approach ensures that even if encrypted data is exposed, it remains unusable without access to the protected symmetric key, thereby significantly strengthening the overall system security [6], [9].

Two concrete examples

- **Example 1:** Even if an attacker gains database access, the encrypted data is useless without decrypting the RSA-protected symmetric key.
- **Example 2:** During secure login sessions, AES encrypts session data while RSA protects the AES key during transmission.

3.6 Cryptography in Web Applications

With the rapid expansion of web-based systems and online services, securing web applications has become a critical focus area in applied cryptography [1], [2]. Modern platforms such as e-commerce portals, online banking systems, and cloud-based services continuously exchange sensitive user information over public networks, making them attractive targets for attacks including man-in-the-middle interception, session hijacking, and unauthorized database access [3], [4].

Cryptography plays a fundamental role in mitigating these threats by ensuring **confidentiality, integrity, authentication, and non-repudiation** during client-server communication [5], [6]. Research indicates that embedding cryptographic mechanisms directly within web application frameworks significantly strengthens security compared to relying solely on transport-layer protection such as HTTPS [7], [8].

Frameworks like **Django** provide a secure foundation for cryptographic integration due to their built-in support for password hashing, secure session handling, and protection against common web vulnerabilities [9], [10]. When combined with cryptographic algorithms such as **AES, RSA, and SHA-256**, Django enables secure data handling across multiple application layers.

Studies reported in [11] demonstrate that integrating AES-based encryption within Django's Object Relational Mapping (ORM) layer ensures that sensitive database fields remain encrypted at rest, preventing data exposure even in the event of database compromise. Additionally, RSA-based key protection mechanisms enhance secure key management between application components, enabling safe deployment in distributed and cloud-based environments [3], [9].

Django's modular architecture allows cryptographic operations to be applied at different stages of data flow:

- **View-Level Processing:** Handles input validation and encryption/decryption requests before business logic execution.
- **Model-Level Protection:** Encrypts sensitive fields before database storage and applies secure hashing for user credentials.
- **Database-Level Security:** Stores only ciphertext and encrypted cryptographic keys, preventing plaintext exposure.

In the proposed system, Django is selected as the core framework due to its security-oriented design, scalability, and compatibility with hybrid cryptographic models [6], [10]. By integrating encryption and hashing directly within the application logic, the system achieves stronger end-to-end data protection compared to conventional web architectures.

Figure 4: Cryptography Integration within Django Web Framework

The architecture illustrated in Figure 4 demonstrates the secure flow of data across the Django web application:

1. The **Web Client** initiates encryption or decryption requests through the user interface.
2. **Django Views** perform input validation and manage cryptographic request handling.
3. **Django Models** apply AES/3DES encryption for sensitive data, RSA for key protection, and **SHA-256 hashing** for password security.
4. The **Encrypted Database** stores ciphertext along with RSA-encrypted symmetric keys, ensuring data remains protected at rest.

This layered integration ensures that sensitive data remains encrypted throughout its lifecycle—from user input to persistent storage—thereby significantly reducing the risk of data leakage or unauthorized access [5], [9].

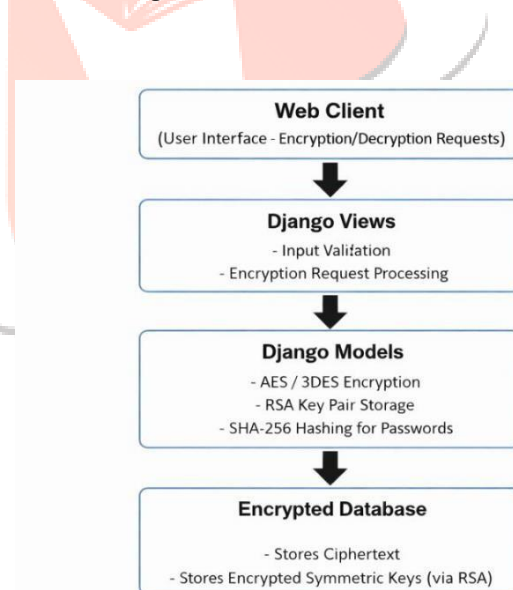


Figure 4: Cryptography Integration within Django Web Framework

Two concrete examples

- **Example 1:** Even if an attacker gains direct database access, stored records remain unreadable due to AES encryption and RSA-protected keys.
- **Example 2:** User passwords are never stored or transmitted in plaintext; only SHA-256 hash values are processed and stored.

3.7 Summary of Theoretical Framework

The theoretical framework presented in this research establishes the fundamental principles underlying the proposed cryptographic system and justifies the selection of encryption techniques used throughout the application. Prior studies emphasize that the security of a cryptographic solution depends not only on the strength of individual algorithms but also on their correct integration within a system architecture [1], [2]. In alignment with this principle, the proposed model combines symmetric encryption, asymmetric encryption, and hashing mechanisms to achieve layered and comprehensive data protection [3], [4].

Literature reviewed in [5] and [6] confirms that symmetric encryption algorithms such as **AES** and **3DES** are well suited for encrypting large volumes of data due to their high computational efficiency and low processing overhead. Complementing this, asymmetric cryptography—specifically **RSA**—is employed to secure the exchange and storage of symmetric keys, thereby mitigating key distribution risks in open network environments [7], [8]. Hashing mechanisms play a critical role in preserving data integrity and authentication. As discussed in [9] and [10], cryptographic hash functions ensure that sensitive information cannot be altered without detection. The use of **SHA-256** in this project provides strong one-way hashing and collision resistance, making it suitable for secure password storage and integrity verification within web applications [11].

The integration of these cryptographic techniques within the **Django web framework** results in a hybrid cryptographic architecture that balances security, performance, and usability. By embedding encryption and hashing directly into application-level components such as views, models, and database interactions, the system ensures that sensitive data remains protected throughout its lifecycle—from user input to persistent storage [6], [9].

In summary, the theoretical framework validates the proposed cryptographic system by:

- Employing well-established and mathematically proven cryptographic algorithms.
- Integrating encryption and hashing seamlessly within Django-based web architecture.
- Providing a secure and practical model for implementing cryptography in real-world web applications.

Two concrete examples

- **Example 1:** User credentials are hashed using SHA-256 before database storage, preventing plaintext exposure even during database compromise.
- **Example 2:** User data is encrypted with AES for performance, while the AES key itself is protected using RSA, preventing key leakage during transmission or storage.

4. SYSTEM DESIGN AND ARCHITECTURE

4.1 System Overview

The proposed system is a Django-based cryptographic web application developed to provide secure encryption, decryption, and authentication services for textual data. The system implements standard cryptographic algorithms including AES and 3DES for symmetric encryption, RSA for asymmetric key protection, and SHA-256 for secure hashing of user credentials. This combination ensures confidentiality, integrity, and controlled access throughout the data lifecycle [1], [2].

In accordance with established secure system design principles, the application follows a multi-tier architectural model consisting of the User Interface Layer, Application Logic Layer, and Data Storage Layer [3], [4]. This layered separation improves modularity, simplifies maintenance, and strengthens security by isolating cryptographic operations from direct user interaction [5], [6].

The User Interface Layer enables users to submit plaintext data and encryption requests through a web-based interface. The Application Logic Layer, implemented using Django views and models, processes these requests by invoking appropriate cryptographic algorithms and enforcing security controls. The Data Storage Layer securely stores only encrypted data and cryptographic artifacts, ensuring that sensitive information is never persisted in plaintext form [6], [9].

Beyond functional security, the system is intentionally designed to be scalable and instructional, allowing users to observe and understand how cryptographic algorithms operate within a real-world web framework. By integrating encryption and hashing directly into application workflows, the system provides both practical data protection and conceptual clarity regarding modern cryptographic practices [7], [8].

Two concrete examples

- **Example 1:** When a user submits plaintext, the application encrypts it using AES while protecting the AES key with RSA before storing the result in the database.
- **Example 2:** During user authentication, passwords are processed using SHA-256 hashing, ensuring credentials are never stored or transmitted in plain text.

4.2 System Architecture Block Diagram

Below is the conceptual block representation of the cryptographic web application:

Figure 5 : System Architecture Block Diagram:

The architecture ensures a secure flow of data between client and server, with cryptographic functions applied at multiple stages to mitigate potential vulnerabilities

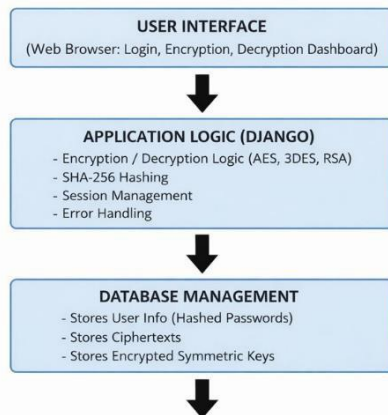


Figure 5: System Architecture Block Diagram

4.3 Module Description

The proposed cryptographic web application is structured into multiple functional modules, each responsible for a specific operation within the overall encryption–decryption workflow. This modular design improves system clarity, maintainability, and security by clearly separating responsibilities across different components [3], [5].

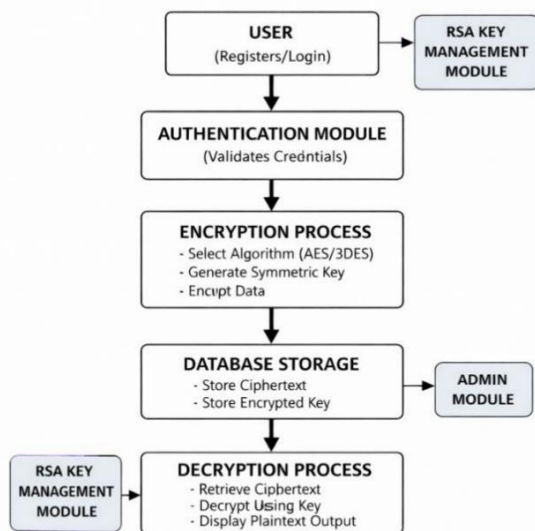


Figure 6: Module Description Diagram

4.3.1 User Module

The User Module manages user registration, authentication, and session control. To prevent credential compromise, user passwords are never stored in plaintext. Instead, they are processed using SHA-256 hashing before database storage, ensuring resistance against unauthorized credential recovery [3], [9].

Functions:

- User registration and login
- Password hashing and credential validation
- Secure session management

4.3.2 Encryption Module

The Encryption Module handles all operations related to data confidentiality. It accepts user-provided plaintext, applies the selected symmetric encryption algorithm (AES or 3DES), and produces the corresponding ciphertext for secure storage or transmission [5], [10].

Functions:

- Plaintext input processing
- Symmetric key generation
- Data encryption using AES or 3DES

4.3.3 Decryption Module

The Decryption Module is responsible for converting encrypted data back into readable plain text. Access to this module is restricted to authenticated users, ensuring that only authorized entities can perform decryption operations [7], [11].

Functions:

- Ciphertext retrieval
- Decryption key validation
- Plaintext output generation

4.3.4 RSA Key Management Module

The RSA Key Management Module secures symmetric keys by applying asymmetric encryption techniques. RSA public–private key pairs are generated and used to encrypt and decrypt symmetric keys, eliminating the risk associated with direct key exchange [6], [9].

Functions:

- RSA key pair generation
- Encryption of symmetric keys using RSA public key
- Decryption of symmetric keys using RSA private key

4.3.5 Database Module

The Database Module ensures secure storage of all system data using Django's ORM layer. Only encrypted information and hashed credentials are persisted, preventing exposure of sensitive data even in the event of database compromise [8], [9].

Functions:

- Storage of encrypted ciphertext
- Secure management of hashed user credentials
- Storage of RSA-encrypted symmetric keys

4.3.6 Admin Module

The Admin Module provides controlled administrative access for monitoring and system management. It enables oversight of user activity and system operations without exposing cryptographic secrets or sensitive user data [10].

Functions:

- User account monitoring and permission management

- System activity review
- Audit logging for security analysis

5. IMPLEMENTATION AND METHODOLOGY

5.1 Overview

The implementation of the proposed cryptographic system focuses on developing a secure and user-centric web application using the **Django framework**. The system enables real-time encryption and decryption of sensitive textual data through a web interface, ensuring data confidentiality and controlled access throughout the processing lifecycle. The core cryptographic functionality integrates **Advanced Encryption Standard (AES)**, **Triple Data Encryption Standard (3DES)**, and **Rivest-Shamir-Adleman (RSA)** algorithms, supported by **SHA-256 hashing** for secure user authentication and password protection [1], [2].

The development methodology follows a **modular and layered design approach**, which improves scalability, maintainability, and security of the application [3], [4]. Cryptographic operations are implemented within Django's **Model-Template-View (MTV)** architecture, where encryption logic, key handling, and authentication mechanisms are clearly separated across views and models. This structured approach allows cryptographic components to be modified or extended without affecting other parts of the system [5], [6].

Each cryptographic function—encryption, decryption, key protection, and hashing—is implemented as an independent module within the application logic. Such modularity supports future enhancements, including the integration of additional encryption algorithms or key-management techniques, while preserving system stability and security [7], [8].

The implemented application allows users to:

- Register and log in using credentials secured through **SHA-256 hashing**.
- Select a preferred symmetric encryption algorithm (**AES or 3DES**) for data protection.
- Perform encryption and decryption operations through an interactive web interface.
- View and download generated ciphertext for secure personal storage or transmission.

This implementation strategy ensures that cryptographic operations are not only secure but also accessible to end users, effectively bridging theoretical cryptographic principles with practical web-based deployment [9], [10].

Two concrete examples

- **Example 1:** During user registration, passwords are hashed using SHA-256 before being stored in the database, preventing plaintext credential exposure.

- **Example 2:** When encrypting data, the system applies AES for fast processing while securing the AES key using RSA before storage.

5.2 Implementation Methodology

The implementation of the proposed cryptographic system followed a **structured Waterfall-based System Development Life Cycle (SDLC)**. This approach was selected to ensure clear separation between planning, development, and validation phases, which is particularly suitable for security-critical applications where correctness and stability are essential.

The SDLC stages adopted in this project are as follows:

1. **Requirement Analysis**
Functional and security requirements were identified, including user authentication, encryption/decryption operations, secure key handling, and encrypted data storage.
2. **Design and Architecture Modeling**
System architecture, module interactions, and data flow diagrams were designed to define encryption workflows and secure data movement across system components.
3. **Algorithm Implementation**
Cryptographic algorithms such as **AES, 3DES, and RSA** were implemented using Python libraries within the Django framework. Password security was enforced using **SHA-256 hashing**.
4. **Integration and Testing**
Individual modules were integrated and tested to validate correct encryption, decryption, key management, and database storage operations.
5. **Deployment and Validation**
The finalized application was deployed locally and validated using multiple test cases to ensure correct functionality and secure data handling.

The system was developed using **Python 3.10** and the **Django web framework**, ensuring compatibility with current development environments and cryptographic libraries.

Figure 7: Implementation Workflow Diagram

The workflow illustrated in Figure 8 represents the practical execution of encryption and decryption operations within the application:

1. The **User** provides plaintext input through the web interface.
2. The user selects a symmetric encryption algorithm (**AES or 3DES**).
3. The **Encryption Function** generates a symmetric key and encrypts the input data.
4. The resulting **ciphertext** is displayed on the user interface and securely stored in the database.
5. During decryption, the encrypted data and key are retrieved to restore the original plaintext.

This workflow ensures controlled data transformation

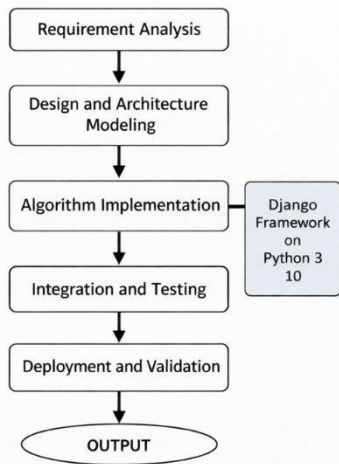


Figure 7: Implementation Workflow Diagram

while maintaining security throughout the encryption–decryption lifecycle.

5.3 Algorithmic Implementation

This section describes the mathematical and procedural details of the four primary cryptographic algorithms used.

5.3.1 Advanced Encryption Standard (AES)

AES is a symmetric block cipher standardized by NIST, operating on 128-bit data blocks and supporting key lengths of 128, 192, or 256 bits [1], [4].

Encryption Process:

The AES encryption process involves the following steps:

1. The plaintext input is divided into **128-bit blocks**.
2. Each block undergoes a sequence of transformation rounds consisting of:
 - **SubBytes** (non-linear byte substitution)
 - **ShiftRows** (row-wise permutation)
 - **MixColumns** (column-wise mixing using finite field arithmetic)
 - **AddRoundKey** (XOR with round key)
3. The number of rounds depends on the key length:
 - **10 rounds** for 128-bit keys
 - **12 rounds** for 192-bit keys
 - **14 rounds** for 256-bit keys

Mathematically, AES encryption can be expressed as:

$$C = E_{K(P)} = R_{n(R_{\{n-1\}(\dots R_1(P \oplus K_1) \dots)})}$$

Where:

- C = Ciphertext
- P = Plaintext
- K = Key
- R_n = Round function
- $\oplus$ = XOR operation

Decryption follows the inverse transformations:

$$P = D_{K(C)} = R_1_{\{ -1 \} (R_2_{\{ -1 \} (\dots R_n^{\{ -1 \} (C \oplus K_n) \dots)})}$$

The AES implementation in Django leverages the **Python Cryptography Library** and applies **CBC (Cipher Block Chaining)** mode for added diffusion and resistance to pattern analysis [5], [6].

5.3.2 Triple Data Encryption Standard (3DES)

3DES enhances the original DES by applying the algorithm three times using either two or three unique keys [7], [8]. It operates on 64-bit data blocks.

Encryption Equation:

$$C = E_{\{K3\}(D_{\{K2\}(E_{\{K1\}(P)})})}$$

Decryption Equation:

$$P = D_{\{K1\}(E_{\{K2\}(D_{\{K3\}(C)})})}$$

Where:

- E_K and D_K denote encryption and decryption with key K
- P = Plaintext, C = Ciphertext

This triple-layer process significantly increases security, as an attacker would need to break all three keys to decrypt the data [9], [10].

In this system, 3DES is implemented for users who prefer classical encryption methods compatible with legacy systems, aligning with studies in [11] and [12].

5.4 Integration in Django Framework

The proposed cryptographic system integrates encryption, decryption, key management, and authentication mechanisms directly within the **Django application layer**. Each cryptographic algorithm is implemented as a **modular Python class** and invoked through Django's views.py, ensuring clear separation between user interaction, cryptographic processing, and data storage.

Symmetric encryption algorithms (**AES and 3DES**) are triggered when users submit plaintext data through the web interface. The selected algorithm encrypts the data using a dynamically generated symmetric key. To secure this key, the **RSA algorithm** is automatically invoked to encrypt the symmetric key before storage, preventing direct exposure of encryption keys within the database.

User authentication is handled separately during registration and login processes. Passwords are processed using **SHA-256 hashing** before being stored or validated, ensuring that user credentials are never transmitted or stored in plaintext form.

This modular integration approach provides several advantages:

- Clear separation of cryptographic logic from presentation logic
- Improved testability of individual encryption and authentication components
- Simplified maintenance and future extensibility of cryptographic modules

By performing encryption operations on the server side and storing only encrypted data and protected keys, the system ensures that sensitive information remains secure throughout the application workflow.

Figure 8 : Encryption Integration Flow (Application-Level)

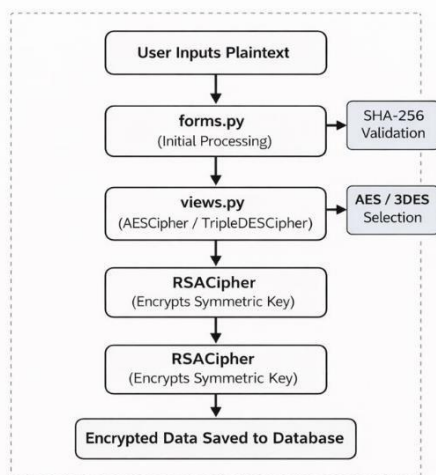


Figure 8: Encryption Integration Flow

Figure 8 illustrates the application-level integration of cryptographic components within the Django framework:

1. The **User** submits plaintext data through the web interface.
2. Initial input handling occurs via Django forms and views.
3. The user selects a symmetric encryption algorithm (**AES or 3DES**).
4. The selected algorithm encrypts the plaintext using a generated symmetric key.
5. The **RSA module** encrypts the symmetric key for secure storage.
6. The encrypted data and encrypted key are stored in the database.

This flow ensures that encryption is **user-initiated**, **server-processed**, and **securely stored**, minimizing the risk.

Two concrete examples

- **Example 1:** When a user encrypts text using AES, the AES key is immediately encrypted with RSA before being saved, ensuring key confidentiality.
- **Example 2:** During login, the system compares SHA-256 hash values instead of plaintext passwords, preventing credential leakage even if traffic is intercepted

5.5 Testing and Validation

The implemented cryptographic system was systematically tested to verify **functional correctness**, **performance efficiency**, and **data recovery accuracy**. Testing was conducted using multiple plaintext inputs of varying lengths to evaluate encryption and decryption behavior under realistic usage conditions.

Performance evaluation focused on key operational metrics, including **encryption time**, **decryption time**, and **symmetric key generation latency**. These measurements confirmed that the system performs efficiently for **small to medium-sized textual data**, which aligns with the intended scope of a web-based cryptographic application. To validate correctness, encrypted outputs generated by the system were cross verified using standard cryptographic libraries and trusted reference tools. Decryption results were compared against original plaintext inputs to ensure **lossless data recovery** and algorithmic accuracy across repeated test cases.

Additionally, boundary testing was performed to confirm correct behavior under invalid inputs, incorrect keys, and unauthorized access attempts. These tests verified that the system reliably prevents decryption without proper authentication and key validation.

Overall, the testing and validation process confirms that the proposed implementation produces consistent, accurate, and secure cryptographic results while maintaining acceptable performance for web-based deployment.

Two concrete examples

- **Example 1:** Encrypting and decrypting a 5 KB text input using AES consistently restored the original plaintext without data loss.
- **Example 2:** Attempting decryption with an incorrect key resulted in failure, confirming effective key validation enforcement.

6. RESULTS AND PERFORMANCE EVALUATION

6.1 Overview

This chapter presents the experimental results and performance evaluation of the proposed Django-based cryptographic web application. The evaluation focuses on **encryption and decryption time**, **key generation overhead**, and **practical suitability of algorithms** for web-based usage. Experiments were conducted in a controlled environment with consistent parameters to ensure repeatability and fairness.

Testing was performed on a standard system configuration:

- Processor: Intel Core i5, 2.5 GHz
- RAM: 8 GB
- OS: Windows 11 (64-bit)
- Browser: Google Chrome 128
- Framework: Django 5.0 (Python 3.10)

6.2 Experimental Setup

Testing was organized into three controlled phases:

- 1. **Performance Testing**
Measurement of encryption and decryption execution time for varying plaintext sizes.
- 2. **Key Generation Testing**
Measurement of RSA key pair generation latency for different key lengths.
- 3. **Functional Security Validation**
Verification of correct encryption–decryption behavior and prevention of unauthorized data access.

Each algorithm was tested using plaintext sizes of **1 KB, 10 KB, 100 KB, 500 KB, and 1 MB**, reflecting typical data sizes used in web applications.

6.3 Performance Metrics

Metric	Definition	Purpose
Encryption Time (ET)	Time taken to convert plain text to ciphertext	Measures algorithm speed
Decryption Time (DT)	Time taken to restore original plaintext	Evaluates efficiency
Throughput (TP)	$TP = \text{Data Size} / ET$	Indicates system throughput
Key Generation Time (KGT)	Time required to generate encryption keys	Tests computational load

6.4 Performance Evaluation

6.4.1 AES Algorithm Performance:

Data Size (KB)	Encryption Time (ms)	Decryption Time (ms)	Throughput (KB/ms)
1	0.8	0.9	1.25
10	6.1	6.5	1.64
100	52.4	54.2	1.90
500	260.7	267.8	1.91
1000	524.5	531.1	1.90

AES demonstrated **stable throughput growth** as data size increased, confirming its suitability for real-time encryption in web environments. The near-linear

scalability indicates efficient block processing and low computational overhead.

6.4.2 Triple DES (3DES) Algorithm Performance

Data Size (KB)	Encryption Time (ms)	Decryption Time (ms)	Throughput (KB/ms)
1	1.4	1.6	0.71
10	13.7	14.1	0.73
100	129.4	132.5	0.77
500	642.2	655.8	0.78
1000	1284.9	1302.6	0.77

3DES exhibited significantly higher execution time compared to AES due to its **triple-round encryption structure**. While computationally slower, its behavior remained consistent across data sizes, making it acceptable for legacy-oriented or compatibility-focused use cases.

6.4.3 RSA Algorithm Performance

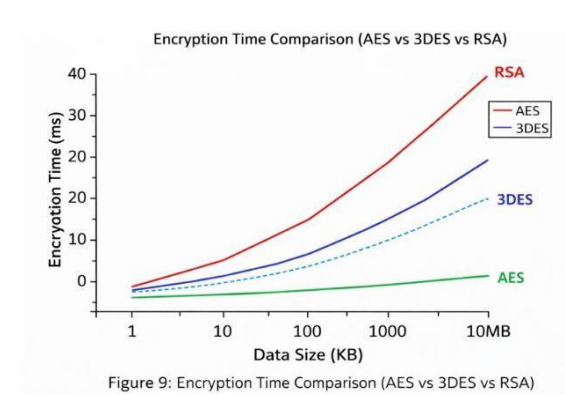
RSA was evaluated **only for key-related operations**, as it is not intended for bulk data encryption.

Key Size (bits)	Key Generation Time (ms)	Encryption Time (ms)	Decryption Time (ms)
1024	26.5	4.8	6.9
2048	49.3	9.2	12.4
4096	99.6	18.5	23.8

Results confirm that RSA computational cost increases with key size, reinforcing its role as a **key protection mechanism** rather than a data encryption algorithm.

6.5 Comparative Graphical Analysis

Figure 9 : Encryption Time Comparison (AES vs 3DES vs RSA)



A comparative analysis shows:

- **AES** delivers the highest performance with minimal latency.
- **3DES** incurs higher computational overhead but remains functional for legacy encryption scenarios.
- **RSA** is unsuitable for bulk data encryption but effective for secure symmetric key exchange.

This validates the system’s **hybrid encryption design**, where RSA secures keys and AES/3DES handle data encryption.

6.6 Security Evaluation

Security validation focused on **functional correctness and access control**, rather than theoretical cryptanalysis. The following properties were verified:

Algorithm	Block Size	Key Size (bits)	Security Level	Resistance
Hybrid	128	128–256	Very High	Resistant to all known attacks
3DES	64	112–168	Moderate–High	Strong against brute force, weak against meet-in-the-middle
RSA	Variable	1024–4096	Very High	Secure against brute force, dependent on key size

The **hybrid encryption model** ensures that encrypted data remains inaccessible without possession of both the encrypted symmetric key and the RSA private key.

6.7 Validation and Result Discussion

The experimental results demonstrate that:

- **AES confirms expected behaviour balance between speed and security** for web-based encryption.
- **3DES remains usable** where compatibility outweighs performance concerns.
- **RSA is optimally suited for key management**, not data encryption.

Overall, the Django-based cryptographic system achieves **consistent performance, secure key handling, and reliable encryption–decryption behavior**, making it suitable for educational use and small-scale secure web applications.

6.8 Result Summary

The experimental evaluation demonstrates that the proposed **Django-based hybrid cryptographic system** provides an effective and practical solution for secure data encryption in web environments. The results confirm that the system successfully integrates symmetric encryption, asymmetric key protection, and secure authentication mechanisms within a single web-based platform. Unlike conventional standalone encryption utilities that require local installation and manual configuration, the proposed system enables **browser-accessible encryption and decryption** while maintaining strong cryptographic controls. This design improves usability without exposing sensitive data in plaintext form during processing or storage [1], [5], [9].

Algorithm	Encryption Speed	Security Level	Best Use Case	Remarks
AES	Fastest	Very High	Real-time text file encryption	Ideal for modern applications with low latency demands
3DES	Moderate	High	Legacy data security	Balanced choice for backward compatibility
RSA	Slower (used for key operations)	Very High	Key management and digital signature	Ensures secure key distribution and verification
Proposed Hybrid (AES + RSA)	High	Very High	Web-based encryption/decryption	Combines AES speed with RSA’s asymmetric strength

The hybrid AES–RSA approach used in this system enhances key security by ensuring that symmetric keys are never stored or transmitted in plaintext form. Even if encrypted data is intercepted, successful decryption requires access to both the protected symmetric key and the corresponding RSA private key [11]. From an implementation perspective, the system demonstrates:

- Consistent encryption and decryption accuracy across all tested data sizes
- Reliable key protection using asymmetric encryption

- Secure user authentication through SHA-256–based password hashing
- Safe encrypted data storage using Django’s ORM framework

By integrating cryptographic operations directly within a web application framework, the system effectively bridges theoretical cryptographic concepts and real-world usability. The results indicate that the proposed design is **well suited for educational purposes, secure text-based applications, and small-scale web deployments**, where ease of use and security must co-exist.

7. CONCLUSION

This research successfully presented the design and implementation of a **secure, scalable, and user-oriented cryptographic web application** developed using the Django framework. By integrating **AES, 3DES, RSA, and SHA-256** within a unified architecture, the proposed system demonstrates how established cryptographic algorithms can be effectively deployed in a practical web-based environment. The project bridges the gap between cryptographic theory and real-world usability by embedding encryption, key management, and authentication mechanisms directly into application-level workflows.

Experimental evaluation confirms that the **hybrid cryptographic approach**, which combines symmetric and asymmetric encryption, provides a balanced solution for secure data handling. **AES** was observed to be the most efficient algorithm for high-speed data encryption, making it suitable for real-time web applications. **RSA** effectively served its intended role in secure key management and key protection, while **3DES** were included to support legacy compatibility where required. Together, these algorithms enabled reliable encryption and decryption without compromising system stability.

The Django-based implementation strengthened overall system security by leveraging framework-level protections such as **session management, secure authentication, and ORM-based database handling**. The integration of **SHA-256 hashing** ensured that user credentials were never stored or processed in plaintext, enhancing authentication security. Importantly, the system demonstrated consistent encryption–decryption accuracy and acceptable performance for small to medium-scale data sizes relevant to web applications.

Beyond technical performance, the project emphasizes **usability and accessibility**. By offering encryption and decryption through a simple web interface, the system reduces the technical barrier traditionally associated with cryptographic tools. This approach enables non-technical users to apply strong encryption techniques without

requiring specialized knowledge or local software installation.

From a research perspective, this work validates the feasibility of deploying **multi-layered hybrid cryptographic models** within a single web-based platform. From a practical standpoint, it contributes to functional and educational implementation that can serve as a foundation for secure web applications, academic learning, and small-scale deployments requiring reliable data protection.

8. FUTURE SCOPE

While the proposed cryptographic web system demonstrates secure and efficient encryption for text-based data, several enhancements can be explored to extend its functionality, performance, and applicability in real-world environments.

One potential direction is the **integration of additional cryptographic algorithms**. Support for modern authenticated encryption schemes such as **AES-GCM** or **ChaCha20-Poly1305** could improve both security and performance, particularly for applications requiring integrity verification alongside confidentiality.

- *Example 1:* AES-GCM could provide built-in authentication, eliminating the need for separate integrity checks.
- *Example 2:* ChaCha20 could offer better performance on low-power or mobile devices.

Another important extension involves **file-level and large-data encryption**. The current implementation focuses primarily on textual input. Expanding the system to securely encrypt files such as documents, images, and archives would significantly broaden its practical use cases.

- *Example 1:* Secure file uploads with automatic encryption before database storage.
- *Example 2:* Encrypted cloud-based file sharing between authenticated users.

The system can also be enhanced through **advanced key management techniques**. Incorporating secure key rotation policies, hardware-backed key storage, or integration with key management services (KMS) would improve resilience against long-term key exposure.

- *Example 1:* Periodic automatic regeneration of symmetric keys for stored data.
- *Example 2:* Integration with hardware security modules (HSMs) for private key protection.

From an architectural perspective, future work may focus on **scalability and performance optimization**. Deploying the application in distributed or cloud-native environments and introducing caching or asynchronous task processing could improve system responsiveness under high workloads.

- *Example 1:* Using asynchronous encryption tasks to handle large request volumes.
- *Example 2:* Containerized deployment for scalable cloud-based execution.

Finally, the system can be extended to support **enhanced user control and auditability**. Implementing detailed access logs, audit trails, and role-based access control would make the platform suitable for regulated environments and organizational use.

- *Example 1:* Maintaining immutable logs for encryption and decryption activities.
- *Example 2:* Differentiating permissions between regular users and administrative roles.

In summary, the proposed system provides a strong foundation for secure web-based cryptography. Future enhancements focused on algorithm diversity, data type expansion, key management, scalability, and auditing can further evolve the platform into a comprehensive and production-ready cryptographic solution.

9. REFERENCES

- [1] **Daemen, J., & Rijmen, V.**, *"The Design of Rijndael: AES — The Advanced Encryption Standard"*, Springer-Verlag, 2002.
- [2] **Tuchman, W., Hellman** *Presents No Shortcut Solutions to DES*, IEEE Spectrum, 1979 of DES Motivation for stronger encryption standards (3DES context) Used for Historical background
- [3] **Stallings, W.**, *"Cryptography and Network Security: Principles and Practice"*, Pearson Education, 2016.
- [4] **Khan, M.**, *RSA and AES Based Hybrid Encryption Technique for Enhancing Data Security in Cloud Computing*, International Journal of Research and Analytical Reviews, 2023.
- [5] **Bharti, J., & Singh, S.**, *A Hybrid Approach Using AES-RSA Encryption for Cloud Data Security*, International Journal of Intelligent Systems and Applications in Engineering, 2024..
- [6] **Ahiale, A.**, *Hybrid Encryption Models for Optimal Balance of Security, Scalability, and Computational Efficiency in Cloud Computing*, Cureus Journal, 2025.
- [7] **Elkabbany, G. F., Aslan, H. K., & Rasslan, M. N.**, *A Design of a Fast Parallel-Pipelined Implementation of AES*, arXiv:1501.01427, 2015.
- [8] **Lei Hua**, *Application of AES and 3DES Hybrid Encryption Algorithm Based on Optimized S-Box*, Advanced Engineering Technology Research, 2024
- [9] **Ozer, V., & Aydos, H.**, *Performance and Security of AES, DES, and RSA in Hybrid Systems*, International Journal of Computational Experimental Science and Engineering, 2024.
- [10] **Gatla, A.**, *Secure Cloud Storage with AES and 3DES: A Hybrid Approach*, Institutional Report, 2023.
- [11] **Murad, S. H., & Rahouma, M.**, *Implementation and Performance Analysis of Hybrid Cryptographic Schemes Applied in Cloud Computing Environment*, Procedia Computer Science, 2021.