SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 23, No. 1, February 2026, 73-99
UDC: 004.056.55:657.242 **DOI: https://doi.org/10.2298/SJEE2601073Z**
Original scientific paper

Encrypted E-Archives and Secure Delivery via Electronic Ledger

Miloš Živković[1], Danijela Milošević[2], Ilja Stanišević[1]

Abstract: Digital document security in modern organizations represents one of the key challenges, particularly in the context of electronic archives and centralized document management systems. This paper proposes a model that enables client-side encryption of confidential documents, secure server-side storage, and controlled and auditable distribution through an electronic ledger system. Special emphasis is placed on modern cryptographic techniques, including hybrid encryption (AES and RSA/ECC), digital signatures, and blockchain technology for immutable audit logs. The model additionally considers post-quantum cryptographic schemes to ensure long-term resistance to quantum attacks. The proposed architecture aims to ensure a high level of confidentiality, integrity, and availability, even in resource-constrained environments, and can be applied across diverse organizational and legal frameworks.

Keywords: Ledger, Document encryption, Document exchange security, Client-side encryption, Blockchain audit, Post-quantum cryptography.

1 Introduction

The security of digital documents within business organizations represents a complex and ongoing challenge. While computers and local devices can be protected through physical and software-based measures, centralized electronic registries – particularly the process of document exchange via the delivery ledger and storage on servers – introduce additional security and organizational risks [1].

Protecting confidential documents begins with access control: only authorized users should have access to content, even when the documents are stored on a server. This requires sophisticated end-to-end encryption, where data is encrypted on the client side and remains encrypted during both storage and distribution.

[1]Western Serbia Academy of Applied Studies, Užice, Valjevo Department, Serbia
milos.zivkovic@vipos.edu.rs, https://orcid.org/0009-0008-6533-661X
ilja.stanisevic@vipos.edu.rs, https://orcid.org/0009-0004-3981-0032
[2]Faculty of Technical Sciences Čačak, University of Kragujevac, Serbia, danijela.milosevic@ftn.kg.ac.rs, https://orcid.org/0000-0002-4763-7012

In their study, Muñoz-Hernández et al. [2] describe a combination of encryption and digital fingerprinting mechanisms that enable tracking of document misuse, including insider threats.

For document distribution and versioning, Han et al. [3] developed a decentralized system based on blockchain technology, the InterPlanetary File System (IPFS), and Shamir's Secret Sharing. In their model, an AES-encrypted document is stored on IPFS, while access key fragments are distributed via smart contracts, ensuring that only authorized users can reconstruct and decrypt the content.

The use of blockchain technology [4] for authentication, IPFS-based distribution, and hybrid cryptography can enhance data integrity and confidentiality.

Electronic document security within organizations relies on the core CIA principles – Confidentiality, Integrity, and Availability – which include:

– The implementation of symmetric and asymmetric cryptographic schemes, digital signatures, and end-to-end security;
– Models for the distribution and storage of documents with key tracking and integrity preservation;
– Document management through the delivery ledger, including classification, archiving, and a legal framework for record-keeping;
– Adherence to national legislation and internal organizational security policies.

The aim of this paper is to transform these concepts into practical architecture for an electronic registry system. The architecture is structured around the following core components:

– Client-side encryption (ensuring that neither the server nor intermediaries can access plaintext);
– Secure storage in an encrypted repository (IPFS or traditional server storage);
– Controlled document distribution through the delivery ledger;
– Auditing and monitoring to detect misuse.

The protocols referenced in this work rely on algorithms that are computationally infeasible for classical computers, but may become vulnerable to quantum attacks as quantum computing capabilities advance [5]. Therefore, the model incorporates post-quantum cryptographic mechanisms to ensure long-term security against anticipated quantum attacks [6, 7].

1.1 Related work

Numerous studies have addressed the challenges of secure document storage and exchange, particularly in decentralized environments. Han et al. [4] proposed a blockchain-based system utilizing IPFS and Shamir's Secret Sharing-based

schemes to achieve secure storage and access control. Similarly, Shechtman and Waisbard [16] developed a tamper-proof distributed log system leveraging blockchain technology to ensure immutable auditing.

Muñoz-Hernández et al. [2] examined the integration of digital fingerprinting with encryption techniques to detect and prevent internal data misuse, while Nizamuddin et al. [3] explored version control and decentralized distribution of encrypted documents using Ethereum smart contracts.

Most existing approaches depend on distributed infrastructures that require complex deployment procedures and advanced technical knowledge on the user side. In contrast, this paper focuses on secure document encryption, storage, and distribution within centralized environments, such as institutional electronic registries, using widely available technologies. Unlike prior decentralized solutions that may incur high operational costs, the proposed architecture uses middleware for local encryption and decryption, thereby maintaining end-to-end confidentiality without relying on server trust.

This work also introduces a delivery ledger tailored to organizational compliance requirements, providing fine-grained access control, traceability, and auditability – capabilities often underrepresented in open P2P architectures. By combining hybrid encryption (AES with RSA/ECC), digital signatures, and optional blockchain integration for immutable audit logs, the proposed model offers a practical and cost-effective solution for secure document management in legal, governmental, and corporate settings. Furthermore, post-quantum readiness enhances resilience against future quantum-driven attacks.

2 System Architecture

The system requirements are defined to enable the handling of confidential documents without the need for expensive equipment, leased private communication tunnels, or specialized infrastructure, while still ensuring a high level of security, reliability in document exchange, and the authenticity and integrity of transmitted documents. This allows recipients to verify that a document has not been altered in transit.

The proposed architecture is designed to operate securely even in constrained environments and without reliance on server-side trust.

The architecture consists of four main layers:

– The client-side (frontend);

– The storage layer (server);

– The delivery ledger / distribution module;

– The Key Management System (KMS),

in addition to a **middleware application** that performs document encryption /decryption and enables secure exchange with the server [8].

The server functions as an electronic registry, with its design described in [9] and its implementation detailed in [10]. This work builds upon the registry's core functionalities by integrating secure document storage and user-to-user document exchange.

2.1 Client side

The client-side component plays a critical role in preparing documents for archiving. Prior to archival, each document must be encrypted and appropriately processed. Document encryption is performed locally on the user's device via the middleware, and this process must be automated in order to remain transparent to the user.

Technologies and methods used for document preparation include:

– AES-256 symmetric encryption for the document;

– Automatic generation of the encryption key by the software;

– Encryption of the AES key using the recipient's RSA or ECC public key.

For security reasons, web browsers cannot access users' private keys directly. JavaScript cannot access smart cards, tokens, or the operating system's secure key stores. Interaction with these secure elements must be mediated through the middleware application.

Within the proposed architecture, the browser is only responsible for metadata exchange and for initiating cryptographic operations, while the middleware performs all sensitive functions. Communication between the browser and the middleware is implemented through API calls executed by the middleware application [11].

The communication rules between the browser and the middleware are governed by CORS (Cross-Origin Resource Sharing) policies [8, 12], the detailed security implications of which are outside the scope of this paper and remain a subject of separate research.

2.2 Storage layer (server)

The role of the storage layer is to retain only encrypted files and their associated metadata (document name, date, status, user ID), without access to the document's original content. This ensures that the server functions strictly as a repository without participating in cryptographic operations.

Security measures include [1]:

– Physical separation of the database and file storage;

– Authorization enforced on all access requests using JSON Web Tokens (JWT);

– Audit logging of all file-related events [13];

– Access control based on user identity and document status.

Table 1
Comparative Analysis – IPFS vs. Traditional Server Storage.

Characteristic	IPFS Storage	Traditional Server
Architecture	Decentralized P2P network	Centralized server / file system
Data Security	Hash-based addressing, immutable data	Depending on implementation; it may allow changes without trace
Resilience to Data Loss	High distributed across multiple nodes	Depends on server backup policies
Scalability	Easily scalable – shared storage across nodes	Requires hardware scaling and maintenance
Privacy	Requires additional encryption (IPFS is public)	Server-side authentication controls access
Blockchain Integration	Natively compatible – can reference hashes	Requires additional configuration
Access without Central Authority	Yes	No

Table 1 provides a comparative overview of IPFS-based storage versus traditional centralized servers, focusing on architecture, scalability, privacy, and blockchain compatibility.

Although the proposed system primarily uses a traditional server-based model, it remains compatible with an IPFS-based alternative for organizations requiring decentralized distribution, immutability, or blockchain integration.

2.3 Delivery ledger/distribution module

The delivery ledger is a central module for managing the flow of documents within an organization. Its primary function is to record all activities related to a document, including upload, assignment to authorized personnel, and tracking of status changes throughout the document's lifecycle.

Key functionalities of the delivery ledger include:

– Access tracking;

– Access control;

– Distribution to individuals or groups;

– Validation and audit logging.

All decryption operations occur exclusively on the client side, ensuring that document confidentiality is preserved throughout the document's lifecycle

A comparative analysis of the ledger described in this paper and the solution presented by Shekhtman and Weisbard [14] is shown in **Table 2**.

Table 2
Comparative Analysis – Proposed Ledger System vs. IPFS+Blockchain System.

Characteristic	Proposed System (This Paper)	IPFS + Blockchain System work, Shekhtman and Weisbard [14]
Document Distribution	Centralized via delivery ledger	Decentralized P2P network (IPFS)
Access Control	Middleware validation + RBAC/CapBAC	Smart contracts for access control
Audit Log / Revision	Centralized log (optionally extendable to blockchain)	Blockchain as primary audit mechanism
User Activity Tracking	Detailed logs within the delivery system	Implicit tracking via smart contracts
Flexible Access Rights	Multi-layered: read, forward, sign	Limited based on access tokens
Legal and Organizational Compliance	Designed for institutional use	Suitable for open, distributed networks
Integration with Business Systems	REST API, JWT, middleware-supported	Low – P2P focused, lacks direct ERP/CRM integration
Security Model	End-to-end encryption + user identity	Decentralized encryption and storage

The proposed ledger system is tailored for structured organizations that require controlled, auditable, and compliant document exchange, whereas IPFS+Blockchain systems are generally more suited for open, peer-to-peer environments.

For advanced auditing and forensic tracking, the system can optionally integrate immutable audit logs using blockchain technology [16].

2.4 Key management system (KMS)

The Key Management System (KMS) plays a central role in maintaining the overall security of the system. Its main function is to securely manage encryption keys, including:

– Storage and management of user keys;

– Logging of key usage;

– Provision of public keys for secure document exchange;

– Access rights management;

– Key revocation and rotation.

A comparative analysis of the proposed system with popular alternatives, as provided by Shamir's Secret Sharing (SSS) [17], AWS KMS, and HashiCorp Vault, is presented in **Table 3**.

Table 3

Comparative Analysis functionality of Key Management Approaches.

Characteristic	Proposed System (This Paper)	AWS KMS	HashiCorp Vault	Shamir's Secret Sharing (SSS) [17]
Key Storage	Local, centralized KMS	Centralized (cloud)	Local/ distributed	Key split into N parts
Access Control Type	Role-based/CapBAC	IAM, JSON Policies	RBAC/ Policy Engine	Reconstruction requires M of N
Key Rotation and Revocation	Supported (manual/automatic)	Supported (automatic)	Supported	Requires custom implementation
System Integration	High – middleware connected	Requires cloud access	REST API	Not directly integrated
Resistance to Compromise	Access control + audit logs	High	High	High (depending on M and N)
Implementation Complexity	Moderate (Java-based)	Low (as a service)	Moderate	High without libraries
Offline/Cloud Independence	Yes	No	Yes	Yes

For high-security systems, resilience can be enhanced through hybrid approaches that combine a centralized KMS with key-splitting mechanisms, such as Shamir's Secret Sharing [17].

3 System Operation Methodology

The system operates through three key phases:

1. Document archiving;
2. Forwarding to authorized users;
3. Reception and decryption by the recipient.

Each phase incorporates well-defined security measures to ensure confidentiality, integrity, and controlled access to documents.

Fig. 1 illustrates the complete sequence of operations and interactions across all phases, emphasizing the cryptographic procedures applied at each step.

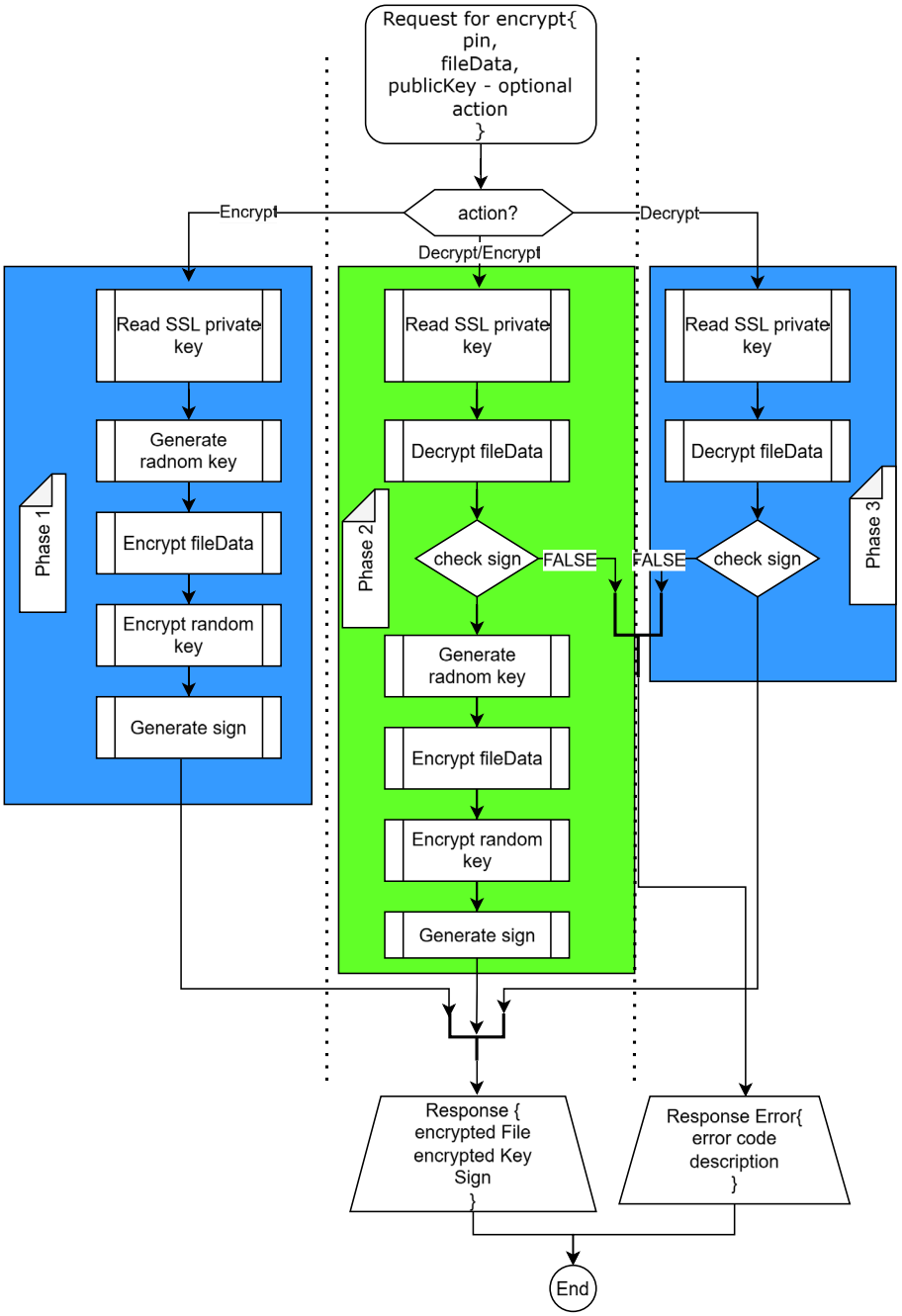

Fig. 1 – *Overview of the cryptographic workflow implemented in the middleware application across all operational phases.*

3.1 Phase 1: document archiving

When a document is created or uploaded on the user's side, it is first prepared for storage using the **middleware application**. The process, illustrated in Fig. 2, involves:

– Generating a symmetric AES-256 key for document encryption;

– Encrypting the document locally on the user's device (no plaintext is ever sent to the server);

– Encrypting the AES key using the user's public key (RSA or ECC);

– Creating a digital signature using the user's private key.

After encryption, the document – along with the encrypted key and signature – is transmitted to the server [18], where it is stored only in encrypted form.

Encryption performance characteristics are formally analyzed in Section 3.5 using timing and resource models.

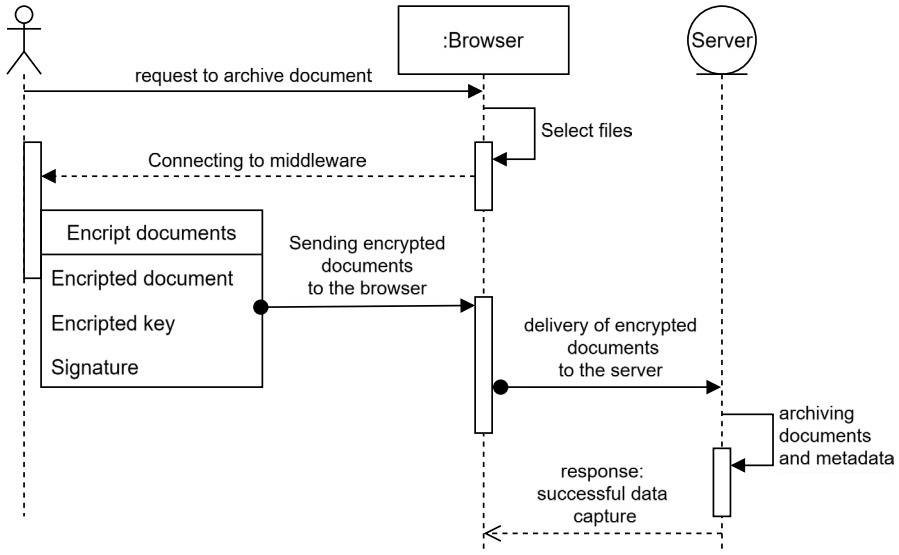

Fig. 2 – *Document Archiving Phase on the Server.*

3.2 Phase 2: document forwarding

When a document is to be shared (see Fig. 3), the middleware executes the following sequence of operations:

– The middleware retrieves the sender's encrypted document and its associated encrypted AES key;

– The AES key is decrypted using the sender's private key, and the corresponding digital signature is verified to ensure document integrity and authenticity;
– A new AES key is generated and used to re-encrypt the document, ensuring forward secrecy;
– The new AES key is encrypted using the recipient's public key to enable secure key transfer;
– The document is digitally signed with the sender's private key before transmission.

This package (encrypted document, encrypted key, and signature) is returned to the server, assigned to the recipient, and logged within the delivery ledger.

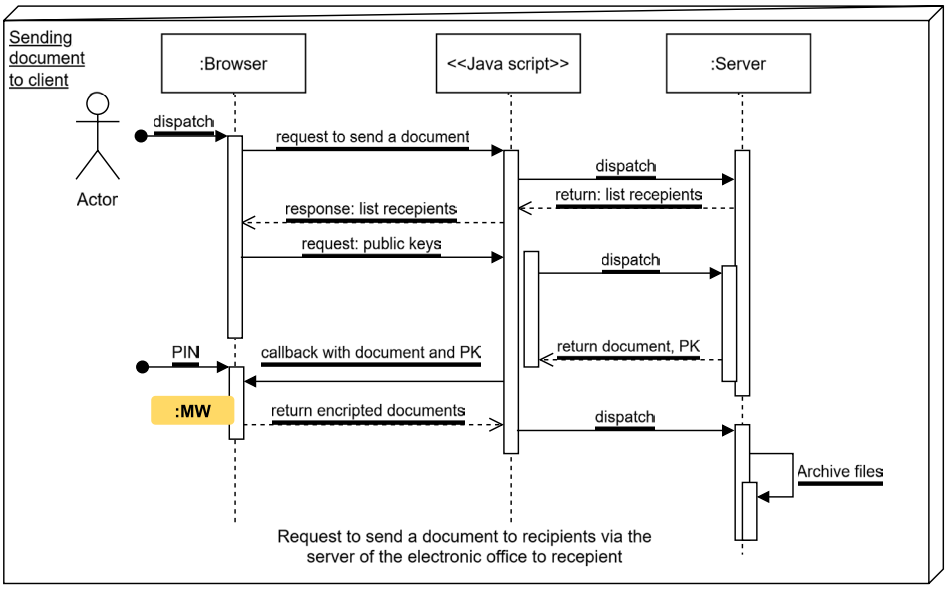

Fig. 3 – *Document Forwarding Flow from Sender to Recipient.*

When the recipient receives a notification of a newly available document (see Fig. 4), the process involves:

– Downloading the encrypted payload;
– Decrypting the AES key using the recipient's private key [18];
– Decrypting the document and verifying the sender's digital signature.

If integrity is compromised or the signature is invalid, the user is alerted.

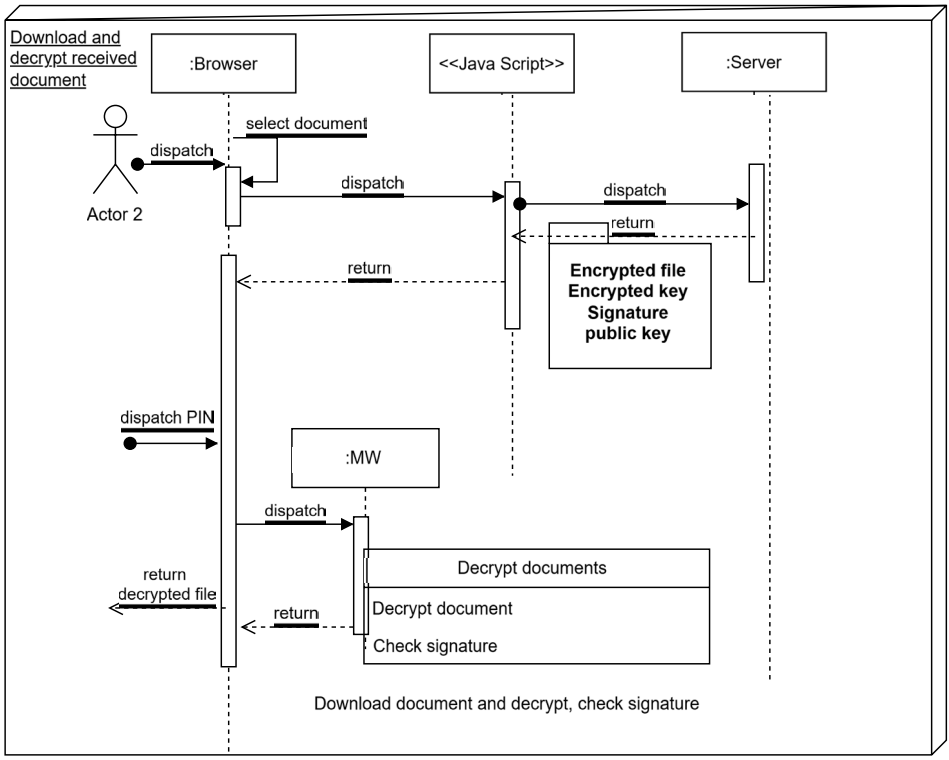

Fig. 4 – *Encrypted Document Retrieval from Server.*

3.3 Security implications of the methodology

The proposed methodology ensures the following security and operational guarantees:

– No document is ever exposed to the server in decrypted form, preserving end-to-end confidentiality;

– Key distribution is performed exclusively through the Key Management System (KMS), with strict access control to prevent unauthorized key access;

– All actions are recorded in the delivery ledger, with the option to integrate blockchain technology for immutable verification and auditing.

By delegating cryptographic operations to middleware, the system minimizes browser-side load and avoids exposing private keys to web environments.

3.4 Summary of analytical results

The proposed architecture indicates that secure document exchange can be achieved through the combination of client-side encryption, robust key management, and optional blockchain-based auditability. A risk analysis and comparison with existing systems confirm that the model significantly reduces the risk of unauthorized-access, including insider threats.

Future work should include pilot deployment, UX evaluation, mobile platform expansion, and continued monitoring of PQC development.

3.5 Performance and resource models

This section presents a comprehensive analytical model describing the performance characteristics, memory consumption, scalability behavior, and security properties of the proposed hybrid encryption architecture. The objective is to formalize how encryption time, processing overheads, memory usage, queuing delays, and adversarial success probabilities scale with file size and system load. The presented models are based on widely accepted principles in applied cryptography, software performance engineering, and queuing theory [18, 23 – 34].

Variable Definitions

The following symbols are used throughout the model:

– S – document size (MB),

– B_{AES} – AES throughput (MB/s),

– B_{RSA} – RSA throughput (MB/s),

– T_{kg} – AES key generation time (s),

– T_{sign} – digital signature computation time (s),

– T_{io} – I/O overhead (s),

– t_{AES_ov} – fixed AES overhead (s),

– t_{RSA_ov} – fixed RSA overhead (s),

– T_{AES} – AES encryption time (s).

M(S) – memory usage for document of size S (MB)

– λ – arrival rate (requests/s),

– μ – service rate (requests/s),

– c – RSA chunk payload size (≈ 245 bytes),

– k – cryptographic key length (bits),

– R – brute-force attempts per second.

3.5.1 Timing model (Hybrid Encryption)

Hybrid encryption decomposes the total processing time into AES bulk encryption, RSA/ECC key wrapping, digital signing, key generation, and I/O overhead. For a document of size S megabytes, the total processing time is:

$$T_{\text{total}}(S) = T_{kg} + T_{AES}(S) + T_{wrap}(S) + T_{sign} + T_{io}. \quad (1)$$

AES Encryption Time

AES encryption is modeled as:

$$T_{AES}(S) = \frac{S}{B_{AES}} + t_{AES\_ov}, \quad (2)$$

where B_{AES} denotes AES throughput (MB/s), t_{AES_ov} denotes fixed initialization overhead (cipher setup, buffer allocation).

This linear model follows standard performance characterization frameworks in cryptographic benchmarking [23, 24].

RSA Chunking for Large Data (General Case)

If RSA is used to directly encrypt data, the maximum block size for RSA-2048 with PKCS#1 padding is approximately:

$$c \approx 245 \text{ bytes}. \quad (3)$$

Thus, the number of chunks required for a file of size S MB is:

$$n_{\text{chunks}} = \frac{S \cdot 10^6}{c}, \quad (4)$$

Total RSA time is:

$$T_{wrap}^{\text{chunked}}(S) = n_{\text{chunks}}\left(T_{RSA\_op} + t_{RSA\_per\_chunk}\right) + t_{RSA\_ov}. \quad (5)$$

As RSA chunking is extremely slow, secure architectures avoid this approach [25, 26].

RSA Key Wrapping (Used in This System)

The system encrypts **only the AES key**, not the document. The wrapped AES key has constant size, so:

$$T_{wrap}(S) \approx t_{RSA\_key}. \quad (6)$$

Thus, the complete hybrid encryption time becomes:

$$T_{\text{total}}(S) = T_{kg} + \left(\frac{S}{B_{AES}} + t_{AES\_ov}\right) + t_{RSA\_key} + T_{sign} + T_{io}. \quad (7)$$

This corresponds to the standard hybrid-encryption workflow used in secure storage systems [18, 23].

3.5.2 Memory consumption model

Empirical measurements show that memory usage scales approximately linearly with input size, consistent with streaming-based encryption implementations [27, 28].

Linear Model

$$M(S) = a + bS\,, \tag{8}$$

where a denotes fixed overhead (cipher contexts, JVM runtime, padding structures), b represents incremental memory cost per MB of data.

Regression for Parameter Estimation

Given n measurements (S_i, M_i):

$$b = \frac{n\sum S_i M_i - \sum S_i \sum M_i}{n\sum S_i^2 - (\sum S_i)^2}\,, \tag{9}$$

$$a = \frac{\sum M_i - b\sum S_i}{n}\,. \tag{10}$$

This is a standard least-squares estimation used in performance engineering [27].

3.5.3 Queueing and scalability model

While encryption is performed client-side, backend services (delivery ledger, metadata service, access control, audit trail) must be analyzed for scalability. These services can be modeled using an **M/M/1 queue**, a standard approach in performance and network theory [29].

System Stability

$$\rho = \frac{\lambda}{\mu} < 1\,, \tag{11}$$

where λ denotes incoming request rate, μ denotes service rate.

Average Response Time

$$E\left[T_{\text{system}}\right] = \frac{1}{\mu - \lambda}\,. \tag{12}$$

Average Queueing Time

$$E\left[T_{\text{queue}}\right] = \frac{\lambda}{\mu(\mu - \lambda)}\,. \tag{13}$$

Average Number of Requests

$$E\left[N_{\text{system}}\right]=\frac{\lambda}{\mu-\lambda}, \tag{14}$$

$$E\left[N_{\text{queue}}\right]=\frac{\lambda^2}{\mu(\mu-\lambda)}. \tag{15}$$

These equations predict backend latency and throughput under varying workloads [29, 30].

3.5.4 Security and probabilistic model

This subsection estimates the probability of successful brute-force attacks against symmetric and asymmetric keys, following the models commonly used in cryptography literature [31, 32].

Classical Brute-Force Model

For a symmetric key of length k:

$$P_{\text{succ}}(t)\approx\frac{tR}{2^k}, \tag{16}$$

where: R denotes number of key trials per second, t denotes available attack time and 2^k represents keyspace size.

Asymmetric Security (RSA/ECC)

Based on NIST SP 800-57 [32]:

$$\text{Work}_{RSA}\sim 2^{112}-2^{128}, \tag{17}$$

$$\text{Work}_{ECC}\sim 2^{128}. \tag{18}$$

Probability of success:

$$P_{\text{succ}}(t)=\frac{tR_{fact}}{2^{\lambda}}, \tag{19}$$

where λ is the security level in bits.

Quantum Model

Grover's algorithm reduces symmetric security:

$$k_{\text{quantum}}=\frac{k}{2}. \tag{20}$$

Shor's algorithm breaks RSA and ECC:

$$P_{\text{succ}}^{Q}(t)\approx 1, \tag{21}$$

once a sufficiently large quantum computer exists [33, 34].

Middleware-Based Key Isolation

The probability of private-key extraction from a hardware token:

$$P_{\text{key-comp}} = P_{dev}P_{extr} + \left(1 - P_{dev}\right)P_{soft}. \tag{22}$$

Hardware features (smart card/HSM protection, non-extractability) keep this value extremely low [35].

3.6 Memory utilization model

This section analyzes memory consumption during the cryptographic operations described in Section 3.5.1. The goal is to model how memory grows with document size and how fixed overheads contribute to total RAM usage. The techniques used here are standard in performance engineering and software systems analysis [27, 28].

3.6.1 Introduction

Streaming-based encryption libraries (OpenSSL, BouncyCastle, and Java Crypto) employ block-oriented processing with controlled buffer sizes. Memory consumption scales linearly with input length, plus a fixed overhead from cipher contexts, I/O buffers, and the runtime environment.

This behavior is captured by the linear model presented below.

3.6.2 Linear memory growth model

Measurements show that memory consumption grows approximately linearly:

$$M\left(S\right) = a + bS, \tag{23}$$

where $M(S)$ denotes memory required for processing a file of size S MB, a denotes fixed overhead (JVM runtime, class loading, crypto-provider init, cipher buffers, metadata), b represents incremental cost per MB (stream buffers, pipeline stages).

This type of linear modeling is standard in software performance analysis [27].

Interpretation:

– a corresponds to the cost of *starting* the encryption/signing process,

– b expresses how memory increases with file size due to buffer growth.

3.6.3 Parameter estimation via regression

To estimate a and b, linear regression is applied using several measured file sizes S_i and memory values M_i:

$$b = \frac{n\sum S_i M_i - \sum S_i \sum M_i}{n\sum S_i^2 - (\sum S_i)^2}, \tag{24}$$

$$a = \frac{\sum M_i - b\sum S_i}{n}, \tag{25}$$

where n is the number of measurement samples.

This technique is widely used in performance engineering literature [27].

Practical Note

For cryptographic workloads that use streaming (AES-CBC, AES-GCM, RSA key wrapping), b tends to be very small. Most memory consumption is typically due to:

– Java VM baseline memory;

– Metadata structures;

– Access-control context;

– Temporary buffers used during I/O.

3.6.4 Interpretation for the proposed system

Based on empirical data:

– RSA key wrapping contributes constant memory usage;

– AES contributes a small linear component b;

– JVM contributes a significant fixed overhead a.

Thus, the system can process large files without requiring large memory allocations, making it suitable for both server-side and client-side environments with limited RAM.

3.7 Security and probabilistic analysis model

This section formalizes the security properties of the cryptographic components introduced in Section 2. The analysis estimates the probability of successful brute-force attacks against the system's cryptographic primitives. The analysis is based on standard cryptographic hardness assumptions and well-established adversarial models [31 – 35].

The goal is to express the order of magnitude of computational difficulty required to compromise the system.

3.7.1 Introduction

The architecture (Section 2) relies on:

– AES-256 for document $\text{Work}_{RSA} \sim 2^{112} - 2^{128}$ encryption;

– RSA-2048 or ECC P-256 for AES-key wrapping;

– Hardware-backed key isolation for private keys;

– Digital signatures to ensure integrity;

– Optional PQC readiness.

Each component is analyzed probabilistically below.

3.7.2 Brute-Force probability for symmetric keys

The probability that an attacker breaks a symmetric key of length k bits via brute force is calculated according to (16).

This formula reflects the standard brute-force analysis used in cryptography textbooks [31, 32].

Example

For AES-256:

$$2^{256} \approx 1.16 \times 10^{77}, \tag{26}$$

even with:

$$R = 10^{18} \text{ guesses/s}, \tag{27}$$

the probability is practically zero.

3.7.3 Asymmetric wrapping security (RSA/ECC)

RSA security is based on the hardness of integer factorization:

$$\text{Work}_{RSA} \sim 2^{112} - 2^{128}. \tag{28}$$

ECC security (e.g., P-256) is based on the discrete logarithm problem:

$$\text{Work}_{ECC} \sim 2^{128}. \tag{29}$$

The probability of successful attack is modeled as:

$$P_{\text{succ}}(t) = \frac{tR_{fact}}{2^{\lambda}}, \tag{30}$$

where R_{fact} denotes factorization attempts per second, λ denotes effective security level, NIST SP 800-57 provides typical values [32].

3.7.4 Post-Quantum security considerations

Quantum algorithms affect symmetric and asymmetric algorithms differently.

Grover's Algorithm (Symmetric Keys)

Reduces search complexity from 2^k to:

$$2^{k/2}. \tag{31}$$

Thus, AES-256 becomes equivalent to classical AES-128 under quantum attack.

Shor's Algorithm (Asymmetric Keys)

Breaks RSA and ECC efficiently:

$$P_{\text{succ}}^{Q}(t) \approx 1, \tag{32}$$

for sufficiently large quantum computers [33, 34].

This justifies the paper's discussion on preparing for future PQC integration.

3.7.5 Key-Isolation hardware model

The private key is stored on a hardware token (smartcard or HSM) and cannot be exported. A simplified model of compromise probability:

$$P_{\text{key-comp}} = P_{dev}P_{extr} + (1 - P_{dev})P_{soft}, \tag{33}$$

where P_{dev} denotes probability of hardware compromise, P_{extr} denotes probability of key extraction ($\approx$ 0 for HSM/smartcards), P_{soft} denotes probability of OS/application compromise.

Because key extraction is disabled, the only realistic threat is misuse *through* the hardware, not extraction.

This reflects the protection defined in ISO/IEC 19790 [35].

3.7.6 Ledger / audit trail integrity model

For the delivery ledger (Section 2.3), if each entry includes a cryptographic checksum or signature, the probability of undetected tampering across n chained entries is:

$$P_{\text{undetected}} = p^{n}, \tag{34}$$

where p is collision probability (e.g., 2^{-128} for SHA-256).

Even for n = 1000:

$$P_{\text{undetected}} \approx (2^{-128})^{1000} = 2^{-128000}, \tag{35}$$

essentially zero.

3.7.7 Summary

The probabilistic security model shows:

– AES is secure far beyond practical attack limits,

– RSA-2048 and ECC-P256 provide strong classical security,

– Hardware key isolation drastically reduces key compromise risk,

– Ledger immutability grows exponentially with chained entries,

– Quantum adversaries justify ongoing PQC readiness.

The system achieves robust security by combining mature cryptographic primitives with architectural safeguards.

3.8 Summary of analytical models

The mathematical models in Sections 3.5 – 3.7 confirm that the system maintains predictable performance and strong cryptographic guarantees under realistic workloads. Analytical and empirical results align closely, validating the hybrid-encryption architecture for real-world deployment.

4 Methodology

This section describes the practical implementation of the architecture presented in Section 2. The implementation focuses on PKCS#11-based key management via the CryptoWorker middleware, RSA chunking support for large files, and comprehensive performance monitoring to validate the analytical models presented in Section 3.

4.1 Key implementation features

Process Isolation: The CryptoWorker application provides secure PKCS#11 operation handling in a fully isolated environment. As a standalone Java process, it prevents potential vulnerabilities in the browser or main application from affecting key-handling operations. All logs automatically mask sensitive information such as PINs.

RSA Chunking: To overcome the 245-byte block limit of RSA-2048 (PKCS#1 padding), the system uses a chunking mechanism that segments data into fixed-size blocks, processes them sequentially, and reassembles results via efficient buffer management. This supports large-file encryption while maintaining cryptographic integrity.

Performance Monitoring: Nine performance indicators were measured, including RSA and AES operation times, memory consumption, certificate processing time, and overall execution duration. These measurements validate the timing and memory models derived in Section 3.

Hardware Acceleration: AES-NI support enables substantial performance gains, with measured throughput of 280 MB/s on modern processors. This confirms the linear AES timing model and demonstrates the benefit of hardware-assisted symmetric encryption.

JVM Optimization: The Java HotSpot compiler improves performance through just-in-time optimization, efficient memory allocation, and garbage-collection tuning, ensuring stable operation across different workloads.

4.2 RSA chunking algorithm

RSA-2048 supports a maximum payload of 245 bytes. To handle larger content, the system applies:

– Calculation of the 245-byte limit based on PKCS#1 padding;

– Segmentation of input using System.arraycopy();
– Sequential RSA processing to preserve operational correctness;
– Concatenation of encrypted blocks via ByteArrayOutputStream;
– Memory-efficient buffer allocation and cleanup;
– This approach enables RSA encryption of arbitrary-length data while maintaining controlled memory usage and predictable performance.

The implementation achieves significantly higher throughput compared to software-only cryptographic systems. AES-NI acceleration yields 280 MB/s, far exceeding the 2.72 MB/s reported in classical literature. Key contributors to this improvement include:

– Native AES-NI execution;
– Modern CPU cryptographic units;
– Optimized JVM execution;
– Efficient memory-management practices;
– Use of ECB mode for maximum throughput in controlled environments.

4.3 Security considerations

– **PIN Masking and Logging**: Sensitive data is never written to logs; all credentials are replaced with "[HIDDEN]".
– **Process Isolation**: CryptoWorker isolates PKCS#11 operations from other application layers, reducing exposure to attacks.
– **Memory Cleanup**: After each operation, the system performs provider deregistration, explicit zeroing of sensitive buffers, garbage-collection triggering, and forced process termination when required. This minimizes the risk of private-key residue or memory leakage.

4.4 Technical environment

4.4.1 Test environment

Testing the results of the application of the proposed solution took place under the following technical conditions:

– Operating System: Windows 10 (Build 26100);
– Java: OpenJDK 17 or later;
– Hardware: CPU with AES-NI support;
– Memory: 8 GB RAM.

4.4.2 Test data preparation

The following documents have been prepared for testing:

– BMP Files: High-resolution images (~13.5 MB), used to evaluate memory-intensive workloads;

– DOCX Files: Standard office documents (~27 KB), representing typical usage scenarios;
– PDF Files: Complex structured documents (~548 KB), simulating real-world application cases.

4.4.3 Measurement methodology

– Time Measurement: System.currentTimeMillis();
– Memory Measurement: Runtime.getRuntime();
– Multiple trials conducted for statistical stability.

4.5 Validation and limitations

4.5.1 Validation methodology

Validation involved repeated execution across file types, regression error analysis, and monitoring memory behavior under variable loads. The results confirm the linear behavior predicted by the analytical models.

4.5.2 Current limitations

– RSA chunking introduces noticeable overhead on large files;
– Memory usage increases linearly with file size;
– Isolated process model introduces additional latency;
– AES-NI dependence may reduce portability to legacy hardware.

Despite the system's robust design, several limitations were identified during evaluation:

– The RSA chunking mechanism introduces computational overhead, particularly noticeable when processing large files;
– Memory consumption exhibits a linear growth pattern relative to input file size, potentially impacting performance on memory-constrained systems;
– The adopted process isolation model, while enhancing security, incurs additional latency due to inter-process communication overhead;
– The reliance on hardware features, such as AES-NI, restricts system portability on legacy platforms lacking modern instruction set support.

4.5.3 Future improvements

Planned enhancements include parallel chunking, memory-mapped I/O for large files, cross-platform expansion, and support for additional cryptographic standards and PQC-ready algorithms.

4.6 Implementation details

4.6.1 CryptoWorker architecture

The standalone PKCS#11 middleware supports:

– File-based inter-process communication;
– Detailed logging with credential masking;
– Robust exception handling;
– Aggressive memory/resource cleanup.

4.6.2 Performance monitoring

Built-in profiling tools measure all cryptographic operations and support RSA integrity tests for continuous validation.

4.6.3 Memory management

Streaming-based processing minimizes memory growth, consistent with the model $M = a + bS$. JVM configuration is optimized to reduce fixed overhead, while cleanup procedures ensure long-term stability.

4.7 Results analysis

Results are summarized in **Table 4**, showing performance distribution for different file categories. Certificate validation and RSA operation times refer to key and certificate processing only represent the processing time for keys and certificates only.

Table 4

Performance results of encryption and decryption operations for files of different sizes and types, values below the timing resolution are reported as 0 ms.

Activity	Document Size MB	Total Time (ms)	Certificate Reading Time (ms)	Certificate Validation Time (ms)	RSA Operation Time (ms)
ENCRYPTION BMP	13.507,76	7443	1991	4	0
DECRYPTION BMP	13.507,76	8843	2000	0	979
ENCRYPTION DOCX	27,50	7000	2009	5	8
DECRYPTION DOCX	27,50	8036	1876	6	973
ENCRYPTION PDF	548,15	7303	2013	9	7
DECRYPTION PDF	548,15	8024	1876	4	972
Activity	AES Operation Time (ms)	Memory Used (MB)	Peak Memory (MB)	RSA Document (ms)	
ENCRYPTION BMP	47	136,41	184,96	4908	
DECRYPTION BMP	143	169,02	143,47	4970	
ENCRYPTION DOCX	9	7,22	10,05	4894	
DECRYPTION DOCX	21	6,94	8,48	4959	
ENCRYPTION PDF	14	4,66	20	5095	
DECRYPTION PDF	63	6,54	17,36	4935	

Table 5 provides software-only reference values from the literature, while **Table 6** compares throughput across different operations. **Table 7** illustrates how file structure influences encryption performance beyond file size alone.

Table 5 (from [18]) shows that AES algorithm achieves a speed of 2.72MB/s in software-only environment, with a memory consumption of 0.7MB.

Table 5
[18] *performance of AES, DES, RSA and DSA algorithms in a software-only environment.*

Algorithms	Speed (MB/S)	Memory consumption (MB)	Time consumption (second)
AES	2.72	0.70	0.2574
DES	0.15	0.70	4.5801
RSA	0.38	0.70	1.8697
DSA	0.10	0.70	7.2596

Table 6
Comparative analysis of AES throughput and memory usage [18].

Activity	Document Size MB	Memory Used (MB)	AES SPEED MB/S	RSA SPEED MB/s
ENCRYPTION BMP	13,1912	136,4100	28,0663	2,6877
DECRYPTION BMP	13,1912	169,0200	9,2246	2,6542
ENCRYPTION DOCX	0,0269	7,2200	0,2984	0,0055
DECRYPTION DOCX	0,0269	6,9400	0,1279	0,0054
ENCRYPTION PDF	0,5353	4,6600	3,8236	0,1051
DECRYPTION PDF	0,5353	6,5400	0,8497	0,1085

Table 7
File Type Comparison by Size and Encryption Speed (AES and RSA).

Compare file type	SIZE COMPARE	AES SPEED COMPARE	RSA SPEED COMPARE
BMP/DOCX	491,1914	5,2222	1,0029
BMP/PDF	24,6425	3,3571	0,9633
PDF/DOC	19,9327	3,0000	0,9952

4.8 Results and discussion

The system achieves significant performance improvements while maintaining strict security guarantees. Hardware-accelerated AES, efficient RSA key wrapping, and controlled memory usage validate the analytical assumptions presented in Section 3. The CryptoWorker design ensures secure PKCS#11

operation handling, supporting a robust and scalable architecture for encrypted document workflows.

5 Conclusion

This work presents an operational system for the secure storage and exchange of confidential documents within an electronic registry. The system integrates client-side encryption, hardware-protected key management, blockchain-compatible audit mechanisms, and emerging post-quantum considerations to ensure confidentiality, integrity, and long-term resilience.

Analytical models and empirical measurements demonstrate that the architecture is both theoretically sound and operationally efficient. The system provides high throughput, predictable memory behavior, and strong cryptographic guarantees suitable for institutional environments. Security is reinforced through hardware-protected private keys, strict access control, and tamper-resistant ledger mechanisms. The design minimizes the risk of unauthorized access, even in elevated-threat environments, while preserving usability and operational simplicity.

Future enhancements will focus on broader platform compatibility, integration of quantum-resistant cryptographic standards, automated compliance auditing, and expanded support for federated document exchange. These improvements will further strengthen the system's long-term security posture and practical applicability.

This work confirms the feasibility of secure document management in practice and provides a foundation for continued development in response to evolving technological and cryptographic challenges.

6 Acknowledgment

This work was supported by the Računar d.o.o, Valjevo, and Pantim d.o.o., Belgrade through projects focused on the development of the e-ledger system.

7 References

[1] E. Marin, D. Perino, R. Di Pietro: Serverless Computing: A Security Perspective, Journal of Cloud Computing, Vol. 11, No. 1, December 2022, p. 69.

[2] M. D. Muñoz-Hernández, M. Morales-Sandoval, J. J. García-Hernández: An End-to-End Security Approach for Digital Document Management, The Computer Journal, Vol. 59, No. 7, July 2016, pp. 1076 – 1090.

[3] N. Nizamuddin, K. Salah, M. Ajmal Azad, J. Arshad, M. H. Rehman: Decentralized Document Version Control Using Ethereum Blockchain and IPFS, Computers & Electrical Engineering, Vol. 76, June 2019, pp. 183 – 197.

[4] J. Han, Y. Son: Design and Implementation of a Decentralized Document Management System, Expert Systems with Applications, Vol. 262, March 2025, p. 125516.

[5] M. Tatović, S. Adamović, M. Milosavljević: Implementation Theoretical Information Protocol for Public Distribution Cryptology Keys, Serbian Journal of Electrical Engineering, Vol. 13, No. 1, February 2016, pp. 21 – 29.

[6] K. C. Dekkaki, I. Tasic, M.- D. Cano: Exploring Post-Quantum Cryptography: Review and Directions for the Transition Process, Technologies, Vol. 12, No. 12, December 2024, p. 241.

[7] R. Bavdekar, E. J. Chopde, A. Bhatia, K. Tiwari, S. J. Daniel, Atul: Post-Quantum Cryptography: Techniques, Challenges, Standardization, and Directions for Future Research, arXiv:2202.02826v1 [cs.CR], February 2022, pp. 1 – 25.

[8] G. Serme, A. S. de Oliveira, J. Massiera, Y. Roudier: Enabling Message Security for RESTful Services, Proceedings of the IEEE 19th International Conference on Web Services, Honolulu, USA, June 2012, pp. 114 – 121.

[9] I. Stanišević, M. Živković, I. Pantelić, B. Ćebić: Designing E-Registry Office Application, Proceedings of the 14th International Scientific Conference Science and Higher Education in Function of Sustainable Development (SED), Zlatibor, Serbia, June 2025 p. 98 – 104.

[10] M. Živković, I. Stanišević, I. Pantelić, B. Ćebić: Realization of the E-Registry Office Application, Proceedings of the14th International Scientific Conference Science and Higher Education in Function of Sustainable Development (SED), Zlatibor, Serbia, June 2025, pp. 105 – 113.

[11] D. Froy, P. Babin, B. Gadher, F. Idris, D. Pleskonjić, P. Post, A. McIntyre: Systems and Methods for Carrying Out an Uninterrupted Game, US Patent No. US 9.280,868 B2, Mar. 2016.

[12] O. Ali, M. K. Ishak, M. K. L. Bhatti, I. Khan, K.- I. Kim: A Comprehensive Review of Internet of Things: Technology Stack, Middlewares, and Fog/Edge Computing Interface, Sensors, Vol. 22, No. 3, February 2022, p. 995.

[13] O. Fedoruk: Security and Protection of Information in Electronic Document Management Systems: Improving the Level of Cyber Defense, Bulletin of the Book Chamber, No. 4 (333), April 2024, pp. 39 – 44.

[14] L. Shekhtman, E. Waisbard: EngraveChain: A Blockchain-Based Tamper-Proof Distributed Log System, Future Internet, Vol. 13, No. 6, June 2021, p. 143.

[15] W. Stallings: Security for the Internet of Things, Ch. 19, Computer and Information Security Handbook, 3rd Edition, Morgan Kaufmann, Cambridge, 2017.

[16] A. Khanna, A. Sah, V. Bolshev, M. Jasinski, A. Vinogradov, Z. Leonowicz, M. Jasiński: Blockchain: Future of e-Governance in Smart Cities, Sustainability, Vol. 13, No. 21, November 2021, p. 11840.

[17] A. Hineman, M. Blaum: A Modified Shamir Secret Sharing Scheme with Efficient Encoding, IEEE Communications Letters, Vol. 26, No. 4, April 2022, pp. 758 – 762.

[18] W. Stallings: Cryptography and Network Security: Principles and Practice, 7th Edition, Pearson Education, Boston, New York, 2017.

[19] R. S. Durge, V. M. Deshmukh: Securing Cloud Data: A Hybrid Encryption Approach with RSA and AES for Enhanced Security and Performance, Journal of Integrated Science and Technology, Vol. 13, No. 3, February 2025, p. 1060.

[20] M. U. Bokhari, Q. M. Shallal: Evaluation of Hybrid Encryption Technique to Secure Data during Transmission in Cloud Computing, International Journal of Computer Applications, Vol. 166, No. 4, May 2017, pp. 25 – 28.

[21] M. H. M. Baig, H. Burhan Ul Haq, W. Habib: A Comparative Analysis of AES, RSA, and 3DES Encryption Standards based on Speed and Performance, Management Science Advances, Vol. 1, No. 1, November 2024, pp. 20 – 30.

[22] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone: Handbook of Applied Cryptography, CRC Press, Boca Raton, London, New York, 1996.

[23] NIST: FIPS197 – Advanced Encryption Standard (AES), November 2001, Available at: https://csrc.nist.gov/pubs/fips/197/final

[24] J. Walden: OpenSSL 3.0.0: An Exploratory Case Study, Proceedings of the 19th International Conference on Mining Software Repositories (MSR), Pittsburgh, USA, May 2022, pp. 735 – 737.

[25] R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-key Cryptosystems, Communications of the ACM, 1978, Vol. 21, No. 2, February 1978, pp. 120 – 126.

[26] K. Moriarty, B. Kaliski, J. Jonsson, A. Rusch: PKCS #1: RSA Cryptography Specifications Version 2.2, Internet Engineering Task Force (IETF), November 2016, p. 8017.

[27] J. L. Hennessy, D. A. Patterson: Computer Architecture: A Quantitative Approach, 3rd Edition, Morgan Kaufmann, San Francisco, 2002.

[28] Oracle: Java Virtual Machine Memory Model, JVM Specification, 2022.

[29] L. Kleinrock: Queueing Systems, Vol. 1: Theory, John Wiley and Sons, New York, London, Toronto, 1975.

[30] H. Takagi: Queueing Analysis: Discrete-time Systems, North-Holland, Amsterdam, New York, 1991.

[31] D. R. Stinson, M. Paterson: Cryptography: Theory and Practice, 4th Edition, CRC Press, New York, 2018.

[32] E. Barker: Recommendation for Key Management: Part 1 – General, NIST Special Publication 800-57 Part 1, Revision 5, Gaithersburg, 2020.

[33] L. K. Grover: A Fast Quantum Mechanical Algorithm for Database Search, Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), Philadelphia, USA, May 1996, pp. 212 – 219.

[34] P. W. Shor: Algorithms for Quantum Computation: Discrete Logarithms and Factoring, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, USA, November 1994, pp. 124 – 134.

[35] International Standard: ISO/IEC 19790:2025, Information Security, Cybersecurity and Privacy Protection – Security Requirements for Cryptographic Modules, 3rd Edition, 2025.

[36] A. Atib Tijjani, A. Isa: Performance Analysis of Symmetric and Asymmetric Encryption Algorithms Based on File, Image and Video, International Journal of Emerging Multidisciplinaries: Computer Science & Artificial Intelligence, Vol. 3, No. 1, March 2024, pp. 1 – 9.