People's Democratic Republic of Algeria
Ministry of Higher Education for Scientific Research
University 8 May 45 –Guelma-
Faculty of Mathematics, Computer Science and Sciences of Matter
Department of Computer Science

Master Thesis

Specialty: Computer science
Option: Computer system

Theme

Toward a Startup for Electronic Signature and Certificates Services under the Algerian Regulation

Presented by: Alaeddine BOUSSAID

Jury Members

Chairman	Dr. Samir HALLACI
Supervisor	Mr. Nabil BERREHOUMA
Examiner	Dr. Mohamed Amine FERRAG

June 2025

Acknowledgements & Dedication

First and foremost, I thank Almighty God, whose mercy and guidance have enabled me to reach this point, overcome every obstacle, and successfully complete this final year project.

I would like to express my deep gratitude to my supervisor, **Mr. Nabil BERREHOUMA**, for his valuable advice, insightful guidance, and the positive energy he has provided me throughout this journey. His professionalism, experience, and wisdom have been a great support in completing this work.

My sincere thanks also go to all the faculty members of the Computer Science Department, for the quality of their teaching and support throughout my academic path. I extend special appreciation to **Mr. Zineddine KOUAHLA**, Head of Department, for his dedication and availability.

I am also grateful to the **Professional Pole of the university**, led by **Mr. Abdelaziz BENKIRAT**, for accepting my final year project within the framework of a startup initiative. Their support, wise advice, and the enriching training programs they offer have significantly contributed to the maturity of my project. Their continued commitment to innovation and entrepreneurship in the academic environment has been a true source of inspiration throughout this experience.

I dedicate this work with all my love and gratitude to my **father, mother, and brothers**—being the first graduate in our family is a great honor for me. Your presence, patience, and sacrifices are the foundation of this success.

To all my friends and colleagues, and especially to **Zakaria** and **Ouail**, who have become true brothers to me. Over the past four years, we shared everything together at the university residence—our second home. May God preserve this sincere friendship for life.

I would also like to dedicate this work to my students from the Department of Arabic Language and Literature, whom I had the honor of teaching this past semester as a temporary lecturer in the computer science module. I sincerely wish them success in their studies and in their own graduation projects next year.

Finally, to anyone I may have wronged or hurt unintentionally, I ask for your forgiveness. And to all those who love and support me, I offer my heartfelt thanks.

Note on the Use of AI Tools

In the interest of academic transparency and integrity, we hereby acknowledge the use of artificial intelligence tools during the preparation of this thesis.
These tools were employed in a limited and responsible manner, specifically for:

- improving the linguistic formulation of certain paragraphs;
- assisting with the translation of technical content (mainly from French to English);
- providing suggestions regarding structural organization and formatting.

It is important to emphasize that all technical content, conceptual development, system architecture, implementation, and experimental work presented in this thesis were fully developed by the authors. Artificial intelligence was used solely as a language and drafting assistant and did not generate any substantive content or project results.
This mention reflects our commitment to academic honesty while leveraging modern tools for enhancing clarity and readability.

Abstract

As Algeria undergoes its digital transformation, the establishment of legally recognized and technically secure electronic signature services has become a national imperative. Despite the adoption of Law No. 15-04 and its executive decrees providing a legal framework for electronic signatures, the market still lacks a practical, localized, and standards-compliant solution tailored to the Algerian context.

This thesis presents the design, implementation, and evaluation of CertiSafe—a scalable and secure platform for electronic certification and digital signatures aligned with Algerian regulations. CertiSafe serves both as a functional prototype and as a strategic foundation for launching a trusted Certification Authority (CA) capable of serving government, business, and civil society. The platform integrates a React.js frontend, a Node.js backend, and a MySQL database, and relies on OpenSSL and the Node.js crypto module for cryptographic operations.

Core functionalities include secure user registration, X.509 certificate issuance, document signing, signature validation, and full certificate lifecycle management. The architecture is modular, allowing for future extensions such as mobile access, multi-signature workflows, secure offline signing, and integration with enterprise systems (ERP).

Beyond the technical stack, the thesis addresses regulatory compliance with Algerian laws and compares them to international standards . It also proposes a business-oriented vision, paving the way for a startup capable of bridging legal, technical, and market needs in the realm of digital trust services.

This work ultimately contributes to building a sovereign digital trust ecosystem in Algeria, combining legal compliance, robust cryptographic practices, and a user-centered software architecture.

Contents

List of Figures

Introduction

The digital transformation engaged by the Algerian government in the recent years has revolutionized how businesses, governments, and individuals interact, conduct transactions, and manage data. One significant facet of this transformation is the adoption of electronic signatures and certification services, which have become pivotal for ensuring secure, efficient, and legally binding digital interactions. Faced to this technological revolution, New kind of services in relation with digital data security becomes a reality.

Recent legislative advancements in Algeria have laid the groundwork for the proliferation of these secure digital services. Law No. 15-04, addressing electronic commerce and transactions, and Decree No. 16-142, which provides detailed provisions on the use and recognition of electronic signatures, form the cornerstone of this regulatory framework. These regulations not only validate the legal equivalence of electronic and handwritten signatures but also outline the technical and procedural requirements for their implementation.

This legislative progress reflects Algeria's commitment to fostering a secure digital economy and aligns with global trends. However, the challenge lies in bridging the gap between regulatory stipulations and practical implementation. The lack of readily available, localized solutions tailored to Algerian standards presents a unique market opportunity for innovative startups. By aligning services with these legal frameworks, this project aims to address a critical need for compliant, secure, and efficient electronic signature solutions.

The growing reliance on digital platforms in sectors such as finance, healthcare, education, and public administration has heightened the demand for trustworthy electronic signature services. Businesses and government entities require solutions that ensure data integrity, authentication, and non-repudiation while adhering to legal standards. This demand is particularly pronounced in Algeria, where digitalization initiatives are gaining momentum.

Despite this burgeoning demand, the Algerian market faces challenges such as limited awareness of electronic signature technologies, skepticism regarding their security, and a scarcity of tailored solutions that align with local regulatory requirements. This project identifies these gaps as opportunities for a startup to establish itself as a pioneer in the field. By offering user-friendly, secure, and legally compliant electronic signature services, the proposed startup aims to facilitate the digital transformation of various sectors while addressing existing challenges.

The primary objective of this master's project is to design, implement, and evaluate a comprehensive application for electronic signature and certification services. This application will serve as a cornerstone for the proposed startup, encompassing both frontend and backend functionalities to ensure a seamless user experience and robust security.

On the frontend, the application will provide an intuitive interface for users to manage their electronic signatures and certificates. Features will include signature creation, certificate management, and validation tools, all designed with usability and accessibility in mind. The backend, on the other hand, will leverage Public Key Infrastructure (PKI) to ensure secure certificate lifecycle management, including issuance, storage, and

revocation. Tools like OpenSSL will be utilized to implement cryptographic operations, ensuring data integrity and compliance with Algerian standards.

Adherence to Algerian regulatory standards is a cornerstone of this project. The application will incorporate mechanisms to ensure compliance with the technical and procedural requirements outlined in Law No. 15-04 and Decree No. 16-142. This includes implementing secure authentication protocols, maintaining audit trails, and ensuring data confidentiality and integrity. By aligning with these standards, the project not only guarantees legal validity but also builds trust among users and stakeholders.

The technical implementation of this project involves the integration of advanced cryptographic techniques to secure electronic signatures and certificates. PKI will serve as the backbone of the application, providing a hierarchical structure for certificate management. Key functionalities will include:

- Certificate Issuance and Management: The backend will generate and manage certificates, ensuring secure key storage and lifecycle management.
- Electronic Signature Creation and Validation: Users will be able to create electronic signatures that are legally binding and verifiable.
- Data Integrity and Non-Repudiation: The application will implement hashing and encryption techniques to ensure that signed documents remain tamper-proof and traceable.
- Compliance Mechanisms: Built-in features will ensure adherence to Algerian regulations, including timestamping, secure storage, and certificate revocation lists (CRLs).

To achieve the outlined objectives , the project will follow a structured work plan comprising the following phases:

Chapter 1 : Theoretical Background This phase involves a comprehensive review of existing electronic signature technologies, regulatory frameworks, and market trends. It will provide a foundation for identifying best practices and potential areas for innovation.

Chapter 2 :Technical Background This phase will explore the underlying technologies and tools required for the project, including PKI, cryptographic algorithms, and OpenSSL.

Chapter 3 System Design The design phase will focus on defining the architecture of the application, encompassing both frontend and backend components. User requirements, security protocols, and regulatory compliance will guide the design process.

Chapter4 : Experiments and Results The final phase involves implementing the application, conducting tests to evaluate its functionality and security, and analyzing the results. Feedback from potential users and stakeholders will be incorporated to refine the solution.

By addressing the intersection of regulatory compliance, market demand, and technical innovation, this master's project aims to establish a solid foundation for a startup specializing in electronic signature and certification services in Algeria. The proposed application will not only meet the current needs of various sectors but also pave the way for future advancements in secure digital transactions. Through meticulous design, implementation, and evaluation, this project seeks to contribute to Algeria's digital transformation and inspire confidence in electronic signature technologies.

Chapter 1

Theoretical Background

1.1 Introduction

The implementation of electronic certification in Algeria represents a crucial advancement in the digital transformation landscape. With the rapid expansion of digital technologies, ensuring secure and legally recognized electronic transactions has become essential. In this chapter, we will be interested by the foundation concepts related to our project. We start by giving a comprehensive review of existing electronic signature technologies, then we focus on the regulatory frameworks focusing on relevant laws, regulations, and institutional structures that support this vital aspect of information security. this chapter will provide a global vision for identifying best practices, market trends and potential areas for innovation.

1.2 Electronic Signature and Certification Technologies

1.2.1 Introduction to Cryptography

Cryptography is the science of securing communication and information from adversaries by transforming data into an unreadable format. It plays a crucial role in ensuring confidentiality, integrity, authentication, and non-repudiation in digital communications. Cryptography has evolved from ancient techniques to sophisticated mathematical algorithms used in modern cybersecurity. The primary objective of cryptography is to protect data from unauthorized access and tampering while ensuring that only intended recipients can decipher the information.

The fundamental principles of cryptography include:

Confidentiality Ensuring that only authorized users can access sensitive information.

Integrity Protecting data from unauthorized modifications or corruption.

Authentication Verifying the identity of communicating parties.

Non-repudiation Preventing parties from denying their actions in a transaction.

Cryptographic techniques can be broadly categorized into symmetric and asymmetric cryptography, each serving distinct purposes in securing digital transactions and communications.

Symmetric Cryptography

Symmetric cryptography, also known as secret-key cryptography, employs a single key for both encryption and decryption. The sender and receiver must share the same secret key, which must be kept secure to prevent unauthorized access. Symmetric algorithms are generally faster and more efficient, making them ideal for encrypting large volumes of data. However, the main challenge lies in securely distributing the secret key between parties.

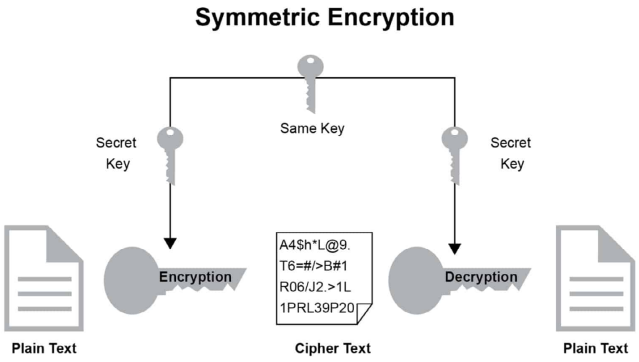

Figure 1.1: Symmetric Encryption Schema

Some widely used symmetric encryption algorithms include:

Advanced Encryption Standard (AES) A widely adopted encryption standard known for its robustness and efficiency. It supports key sizes of 128, 192, and 256 bits, offering strong security.

Data Encryption Standard (DES) An older encryption method that uses a 56-bit key; however, it is now considered insecure due to vulnerability to brute-force attacks.

Triple DES (3DES) An enhanced version of DES that applies the encryption process three times using multiple keys, increasing security but at the cost of performance.

Blowfish A fast and flexible block cipher with a variable key length ranging from 32 to 448 bits, widely used in security applications.

Twofish A successor to Blowfish, offering improved security and efficiency with a block size of 128 bits and key sizes up to 256 bits.

Asymmetric Cryptography

Asymmetric cryptography, also known as public-key cryptography, uses a pair of keys: a public key for encryption and a private key for decryption. Unlike symmetric cryptography, the public key can be openly shared, while the private key remains confidential. This approach eliminates the need for secure key distribution and is widely used in secure communications, digital signatures, and authentication mechanisms.

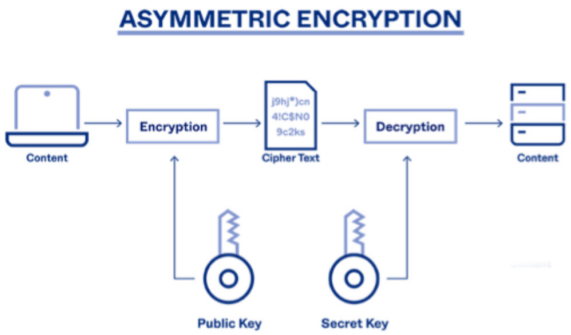

Figure 1.2: asymmetric encryption

Key asymmetric encryption algorithms include:

Rivest-Shamir-Adleman (RSA) One of the most commonly used public-key encryption algorithms, offering strong security based on the difficulty of factoring large prime numbers. RSA is widely implemented in secure web communications and digital signatures.

Elliptic Curve Cryptography (ECC) A more efficient alternative to RSA, providing high security with smaller key sizes. ECC is commonly used in mobile devices and modern cryptographic protocols due to its efficiency.

Diffie-Hellman (DH) Key Exchange A protocol that allows two parties to securely establish a shared secret over an insecure channel. It is foundational for secure key agreement protocols.

Digital Signature Algorithm (DSA) Used for verifying the authenticity and integrity of messages through digital signatures, ensuring that data has not been tampered with.

Symmetric vs. Asymmetric Encryption

Both symmetric and asymmetric cryptography play vital roles in modern cybersecurity. While symmetric encryption is preferred for high-speed data encryption, asymmetric encryption is essential for secure key exchanges and authentication processes. In the context of a startup for electronic signature and certificate services, cryptographic techniques ensure the integrity, authenticity, and non-repudiation of digital documents. Symmetric encryption can be used to encrypt stored documents efficiently, while asymmetric cryptography enables secure digital signatures and certificate management. Combining both methods, as seen in hybrid encryption systems, allows for enhanced security and efficiency in digital document processing and electronic transactions. As cyber threats continue to evolve, cryptographic advancements remain a cornerstone of secure electronic signature services and regulatory compliance. Understanding the strengths and limitations of different cryptographic approaches is crucial for designing a robust platform that meets the legal and security demands of digital certification. [1]

Hash Functions

Hash Functions are a one-way function which are easier to compute in one direction. The forward direction of a function can be computed fast, while the inverse can take months. Changing one bit of input data of a Hash function results in changing around half of the output bits. Hash Function should be free of collisions. This

means it can be extremely difficult to locate two sequences that give the same result. It converts input messages of different lengths into output sequences of fixed length. The output sequence is often referred a *hash value* or a *message digest*. Hash Functions are very useful for different cryptographic purposes. For example, Instead of storing plaintext passwords, systems store their hashes. Even if the database is breached, attackers cannot easily retrieve the original passwords. also it is more appropriate to apply heavy asymmetric cryptographic on hash value instead of the hole message for efficiency considerations. the most common algorithms used for Hash functions are

- SHA-2 (SHA-256, SHA-512): Widely used in TLS, PGP, and Bitcoin.
- SHA-3: A newer family designed as an alternative to SHA-2.
- BLAKE2: Faster than SHA-3 and used in some security applications.
- MD5 : Deprecated due to vulnerabilities (collision attacks).

1.2.2 Digital Signature Schemes

A digital signature schema is a mathematical process used to validate the authenticity and integrity of a message. Formally, Digital Signature Schemes consists of triplet (Gen, Sign, Vrfy) where

- Gen is the key-generation algorithm
- Sign the signing algorithm
- Vrfy is the verification algorithm

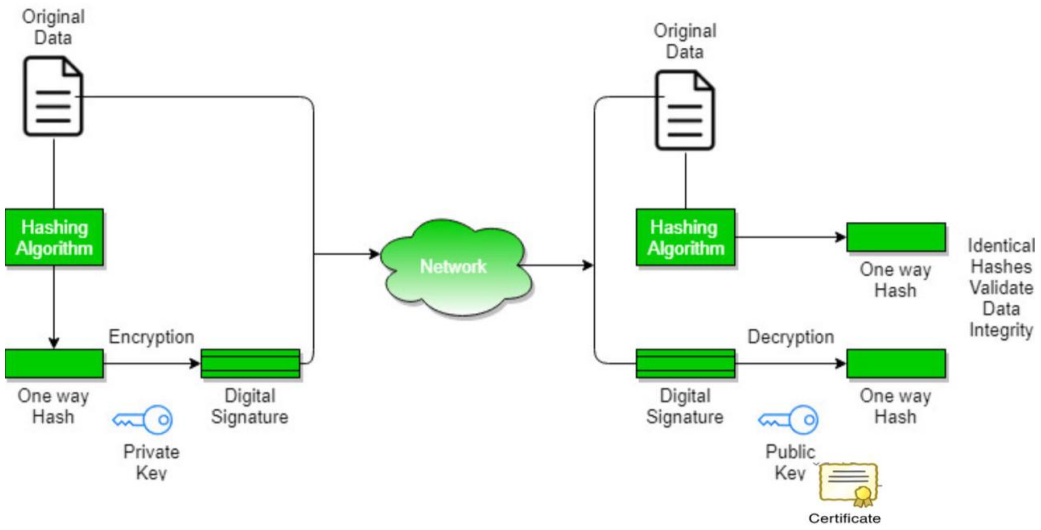

Figure 1.3: Digital Signature Schema

The digital signature scheme follows the following steps as illustrated in 1.3:

1. a hash value of the original message is computed by applying the hash function then it is encrypted using the private key of the sender to form the digital signature. (digital signature = encryption (private key of sender, hash value).

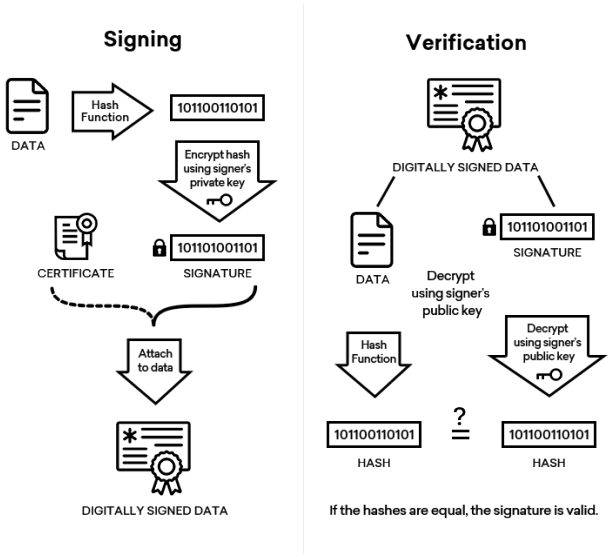

Figure 1.4: Digital Signature with Certificate Usage

2. A digital signature is then transmitted with the message. (message + digital signature is transmitted)
3. The receiver decrypts the digital signature using the public key of the sender. (This assures authenticity, as only the sender has his private key so only the sender can encrypt using his private key which can thus be decrypted by the sender's public key).
4. The receiver now has the hash value.
5. The receiver can compute the hash value again from the message (actual message is sent with the digital signature).
6. The hash value computed by receiver and the hash value (got by decryption on digital signature) need to be same for ensuring integrity.

A critical limitation of the previous digital signature scheme is its susceptibility to man-in-the-middle (MITM) attacks. In this scenario, an attacker intercepts communications and spoofs the legitimate sender's public key. When the recipient verifies the signature using the compromised public key, they falsely assume the message originated from the trusted sender. To mitigate this vulnerability, the recipient must verify the authenticity of the sender's public key before validating the signature. This ensures the public key truly belongs to the legitimate sender and has not been spoofed by an attacker.

1.2.3 Digital Certificates

To answer to the limitation of the previous digital signature schema, Digital Certificates provide a good solution. A digital certificate is a cryptographically signed document that binds a public key to the identity of its legitimate owner, verified by a trusted third party known as a Certificate Authority (CA). A digital certificate is fundamentally a cryptographically signed document issued by a Certification Authority (CA). The CA signs the certificate using its private key, which means anyone can verify its authenticity using the corresponding public key of that CA. This verification process ensures the certificate's integrity and confirms it was indeed issued by the claimed authority. However, to establish complete trust, this process doesn't stop at a single certificate. The CA's own certificate must itself be signed by a higher-level CA, creating a chain of trust. This chain is recursively verified until reaching a root CA certificate - the ultimate trust anchor that is:

- Self-signed (its own issuer and subject are identical)
- Pre-installed in operating systems and browsers
- Universally trusted by all participating parties

Only when this entire chain validates correctly - with each link properly authenticating the next - can we have full confidence in the original certificate's authenticity. This hierarchical trust model is what makes PKI (Public Key Infrastructure) both scalable and secure against impersonation attacks.

1.2.4 Public Key Infrastructure

A Public Key Infrastructure (PKI) is a comprehensive system designed to provide secure communication over networks by using cryptographic technologies. It involves the use of digital certificates, private and public keys, and trusted third-party authorities, primarily the Certification Authority (CA). PKI enables the management, distribution, and verification of public keys and certificates, ensuring that sensitive data remains confidential and authentic. Below is a detailed breakdown of the basic functions of PKI.

Public and Private Key Management

PKI relies heavily on asymmetric cryptography, which involves two keys: a public key and a private key. The public key is shared openly, while the private key is kept confidential. These two keys are mathematically linked, such that data encrypted with one key can only be decrypted using the other. This provides security for communication, as only the intended recipient with the private key can decrypt the message. Public keys are used in a variety of tasks, including encryption and digital signatures.

- **Private Key** Kept secret by the key's owner. It is used to decrypt messages encrypted with the public key or to create a digital signature.
- **Public Key** Available to anyone who wants to communicate securely with the key's owner. It is used for encrypting messages or verifying digital signatures.

Certification Authority (CA)

The Certification Authority (CA) is the trusted entity within a PKI responsible for issuing and managing digital certificates. The CA verifies the identity of the entities requesting certificates, ensuring that public keys are tied to the correct individuals or organizations. The CA also signs the certificates, creating a chain of trust. Without a reliable CA, the integrity of the PKI system would be compromised, as users would not be able to trust the public keys in the certificates.

- The CA plays a pivotal role in establishing trust by certifying that the identity tied to a public key is legitimate.
- The CA is also responsible for revoking certificates if they are no longer valid or if they are compromised.

Certification authorities fall into two categories. Root certification authorities are at the top of the hierarchy, while intermediate certification authorities distribute certificates under their authority.

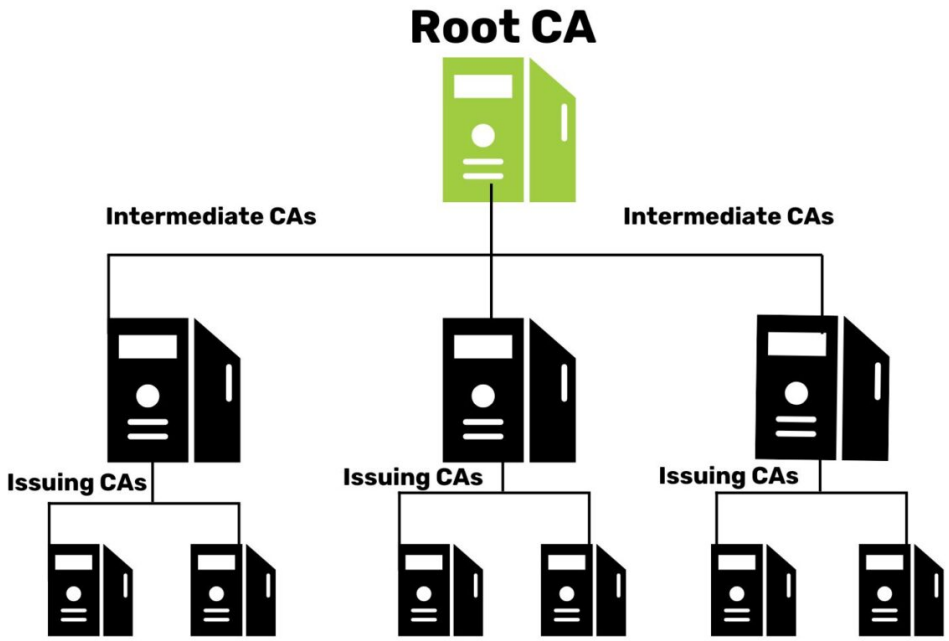

Figure 1.5: PKI-Hierarchy

Digital Certificate

A digital certificate is a cryptographic credential that binds a public key to an identity (person, organization, or device). It includes:

- The public key of the entity.
- Information about the entity (such as the name and email address).
- The certificate's expiration date.
- A signature from the CA verifying the certificate's authenticity.

The certificate acts as an electronic passport, allowing users to trust that the public key truly belongs to the entity claiming it.

Certificate Revocation

Sometimes, certificates must be revoked before their expiration date, especially if the private key has been compromised or if the certificate is no longer required. The CA can revoke certificates, and this revocation is tracked through Certificate Revocation Lists (CRLs) or through an Online Certificate Status Protocol (OCSP). These protocols allow users to verify if a certificate is still valid or has been revoked.

- **CRLs (Certificate Revocation List)**: A list of certificates that have been revoked before their expiration.
- **OCSP (Online Certificate Status Protocol)** : A real-time method of checking the status of a certificate.

PKI Components

The following components work together to ensure the proper functioning of a PKI system:

- **Certification Authority (CA)**: Issues and manages digital certificates. It ensures that only valid and trusted certificates are distributed.
- **Registration Authority (RA)** :Acts as an intermediary between the end-user and the CA. It authenticates the user's identity before forwarding the request for a certificate to the CA.
- **Directory**: A repository or database that stores digital certificates and other critical information necessary for verification. Directories can be accessed by users to find public keys or certificates.
- **Key Management** : The handling, storage, and protection of the cryptographic keys (both private and public) are crucial in a PKI. This includes key generation, distribution, storage, and disposal.

The Role of PKI in Securing Digital Communications

PKI is not just a theoretical framework — it enables the implementation of several common technologies:

- **Secure Web Activity:**: HTTPS and SSL/TLS protocols, which establish secure connections between browsers and servers, use PKI for the underlying encryption powered by digital certificates and public key cryptography.
- **Email Encryption and Signing** : Exchanging sensitive information via email benefits from PKI standards through certificate types such as S/MIME (Secure/Multipurpose Internet Mail Extensions) to encrypt and digitally sign emails.
- **Code Signing Certificates** : Applies a cryptographic signature to software code, sealing it against unauthorized modifications and identifying the publisher.
- **Authentication and Access Control** : Certificate-based authentication mechanisms for enterprise VPN access and building entry systems offer much stronger security than traditional ID-password protocols, using PKI methodologies.
- **Blockchain Transactions** : All transactions on blockchain ledgers involve transferring digital currency by signing with asymmetric cryptographic keys, enabled by PKI principles around the mathematical link between keys.

As global digital connectivity continues to grow across all industries, PKI will remain a foundational element that enables privacy, trust, and security through established standards around certificates, identity management, and encryption.

1.3 Legal Framework for Digital Government in Algeria

Upon the foundational concepts of cryptography, digital security, digital signature schemes, and Public Key Infrastructure (PKI), presented in the previous sections. We examine in this section how our country has implemented these principles and the legal and institutional framework to govern digital life. Cryptography ensures confidentiality, integrity, and authentication, while digital signatures and PKI provide trust in electronic

transactions—elements that are crucial for a secure digital economy. Algeria has established laws such as Loi n° 15-04 on electronic signatures and Loi n° 18-07 on data protection , alongside institutions like the Agence Nationale de Certification Électronique (ANCE) [1] and the Autorité de Protection des Données Personnelles (APDP) [2], to enforce these security mechanisms. in the following paragraphs , We try to explores Algeria's regulatory approach to digital signatures, certification, data hosting, cybersecurity, and privacy, assessing how well the country aligns with global standards and where challenges remain in securing its digital ecosystem.

1.3.1 Electronic Signatures and Certification

Loi n° 15-04 (2015) on electronic signatures: Scope, legal validity, and requirements Algeria's Loi n° 15-04 (2015), enacted on February 1, 2015, establishes the legal framework for electronic signatures (e-signatures) and electronic certification services, aligning with international standards such as the EU eIDAS Regulation and UNCITRAL Model Law on Electronic Signatures. The law defines three types of e-signatures—simple, advanced, and qualified—with qualified electronic signatures (QES) holding the highest legal validity, equivalent to handwritten signatures under Article 9. To ensure trust, the law mandates that qualified signatures must be issued by accredited certification authorities (Prestataires de Services de Certification Électronique, PSCE) under the supervision of the Agence Nationale de Certification Électronique (ANCE). Key requirements include secure signature creation devices (SSCD), identity verification of signatories, and cryptographic integrity checks. The law also outlines liability for certification providers in case of breaches, reinforcing security and accountability in digital transactions. By recognizing e-signatures in administrative, commercial, and judicial contexts, Loi n° 15-04 facilitates Algeria's transition toward a paperless economy, though challenges remain in widespread adoption and interoperability with global PKI systems.

Décret exécutif n°17-156(2017):Implementing Framework for Electronic Certification Services Complementing Loi n° 15-04 (2015) on electronic signatures, Décret exécutif n° 17-156 (2017), enacted on June 30, 2017, provides the technical and administrative requirements for accrediting Prestataires de Services de Certification Électronique (PSCE)—Algeria's trusted providers of digital certificates. The decree establishes a rigorous approval process supervised by the Agence Nationale de Certification Électronique (ANCE), requiring PSCEs to meet strict security, financial, and operational standards before issuing qualified certificates. Key provisions include:

- **Accreditation Process:** PSCEs must submit technical documentation, undergo audits, and demonstrate compliance with ISO 27001 (information security) and PKI best practices.
- **Obligations of Providers:**
    1. Secure storage of cryptographic keys in Hardware Security Modules (HSMs).
    2. Mandatory identity verification for certificate applicants.
    3. Revocation mechanisms for compromised certificates (via Certificate Revocation Lists, CRLs or OCSP).
    4. **Supervision and Penalties:** ANCE conducts periodic inspections, and non-compliant PSCEs face suspension or revocation of accreditation.

By formalizing the roles of PSCEs, Décret 17-156 strengthens trust in Algeria's e-government, e-commerce, and digital banking ecosystems. However, challenges persist in interoperability with foreign certificates and market adoption due to limited PSCE availability.

[1] https://www.agce.dz/
[2] https://anpdp.dz/fr/accueil/

1.3.2 Data Hosting and Cybersecurity

Loi n° 08-09 (2008): Algeria's Cybercrime Law Enacted on March 4, 2008, Law No. 08-09 represents Algeria's first comprehensive legal framework to combat cybercrime. It aligns partially with international standards like the Budapest Convention and addresses offenses ranging from hacking to online fraud, with strict penalties to deter digital crimes. The law categorizes cybercrimes into five core areas:

1. System Attacks (Articles 5–9):
    - Unauthorized access to IT systems (Article 5).
    - Data interception, deletion, or sabotage (Articles 6–7).
    - Malware deployment (Article 9).
2. Digital Fraud (Article 10):
    Identity theft, bank card forgery, and phishing scams.
3. Privacy Violations (Article 11):
    Unlawful recording/sharing of personal data.
4. Illicit Content (Article 12):
    Terrorism propaganda, defamation, child pornography.
5. Cyber-Enabled Money Laundering (Article 13).

Décret exécutif n° 10-104 (2010): Algeria's Data Hosting Regulations Issued on September 30, 2010, this executive decree establishes the legal framework for data hosting activities in Algeria, implementing provisions of the broader telecommunications law (Loi n° 09-04). It applies to all entities offering hosting services within Algerian territory. It englobes the following key provisions

1. Licensing Requirements
    - Mandatory authorization from the ARPT (Telecom Regulatory Authority)
    - Separate licenses required for:
        - Basic web hosting
        - Cloud hosting services
        - Critical infrastructure hosting
2. Data Localization Obligations
    - Hosting providers must maintain primary servers physically located in Algeria
    - Mirroring of sensitive government data required on national territory
    - Foreign providers must partner with Algerian-licensed hosts
3. Security Measures
    - Minimum ISO 27001 certification
    - Mandatory data encryption for personal information
    - 24/7 monitoring systems
    - Incident reporting within 72 hours

4. Content Management

    - Cooperation with judicial takedown requests
    - Log retention for 12 months
    - Identification verification for certain clients

1.3.3 Personal Data Protection

Law No. 18-07 of June 10, 2018 In parallel, Law No. 18-07, enacted on June 10, 2018, focuses on the protection of individuals in the processing of personal data. This legislation aims to ensure respect for individuals' privacy during data processing activities and imposes strict conditions for the collection, storage, and use of personal information, thereby enhancing user trust in digital services. Key provisions of Law No. 18-07 include:

Scope

The law applies to both public and private entities that process personal data within Algeria or use automated means located within the country.

Rights of Individuals

It grants individuals several rights regarding their personal data, including access, rectification, erasure, and objection to processing.

National Authority for the Protection of Personal Data (ANPDP)

This authority is responsible for overseeing compliance with data protection laws and ensuring that citizens' rights regarding their personal data are respected .

1.3.4 E-Commerce and Digital Transactions

Loi n° 18-05 (2018): Algeria's E-Commerce Framework Enacted on February 26, 2018, this law establishes Algeria's first comprehensive legal framework for electronic commerce, addressing consumer protection, business obligations, and digital transaction validity. the following point resumes its main outputs :

1. Consumer Rights Protections
2. Business Obligations
3. Transaction Validity
4. Dispute Resolution

1.4 Regulatory Authorities

The implementation of Law No. 15-04 led to the establishment of a structured framework for certification authorities in Algeria:

1.4.1 National Authority for Electronic Certification (ANCE)

This authority oversees the entire certification ecosystem, ensuring compliance with legal standards and policies related to electronic certification

1.4.2 Governmental Authority for Electronic Certification (AGCE)

Responsible for managing governmental certification services and ensuring that public sector entities adhere to established regulations [2]

Figure 1.6: Governmental Authority for Electronic Certification (AGCE) Logo

1.4.3 Economic Authority for Electronic Certification (AECE)

Tasked with regulating private sector certification services, this authority operates under the supervision of the Regulatory Authority for Post and Electronic Communications (ARPCE) [3]

Figure 1.7: Economic Authority for Electronic Certification (AECE) Logo

These authorities are responsible for:

- Developing and approving certification policies.
- Auditing certification service providers to ensure compliance with legal requirements.
- Issuing certificates that authenticate identities and secure digital communications.

1.5 Conclusion

Throughout this chapter, we provide a broad overview of the key concepts related to our project. Our focus has been primarily on the PKI framework and the Algerian regulatory context, as our goal is to develop a startup that aligns with these legal and technical requirements. In the next chapter, we will explore the technical aspects that will enable us to develop the necessary software for our future enterprise.

Chapter 2

Technical Background

2.1 Introduction

The development of a secure and compliant electronic signature and certification platform requires the integration of multiple technological components, each fulfilling a specific role within a layered architecture. This chapter provides an overview of the core technologies employed in the implementation of the proposed solution. These technologies are selected based on their reliability, security features, scalability, and compatibility with public key infrastructure (PKI) systems.

The technical stack is structured into four main layers:

- **Cryptographic foundation**, where OpenSSL and Node.js's native `crypto` module are used for generating keys, creating digital signatures, and managing certificates;
- **Backend application logic**, implemented with Node.js and Express.js, responsible for processing requests and orchestrating interactions between system components;
- **Authentication and session management**, secured using JSON Web Tokens (JWT), which enable stateless and tamper-evident user authentication;
- **Frontend interface**, designed using React.js to provide an intuitive and responsive user experience.

Additionally, data storage is handled through a relational SQL-based database (MySQL), which supports structured storage of user information, certificates, and audit logs. Each technology has been evaluated and adopted according to current best practices and its ability to comply with legal and technical requirements for electronic certification systems, particularly in alignment with Algerian regulations.

This chapter aims to contextualize the usage of each tool within the system architecture, highlighting not only its functionality but also the rationale behind its selection, supported by official documentation and international standards

2.2 Security and Cryptographic Infrastructure

Electronic signature systems are fundamentally dependent on robust cryptographic mechanisms to ensure data integrity, confidentiality, user authentication, and non-repudiation. This section presents the core technologies

that form the cryptographic backbone of the proposed platform: OpenSSL and Node.js's built-in `crypto` module.

These tools enable the generation and management of public-private key pairs, digital signatures, and certificate-based trust systems aligned with Public Key Infrastructure (PKI) standards. The following subsections detail their respective roles and implementation in the system.

2.2.1 OpenSSL: Core Capabilities and Role in the Platform

OpenSSL is an open-source cryptographic toolkit that provides comprehensive support for secure communications and cryptographic operations. It implements widely used protocols such as TLS (Transport Layer Security) and SSL (Secure Sockets Layer), and offers a wide range of features critical to the implementation of Public Key Infrastructure (PKI)-based systems [4].

Overview of OpenSSL Capabilities

OpenSSL's core functionalities relevant to this project include:

- **Support for various cryptographic algorithms:** Symmetric (e.g., AES, DES), asymmetric (RSA, DSA, ECC), and hashing functions (SHA-1, SHA-2, MD5).
- **TLS/SSL protocol support:** Including modern and secure versions such as TLS 1.2 and TLS 1.3.
- **Certificate management:** Generation, signing, validation, and revocation of X.509 certificates.
- **Command-line utilities:** For executing cryptographic tasks such as encryption, key generation, and CSR creation.
- **Multi-language bindings:** Usable from languages like C/C++, Python (via `ssl` or `pyOpenSSL`), and Node.js (indirectly via the `crypto` module).

Installation and Setup

On Unix-like systems, OpenSSL can typically be installed via package managers. For example:

```
sudo apt update
sudo apt install openssl
openssl version
```

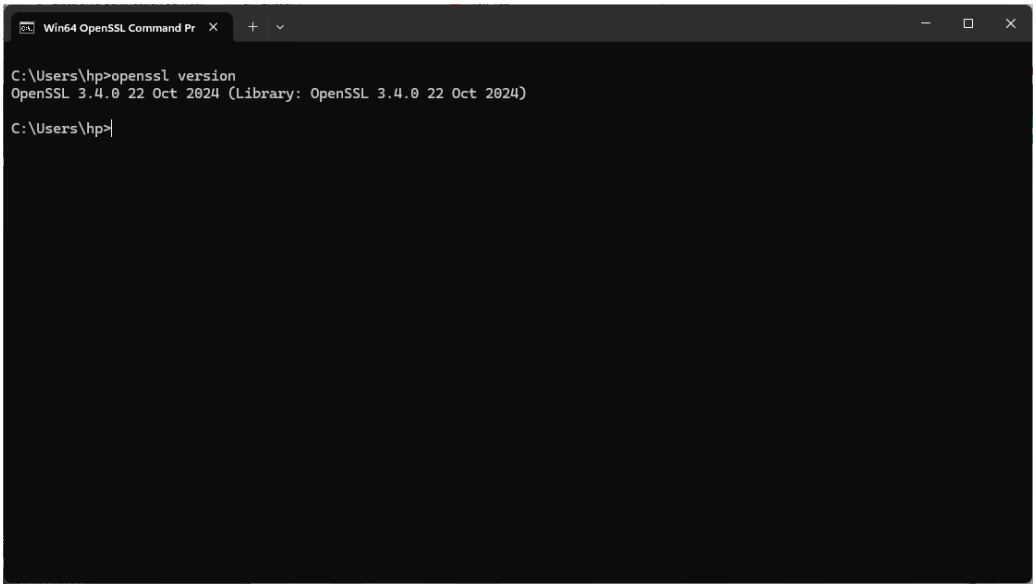

Figure 2.1: Checking the installed OpenSSL version

Key Operations Using OpenSSL

Private Key Generation The following command generates a 2048-bit RSA private key and stores it in PEM format:

```
openssl genrsa -out mykey.pem 2048
```

Certificate Signing Request (CSR) A CSR is required when requesting a certificate from a Certificate Authority (CA). It includes public key info and identity details:

```
openssl req -newkey rsa:2048 -nodes -keyout domain.key -out domain.csr
```

During the process, you will be prompted for information such as:

1. Country Name (2 letter code)
2. State or Province Name (full name)
3. Locality Name (eg, city)
4. Organization Name (eg, company)
5. Organizational Unit Name (eg, department)
6. Common Name (eg, www.example.com)

It is very important to keep the private key secure and to submit the CSR to a trusted CA. The CA will review the request and issue a signed certificate if approved.

Self-signed Certificate Creation Self-signed certificates can be useful for internal or testing environments:

```
openssl req -x509 -newkey rsa:2048 -keyout domain.key -out domain.crt -days 365
```

This command will:

1. Generate a private key: `-newkey rsa:2048` generates a 2048-bit RSA private key.
2. Store the private key: `-keyout domain.key` saves the private key to a file named `domain.key`.
3. Create a self-signed certificate: `-x509` specifies that a self-signed certificate should be created.
4. Store the certificate: `-out domain.crt` saves the certificate to a file named `domain.crt`.
5. Set validity period: `-days 365` sets the validity period of the certificate to 365 days.

Self-signed certificates are not trusted by default. Browsers will typically display warnings when accessing websites using self-signed certificates. Self-signed certificates have to be used with caution. They are primarily intended for testing and development purposes.

Symmetric Encryption / Decryption Encrypt a file:

```
openssl enc -aes-256-cbc -salt -in plaintext.txt -out encrypted.bin
```

Decrypt a file:

```
openssl enc -aes-256-cbc -d -in encrypted.bin -out decrypted.txt
```

Hashing and Integrity Verification Generate a hash (SHA256):

```
openssl dgst -sha256 -out file.hash plaintext.txt
```

Hash verification is a forward operation. If the hash values differ, it indicates that the integrity of the file has been compromised.

Viewing Certificate Information We can inspect a certificate information as following:

```
openssl x509 -in certificate.crt -text -noout
```

Public Key Extraction Extract the public key from a private key:

```
openssl rsa -in private.key -pubout -out public.key
```

Creating a Certificate Authority (CA)

A Certificate Authority (CA) is a trusted entity responsible for issuing digital certificates, which validate the ownership of public keys and establish trust in cryptographic systems. A CA plays a critical role in Public Key Infrastructure (PKI), enabling secure communication and data integrity across various digital platforms. OpenSSL, a powerful and flexible cryptographic tool, facilitates the creation and management of a CA.

To create a CA, the first step is generating a private key that will serve as the foundation of the CA's trust. For example, the command:

```
openssl genrsa -out ca.key 2048
```

generates a 2048-bit RSA private key stored in `ca.key`.

Next, a self-signed certificate for the CA is generated using this private key. This certificate acts as the CA's identity. The command:

```
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt
```

creates a self-signed certificate valid for 10 years and stores it in `ca.crt`.

Once the CA is established, it can issue certificates to other entities. For instance, a Certificate Signing Request (CSR) must be submitted by the entity seeking a certificate. This is done using a command like `openssl req -new -key server.key -out server.csr`, where `server.key` is the private key of the entity. The CA then signs the CSR using its own private key to generate a certificate. The command:

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key \
-CAcreateserial -out server.crt -days 365 -sha256
```

issues a signed certificate valid for one year and stores it in `server.crt`.

This process allows the CA to authenticate entities and establish a chain of trust. To manage revocations, the CA can maintain a Certificate Revocation List (CRL), which enumerates invalidated certificates. The CRL is created using:

```
openssl ca -gencrl -out crl.pem
```

ensuring that the system adheres to strict security policies.

By setting up a CA, organizations can efficiently manage digital certificates, enabling secure communications and robust authentication mechanisms tailored to their specific needs.

Trust Store and Certificate Lifecycle

Building a trust store involves collecting and managing root and intermediate CA certificates. OpenSSL supports trust validation, revocation checking, and CRL distribution, which are vital for enforcing security policies and maintaining trustworthiness in PKI systems.

2.2.2 Node.js `crypto` Module

To complement OpenSSL, the system leverages the native `crypto` module available in Node.js for programmatic cryptographic operations. This module, built on top of OpenSSL, enables direct access to low-level cryptographic APIs within the application logic [5].

Key functionalities used in this project include:

- **Digital signatures:** The platform uses RSA with SHA-256 for signing and verifying documents. Each signature is generated using the private key stored securely for each user.
- **Hashing:** The SHA-256 algorithm is used to compute digests of documents before signing to guarantee data integrity.
- **Key generation:** In some flows, keys can be generated directly using `crypto.generateKeyPair()`, allowing full automation.

The use of the `crypto` module reduces the dependency on external shell commands and provides tighter integration with the Node.js backend. It is also used to verify signatures during the validation phase of signed documents, ensuring they match the corresponding public key extracted from the user's certificate.

The combined use of OpenSSL and Node.js crypto offers a complete and secure cryptographic infrastructure for building compliant, efficient, and verifiable electronic signature systems.

2.3 Backend Application Logic

The backend of the platform is built using **Node.js**, a JavaScript runtime environment designed for building scalable network applications. It is coupled with the **Express.js** framework to simplify the creation of robust web APIs. This combination provides a lightweight and efficient foundation for handling the core business logic of the certification and electronic signature system. In addition, **JSON Web Tokens (JWT)** are used to manage authentication and session control, ensuring secure and stateless communication between clients and the server.

2.3.1 Node.js and Express.js Framework

Node.js is a cross-platform, event-driven runtime environment built on Google's V8 JavaScript engine [6]. Its non-blocking I/O model and single-threaded architecture make it well-suited for applications that handle a large number of concurrent requests, such as web servers or API-based platforms.

Express.js is a minimal and flexible Node.js web application framework that provides a set of features for building APIs and web applications [7]. In this project, Express.js is used to define RESTful routes for user management, certificate generation, digital signature requests, and verification endpoints.

Key features include:

- Routing system to define secure endpoints for frontend interaction.
- Middleware support for authentication, error handling, and data validation.
- Integration with modules such as `crypto`, `jsonwebtoken`, and `mysql`.

2.3.2 Role of the Backend in the System

The backend is responsible for:

- Receiving and processing user requests related to account creation, authentication, and document signing.
- Interfacing with the OpenSSL command-line utility or Node's `crypto` module to generate and manage cryptographic keys.
- Communicating with the SQL database to store and retrieve certificates, user profiles, and signed documents.
- Exposing RESTful endpoints consumed by the React frontend interface.

This separation of concerns enhances modularity, maintainability, and facilitates secure development practices through clearly defined logic layers.

2.3.3 Authentication with JSON Web Tokens (JWT)

Authentication and session control are managed through **JWT** (JSON Web Tokens), a compact, URL-safe means of representing claims to be transferred between two parties [8]. JWTs are used to authenticate users after login and to protect access to private resources.

A JWT is composed of three parts:

- **Header:** Indicates the token type and the signing algorithm (e.g., HS256).
- **Payload:** Contains user-related claims such as user ID, role, and token expiry.
- **Signature:** Ensures integrity and authenticity by signing the header and payload with a secret or RSA private key.

Tokens are generated upon successful login and sent to the client, which stores them (e.g., in HTTP-only cookies or local storage). For each subsequent request to protected routes, the token is included in the Authorization header as follows:

```
1 Authorization: Bearer <token>
```

The backend validates the token's signature and payload before granting access. This approach ensures stateless session management and reduces server-side overhead.

2.3.4 Security Considerations

Security practices implemented in the backend include:

- Passwords hashed using `bcrypt` before storage.
- JWT expiry time and optional refresh token mechanism.

- Use of HTTPS for encrypted client-server communication.
- Input validation to prevent injection and XSS attacks.

Together, these practices ensure that authentication, authorization, and backend operations align with industry standards and the requirements of electronic certification systems.

2.4 Database Management System

A robust and secure data storage layer is essential for supporting the functionality of an electronic certification platform. In this project, a **relational database system** based on **MySQL** is employed to store and manage structured data related to users, certificates, signed documents, and audit logs. MySQL offers high performance, ACID compliance, wide compatibility, and ease of integration with Node.js applications [9].

2.4.1 Rationale for Using MySQL

MySQL is a widely adopted open-source relational database management system (RDBMS) maintained by Oracle. It offers the following advantages for this project:

- **Structured schema:** Allows the modeling of complex relationships between users, certificates, and documents using primary and foreign keys.
- **Security features:** Includes user access controls, encrypted connections, and support for hashing algorithms.
- **Scalability:** Suitable for medium to large-scale deployments, with options for replication and clustering.
- **Community and tooling:** Supported by rich administrative tools such as phpMyAdmin and MySQL Workbench.

2.4.2 Database Structure and Entities

The relational model is organized into key tables, such as:

- **Users:** Stores personal details, login credentials (hashed), role (e.g., admin, user), and identification numbers (e.g., NIN).
- **Certificates:** Contains metadata about generated certificates, including keys, expiration dates, and issuing authority.
- **Documents:** Represents user-uploaded or signed documents with fields such as filename, hash value, timestamp, and associated certificate ID.
- **Audit logs:** Records actions performed by users or the system, ensuring traceability and accountability.

Entity relationships are enforced through foreign keys and normalized design to maintain consistency and integrity.

2.4.3 Integration with the Backend

The backend, implemented in Node.js, connects to the MySQL database using official drivers such as `mysql2` or ORMs like `Sequelize`. This integration allows for secure and asynchronous data transactions via prepared statements and parameterized queries to prevent SQL injection.

Example connection snippet:

```
const mysql = require('mysql2');
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'password',
    database: 'certification_db'
});
```

2.4.4 Security and Data Integrity

The database layer adheres to best practices for securing user data and maintaining consistency:

- **Password hashing:** All passwords are stored using bcrypt hashing with salting.
- **Input validation and query sanitization:** Prevents injection and ensures database consistency.
- **Access control:** Users are granted minimum required privileges, adhering to the principle of least privilege.
- **Data backup and recovery:** Backup scripts and SQL dump routines are scheduled to ensure data recoverability.

Overall, MySQL provides a reliable and secure data layer that supports the operational, legal, and security requirements of an electronic certification system.

2.5 Frontend Interface

The frontend of the platform is developed using **React.js**, a modern JavaScript library for building user interfaces, particularly single-page applications (SPAs). React provides a component-based architecture, virtual DOM rendering, and efficient state management, making it ideal for creating dynamic and responsive web applications [10].

2.5.1 Why React.js?

React was selected for this project due to its strong community support, modularity, and integration capabilities with RESTful APIs. Its declarative syntax and unidirectional data flow simplify the development of interactive components, reduce debugging time, and improve maintainability.

Key advantages include:

- **Component-based design:** Reusable and encapsulated components enable scalable UI development.
- **Virtual DOM:** Improves performance by minimizing real DOM manipulations.
- **Strong ecosystem:** Integration with tools like Axios (HTTP requests), React Router (routing), and Context API (state sharing).
- **Easy backend integration:** Compatible with JSON-based APIs exposed by the Node.js server.

2.5.2 Frontend Features

The React frontend enables several core features for users and administrators:

- **User authentication interface:** Login and registration forms connected to backend JWT authentication.
- **Dashboard:** Displays information such as uploaded documents, signed certificates, and verification results.
- **Document upload and signature module:** Allows users to select files, submit them for signing, and retrieve signed copies.
- **Admin tools:** Interfaces for certificate generation, user management, and system monitoring.

2.5.3 Interaction with the Backend

Communication between the frontend and backend is done via HTTP requests (typically using Axios). JSON payloads are exchanged with proper authentication headers using JWT tokens.

Example request to retrieve a signed document:

```
axios.get('/api/documents/signed', {
    headers: {
        Authorization: `Bearer ${token}`
    }
});
```

React's useEffect and useState hooks are leveraged to handle component lifecycle and asynchronous data fetching.

2.5.4 State Management and Routing

For managing local and shared component state, the application uses:

- **useState/useEffect:** For local component logic and side effects.
- **React Context API:** For global state such as authentication status or user profile.
- **React Router:** For handling navigation between pages such as login, dashboard, and certificate views.

This architecture allows the frontend to remain responsive, modular, and extensible.

2.5.5 Security Considerations

Security measures in the frontend include:

- Storage of JWT tokens in `httpOnly` cookies to reduce XSS risk (or localStorage with caution).
- Input validation to prevent form-based attacks.
- Secure file uploads with content-type checks and size restrictions.
- HTTPS communication with the backend API.

These practices ensure that sensitive operations such as signing requests and certificate access are protected on the client side.

2.6 Integration of System Components

The proposed electronic certification and signature platform is composed of several interdependent technological layers. These layers — frontend, backend, cryptographic services, and the database — work together to deliver a secure, responsive, and scalable system for managing digital certificates and signed documents.

This section explains how these components interact, both logically and through data exchange, to accomplish the core functionalities of the system.

2.6.1 Logical Architecture Overview

The system follows a modular client-server architecture, where the frontend communicates with the backend via RESTful APIs, and the backend handles business logic, database operations, and cryptographic tasks.

- **Frontend (React.js):** Provides an intuitive interface for users and administrators to interact with the system, including uploading documents, requesting signatures, and viewing certificates.
- **Backend (Node.js + Express):** Acts as the central orchestrator, processing requests from the frontend, performing authentication, invoking cryptographic operations, and managing database records.
- **Cryptographic Services (OpenSSL and crypto module):** Responsible for generating RSA key pairs, signing documents, issuing certificates, and verifying signatures.
- **Database (MySQL):** Stores user data, certificate metadata, signed document hashes, and audit logs.

2.6.2 Component Interaction Flow

The integration is based on a secure and asynchronous communication model. A typical workflow can be described as follows:

1. A user logs in through the React interface. Upon successful authentication, a JWT is issued by the backend.

2. The user uploads a document via the frontend, which is sent to the backend API along with the JWT in the Authorization header.
3. The backend verifies the token, then computes the document's hash using the `crypto` module.
4. A digital signature is created using the user's private key (stored or generated via OpenSSL).
5. The signed document hash and associated metadata are stored in the MySQL database.
6. The user can later view, download, or verify the signature through the React interface.

2.6.3 Benefits of Layered Integration

This decoupled, service-oriented architecture offers the following benefits:

- **Maintainability:** Clear separation of concerns facilitates easier debugging and future enhancements.
- **Security:** Sensitive operations (e.g., signing, key generation) are isolated within secure backend services.
- **Scalability:** Frontend and backend can be scaled independently based on usage.
- **Extensibility:** Additional features such as timestamping, OCSP validation, or LDAP integration can be added without overhauling the core architecture.

The integration strategy adopted ensures that each component contributes efficiently and securely to the overall goal of providing a trusted and legally compliant e-certification system.

2.7 Conclusion

This chapter has presented the technical foundation of the proposed electronic signature and certification platform, covering the four main layers of the system architecture. The cryptographic infrastructure, built on OpenSSL and Node.js's crypto module, provides the security mechanisms necessary for digital signatures and certificate management. The backend application logic, implemented with Node.js and Express.js, orchestrates the system operations while ensuring secure authentication through JWT tokens. The MySQL database offers reliable and structured data storage, while the React.js frontend delivers an intuitive user experience.

The integration of these components creates a comprehensive, secure, and scalable platform that meets the requirements of electronic certification systems and aligns with current industry standards and legal frameworks. This technical foundation enables the implementation of the core functionalities presented in subsequent chapters, ensuring both security and usability for end users and administrators.

Chapter 3

System Design and Implementation

3.1 Introduction

The design and implementation of an electronic signature and certificate management system require a well-structured approach to ensure security, efficiency, and compliance with international best practices as well as local legal frameworks. This chapter presents the overall architecture, functional and non-functional requirements, and key design choices that contribute to the development of a robust and scalable platform for digital signatures and certificate lifecycle management, in alignment with Algerian regulations governing electronic certification and digital trust services.

The system leverages Public Key Infrastructure (PKI) to guarantee the authenticity and integrity of signed documents. By integrating cryptographic techniques and OpenSSL, the platform enables users to securely sign and verify documents while ensuring proper management of digital certificates. The platform is specifically tailored to support deployment in the Algerian market, with features designed to reflect the requirements of national digital transformation policies.

This chapter is structured as follows: first, we define the system requirements, followed by an overview of the solution's architecture. Next, we provide a detailed design phase, which includes the use case diagram, class diagram, and sequence diagrams to illustrate the interactions between system components. These diagrams serve as a foundation for implementing the system, ensuring clarity, maintainability, and regulatory alignment throughout the development process.

3.2 Software Development Lifecycle

The development of CertiSafe followed a structured and iterative approach based on the principles of software engineering. Given the nature of the project — a secure and regulation-compliant platform for digital certification and electronic signatures — it was essential to adopt a development lifecycle that ensures both functional completeness and technical reliability.

We opted for an incremental development model inspired by the agile methodology, which promotes early delivery of working components, frequent validation, and the flexibility to adapt to feedback. This approach was particularly suitable for our context, where different system components such as certificate generation, signature mechanisms, user authentication, and API integration were developed in parallel and progressively

refined.

The development lifecycle can be summarized in the following phases:

1. **Requirements Analysis:** Functional and non-functional requirements were gathered through a study of Algerian digital signature regulations (law n°15-04) and international PKI standards. This ensured that the system would meet legal, security, and usability expectations.
2. **System Design:** A modular and scalable architecture was defined, with clear separation between the frontend, backend, and cryptographic services. UML diagrams (use case, class, and sequence diagrams) were developed to clarify component interactions.
3. **Implementation:** The core functionalities were developed incrementally. Backend services were implemented using Node.js and integrated with OpenSSL for certificate and key operations. The frontend was built using React.js, with role-based interfaces for users and administrators.
4. **Testing and Validation:** Unit and integration testing were carried out for each module. Certificate generation, signature verification, timestamping were validated in test environments to ensure correctness and robustness.
5. **Deployment and Demonstration:** A web-based prototype was deployed as a proof of concept, simulating the operations of a future national Certification Authority (CA). This prototype serves as the technical foundation of the startup vision of CertiSafe.
6. **Future Evolution:** The project is designed to be extendable. Planned enhancements include a mobile application, offline support, B2B API endpoints, multi-signature capabilities, and exploration of secure electronic voting in the context of PKI.

This development lifecycle enabled us to maintain a clear project direction while allowing flexibility in technical decisions. It ensured that each iteration added meaningful value and aligned with the overarching goals of regulatory compliance, scalability, and digital trust enablement in the Algerian context.

3.3 Functional and Non-Functional Requirements

The table below presents a side-by-side overview of the functional and non-functional requirements of the CertiSafe platform. This format provides a consolidated view of what the system must do and the quality standards it must meet.

Table 3.1: Functional vs. Non-Functional Requirements

Functional Requirements	**Non-Functional Requirements**
User registration with personal and organizational data .	All communications must be encrypted via HTTPS and authenticated using JWT.
User login with secure credentials (NIN and password).	Sensitive data (certificates, keys) must be encrypted at rest and access must be role-based.
Administrator dashboard to manage users, review data, issue or revoke certificates.	The system must handle at least 100 concurrent users without performance degradation.
Generation of X.509 certificates signed by CertiSafe's internal CA.	The system must be portable, maintainable, and easily deployable via standard technologies.
Digital signature generation for uploaded documents.	Operations such as signing, revocation, and validation must be logged for auditability.
Export of ZIP package (document, signature, certificate).	The user interface must be responsive and accessible from both desktop and mobile browsers.
Verification of signatures using uploaded files.	The system must comply with Algerian law n°15-04 and international PKI standards.
Timestamping of signed documents.	APIs must be designed for extensibility and enterprise integration (ERP, DMS, etc.).
API endpoints for signing, verifying, retrieving certificates, and revocation checking.	The architecture must support future upgrades (e.g., OCSP, mobile apps, offline support).

3.4 System Design

The design phase plays a crucial role in the software development lifecycle, serving as the bridge between system requirements and actual implementation. A well-structured design not only enhances maintainability and scalability but also ensures that the system meets functional and non-functional requirements efficiently.

In this project, the design is carried out using the Unified Modeling Language (UML), a standardized modeling language widely used for specifying, visualizing, and documenting software architectures. UML provides a comprehensive framework for system modeling, enabling developers to represent structural and behavioral aspects through various diagrams. These diagrams serve as blueprints that facilitate clear communication between developers, analysts, and stakeholders, reducing ambiguities and improving project clarity.

The adoption of UML is particularly beneficial for complex systems such as electronic signature and certificate management platforms, where security, data integrity, and access control are critical concerns. One of the primary advantages of UML is its ability to offer multiple perspectives on the system's architecture. In our case, this is achieved through different diagram types that capture various aspects of the system:

- The **Use Case Diagram** provides a high-level representation of user interactions with the system, defining the key functionalities accessible to different actors.
- The **Class Diagram** illustrates the static structure of the system, detailing entities, attributes, methods,

and their relationships, thereby forming the foundation for object-oriented implementation.

- The **Sequence Diagram** describes the dynamic behavior of the system by modeling interactions between objects over time, highlighting message exchanges and execution flow.

For this project, UML serves as a methodological tool to ensure a structured and efficient system design. The diagrams presented in this section outline the key components and interactions that govern the functionality of our electronic signature platform.

By employing UML, the system architecture is not only well-defined but also easier to extend and maintain, ensuring long-term adaptability.

3.4.1 Use Case Diagram

This diagram allows visualization of the interactions between system actors and the main functionalities they can use.

The system is based on two types of actors:

- **User**: They access electronic signature and document verification services.
- **Admin**: They manage the certificates required for document signing.

All operations require prior authentication (**Login**). The admin assigns certificates to users, which they need to sign electronic documents. Document verification is accessible to all users, even without a certificate, but only after logging into the platform.

User

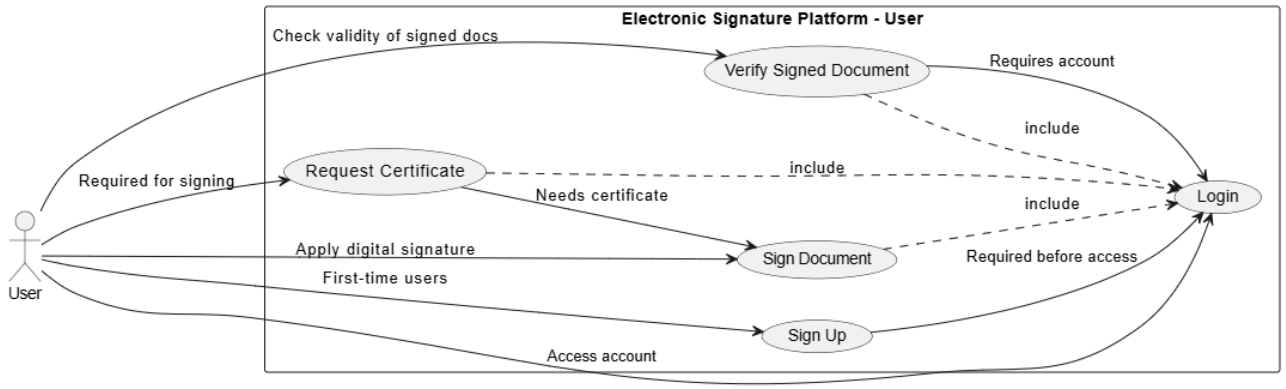

Figure 3.1: Use case diagram "user"

The user is one of the primary system actors. Their role is to interact with the platform to perform several essential actions.

- **Sign Up**: A first-time user must register by providing personal information and company details. This step is mandatory before accessing the platform's services.
- **Login**: After registration, the user must log in to the platform. Authentication is required to access the following functionalities.
- **Request Certificate**: To electronically sign documents, the user must request a certificate. This certificate is assigned by the admin and ensures the authenticity of signatures.
- **Sign Document**: Once the certificate is obtained, the user can digitally sign documents. This functionality relies on cryptographic mechanisms that ensure the security and integrity of signed documents.
- **Verify Signed Document**: The user can verify the authenticity of signatures applied to documents by other users. Unlike document signing, this operation does not require a certificate, but authentication on the platform is still mandatory.

All actions performed by the user require prior login (**include Login**), ensuring the security of operations within the platform.

Admin

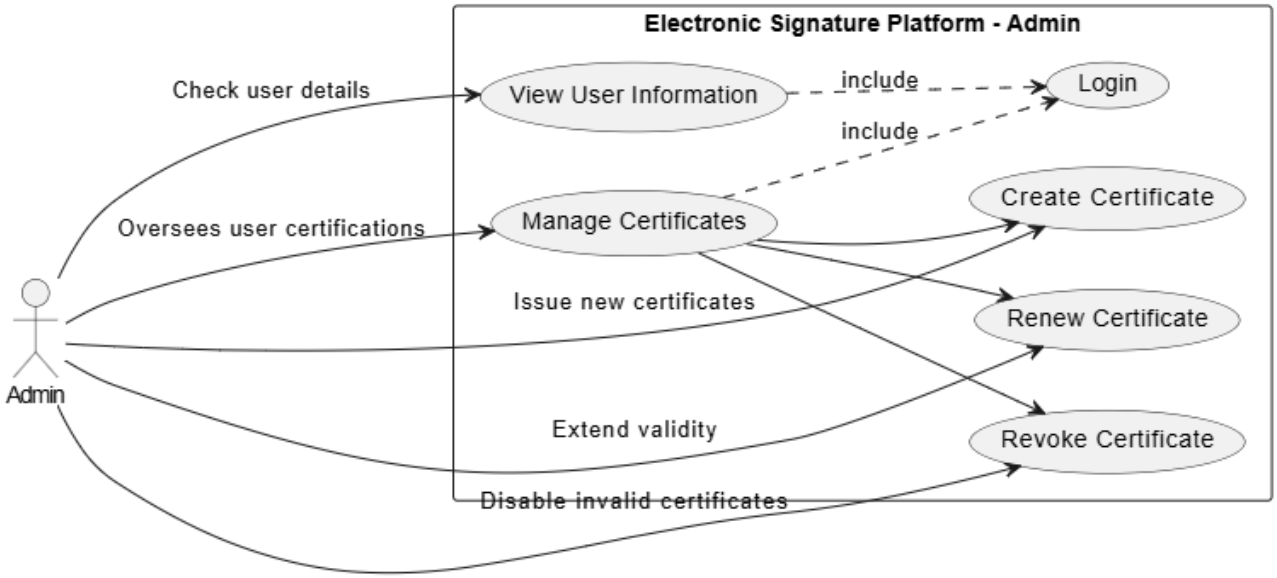

Figure 3.2: Use case diagram "admin"

The administrator plays a crucial role in managing certificates and users. They have access to specific functionalities that allow them to control and oversee the system's proper functioning.

- **Manage Certificates**: The administrator is responsible for assigning and managing certificates required for electronic document signing.
- **Create Certificate**: The administrator can issue a new certificate for a user upon request. This certificate is a mandatory condition for performing a valid electronic signature.
- **Renew Certificate**: When a certificate nears expiration, the administrator can renew it to extend its validity and prevent interruptions in service usage.

- **Revoke Certificate**: In cases of security breaches or invalid certificates, the administrator can revoke them to prevent unauthorized use.
- **View User Information**: The administrator can access details of users registered on the platform, including their company information and certification status.

All these actions require prior authentication (**include Login**), ensuring that only authorized administrators can manage certificates and control platform access.

API Integration

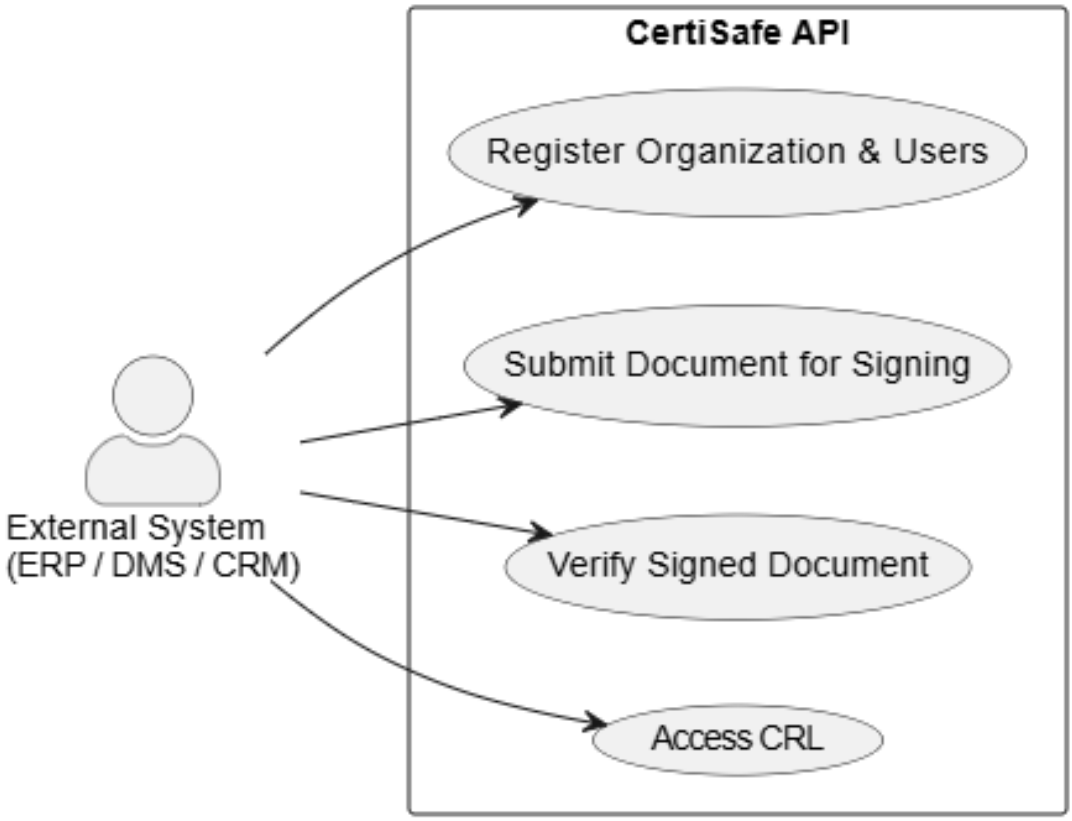

Figure 3.3: Use case diagram "API Integration"

The API integration layer enables external systems—such as ERPs, DMSs, or CRMs—to interact securely and programmatically with the CertiSafe platform. These systems leverage standardized RESTful APIs to automate digital signature workflows and manage certificate-related operations without direct user intervention.

- **Register Organization & Users**: External systems can programmatically register a company and its users into CertiSafe, enabling fully digital onboarding.
- **Submit Document for Signing**: A document can be sent to CertiSafe through the API for automatic signing using a previously generated user certificate.
- **Verify Signed Document**: External systems can upload a signed document, its associated signature, and the corresponding certificate to validate authenticity and integrity.
- **Retrieve Certificate Metadata**: This functionality allows third-party systems to retrieve public details of a user's certificate, such as subject name, validity period, and issuer.

- **Access CRL (Certificate Revocation List)**: External verifiers can access the CRL to check whether a certificate has been revoked by the authority.

All use cases in this interface are secured via HTTPS and require proper API authentication mechanisms (**include Token Authentication**), ensuring that only authorized systems can initiate critical actions like signing or certificate access.

Audit and Maintenance Agent

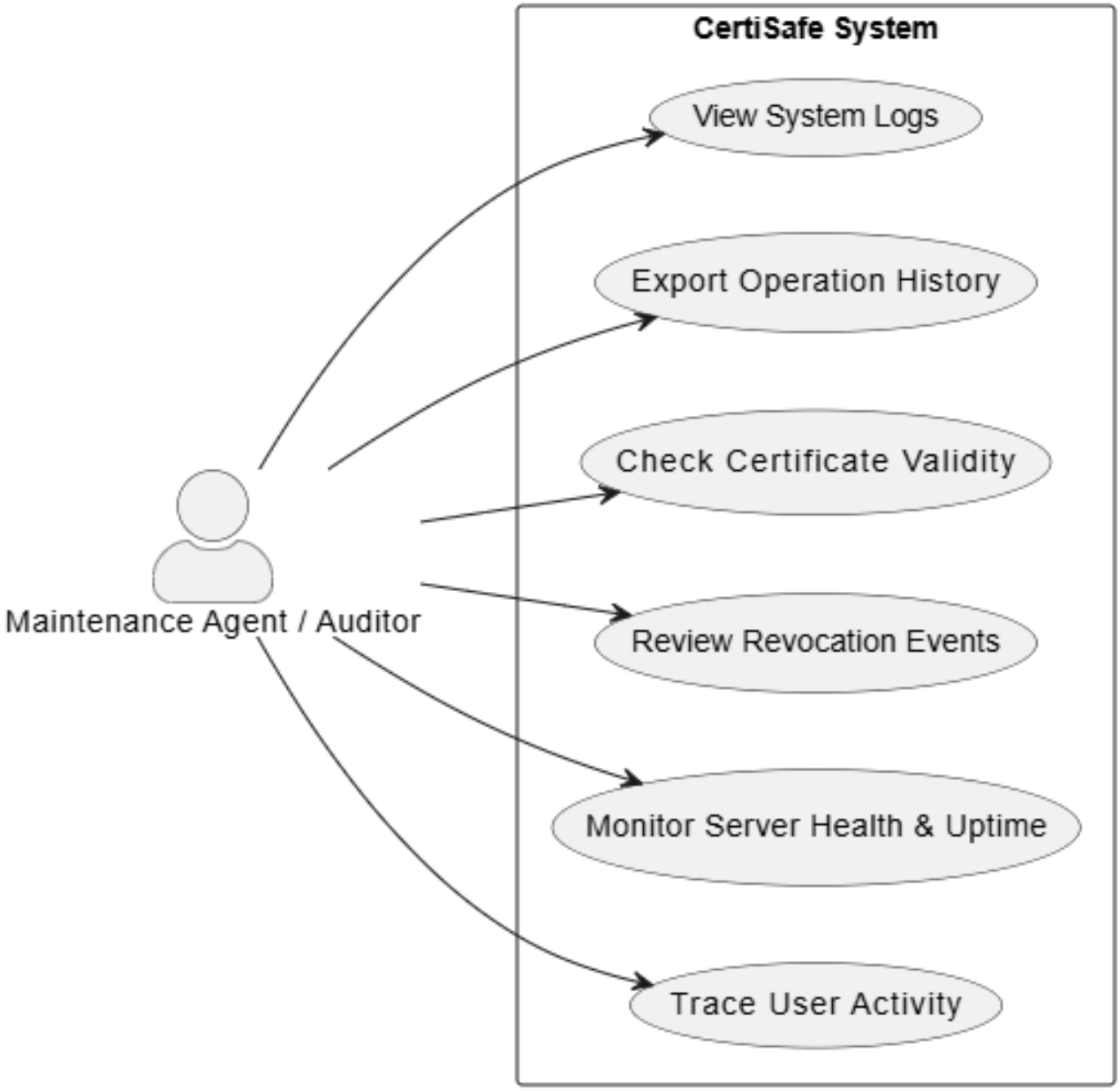

Figure 3.4: Use case diagram "Audit and Maintenance Agent"

The Audit and Maintenance Agent plays a crucial role in ensuring the operational integrity, traceability, and compliance of the CertiSafe platform. This actor is responsible for monitoring logs, reviewing critical events, and validating system reliability. Their activities contribute to maintaining a secure and auditable digital certification environment.

- **View System Logs**: The agent can consult detailed system logs to trace actions related to certificate issuance, revocation, user access, and document signatures.
- **Export Operation History**: Enables the extraction of logs and audit trails for compliance reports or external inspection purposes.
- **Check Certificate Validity**: Allows periodic review of certificate status, including expiration, revocation status, and chain of trust integrity.
- **Review Revocation Events**: The agent can access a history of certificate revocations, including justifications and timestamps.
- **Monitor Server Health & Uptime**: Includes the observation of backend system metrics to detect anomalies or downtime that may affect availability.
- **Trace User Activity**: All user and admin interactions are recorded and can be reviewed by the auditor to detect misuse or irregularities.

All audit and monitoring operations are read-only and strictly restricted to authorized roles. This ensures that audit integrity is preserved, while still allowing comprehensive oversight of system activity.

3.4.2 Class Diagram

The UML class diagram represents the logical structure of the CertiSafe project, highlighting the main entities, their attributes, and the relationships between them. It reflects the responsibilities of different classes in the application and their interaction at the business and database levels.

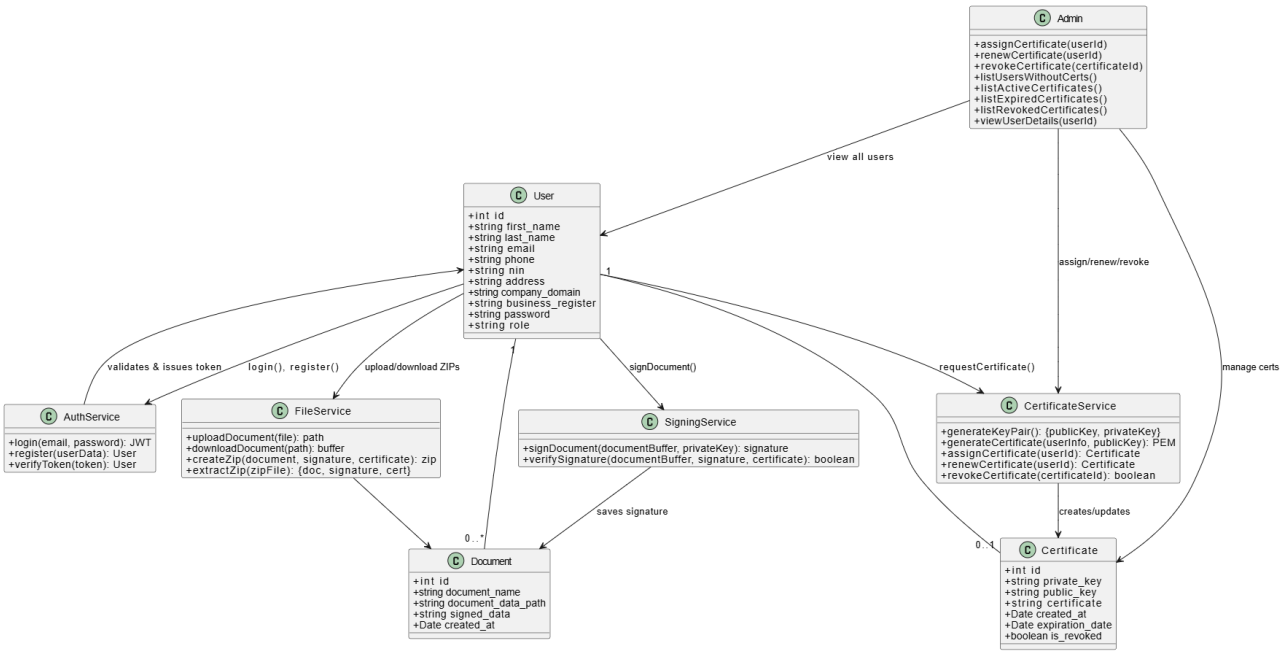

Figure 3.5: Class Diagram

User Class

This class represents a platform user, who can be either an end user or an administrator.

Attributes:

- `id, first_name, last_name`: User identity
- `email, phone, address`: Contact information
- `company_domain, business_register`: Professional details (optional)
- `nin`: Unique national identification number
- `role`: Can be "user" or "admin"

Relationships:

- A User can have one Certificate.
- A User can have multiple Documents.

Certificate Class

A Certificate is an entity representing an X.509 digital certificate assigned to a user.

Attributes:

- `id, user_id`: Link to the User
- `private_key`: RSA private key (PEM format)
- `public_key`: Public key (PKCS1 or X.509)
- `certificate`: Signed certificate in base64 PEM
- `expiration_date`: Expiration date
- `revoked`: Boolean indicating whether the certificate is revoked
- `created_at`: Issuance date

Relationships:

- Each Certificate is linked to a single User.

Document Class

The Document class represents an electronically signed file.

Attributes:

- `id, user_id`: Link to the owner
- `document_name`: File name
- `document_data`: Local file path
- `signed_data`: Signature (base64)

- `created_at`: Date and time of signature

Relationships:

- Each Document is signed by a single User.

Admin Class

Admin is a role inherited from the User class, with additional privileges.

Main Functions:

- `assignCertificate(User)`: Generate a signed certificate for a user.
- `revokeCertificate(Certificate)`: Revoke a compromised certificate.
- `renewCertificate(User)`: Renew an expired certificate.

Although Admin is not a separate class in the database, it is logically represented in the diagram to indicate its role.

Relationships and Business Rules

- `1..1` between User and Certificate: A user has only one active certificate.
- `1..N` between User and Document: A user can sign multiple documents.
- Revocation is managed via a boolean attribute in Certificate.
- Signature verification uses the `public_key` contained in the certificate.

Integrated Security in Classes

- The Certificate class securely stores cryptographic keys.
- The signing and verification process relies on the correspondence between `private_key` and `public_key`.

3.4.3 Sequence Diagram

In these sequence diagrams, we present the interactions between users, administrators, and system components for different operations. These diagrams illustrate how requests are processed, ensuring secure authentication, certificate management, document signing, and verification.

User Registration

The user registration process ensures that only new users can create accounts while preventing duplicate registrations. The steps involved are:

- The user accesses the registration interface and submits their details.

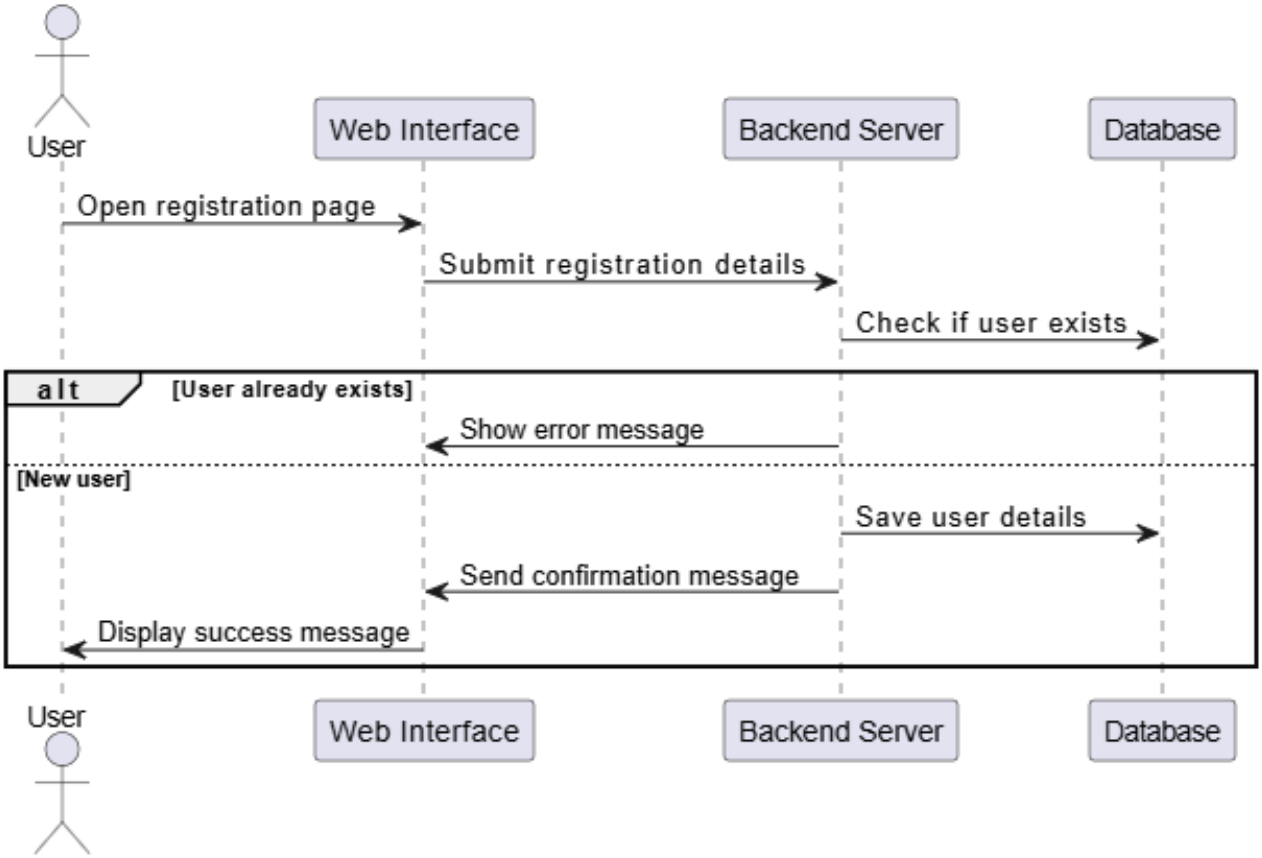

Figure 3.6: User Registration

- The backend server verifies whether the user already exists in the database.
- If the user exists, an error message is displayed.
- If the user is new, their information is stored in the database.
- A confirmation message is sent to the user, indicating successful registration.

User Authentication

Authentication is required for accessing platform functionalities. The process follows these steps:

- The user provides their login credentials.
- The backend server retrieves stored credentials from the database.
- If the credentials match, access is granted.
- If authentication fails, an error message is displayed.

Certificate Request and Generation

To sign documents electronically, a user must obtain a certificate. This process involves both the user and the administrator. The sequence includes:

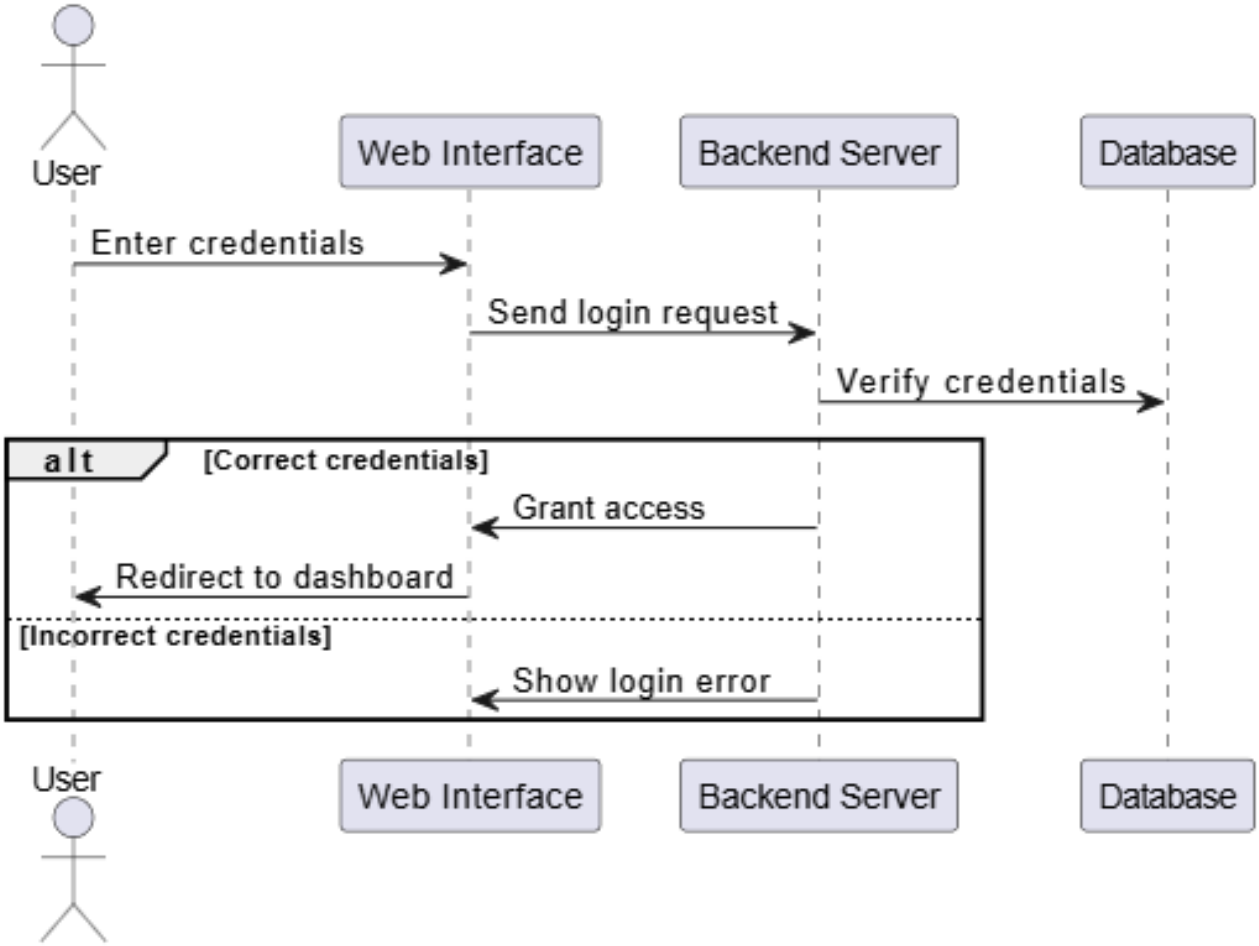

Figure 3.7: User Authentication

- The user submits a certificate request through the web interface.
- The backend server forwards the request to the administrator.
- The administrator validates the request.
- The administrator initiates certificate generation.
- The PKI system generates a new certificate along with the user's private key.
- The backend server stores the certificate and private key securely in the database.
- The system notifies the user that their certificate is ready.
- The user can now access their certificate for digital signing.

This ensures that every digital signature is backed by a **securely issued and stored cryptographic key**.

Electronic Document Signing

Users sign documents digitally using their private key stored in the database. The sequence includes:

- The user uploads a document.

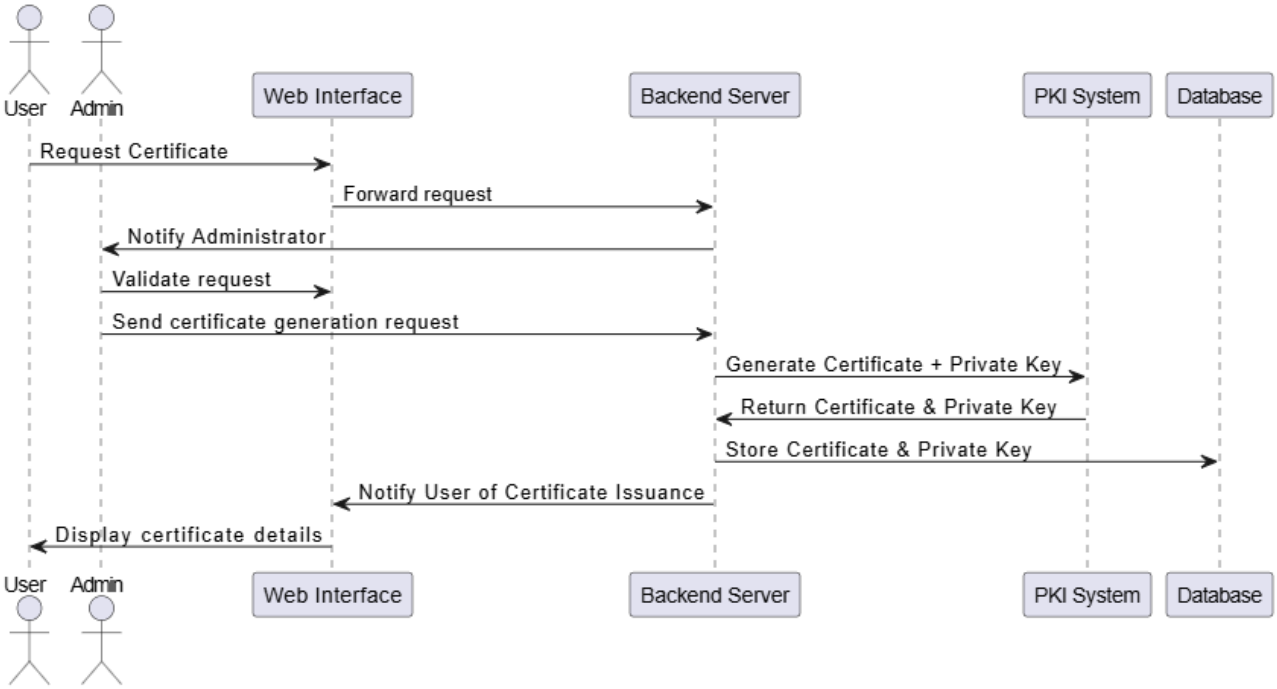

Figure 3.8: Certificate Request and Generation

- The backend server retrieves the private key from the database.
- The PKI system applies a digital signature using the private key.
- The signed document is stored securely in the system.
- The user can view and download the signed document.

Verification of Signed Documents

Verifying signed documents ensures their authenticity using a public key extracted from the certificate. The process consists of:

- The user uploads the signed document for verification.
- The backend server retrieves the certificate from the database.
- The PKI system extracts the public key from the stored certificate.
- The system verifies the document signature against the extracted public key.
- If valid, a confirmation is displayed; if invalid, an error message appears.

Administrator's Certificate Management

The administrator is responsible for managing certificates, including renewal and revocation. The sequence includes:

- The administrator selects certificate management options.

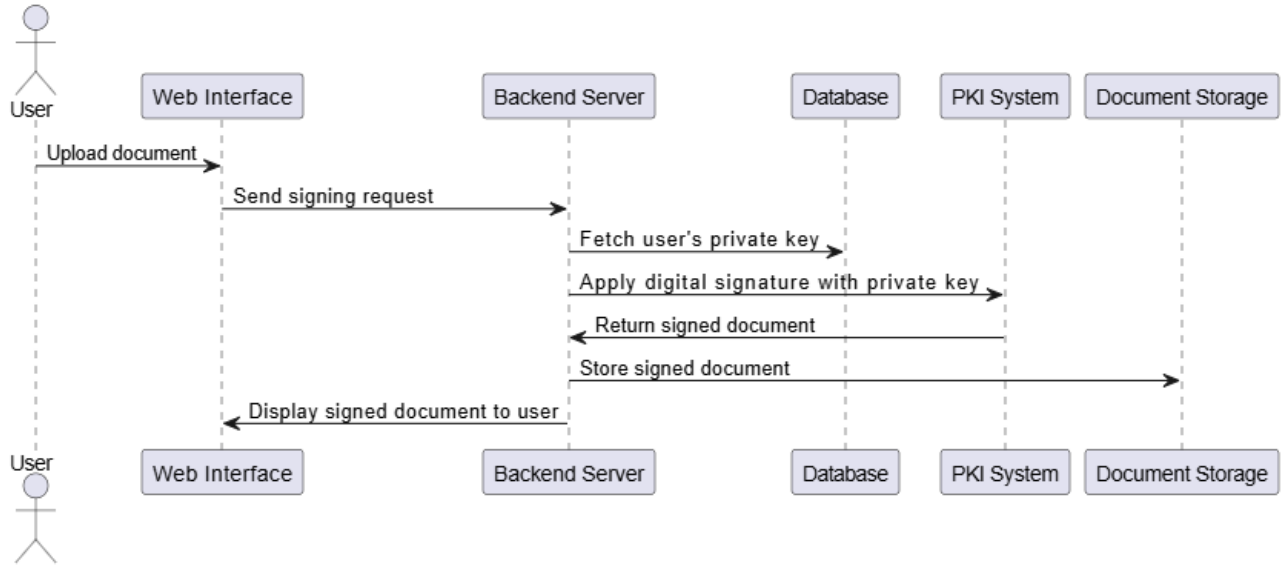

Figure 3.9: Electronic Document Signing

- The backend server retrieves the relevant certificate details.
- The administrator renews or revokes certificates as required.
- The PKI system updates the validity of the certificate.
- The updated information is stored in the database.
- The user is notified of any changes affecting their certificate status.

3.5 CertiSafe Architecture

The CertiSafe platform is designed with a modular, layered architecture that ensures security, scalability, and maintainability. This structure facilitates integration into existing enterprise infrastructures and aligns with best practices in software engineering for Public Key Infrastructure (PKI) systems.

3.5.1 Overview of the System Architecture

The architecture is organized into three main layers, each with distinct responsibilities:

Presentation Layer (Frontend)

- Developed using React.js, this layer provides a responsive and intuitive user interface for both end users and administrators.
- It handles user input, form validation, interaction with the REST API, and the display of results (signatures, certificate details, verification).
- Authentication is managed through JSON Web Tokens (JWT), and all communication is encrypted via HTTPS.

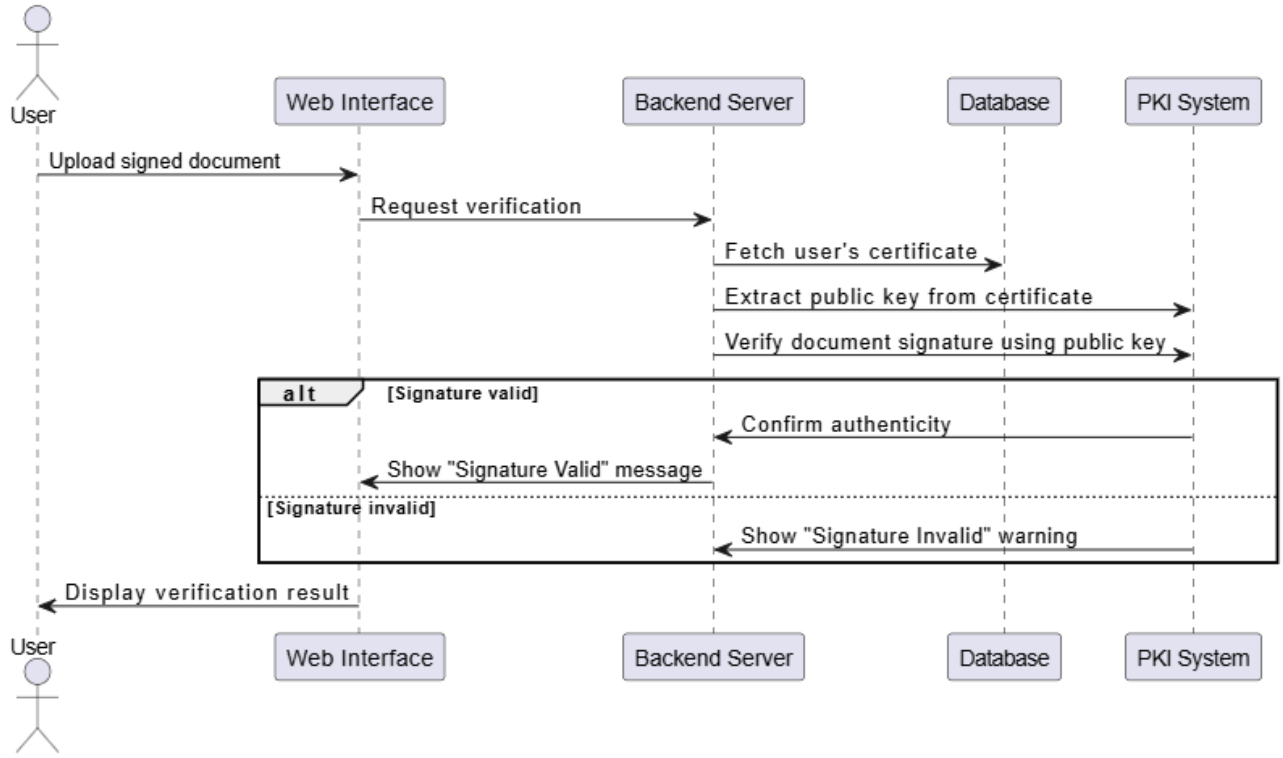

Figure 3.10: Verification of Signed Documents

Application Layer (Backend)

- Implemented in Node.js, this layer contains the business logic, user and certificate management, and cryptographic operation handlers.
- It interfaces directly with OpenSSL via secure system calls to perform certificate generation, digital signing, verification, and timestamping.
- Administrative actions such as revocation and CRL management are also performed at this level.

Data Layer (Database and Cryptographic Assets)

- A MySQL relational database is used to securely store user information, certificate records in PEM format, logs of operations, and revocation data.
- Certificate revocation is handled by generating a Certificate Revocation List (CRL) and storing associated metadata.
- Sensitive data is encrypted at rest and access is role-restricted.

The following diagram summarizes the full system architecture and component interactions of CertiSafe:

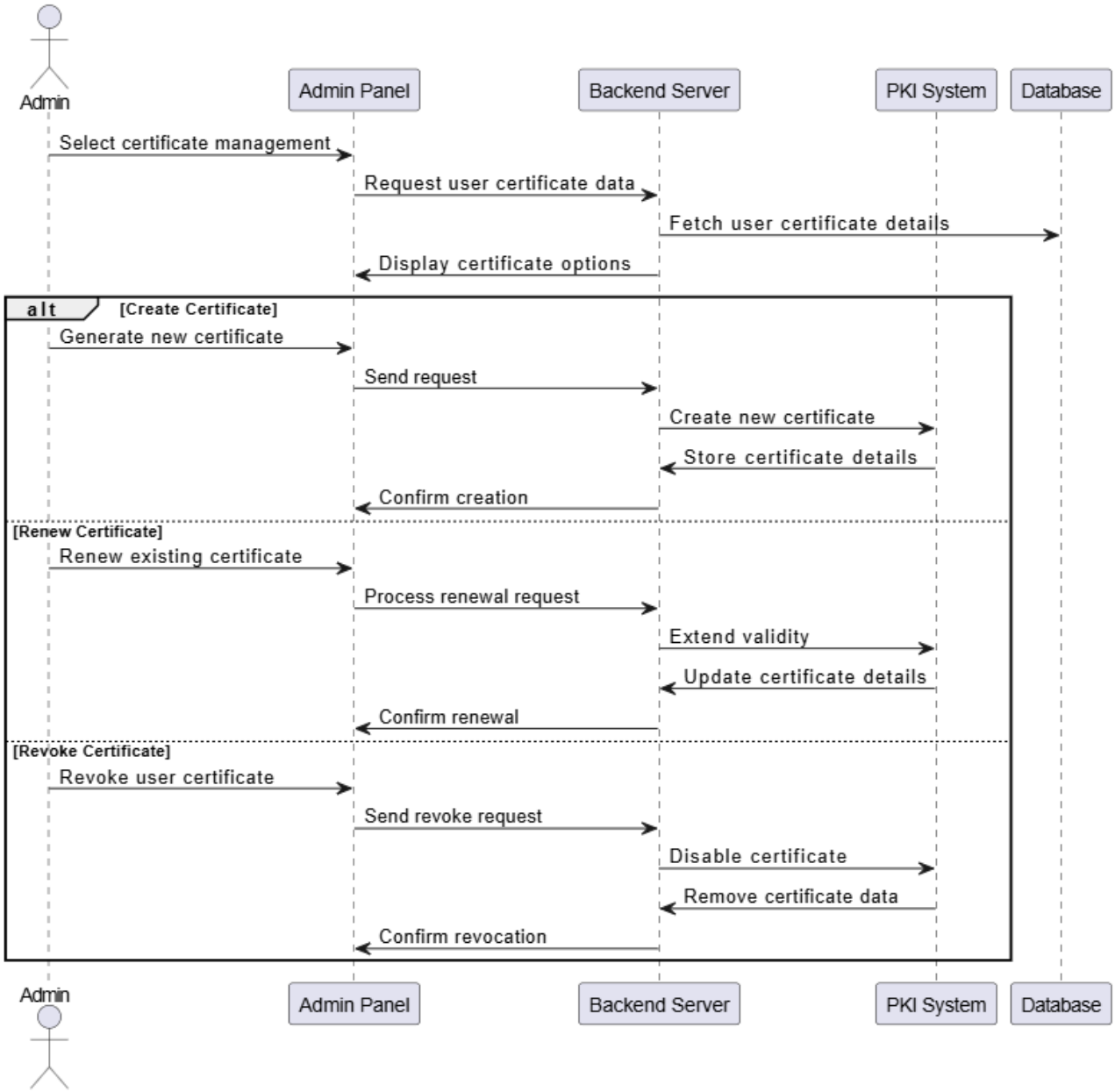

Figure 3.11: Administrator's Certificate Management

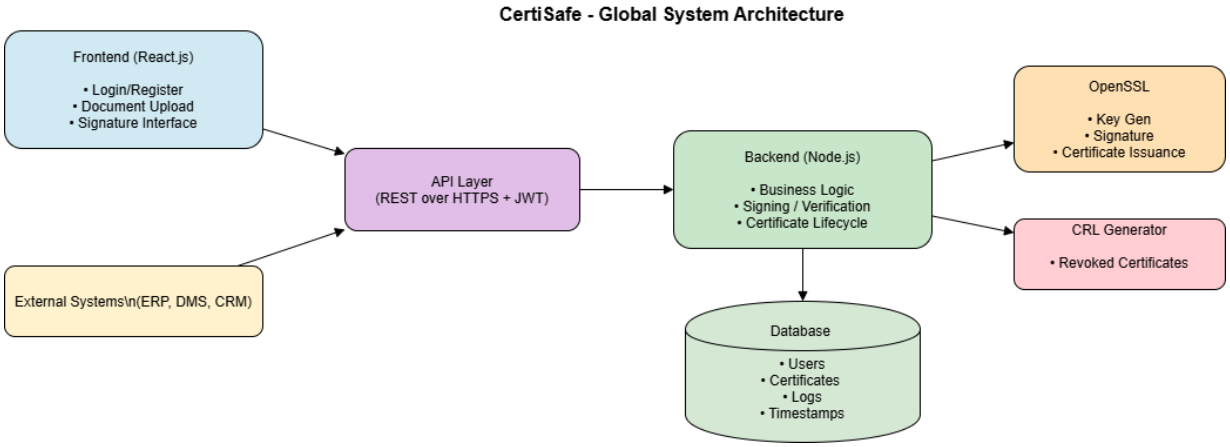

Figure 3.12: CertiSafe System Architecture

3.6 Conclusion

This chapter laid the architectural and design foundations for the development of CertiSafe, a secure and regulation-compliant electronic certification and digital signature platform. The design process began by identifying and documenting both functional and non-functional requirements to ensure that the system would meet legal obligations and technical performance expectations. Through the use of UML modeling techniques—including use case, class, and sequence diagrams—the interactions and responsibilities of each system component were clarified.

The chapter also introduced the layered system architecture, emphasizing the separation of concerns across the frontend, backend, cryptographic services, and data layers. Furthermore, we detailed the integration of secure RESTful APIs to enable enterprise system interoperability and future scalability. These design decisions ensure that the platform is not only technically robust and secure, but also aligned with Algerian e-signature laws and PKI standards.

By following a structured software development lifecycle, this chapter provided a comprehensive blueprint for implementation. It also laid the groundwork for the future evolution of CertiSafe, with design provisions for features like mobile access, offline operation, and advanced audit and maintenance capabilities.

Chapter 4

Experiments and Results

4.1 Introduction

In a context where the digital transformation of administrations and businesses has become a strategic priority in Algeria, securing electronic exchanges represents a major challenge. Although electronic signatures are legally recognized under Law No. 15-04, concrete, reliable, and internationally compliant local solutions remain limited. The project developed in this thesis aims to address this gap by implementing a functional prototype of a web-based platform dedicated to digital certification, electronic signature, and document verification, while also integrating timestamping capabilities. This platform enables user registration, digital certificate management, electronic document signing, and signature verification through a secure pack generated by the system.

Although it currently exists as a web prototype, the project is part of a much broader vision. It represents the first step toward the creation of a nationally accredited and sovereign Certification Authority. In the medium term, several extensions are planned to broaden the use cases and access channels of the solution. These include the development of a cross-platform mobile application, an offline version for environments with limited connectivity, and programming interfaces (APIs) to integrate the system into the internal platforms of businesses and institutions. Advanced features are also under consideration, such as multi-signature support on the same document and the exploration of secure electronic voting possibilities based on a Public Key Infrastructure (PKI) model. This chapter presents the effective implementation of the prototype and the results obtained through various test scenarios that illustrate the system's operation.

4.2 Functional Specifications

4.2.1 Certificate Generation and Management

Certificate generation is a central feature of the CertiSafe platform. It enables the issuance of digital identity credentials to verified users, simulating the function of a real Certification Authority (CA). Once a user completes the registration process and their information is manually validated by the administrator, a certificate is generated by the backend using OpenSSL.

The process is initiated via a secure API call from the admin dashboard. Upon receiving the request, the backend performs the following operations:

- Generates an RSA key pair for the user
- Constructs a Certificate Signing Request (CSR) using the user's identity data
- Signs the CSR using the platform's root private key
- Outputs a valid X.509 certificate

The generated certificate is encoded in **PEM (Privacy-Enhanced Mail)** format. This format represents cryptographic data in Base64 encoding with clear delimiters such as `-----BEGIN CERTIFICATE-----`. In the CertiSafe platform, each generated PEM certificate is stored directly as a plain text string in the system's relational database and is linked to the user's profile. This approach simplifies certificate access and management, and ensures portability across different systems.

When needed, the backend uses OpenSSL commands (e.g., `openssl x509 -text -noout`) to extract the certificate's metadata such as subject name, issuer, validity period, and serial number. This metadata is then displayed to users or administrators without exposing any private key.

The table below presents the content of an actual certificate generated by the system, along with a description of each field:

Table 4.1: Detailed Contents of a Generated Certificate

Field	**Value (Example)**	**Description**
Country (C)	DZ	Country code (Algeria) of the certificate holder
State or Province (ST)	Skikda	Administrative region or state of the user
Locality (L)	Skikda	City or locality of the certificate owner
Organization (O)	CertiSafe	Name of the user's affiliated organization or company
Common Name (CN)	alaeddine boussaid	Full name of the certificate holder (used as their public identity)
Email Address (E)	boussaid.ala21@gmail.com	Professional email address associated with the user
User Identifier (UID)	11	Internal identifier used by the system to uniquely identify the user
Issuer CN	CertiSafe Authority	Name of the Certification Authority that issued the certificate
Validity Start Date	10 June 2025 at 19:07:16	Date and time when the certificate becomes valid
Validity End Date	10 June 2026 at 19:07:16	Expiration date and time of the certificate
Serial Number	481a4c369e21c1b393eeb61607471609e073a7bb	Unique hexadecimal identifier assigned to the certificate by the CA
Signature Algorithm	sha256WithRSAEncryption	Algorithm used to digitally sign the certificate contents

A visual example of a certificate generated by the system is presented below:

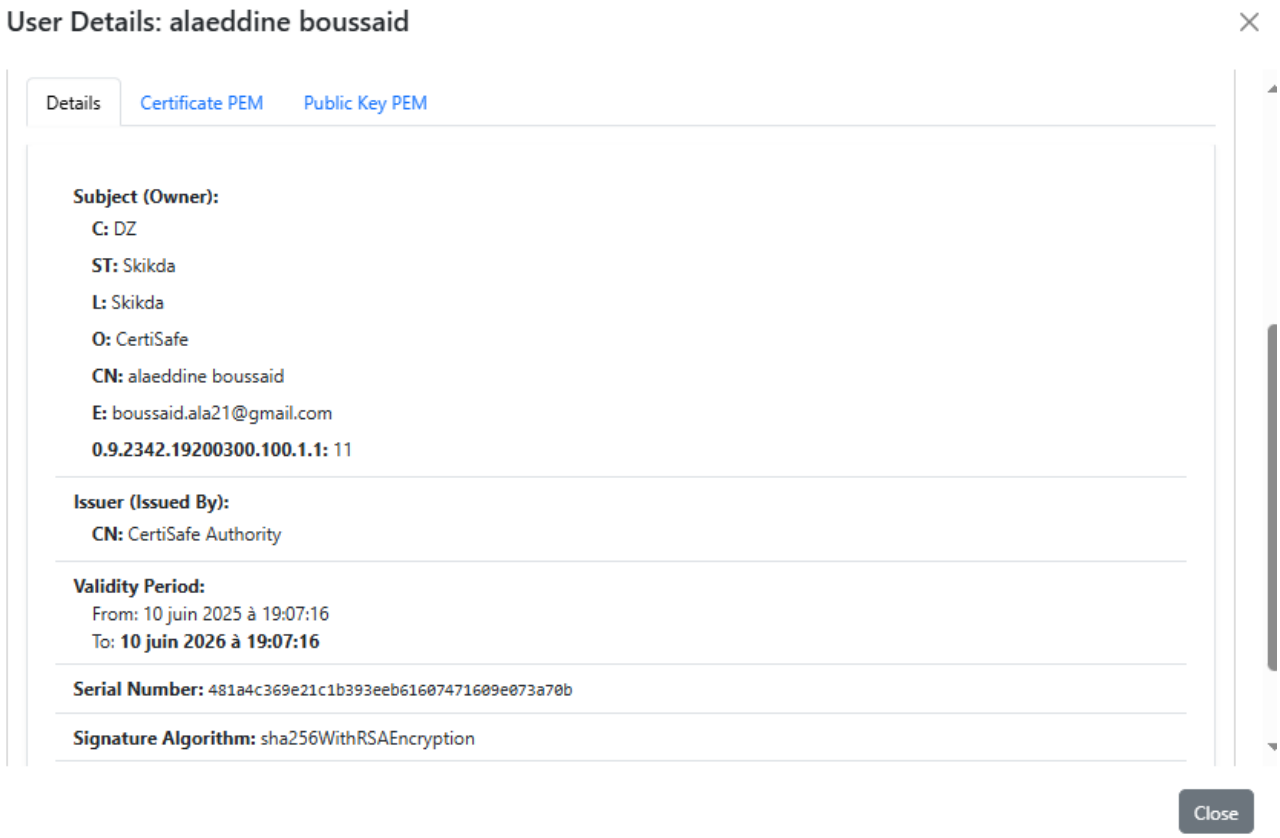

Figure 4.1: Screenshot of a certificate generated by CertiSafe

The following image shows the raw PEM content of the same certificate stored in the system:

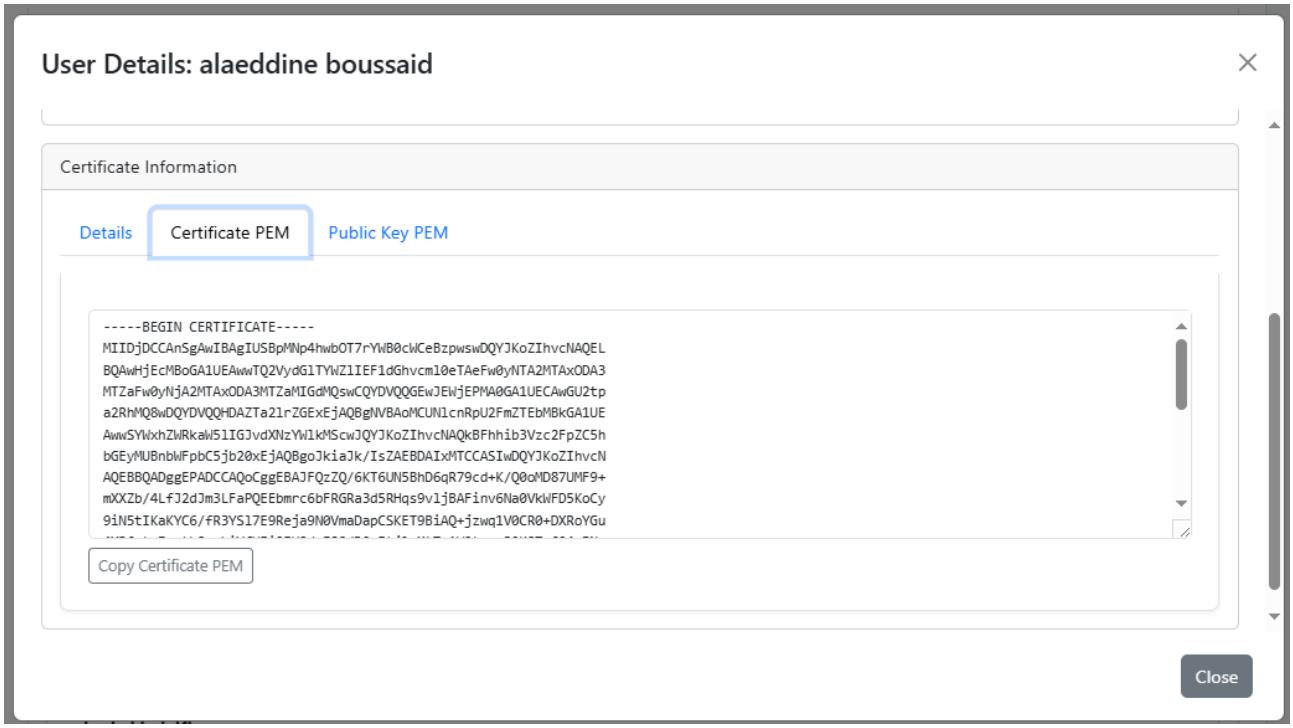

Figure 4.2: PEM-formatted certificate content

File Format Considerations:

Digital certificates can be stored in different formats depending on their usage context. The table below provides a comparison between common certificate file formats, including PEM, DER, and PFX. CertiSafe uses the PEM format for its readability, compatibility with OpenSSL, and ease of storage in databases.

Table 4.2: Comparison of Certificate File Formats

Format	**Extension(s)**	**Encoding Format**	**Content**	**Typical Usage**
PEM	.pem, .crt, .cer, .key	Base64 (ASCII) with headers	Certificate, private key, or chain	Widely used in Linux servers, Apache, OpenSSL
DER	.der, .cer	Binary (ASN.1)	Certificate only	Used in Java-based platforms (e.g., keystore), Windows
CRT	.crt	Usually PEM, sometimes DER	Certificate only	Generic certificate format, used on various systems
CER	.cer	PEM or DER (depends on OS)	Certificate only	Interchangeable with .crt; used on Windows
KEY	.key	PEM (Base64) or DER	Private key only	Stores unencrypted or encrypted private keys
PFX / PKCS#12	.pfx, .p12	Binary	Certificate + private key (often encrypted)	Used in Windows, IIS, and for certificate import/export

This comparison highlights the interoperability needs across platforms and the reason why PEM was selected in CertiSafe's architecture for ease of parsing, portability, and security.

4.2.2 Document Signing

Once a user has obtained a valid digital certificate, the platform allows them to digitally sign electronic documents in a secure and verifiable manner. The signing process begins when the user uploads a document through their personal dashboard. The system accepts multiple file formats, such as PDF, TXT, or DOCX, which are temporarily stored on the server for further processing.

Upon receiving the document, the backend initiates the signing procedure by computing a cryptographic hash of the file using the SHA-256 algorithm. This hash serves as a unique digital fingerprint of the content, ensuring that any modification to the file will result in a completely different hash.

Using the private key associated with the user's certificate, the backend signs the generated hash. This operation is carried out using OpenSSL commands invoked through the backend server (Node.js), resulting in a binary or base64-encoded digital signature. The signature is saved as a separate file .

To facilitate portability and future verification, the system automatically packages all relevant components into a compressed ZIP archive. This archive contains:

- The original, unaltered document
- The digital signature file
- The user's certificate

This ZIP file can be downloaded by the user and later used for signature verification either on the platform or by external parties. It ensures that all cryptographic and contextual materials are bundled together and remain consistent.

The following figure illustrates the full document signing process, from upload to ZIP archive generation:

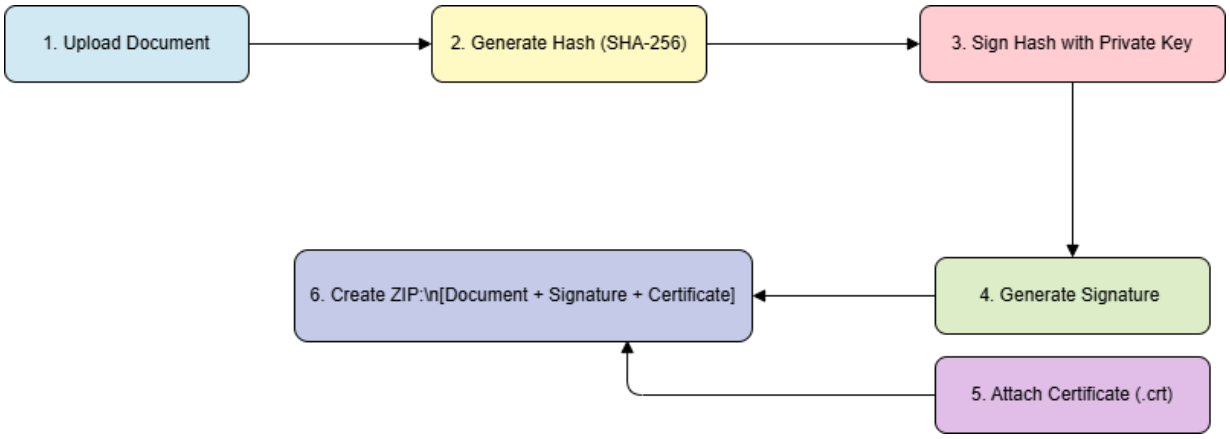

Figure 4.3: Illustration of the document signing process

The following figure shows an example of a ZIP archive generated by the platform after a successful signing process:

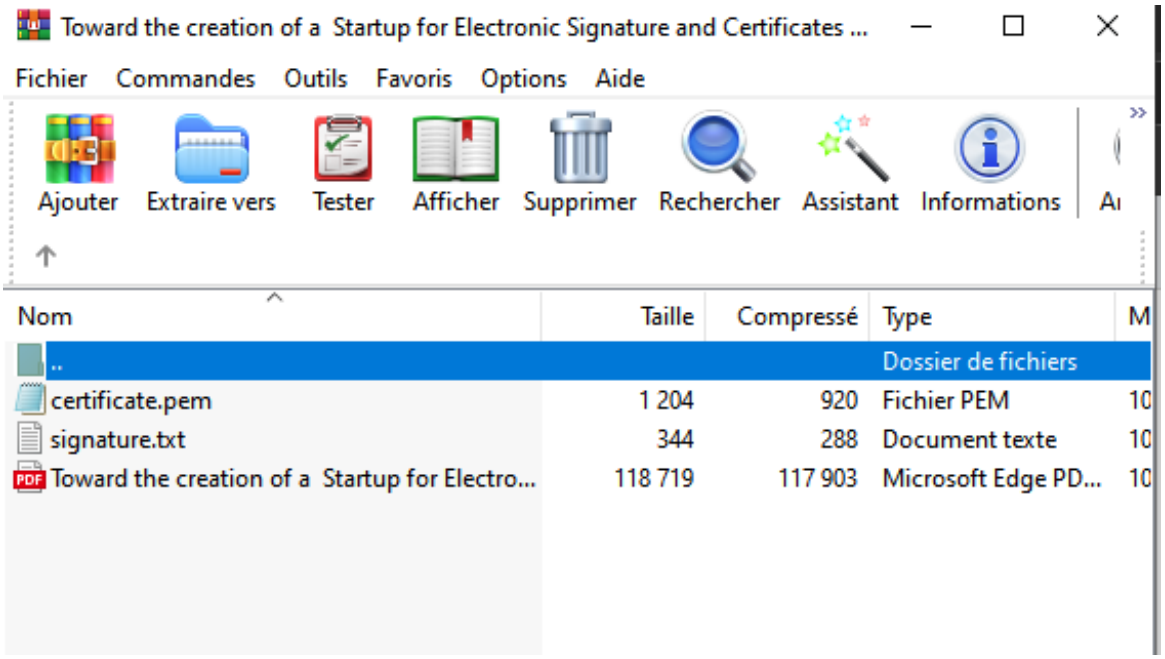

Figure 4.4: ZIP archive containing the original document, signature, and certificate

4.2.3 Signature Verification

The CertiSafe platform provides a dedicated module to verify the authenticity and integrity of digitally signed documents. This feature allows users and third parties to ensure that a document was signed by a legitimate certificate holder and has not been tampered with since it was signed.

The verification process begins when the user uploads the following three components via the verification interface:

- The original document
- The digital signature file
- The associated certificate

Once uploaded, the backend performs a series of cryptographic operations:

1. **Document hash calculation:** The system computes the hash of the uploaded document using the SHA-256 algorithm, producing a unique fingerprint of the file.
2. **Public key extraction:** The backend extracts the public key from the provided certificate using OpenSSL.
3. **Signature decryption:** The system decrypts the '.sig' file using the extracted public key, recovering the original hash value that was signed.
4. **Hash comparison:** The decrypted hash is compared with the newly computed hash of the uploaded document.

If the two hash values match, the system confirms that the signature is valid and that the document has not been altered. Otherwise, the signature is considered invalid. Additionally, the certificate's validity is checked (revocation, expiration) before completing the verification.

The figure below illustrates the full technical verification process as implemented in CertiSafe:

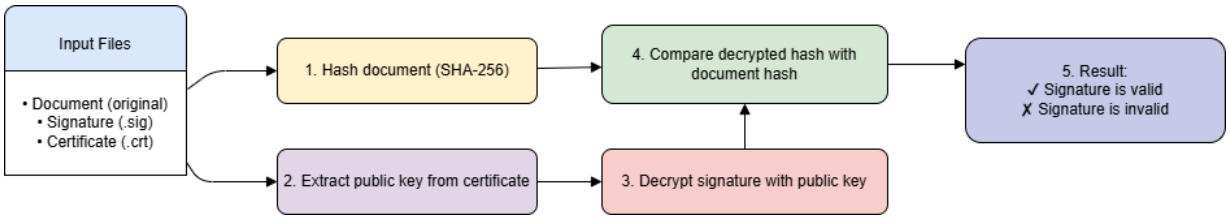

Figure 4.5: Technical diagram of signature verification workflow

4.2.4 Timestamping

In digital signature systems, timestamping is a critical component that ensures the temporal integrity of signed documents. It provides cryptographic proof that a document was signed at a specific moment in time, which is essential for legal non-repudiation and for validating the signature even after the certificate has expired or been revoked.

In the CertiSafe platform, a basic but effective timestamping mechanism is implemented. When a user signs a document, the backend system appends a timestamp to the signing operation. This timestamp is automatically generated using the server's system clock and formatted according to the ISO 8601 standard (e.g., `2025-06-07T14:32:10Z`). It is either embedded within the signature metadata or included as a separate file inside the ZIP archive returned to the user after signing.

This local timestamp acts as a trusted reference in the context of internal systems or low-risk environments. However, for stronger guarantees and legal traceability, the architecture of CertiSafe is designed to support future integration with a Time Stamping Authority (TSA). In that scenario, the document's hash would be submitted to an external, trusted TSA, which would return a TimeStamp Token (TST) digitally signed by the authority. That TST would then be stored with the signed document, offering higher legal admissibility.

The following diagram illustrates the current and future timestamping workflow supported by CertiSafe:

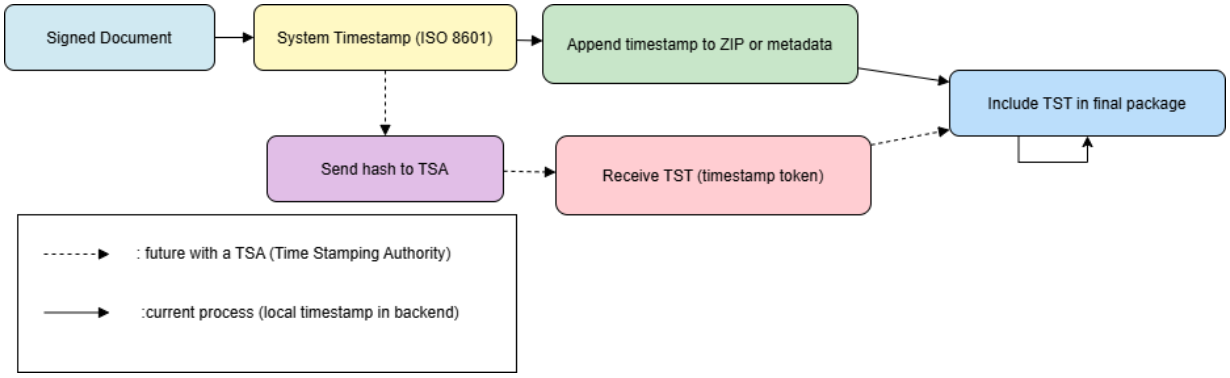

Figure 4.6: Timestamping workflow: current and future with TSA

The next figure shows an example of a timestamp included in a signed document generated by CertiSafe. The time is recorded at the moment of signature and helps verify when the action was performed:

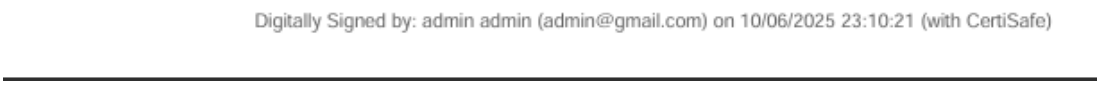

Figure 4.7: Example of timestamp attached to a signed document

4.3 Non-Functional Specifications

In addition to functional requirements, the CertiSafe platform adheres to several non-functional specifications to ensure quality, sustainability, and reliability. These specifications aim to guarantee a high level of user trust, operational efficiency, and readiness for enterprise deployment.

- **Security:** All communications between frontend and backend are encrypted via HTTPS. JSON Web Tokens (JWT) are used for session authentication, and sensitive data (such as passwords and certificates) are encrypted at rest using secure cryptographic algorithms. File uploads are restricted by size and format to prevent injection attacks. Backend interactions with OpenSSL are sandboxed and validated.
- **Scalability:** The system is designed with separation of concerns, allowing independent scaling of the frontend (React), backend (Node.js), and database (MySQL). This architecture supports horizontal scaling

in cloud environments.

- **Maintainability:** The source code follows modular design principles with clean separation between business logic, data access, and presentation layers. The project uses environment variables and configuration files to simplify deployment and future upgrades.
- **Availability:** The system provides basic fault tolerance by handling backend errors gracefully. In production deployment, further measures like load balancers and database replication will ensure high availability.
- **Performance:** Signing and verification operations are optimized through asynchronous I/O, and heavy cryptographic tasks are offloaded to subprocesses. Preliminary tests show response times under 1 second for signing standard documents.
- **Portability:** The application is built with cross-platform compatibility and uses widely supported technologies (JavaScript, SQL, OpenSSL). Dockerization is planned to ensure easy deployment across cloud and on-premises infrastructures.
- **Usability and Accessibility:** The web interface is designed to be intuitive, with clear user guidance and form validation. Accessibility practices such as contrast, keyboard navigation, and responsive layout are partially respected and will be improved in future iterations.
- **Auditability and Logging:** All sensitive operations such as signing, certificate issuance, and revocation are logged in a secure database table. Logs include timestamps, user IDs, IP addresses, and operation types to ensure traceability and enable future audit processes.
- **Legal Compliance:** Although CertiSafe is currently a prototype, its architecture anticipates alignment with Algerian law n°15-04 on electronic signature and international PKI standards .

4.4 Maintenance and Audit Strategy

To ensure long-term sustainability, traceability, and system reliability, CertiSafe includes components dedicated to system maintenance and audit tracking. These features are essential to guarantee trust in digital certification services, detect anomalies, and support compliance with regulatory frameworks.

4.4.1 Maintenance Plan

The maintenance strategy is centered around the proactive monitoring and stability of core services:

- **Uptime Monitoring:** A background process checks service availability and certificate generation workflows.
- **CRL Health Check:** The Certificate Revocation List (CRL) is verified periodically to ensure revoked certificates are properly flagged and inaccessible.
- **Error Logging and Alerts:** System errors and failed cryptographic operations are logged and trigger alert notifications for manual review.
- **System Update Readiness:** The modular architecture supports hot-swapping and updating OpenSSL versions or database schemas without system downtime.

These tasks are managed by a dedicated maintenance interface accessible to authorized maintenance agents.

4.4.2 Audit Functionality

The platform integrates an audit trail mechanism that tracks administrative actions and cryptographic operations:

- **User Activity Logs:** Signing actions, certificate requests, and downloads are timestamped and associated with user identities.
- **Administrative Actions:** All certificate creation, revocation, and renewal activities performed by the admin are logged and stored securely.
- **Log Exporting:** The system allows secure export of logs for compliance or incident investigation purposes.
- **Tamper Resistance:** Audit logs are signed and protected from modification using digital hashing to ensure integrity.

This dual-layer approach combining maintenance and audit ensures operational transparency and trustworthiness of CertiSafe as a future Certification Authority.

4.5 Scalable Functional Roadmap

4.5.1 Multi-signature Support

CertiSafe will support multi-signature mechanisms to enable collaborative validation of a single document by multiple parties. Two types of multi-signature flows will be implemented:

- **Parallel Signatures:** All authorized signers can sign the document independently and simultaneously. Each signature is individually verifiable and attached without modifying prior signatures.
- **Sequential Signatures:** Signatures are added in a predefined order. Each signer must wait for the previous signature to be applied before signing. This model reflects approval workflows or hierarchical validation.

4.5.2 Offline Version

To accommodate users operating in environments with limited or no internet connectivity, CertiSafe plans to offer an offline desktop version of the platform. This local client will provide essential functionalities, such as signing and verifying documents, without needing continuous access to the central server.

A key feature of the offline mode will be the use of **RFID smart cards** that securely store the user's digital certificate and private key. The signing process will operate as follows:

- The user uploads a document into the offline application.
- The user presents their RFID card to a connected card reader .
- The system extracts the certificate and private key from the card (in a secure session), then performs a cryptographic signature on the document.

- The signed document is saved locally and can later be synchronized with the main CertiSafe server when connectivity is available.

This solution offers strong security guarantees by isolating the private key from the local file system and preventing unauthorized duplication. Technically, the application will interface with standard PC/SC-compatible smart card readers and use middleware libraries to communicate with the card.

All offline actions (signature, verification) will be logged locally and queued for synchronization. Once the device is reconnected to the network, the logs and signed documents will be pushed to the central server over a secure TLS channel.

This approach ensures security, portability, and legal compliance even in offline scenarios, which is crucial for rural deployments, mobile agents, or defense-related use cases.

4.5.3 Secure Electronic Voting

As part of its future functional roadmap, CertiSafe aims to support secure electronic voting mechanisms, leveraging its PKI-based infrastructure. This feature is not intended for national or governmental elections but rather for internal or private use cases, such as board member elections within companies, trade unions, associations, or organizational decision-making processes that require authenticated and auditable voting.

The voting module would integrate the following components:

- **Voter Authentication:** Each voter must authenticate using their digital certificate before accessing the ballot, ensuring the identity and eligibility of participants.
- **Ballot Signing:** Once a vote is cast, it is digitally signed with the voter's private key to guarantee authenticity, integrity, and non-repudiation.
- **Anonymity and Integrity:** To preserve voter privacy, the system may adopt cryptographic anonymization methods such as blind signatures or homomorphic encryption, ensuring the separation of voter identity from ballot content.
- **Tamper-Proof Logs:** All operations will be recorded in immutable logs, using hash chaining techniques to detect any alteration and provide a reliable audit trail.

Although not yet implemented, this module highlights CertiSafe's scalability and its potential to deliver trust-based services beyond traditional document certification, particularly in structured private governance scenarios.

4.6 Conclusion

This chapter demonstrated the practical realization of the CertiSafe platform through the implementation of its core functionalities. Starting from user registration and progressing to certificate issuance, document signing, signature verification, and timestamping, the prototype validates the technical feasibility of the envisioned system. Special attention was given to how backend services—built with Node.js and OpenSSL—interact with the frontend and database layers to simulate a functional Certification Authority (CA) environment.

The chapter also highlighted security mechanisms such as digital certificate storage in PEM format, JWT-based API authentication, and the inclusion of Certificate Revocation Lists (CRLs). Visual evidence from

system interfaces and test scenarios reinforced the platform's reliability and usability. In addition to current capabilities, the chapter outlined planned enhancements, including multi-signature workflows, offline certificate storage via RFID cards, and mobile application support.

Collectively, these results affirm that CertiSafe is not only technically sound as a prototype but also scalable for future enterprise integration and legal compliance in the Algerian context. The implementation sets the stage for transitioning from a proof of concept to a production-grade platform capable of supporting a national-level certification service.

General Conclusion

This thesis presented the design and development of a web-based prototype for an electronic certification and digital signature system tailored to the Algerian context. The growing need for secure, legally recognized, and efficient digital transactions has made Public Key Infrastructure (PKI) a critical enabler of trust in digital services. While the legal framework for electronic signatures exists in Algeria, there remains a lack of locally developed solutions that comply with both international standards and national regulations.

The proposed platform, CertiSafe, addresses this gap by offering a comprehensive system that supports certificate issuance, digital document signing, verification, and secure timestamping. Throughout this work, we detailed the system architecture, the technologies involved, and the interactions between components. Implementation efforts focused on developing an intuitive and secure platform using technologies such as React.js, Node.js, OpenSSL, and MySQL. Key functionalities such as role-based access, certificate lifecycle management, signature packaging, and revocation mechanisms were successfully demonstrated.

In addition to the current prototype, the project envisions future extensions including an offline mode (using RFID smart cards), mobile application support, multiple-signature workflows, and secure API integration for enterprise systems. These enhancements aim to make the solution more versatile and applicable to real-world corporate environments.

The work achieved in this thesis lays a solid technical and strategic foundation for the creation of a certified national Certification Authority (CA) and a legal digital trust service provider. Beyond the technical implementation, this project proposes a scalable, locally governed alternative to foreign digital signature solutions, contributing to Algeria's digital sovereignty and cybersecurity goals.

Bibliography

[1] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson, 8th edition, 2020.

[2] Agence de Gestion et de Coopération entre l'Algérie et l'Union Européenne. AGCE - Agence de Gestion et de Coopération entre l'Algérie et l'Union Européenne. `http://www.agce.dz`. Accessed December 30, 2024.

[3] Autorité Économique de certification Électronique. `https://aece.dz/`, 2024. Consulté le 30 décembre 2024.

[4] OpenSSL Project. *OpenSSL Documentation*, 2024.

[5] Node.js Foundation. *Node.js Crypto Module Documentation*, 2024.

[6] Node.js Foundation. *Node.js Documentation*, 2024.

[7] Express.js Team. *Express.js Guide*, 2024.

[8] Auth0. Introduction to json web tokens, 2024.

[9] Oracle Corporation. *MySQL 8.0 Reference Manual*, 2024.

[10] Meta Platforms Inc. *React Documentation*, 2024.

[11] J. R. Vacca. *Public key infrastructure: Building trusted applications*. Syngress, 2012.

[12] Official Journal of the People's Democratic Republic of Algeria. Official journal of the people's democratic republic of algeria, 2015, 2015. Accessed December 24, 2024.

[13] Symeon (Simos) Xenitellis and OpenCA Team. *The Open-Source PKI Book: A Guide to PKIs and Open-Source Implementations*. OpenCA Project, 2004.

[14] Ivan Ristić. *Bulletproof TLS and PKI*. Feisty Duck, second edition edition, 2019.

[15] Ivan Ristić. *OpenSSL Cookbook*. Feisty Duck, third edition edition, 2023.

[16] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 2005.

[17] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Object-Oriented Modeling and Design with UML*. Pearson Education, 2004.

[18] DevsData. Deno vs node.js: Which one is better in 2025? `https://devsdata.com/deno-vs-node-which-one-is-better-in-2025/`, 2025. Accessed: 2025-05-17.

[19] Centre for Digital Public Infrastructure. Signatures numériques et pki. `https://docs.cdpi.dev/fr/notes-techniques/infrastructure-de-confiance/signatures-numeriques-et-pki`, 2024. Accessed: 2025-05-17.

[20] ANTIC. Livre blanc - opérations sur les certificats. https://camgovca.cm/images/downloads/articles/LIVRE%20BLANC%20SUR%20LA%20PKI.pdf. Accessed: 2025-05-17.

[21] ARPCE. Certification électronique et sécurité de l'information. https://www.arpce.dz/fr/file/m7e5z1, 2025. Accessed: 2025-05-17.

[22] Scribd. Infrastructure pki. https://fr.scribd.com/presentation/641208833/Untitled. Accessed: 2025-05-17.

[23] Demaeter. Pki - public key infrastructure. https://www.demaeter.fr/concepts/pki.html, 2023. Accessed: 2025-05-17.

[24] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.