Advanced Programming of Cryptographic Methods

Project Report

TITOLO

Emanuele Civini, Alessia Pivotto

Academic year 2024/2025

Contents

1 Introduction

This project implements a Certificate Authority (CA) system that manages digital certificates for secure communications. A CA is a trusted entity that issues digital certificates to verify the identity of users, servers, or devices in a network.

The system provides a complete solution for certificate management, including issuing new certificates, checking their validity, renewing them before expiration, and revoking them when necessary.

The implementation focuses on security best practices by using Hardware Security Modules (HSMs) to protect the CA's private key and implementing proper verification procedures.

The system provides user-friendly interfaces through both API endpoints for developers and a web interface for end users. Additionally, it ensures system reliability by handling multiple requests efficiently while maintaining data consistency.

1.1 Key Features

The Certificate Authority system supports the complete certificate lifecycle. For certificate issuance, the system processes certificate requests, verifies the requester's identity, and issues signed digital certificates. It provides certificate validation by checking if certificates are valid, not expired, and haven't been revoked.

The system allows users to renew their certificates before they expire and handles certificate revocation for compromised or no longer needed certificates, maintaining a Certificate Revocation List (CRL). Identity verification ensures that certificate requesters own the private key and control the email address associated with their request.

1.2 System Architecture

The system consists of several integrated components. The backend server is a Go-based API that handles all certificate operations and communicates with the database and HSM. A Next.js frontend application provides an easy-to-use web interface for certificate management.

Data persistence is handled by MongoDB storage for certificate records, user commitments, and system data. Security operations are performed by a Hardware Security Module (HSM) that stores the CA's private key and performs cryptographic operations. An automated email service supports identity verification during certificate requests.

1.3 Report Structure

This report provides a comprehensive overview of the Certificate Authority system implementation. Chapter 2 presents the requirements analysis and system specifications. Chapter 3 details the system design and architecture. Chapter 4 covers implementation details and technology choices, while Chapter 5 discusses security considerations and best practices. Finally, Chapter 6 provides deployment guidance and usage instructions.

Each chapter builds upon the previous one, providing a complete understanding of the Certificate Authority system from concept to deployment.[**?**]

2 Requirements

This chapter defines the comprehensive set of requirements that guide the design and implementation of our Certificate Authority system. The requirements are categorized into functional requirements, which specify what the system must do, and security requirements, which define how the system must protect itself and its users from various threats and vulnerabilities.

2.1 Functional Requirements

The functional requirements define the core capabilities and operations that the Certificate Authority must provide to fulfill its role in a Public Key Infrastructure. Each requirement is justified based on standard PKI practices and the specific needs of secure certificate management.

2.1.1 FR1: Certificate Issuance

Requirement: The CA must generate and issue digital certificates after verifying the identity of the certificate requester.

Justification: Certificate issuance is the primary function of any Certificate Authority. This process involves receiving Certificate Signing Requests (CSRs) from entities, validating the authenticity of the request, and creating signed digital certificates that bind public keys to verified identities. Without this capability, the CA cannot fulfill its fundamental role as a trusted third party in the PKI ecosystem. The verification step is crucial to prevent unauthorized certificate issuance, which could compromise the entire trust infrastructure.

Implementation Details: The system must support standard CSR formats, perform comprehensive validation of request parameters, and generate certificates compliant with X.509 standards.

2.1.2 FR2: Certificate Revocation

Requirement: Users can request the revocation of any of their certificates at any moment, for example when they suspect a private key leakage.

Justification: Certificate revocation is essential for maintaining the integrity of the PKI when certificates become compromised, are no longer needed, or when the associated private keys are suspected to be leaked. Without a revocation mechanism, compromised certificates would remain valid until their natural expiration, potentially allowing unauthorized access to resources. The ability for certificate holders to initiate revocation ensures that they maintain control over their digital identities and can respond quickly to security incidents.

Implementation Details: The system must provide authenticated revocation requests, requiring proof of private key ownership to prevent malicious revocation attacks by unauthorized parties.

2.1.3 FR3: Certificate Revocation List (CRL) Accessibility

Requirement: The CA must maintain a Certificate Revocation List to track revoked certificates. Anyone must be able to check the status of any certificate by querying the CRL to verify whether a specific certificate has been revoked or not.

Justification: The CRL serves as the authoritative source for certificate revocation status information. Public accessibility is crucial because certificate validation is performed by various parties across the network, not just by the certificate holders themselves. Without a publicly accessible CRL, relying parties would have no way to determine if a certificate has been revoked, potentially accepting invalid certificates and compromising security. This requirement aligns with RFC 5280 standards for certificate validation.

Implementation Details: The CRL must be regularly updated, digitally signed by the CA, and made available through standard protocols such as HTTP or LDAP.

2.1.4 FR4: Identity Verification

Requirement: The CA must implement processes to verify the authenticity of the certificate applicant's email and the ownership of the proper private key. This process simplifies the identity verification typically required by real-world CAs.

Justification: Identity verification is fundamental to establishing trust in the PKI. Without proper verification, malicious actors could obtain certificates for identities they do not control, leading to impersonation attacks and breakdown of trust. Email verification ensures that the applicant controls the email address being certified, while private key ownership verification ensures that only the legitimate key holder can obtain or manage certificates for that key pair. This dual verification approach provides a reasonable balance between security and usability for a simplified CA implementation.

Implementation Details: Email verification can be implemented through challenge-response mechanisms, while private key ownership can be verified through cryptographic challenges requiring digital signatures.

2.1.5 FR5: Public Key Publishing

Requirement: The CA must publish the issued certificates and its own public key so that users can validate certificate authenticity.

Justification: Public availability of certificates and the CA's public key is essential for the certificate validation process. Relying parties need access to both the certificates they want to validate and the CA's public key to verify the certificate signatures. Without this public accessibility, the certificates would be useless for their intended purpose of enabling secure communications. The CA's public key serves as the trust anchor for the entire PKI, and its availability is crucial for establishing the chain of trust.

Implementation Details: The system must provide standard interfaces for certificate retrieval and maintain a publicly accessible repository of issued certificates and CA public key information.

2.1.6 FR6: Certificate Renewal

Requirement: The CA must allow renewal of certificates before they expire, ensuring continuity of trust. This process must include identity verification challenges and verification that the certificate has not been revoked.

Justification: Certificate renewal is critical for maintaining continuous service availability and trust relationships. Certificates have limited lifespans for security reasons, but the underlying trust relationships often need to persist beyond individual certificate validity periods. Renewal allows for the seamless transition from expiring certificates to new ones without disrupting established trust relationships. The verification requirements ensure that only legitimate certificate holders can renew their certificates and that revoked certificates cannot be renewed, maintaining security integrity.

Implementation Details: The renewal process must verify the current certificate's validity status, perform identity verification equivalent to initial issuance, and ensure proper overlap periods to prevent service interruptions.

2.1.7 FR7: Cryptographic Algorithm Support

Requirement: The CA must support RSA and ECDSA cryptographic algorithms for certificate operations.

Justification: Supporting multiple cryptographic algorithms ensures compatibility with diverse client requirements and provides flexibility for different security and performance needs. RSA remains widely deployed in legacy systems and provides well-understood security properties, while ECDSA offers better performance and smaller key sizes for equivalent security levels. Supporting both algorithms ensures that the CA can serve a broad range of clients while accommodating both current and emerging cryptographic preferences.

Implementation Details: The system must handle key generation, signature creation, and verification for both RSA (minimum 2048-bit keys) and ECDSA (P-256 curve) algorithms.

2.2 Security Requirements

Security requirements define the protective measures and security properties that the Certificate Authority must maintain to ensure the integrity, confidentiality, and availability of the PKI services. These requirements address various threat models and attack vectors that could compromise the CA's operations.

2.2.1 SR1: Certificate Authenticity

Requirement: A certificate that contains a valid and correct signature from the CA must be considered authentic and trustworthy.

Justification: Certificate authenticity forms the foundation of trust in the PKI system. The CA's digital signature on a certificate serves as the cryptographic proof that the certificate was issued by the legitimate CA and has not been tampered with. This requirement ensures that relying parties can confidently trust certificates that pass signature verification, enabling secure communications. Without reliable authenticity verification, the entire PKI system would be vulnerable to certificate forgery and impersonation attacks.

Implementation Details: The system must use cryptographically strong signature algorithms, maintain secure signing procedures, and ensure that signature verification mechanisms are readily available to all relying parties.

2.2.2 SR2: Certificate Validity Verification

Requirement: A certificate that is expired or appears in the CRL must be considered invalid, even if it contains an authentic signature from the CA.

Justification: Certificate validity encompasses more than just authenticity; it also includes temporal validity and revocation status. Expired certificates should not be trusted because they may represent outdated information or compromised keys that have exceeded their intended lifespan. Similarly, revoked certificates must be rejected regardless of their authentic signatures because they have been explicitly invalidated due to compromise or other security concerns. This requirement prevents the acceptance of certificates that may pose security risks despite being technically authentic.

Implementation Details: All certificate validation processes must include expiration date checking and CRL consultation, with clear rejection of certificates that fail either test.

2.2.3 SR3: Secure Key Management

Requirement: The private key of the CA must be stored securely using Hardware Security Modules (HSMs) and protected against tampering, unauthorized access, and extraction.

Justification: The CA's private key is the most critical security asset in the entire PKI system. If this key is compromised, attackers could forge certificates for any identity, completely undermining the trust infrastructure. Traditional software-based key storage is vulnerable to various attacks, including malware, insider threats, and system compromises. HSMs provide tamper-resistant hardware protection that ensures the private key never exists in plaintext outside the secure hardware boundary and that all cryptographic operations are performed within the protected environment.

Implementation Details: The system must integrate with cloud-based HSM services, ensure all signing operations occur within the HSM, implement secure authentication for HSM access, and maintain audit trails of all key usage.

2.2.4 SR4: Authentication and Authorization

Requirement: All certificate management operations must be properly authenticated and authorized to prevent unauthorized access and malicious operations.

Justification: Without proper authentication and authorization controls, malicious actors could perform unauthorized certificate operations such as requesting certificates for identities they don't control, revoking legitimate certificates, or accessing sensitive certificate information. These controls ensure that only authorized parties can perform specific operations and that all actions are traceable to authenticated identities.

Implementation Details: The system must implement multi-factor authentication where appropriate, role-based access controls, and comprehensive audit logging of all administrative actions.

2.2.5 SR5: Data Integrity and Confidentiality

Requirement: All certificate data, configuration information, and audit logs must be protected against unauthorized modification and inappropriate disclosure.

Justification: Data integrity ensures that certificate information remains accurate and trustworthy throughout its lifecycle. Unauthorized modifications could lead to invalid certificates being accepted or valid certificates being rejected. Confidentiality protections prevent sensitive information from being disclosed to unauthorized parties, which could facilitate attacks or privacy violations.

Implementation Details: The system must implement database encryption, secure communication protocols, access controls, and integrity checking mechanisms for all stored data.

2.2.6 SR6: Availability and Resilience

Requirement: The CA services must maintain high availability and resilience against various failure modes and attack scenarios.

Justification: CA unavailability can disrupt certificate validation processes across the entire PKI, potentially preventing legitimate users from accessing services or causing applications to reject valid certificates. High availability ensures that critical PKI services remain accessible when needed, maintaining trust and usability of the infrastructure.

Implementation Details: The system must implement redundancy, failover mechanisms, backup procedures, and monitoring to detect and respond to availability threats.

These requirements collectively define a comprehensive security posture that addresses the primary threats and vulnerabilities associated with Certificate Authority operations while ensuring the functional capabilities necessary for effective PKI services.

3 Technical Details

This chapter provides comprehensive technical details about the Certificate Authority implementation, covering the system architecture, implementation technologies, and detailed code structure. The information presented here enables a thorough understanding of the technical decisions and implementation approaches used in this project.

3.1 Architecture

The Certificate Authority system follows a microservices-based architecture designed for scalability, security, and maintainability. The system is composed of several distinct modules that work together to provide comprehensive PKI services.

3.1.1 High-Level Architecture

The system architecture consists of four primary components:

1. **CA Backend Server**: A Go-based REST API server that handles all certificate operations, including issuance, revocation, and validation.
2. **Web User Interface**: A Next.js-based frontend application that provides an intuitive interface for certificate management operations.
3. **Database Layer**: MongoDB database for persistent storage of certificates, revocation lists, and audit information.
4. **Hardware Security Module (HSM)**: Cloud-based HSM integration for secure cryptographic operations and private key protection.

3.1.2 Component Interaction

The architecture follows a layered approach with clear separation of concerns:

- **Presentation Layer**: The web UI communicates with the CA backend through RESTful API calls, handling user interactions and data visualization.
- **Business Logic Layer**: The CA server implements all PKI business logic, including certificate lifecycle management, validation rules, and identity verification processes.
- **Data Persistence Layer**: MongoDB provides reliable storage for certificates, CRL entries, and operational logs.
- **Security Layer**: HSM integration ensures that all cryptographic operations involving the CA's private key are performed within secure hardware boundaries.

3.1.3 Security Architecture

The security architecture implements defense-in-depth principles:

- **HSM Integration**: All CA private key operations are performed within the HSM, ensuring the key never exists in plaintext outside secure hardware.
- **Authentication Mechanisms**: Multi-layered authentication including email verification and private key ownership proof.

3.1.4 Deployment Architecture

The system is containerized using Docker for consistent deployment across different environments:

- Each major component runs in its own Docker container
- Docker Compose orchestrates multi-container deployment
- Environment-specific configurations through environment variables
- Persistent storage volumes for database data

3.2 Implementation

This section details the technical implementation aspects, including programming languages, frameworks, libraries, and key implementation decisions.

3.2.1 Backend Implementation

Programming Language: Go (Golang) 1.21+

Justification: Go was chosen for the backend implementation due to its performance characteristics, strong standard library for cryptographic operations, robust concurrency support, and extensive ecosystem for web service development.

Key Libraries and Frameworks:

- **Go net/http**: HTTP router and URL matcher for building REST APIs with support for middleware and route variables.
- **MongoDB Driver**: Official Go driver for MongoDB providing high-performance database operations with built-in connection pooling.
- **AWS SDK for Go**: Integration with AWS KMS for HSM operations, providing secure key management and cryptographic operations.
- **Go Standard Crypto Libraries**: Extensive use of crypto/x509, crypto/rsa, crypto/ecdsa, and crypto/rand for certificate operations and cryptographic functions.
- **Email**: Email verification and notification capabilities using Go's Resend package.

3.2.2 Frontend Implementation

Framework: Next.js 15 with React 19

Justification: Next.js provides server-side rendering capabilities, excellent developer experience, and robust production optimizations. React's component-based architecture enables maintainable and reusable UI components.

Key Technologies:

- **Tailwind CSS 4.0**: Utility-first CSS framework for rapid UI development with consistent design systems and responsive layouts.
- **Web Crypto API**: Browser-native cryptographic operations for client-side key generation, signing, and certificate validation without requiring external libraries.
- **Next.js App Router**: Modern routing system with nested layouts and server components for optimal performance.
- **React Hooks**: State management and lifecycle handling using modern React patterns including useState, useEffect, and custom hooks.

3.2.3 Database Implementation

Database: MongoDB 7.0+

Justification: MongoDB's document-oriented structure is well-suited for storing certificate data with varying fields and complex nested structures. Its built-in indexing and query capabilities support efficient certificate lookups and CRL operations.

Key Features Utilized:

- **Document Storage**: Flexible schema for certificate metadata, CRL entries, and audit logs
- **Indexing**: Optimized queries for certificate serial numbers, email addresses, and revocation status

3.2.4 HSM Integration

HSM Provider: AWS KMS (Key Management Service)

Implementation Approach:

- **Customer Master Keys (CMK)**: Dedicated CMK for CA operations with hardware-level protection
- **AWS SDK Integration**: Programmatic access to KMS operations through official AWS SDK
- **Signing Operations**: All certificate signing performed via KMS API calls
- **Key Rotation**: Support for automatic key rotation policies
- **Audit Logging**: Comprehensive logging of all HSM operations through AWS CloudTrail

3.2.5 Containerization and Deployment

Container Technology: Docker with Docker Compose

Container Configuration:

- **Multi-stage Builds**: Optimized Docker images with separate build and runtime stages
- **Environment Variables**: Externalized configuration for different deployment environments
- **Health Checks**: Container health monitoring and automatic restart capabilities
- **Volume Management**: Persistent storage for database data and certificates

3.3 Code Structure

This section provides a detailed description of the project's code organization, explaining the purpose and functionality of each component.

3.3.1 Project Root Structure

The project follows a monorepo structure with clear separation between different services:

```
advanced-programming-of-cryptographic-methods/
|-- ca/                 # Backend CA server
|-- ui/                 # Frontend web application
|-- dev-certs/          # Development certificates
|-- report/             # Project documentation
|-- docker-compose.yml  # Container orchestration
|-- mongod.conf         # MongoDB configuration
`-- README.md           # Project overview
```

3.3.2 Backend Code Structure (ca/)

The backend follows Go's standard project layout with clear separation of concerns:

Main Application (cmd/)

- **cmd/server/main.go**: Application entry point that initializes all services (HSM, database, email) and starts the HTTP server. Handles dependency injection and graceful shutdown procedures.

Internal Packages (internal/)

- **internal/config/config.go**: Centralized configuration management using environment variables. Handles HSM and server parameters.
- **internal/db/**: Database abstraction layer
  - **db.go**: Database connection management and initialization
  - **models.go**: Data models for certificates, CRL entries, and logs
- **internal/email/email.go**: Email service implementation for identity verification and notifications.
- **internal/hsm/hsm.go**: Hardware Security Module integration layer. Provides abstraction over AWS KMS operations including key creation, signing, and key management.
- **internal/server/**: HTTP server implementation
  - **server.go**: HTTP server initialization, middleware configuration, and route setup
  - **handlers/**: Request handlers organized by functionality
    - **health.go**: Health check endpoints for monitoring
    - **info.go**: CA information and public key endpoints
    - **utils.go**: Common handler utilities and response formatting
    - **certificate/**: Certificate-specific handlers
      - **handler.go**: Certificate issuance, revocation, and validation endpoints
      - **repository.go**: Database operations for certificate management

Configuration Files

- **go.mod/go.sum**: Go module dependencies and version management
- **Dockerfile**: Multi-stage Docker build configuration for optimized container images

3.3.3 Frontend Code Structure (ui/)

The frontend follows Next.js 13+ app directory structure with modern React patterns:

Application Core (app/)

- **layout.js**: Root layout component with global styles, navigation, and shared UI elements
- **page.js**: Home page component with CA overview and navigation links
- **globals.css**: Global CSS styles including Tailwind CSS imports and custom component styles
- **favicon.ico**: Application favicon

Feature Pages (app/)

- **sign/page.js**: Certificate signing interface with CSR upload, key generation, and email verification workflows
- **certs/page.js**: Certificate viewer and revocation interface with ASN.1 parsing and cryptographic operations
- **crl/page.js**: Certificate Revocation List viewer with pagination and search capabilities
- **commit/page.js**: Certificate commitment and validation interface

Utility Modules (utils/)

- **crypto.js**: Client-side cryptographic utilities including key generation, signing, and certificate validation using Web Crypto API
- **pemUtils.js**: PEM format parsing and manipulation utilities for certificate and key handling

Configuration Files

- **package.json**: Node.js dependencies and build scripts
- **next.config.mjs**: Next.js configuration including build optimizations and deployment settings
- **postcss.config.mjs**: PostCSS configuration for Tailwind CSS processing
- **jsconfig.json**: JavaScript/TypeScript configuration for IDE support
- **Dockerfile**: Frontend container build configuration

Static Assets (public/)

- ***.svg**: UI icons and graphics including Next.js branding and custom icons

3.3.4 Development and Deployment Support

Development Certificates (dev-certs/)

- **root.pem**: CA root certificate for the trust chain. It is generated during the initial setup.

Container Orchestration

- **docker-compose.yml**: Multi-container application orchestration including MongoDB, local KMS, CA server, and frontend services
- **mongod.conf**: MongoDB server configuration including security settings and connection parameters

3.3.5 Code Quality and Maintainability

The codebase implements several best practices for maintainability and reliability:

- **Separation of Concerns**: Clear boundaries between presentation, business logic, and data layers
- **Error Handling**: Comprehensive error handling with appropriate logging and user feedback
- **Configuration Management**: Externalized configuration through environment variables

- **Security Practices**: Input validation, secure defaults, and defense-in-depth implementation
- **Documentation**: Inline code comments and comprehensive API documentation
- **Testing Support**: Structure conducive to unit testing and integration testing

This code structure provides a solid foundation for a production-ready Certificate Authority system while maintaining flexibility for future enhancements and scaling requirements.

4 Security Considerations

This chapter critically examines the security architecture of the implemented Certificate Authority system, focusing on identifying fundamental vulnerabilities and proposing concrete pathways for improvement.

4.1 Missing certificate extensions

The certificate generation process lacks Authority Key Identifier and Subject Key Identifier extensions, which may cause problems for certificate chain validation. The absence of Certificate Policies extensions prevents policy-based validation required in production environments.

This issue can be addressed by fully supporting RFC 5280 [1] compliance, including configurable policy engines, name constraints support, and complete certificate path validation logic.

4.2 Side channel attacks and information leakage TODO: FIX ISSUE OR DESCRIPTION

While Go's crypto library provides constant-time operations for most of its primitives, the CA relies on database queries and nonce lookup mechanisms that can leak information through timing patterns.

Database queries reveal certificate existence through timing patterns, while nonce lookup mechanisms use standard hash operations with predictable performance characteristics. Error message patterns provide another disclosure channel, allowing attackers to distinguish between different failure conditions and map internal system architecture.

Resolution requires implementing constant-time operations, normalizing response timing through artificial delays, and sanitizing error responses to provide uniform feedback regardless of underlying failure conditions.

4.3 HSM as a single point of failure

The current architecture relies on a single HSM instance for all cryptographic operations. This design creates a single point of failure for the entire CA, making it inoperable if the HSM becomes unavailable due to hardware failure, network connectivity issues, or maintenance requirements.

Furthermore, the HSM represents a single point of compromise where an attacker who gains access to the HSM could potentially sign malicious certificates, extract private key material, or manipulate the cryptographic operations.

This issue can be addressed by requiring HSM redundancy through clustered HSM deployments, threshold cryptography that distributes signing operations across multiple HSMs, automated failover mechanisms that maintain service continuity during HSM maintenance or failure, and comprehensive HSM monitoring with real-time availability checking.

4.4 Simplified revocation mechanism

The revocation mechanism lacks reason codes that specify the cause of the revocation events, limiting incident response capabilities. Because of the same reason, the system provides no mechanism for CA-initiated revocation, creating problems when certificates need revocation due to external threats but certificate holders are unavailable.

This problem can be addressed by extending the revocation mechanism to include reason codes that specify the cause of the revocation events, and by implementing a mechanism for the CA to initiate revocation when necessary. This would allow the CA to revoke certificates in response to external threats, even when certificate holders are unavailable. To prevent censorship, the reason codes should be included in the OCSP responses, allowing the relying parties to understand the reason for the revocation without trusting blindly the CA.

4.5 Lack of post-quantum cryptography support

The current implementation lacks post-quantum cryptography support, creating a significant long-term security vulnerability as quantum computing technology advances. The system relies entirely on ECDSA-SHA256, which will become vulnerable as quantum computers becomes more powerful.

While we initially thought about adding support for post-quantum algorithms, we were limited by the ones supported by *local-hsm*, an emulated version of AWS KMS that provides a simplified HSM capabilities for development purposes. Unlike production AWS KMS, *local-hsm* does not support post-quantum algorithms such as CRYSTALS-Dilithium or CRYSTALS-Kyber, preventing the implementation of quantum-resistant cryptography within our development environment. because of this, the CA cannot generate hybrid certificates that provide both current ECDSA compatibility and quantum resistance.

To address this issue, it is enough to use a production HSM that supports post-quantum algorithms, such as AWS KMS, but unfortunately there is not free plan to use it.

4.6 Use of custom JSON-based OCSP responses

The implementation uses custom JSON-based response formats due to simplicity and speed of development for this MVP. These responses try to emulate the ones in RFC 6960 [2], implementing a subset of all specified cases due to time constraints. Because of this, a significant future improvement would be to use ASN.1 OCSP responses, making this implementation more compatible with the already existing PKI ecosystem. Considering that the core functionalities of OCSP are already implemented, this change should not require signficant effort.

5 Known Limitations

Despite the project aimed at creating a certificate authority capable of issuing X.509 certificates able to verify the identity of the certificate holder, we had to make some compromises due to time constraints, high cost of services and practicality of the implementation.

TODO: Add limitation that Resend can only send email to a single email address.

5.1 Limited identity verification

The current implementation uses a simplified identity verification process that only checks email ownership and private key possession, thus lacking a proper identity validation processes.

5.2 Simplified certificates usage

The system implements basic X.509 certificate profiles without the full range of extensions and policies, for example:

- Authority Information Access (AIA) extensions
- Subject Alternative Names (SAN) for multiple identities

5.3 Lack of revocation cause

While the CA implements a revocation mechanism to revoke certificates, it does not include reason codes that specify the cause of the revocation events. As a consequence, the system itself is not able to revoke certificate on behalf of the certificate holder.

5.4 System scalability

The system is implemented as a single-instance application without cosnidering scaling solutions, which would improve availability and overall security. In particular, a signficant improvement would be to use a clustered HSM deployment, which would allow the CA to continue operating even if one of the HSMs fails or is under maintenance, and to use threshold cryptography, which would allow the CA to distribute the signing operations across multiple HSMs.

5.5 Standards compliance

While the system implements main operations for managing X.509 standards, it lacks full compliance with advanced PKI standards, in particular tries replicate RFC 5280 and RFC 6960 in a simplified way.

5.6 Single-email address limitation

As domain providers always require a paid plan to create custom email addresses, the system is limited to using a single email address for sending emails, which is the default one provided by

Resend.com, and is able to send challenge emails only to the address that is associated to the Resend.com account.

6 Instructions for Installation and Execution

This chapter provides instructions for installing, configuring, and executing the Certificate Authority. These instructions are designed to enable readers to successfully deploy and test the system in their own environments.

6.1 Software Prerequisites

The following software must be installed on the target system:

Required Software:

- **Operating System**: Linux, macOS, or Windows with WSL2 (tested on Linux)
- **Docker**: Version 28.0 or higher (tested with 28.3)
- **Docker Compose**: Version 2.37 or higher (tested with 2.37.3)
- **Git**: Version 2

6.2 Installation process

6.2.1 Environment Setup

```
# Verify Docker and Docker Compose are installed
$ docker --version
Docker version 28.3.0, build 38b7060a21

$ docker compose version
Docker Compose version 2.37.3

# Clone the repository
$ git clone https://github.com/ecivini/advanced-programming-of-cryptographic-methods.git

# Move to the project directory
$ cd advanced-programming-of-cryptographic-methods
```

6.2.2 Configuration

Create a `.env` file in the project root directory with the following configuration: TODO: Add resend api for shared email address account

```
# MongoDB Configuration
MONGO_USERNAME=camanager
MONGO_PASSWORD=912k83hb0slW)s2

# AWS/HSM Configuration
AWS_REGION=eu-west-1
```

```
AWS_ACCESS_KEY_ID=111122223333
AWS_SECRET_ACCESS_KEY=aaaabbbb11111

# Email Service Configuration (Resend.com)
RESEND_API_KEY=your_resend_api_key_here
RESEND_FROM=onboarding@resend.dev
```

As the CA is using an emulated version of AWS KMS, the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` can be set to the proposed test values which emulates real ones. In addition, as the free plan of Resend.com allows sending emails only to a single email address, and there are no free domain providers, RESEND_FROM must be set to the default email address provided by Resend.com.

6.2.3 Starting the CA

Initial Startup:

```
# Build and start all services
docker compose up --build
```

During the first startup, the system automatically:

1. Creates a new root ECDSA key pair in the HSM using curve P256
2. Generates the root certificate for the CA
3. Initializes the database schema
4. Sets up necessary indexes

After that, and in any execution, the system will start the following services:

- **MongoDB**: Database on port 27017
- **Local KMS**: HSM on port 8080
- **Backend**: CA server on port 5000
- **Frontend**: Web interface on port 3000

6.2.4 Service Verification

Health Check Endpoints:

```
# Backend health check
curl http://localhost:5000/v1/health

# Frontend accessibility
curl http://localhost:3000
```

TODO: Consider adding documentation for the endpoints, or a guide for using the frontend.

Bibliography

[1] D. Cooper et al. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, Network Working Group, 2008.

[2] S. Santesson et al. X.509 internet public key infrastructure online certificate status protocol - ocsp. RFC 6960, Internet Engineering Task Force, 2013.