UNIVERSITÀ
DI TRENTO

Advanced Programming of Cryptographic Methods

Project Report

TITOLO

Emanuele Civini, Alessia Pivotto

Academic year 2024/2025

Contents

1 Introduction **3**
1.1 Key Features . . . 3
1.2 System Architecture . . . 3
1.3 Report Structure . . . 3

2 Requirements **4**
2.1 Functional Requirements . . . 4
2.1.1 FR1: Certificate Issuance . . . 4
2.1.2 FR2: Certificate Revocation . . . 4
2.1.3 FR3: Certificate Revocation List (CRL) Accessibility . . . 4
2.1.4 FR4: Identity Verification . . . 5
2.1.5 FR5: Public Key Publishing . . . 5
2.1.6 FR6: Certificate Renewal . . . 5
2.1.7 FR7: Cryptographic Algorithm Support . . . 5
2.2 Security Requirements . . . 6
2.2.1 SR1: Certificate Authenticity . . . 6
2.2.2 SR2: Certificate Validity Verification . . . 6
2.2.3 SR3: Secure Key Management . . . 6
2.2.4 SR4: Authentication and Authorization . . . 7
2.2.5 SR5: Data Integrity and Confidentiality . . . 7
2.2.6 SR6: Availability and Resilience . . . 7

3 System Architecture and Certificate Lifecycle **8**
3.1 Source code structure . . . 8
3.1.1 CA backend server . . . 8
3.1.2 User interface . . . 8
3.1.3 Database . . . 8
3.1.4 Hardware Security Module (AWS KMS) . . . 8
3.2 Security-first architecture . . . 8
3.2.1 Cryptographic isolation . . . 8
3.2.2 Identity verification . . . 9
3.2.3 Signed responses . . . 9
3.2.4 Replay protection . . . 9
3.3 Certificate lifecycle operations . . . 9
3.3.1 Phase 1: identity commitment . . . 9
3.3.2 Phase 2: certificate generation . . . 10
3.3.3 Phase 3: certificate revocation . . . 10
3.3.4 Phase 4: certificate renewal . . . 11
3.4 Component interactions and message flows . . . 12
3.4.1 Frontend-backend communication . . . 12
3.4.2 HSM integration patterns . . . 12

3.4.3 Key creation . . . 12
3.4.4 Certificate signing . . . 12
3.4.5 Root certificate generation . . . 12
3.5 Signed responses . . . 12
3.5.1 Response structure . . . 12
3.5.2 Certificate status responses . . . 13
3.5.3 Revocation list responses . . . 13
3.6 Code Structure and organization . . . 13
3.6.1 Backend service (ca/) . . . 13
3.6.2 Frontend application (ui/) . . . 13
3.6.3 Certificates (dev-certs/) . . . 13
3.6.4 Deployment configuration . . . 13
3.7 Dependencies and libraries . . . 13
3.7.1 Backend dependencies . . . 14
3.7.2 Frontend dependencies . . . 14
3.7.3 Infrastructure dependencies . . . 14
3.7.4 Security and compliance dependencies . . . 14

4 Security Considerations 15
4.1 Missing certificate extensions . . . 15
4.2 Side channel attacks and information leakage TODO: FIX ISSUE OR DESCRIPTION 15
4.3 HSM as a single point of failure . . . 15
4.4 Simplified revocation mechanism . . . 16
4.5 Lack of post-quantum cryptography support . . . 16
4.6 Use of custom JSON-based OCSP responses . . . 16

5 Known Limitations 17
5.1 Limited identity verification . . . 17
5.2 Simplified certificates usage . . . 17
5.3 Lack of revocation cause . . . 17
5.4 System scalability . . . 17
5.5 Standards compliance . . . 17
5.6 Single-email address limitation . . . 17

6 Instructions for Installation and Execution 19
6.1 Software Prerequisites . . . 19
6.2 Installation process . . . 19
6.2.1 Environment Setup . . . 19
6.2.2 Configuration . . . 19
6.2.3 Starting the CA . . . 20
6.2.4 Service Verification . . . 20

Bibliography 20

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 and the possession of a private key. 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 by using an Hardware Security Module (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.

1.1 Key Features

The Certificate Authority supports the complete certificate lifecycle. For certificate issuance, the system processes certificate requests, verifies the requester's identity, and issues signed 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 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 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 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 from idea 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 System Architecture and Certificate Lifecycle

This chapter presents the architectural design decisions and operational flows of the Certificate Authority. The focus is on explaining how the various components interact throughout the complete certificate lifecycle, from initial identity commitment to certificate issuance, renewal, and revocation.

3.1 Source code structure

The system consists of four primary components, each serving a specific role in the certificate lifecycle:

3.1.1 CA backend server

The core service responsible for all cryptographic operations and certificate lifecycle management. Go was selected for its strong and complete cryptographic standard library, excellent performance characteristics, and simplicity of development

3.1.2 User interface

A React-based frontend providing certificate management capabilities. As this UI is intended to be used only for demo purposes, Next.js was chosen for its ease of use, built-in routing, and vast ecosystem of libraries.

3.1.3 Database

Persistent storage for certificates and revocation lists. We decided to use MongoDB for its document-oriented structure, which allows flexible storage and suits well with the certificate and metadata requirements. It also provides indexing capabilities for efficient querying and retrieval of certificate data.

3.1.4 Hardware Security Module (AWS KMS)

We decided to use an HSM, in particulare an emulated version of AWS KMS, to ensure that all CA private key operations occur within FIPS 140-2 Level 3 certified hardware. This design decision eliminates the risk of private key exposure while providing enterprise-grade security and audit capabilities.

3.2 Security-first architecture

The architecture implements security principles through multiple layers of protection.

3.2.1 Cryptographic isolation

The CA's root private key never exists outside the HSM boundary. All signing operations are performed through secure API calls to the HSM, ensuring the key material remains protected even if other system components are compromised.

3.2.2 Identity verification

A two-step identity verification process combines email ownership verification with cryptographic proof of private key possession. This dual verification ensures only legitimate key owners can obtain certificates.

3.2.3 Signed responses

All CA responses are cryptographically signed to ensure integrity and authenticity. This prevents response tampering and provides non-repudiation for all CA operations.

3.2.4 Replay protection

Nonce-based replay protection mechanisms prevent malicious reuse of previously captured requests, protecting against replay attacks.

3.3 Certificate lifecycle operations

This section details the complete certificate lifecycle, explaining the message flows, cryptographic operations, and component interactions for each phase.

3.3.1 Phase 1: identity commitment

The certificate issuance process begins with an identity commitment phase that establishes and verifies user identity through a secure multi-step protocol.

Client-side operations

The process initiates when a user accesses the certificate request interface. The client application, running in the browser, generates a new cryptographic key pair using the Web Crypto API. The user selects their preferred key type (ECDSA P-256, RSA 2048-bit, or RSA 4096-bit) based on their security requirements and intended certificate usage.

The client extracts the public key from the generated pair and formats it as a PEM-encoded certificate signing request. Along with the user's email address and selected key type, this information forms the identity commitment request.

CA backend processing

When the CA backend receives the commitment request at the '/commit-identity' endpoint, it performs comprehensive validation:

The system validates the email address format using standard RFC 5322 compliance checking. The provided key type must match one of the supported algorithms (ECDSA, RSA_2048, RSA_4096). The public key undergoes cryptographic validation to ensure it is mathematically valid and properly formatted.

Upon successful validation, the CA generates a unique challenge string and creates an identity commitment record in the database. This record contains the user's email, public key in DER format, selected key type, challenge string, and an expiration timestamp. The CA reserves a unique serial number for the eventual certificate, ensuring no conflicts occur during concurrent requests.

Email verification

The identity verification process leverages email ownership as proof of identity. The CA's email service sends the challenge string to the user's provided email address using a secure email delivery service.

This email-based verification serves dual purposes: it confirms the user has access to the claimed email address and provides the challenge string needed for the subsequent cryptographic proof phase. The challenge has a limited validity period (typically 24 hours) to prevent indefinite pending commitments.

3.3.2 Phase 2: certificate generation

The certificate generation phase requires cryptographic proof of private key ownership before the CA will issue a certificate.

Challenge response

After receiving the challenge via email, the user returns to the certificate interface and provides the challenge string. The client application retrieves the challenge and prompts the user to sign it using their private key.

The signing operation occurs entirely within the browser using the Web Crypto API. The client signs the raw challenge bytes using the private key corresponding to the public key submitted during commitment. This creates a digital signature that proves the user possesses the private key without exposing it.

CA verification and certificate issuance

The CA receives the challenge response containing the challenge string and the cryptographic signature. The system performs several critical verification steps:

First, the CA retrieves the identity commitment record using the provided challenge. It verifies the commitment hasn't expired and hasn't been previously used, preventing replay attacks and ensuring temporal validity.

Next, the CA performs signature verification using the public key from the commitment record. This cryptographic verification proves the requester possesses the corresponding private key. The verification process uses the appropriate algorithm (ECDSA or RSA) based on the committed key type.

Upon successful verification, the CA initiates certificate generation. The system constructs an X.509 certificate containing the user's public key, email address as the subject, and the reserved serial number. The certificate includes standard extensions such as key usage, basic constraints, and subject alternative names.

HSM-Based certificate signing

The certificate signing operation represents the most security-critical component of the entire system. The CA never signs certificates using local private keys; instead, all signing operations occur within the Hardware Security Module.

The CA formats the certificate structure and sends it to the HSM through the AWS KMS API. The HSM performs the cryptographic signing operation using the CA's root private key, which never leaves the secure hardware boundary. This approach ensures the highest level of security for the CA's signing operations.

The HSM returns the signature, which the CA combines with the certificate data to produce the final signed X.509 certificate. The completed certificate is returned to the client in PEM format, ready for immediate use.

3.3.3 Phase 3: certificate revocation

Certificate revocation enables certificate owners to invalidate their certificates before their natural expiration, essential for handling key compromise or changing security requirements.

Revocation request authentication

The revocation process begins when a certificate owner accesses the revocation interface and provides their certificate's serial number. The system requires cryptographic proof that the requester owns the certificate's corresponding private key.

The client constructs a revocation message in the format "Revoke: <serial number>" where the serial number matches the certificate being revoked. The user signs this message using their private key, creating a revocation signature that proves their authority to revoke the certificate.

CA revocation processing

The CA receives the revocation request containing the serial number and signature. The system retrieves the original identity commitment record associated with the serial number to obtain the corresponding public key.

The CA verifies the revocation signature against the constructed message "Revoke: <serial number>" using the certificate's public key. This verification ensures only the legitimate certificate owner can initiate revocation.

Upon successful verification, the CA updates the certificate record in the database, marking it as revoked with the current timestamp. The revoked certificate is immediately added to the Certificate Revocation List (CRL), making the revocation status publicly available through the CRL endpoint.

3.3.4 Phase 4: certificate renewal

Certificate renewal allows users to extend their certificate validity period without generating new key pairs, maintaining continuity while refreshing the certificate's temporal validity.

Renewal request process

TODO: Risk of replay attack? request can be replicated The renewal process requires the certificate owner to prove continued possession of the private key. The client constructs a renewal message in the format "Renew: <serial number>" and signs it using the certificate's private key.

This signature proves the requester still controls the private key associated with the certificate, ensuring only legitimate certificate owners can renew their certificates.

CA renewal validation and processing

The CA performs several validation steps before processing renewal requests:

The system verifies the certificate exists and hasn't been revoked. It confirms the certificate hasn't expired, as expired certificates cannot be renewed. The CA validates the renewal signature against the message "Renew: <serial number>" using the certificate's public key.

Upon successful validation, the CA extends the certificate's validity period (typically by one year) and generates a new certificate with the updated expiration date. The renewed certificate maintains the same serial number and subject information while reflecting the extended validity period.

HSM integration in renewal

Similar to initial certificate issuance, the renewal process involves HSM-based signing operations. The CA constructs the renewed certificate structure and submits it to the HSM for signing using the root private key.

This ensures renewed certificates maintain the same level of cryptographic integrity as originally issued certificates, with all signing operations occurring within the secure hardware boundary.

3.4 Component interactions and message flows

This section provides detailed analysis of how system components communicate during certificate operations, emphasizing the role of the HSM and signed responses.

3.4.1 Frontend-backend communication

TODO: Add diagrams?

The web interface communicates with the CA backend through RESTful API endpoints, with each operation following specific message flow patterns.

The process starts with the identity commitment flow, where the frontend sends a POST request to '/v1/identity' containing the user's email, public key in PEM format, and selected key type. The backend responds with an HTTP 200 status upon successful commitment creation, triggering the email verification process.

Then, once the use has received the challenge via email and wrote it into the 'Sign' section together with the associate private key, the frontend computes the required signature and submits a POST request to '/v1/certificate' with the challenge string and signature. The backend validates the signature and returns the signed certificate in PEM format.

When a user wants to revoke a certificate, the frontend sends a POST request to '/v1/certificate/revoke' containing the revocation signature. The backend validates the signature and confirms revocation through a JSON response.

TODO: Maybe move serial parameter into payload? Similarly, when a user wants to renew a certificate, the frontend sends a POST requests to '/v1/certificate/renew/serial' containing renewal signatures. Successful renewal returns both confirmation and the renewed certificate.

3.4.2 HSM integration patterns

The HSM integration follows secure communication patterns that ensure private key material never leaves the secure boundary:

3.4.3 Key creation

During system initialization, the CA creates a root key within the HSM using the AWS KMS CreateKey API. The key specification uses ECC NIST P-256 for its security and performance. The HSM generates the key pair entirely within secure hardware, returning only the key identifier.

3.4.4 Certificate signing

For each certificate signing operation, the CA constructs the certificate structure locally and sends it to the HSM via the KMS Sign API. The HSM performs the cryptographic signature using the stored private key and returns only the signature bytes. This pattern ensures the private key never exists outside the HSM boundary.

3.4.5 Root certificate generation

The system generates the CA's root certificate by creating the certificate structure and requesting HSM signing. The resulting self-signed root certificate establishes the trust anchor for all issued certificates.

3.5 Signed responses

To ensure response integrity and prevent tampering, the system implements OCSP-like signed responses for critical operations:

3.5.1 Response structure

Each signed response contains response data, signature algorithm identifier, Base64-encoded signature, and optional signing certificate chain. The response data includes nonces for replay protection and timestamps for freshness validation.

3.5.2 Certificate status responses

When clients query certificate status, the CA constructs a response containing the serial number, status (good/revoked/unknown), update timestamps, and revocation details if applicable. The complete response is signed using the HSM to ensure authenticity.

3.5.3 Revocation list responses

Certificate Revocation List queries return signed responses containing the list of revoked certificates with pagination support. Each response includes update timestamps and replay protection nonces.

3.6 Code Structure and organization

The project follows a monorepo structure with clear separation between different services and components, enabling independent development and deployment while maintaining code organization.

3.6.1 Backend service (ca/)

This directory contains the complete Go-based Certificate Authority server implementation. This module includes the REST API server, HSM integration, database operations, and email services. The structure follows Go's standard project layout with clear separation between commands, internal packages, and configuration. It is organized as follows:

- **cmd/**: Contains the main application entry point and service initialization logic.
- **internal/**: Core business logic modules including configuration, database operations, HSM integration, email services, and HTTP server implementation.
- **handler/**: RESTful API endpoint implementations organized by functionality (e.g., certificate operations, health monitoring).

3.6.2 Frontend application (ui/)

This directory contains the Next.js-based web interface providing user-facing certificate management capabilities. The structure follows Next.js App Router conventions with organized pages, components, and utility functions for cryptographic operations. In particular:

- **app/**: Contains the main application entry point, routing, and layout definitions. The directory structure follows Next.js conventions with pages organized by functionality.
- **components/**: Reusable UI components such as forms, buttons, and modals used across different pages.
- **utils/**: Utility functions for cryptographic operations, API communication, and data formatting.

3.6.3 Certificates (dev-certs/)

This directory, initially empty, contains the root certificate generated by the CA.

3.6.4 Deployment configuration

In the root directory of the project there are files including Docker Compose orchestration, MongoDB configuration, and environment setup scripts for containerized deployment.

3.7 Dependencies and libraries

The system uses external dependencies to provide robust functionality while minimizing security risks and maintaining code quality.

3.7.1 Backend dependencies

Go 1.21+ provides all the packages required for cryptographic operations, thanks to its standard library including crypto/x509, crypto/rsa, crypto/ecdsa, but also net/http for API server functionality. To integrate MongoDB, we decided to use the official MongoDB Go driver (go.mongodb.org/mongo-driver/v2) as it provides secure and comprehensive access to MongoDB features, including document-oriented data storage, indexing, and querying capabilities. For the HSM, we decided to opt for the official AWS SDK Go library (github.com/aws/aws-sdk-go-v2) as it provides programmatic access to AWS KMS for secure key management and signing operations. Lastly, we opted for Resend Go package as it is the official client of the email delivery service we decided to use.

3.7.2 Frontend dependencies

At the core we decided to use Next.js 15 with React 19 as it provides easy routing and lots of ui packages with pre-built compotents, which allowed us to speed up the development process.

TODO: Are we using this? We opted for Web Crypto API (browser native) to handle client-side key generation, signing, and certificate validation without requiring external cryptographic libraries.

3.7.3 Infrastructure dependencies

We opted for Docker and Docker Compose as they allow for easy and consistent deployment environments with isolated service containers. For the database, as we wanted a document-based one, we decided to use MongoDB since we were already familiar with it and it provides document-oriented data storage with indexing, and querying capabilities for scalable management. For the HSM, the only free solution we found was LocalStack KMS, which provides a local HSM emulation for development and testing environments without requiring cloud resources, but emulating the AWS KMS API.

3.7.4 Security and compliance dependencies

All selected dependencies prioritize security and maintain active development with regular security updates. The minimal dependency approach reduces attack surface while leveraging proven, well-maintained libraries from reputable sources.

Cryptographic operations rely primarily on standard library implementations and certified hardware modules rather than third-party cryptographic libraries, ensuring compliance with security standards and reducing implementation risks.

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.