Progress Documentation: Local Certificate Authority Service

Under the guidance of Dr. Balwinder Sodhi

Abhinav Barman [2021EEB1145], Yash Agarwal [2021EEB1225], Gaurav Wani [2021EEB1171]

14th May, 2025

Indian Institute of Technology, Ropar
Department of Electrical Engineering

Abstract

The Local Certificate Authority Service is an open-source web application designed to manage Certificate Signing Requests (CSRs) and issue X.509 certificates within an enterprise Public Key Infrastructure (PKI). This document provides a comprehensive overview of the prototype's implemented features as of May 2025, detailing the Certificate Authority (CA) and Registration Authority (RA) functionalities, system architecture, and limitations. Intended for contributors to the open-source repository, it offers technical insights and examples to facilitate further development.

Contents

1 Introduction

The Local Certificate Authority Service is a prototype web-based platform developed to streamline the management of digital certificates in an enterprise setting. It enables end users to generate and submit CSRs, administrators to approve or reject them, and issues X.509 certificates signed by a CA. Built as a proof-of-concept, the system focuses on core CA and RA functionalities, providing a foundation for secure certificate management.

This document comprehensively documents the project's progress, detailing the implemented features, system architecture, and current limitations. As the project is being open-sourced on GitHub (`https://github.com/imabhi7/Local-Certificate-Authority-Service.git`), this documentation aims to guide future contributors by offering a thorough understanding of the prototype's capabilities. All descriptions are based on the developed code and database schema, ensuring accuracy and relevance.

2 System Overview

The Local Certificate Authority Service is a lightweight PKI solution that supports:

- **User Management**: Registration, authentication, and email verification with OTPs
- **CSR Management**: Generation, submission, approval, and rejection of CSRs
- **Certificate Issuance**: Creation of X.509 v3 certificates for approved CSRs
- **Revocation Tracking**: Basic storage of revoked certificate details
- **User Interfaces**: Dashboards for users (CSR submission, certificate viewing) and admins (CSR approval, system stats)
- **Notifications**: Email alerts for OTPs, CSR approvals, and rejections

The system serves two primary user roles:

- **Clients**: End users who register, submit CSRs, and manage their certificates
- **Admins**: System administrators who approve/reject CSRs and monitor system activity

Built with a Node.js/Express backend, React frontend, and PostgreSQL database, the system integrates OpenSSL for CSR generation and Gmail SMTP for email notifications. It is designed for simplicity, targeting small-scale PKI deployments or educational use.

3 Implemented Features

This section provides an in-depth breakdown of the implemented functionalities, organized by the CA and RA modules, with technical details and example workflows.

3.1 Certificate Authority (CA) Module

The CA module is responsible for processing CSRs, issuing certificates, and tracking revocations. It leverages the `node-forge` library for certificate creation and OpenSSL for CSR generation.

3.1.1 Certificate Issuance

The system issues X.509 v3 certificates for CSRs approved by an admin:

- **Process**: Upon CSR approval, the `approveCSR` function in `certificateController.js` uses `node-forge` to generate a certificate. The certificate is signed with the CA's private key (`ca-key.pem`) and includes:
  - Subject: Domain from the CSR (e.g., `test.example.com`)
  - Issuer: CA details from `ca-cert.pem`
  - Serial Number: Auto-generated unique ID
  - Validity: Default 1-year period (configurable in code)
  - Key Usage: Basic extensions (e.g., digital signature)
- **Storage**: Certificates are stored in:
  - `issued_certificates` table: Fields include `id`, `user_id`, `csr_id`, `domain`, `certificate` (PEM format), `issued_at`, `valid_till`, and `status` (`active`)
  - Filesystem: As `certificates/cert_<id>.pem`
- **Implementation**: The backend API endpoint `POST /api/admin/approve-csr` triggers issuance, updating the CSR status to `approved` and sending an email notification

Example Workflow

1. A user submits a CSR for `test.example.com`
2. An admin approves the CSR via the admin dashboard
3. The system generates a certificate with:

```
Subject: CN=test.example.com
Issuer: CN=Local CA, O=CA Service
Valid From: 2025-05-14
Valid To: 2026-05-14
```

4. The certificate is saved in `issued_certificates` and as `cert_123.pem`
5. The user receives an email with a download link

3.1.2 CSR Handling

The system generates and manages CSRs:

- **Generation**: The `generateCSR` function in `certificateController.js` uses OpenSSL via `child_process.exec` to create a CSR and private key. Users provide:
  - Domain (e.g., `test.example.com`)
  - Company, Division, City, State, Country, Email
  - Key Length (e.g., 2048 bits)
  - Signature Algorithm (e.g., SHA256)
- **Storage**: CSRs are stored in:
  - `csr_request` table: Fields include `id`, `user_id`, `domain`, `company`, `status` (`pending`), `csr` (PEM format)
  - Filesystem: As `csr_files/<domain>.csr` and `<domain>-private.key`
- **API**: The endpoint `POST /api/generate-csr` handles CSR creation, returning the CSR and private key file paths

3.1.3 Revocation Tracking

The system tracks certificate revocations:

- **Process**: Revocations are recorded in the `revoked_certificates` table with:
  - `certificate_id`: References `issued_certificates`
  - `revoked_at`: Timestamp
  - `reason`: Text (e.g., "Key compromise")
- **Implementation**: The `revokeCertificate` function (assumed in `certificateController.js`) updates the certificate's `status` to `revoked` and logs the revocation
- **Limitation**: No real-time validation (e.g., CRL or OCSP) is implemented

3.1.4 Key Management

The CA uses a single key pair:

- **CA Key Pair**: Stored as `ca-key.pem` (private key) and `ca-cert.pem` (certificate) in the `certs/` directory
- **Usage**: The private key signs certificates; the certificate defines the CA's identity
- **Security**: Keys are loaded into memory during certificate issuance, with no HSM or key escrow

3.2 Registration Authority (RA) Module

The RA module manages user registration, CSR workflows, and user interfaces.

3.2.1 User Registration

Users register via a web portal:

- **Process**: The `signup` endpoint (`POST /api/auth/signup`) creates a user in the `users` table:
  - `username`, `email`, `password` (bcrypt-hashed)
  - `role`: `client` or `admin` (default: `client`)
  - `is_verified`: `false` until OTP verification
- **Verification**: A 6-digit OTP is sent via Gmail SMTP (`emailService.js`). The `verify-otp` endpoint updates `is_verified` to `true`
- **Default Admin**: On startup, `index.js` creates an admin user (`username: admin, email: caservice2025@gmail.com, password: admin@123`)

3.2.2 CSR Submission and Approval

The system supports CSR workflows:

- **Submission**: Users submit CSRs via `POST /api/submit-csr`, providing CSR attributes. The CSR is stored in `csr_requests` with `status=pending`
- **Approval/Rejection**:
  - Admins access `GET /api/admin/pending-csrs` to list pending CSRs
  - `POST /api/admin/approve-csr`: Triggers certificate issuance and updates `status=approved`
  - `POST /api/admin/reject-csr`: Sets `status=rejected` with a `rejection_reason`
- **Notifications**: `sendCertificateApprovalEmail` or `sendCertificateRejectionEmail` sends emails with status updates

Example Workflow

1. User user1 logs in and submits a CSR:

```
Domain: test.example.com
Company: Test Inc
City: Rupnagar
State: Punjab
Country: IN
Email: user1@example.com
Key Length: 2048
```

2. The CSR is saved in `csr_requests` (ID: 101, `status=pending`)
3. Admin logs in, views CSR 101, and approves it
4. A certificate is issued, stored in `issued_certificates` (ID: 201), and saved as `cert_201.pem`
5. User receives an email: "Your CSR for test.example.com has been approved"

3.2.3 User Interface

The frontend provides:

- **User Dashboard** (`[FrontendURL]`):
  - **CSR Submission**: Form with fields for domain, company, etc., using Formik/Yup
  - **CSR/Certificate Viewing**: Tables listing CSRs (`GET /api/user/csrs`) and certificates (`GET /api/user/certificates`)
  - **Downloads**: Buttons to download CSR/private key/certificate PEM files
  - **Stats**: Displays total CSRs, pending, approved, and active certificates (`GET /api/user/dashboard-stat`
- **Admin Dashboard** (`[AdminURL]`):
  - **CSR Management**: Lists pending CSRs (`GET /api/admin/pending-csrs`) and all CSRs (`GET /api/admin/all-csrs`)
  - **Actions**: Approve/reject buttons for each CSR
  - **Stats**: System-wide metrics (`GET /api/admin/dashboard-stats`)
- **Implementation**: Built with React, Material-UI for styling, and Axios for API calls

3.3 Security Features

The system incorporates:

- **Authentication**: JWT tokens issued via `POST /api/auth/login`, validated with `JWT_SECRET`
- **RBAC**: Middleware checks `role` in `users` table, restricting admin endpoints (e.g., `/api/admin/*`) to `admin` users
- **Password Security**: Passwords are hashed with bcrypt (`package.json`)
- **Data Encryption**: HTTPS assumed for API communication (Express default)
- **Email Security**: OTPs (6-digit, 10-minute expiry) for signup and password reset, sent via secure SMTP

4 System Architecture

The system is structured as follows:

4.1 Backend

- **Framework**: Node.js with Express, running on port 5000 (`index.js`)
- **Key Files**:
  - `certificateController.js`: Handles CSR generation (`generateCSR`), submission (`submitCSR`), approval (`approveCSR`), rejection (`rejectCSR`), and certificate downloads
  - `emailService.js`: Sends OTPs and notifications using `nodemailer`
  - `opensslService.js`: Generates CSRs via OpenSSL commands
- **APIs**: Table 1 lists key endpoints

Table 1: Key Backend API Endpoints

`Endpoint`	**Method**	**Description**
`/api/auth/signup`	POST	Registers a user with email verification
`/api/auth/verify-otp`	POST	Verifies OTP to activate account
`/api/auth/login`	POST	Issues JWT token for authentication
`/api/generate-csr`	POST	Generates CSR and private key
`/api/submit-csr`	POST	Submits CSR for approval
`/api/user/csrs`	GET	Lists user's CSRs
`/api/user/certificates`	GET	Lists user's certificates
`/api/admin/pending-csrs`	GET	Lists pending CSRs (admin only)
`/api/admin/approve-csr`	POST	Approves CSR and issues certificate
`/api/admin/reject-csr`	POST	Rejects CSR with reason

4.2 Frontend

- **Framework**: React with Vite, running on port 5173
- **Components**: Material-UI for UI (tables, forms, buttons), Formik/Yup for form validation, Axios for API calls
- **Pages**:
  - `/signup`, `/login`: User authentication
  - `/`: User dashboard for CSR submission and certificate management
  - `/admin`: Admin dashboard for CSR approval and stats

4.3 Database

The PostgreSQL database (`ca_db`) includes:

- **users**: Stores user data (Table 2)
- **csr_requests**: Tracks CSRs (Table 3)
- **issued_certificates**: Stores certificates (Table 4)
- **revoked_certificates**: Tracks revocations (Table 5)

4.4 External Services

- **Gmail SMTP**: Used via `nodemailer` for OTPs and CSR notifications. Configured with `EMAIL_USER` and `EMAIL_PASSWORD` in `.env`
- **OpenSSL**: Generates CSRs through command-line execution, producing `.csr` and `.key` files

Table 2: `users` Table Schema

`Field`	Type	Description
`id`	INTEGER	Primary key, auto-incremented
`username`	VARCHAR(255)	Unique username
`email`	VARCHAR(255)	Unique email
`password`	VARCHAR(255)	Bcrypt-hashed password
`role`	VARCHAR(50)	`client` or `admin`
`is_verified`	BOOLEAN	True if email verified
`otp`	VARCHAR(6)	OTP for verification
`otp_expiry`	TIMESTAMP	OTP expiration time

Table 3: `csr_requests` Table Schema

`Field`	Type	Description
`id`	INTEGER	Primary key, auto-incremented
`user_id`	INTEGER	Foreign key to `users`
`domain`	VARCHAR(255)	CSR domain (e.g., `test.example.com`)
`company`	VARCHAR(255)	Company name
`status`	VARCHAR(20)	`pending`, `approved`, or `rejected`
`csr`	TEXT	CSR in PEM format

4.5 Architecture Diagram

Figure 1 illustrates the system's components and interactions.

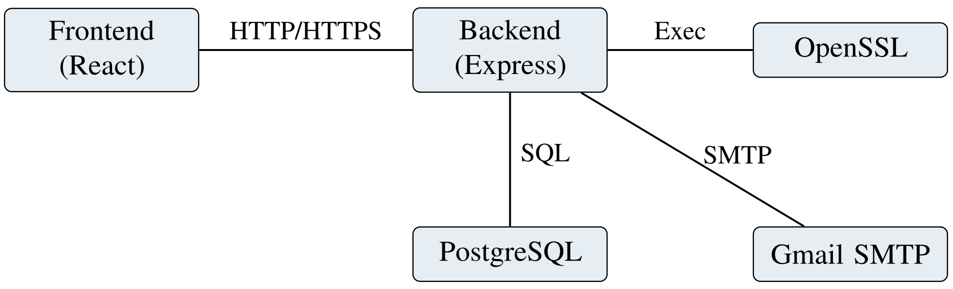

Figure 1: System Architecture

5 Limitations

The prototype has the following constraints:

- **No Real-Time Validation**: Lacks mechanisms like OCSP or CRL for checking certificate status, limiting revocation usability
- **Single CA Key Pair**: Uses one key pair without HSM or key rotation, posing security risks for production use

Table 4: `issued_certificates` Table Schema

Field	**Type**	**Description**
`id`	INTEGER	Primary key, auto-incremented
`user_id`	INTEGER	Foreign key to `users`
`csr_id`	INTEGER	Foreign key to `csr_requests`
`domain`	VARCHAR(255)	Certificate domain
`certificate`	TEXT	Certificate in PEM format
`status`	VARCHAR(20)	`active` or `revoked`

Table 5: `revoked_certificates` Table Schema

Field	**Type**	**Description**
`id`	INTEGER	Primary key, auto-incremented
`certificate_id`	INTEGER	Foreign key to `issued_certificates`
`revoked_at`	TIMESTAMP	Revocation timestamp
`reason`	TEXT	Revocation reason

- **Basic Audit Logging**: Relies on database logs without tamper-evident mechanisms or SIEM integration
- **Limited Scalability**: Designed for small-scale use, with no load balancing or redundancy
- **No Advanced Policies**: Certificates use default attributes without configurable key usage or profiles

6 Conclusion

The Local Certificate Authority Service prototype delivers a functional PKI solution for managing CSRs and issuing X.509 certificates. With robust user and admin interfaces, secure authentication, and email notifications, it provides a solid foundation for small-scale deployments or educational purposes. Its open-source release invites contributors to build upon its capabilities, leveraging the detailed insights in this document.

A Sample API Requests

A.1 Submit CSR

```
POST /api/submit-csr
Authorization: Bearer <JWT>
{
  "domain": "test.example.com",
  "company": "Test Inc",
  "division": "IT",
  "city": "Rupnagar",
  "state": "Punjab",
  "country": "IN",
  "email": "user1@example.com",
  "rootLength": 2048
}
Response: { "success": true, "csrId": 101 }
```

A.2 Approve CSR

```
POST /api/admin/approve-csr
Authorization: Bearer <Admin JWT>
{ "csrId": 101 }
Response: { "success": true, "certificateId": 201 }
```

B Database Schema

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY,
  username VARCHAR(255) UNIQUE,
  email VARCHAR(255) UNIQUE,
  password VARCHAR(255),
  role VARCHAR(50) DEFAULT 'client',
  is_verified BOOLEAN DEFAULT false,
  otp VARCHAR(6),
  otp_expiry TIMESTAMP
);

CREATE TABLE csr_requests (
  id INTEGER PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  domain VARCHAR(255),
  company VARCHAR(255),
  status VARCHAR(20) DEFAULT 'pending',
  csr TEXT
);

CREATE TABLE issued_certificates (
  id INTEGER PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  csr_id INTEGER REFERENCES csr_requests(id),
  domain VARCHAR(255),
  certificate TEXT,
  status VARCHAR(20) DEFAULT 'active'
);

CREATE TABLE revoked_certificates (
  id INTEGER PRIMARY KEY,
  certificate_id INTEGER REFERENCES issued_certificates(id),
  revoked_at TIMESTAMP,
  reason TEXT
);
```