Company Confidential

# Pikachu-CA PKI Hands-On 2.0.1

| | |
|---|---|
| Last updated | 2-Jan-2026 |
| Doc. version | 2.0.1 |
| Doc. owner | Uzi Golan |
| Approved by | |
| Customer | R&D |
| Project or installation name | |
| Project number | |
| Solution name | Choose an item. |
| RAD products and versions included | |
| Content type | |
| Keywords | |

# Contents

# 1 Introduction

## 1.1 Glossary

- SCEP – Simple Certificate Enrollment Protocol. Enables automated certificate management by allowing devices to securely request and retrieve certificates from a CA over HTTP.
- EST – Enrollment over Secure Transport. Automates certificate issuance by allowing devices to submit CSRs and receive signed certificates over secure HTTPS or mTLS channels.
- MQTTs Broker – MQTT broker running over TLS. Routes messages between clients using publish/subscribe and enforces certificate-based client authentication.
- CRL (Certificate Revocation List) – List of certificates that have been revoked by a CA. Clients can download the CRL to verify whether a certificate is still valid.
- Certificate Revocation – The process of invalidating a certificate before its expiration so that it is no longer trusted.
- OCSP (Online Certificate Status Protocol) – Real-time certificate status checking protocol. Allows clients to verify whether a certificate has been revoked without downloading the full CRL.
- Quantum-Safe OpenSSL Provider (oqsprovider) – Open Quantum Safe provider for OpenSSL 3.x, adding post-quantum algorithms (e.g., Dilithium) for generating and using quantum-safe keys and signatures.

## 1.2 PKI Concept

Pikachu CA operates as part of a standard Public Key Infrastructure (PKI) built on X.509 and IETF-PKIX concepts.

**Root CA**
A Root CA has a self-signed certificate (Trusted Root). All certificate chains terminate at this root. The Root CA certificate must be distributed and configured as trusted on clients.

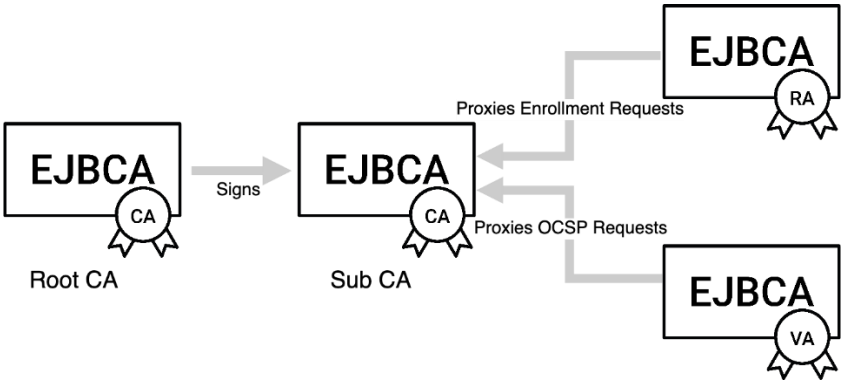
**Subordinate / Intermediate CA (Sub CA)**
A Sub CA certificate is signed by a Root CA (or another intermediate). It does not need to be directly trusted by clients; instead, it forms a chain that ends at the Root CA.

**Registration Authority (RA)**
The RA is responsible for registering entities, validating identity, and approving certificate requests

according to CA policy. Pikachu CA's RA functions are implemented via the Web UI and APIs (manual signing, enrollment policies, etc.).

**Validation Authority (VA)**
The VA is responsible for reporting certificate status. In Pikachu CA, this is implemented via:

- Generated **CRL** (downloadable file)
- **OCSP** responder for real-time status




From : https://docs.keyfactor.com/ejbca/9.0/ejbca-concepts

## 1.3 Pikachu CA Overview

Pikachu CA is a lightweight yet powerful Public Key Infrastructure (PKI) server built on Python/Flask. It provides full certificate lifecycle management, automated device enrollment, MQTT TLS integration, and optional HashiCorp Vault-based key isolation.

**Core Capabilities**

From the README (condensed and cleaned):

- Generate **RSA**, **ECC**, and **post-quantum (Dilithium)** private keys
- Submit and sign Certificate Signing Requests (CSR)
- View, revoke, and delete issued certificates
- Download CA certificates, chains, and CRLs
- Real-time certificate validation via **OCSP**

- Automated device enrollment via **SCEP** and **EST**
- TLS client-certificate authentication for MQTT, including CRL enforcement

## Feature Matrix

| Feature Category | Capability | Supported | Notes |
|---|---|---|---|
| **CA Operations** | Issue / revoke / delete certificates | ✓ | CRL + OCSP supported |
| | CSR signing | ✓ | Manual & automated |
| | Multi-CA mode (RSA/EC) | ✓ | Selectable via config.ini |
| | Post-Quantum keys | ✓ | Dilithium (mldsa) |
| **Enrollment** | SCEP | ✓ | Support challenge password (optional) |
| | EST | ✓ | HTTPS + mTLS, Vault (optional) |
| | Manual RA signing | ✓ | Via Web UI |
| **Identity & Access** | Role-based access control (RBAC) | ✓ | Admin / User |
| | LDAP authentication | ✓ | Optional |
| | User state management | ✓ | Pending / Active / Suspended |
| | REST API token authentication | | |
| **Security** | Vault private key isolation | ✓ | Optional, recommended |
| | Zero-trust key handling | ✓ | Keys never leave Vault |
| | Server audit logs | ✓ | Events for all operations |

| Feature Category | Capability | Supported | Notes |
|---|---|---|---|
| | Vault audit logs | ✓ | If enabled |
| **Protocols** | OCSP | ✓ | Real-time validation |
| | CRL | ✓ | Download + MQTT enforcement |
| **MQTT** | TLS client auth | ✓ | Tested with Mosquitto |
| **DevOps** | PowerShell/CLI helper scripts | ✓ | Setup, cleanup, migration (where used) |
| | Health checks | ✓ | Vault fallback logic |
| **Usability** | Web UI | ✓ | Browser-based management |
| | REST API | ✓ | All key operations exposed |
| | Data inspection tool | ✓ | PEM/DER parsing |
| | Templates & profiles | ✓ | Jinja-based profile generation |

## Feature List

The system includes both:

- A modern, user-friendly **Web UI**
- A fully documented **REST API**

Both expose the same lifecycle actions: key creation, CSR submission, certificate issuance, revocation, inspection, and enrollment operations.

**Web Interface & API**

- Modern Web UI for all PKI operations

- Full RESTful API for automation and integration

**Certificate Lifecycle Management**

- Generate keys: **RSA**, **ECC**, and **post-quantum (Dilithium)**
- Create, upload, and sign CSRs
- View, revoke, and delete issued certificates
- Download CA certificates, chains, and CRLs
- Real-time checks using OCSP

**Automated Device Enrollment**

- **SCEP** for automated enrollment (local keys only)
- **EST** with HTTPS or mTLS (Vault optional)
- MQTT TLS authentication with CRL-based revocation

**Access Control & Multi-Tenancy**

- Role-based access control (User / Admin)
- User isolation and states (Pending, Active, Suspended)

**HashiCorp Vault Integration (Optional)**

- CA private keys stored in Vault, never on disk
- Signing via Vault PKI engine
- Separate Vault engines for RSA and EC
- Automatic fallback to file-based keys if Vault is unreachable
- AppRole authentication and policy-based authorization
- Vault used for Web UI and EST enrollment
  (SCEP always uses local keys)

**Security & Architecture**

- Strong key isolation in Vault mode
- Secure signing workflows that never expose raw keys
- Granular authorization for CA operations
- Local event log + optional Vault audit logs
- Automatic failover to legacy file-based keys

**Management & Operations**

- Helper scripts for starting/stopping services, clearing logs, setting credentials, and DB initialization
- Centralized configuration using config.ini
- Configurable logging (level + path)
- Vault health checks and fallback logic
- Support for Vault-based key rotation

**Protocols & Integrations**

- SCEP, EST, OCSP, CRL
- Optional LDAP integration for user authentication
- MQTT integration for TLS & CRL enforcement

**Usability & UI**

- Filterable, paginated tables for certificates, keys, users, and events
- Download options for certs, chains, CRLs, PFX, and keys
- Admin and user dashboards with live status
- Built-in help PDF and hands-on instructions

## Quantum safe keys

Quantum-safe keys are generated using algorithms designed to remain secure even against quantum-capable adversaries.

Pikachu CA supports Dilithium-based signatures via the **Open Quantum Safe** provider (oqsprovider) for OpenSSL 3.x:

- mldsa44 – comparable to **Dilithium2**, NIST Level 1
- mldsa65 – comparable to **Dilithium3**, NIST Level 3
- mldsa87 – comparable to **Dilithium5**, NIST Level 5

These are available when the oqsprovider is installed and enabled.

## 1.4 Purpose

The goal of this platform is to provide a robust yet lightweight PKI solution for:

- Securely provisioning and managing RAD devices and servers
- Supporting MQTT and other TLS-based infrastructures
- Demonstrating multi-tenant PKI with modern protocols and optional Vault-backed key isolation

## 1.5 Scope

This document focuses on:

- **Non-production environments** (lab, R&D, PoC)
- Multi-tenant setups with tenant isolation
- Secure enrollment via SCEP and EST
- MQTT integration and OCSP/CRL validation
- Basic operational and troubleshooting steps

# 2 System Architecture & Platform

The Lifecycle maintenance ansible playbook contains files and ansible playbooks.

Once installed the user can perform the maintenance operations.

## 2.1 PKI Platform Overview

Pikachu CA manages certificate lifecycles end-to-end, including key generation, CSR handling, issuance, revocation (CRL/OCSP), automated enrollment (SCEP/EST), and MQTT TLS authentication workflows.

Pikachu CA is distributed as a set of files and Ansible playbooks (lifecycle maintenance).
Once installed, the platform allows you to:

- Generate and manage CA material (Root/Sub)
- Run a CA web server (Flask-based)
- Configure enrollment policies, templates, and protocols
- Integrate with MQTT, LDAP, and Vault (optional)

## 2.2 Server OS requirements

| Operating System | Version |
|---|---|
| Rocky | 9.x |

Other RHEL-compatible distributions may also work with minor adjustments.

## 2.3 High-Level Architecture

At a high level, Pikachu CA consists of:

- **Web UI + REST API** (Flask application)
- **CA logic** for issuing and revoking certificates

- **SQLite database** for storing certificates, keys metadata, users, and events
- **CRL/OCSP** components for revocation status
- Optional **HashiCorp Vault** for key isolation
- Integration with **SCEP**, **EST**, and **MQTT**
- Supports file-based or Vault-based CA keys
- Vault mode enables:
    - key isolation
    - secure signing using Vault PKI engine
    - separate RSA and EC CA engines

In Vault mode, the private CA keys are never loaded into application memory. All issuance operations are delegated to Vault. If Vault is unavailable, the server falls back automatically to local file-based keys.

## 2.4 Network Ports

| Capability | Configuration Section | Port | Description |
|---|---|---|---|
| HTTPS without client certificate | [HTTPS].port | 443 | Standard one-way TLS (UI & APIs) |
| HTTPS with client certificates | [TRUSTED_HTTPS].trusted_port | 4443 | Mutual TLS (mTLS) |
| HTTP (OCSP) | [DEFAULT].http_port | 80 | OCSP responder (legacy HTTP) |
| HTTP (SCEP) | [DEFAULT].http_port | 80 | SCEP enrollment endpoint |

Ports are controlled by config.ini.

# 3 Installation & Configuration

## 3.1 Python and System Dependencies

Install Python 3.11 and build tools:

```
sudo dnf install -y epel-release
sudo dnf install -y python3.11 python3.11-devel python3.11-pip git
sudo dnf groupinstall -y "Development Tools"

sudo alternatives --install /usr/bin/python3 python3 /usr/bin/python3.11 100
sudo alternatives --install /usr/bin/pip3 pip3 /usr/bin/pip3.11 100
```

Create `requirements.txt`:

```
@'
Flask==3.1.2
Flask-SQLAlchemy==3.1.1
Flask-Login==0.6.3
cryptography==46.0.3
asn1crypto==1.5.1
Werkzeug==3.1.4
SQLAlchemy==2.0.44
Jinja2==3.1.6
MarkupSafe==3.0.3
itsdangerous==2.2.0
click==8.3.1
blinker==1.9.0
cffi==2.0.0
pycparser==2.23
greenlet==3.2.4
typing_extensions==4.15.0
colorama==0.4.6
oscrypto==1.3.0
tzdata==2025.2
hvac==2.1.0
certifi==2025.11.12
charset-normalizer==3.4.4
idna==3.11
requests==2.32.5
urllib3==2.5.0
ldap3
'@ | Set-Content requirements.txt
```

Install Python dependencies:

```
pip3 install -r requirements.txt
```

Allow Python to bind privileged ports (<1024):

```
sudo setcap 'cap_net_bind_service=+ep' /usr/bin/python3.11
```

## 3.2 Server Installation and Layout

### GitHub Repository

```
git clone https://github.com/uzigolan/pikachu-ca.git
cd pikachu-ca
```

### Tar file

Extract the server archive:

```
tar xvfz pki_server_102.tar.gz
cd pki-server-2
```

Key directories and files:

- app.py – Main Flask application
- config.ini – Server configuration
- db/pikachu-ca.db – SQLite database
- pki-root/ – Root CA certificates
- pki-subca/ – Sub-CA keys and certificates
- pki-https/ – HTTPS server certificates
- pki-misc/ – CRL, validity, and server extension configs
- x509_profiles/ – Generated certificate profiles
- x509_templates/ – Jinja .cnf.j2 templates
- html_templates/ – Web UI templates
- logs/ – Server logs

## 3.3 Complementary Tools

### sscep (SCEP Client)

Rocky 9 does not provide sscep in the base repos, so it must be built from source:

```
git clone https://github.com/zhaozg/openscep.git
cd openscep
chmod u+x configure
./configure
make
sudo make install
```

Install any missing libraries using dnf if configure complains.

### estclient (EST Client)

```
sudo dnf install -y golang
git clone https://github.com/globalsign/est.git
cd est
go install github.com/globalsign/est/cmd/estclient@latest
sudo cp ~/go/bin/estclient /usr/local/bin/estclient
```

### Quantum-Safe (oqsprovider)

```
sudo dnf config-manager --set-enabled crb
sudo dnf install -y epel-release
sudo dnf install -y ninja-build
sudo dnf groupinstall -y "Development Tools"
sudo dnf install -y cmake ninja-build git openssl-devel libffi-devel

git clone https://github.com/open-quantum-safe/oqs-provider.git
cd oqs-provider
./scripts/fullbuild.sh
sudo cmake --install _build
```

Enable oqsprovider in your openssl.cnf (typically /etc/pki/tls/openssl.cnf):

```
[provider_sect]
default = default_sect
oqsprovider = oqsprovider_sect

[default_sect]
activate = 1

[oqsprovider_sect]
activate = 1
```

Verify:

```
openssl list -providers
```

You should see oqsprovider as **active**.

## MQTT Broker (Mosquitto)

Example Docker Compose based setup:

```
mkdir mqtt
cd mqtt
```

docker-compose.yml:

```
version: '3'
services:
  mosquitto:
    image: eclipse-mosquitto:latest
    container_name: mosquitto
    network_mode: "host"
    ports:
      - "1883:1883"
      - "2883:2883"
      - "9001:9001"
    volumes:
      - ./mosquitto.conf:/mosquitto/config/mosquitto.conf
      - ./mosquitto/certs:/mosquitto/certs
    restart: unless-stopped
```

mosquitto.conf:

```
per_listener_settings true

listener 1883 0.0.0.0
allow_anonymous true
log_type all

listener 2883 0.0.0.0
cafile /mosquitto/certs/CA.cert
certfile /mosquitto/certs/est_mqtt_server_1.pem
keyfile /mosquitto/certs/est_mqtt_server_1.key
crlfile /mosquitto/certs/crl.pem
require_certificate true
use_identity_as_username true
log_type all
```

Populate mosquitto/certs/ with:

- est_mqtt_server_1.pem
- est_mqtt_server_1.key
- CA.cert
- crl.pem (downloaded from Pikachu CA)

## 3.4 Server Configuration (config.ini)

Pikachu CA is configured via config.ini.

After any configuration changes, the server must be restarted.

Key sections:

**[DEFAULT]**

- SECRET_KEY – Flask session secret
- http_port – HTTP port for unauthenticated OCSP
- max_idle_time – Idle timeout (e.g. 30m, 10h, 4d)

**[LOGGING]**

- log_level – TRACE, DEBUG, INFO, WARNING, ERROR, CRITICAL

- log_file – Path to server log file

**[CA]**

- mode – Default sub-CA: EC or RSA
- SUBCA_KEY_PATH_* / SUBCA_CERT_PATH_* / CHAIN_FILE_PATH_* – Paths to EC/RSA sub-CA keys and chains
- ROOT_CERT_PATH – Root CA certificate

**[VAULT]**

Vault integration is only supported for Web UI and EST enrollment.
SCEP **always** uses local file-based keys due to protocol requirements.

- enabled – true / false
- address – Vault server URL
- role_id, secret_id – AppRole credentials (via env vars)
- pki_rsa_path, pki_ec_path – PKI mount paths
- transit_path – Transit engine path
- timeout, retry_attempts – Connectivity settings
- verify_ssl, ca_cert_path – TLS validation settings
- role_scep, role_est, role_default – Vault roles per use-case

**[LDAP]**

- LDAP_HOST, LDAP_PORT, BASE_DN, PEOPLE_DN
- ADMIN_DN, ADMIN_PASSWORD
- enabled – Enable/disable LDAP login

**[SCEP]**

- enabled – Enable/disable SCEP
- serial_file – SCEP serial persistence file
- dump_dir – Optional dump directory for raw requests
- challenge_password_enabled – Enable challenge passwords

  Challenge passwords apply **only** to SCEP and must be enabled using challenge_password_enabled = true.

- challenge_password_validity – Validity window (e.g. 2m, 60m)

**[HTTPS]**

- ssl_cert – Main HTTPS UI certificate
- ssl_key – Matching private key
- port – HTTPS port

**[TRUSTED_HTTPS]**

- trusted_ssl_cert – mTLS HTTPS certificate
- trusted_ssl_key – mTLS HTTPS private key
- trusted_port – mTLS port (default 4443)

**[PATHS]**

- crl_path – CRL file path
- server_ext_cfg – Server extension config ([v3_ext])
- validity_conf – Default validity config
- db_path – SQLite DB (db/pikachu-ca.db)

config.ini file example:

```
[DEFAULT]
# general Flask settings
SECRET_KEY = *******
# HTTP port for unauthenticated OCSP
http_port = 80
# Maximum idle time before automatic logout (e.g., "10h" or "4d" or "20m")
max_idle_time = 30m

[LOGGING]
# Logging level: TRACE (5), DEBUG (10), INFO (20), WARNING (30), ERROR (40), CRITICAL (50)
# TRACE: Very verbose - includes idle checks, session tracking
# DEBUG: Detailed diagnostic information
# INFO: General informational messages
# WARNING: Warning messages
# ERROR: Error messages
# CRITICAL: Critical errors
log_level = INFO
log_file = logs/server.log


[CA]
# Which subordinate CA to use by default: "EC" or "RSA"
mode = RSA
# Paths for both modes; the get_ca_config() helper below will pick the right
SUBCA_KEY_PATH_EC    = pki-subca/rad_ca_sub_ec.key
SUBCA_CERT_PATH_EC   = pki-subca/rad_ca_sub_ec.crt
CHAIN_FILE_PATH_EC   = pki-subca/rad_chain_ec.crt
SUBCA_KEY_PATH_RSA   = pki-subca/rad_ca_sub_rsa.key
SUBCA_CERT_PATH_RSA  = pki-subca/rad_ca_sub_rsa.crt
CHAIN_FILE_PATH_RSA  = pki-subca/rad_chain_rsa.crt
ROOT_CERT_PATH       = pki-root/rad_ca_root.crt

[VAULT]
# HashiCorp Vault integration for CA key isolation
# Set enabled=false to use traditional file-based keys (current behavior)
# Set enabled=true to use Vault PKI engine (enhanced security with key isolation)
enabled = false

# Vault server connection settings (required when enabled=true)
# For same server: http://127.0.0.1:8200 (dev mode without TLS)
# For separate server: https://192.168.1.20:8200
address = http://127.0.0.1:8200

# Authentication credentials (read from environment variables for security)
# Set these as environment variables: VAULT_ROLE_ID and VAULT_SECRET_ID
role_id = ${VAULT_ROLE_ID}
secret_id = ${VAULT_SECRET_ID}

# PKI mount paths in Vault
pki_rsa_path = pki-subca-rsa
pki_ec_path = pki-subca-ec

# Transit engine path for custom signing operations (OCSP, PQC)
transit_path = transit

# Connection settings
timeout = 30
```

```
retry_attempts = 3

# TLS settings
verify_ssl = true
ca_cert_path =

# Vault roles for different operations
role_scep = scep-enrollment
role_est = est-enrollment
role_default = server-cert

[LDAP]
LDAP_HOST = 172.18.178.24
LDAP_PORT = 389
BASE_DN = dc=rnd-rad,dc=com
PEOPLE_DN = ou=People,dc=rnd-rad,dc=com
ADMIN_DN = cn=Manager,dc=rnd-rad,dc=com
ADMIN_PASSWORD = marketing
enabled = false

[SCEP]
# enable or disable SCEP entirely
enabled = true
# optional path to a file where we persist the serial across restarts
serial_file = pki-subca/serial.txt
# Dump directory for raw SCEP requests (optional)
dump_dir = pki-subca/dumps

# Challenge-password feature
challenge_password_enabled = true
# Validity period for challenge-passwords (default: 60m)
challenge_password_validity = 2m

[HTTPS]
# HTTPS certificate & key for your main CA UI
ssl_cert = pki-https/tls.cert.pem
ssl_key  = pki-https/tls.key.pem
port     = 443

[TRUSTED_HTTPS]
# HTTPS certificate & key for your main CA UI
#trusted_ssl_cert = pki-https/pikachu_issued_https.crt
trusted_ssl_cert = pki-https/pikachu_issued_https_localhost.crt
trusted_ssl_key  = pki-https/pikachu_issued_https.key
trusted_port     = 4443

[PATHS]
# everything else that was hard-coded
crl_path      = pki-misc/crl.pem
server_ext_cfg = pki-misc/server_ext.cnf
validity_conf  = pki-misc/validity.conf
db_path       = db/pikachu-ca.db
```

## 3.5 Database Initialization

```
python migrate_db.py
```

This script is idempotent and safe to run multiple times.
It initializes tables, applies schema updates, and creates the default admin user if missing.

# 4 CA Initialization (Root & Sub CA)

## 4.1 Generate Root CA Certificate (EC)

Create an EC root key:

```
openssl ecparam -name prime256v1 -genkey -noout -out rad_ca_root.key
```

Create rad_ca_root.cnf:

```
[ req ]
default_bits       = 4096
default_md         = sha256
prompt             = no
distinguished_name = dn
x509_extensions    = v3_ca

[ dn ]
C  = IL
ST = TLV
L  = Tel Aviv
O  = RAD
OU = RD
CN = RAD Test ECDSA

[ v3_ca ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints       = critical, CA:true, pathlen:1
```

Self-sign the root certificate:

```
openssl req -config rad_ca_root.cnf -key rad_ca_root.key \
  -new -x509 -days 3650 -sha256 -out rad_ca_root.crt
```

Create ca_root_ext.cnf (used to sign intermediates):

```
[ v3_intermediate ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = keyCertSign, cRLSign
crlDistributionPoints  = URI:https://pikachu-ca.rnd-rad.com/downloads/crl
```

## 4.2 Generate Intermediate (Sub) CA Certificates

### EC Sub CA

Create EC sub-CA key:

```
openssl ecparam -name prime256v1 -genkey -noout -out rad_ca_sub_ec.key
```

Create rad_ca_sub_ec.cnf:

```
[ req ]
default_bits       = 2048
default_md         = sha256
prompt             = no
distinguished_name = dn
req_extensions     = v3_intermediate

[ dn ]
C  = IL
ST = TLV
L  = Tel Aviv
O  = RAD
OU = RD
CN = RADSubTestECDSA

[ v3_intermediate ]
subjectKeyIdentifier   = hash
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = keyCertSign, cRLSign
```

Generate CSR:

```
openssl req -new -config rad_ca_sub_ec.cnf \
  -key rad_ca_sub_ec.key -out rad_ca_sub_ec.csr
```

Sign with the Root CA:

```
openssl x509 -req -in rad_ca_sub_ec.csr \
  -CA rad_ca_root.crt -CAkey rad_ca_root.key -CAcreateserial \
  -out rad_ca_sub_ec.crt -days 3650 -sha256 \
  -extfile ca_root_ext.cnf -extensions v3_intermediate
```

### RSA Sub CA

Create RSA sub-CA key:

```
openssl genpkey -algorithm RSA -out rad_ca_sub_rsa.key \
  -pkeyopt rsa_keygen_bits:4096
```

Create rad_ca_sub_rsa.cnf similar to EC, then:

```
openssl req -new -config rad_ca_sub_rsa.cnf \
  -key rad_ca_sub_rsa.key -out rad_ca_sub_rsa.csr
```

Sign:

```
openssl x509 -req -in rad_ca_sub_rsa.csr \
  -CA rad_ca_root.crt -CAkey rad_ca_root.key -CAcreateserial \
  -out rad_ca_sub_rsa.crt -days 3650 -sha256 \
  -extfile ca_root_ext.cnf -extensions v3_intermediate
```

## Copying Keys and Chains

Move the generated files into the project structure:

```
PROJ_DIR=~/pki-server-2

cp rad_ca_root.key rad_ca_root.crt $PROJ_DIR/pki-root/
cp rad_ca_sub_ec.crt rad_ca_sub_rsa.crt \
   rad_ca_sub_ec.key rad_ca_sub_rsa.key \
   $PROJ_DIR/pki-subca/

cd $PROJ_DIR

cat pki-subca/rad_ca_sub_ec.crt  pki-root/rad_ca_root.crt > pki-
subca/rad_chain_ec.crt
cat pki-subca/rad_ca_sub_rsa.crt pki-root/rad_ca_root.crt > pki-
subca/rad_chain_rsa.crt
```

Ensure that the generated Root/Sub CA certificates and keys match the paths defined under [CA] in config.ini before starting the server.

# 5 Running the Server

## 5.1 Running from Shell

From the project directory:

```
cd pki-server-2
nohup python app.py > app.log 2>&1 &
```

Logs are written both to stdout and app.log (and to logs/server.log if configured).

When Vault integration is enabled, the following environment variables must be set before starting the server:

```
export VAULT_ROLE_ID="..."
export VAULT_SECRET_ID="..."
export VAULT_ADDR="http://127.0.0.1:8200"
```

## 5.2 Running as a systemd Service

### Main service

Create /etc/systemd/system/pikachu-ca.service:

```
[Unit]
Description=Pikachu CA Python App
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=rocky
Group=rocky
WorkingDirectory=/home/rocky/pki-server-2
ExecStart=/usr/bin/python3.11 /home/rocky/pki-server-2/app.py
Restart=always
RestartSec=5s
Environment=PYTHONUNBUFFERED=1

AmbientCapabilities=CAP_NET_BIND_SERVICE
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
NoNewPrivileges=true

StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
```

## Health Check Service & Timer

pikachu-ca-healthcheck.service:

```
[Unit]
Description=Healthcheck for pikachu-ca (restart if https://localhost fails)
After=pikachu-ca.service

[Service]
Type=oneshot
ExecStart=/usr/bin/bash -c '/usr/bin/curl -ksf --max-time 5
https://localhost/ || /usr/bin/systemctl restart pikachu-ca.service'
pikachu-ca-healthcheck.timer:
[Unit]
Description=Periodic healthcheck for pikachu-ca

[Timer]
OnBootSec=30s
OnUnitActiveSec=60s
Unit=pikachu-ca-healthcheck.service

[Install]
WantedBy=timers.target
```

Enable and start:

```
sudo systemctl daemon-reload
sudo systemctl enable pikachu-ca.service
sudo systemctl enable pikachu-ca-healthcheck.timer

sudo systemctl start pikachu-ca.service
sudo systemctl start pikachu-ca-healthcheck.timer
```

Check status:

```
sudo systemctl status pikachu-ca.service
sudo systemctl status pikachu-ca-healthcheck.timer
journalctl -u pikachu-ca.service -f
```

To verify Vault mode, check server logs for:

```
INFO [app] Vault integration is ENABLED
INFO [vault_client] Authenticated with Vault
```

## 5.3 Accessing the Web UI

By default, the Web UI is available at:

```
https://pikachu-ca.<subdomain>.com:4443    (mTLS, trusted HTTPS)

https://pikachu-ca.<subdomain>.com:443     (standard HTTPS)

http://pikachu-ca.<subdomain>.com:80     (standard HTTP)
```

Actual port and certificate settings depend on [HTTPS] , [DEFAULT] and [TRUSTED_HTTPS] in config.ini.

# 6 Web UI Overview & Core Features

## 6.1 Top Navigation Layout

The Pikachu CA Web UI provides fast access to all PKI operations.
Navigation items (and their roles) are:

- **Certs** – List, filter, view, download, revoke, or delete issued certificates.
- **RA (Sign)** – Submit a CSR and sign it manually.
- **CSR Requests** – Create, view, and manage certificate signing requests.
- **Keys** – Create, list, view, download, or delete cryptographic keys.
- **Profiles** – Create or edit certificate profiles used for CSR generation or extension configuration.
- **Enrollment Policies** – Configure policy rules used for automated enrollment.
- **VA (CRL)** – View revoked certificates and download the most recent CRL.
- **Server Extensions** – Configure default X.509 extensions applied during certificate issuance.
- **CA Mode** – View Root CA and Sub-CA details.
- **Templates** *(Admin only)* – Create certificate profiles using Jinja-based templates.
- **APIs** – Browse all REST API endpoints (CSR, SCEP, EST, OCSP, CRL, CA chain).
- **Inspect** – Analyze PEM/DER objects (cert, CSR, CRL, PFX, OCSP).
- **Config** – View server configuration (secret-protected).
- **ChatPikachu** – Built-in assistant for CA operations.
- **Help** – Open this documentation.
- **Logs** – View live server logs.
- **Users** *(Admin only)* – Manage user accounts.
- **Account** – Change password(none LDAP user) , Managed API tokens and log out.

## Python

Install python 3 and complimentary packages

```
sudo dnf install -y epel-release

#sudo dnf module enable -y python3.11

sudo dnf install -y python3.11 python3.11-devel python3.11-pip git

sudo dnf groupinstall -y "Development Tools"


sudo alternatives --install /usr/bin/python3 python3 /usr/bin/python3.11 100
sudo alternatives --install /usr/bin/pip3 pip3 /usr/bin/pip3.11 100

@'
Flask==3.1.2
Flask-SQLAlchemy==3.1.1
Flask-Login==0.6.3
cryptography==46.0.3
asn1crypto==1.5.1
Werkzeug==3.1.4
SQLAlchemy==2.0.44
Jinja2==3.1.6
MarkupSafe==3.0.3
itsdangerous==2.2.0
click==8.3.1
blinker==1.9.0
cffi==2.0.0
pycparser==2.23
greenlet==3.2.4
typing_extensions==4.15.0
colorama==0.4.6
oscrypto==1.3.0
tzdata==2025.2
hvac==2.1.0
certifi==2025.11.12
charset-normalizer==3.4.4
idna==3.11
requests==2.32.5
urllib3==2.5.0
'@ | Set-Content requirements.txt;

pip3 install -r requirements.txt
```

allow python39 programs to use privileged ports (below 1024)

```
sudo setcap 'cap_net_bind_service=+ep' /usr/bin/python3.11
```

## Server

### *Tar file*

Extract the pki_server_102.tar.gz

```
tar xvfz pki_server_102.tar.gz
cd pki-server-2
```

### *GitHub*

Clone repo in GitHub

```
gh auth login


gh repo clone uzigolan/pikachu-ca -- --branch 2.0

Note : currently private repo
```

### *Files*

it will open the following directories and files

```
├── app.log
├── app.py
├── asn1.py
├── builders.py
├── ca_mode.conf
├── ca.py
├── certificates
├── certs.db
├── chain.crt
├── chainx.crt
├── config_bp.py
├── config.ini
├── config_storage.py
├── db
│   └── certs.db
├── dbtypes.py
├── dummy.key
├── dumps
├── enums.py
├── envelope.py
├── est_cert_chain.p7
├── est_chain.p7
├── est_signed_cert.pem
├── extensions.py
├── FETCH_HEAD
├── html_templates
│   ├── api.html
│   ├── app.log
│   ├── ca.html
│   ├── _certificate_detail.html
│   ├── config.html
│   ├── edit_profile.html
│   ├── generate_csr.html
│   ├── generate_key.html
│   ├── index.html
│   ├── inspect.html
│   ├── layout.html
│   ├── list_certificates.html
│   ├── list_certificates.html.171125
│   ├── list_certificates.html.delete
│   ├── list_csrs.html
│   ├── list_keys.html
│   ├── list_profiles.html
│   ├── list_rendered.html
│   ├── list_templates.html
│   ├── logs.html
│   ├── profile_file.html
│   ├── profile_result.html
│   ├── rendered_file.html
│   ├── rendered_template.html
│   ├── server_ext.html
│   ├── sign.html
│   ├── template_form.html
│   ├── va.html
```

```
│   ├── view_csr.html
│   ├── view.html
│   └── view_key.html
├── logs
│   ├── server.log
│   └── server.log.1
├── pki-https
│   ├── pikachu_issued_https.crt
│   ├── pikachu_issued_https.key
│   ├── tls.cert.pem
│   └── tls.key.pem
├── pki-misc
│   ├── crl.pem
│   ├── server_ext.cnf
│   └── validity.conf
├── pki-root
│   └── rad_ca_root.crt
├── pki-server-2.code-workspace
├── pki-subca
│   ├── rad_ca_sub_ec.crt
│   ├── rad_ca_sub_ec.key
│   ├── rad_ca_sub_rsa.crt
│   ├── rad_ca_sub_rsa.key
│   ├── rad_chain_ec.crt
│   └── rad_chain_rsa.crt
├── __pycache__
├── README.md
├── static
│   ├── favicon-16x16.png
│   ├── favicon-32x32.png
│   ├── help.pdf -> PKI-Hands-On1.03.pdf
│   ├── PKI-Hands-On1.02.pdf
│   └── PKI-Hands-On1.03.pdf
├── version.txt
├── x509_keys.py
├── x509_profiles
│   ├── 6w_ca_root_2.cnf
│   ├── 6w_ca_root_2x.cnf
│   ├── 6w_ca_root.cnf
│   ├── 6w_ca_root_x.cnf
│   ├── bitbucket_server.cnf
├── x509_profiles.py
├── x509_requests.py
└── x509_templates
    ├── 6w_ca_root_2.cnf.j2
    ├── 6w_ca_root.cnf.j2
    ├── 6w_ca_root_ext.cnf.j2
    ├── 6w_ca_sub.cnf.j2
    ├── crl_ocsp_ext_alt.cnf.j2
    ├── crl_ocsp_ext.cnf.j2
    ├── eon_server_ext.cnf.j2
```

## Complementary

Following installation of commands and libraries essential for testing and using the server

### *SSCEP*

Rocky 9 lack OS repository installation there for it needed to be complied, linked and installed

```
git clone https://github.com/zhaozg/openscep.git
cd openscep
chmod u+x configure
./configure

make

sudo make install
```

Note : If missing libraries install using  the OS dnf

### *EST-Client*

From repo https://github.com/globalsign/est

```
sudo dnf install -y golang

git clone https://github.com/globalsign/est.git

go install github.com/globalsign/est/cmd/estclient@latest

sudo cp go/bin/estclient   /usr/local/bin/estclient
```

### *Quantum Safe Algorithm*

Quantum safe algorithms successful installation requires openssl 3.x and other development packages

Prerequisite installation of Ninja

```
sudo dnf config-manager --set-enabled crb
sudo dnf install -y epel-release
sudo dnf install -y ninja-build

sudo dnf groupinstall -y "Development Tools"
sudo dnf install -y cmake ninja-build git openssl-devel libffi-devel
```

install quantum resistant algorithms (like Dilithium, Falcon, Kyber, and SPHINCS+)

```
git clone https://github.com/open-quantum-safe/oqs-provider.git
cd oqs-provider
./scripts/fullbuild.sh
sudo cmake --install _build
```

find the openssl.cnf file  and add it manually

```
ls -l /etc/ssl/openssl.cnf
lrwxrwxrwx. 1 root root 24 Aug 21  2024 /etc/ssl/openssl.cnf ->
/etc/pki/tls/openssl.cnf

# add the new provider

sudo vi /etc/pki/tls/openssl.cnf

[provider_sect]
default = default_sect
oqsprovider = oqsprovider_sect
[default_sect]
activate = 1
[oqsprovider_sect]
activate = 1
```

test the openssl

```
#run command
openssl list -providers
#output
Providers:
  default
    name: OpenSSL Default Provider
    version: 3.2.2
    status: active
  oqsprovider
```

```
    name: OpenSSL OQS Provider
    version: 0.8.1-dev
    status: active
```

we can see that oqsprovider provider is active

### *MQTTs broker*

Using distribution mosquitto  with docker compose

```
mkdir mqtt
cd mqtt


cat > docker-compose.yml <<EOL


version: '3'
services:
  mosquitto:
    image: eclipse-mosquitto:latest
    container_name: mosquitto
    network_mode: "host"  # Use host network mode
    ports:
      - "1883:1883"    # Default MQTT
      - "2883:2883"    # Secure MQTT with TLS
      - "9001:9001"    # WebSocket (if needed)
    volumes:
      - ./mosquitto.conf:/mosquitto/config/mosquitto.conf
      - ./mosquitto/certs:/mosquitto/certs
    restart: unless-stopped
EOL
```

make configuration file

```
cat > mosquitto.conf <<EOL

per_listener_settings true

listener 1883 0.0.0.0
allow_anonymous true
log_type all

listener 2883 0.0.0.0
cafile /mosquitto/certs/CA.cert
certfile /mosquitto/certs/est_mqtt_server_1.pem
```

```
keyfile /mosquitto/certs/est_mqtt_server_1.key

crlfile /mosquitto/certs/crl.pem

require_certificate true
use_identity_as_username true
log_type all
EOL
```

populate certification directory

```
mkdir certs
cd certs
```

with files

```
est_mqtt_server_1.pem
est_mqtt_server_1.key
crl_client_2.pem
CA.cert
```

By executing the following command Server is up and running

```
cd pki-srever-2
python app.py
```

all logs are written to the stdout and file output.log

## 6.2 Generate Root and intermediate certification

Every CA server is based on Root certificate and one or more intermediate (sub) certificates

Following procedure on how to generate a root and intermediate certificates for the CA server

## Generate Root Certificate

Create root EC key

```
openssl ecparam -name prime256v1 -genkey -noout -out rad_ca_root.key
```

Prepare certificate configuration file

```
cat > rad_ca_root.cnf <<EOL
# CA Certificate Configuration Template for Root ECC Certificates
[ req ]
default_bits       = 4096
default_md         = sha256
prompt             = no
distinguished_name = dn
x509_extensions    = v3_ca

[ dn ]
C  = IL
ST = TLV
L  = Tel Aviv
O  = RAD
OU = RD
CN = RAD Test ECDSA

[ v3_ca ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints       = critical, CA:true, pathlen:1
EOL
```

Self-sign root certificate

```
openssl req -config rad_ca_root.cnf -key rad_ca_root.key -new -x509 -days
3650 -sha256 -out rad_ca_root.crt
```

Prepare signing server extension configuration file

```
cat > ca_root_ext.cnf <<EOL
[ v3_intermediate ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = keyCertSign, cRLSign
crlDistributionPoints  = URI:https://pikachu-ca.rnd-rad.com/downloads/crl
EOL
```

# Generating Intermediate (Sub) Certificate

### *EC based Key Certificate*

Create sub key

```
openssl ecparam -name prime256v1 -genkey -noout -out rad_ca_sub_ec.key
```

Prepare certificate configuration file

```
cat > rad_ca_sub_ec.cnf<<EOL
[ req ]
default_bits       = 2048
default_md         = sha256
prompt             = no
distinguished_name = dn
req_extensions     = v3_intermediate
[ dn ]
C  = IL
ST = TLV
L  = Tel Aviv
O  = RAD
OU = RD
CN = RADSubTestECDSA
[ v3_intermediate ]
subjectKeyIdentifier   = hash
#authorityKeyIdentifier = keyid,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = keyCertSign, cRLSign
EOL
```

Generate Certificate request

```
openssl req -new -config rad_ca_sub_ec.cnf  -key rad_ca_sub_ec.key -out
rad_ca_sub_ec.csr
```

Sign certificate

```
openssl x509 -req -in rad_ca_sub_ec.csr -CA rad_ca_root.crt -CAkey
rad_ca_root.key -CAcreateserial -out rad_ca_sub_ec.crt -days 3650 -sha256 -
extfile ca_root_ext.cnf -extensions v3_intermediate
```

### *RSA based Key Certificate*

Create sub key

```
openssl genpkey -algorithm RSA -out rad_ca_sub_rsa.key -pkeyopt
rsa_keygen_bits:4096
```

Prepare certificate configuration file

```
cat > rad_ca_sub_rsa.cnf<<EOL
[ req ]
default_bits       = 4096
default_md         = sha256
prompt             = no
distinguished_name = dn
req_extensions     = v3_intermediate
[ dn ]
C  = IL
ST = TLV
L  = Tel Aviv
O  = RAD
OU = RD
CN = RADSubTestECDSA
[ v3_intermediate ]
subjectKeyIdentifier   = hash
#authorityKeyIdentifier = keyid,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = keyCertSign, cRLSign
EOL
```

Generate certificate request

```
openssl req -new -config rad_ca_sub_rsa.cnf  -key rad_ca_sub_rsa.key -out
rad_ca_sub_rsa.csr
```

Sign certificate

```
openssl x509 -req -in rad_ca_sub_rsa.csr -CA rad_ca_root.crt -CAkey
rad_ca_root.key -CAcreateserial -out rad_ca_sub_rsa.crt -days 3650 -sha256 -
extfile ca_root_ext.cnf -extensions v3_intermediate
```

### *Copy keys Certificates*

The Certificates and Keys must be located according to the attributes in the CA server configuration file (config.ini)

```
PROJ_DIR=~/pki-server-2
Cp rad_ca_root.key rad_ca_root.crt $PROJ_DIR/pki-root/
cp rad_ca_sub_ec.crt rad_ca_sub_rsa.crt rad_ca_sub_ec.key rad_ca_sub_rsa.key
$PROJ_DIR/pki-subca

cd $PROJ_DIR

cat pki-subca/rad_ca_sub_ec.crt pki-root/rad_ca_root.crt > pki-subca/rad_chain_ec.crt
cat pki-subca/rad_ca_sub_rsa.crt pki-root/rad_ca_root.crt > pki-
subca/rad_chain_rsa.crtv
```

## 6.3 Configuration

The Pikachu CA server is configured via the config.ini file.
This file controls CA selection, network ports, logging, LDAP/Vault integration, SCEP behaviour, and all path locations.

**[DEFAULT]**

- **SECRET_KEY** – Flask secret key used for session security.
- **http_port** – HTTP port used for unauthenticated OCSP responses (default: 80).
- **max_idle_time** – Maximum user idle time before automatic logout (for example: 10h, 4d, 20m).
- **allow_self_registration** – Allow/disallow self-registration ( none LDAP system)
- **api_token_default_validity** – token for automatic tests validity time
- **api_token_length** – token for automatic tests length
- **tests_api_token** – token for automatic tests name

**[LOGGING]**

- **log_level** – Log verbosity: TRACE, DEBUG, INFO, WARNING, ERROR, or CRITICAL.
- **log_file** – Path to the main server log file.

**[CA]**

Controls which sub-CA is used and where CA material is stored.

- **mode** – Default subordinate CA to use: EC or RSA.
- **SUBCA_KEY_PATH_EC / SUBCA_CERT_PATH_EC / CHAIN_FILE_PATH_EC** – Paths for EC sub-CA private key, certificate, and chain file.
- **SUBCA_KEY_PATH_RSA / SUBCA_CERT_PATH_RSA / CHAIN_FILE_PATH_RSA** – Paths for RSA sub-CA private key, certificate, and chain file.
- **ROOT_CERT_PATH** – Path to the root CA certificate.

**[VAULT]**

Optional HashiCorp Vault integration for key and signing operations.

- **enabled** – Enable or disable Vault (true / false).
- **address** – Vault server URL (for example http://127.0.0.1:8200).

- **role_id**, **secret_id** – AppRole credentials (typically provided via environment variables).
- **pki_rsa_path**, **pki_ec_path** – Vault PKI mount paths for RSA and EC.
- **transit_path** – Vault transit engine path for custom signing.
- **timeout** – Connection timeout (seconds).
- **retry_attempts** – Number of retries on Vault connection failure.
- **verify_ssl** – Whether to verify Vault's TLS certificate.
- **ca_cert_path** – CA certificate used to verify Vault when verify_ssl is enabled.
- **role_scep**, **role_est**, **role_default** – Vault roles used for SCEP, EST, and general operations.

**[LDAP]**

Optional LDAP integration for user management.

- **LDAP_HOST / LDAP_PORT** – LDAP server address and port.
- **BASE_DN** – Base DN for LDAP searches.
- **PEOPLE_DN** – DN under which user entries are stored.
- **ADMIN_DN / ADMIN_PASSWORD** – Bind DN and password for the LDAP admin account.
- **enabled** – Enable or disable LDAP authentication (true / false).

**[SCEP]**

Controls the Simple Certificate Enrollment Protocol behaviour.

- **enabled** – Enable or disable SCEP support.
- **serial_file** – File used to persist the SCEP serial counter across restarts.
- **dump_dir** – Directory where raw SCEP requests are stored (optional).
- **challenge_password_enabled** – Enables challenge-password protection for SCEP enrollment.
- **challenge_password_validity** – Validity period for generated challenge passwords (for example 60m, 2m).

**[HTTPS]**

Configuration for the main HTTPS interface (UI and APIs).

- **ssl_cert** – Path to the HTTPS server certificate.
- **ssl_key** – Path to the HTTPS private key.
- **port** – HTTPS listen port (default 443).

**[TRUSTED_HTTPS]**

Configuration for the mTLS / “trusted” HTTPS port.

- **trusted_ssl_cert** – Certificate used for the trusted HTTPS endpoint.
- **trusted_ssl_key** – Private key for the trusted HTTPS endpoint.
- **trusted_port** – Trusted HTTPS port (default 4443).

**[PATHS]**

Filesystem locations for core PKI artifacts.

- **crl_path** – Path to the CRL file generated by the CA.
- **server_ext_cfg** – Path to the server extension configuration file.
- **validity_conf** – Path to the validity configuration file (default certificate lifetimes).
- **db_path** – Path to the SQLite database (certs.db).

```
[DEFAULT]
# general Flask settings
SECRET_KEY = *******
# HTTP port for unauthenticated OCSP
http_port = 80
# Maximum idle time before automatic logout (e.g., "10h" or "4d" or "20m")
max_idle_time = 30m
# Allow/disallow self-registration (controls /users/register route and Register link)
allow_self_registration = false

api_token_default_validity = 60d
api_token_length = 64
tests_api_token = tests_api_token

[LOGGING]
# Logging level: TRACE (5), DEBUG (10), INFO (20), WARNING (30), ERROR (40), CRITICAL (50)
# TRACE: Very verbose - includes idle checks, session tracking
# DEBUG: Detailed diagnostic information
# INFO: General informational messages
# WARNING: Warning messages
# ERROR: Error messages
# CRITICAL: Critical errors
log_level = INFO
log_file = logs/server.log


[CA]
# Which subordinate CA to use by default: "EC" or "RSA"
mode = RSA
# Paths for both modes; the get_ca_config() helper below will pick the right
SUBCA_KEY_PATH_EC    = pki-subca/rad_ca_sub_ec.key
SUBCA_CERT_PATH_EC   = pki-subca/rad_ca_sub_ec.crt
CHAIN_FILE_PATH_EC   = pki-subca/rad_chain_ec.crt
SUBCA_KEY_PATH_RSA   = pki-subca/rad_ca_sub_rsa.key
SUBCA_CERT_PATH_RSA  = pki-subca/rad_ca_sub_rsa.crt
CHAIN_FILE_PATH_RSA  = pki-subca/rad_chain_rsa.crt
ROOT_CERT_PATH       = pki-root/rad_ca_root.crt

[VAULT]
# HashiCorp Vault integration for CA key isolation
# Set enabled=false to use traditional file-based keys (current behavior)
# Set enabled=true to use Vault PKI engine (enhanced security with key isolation)
enabled = false

# Vault server connection settings (required when enabled=true)
# For same server: http://127.0.0.1:8200 (dev mode without TLS)
# For separate server: https://192.168.1.20:8200
address = http://127.0.0.1:8200

# Authentication credentials (read from environment variables for security)
# Set these as environment variables: VAULT_ROLE_ID and VAULT_SECRET_ID
role_id = ${VAULT_ROLE_ID}
secret_id = ${VAULT_SECRET_ID}

# PKI mount paths in Vault
pki_rsa_path = pki-subca-rsa
pki_ec_path = pki-subca-ec
```

```
# Transit engine path for custom signing operations (OCSP, PQC)
transit_path = transit

# Connection settings
timeout = 30
retry_attempts = 3

# TLS settings
verify_ssl = true
ca_cert_path =

# Vault roles for different operations
role_scep = scep-enrollment
role_est = est-enrollment
role_default = server-cert

[LDAP]
LDAP_HOST = 172.18.178.24
LDAP_PORT = 389
BASE_DN = dc=rnd-rad,dc=com
PEOPLE_DN = ou=People,dc=rnd-rad,dc=com
ADMIN_DN = cn=Manager,dc=rnd-rad,dc=com
ADMIN_PASSWORD = marketing
enabled = false

[SCEP]
# enable or disable SCEP entirely
enabled = true
# optional path to a file where we persist the serial across restarts
serial_file = pki-subca/serial.txt
# Dump directory for raw SCEP requests (optional)
dump_dir = pki-subca/dumps

# Challenge-password feature
challenge_password_enabled = true
# Validity period for challenge-passwords (default: 60m)
challenge_password_validity = 2m

[HTTPS]
# HTTPS certificate & key for your main CA UI
ssl_cert = pki-https/tls.cert.pem
ssl_key  = pki-https/tls.key.pem
port     = 443

[TRUSTED_HTTPS]
# HTTPS certificate & key for your main CA UI
#trusted_ssl_cert = pki-https/pikachu_issued_https.crt
trusted_ssl_cert = pki-https/pikachu_issued_https_localhost.crt
trusted_ssl_key  = pki-https/pikachu_issued_https.key
trusted_port     = 4443

[PATHS]
# everything else that was hard-coded
crl_path      = pki-misc/crl.pem
server_ext_cfg = pki-misc/server_ext.cnf
validity_conf  = pki-misc/validity.conf
```

```
db_path      = db/pikachu-ca.db
```

## 6.4 Run Server

### Shell Command

By executing the following command Server is up and running

```
cd pki-srever-2
nohup python app.py > app.log 2>&1 &
```

all logs are written to the stdout and file output.log

### OS Service

Create the main service: pikachu-ca.service

```
sudo tee /etc/systemd/system/pikachu-ca.service > /dev/null << 'EOF'
[Unit]
Description=Pikachu CA Python App
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=rocky
Group=rocky
WorkingDirectory=/home/rocky/pki-server-2
ExecStart=/usr/bin/python3.11 /home/rocky/pki-server-2/app.py
Restart=always
RestartSec=5s
Environment=PYTHONUNBUFFERED=1

# Allow binding to ports <1024
AmbientCapabilities=CAP_NET_BIND_SERVICE
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
NoNewPrivileges=true

StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
EOF
```

Create the healthcheck service: pikachu-ca-healthcheck.service

```
sudo tee /etc/systemd/system/pikachu-ca-healthcheck.service > /dev/null <<
'EOF'
[Unit]
Description=Healthcheck for pikachu-ca (restart if https://localhost fails)
After=pikachu-ca.service

[Service]
Type=oneshot
ExecStart=/usr/bin/bash -c '/usr/bin/curl -ksf --max-time 5
https://localhost/ || /usr/bin/systemctl restart pikachu-ca.service'
EOF
```

Create the healthcheck timer: pikachu-ca-healthcheck.timer

```
sudo tee /etc/systemd/system/pikachu-ca-healthcheck.timer > /dev/null <<
'EOF'
[Unit]
Description=Periodic healthcheck for pikachu-ca

[Timer]
OnBootSec=30s
OnUnitActiveSec=60s
Unit=pikachu-ca-healthcheck.service

[Install]
WantedBy=timers.target
EOF
```

Reload systemd to pick up the new units

```
sudo systemctl daemon-reload
```

Enable services & timer at boot

```
sudo systemctl enable pikachu-ca.service
sudo systemctl enable pikachu-ca-healthcheck.timer
```

Start everything now

```
sudo systemctl start pikachu-ca.service
sudo systemctl start pikachu-ca-healthcheck.timer
```

Check status / logs

```
# Main service status
sudo systemctl status pikachu-ca.service

# Timer and healthcheck
sudo systemctl status pikachu-ca-healthcheck.timer
sudo systemctl status pikachu-ca-healthcheck.service

# See timers
systemctl list-timers | grep pikachu

# Logs
journalctl -u pikachu-ca.service -f
journalctl -u pikachu-ca-healthcheck.service -f
```

server can be access using the URL https://pikachu-ca.iot-rad.com:4443

## 6.5 Ports

The application as development tool can be used in the following ways

| Capability | Configuration | Support Technology | Description |
|---|---|---|---|
| HTTPS without client certificate | [HTTPS]<br>Port = 443 | All | One-way TLS<br>Server-authenticated TLS<br>Standard HTTPS |
| HTTPS that *requires* client certificates | [TRUSTED_HTTPS]<br>trusted_port = 4443 | All | Mutual TLS (mTLS)<br>Two-way TLS<br>TLS with client authentication |

| | | | |
|---|---|---|---|
| HTTP | [DEFAULT]<br>http_port = 80 | OCSP<br>SCEP | For http older technologies. |

# 7 Enrollment Workflows

Enrollment is supported through three mechanisms:
**UI Signing**, **EST**, and **SCEP**.

## 7.1 UI Signing

Manual signing through the browser:

1. Paste a CSR into the text box or load one from pending CSR list.
2. Select a certificate profile.
3. Select a server extension (None, User-level, or System-level).
4. Click **Sign CSR**.

The newly issued certificate immediately appears in the **Certificates** table.

## 7.2 EST Enrollment

Supported operations:

- **/cacerts** – Retrieve CA chain
- **/simpleenroll** – Submit DER-encoded CSR to receive a PKCS#7 response

Supports:

- HTTPS
- Optional mTLS
- Vault signing (when enabled)

EST supports Vault-based signing, unlike SCEP.

## 7.3 SCEP Enrollment

SCEP supports:

- GetCACaps

- GetCACert
- PKIMessage enrollment

SCEP **does not support Vault**.
SCEP **always uses local file-based CA keys**, even when Vault mode is enabled.

## Challenge Password Support

If enabled in config.ini:

[SCEP]

challenge_password_enabled = true

SCEP requests must include a valid, unconsumed challenge password.

Users can generate challenged passwords using UI and API (token authentication)

```
curl -X POST \
  -H "Authorization: Bearer <API_TOKEN>" \
  http://<server DNS/IP>:< HTTP port>/api/challenge_passwords
```

Generates a challenge password tied to the token's user. Response: value, expires_at, validity, user_id.

# 8 Integrations

## 8.1 LDAP Integration

When enabled via [LDAP]:

- Login authentication uses LDAP directory.
- Local users remain supported when LDAP is disabled.
- User role is determined by system configuration.

Keys from README correctly integrated.

## 8.2 HashiCorp Vault Integration

Vault integration allows CA keys to be fully isolated from disk.

Capabilities:

- CA private keys never stored on server filesystem.
- All signing performed inside Vault PKI engines.
- Separate engines for **RSA** and **EC**.
- Automatic fallback to file-based keys if Vault unavailable.
- Supported enrollment methods:
    - **Web UI signing** → Yes
    - **EST** → Yes
    - **SCEP** → **No** (file-based only)

Important logs (from README) should be mentioned:

```
INFO [app] Vault integration is ENABLED
INFO [vault_client] Authenticated with Vault
```

## 8.3 MQTT TLS Integration

The CA can be used to:

- Issue server certificates for MQTT brokers.
- Issue client certificates for devices.
- Provide CRL for client revocation enforcement.

# 9 OCSP & CRL

## 9.1 Certificate Revocation List (CRL)

The VA page provides:

- List of revoked certificates
- Downloadable CRL via UI or API
- OpenSSL-based inspection

Used by MQTT or other TLS-based services for revocation enforcement.

## 9.2 OCSP Responder

OCSP provides real-time certificate status without downloading the CRL.

Supported statuses:

- **good**
- **revoked**
- **unknown**

```
openssl ocsp \
  -issuer subca.crt \
  -cert device.crt \
  -url http://localhost:80/ocsp
```

OCSP signing supports both Vault mode and file-based mode, depending on CA configuration.

To embed OCSP into certificates, configure AIA in server extensions (Section 6).

# 10 Post-Quantum Cryptography

Pikachu CA supports PQC algorithms through the Open Quantum Safe provider.

Available PQ algorithms:

- **mldsa44** – Dilithium2
- **mldsa65** – Dilithium3
- **mldsa87** – Dilithium5

Limitations:

- Must install oqsprovider
- PQC signing in the UI depends on using system OpenSSL (Python crypto libs do not support PQ signatures)

# 11 REST API Reference

The API page contains:

- CSR submission
- Certificate issuance and listing
- Key operations
- CRL and CA chain download
- OCSP endpoint
- SCEP + EST URLs
- Enrollment examples (curl + estclient)

A note from README:

API examples for OCSP, SCEP, EST, and REST endpoints appear in the Web UI under **APIs**.

# 12 User Management

**Roles:**

- **Admin**
- **User**

**Capabilities:**

- Admin can manage all users:
  - Add
  - Edit status
  - Change role
  - Logout user
  - Delete user
- Users can:
  - Change their own password
  - View/edit their own resources

**Registration Workflow:**

- Users register through the UI.
- Admin approves pending accounts.
- User status enforced:
  - Pending
  - Active
  - Suspended
  - Deactivated

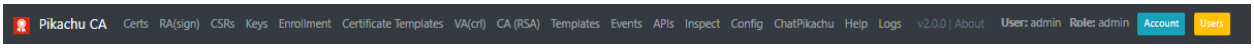
**Events Page:**

- Displays login, logout, failed login attempts, role changes.
- Filterable and sortable table.

# 13 Detailed Features

## 13.1 Top Navigation Layout

The top navigation bar provides quick access to all major Pikachu CA server functions. Available menu items depend on user role (Admin/User) and authentication mode.




**Certs**

View, download, revoke, or delete issued certificates.

**RA (sign)**

Manually sign CSRs to issue certificates.

**CSRs**

Create, view, download, or delete certificate signing requests.

**Keys**

Generate, view, download, and delete private keys.

**Enrollment**

Manage EST/SCEP enrollment policies and extension settings.

**Certificate Templates**

Create, edit, view, or delete certificate profile templates.

**VA (crl)**

View revoked certificates and download the CRL.

**CA (RSA/EC)**

Display root and subordinate CA certificates and chain.

**Templates *(Admin)***

Render new certificate templates from Jinja templates.

**Events**

Audit log of all system actions (create, delete, revoke, update).

**APIs**

List all CA APIs including EST, SCEP, Manual RA, CRL, and OCSP.

**Inspect**

Analyze PEM or Base64 DER data (certs, keys, CSRs, CRLs, etc.).

**Config**

View the server's active configuration (config.ini).

**ChatPikachu**

Built-in AI assistant for CA operations.

**Help**

Open the system help PDF.

**Logs**

Live server log viewer with filtering.

**Version**

Displays current server version. v2.0.0

**User / Role**

Shows logged-in user and role.

**Account**

Change password(none LDAP user), Manage API tokensand log out.

**Users *(Admin)***

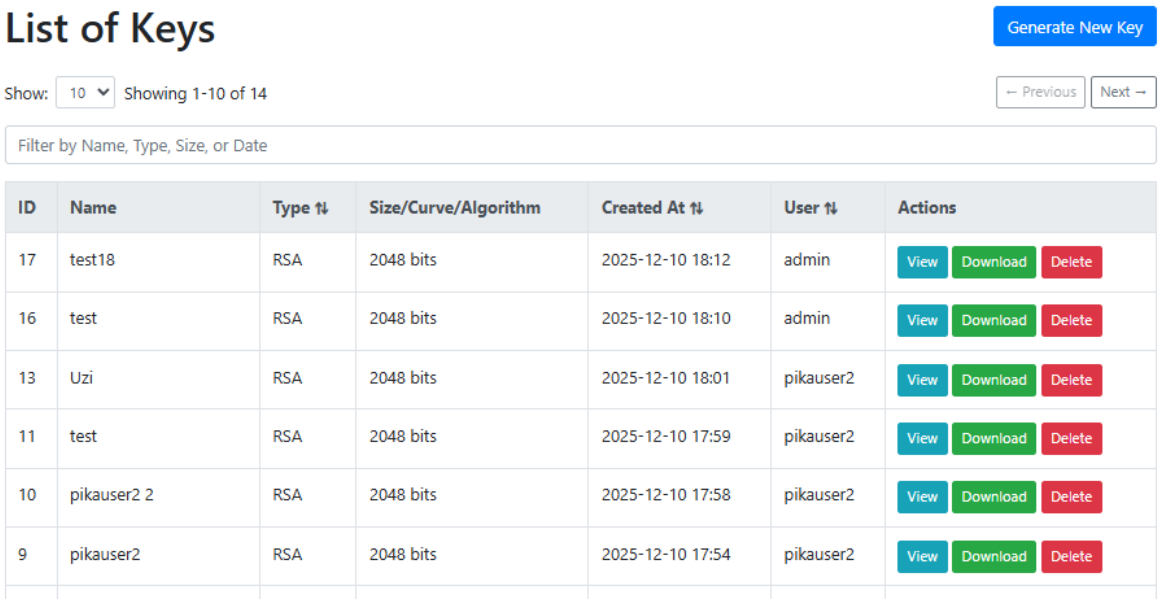
Manage user accounts, roles, states, and events.

# 13.2 Keys Management

## List of Keys

The **List of Keys** page displays all keys that have been created in the system and allows users to view, download, or delete them.

https://pikachu-ca.<sub domain>.com:4443/keys




### Page elements

- **Generate New Key** – Button in the top-right corner used to create a new key.
- **Show** dropdown – Controls how many rows are shown per page (for example, 10, 25, 50).
- **Pagination** – *Previous* / *Next* buttons to move between pages when there are more keys than can be shown on a single page.
- **Search / Filter box** – Free-text filter labeled **"Filter by Name, Type, Size, or Date"**. Typing in this field narrows the table to rows matching the entered value.

### Table columns

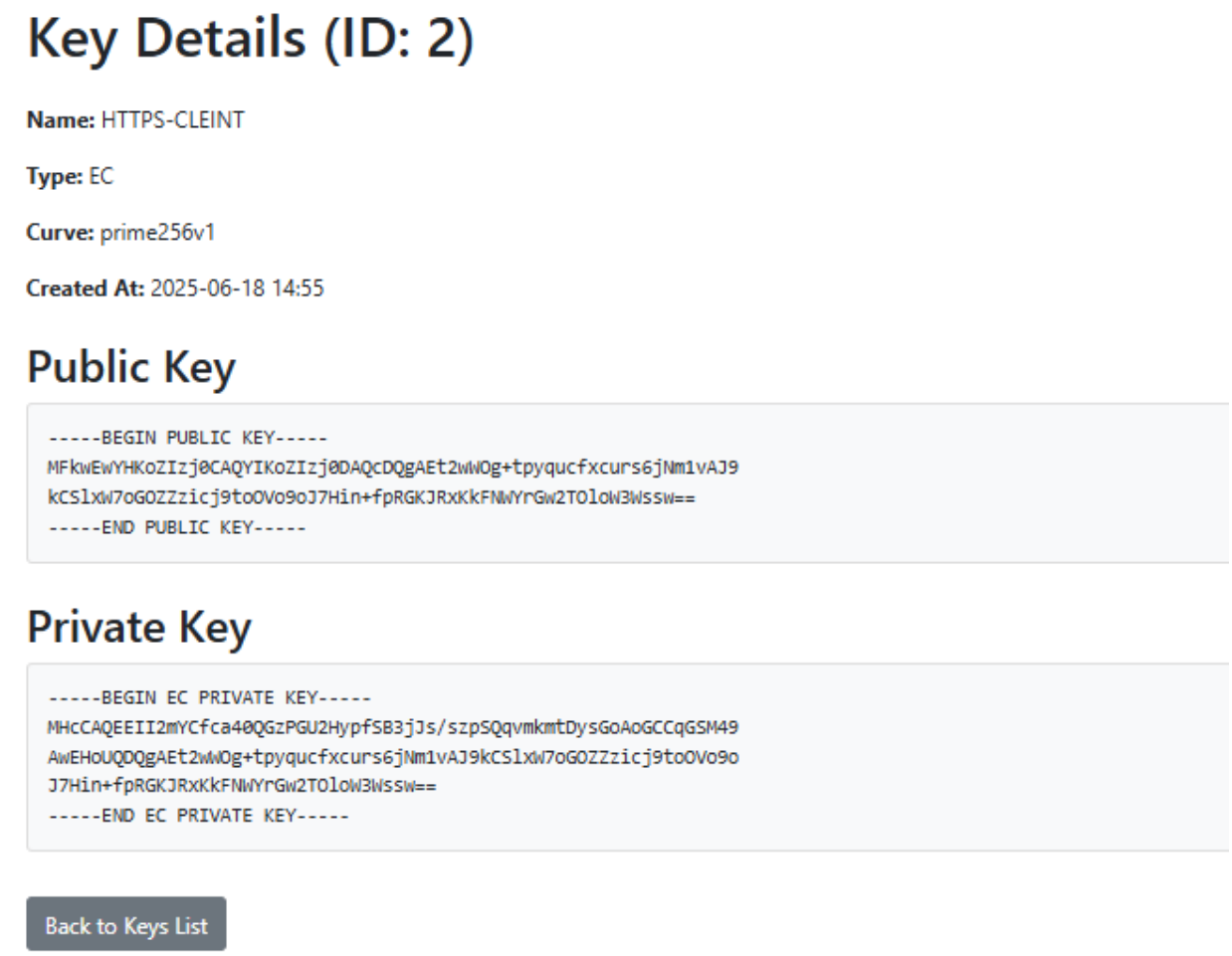
- **ID** – Internal numeric identifier of the key.
- **Name** – Friendly name given to the key (for example, test18, Uzi).
- **Type** – Key type, such as RSA, EC, or PQC (Post-Quantum).

- **Size/Curve/Algorithm** – Indicates key length or algorithm details, for example 2048 bits.
- **Created At** – Date and time when the key was created.
- **User** – Username that owns or created the key.
- **Actions** – Available operations for each key:
    - **View** – Opens a details page for the selected key, where the public and private key material can be inspected (subject to permissions).
    - **Download** – Downloads the key file for backup or use on another system.
    - **Delete** – Permanently removes the key from the system.

Users can sort the table by clicking on sortable column headers such as **Type**, **Created At**, or **User**, making it easy to locate specific keys or review recently created ones.

## View Key

The **View Key** page displays detailed information about a specific key selected from the **List of Keys** table. This page allows users to inspect both the public and private portions of the key for reference or export.




**Key Details Displayed**

- **Key ID**
  The internal identifier of the key (e.g., *ID: 2*).
- **Name**
  The user-assigned label for the key (e.g., *HTTPS-CLIENT*).
- **Type**
  Indicates whether the key is **RSA**, **EC**, or **PQC**.
- **Curve / Size / Algorithm**
  Displays key parameters depending on type:
  - EC keys show the selected curve (e.g., *prime256v1*).
  - RSA keys show the key size (e.g., *2048 bits*).

- PQC keys show the algorithm (e.g., *mldsa44*).

- **Created At**
  The date and time when the key was generated.

**Public Key**

A read-only text block showing the full PEM-encoded public key.
This key can be copied and used in CSRs, device configuration, or other cryptographic operations.

-----BEGIN PUBLIC KEY-----

...

-----END PUBLIC KEY-----

**Private Key**

A read-only text block showing the PEM-encoded private key.
Users can copy the key if needed for externally managed signing operations, devices, or testing tools.

-----BEGIN EC PRIVATE KEY-----

...

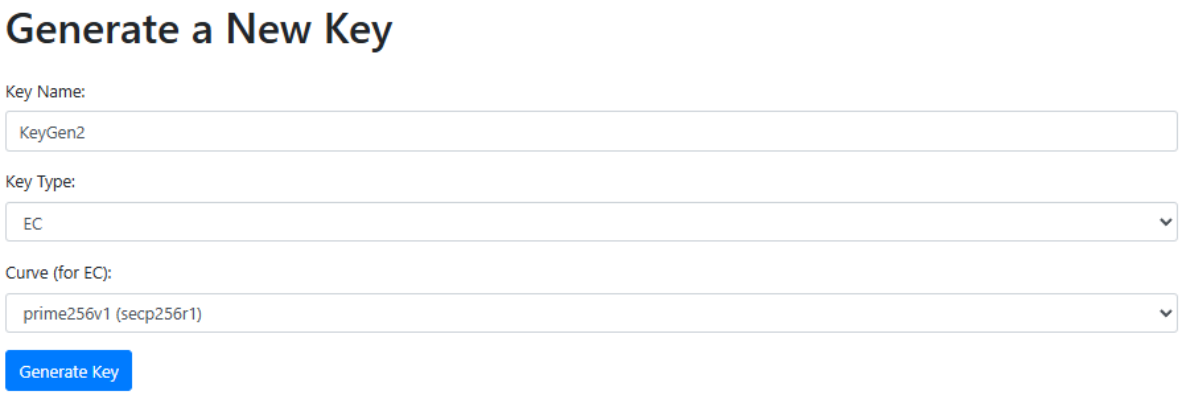
-----END EC PRIVATE KEY-----

**Security Note:**
Private keys displayed here should be handled carefully.
Exporting or copying private keys should be restricted to authorized users only.

## Generate a New Key

The Generate a New Key page allows users to create a new cryptographic key that can later be used for generating CSRs and issuing certificates.

https://pikachu-ca.<sub domain>.com:4443/generate




**Creation Fields on the page:**

**Key Name**
A user-defined label for the new key.
This name will appear in the List of Keys table and helps identify the key for future use (e.g., KeyGen2).

**Key Type**
A dropdown menu used to select the type of key to generate.
Available options typically include:

1. **RSA Keys**

   Widely supported and commonly used for TLS and device authentication.

   Supported key sizes:

   a. 2048 bits – Standard security level

   b. 4096 bits – Higher security for long-term or sensitive deployments

2. **EC (Elliptic Curve) Keys**

   EC keys provide strong security with smaller key sizes and better performance.

   When EC is selected, the Curve dropdown appears, offering the following options:

   a. prime256v1 (secp256r1) – Most widely used NIST P-256 curve

   b. secp384r1 – Higher-security NIST P-384 curve

   c. secp521r1 – Highest NIST-level EC security

   **d.** secp256k1 – K-256 curve, commonly used in blockchain applications

3. **PQC (Post-Quantum Cryptography) Keys**

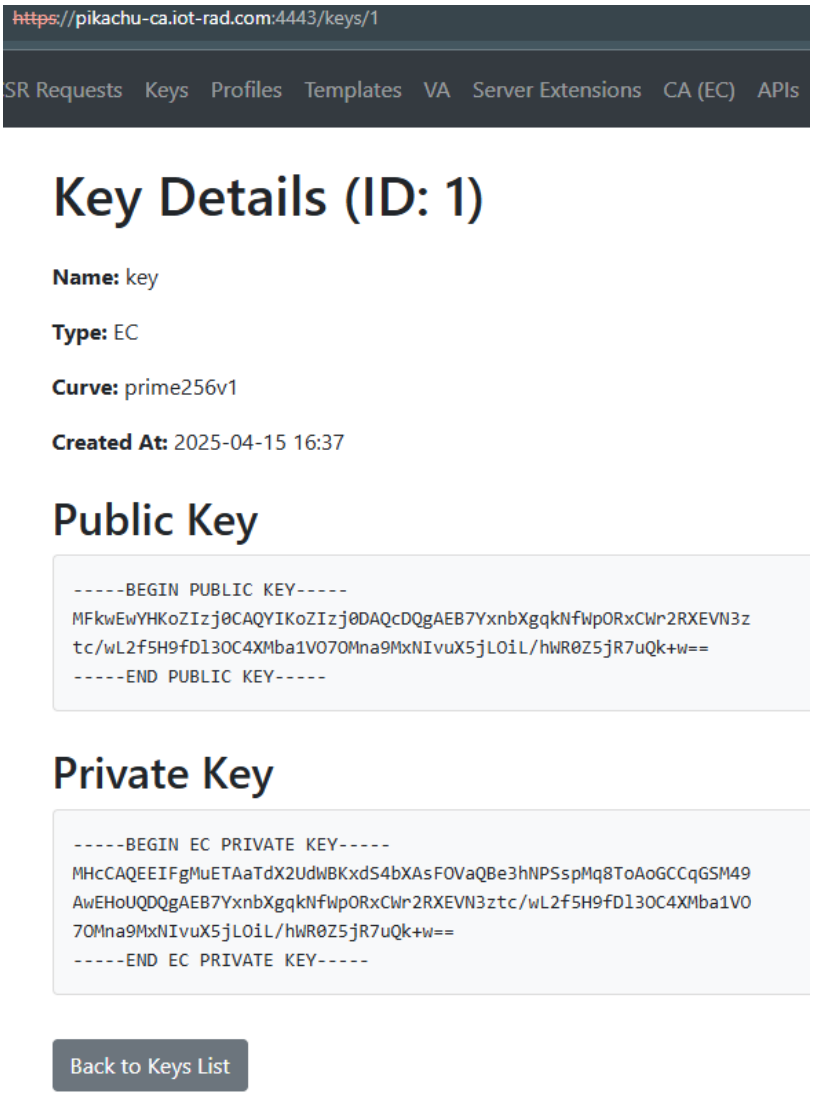
   Post-quantum algorithms designed to resist attacks from future quantum computers. When PQC is selected, the PQC Algorithm dropdown appears with:

   a. mldsa44 (Dilithium2 / NIST L1) – Baseline PQ security

   b. mldsa65 (Dilithium3 / NIST L3) – Stronger PQ protection

   c. mldsa87 (Dilithium5 / NIST L5) – Highest standard PQ strength

These algorithms require an environment configured with OpenSSL 3.x + oqsprovider.

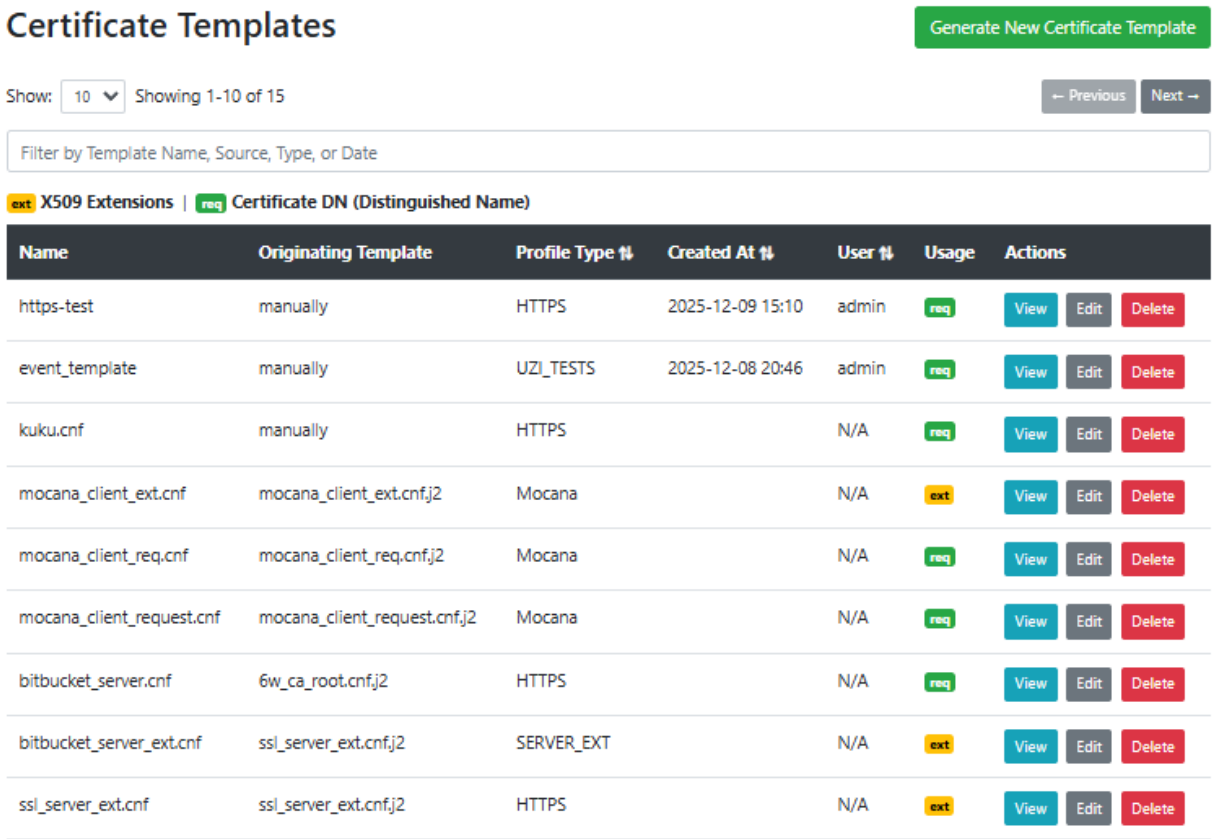


User can copy the Private and Public key data for any purpose

## 13.3 Certificate Templates

### Certificate Templates list

The **Certificate Templates** page displays all certificate templates available on the server.
Templates define the structure and attributes used when generating certificate profiles, including X.509 extensions and Distinguished Name (DN) fields. Templates may be created manually or derived from predefined Jinja-based template files.




**Page Overview**

- **Show** — Adjusts the number of displayed rows (e.g., 10, 25, 50).
- **Pagination Controls** — *Previous* / *Next* buttons navigate through multiple pages.
- **Filter Box** — Searches templates by name, source, type, or date.

At the top of the page **Generate New Certificate Template** Creates a new template manually or based on an existing template file.

Below the toolbar, the page highlights two major categories of template content:

- **X.509 Extensions**
- **Certificate DN (Distinguished Name)**

These labels indicate the two primary components used when rendering certificate profiles.

**Template List Columns**

Each row in the table represents an existing certificate template. The columns are:

- **Name**
  The template's identifier (e.g., *https-test*, *event_template*, *mocana_client_req.cnf*).
- **Originating Template**
  Indicates the source, such as:
  - **manually** — Created by a user from scratch
  - **filename.j2** — Generated from a Jinja certificate template file
- **Profile Type**
  Specifies the intended usage or category of the template (e.g., *HTTPS*, *Mocana*, *UZI_TESTS*).
- **Created At**
  Date and time when the template was created.
- **User**
  The user who created the template (e.g., *admin*, *N/A* for system-generated templates).
- **Usage**
  Indicates which parts the template provides:
  - **req** — Provides CSR configuration
  - **ext** — Provides X.509 extension configuration
    (Some templates provide both depending on their design.)
- **Actions**
  Available operations for managing templates:
  - **View** — Opens a detailed read-only view of the template content

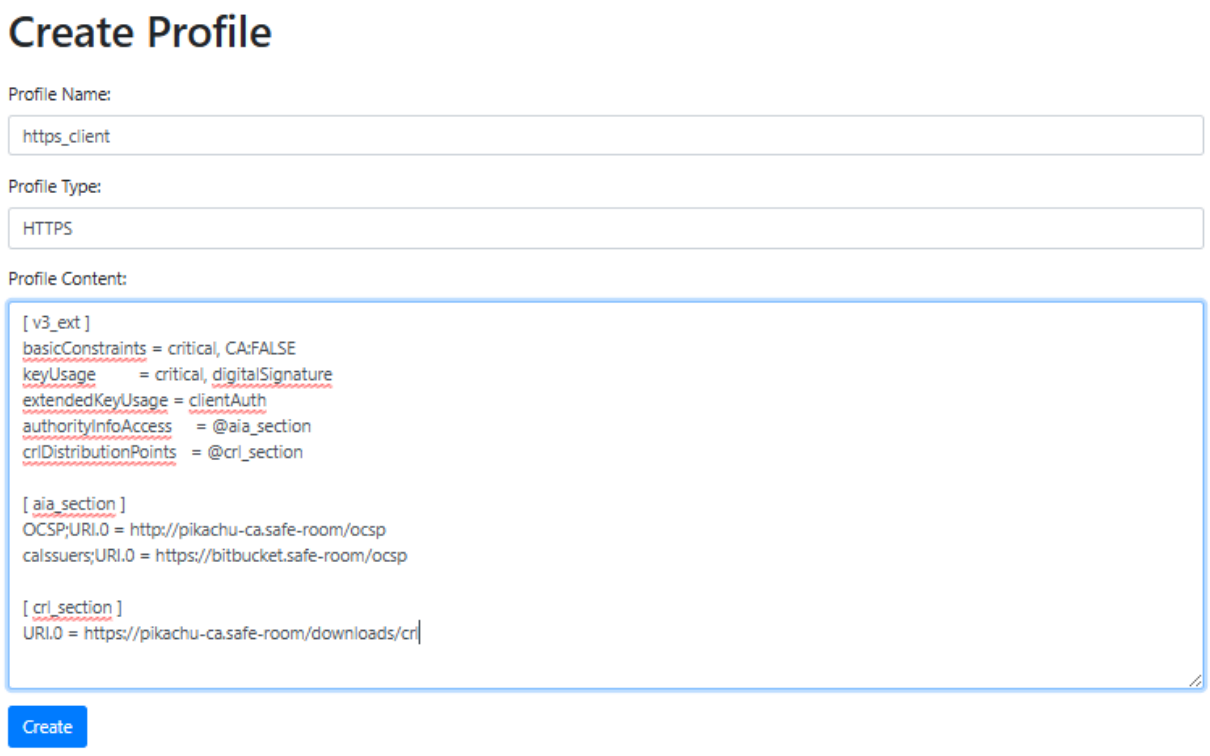
- **Edit** — Opens the template in editor mode for modification
- **Delete** — Removes the template from the system

**Typical Use Cases**

- Creating customized CSR templates for device onboarding
- Managing extension templates for specific certificate enrolment methods
- Revising DN structures for organizational standards
- Maintaining reusable templates that simplify certificate profile creation

## Generate New Certificate Template

The **Generate New Certificate Template** page allows users to create a new certificate template that can later be used to generate profiles for CSRs or X.509 extensions. Templates may be created manually or derived from existing template files, depending on the workflow.



**Fields on the Page**

**1. Profile Name**

A user-defined label for the new template.
This name will appear in the **Certificate Templates** list for easy identification.

**2. Profile Type**

Indicates the intended use of the template (e.g., *HTTPS*, *Mocana*, *Client*, etc.).
This classification helps organize templates and associate them with specific certificate workflows.

**3. Profile Content**

A free-form text area where the user enters the configuration for the certificate template.

The system determines the type of template based on the sections included in the content:

- CSR Configuration Templates ([req])

  If the content contains a **[req]** section, the system identifies the template as a **CSR configuration template**.
  Such templates define:

  - Distinguished Name (DN) fields
  - Request attributes
  - Key parameters
  - Required input for certificate signing

  Use this format when creating templates intended for CSR generation.

- 2. X.509 Extension Templates ([v3_ext])

  If the content contains a **[v3_ext]** section, the system identifies it as an **X.509 extension configuration template**.
  These templates specify certificate extensions such as:

  - Key usage
  - Extended key usage
  - Authority Info Access (AIA)
  - Subject Alternative Names (SAN)
  - CRL and OCSP endpoints

Use this format for templates that provide certificate extensions during certificate issuance.

## View Certificate Template


The **View Certificate Template** page shows the full configuration and basic details of a selected template.

- **Template Name** and **Type** are displayed at the top, along with the **Originating Template** (manual or based on a .j2 file).
- The **Template Content** appears in a read-only text block.
    - A template containing **[req]** defines CSR settings.
    - A template containing **[v3_ext]** defines X.509 extension settings.

Use **Back to Certificate Templates** to return to the main list.

### Edit Certificate Template

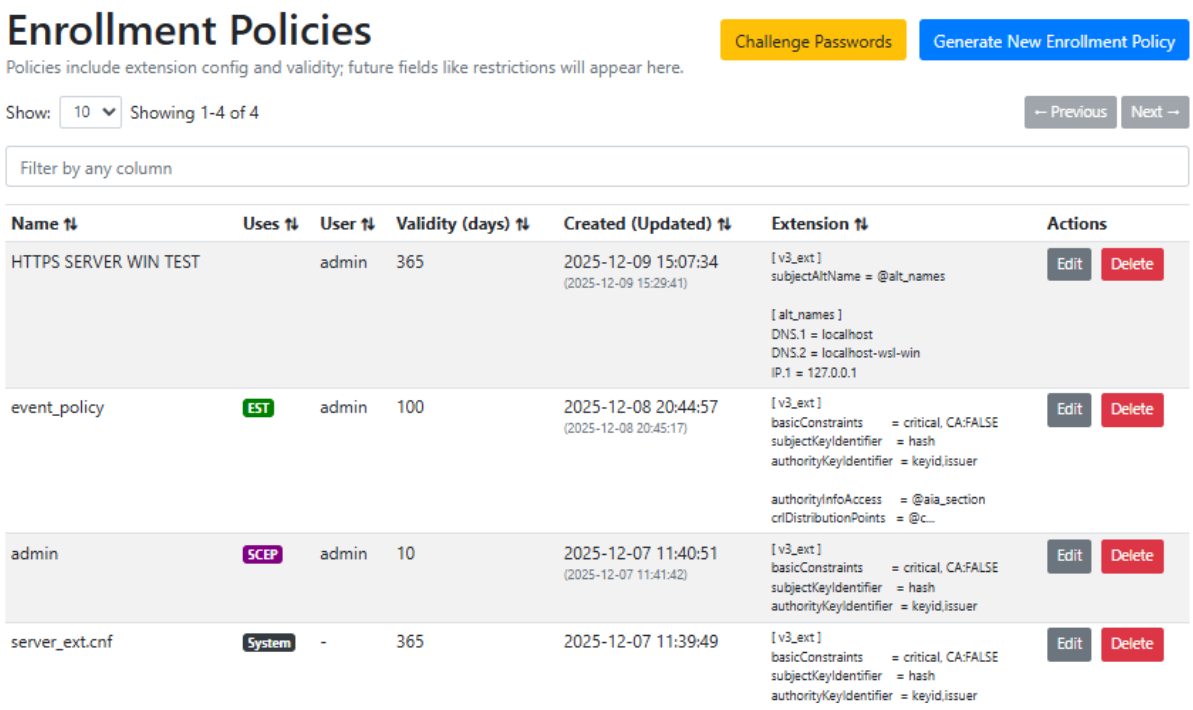
The **Edit Profile** page allows users to modify the content of an existing certificate profile.

- The **Profile Content** field displays the current configuration in an editable text area.
- Users may update CSR-related sections ([req]) or X.509 extension sections ([v3_ext]) depending on the profile's purpose.
- After making changes, click **Save** to update the profile and return to the profile list.

## 13.4 Enrollment

The **Enrollment** page displays all defined enrollment policies.
These policies control certificate validity, required extensions, and how CSR enrollment is handled for various use cases such as HTTPS, EST, SCEP, and system-level policies.

# Enrollment Policies List




### Page Features

- **Show** — Select how many policies to display per page.
- **Pagination** — *Previous* / *Next* buttons navigate through multiple pages.
- **Filter Box** — Allows filtering by any column (name, user, extension type, etc.).
- **Challenge Passwords** — Opens configuration for managing CSR challenge passwords.
- **Generate New Enrollment Policy** — Creates a new policy with custom validity and extension rules.

---

### Table Columns

Each row represents an enrollment policy and includes:

- **Name**
  Human-readable policy name (e.g., *HTTPS SERVER WIN TEST*, *event_policy*).
- **Uses**
  Indicates the enrollment flow the policy applies to, such as:

  - **EST** – Used for EST-based enrollments
  - **SCEP** – Used for SCEP requests
  - **System** – Built-in default or server-level policy

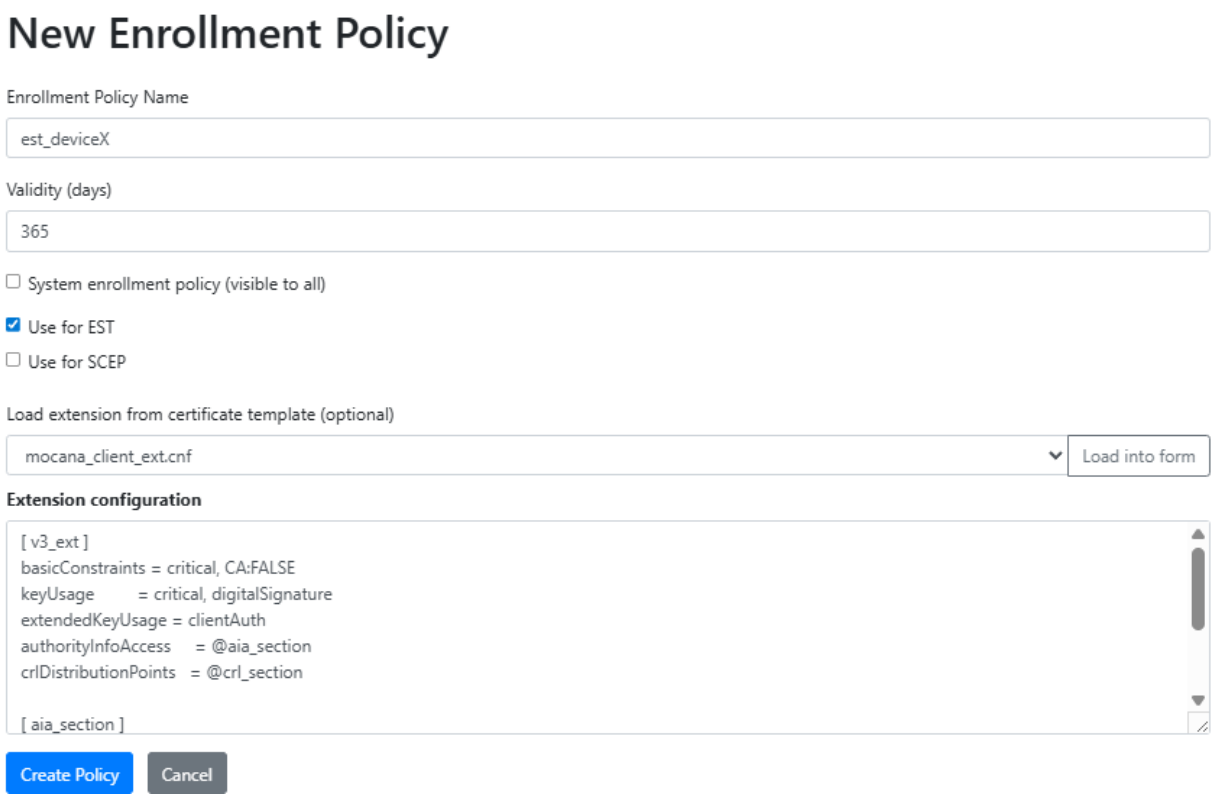
- **User**
  The user who created or owns the policy (e.g., *admin*).
- **Validity (days)**
  The certificate validity period enforced by this policy.
- **Created (Updated)**
  Shows when the policy was created and, if applicable, last modified.
- **Extension**
  Displays the X.509 extension block associated with the policy, such as:
  - [v3_ext] definitions
  - Adding Subject Alternative Name (SAN)
  - Adding Authority Information Access (AIA)
  - Adding Key Usage / Extended Key Usage (KU / EKU)
  - Adding Authority Information Access (AIA) for OCSP
  - Adding CRL Distribution Point (CDP) URL
  - Adding CA Issuers URL for retrieving the CA certificate chain

These fields define what extensions will be added to certificates issued under this policy.

- **Actions**
  - **Edit** — Modify the policy's validity, usage, or X.509 extension content.
  - **Delete** — Remove the policy from the system.

## Generate Enrollment Policy

The **New Enrollment Policy** page allows users to create a policy that defines how certificates will be issued for a specific enrollment workflow (EST, SCEP, or system-wide usage).




### Fields on the Page

- Enrollment Policy Name

  A user-chosen name that identifies the policy (e.g., est_deviceX).
  This name appears in the Enrollment Policies list.

- Validity (days)

  Defines how long the certificates issued under this policy will remain valid.

- System Enrollment Policy

  When enabled, the policy becomes visible and usable by all users.

- Use for EST / Use for SCEP

  Specifies which enrollment method(s) the policy applies to:

  - Use for EST — Policy applies to EST enrollment requests.
  - Use for SCEP — Policy applies to SCEP enrollment requests.
    (One or both options can be selected depending on requirements.)

- Load Extension From Certificate Template (optional)

  Users may optionally select an existing certificate template.
  Click Load into form to insert its extension block into the policy editor automatically.

  This helps quickly reuse predefined X.509 extension settings.

- Extension Configuration

  A text area containing the X.509 extension definitions to apply when signing certificates.

  Typical sections include:

- [v3_ext] — Defines X.509 extensions such as key usage, extended key usage, AIA, CRL distribution points, etc.

  Additional sections (e.g., [aia_section], [crl_section]) may appear depending on the template.

Users may edit these settings manually before creating the policy.

Create / Cancel

- Create Policy — Saves the new enrollment policy and returns to the policy list.
- Cancel — Discards changes and returns without creating a policy.

## Edit Enrollment Policy

The **Edit Enrollment Policy** page allows users to update an existing enrollment policy.

Users can modify:

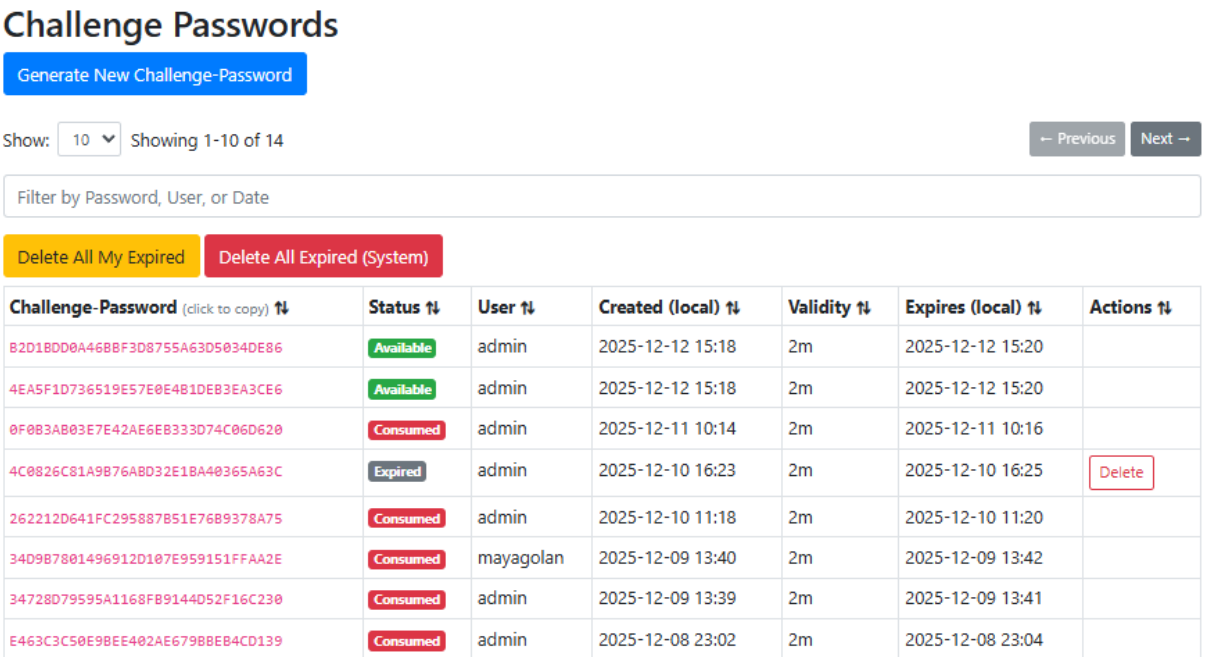
- **Policy Name**
- **Validity (days)**
- Whether the policy is **system-wide**, or used for **EST** or **SCEP**
- **Extension configuration**, including [v3_ext] and any supporting sections

An optional certificate template may be selected and loaded to replace or update the extension block.

Click **Save Changes** to apply updates or **Cancel** to return without saving.

## Challenge Passwords

The **Challenge Passwords** page lists all challenge-password tokens generated for certificate enrollment. These one-time passwords are typically used in SCEP or CSR-based workflows to authenticate enrollment requests.




### Page Features

- **Generate New Challenge-Password**
  Creates a new one-time password with a short validity window.
- **Show / Pagination**
  Adjusts the number of rows displayed and allows navigation through pages.
- **Filter Bar**
  Filters entries by password value, user, or date.
- **Delete All My Expired**
  Removes all expired challenge passwords belonging to the current user.

- **Delete All Expired (System)**
  Removes all expired passwords for all users (admin only).

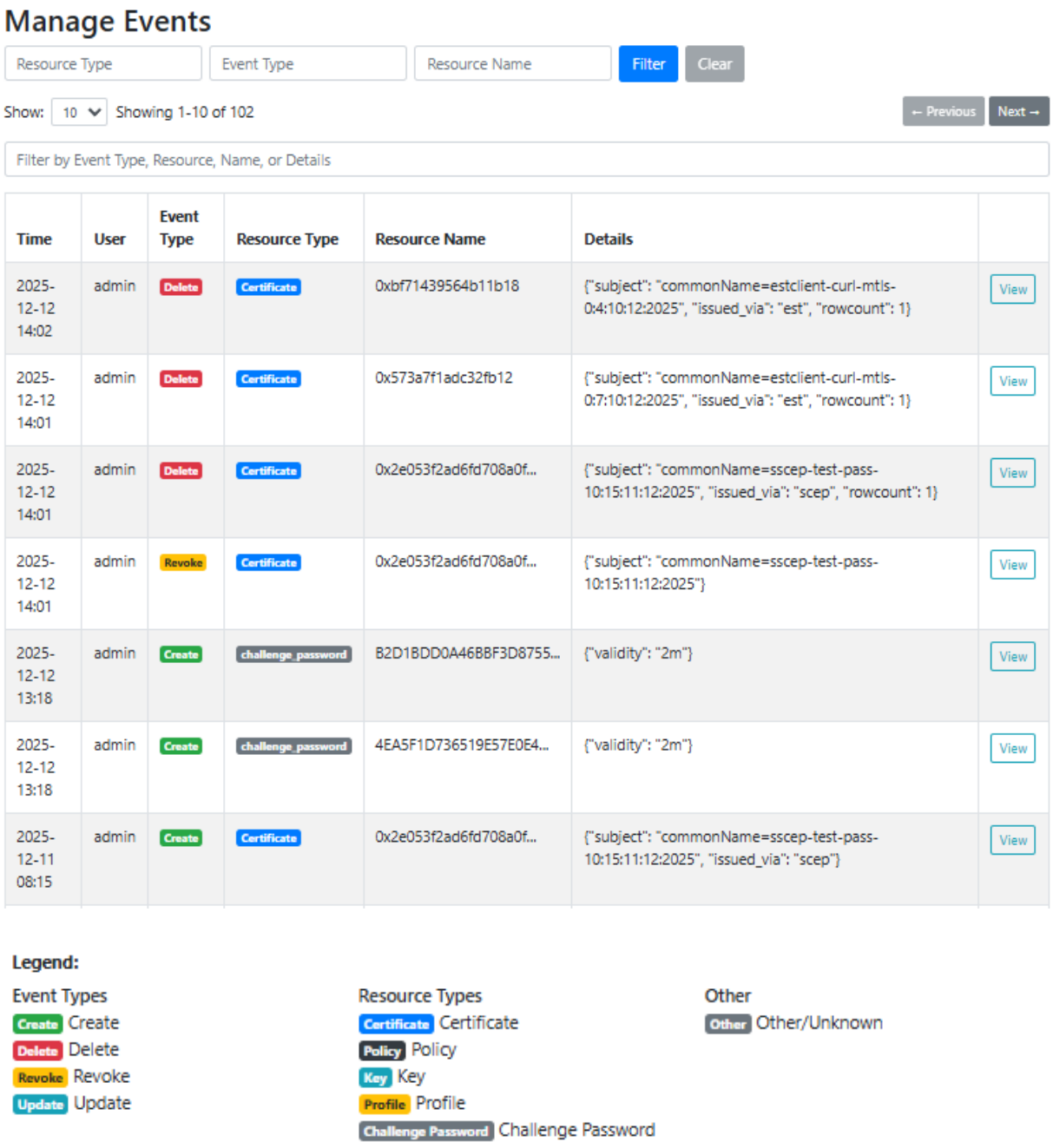
**Table Columns**

- **Challenge-Password** *(click to copy)*
  Displays the generated token. Clicking copies it to the clipboard.
- **Status**
  Indicates the state of each token:
  - **Available** – Valid and unused
  - **Consumed** – Already used for enrollment
  - **Expired** – No longer usable
- **User**
  The user who created the challenge password.
- **Created (local)**
  Local server timestamp of when the password was generated.
- **Validity**
  Duration for which the password is valid (e.g., *60m*).
- **Expires (local)**
  Local time when the password becomes invalid.
- **Actions**
  - **Delete** – Removes the selected password (typically used for for expired or consumed passwords).

The profile can be created either by editing or rendered from template

## 13.5 Events (System)

The **Manage Events** page provides a complete audit log of all system-level actions performed on resources such as certificates, keys, profiles, and enrollment policies. It is designed to help administrators track operations, troubleshoot changes, and maintain compliance visibility.



**Filtering**

You can filter events using:

- **Resource Type**
  (Certificate, Key, Profile, Policy, Challenge Password, Other)
- **Event Type**
  (Create, Delete, Revoke, Update)
- **Resource Name**
  Filter by subject, serial, name, or identifier.
- **Search Bar**
  Free-text search across event type, resource name, and details.

Pagination controls allow browsing through large event histories.

**Table Columns**

- **Time**
  Exact timestamp of the event.
- **User**
  The user who performed the action (actor).
- **Event Type**
  Categorizes the operation:
  - **Create** – A new resource was generated
  - **Delete** – A resource was removed
  - **Revoke** – A certificate was revoked
  - **Update** – A resource was modified
- **Resource Type**
  Indicates the kind of object affected:
  - Certificate
  - Key
  - Profile
  - Policy

  - Challenge Password
  - Other / Unknown

- **Resource Name**
  Identifier of the resource (e.g., certificate serial, key name, policy name).
- **Details**
  Additional JSON metadata, such as:
  - Subject DN
  - Issuance method (est, scep)
  - Validity fields
  - Row counts or system-side info
- **View**
  Opens a detailed view of the event's full log data.

**Legend**

**Event Types**

- **Create** – Resource created
- **Delete** – Resource deleted
- **Revoke** – Certificate revoked
- **Update** – Resource modified

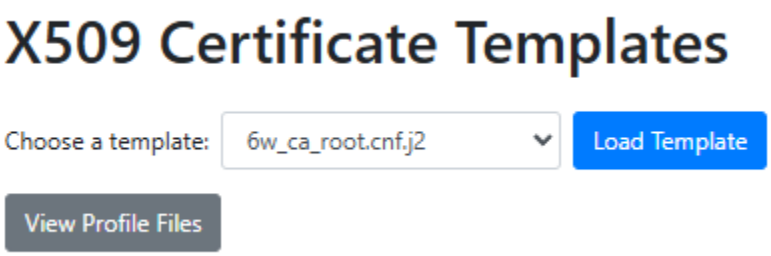
**Resource Types**

- **Certificate**
- **Policy**
- **Key**
- **Profile**
- **Challenge Password**
- **Other** – Unclassified/unknown events

# 13.6 Templates (admin role only)

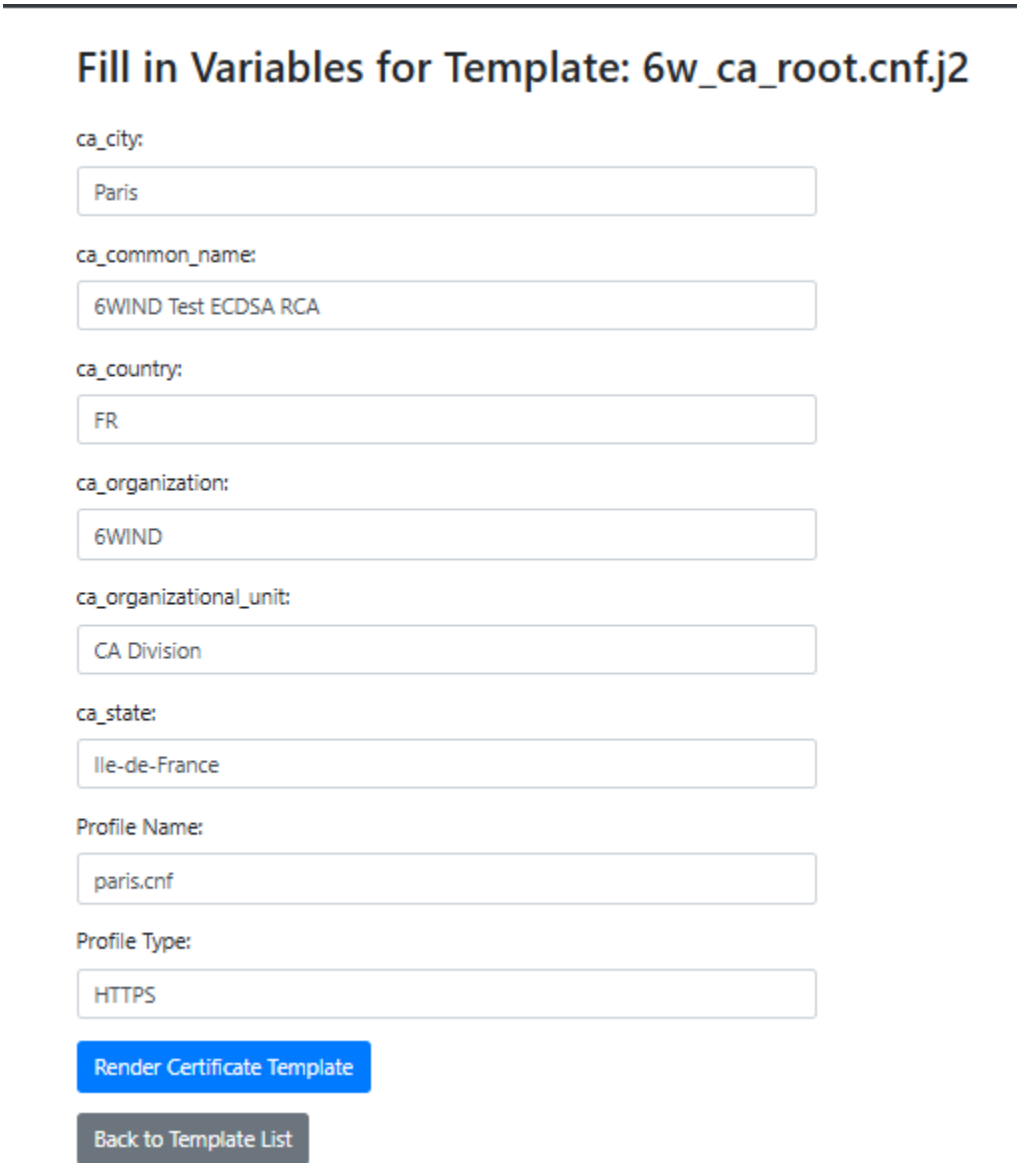
## Loading/Editing

The X509 Certificate Templates page allows users to work with Jinja-based certificate template files (.cnf.j2). These templates contain placeholders for CA information and certificate parameters that must be filled in before generating a usable profile.




**Template Actions**

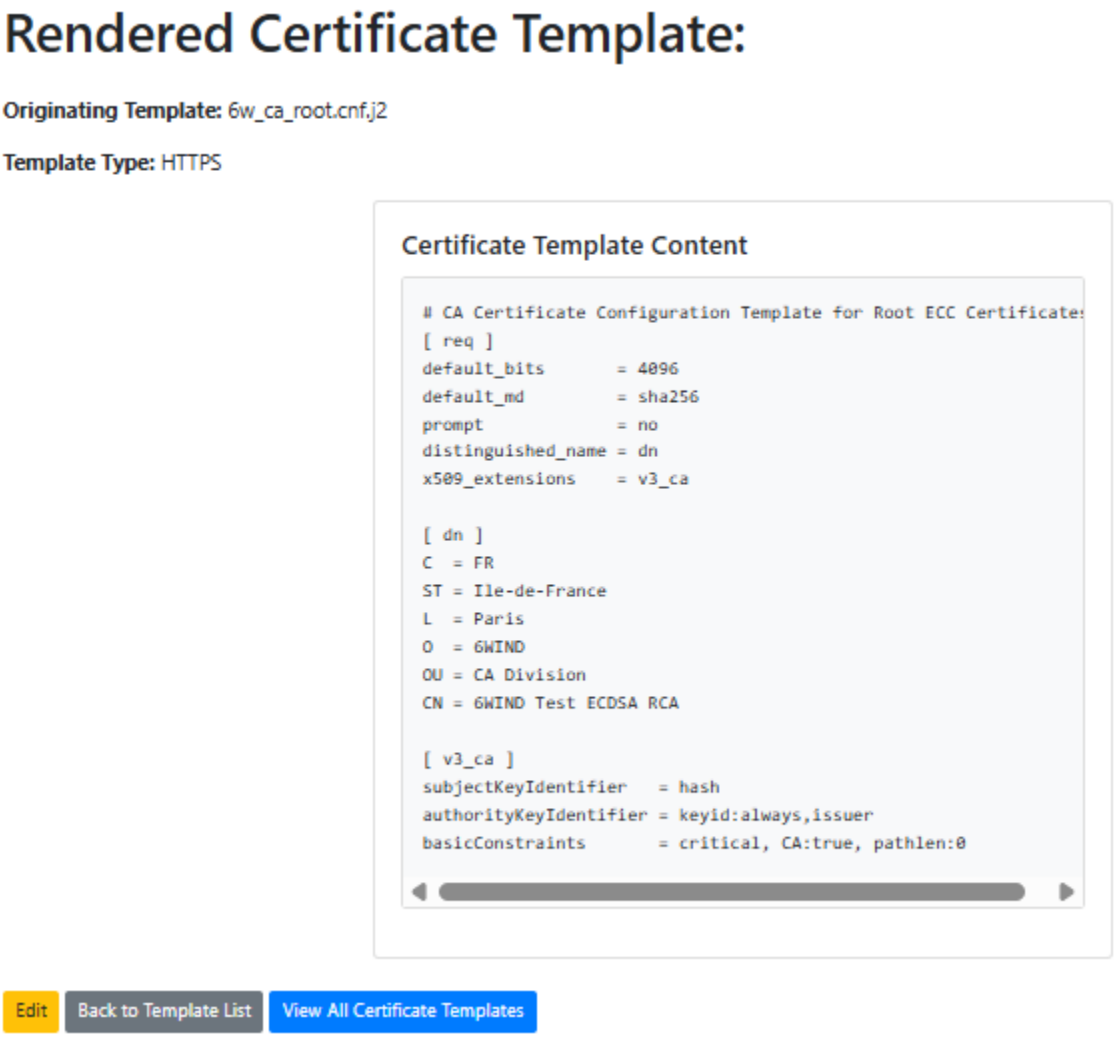
- **Choose a template**
  Select a .cnf.j2 template from the dropdown menu.
- **Load Template**
  After selecting a template, clicking **Load Template** opens the **Fill in Variables** form.
  This form displays all variables defined inside the template




**Rendering**

- **Render Profile**
  Generates a new certificate template using the filled-in values and the selected Jinja template. The created template appears in the Templates list, where the *Originating Template* column shows which .cnf.j2 file it was generated from.
- **Back to Template List**
  Returns to the template selection page.

## View Rendered Certificate Template




## Creating New Template

Templates added manually to the server under folder x509_templates

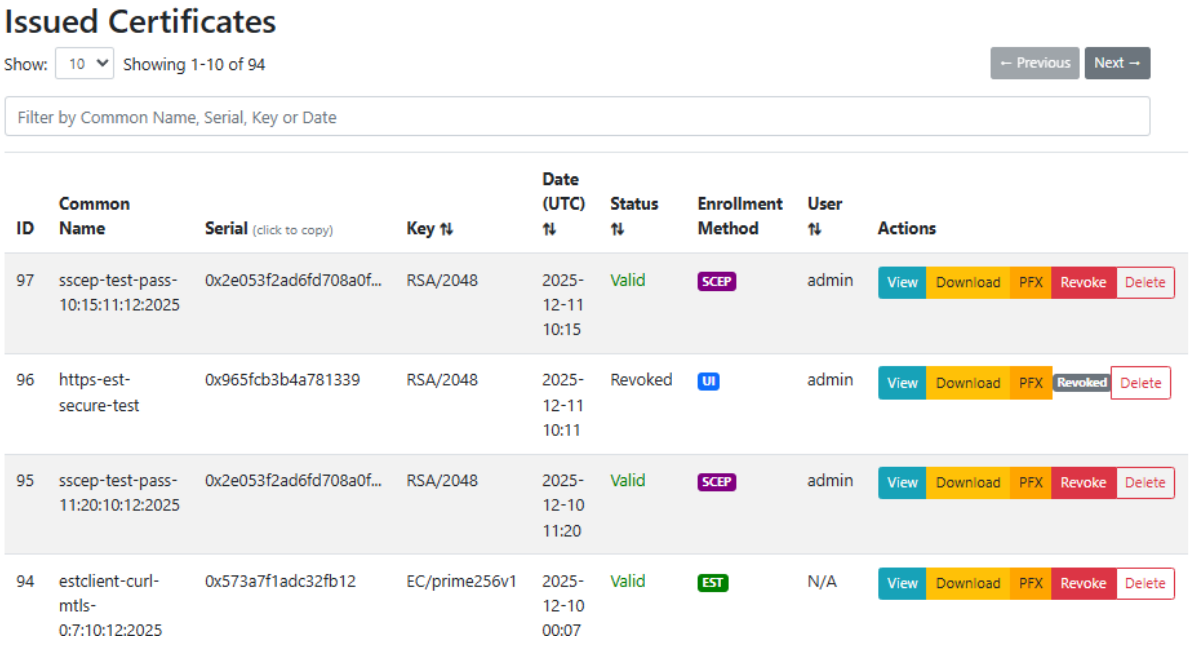
Template is jinja j2 style template for example

```
# CA Certificate Configuration Template for Root ECC Certificates
[ req ]
# Note: For ECC keys the "default_bits" option is not used.
default_md         = sha256
default_days       = 3650
prompt             = no
distinguished_name = dn
x509_extensions    = v3_ca
default_ec_curve   = prime256v1

[ dn ]
C  = {{ ca_country | default("FR") }}
ST = {{ ca_state | default("Ile-de-France") }}
L  = {{ ca_city | default("Paris") }}
O  = {{ ca_organization | default("6WIND") }}
OU = {{ ca_organizational_unit | default("CA Division") }}
CN = {{ ca_common_name | default("6WIND Test ECDSA RCA") }}

[ v3_ca ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = critical, digitalSignature, keyCertSign, cRLSign
```

## 13.7 Certificates

### List

The **Issued Certificates** page lists all certificates that have been signed and issued by the CA. Users can filter, inspect, download, revoke, or delete certificates depending on their role and permissions.




### Page Features

- **Show / Pagination**
  Controls how many certificates are displayed and allows navigation between pages.
- **Filter Bar**
  Filters certificates by common name, serial number, key type, date, or any visible column.

### Table Columns

Each row represents an issued certificate and displays the following information:

- **ID**
  Internal numeric identifier for the certificate.
- **Common Name**
  The CN value from the certificate's Distinguished Name.
  Clicking the name opens the certificate detail view.
- **Serial** *(click to copy)*
  The certificate's serial number.
  Can be copied for debugging, OCSP checks, or revocation workflows.
- **Key**
  The key algorithm and size used for the certificate:
  - RSAs

  - ECs.
  - PQCs (if applicable)

- **Date**
  The date and time the certificate was issued.
- **Status**
  Indicates whether the certificate is:
  - **Valid**
  - **Revoked**
  - **Expired**
- **Enrollment Method**
  Identifies how the certificate was issued:
  - **SCEP** – Certificate issued via SCEP enrollment
  - **UI** – Manually signed via the web UI
  - **EST** – Certificate issued via EST enrollment
- **User**
  The user or system component that performed the signing. (admin user only)

**Actions**

- **View**
  Opens full certificate details (subject, issuer, extensions, validity period, etc.).
- **Download**
  Downloads the certificate chain:
  - End-entity certificate
  - Intermediate certificate
  - Root certificate
- **PFX** *(secret-protected)*
  Generates a PKCS#12 bundle containing:
  - Certificate
  - Private key
  - Intermediate CA

  Suitable for importing into Windows, browsers, or secure devices.

Applicable only if the Certificate original Key is available

Action execution popup dialog for the needed secret

- **Revoke**
  Invalidates the certificate and adds it to the CRL. Requires typing **“revoke”** to confirm.
- **Delete**
  Removes the certificate from the database. Requires typing **“delete”** to confirm.

## View

The **View Certificate** page displays the full details of an issued certificate, including a human-readable summary, the full parsed certificate text, and the raw PEM-encoded certificate. This view is typically accessed from the *Issued Certificates* table by selecting the **View** action.

### Raw Certificate (PEM Format)

-----BEGIN CERTIFICATE-----
MIIEajCCAlKgAwIBAgIULgU/Ktb9cIoPR5LLfeFLeonzvKkwDQYJKoZIhvcNAQEL
BQAwYTELMAkGA1UEBhMCSUwxDDAKBgNVBAgMA1RMVjERMA8GA1UEBwwIVGVsIEF2
aXYxDDAKBgNVBAoMA1JBRDELMAkGA1UECwwCUkQxFjAUBgNVBAMMDVJBRFN1Y1R1
c3RSU0EwHhcNMjUxMjEwMDkyMDEzWhcNMjUxMjIwMDkyMDEzWjArMSkwJwYDVQQD
DCBzc2NlcC10ZXN0LXBhc3MtMTE6MjA6MTA6MTI6MjAyNTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAK0a53wub4hHG7v7vRDCdaPDfLZUNVH2Jnz8QsPt
wTecJDwJnQj/uLIxzvOCl2HNEHFi04qKLYS4uuR6AaxalO3q0YdsInRCVIWgofS1
n3uQLge1n7+xBaL3mPiS9vfRxJ3yV914++vPxslEINqcZwRdM1e76CY5q3b33gVO
N6qZx3WKI+qPxgJ9ulQWm+wOil2a/ltOlRs/9FLnWyZs87FuaN5dc3TXnnvU5ydQ
jkrUVa1abmRqAeZGZUmwK1dUv0bNkpPG7p/J80S9IKNq1iT+tps5WfUNaAnZ0DEp
0t8ka0VR1fzujCzKWawsV5A6kG6Pktntg/PBV9lumRqlqxcCAwEAAaNQME4wDAYD
VR0TAQH/BAIwADAdBgNVHQ4EFgQUVb+ULg8g5rT+12rWX+nI7BmASrUwHwYDVR0j
BBgwFoAUgqAk+aJsta0ewXGhKzDsIxkykEQwDQYJKoZIhvcNAQELBQADggIBAHjr
tRUtzycj1Hc/4/Fm597TNL9tIR++zqHwjJT/Wtx06f5W47fxZLdbIuaJwSwUZZzC
BnEIS10NnVk2Bg1AcvR7VvZUJxO7Cpu3rZjelRjC9qt20DopqSeUU8ZfdjZyRvNt
E/Ehk3bv88iqRQCvlQOOn/e29VMorpIBCxOh3GLS+szw2OOw63m+d4/4WgCcegI6
n5FSEKvWpHj/HKSK42yrXxsqGDYNMh1m6NYOWWYTdvMHJuL6tT8DC64exiZU4jPM
pdL2IiYFqsXXCaNYkMRSDNJszh+z+URzuA5cfy9B+SYZpnQicj+tivRF3mWtRNQo
0/VPzbjnrLEqfrtt34D43UBtcjoNcmbXkRJFZNImuJbIE6/JX6bnjXUzXZX10Txg
fL8Q0mOwsT+HIaPfvN5I4SJ46DulqVWuaCKUjhcxtg/tLaCbvQv2fsglFjyvYEvp
ztw7gRqdLjdIV2OjbepQlpDs3SnoKVCQrNwx30hDbjriU/Nb7qcokD3LKcZwJZrR
q7pTGiRyonySJTlkGf/3Pz4X83gwWKdRBxN7/zkoOxixpDNuMGBMn2NZUAtA+1bu
xlLYWcd5f7SNokZ9WqX5G26K/R6fMV0RyxybCk+T/XvpwFwz2wkZQFmX61WH8A6H
Imo/dANzSMIPyY9Ko/yDI7yapn3CYS46dPN5jCud
-----END CERTIFICATE-----

← Back to Home

### Certificate Summary

The left-hand panel provides a structured summary of the certificate's main attributes:

- **Public Key Algorithm**
  Displays the algorithm used (e.g., RSA, EC, PQC).
- **Public Key Parameters**
  Shows key size or curve (e.g., RSA/2048 bits, EC/prime256v1).
- **Subject**
  Contains the subject DN fields such as commonName.
- **Issuer**
  Displays the issuing CA's distinguished name.
- **Serial Number**
  Full serial number in hexadecimal format.
- **Version**
  Certificate X.509 version (typically v3).
- **Validity**
  - **Not Valid Before**
  - **Not Valid After**

- **Signature Algorithm**
  Displays the signature algorithm used by the issuing CA.

At the top of the page, a badge indicates the **Enrollment Method**, such as:
**Issued via: SCEP**, **UI**, or **EST**.

**Detailed Certificate Text**

The right-hand panel shows the full certificate as parsed by OpenSSL. This includes:

- Certificate version and serial number
- Signature algorithm
- Issuer and subject data
- Validity dates
- Public key details (algorithm, bit size, modulus, etc.)
- Full X.509 extensions, if present

This view is useful for administrators validating certificate structure or debugging enrollment issues.

**Raw Certificate (PEM Format)**

The bottom section displays the certificate in standard PEM format:

-----BEGIN CERTIFICATE-----

MIIB...

...

-----END CERTIFICATE-----

This representation can be copied for:

- Import into external systems
- Manual verification
- OCSP checks
- Certificate decoding tools

**Navigation**

- **Back to Home** returns the user to the main dashboard or previous certificate list.

Once certificate has been signed and issued the user can perform the following actions

1. View – Display the certificate's details and metadata.
2. Download – Retrieve the certificate file for local use or storage.
3. PFX – Generate and download a PFX/PKCS#12 bundle containing the certificate and private key.
4. Revoke – Mark the certificate as invalid so it can no longer be trusted or used.
5. Delete – Permanently remove the certificate record from the system.

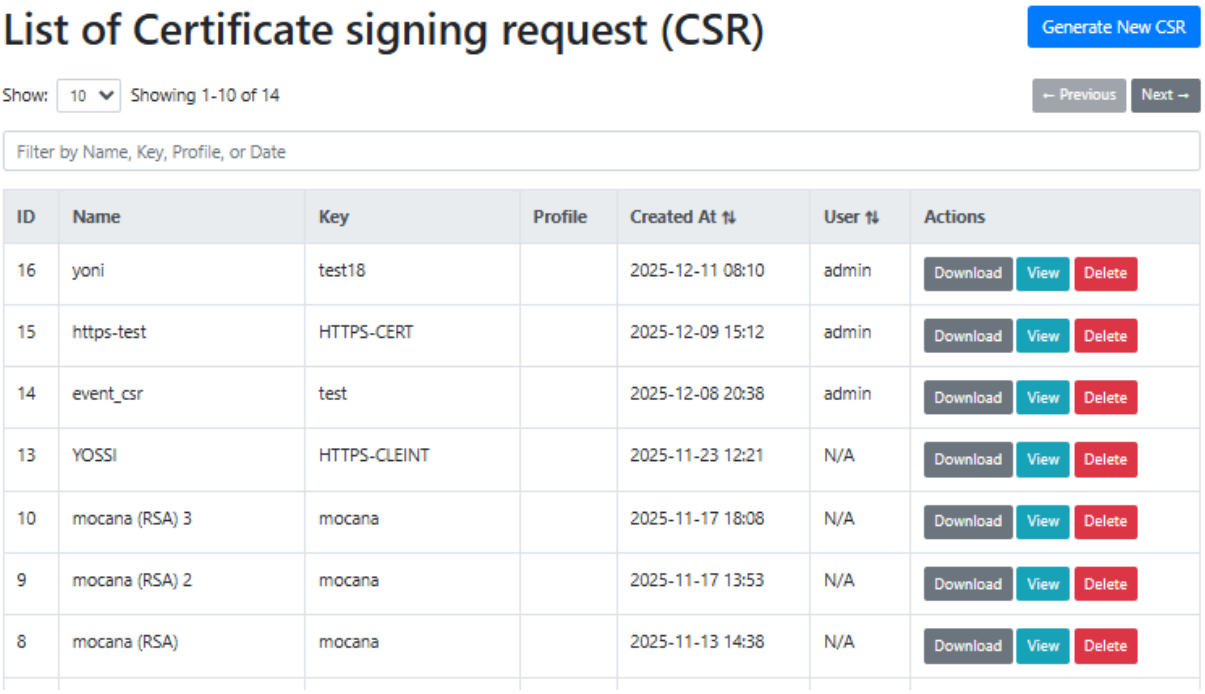
The Actions Delete, revoke and PFX are secret protected

Note: Users can also view the Root and intermediate (Sub) Certificate

## 13.8 CSRs

### List

The **CSR List** page displays all certificate signing requests submitted through the system.
Users can view, download, or delete pending CSRs before they are signed into certificates.




**Page Features**

- **Generate New CSR**
  Opens the form to create a new certificate signing request using an existing key and profile.
- **Show / Pagination**
  Controls how many CSRs are displayed and allows navigation through pages.
- **Filter Bar**
  Filters CSRs by name, key, profile, user, or date.

**Table Columns**

- **ID**
  Internal identifier for the request.
- **Name**
  The user-assigned name for the CSR.
- **Key**
  The key used to generate the CSR.
- **Profile**
  The certificate profile applied during CSR creation.
- **Created At**
  Timestamp of when the CSR was generated.

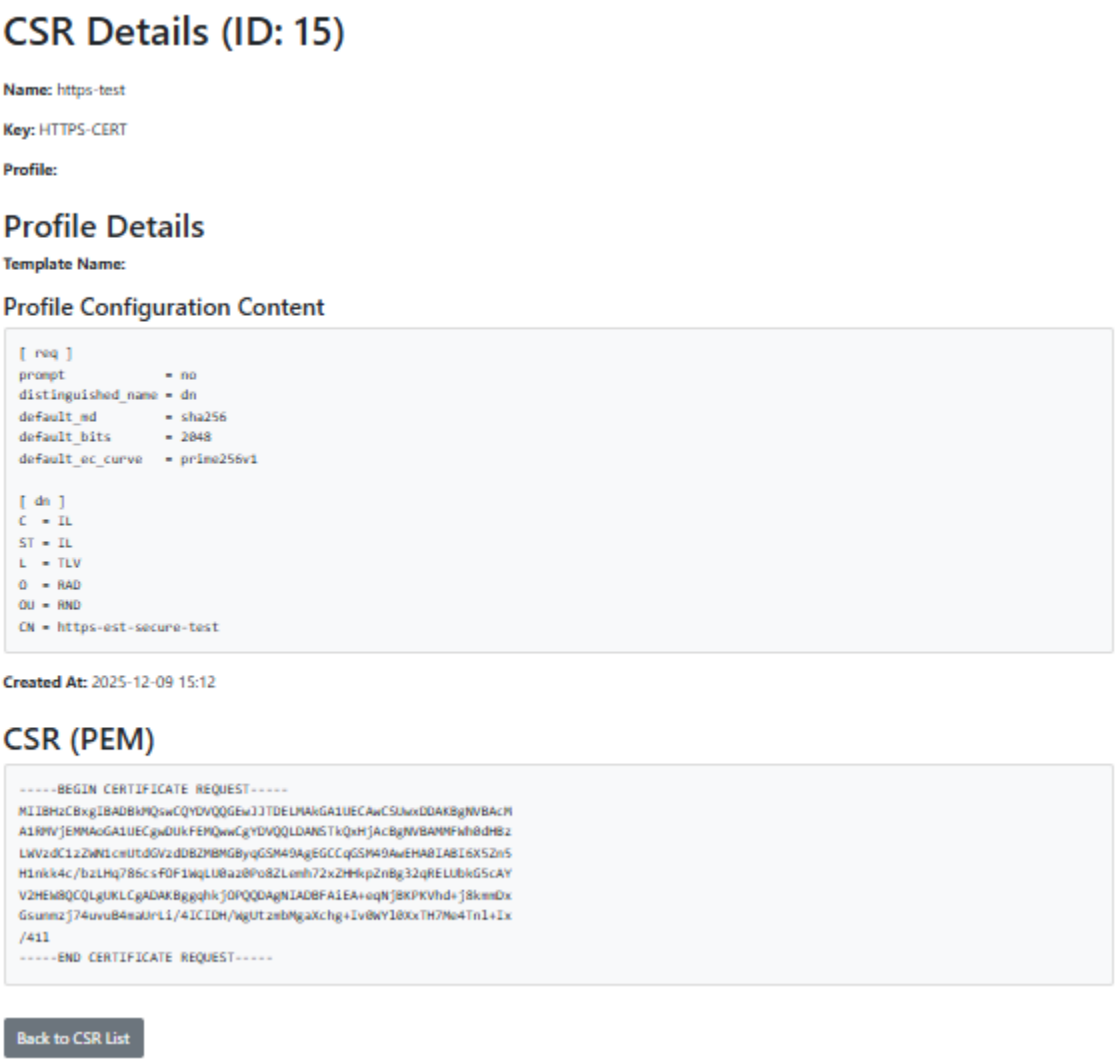
- **User**
  The user who created the CSR. (Visible only to users with the admin role.)

**Actions**

- **Download**
  Downloads the CSR in PEM format.
- **View**
  Displays the full CSR details, including subject information and public key.
- **Delete**
  Removes the CSR from the system.

## View

The **View CSR** page displays the details of a selected certificate signing request.




**CSR Details**

- **Name**, **Key**, and **Profile**
  Basic information about how the CSR was generated.

**Profile Details**

If the CSR was created using a profile, the page shows:

- **Template Name**
- **Profile Configuration Content** — including [req] and DN sections, if present.

This allows the user to review the exact settings used to generate the CSR.

**Creation Time**

Displays when the CSR was created.

**CSR (PEM)**

A read-only block containing the full PEM-encoded CSR:

-----BEGIN CERTIFICATE REQUEST-----

...

-----END CERTIFICATE REQUEST-----

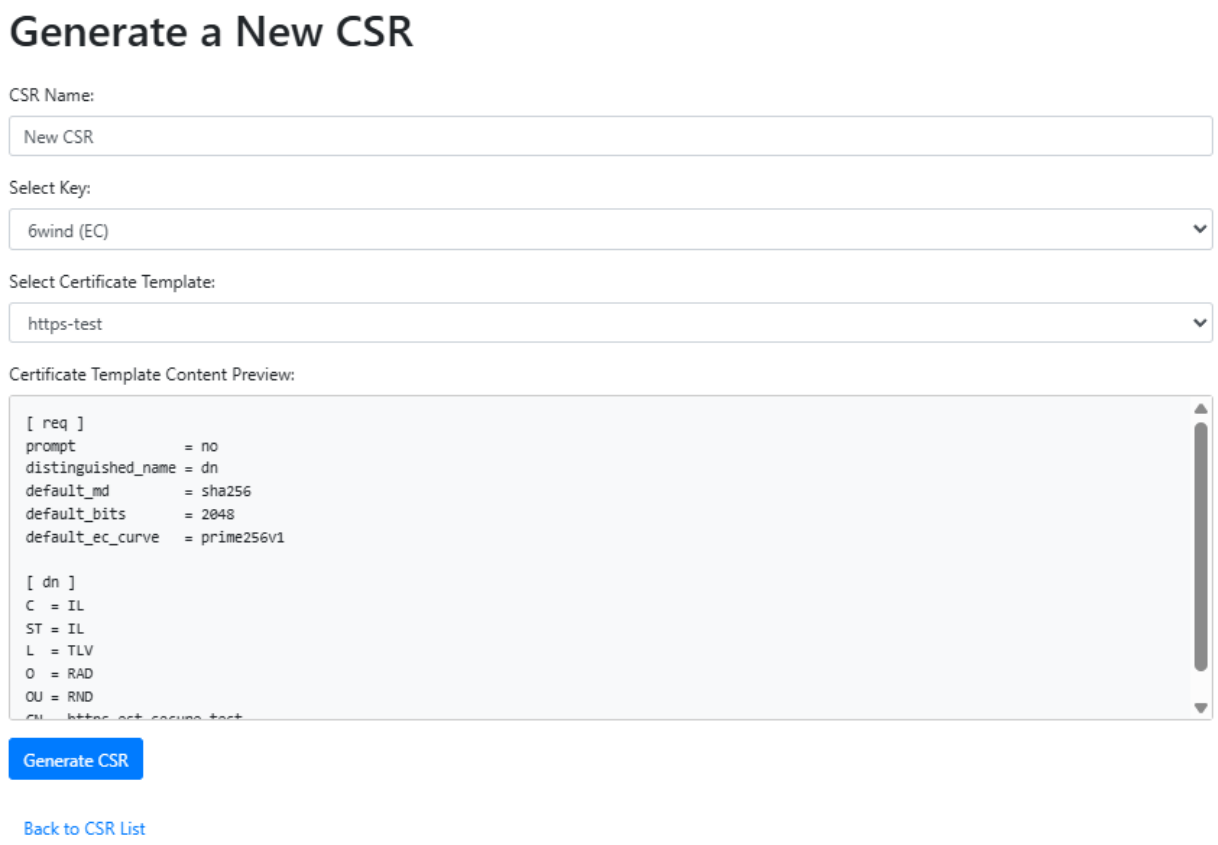
This can be copied for external tools or manual signing.

**Navigation**

- **Back to CSR List** returns to the main CSR table.

## Generate new CSR

The **Generate a New CSR** page lets users create a certificate signing request based on an existing key and certificate template.




### Fields

- **CSR Name**
  Friendly name for the CSR (shown later in the CSR list).
- **Select Key**
  Choose an existing key (RSA, EC, or PQC) that will be used to build the request.
- **Select Certificate Template**
  Select a certificate template that defines the DN and other CSR parameters.
- **Certificate Template Content Preview**
  Read-only preview of the selected template (for example, [req] and DN sections) so the user can verify the configuration before generating the CSR.

### Actions

- **Generate CSR**
  Creates the CSR and returns to the CSR list, where the new request is available for viewing, downloading, or signing.

- **Back to CSR List**
  Cancels the operation and returns to the list without creating a CSR.

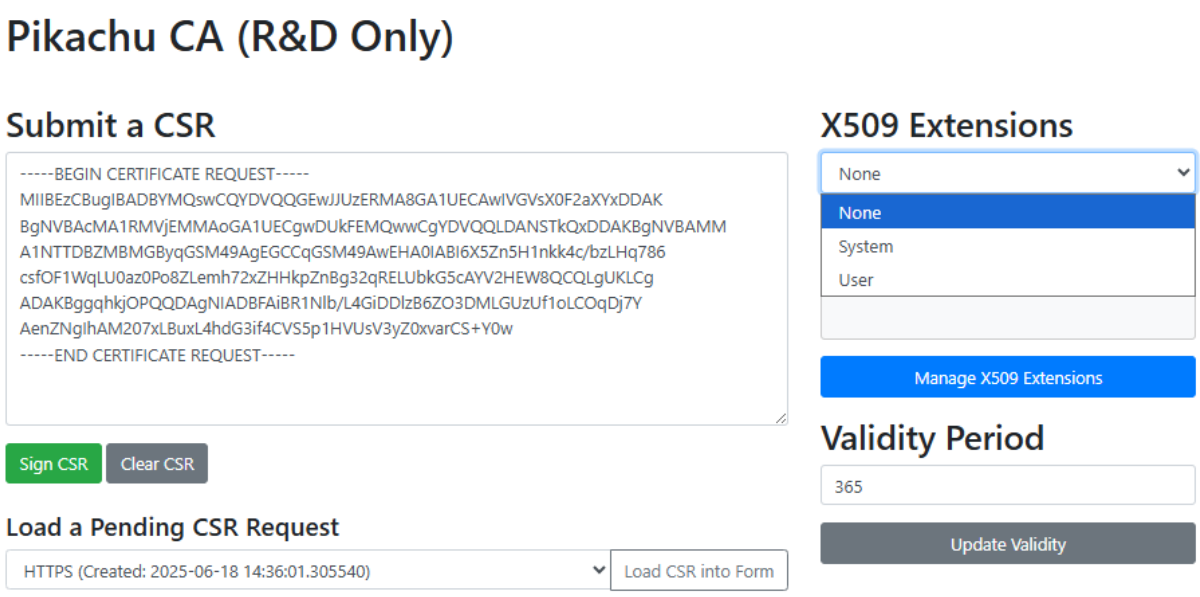
## 13.9 Enrollment

### UI signing

Using the web browser under Submit a CSR past the certificate request content and press Sign CSR button

Users can load existing CSR from the list of Pending CSR Request

Chose  server extension , either None , User or System one.




After signing  the certificate will appear in the issued certificates table below

The table can be filtered by either part of Common Name or Serial values

## EST

EST enrollment technology automates certificate issuance by allowing devices to submit CSRs and receive signed certificates over secure channels.

Convert the certificate request

```
openssl req -outform DER -in client1.csr -out client1.csr.der
```

### *Curl*

send the enrolment command using curl

```
curl -k -X POST --data-binary @client1.csr.der \
https://openxpki.iot-rad.com/.well-known/est/simpleenroll \
  -H "Content-Type: application/pkcs10" \
  --output client1.crt.p7
```

For **mTLS** tests use the following curl

```
curl --cert https.crt --key https.key -X POST --data-binary @client1.csr.der\
https://openxpki.iot-rad.com:4443/.well-known/est/simpleenroll \
  -H "Content-Type: application/pkcs10" \
  --output client1.crt.p7
```

Extract the certificate

```
openssl pkcs7 -inform DER -in client1.crt.p7 -print_certs -out client1.crt
```

the certificate added to the UI issues certificates table

tested issued certificate using openssl

```
openssl x509 -in client1.crt  -noout -text
```

### *Estclient*

enroll using command estclient

```
estclient enroll -server pikachu-ca.iot-rad.com -insecure -csr etx.csr -out
etx.crt
```

**mTLS** enroll using command estclient

```
estclient enroll -server pikachu-ca.iot-rad.com:4443 -cert https.crt -key
https.key -csr etx.csr -out etx.crt
```

## SCEP

For SCEP you must provide the CA of either full chain (Root + Sub CA) or only the intimidate one (Sub CA)

send the enrolment command using sscep command (see Complementary section for installing it)

```
sscep enroll -u http://openxpki.iot-rad.com/scep \
-c ca.cert.pem \
-k client1.key \
-r client1.csr \
-l client1.crt
```

Note: to debug sscep command add -d option

the certificate added to the UI issues certificates table

tested issued certificate using openssl

```
openssl x509 -in client1.crt  -noout -text
```

Convert the certificate request

```
openssl req -outform DER -in client1.csr -out client1.csr.der
```

send the enrolment command using curl

```
curl -k -X POST --data-binary @client1.csr.der \
http://openxpki.iot-rad.com/.well-known/est/simpleenroll \
  -H "Content-Type: application/pkcs10" \
  --output client1.crt.p7
```

Extract the certificate

```
openssl pkcs7 -inform DER -in client1.crt.p7 -print_certs -out client1.crt
```

the certificate added to the UI issues certificates table

tested issued certificate using openssl

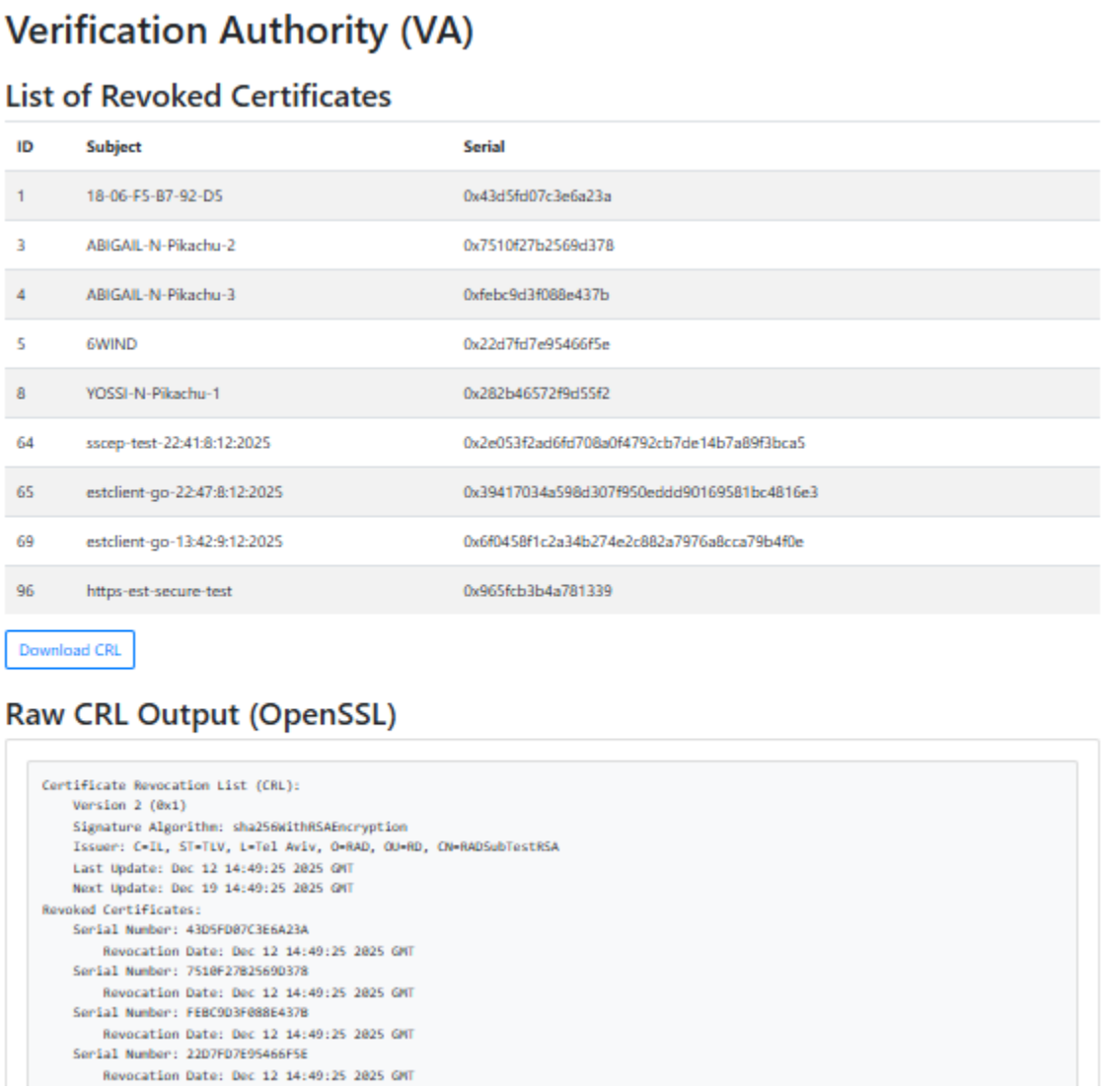
```
openssl x509 -in client1.crt  -noout -text
```

Note: based on GitHub repo https://github.com/mosen/SCEPy

# 13.10 Verification Authority (VA)

The Verification Authority page displays all certificates that have been revoked by the CA and provides access to the generated Certificate Revocation List (CRL).

## List of Revoked Certificates




This table shows all revoked certificates, including:

- **ID**
  Internal index for reference.
- **Subject**
  The certificate's Common Name (CN) or identifying label.
- **Serial**
  The serial number of the revoked certificate, used for validation and OCSP/CRL checks.

This list reflects the current content of the active CRL and updates whenever a certificate is revoked.

**Download CRL**

The **Download CRL** button provides the complete CRL file in PEM format.
This file can be published or distributed to systems that need to validate certificate status.

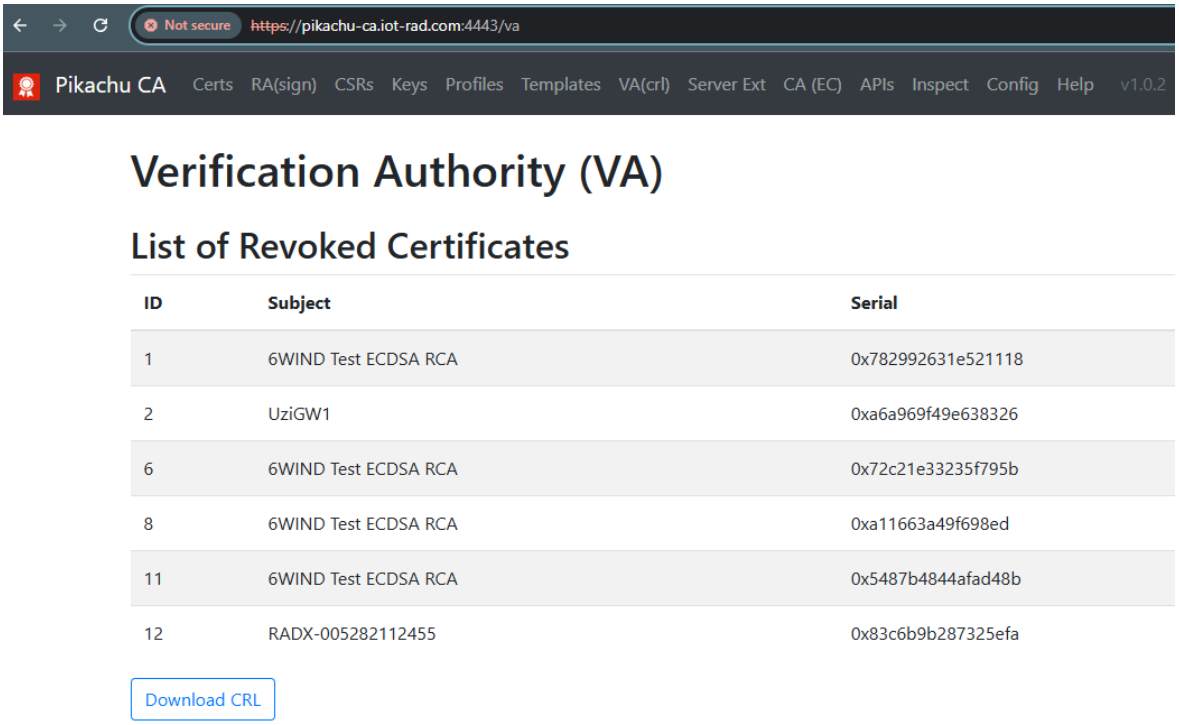
**Raw CRL Output (OpenSSL)**

A raw, decoded view of the CRL is displayed below the table.
It includes:

- **CRL version and signature algorithm**
- **Issuer of the CRL**
- **Next Update** (when a new CRL will be required)
- **Revoked certificates** with:
  - Serial number
  - Revocation date

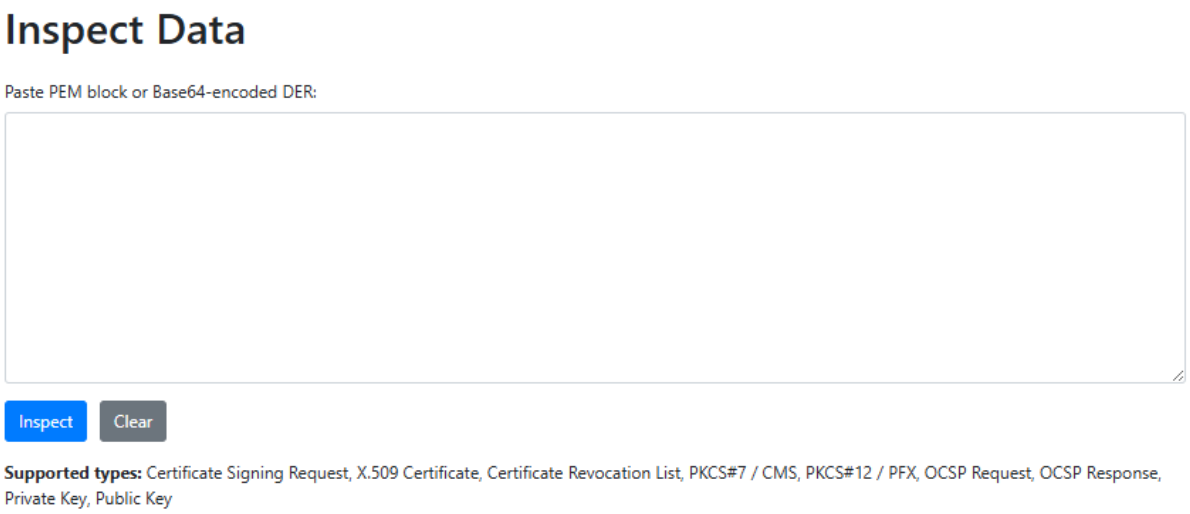
This output is generated using OpenSSL and is useful for troubleshooting and verifying correct CRL structure.

List of revoked certificates




## 13.11 Inspect

The **Inspect Data** tool allows users to paste any PEM block or Base64-encoded DER data and automatically with openssl command decode it into a readable structure.



**How It Works**

- Paste a certificate, CSR, CRL, key, OCSP message, or PKCS file into the text box.
- Click **Inspect** to decode the content and display its parsed details.
- Click **Clear** to reset the input field.

**Supported Types**

The inspector can recognize and decode:

- Certificate Signing Request (CSR)
- X.509 Certificate
- Certificate Revocation List (CRL)
- PKCS#7 / CMS
- PKCS#12 / PFX
- OCSP Request
- OCSP Response
- Private Key
- Public Key

This tool simplifies validation and troubleshooting by providing quick access to decoded cryptographic objects without needing external utilities.

Openssl based info.

# Raw CRL Output (OpenSSL)

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=IL, ST=TLV, L=Tel Aviv, O=RAD, OU=RD, CN=RADSubTestECDSA
        Last Update: Apr 26 16:00:34 2025 GMT
        Next Update: May  3 16:00:34 2025 GMT
Revoked Certificates:
    Serial Number: 782992631E521118
        Revocation Date: Apr 26 16:00:34 2025 GMT
    Serial Number: A6A969F49E638326
        Revocation Date: Apr 26 16:00:34 2025 GMT
    Serial Number: 72C21E33235F795B
        Revocation Date: Apr 26 16:00:34 2025 GMT
    Serial Number: 0A11663A49F698ED
        Revocation Date: Apr 26 16:00:34 2025 GMT
    Serial Number: 5487B4844AFAD48B
        Revocation Date: Apr 26 16:00:34 2025 GMT
    Serial Number: 83C6B9B287325EFA
        Revocation Date: Apr 26 16:00:34 2025 GMT
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:44:02:20:0e:e6:86:5d:15:20:41:8a:7d:2d:b4:63:db:1e:
        18:fe:78:98:d6:8e:d0:ae:d5:d7:7e:9a:e4:71:14:09:cf:92:
```

## 13.12 CA certificate management

User can retrieve the following CA certificate

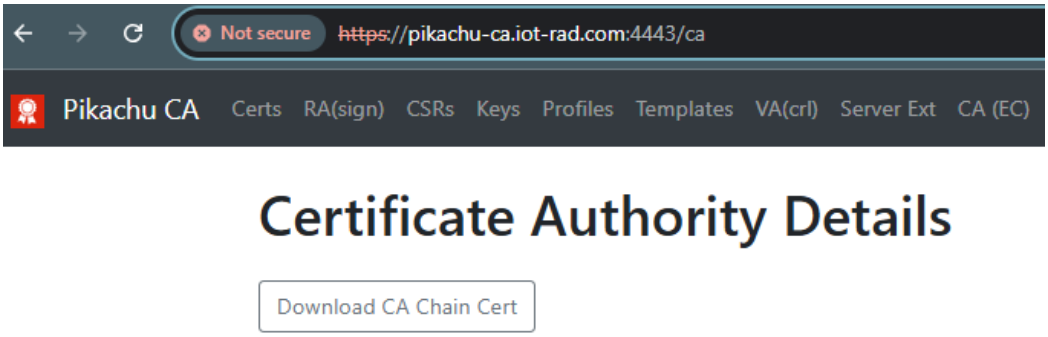
**Root CA Certificate**:
A trusted self-signed certificate that sits at the top of the certificate hierarchy. It is the anchor of trust used to verify all other certificates in the chain.

**Sub CA (Intermediate) Certificate**:
A certificate issued by the Root CA (or another intermediate) to delegate signing authority. It acts as a bridge between the Root CA and end-entity (leaf) certificates, improving security and scalability.

## Download

Download the CA Chain Certificate by two methods:

1. Using UI pressing the link Download the CA Chain Cert




2. Using API endpoint

```
curl -k https://openxpki.iot-rad.com:4443/downloads/chain --output
ca.chain.pem
```

the ca.chain.pem file can be inspected using openssl

```
openssl x509 -in ca.chain.pem -noout -text
```

## View

Pressing the buttons “View Root CA Certificate” and “View Sub CA Certificate” redirect to to view page with all needed Data

## Update

Change the configuration file config.ini

```
[CA]
# Which subordinate CA to use by default: "EC" or "RSA"
mode = EC

# Paths for both modes; the get_ca_config() helper below will pick the right
SUBCA_KEY_PATH_EC    = /home/rocky/pki-subca/rad_ca_sub_ec.key
SUBCA_CERT_PATH_EC   = /home/rocky/pki-subca/rad_ca_sub_ec.crt
CHAIN_FILE_PATH_EC   = /home/rocky/pki-subca/rad_chain_ec.crt

SUBCA_KEY_PATH_RSA   = /home/rocky/pki-subca/rad_ca_sub_rsa.key
SUBCA_CERT_PATH_RSA  = /home/rocky/pki-subca/rad_ca_sub_rsa.crt
CHAIN_FILE_PATH_RSA  = /home/rocky/pki-subca/rad_chain_rsa.crt

ROOT_CERT_PATH       = /home/rocky/pki-root/rad_ca_root.crt
[HTTPS]
# HTTPS certificate & key for your main CA UI
ssl_cert = /home/rocky/pki-https/tls.cert.pem
ssl_key  = /home/rocky/pki-https/tls.key.pem
port     = 4443
```

## Generate

Below example how to generate all needed CA files

### *Root CA*

Generate Key using type EC and curve prime256v1(secp256r1)

Use root CA request configuration

```
cat > rad_ca_root.cnf <<EOL
# CA Certificate Configuration Template for Root ECC Certificates
[ req ]
# Note: For ECC keys the "default_bits" option is not used.
default_md         = sha256
default_days       = 3650
prompt             = no
distinguished_name = dn
x509_extensions    = v3_ca
default_ec_curve   = prime256v1

[ dn ]
C  = IL
ST = TLV
L  = Tel Aviv
O  = RAD
OU = RD
CN =  RADRootTestECDSA

[ v3_ca ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = critical, digitalSignature, keyCertSign, cRLSign
EOL
```

Sign the root by itself

```
openssl req -config rad_ca_root.cnf -key rad_ca_root.key -new -x509 -days
3650 -sha256 -out rad_ca_root.crt
```

### *Sub CA*

Generate Key using type EC and curve prime256v1(secp256r1)

Use sub-CA request configuration

```
cat > rad_ca_sub.cnf <<EOL
# CA Certificate Configuration Template for Subordinate (Intermediate) ECC
Certificates
[ req ]
default_bits       = 2048
default_md         = sha256
prompt             = no
distinguished_name = dn
req_extensions     = v3_intermediate

[ dn ]
C  = IL
ST = TLV
L  = Tel Aviv
O  = RAD
OU = RD
CN = RADSubTestECDSA

[ v3_intermediate ]
subjectKeyIdentifier   = hash
#authorityKeyIdentifier = keyid,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = keyCertSign, cRLSign
EOL
```

Prepare the Sub-CA Certificate request

```
openssl req -config rad_ca_sub.cnf -key rad_ca_sub.key -new -out
rad_ca_sub.csr
```

add the Root-CA signing Server extension

```
cat > ca_root_ext.cnf <<EOL
[ v3_intermediate ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid,issuer
basicConstraints       = critical, CA:true, pathlen:0
keyUsage               = digitalSignature,keyCertSign, cRLSign
crlDistributionPoints  = URI:https://openxpki.iot-rad.com:4443/downloads/crl
EOL
```

Sign the sub-CSR by the root CA

```
openssl x509 -req -in rad_ca_sub.csr -CA rad_ca_root.crt -CAkey
rad_ca_root.key -CAcreateserial -out rad_ca_sub.crt -days 3650 -sha256 -
extfile ca_root_ext.cnf -extensions v3_intermediate
```

this provides the certificates and keys for both root and sub CA.

## 13.13 Online Certificate Status Protocol (OCSP)

OCSP is A real-time certificate status checking protocol that allows clients to verify whether a digital certificate has been revoked, without downloading the full CRL. OCSP improves performance and bandwidth usage compared to traditional revocation lists.

In order to perform the check the user must have the following files:

1. CA certificate ( can be only Sub-CA )
2. Certificate

Check certificate status command ca certificate

```
openssl ocsp -reqout ocsp_request.der \
-CAfile ca.chain.pem \
-issuer ca.cert.pem
-cert client1.crt \
-url http://openxpki.iot-rad.com/ocsp\
-resp_text -respout ocsp_response.der
```

The command stdout return the certificate status

Valid certificate result :

```
Response verify OK
/home/rocky/pki-ocsp/cert_2.pem: good
        This Update: Apr  3 13:15:21 2025 GMT
        Next Update: Apr 10 13:15:21 2025 GMT
```

Revokes certificate result :

```
Response verify OK
/home/rocky/pki-ocsp/cert_revoked.pem: revoked
        This Update: Apr  3 13:15:57 2025 GMT
        Next Update: Apr 10 13:15:57 2025 GMT
        Reason: unspecified
        Revocation Time: Apr  3 13:15:57 2025 GMT
```

the output file ocsp_response.der contains the status as well

command

```
openssl ocsp -respin ocsp_response.der -text -noverify
```

stdout

```
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
    Version: 1 (0x0)
    Responder Id: 9DB644062A4D85759C46D1C4215F4DB7C345149D
    Produced At: Apr  3 10:58:10 2025 GMT
    Responses:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash: 616FC051FA23823B80B63EDD49EBFE18F4FB4E77
      Issuer Key Hash: 9DB644062A4D85759C46D1C4215F4DB7C345149D
      Serial Number: 0DCD62C3C11C144DD8AA313FF654067D3F413E5F
    Cert Status: revoked
    Revocation Time: Apr  3 10:58:10 2025 GMT
    Revocation Reason: unspecified (0x0)
    This Update: Apr  3 10:58:10 2025 GMT
    Next Update: Apr 10 10:58:10 2025 GMT
```

The support OCSP multi certificates checks in one request

Return OCSP response if certificate is unknown to the OCSP CA responder

## Certificate based Servers using OCSP

In order for TLS based Servers to use OCSP the information of the URL to check must be embedded within the Certificate itself.

OCSP requires authorityInfoAccess (AIA)

Adding attributes by following paragraph 2.5 Server Certificate Extension

Some TLS servers like HAProxy only have mode configuration attribute  the URL is taken from the certificate attributes.

## 13.14 Post-Quantum Keys

Users can issue certificate using stronger Keys algorithms

Prerequisite to do so is to add new provider (extension to openssl command)

Check if Quantum safe keys extension oqsprovider Is activated using the following command:

```
#run command
openssl list -providers
#output
Providers:
  default
    name: OpenSSL Default Provider
    version: 3.2.2
    status: active
  oqsprovider
    name: OpenSSL OQS Provider
    version: 0.8.1-dev
    status: active
```

Generate key

```
openssl genpkey -algorithm mldsa44 -provider oqsprovider -out client1.key
```

checking all possible algorithms for the oqsprovider provider with the command:

```
openssl list -public-key-algorithms -provider oqsprovider
```

Generate request

prepare configuration to be added to the certificate request

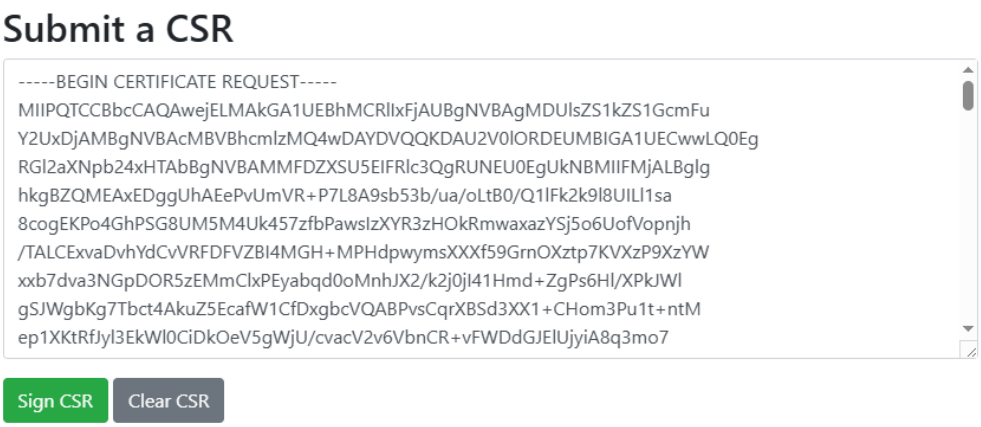
for instance,

```
cat > client1.cnf <<EOL
[ req ]
default_bits       = 2048
default_md         = sha256
distinguished_name = req_distinguished_name
attributes         = req_attrs
prompt             = no

[ req_distinguished_name ]
CN                 = client1.example.com
O                  = My Organization
C                  = US

[ req_attrs ]
challengePassword  = SecretChallenge
EOL
```

Make the request

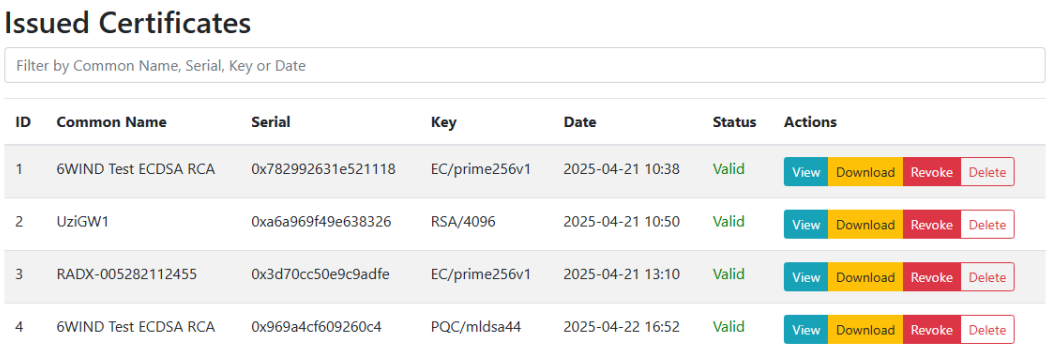
```
openssl req -new -key client1.key -config client1.cnf -out client1.csr
```

using UI




Note: the 2.5 Enrollment using UI can't be used for this operation, code is based on python libraries whereas quantum safe signing is based on OS openssl in the python code.

After signing the certificate will appear in the issued certificates table bellow




## 13.15 APIs

List of supported server APIs

### *General*

# Server APIs & CLI Commands

| Type | Action | Endpoint URL / Command | Misc |
|---|---|---|---|
| CA General | Download CA Chain | `curl -k https://pikachu-ca.iot-rad.com:5443/downloads/chain` | Returns full CA chain file |
| | Download CRL | `curl -k https://pikachu-ca.iot-rad.com:4443/downloads/crl` | Generates and downloads latest Certificate Revocation List |
| | Certificate Status | `curl -k https://pikachu-ca.iot-rad.com:4443/status/0xc75f573d9cb2b581` | Returns status as valid, revoked, or not found in JSON |
| | Expired Certificates | `curl -k https://pikachu-ca.iot-rad.com:5443/expired` | Returns list of certificate IDs that are expired |
| | Download CSR | `curl -k https://pikachu-ca.iot-rad.com:5443/requests/1/download` | Serves saved CSR if available |

### *SCEP*

| | | | |
|---|---|---|---|
| SCEP | SCEP GetCaps | `sscep getcaps -u http://localhost:8090/scep`<br>`curl http://localhost:8090/scep?operation=GetCACaps` | Query server capabilities (AES, DES3, POSTPKIOperation, SHA-256, etc.) |
| | SCEP GetCA | `sscep getca \`<br>`  -u http://localhost:8090/scep \`<br>`  -c ca.crt` | Download CA certificate (prerequisite: RSA-based SubCA) |
| | SCEP Enrollment | `sscep enroll \`<br>`  -u http://localhost:8090/scep \`<br>`  -k client.key \`<br>`  -r client.csr \`<br>`  -c ca.crt \`<br>`  -l client.crt \`<br>`  -E aes \`<br>`  -S sha256` | Enroll certificate via SCEP. Options: -E (encryption: aes\|3des), -S (signature: sha256\|sha512) |
| | Challenge Password (API Token) | `curl -X POST \`<br>`  -H "Authorization: Bearer <API_TOKEN>" \`<br>`  http://localhost:80/api/challenge_passwords` | Generates a challenge password tied to the token's user. Response: value, expires_at, validity, user_id. |

### *EST*

The EST APIs accessible via *curl* and *estclient*.

support operation over both insecure HTTPS endpoints and secure mTLS-protected endpoints.

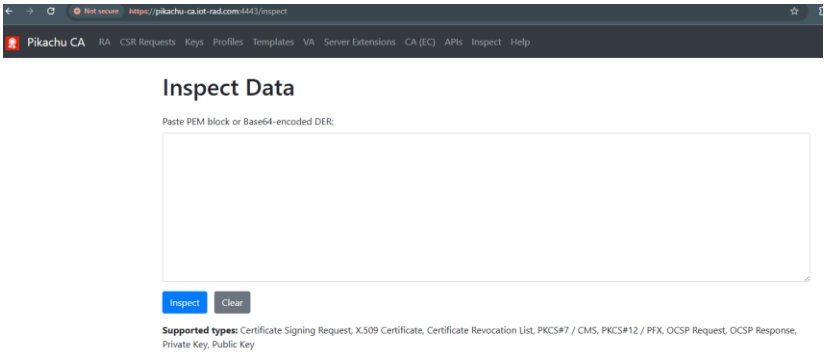
| | | | |
|---|---|---|---|
| EST | EST Enrollment | `curl -k -X POST \`<br>`  --data-binary @etx.csr.der \`<br>`  https://pikachu-ca.iot-rad.com/.well-known/est/simpleenroll \`<br>`  -H "Content-Type: application/pkcs10" \`<br>`  --output etx.crt.p7`<br><br>`estclient enroll \`<br>`  -server pikachu-ca.iot-rad.com \`<br>`  -insecure \`<br>`  -csr etx.csr \`<br>`  -out etx.crt` | Using curl accepts DER CSR and returns signed cert in PKCS#7 |
| | EST Enrollment (mTLS) | `curl --cert https.crt --key https.key \`<br>`  -X POST \`<br>`  --data-binary @etx.csr.der \`<br>`  https://pikachu-ca.iot-rad.com:4443/.well-known/est/simpleenroll \`<br>`  -H "Content-Type: application/pkcs10" \`<br>`  --output etx.crt.p7`<br><br>`estclient enroll \`<br>`  -server pikachu-ca.iot-rad.com:4443 \`<br>`  -cert https.crt \`<br>`  -key https.key \`<br>`  -csr etx.csr \`<br>`  -out etx.crt` | Mutual TLS authentication with client certificate on port 4443 |
| | EST CA Certs | `curl -k \`<br>`  https://pikachu-ca.iot-rad.com/.well-known/est/cacerts \`<br>`  --output chain.crt`<br><br>`estclient cacerts \`<br>`  -server pikachu-ca.iot-rad.com \`<br>`  -insecure \`<br>`  -out ca.pem` | Returns CA chain in PKCS#7 format |
| | EST CA Certs (mTLS) | `curl --cert https.crt --key https.key \`<br>`  https://pikachu-ca.iot-rad.com:4443/.well-known/est/cacerts \`<br>`  --output chain.crt`<br><br>`estclient cacerts \`<br>`  -server pikachu-ca.iot-rad.com:4443 \`<br>`  -cert https.crt \`<br>`  -key https.key \`<br>`  -out ca.pem` | Mutual TLS authentication with client certificate on port 4443 |

***OCSP***

| | | | |
|---|---|---|---|
| OCSP | OCSP Responder | `openssl ocsp \`<br>`  -reqout ocsp_request.der \`<br>`  -issuer rad_ca_sub.crt \`<br>`  -cert valid.crt \`<br>`  -url https://pikachu-ca.iot-rad.com:4443/ocsp \`<br>`  -resp_text \`<br>`  -respout ocsp_response.der` | Returns OCSP status in DER format |

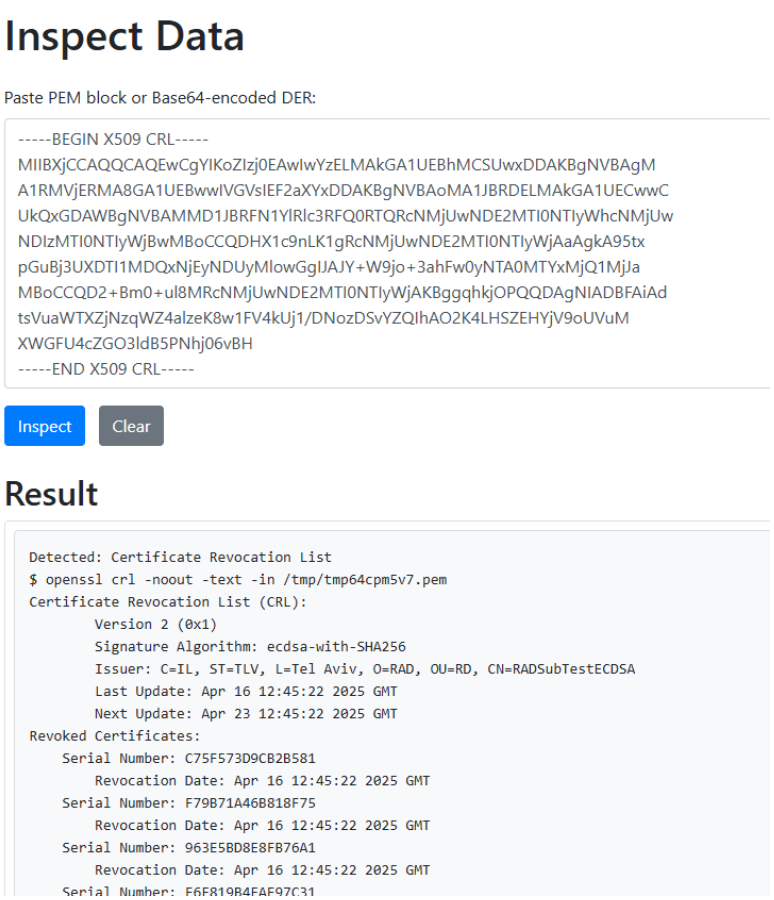
## 13.16 Inspect

User can inspect any PEM block or Base64-encoded DER data




Supported types: Certificate Signing Request, X.509 Certificate, Certificate Revocation List, PKCS#7 / CMS, PKCS#12 / PFX, OCSP Request, OCSP Response, Private Key, Public Key

Inspect provides the designated openssl view stdout

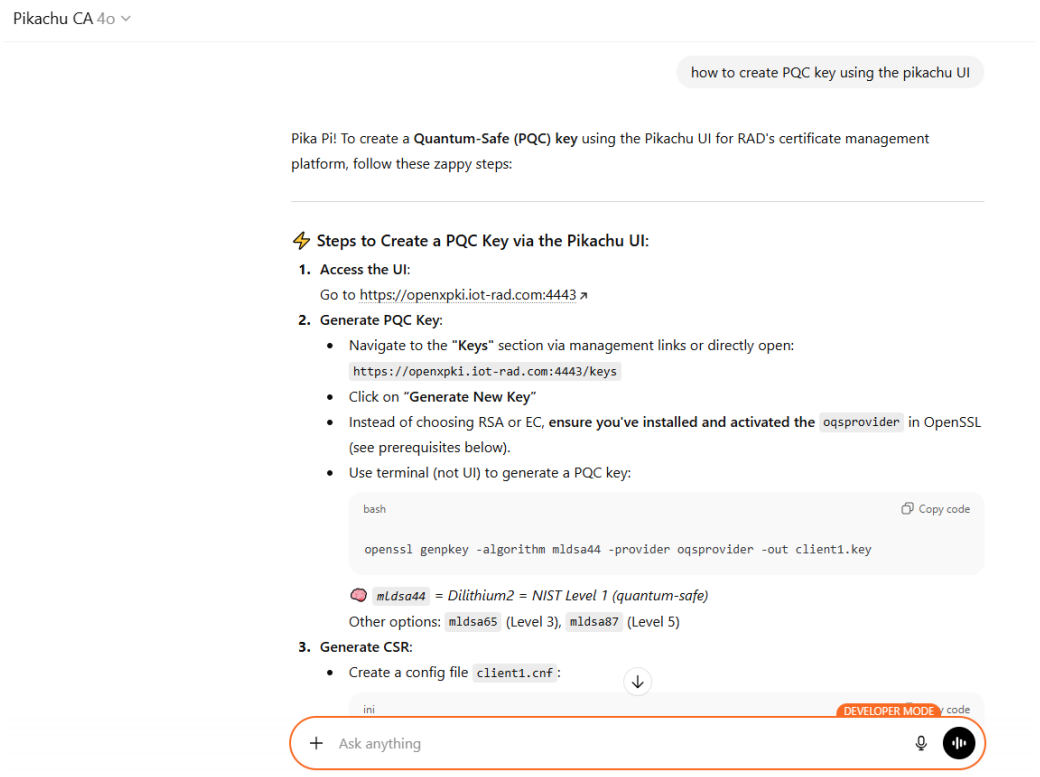



## 13.17 Config

Config Navigation Tab shows the server current configuration, secret protected

## 13.18 ChatPikachu

ChatPGT PGT related to the CA server features and capabilities




# 13.19 Help

PDF format of this word document

Company Confidential

# PKI Hands-On 1.04

| | |
|---|---|
| Last updated | 19-Nov-2025 |
| Doc. version | 1.04 |
| Doc. owner | Uzi Golan |
| Approved by | |
| Customer | R&D |
| Project or installation name | |
| Project number | |
| Solution name | Choose an item. |
| RAD products and versions included | |
| Content type | |
| Keywords | |

## 13.20 Logs

The **Server Logs** page provides a live view of Pikachu CA server output, including warnings, errors, and operational messages.


**Features**

- **Show last *N* lines**
  Controls how many recent log lines are displayed.
- **Auto-refresh interval**
  Refreshes the log display automatically every *N* seconds.
- **Refresh / Resume**
  - **Refresh** updates the log view immediately.
  - **Resume** re-enables automatic updates after pausing.
- **Filter**
  Allows filtering log entries by keyword (e.g., warning, error, vault).
  **Clear Filter** resets the view to show all log messages.

**Log Output**

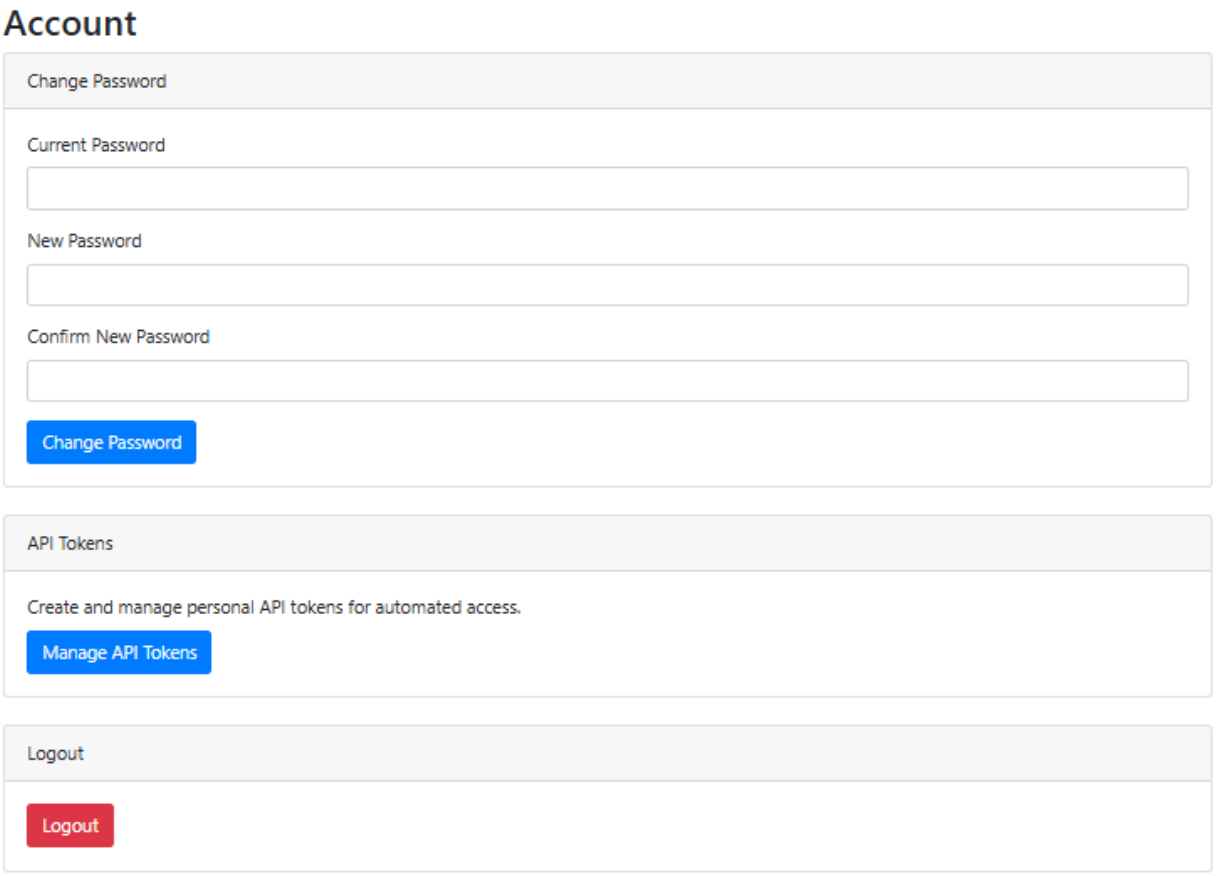
The log window displays real-time server messages such as:

- Initialization warnings
- CA or CRL-related errors
- Flask/Werkzeug runtime notices
- Debug or informational events (based on configured log_level)

This tool is useful for troubleshooting enrollment issues (e.g. SCEP and EST), OCSP backend errors, Vault/LDAP connectivity problems, and general server health.

## 13.21 Account

The **Account** page allows the logged-in user to manage their own login credentials and session.




**Change Password**

Update your account password by entering:

- **Current Password**
- **New Password**
- **Confirm New Password**

Click **Change Password** to apply the update.

Enable only for non LDAP users.

**API Tokens**

Create and manage personal API tokens for automated access. By pressing the button

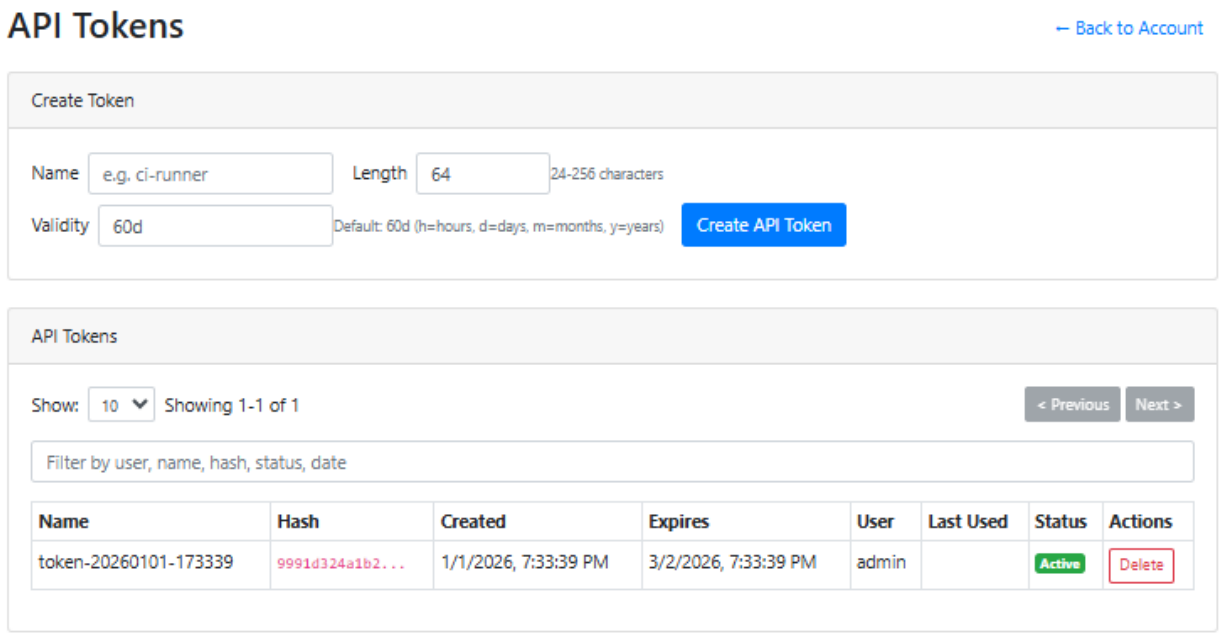
**Logout**

Ends the current session and returns to the login page.

# API Tokens

Dialog for management of API tokens.

Currently support creation of challenge passwords using REST API with token authentication




### Create Token

Create New token by filling the following attributes.

- **Name**
- **Length**
- **Validity**

Click **Create API Token** to create new token and copy its value

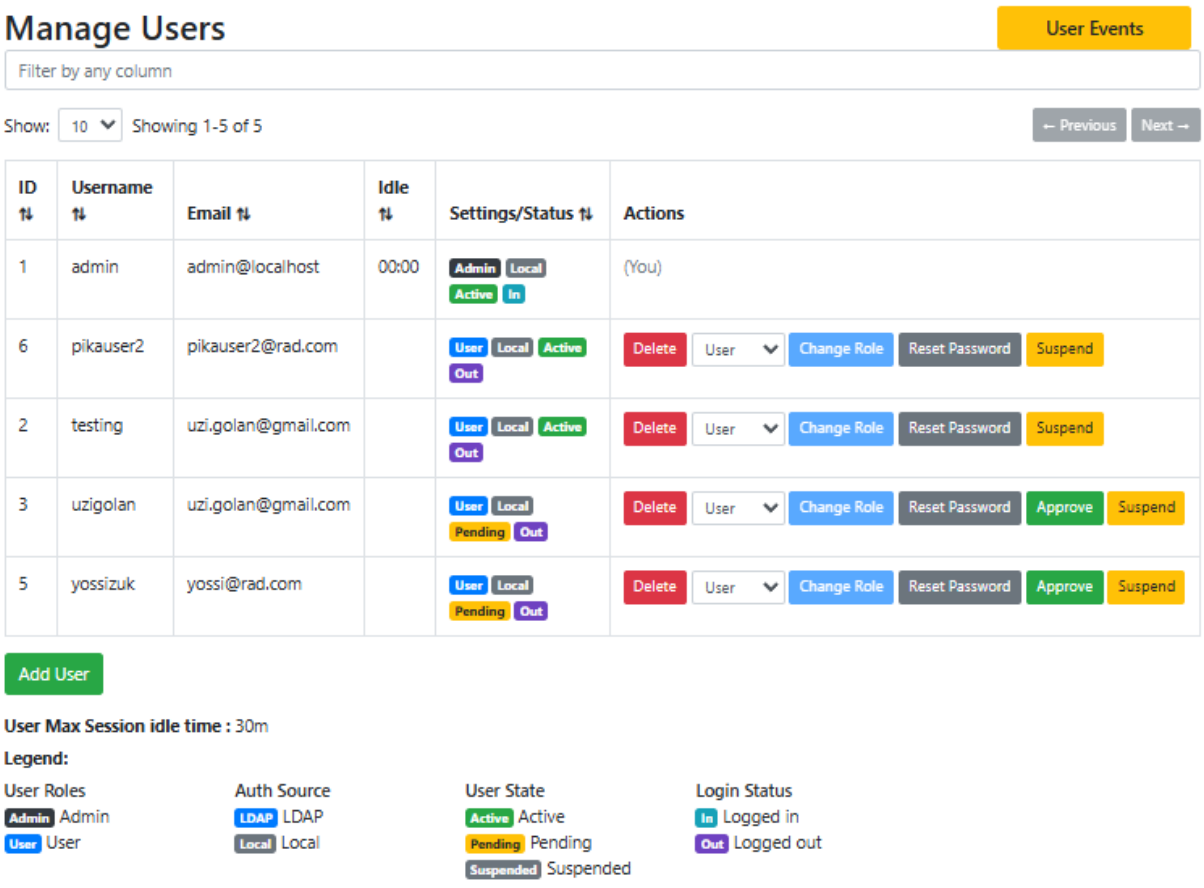

### List of Tokens

The list table based on name, hash, creation date, expiration date, associated user, last usage, and status, with options to delete tokens when no longer needed.

# 13.22 Users

The **Manage Users** page allows administrators to view, approve, suspend, modify, and remove user accounts. It supports both **local** and **LDAP** authentication sources.

***Manage***




**Page Features**

- **Filter Bar** – Search by username, email, role, state, or any visible column.
- **Show / Pagination** – Adjust the number of users displayed and navigate between pages.
- **User Events** – Opens the audit log of user activity.

**Table Columns**

- **ID**
  Internal user ID.
- **Username**
  The login name of the user.
- **Email**
  User's email address.
- **Idle**
  Idle time since last activity (based on configured max idle timeout).

**Settings / Status**

Displays several indicators:

- **Role**: Admin or User
- **Auth Source**: Local or LDAP
- **User State**: Active, Pending, Suspended
- **Login Status**: In (logged in) or Out (logged out)

**Actions**

Administrators may perform the following actions:

- **Delete** – Removes the user account.
- **Change Role** – Switch between *Admin* and *User*.
- **Reset Password** – Available for *Local* users only.
- **Approve** – Activates users with a *Pending* state.
- **Suspend** – Temporarily disables the user account.

**1. Additional Options**

- **Add User** – Creates a new local user account.
- **User Max Session Idle Time** – Displays the configured timeout for automatic logout.

The currently logged-in administrator sees **(You)** next to their own entry and cannot delete themselves.

### *User Events*

The User Events page provides an audit trail of all user-related actions in the system.

It allows administrators to monitor logins, logouts, role changes, account updates, and enforcement actions such as suspensions or forced logouts.

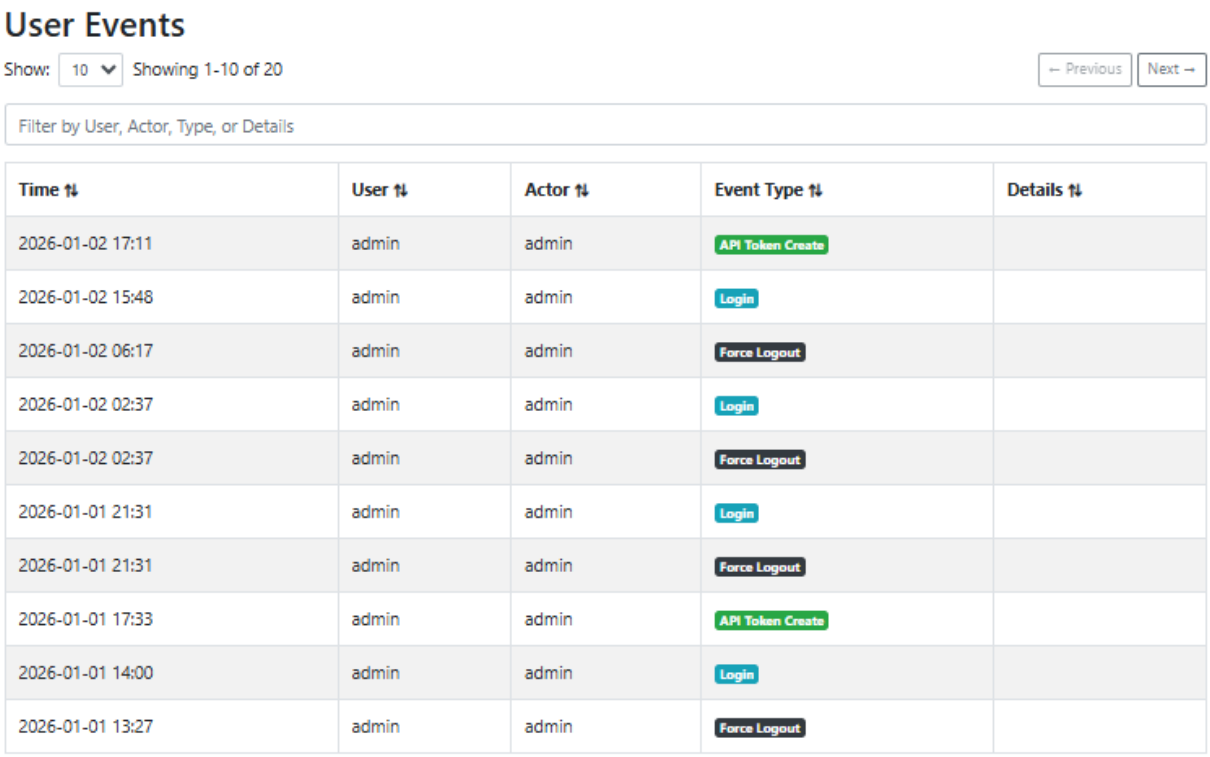
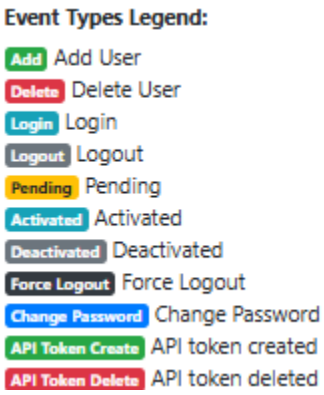



**Table Columns**

- **Time**
  Timestamp of when the event occurred.
- **User**
  The account affected by the event.
- **Actor**
  The user who performed the action
  *(e.g., an admin who suspended another user).*
- **Event Type**
  Categorizes the action, such as:

- **login** – User successfully logged in
- **Force Logout** – Admin forcibly logged out a user
- **Delete** – User account deleted
- **suspend** – User temporarily disabled
- **Change Password** – Password updated
- **Activate / Pending / Deactivated** – Account state transitions
- **API Token Create** – created new API token
- **API Token Delete** – deleted API token

- **Details**
  Extra information such as IP address or system notes.




**Event Types Legend**

- **Add** – User created
- **Delete** – User removed
- **Pending** – Awaiting approval
- **Activated** – User enabled
- **Deactivated** – User disabled
- **Force Logout** – Session terminated by admin
- **Change Password** – Password updated
- API Token Create – created new API token

- API Token Delete – deleted API token

**Details**
Shows additional context such as the source IP, actor name, or system-generated commentary.

### *Add User*

The **Add User** page allows administrators to create new **local** user accounts.


**2. Fields**

- **Username**
  The login name for the new account. Must be unique.
- **Password**
  The user's login password. This applies only to local users (LDAP users authenticate externally).
- **Role**
  Specifies the user's permission level:
  - **User** – Standard user privileges
  - **Admin** – Full administrative access, including user and CA management
- **Email**
  Email address associated with the account. Used for identification and audit purposes.

**3. Actions**

- **Add User**
  Creates the new account and redirects back to the Manage Users page.
- **Cancel**
  Returns to the Manage Users page without creating a user.

Once added User is at pending state and waits for admin approval

### *Register User*

The **Register** page allows new users to create a local account on the system.


Note : using LDAP and Configuration allow_self_registration = false disable this feature

**Fields**

- **Username**
  Must be 6–20 characters long and contain only letters, numbers, and underscores.
- **Email**
  A valid email address is required.

- **Password**
  Must be at least 6 characters long.
- **Confirm Password**
  Must match the password exactly.

**Actions**

- **Register**
  Creates the user account if all fields are valid.
  The account will appear in the *Manage Users* list, typically in a **Pending** state until approved by an administrator (unless auto-approval is enabled).
- **Login**
  Link to the login page for existing users.

Note : using LDAP and Configuration allow_self_registration = false disable this feature

# 14 Logs & Troubleshooting

## 14.1 Logs (Web UI)

The **Logs** page offers an integrated view of server activity.

**Features:**

- **View Last N Lines** — Default: 600
- **Auto-Refresh** — Refresh intervals configurable (default: 60 seconds)
- **Pause/Resume** — Freeze scrolling during inspection
- **Text Filtering** — Search for keywords or error strings

This viewer is helpful for:

- Enrollment errors (SCEP/EST)
- Signing issues
- Vault connection problems
- User authentication events
- OCSP/CRL generation logs

## 14.2 System Logs (systemd)

If running as a service:

```
journalctl -u pikachu-ca.service -f
journalctl -u pikachu-ca-healthcheck.service -f
```

Useful for:

- Server startup issues
- Port binding errors
- Crashes or restarts
- Healthcheck failures

## 14.3 Vault Troubleshooting

When Vault integration is enabled, the following messages should appear at server startup:

```
INFO [app] Vault integration is ENABLED
INFO [vault_client] Authenticated with Vault
INFO [app] Running in VAULT MODE - keys isolated in Vault
```

**Common Vault Issues**

❌ **Cannot authenticate to Vault**

- Missing VAULT_ROLE_ID or VAULT_SECRET_ID
- Wrong AppRole policy
- Incorrect Vault address

❌ **Signing fails in Vault mode**

- Missing PKI mount configured (pki-subca-rsa / pki-subca-ec)
- Certificate roles not configured for issuance
- CA mode mismatch (RSA profile but EC Vault engine enabled)

❌ **SCEP fails in Vault mode**

Expected behavior:

SCEP *always* uses file-based keys and ignores Vault.

Thus, a Vault authentication failure does **not** affect SCEP.
If SCEP fails, check:

- serial_file path
- Missing challenge password
- Incorrect content type or CSR encoding

## 14.4 Enrollment Troubleshooting

### SCEP Errors

| Symptom | Cause |
|---|---|
| “Missing challengePassword” | Feature enabled but password not provided or expired |
| “Invalid PKIMessage” | Incorrect CSR format or corrupted request |
| Certificate not issued | Sub-CA key mismatch or incorrect config.ini CA paths |

### EST Errors

| Symptom | Cause |
|---|---|
| “400 Bad Request” | CSR must be DER, not PEM |
| TLS handshake failure | Missing or wrong client certificate (mTLS mode) |
| No certificate issued | Vault signing role misconfigured |

## 14.5 Certificate Issues

**Certificate appears in UI but fails TLS**

- Missing intermediate in server chain
- Wrong SAN entries
- Required AIA/CRL URLs not added in server extensions
- Using RSA certificate when EC expected (or vice-versa)

**OCSP returns "unknown"**

- Certificate issued by different Sub-CA
- Sub-CA certificate missing from OCSP responder config

- Certificate not stored in database (manual import)

## 14.6 Service Health-check

Pikachu CA supports a periodic healthcheck using systemd timers.

The healthcheck will restart the service if:

- HTTPS endpoint does not respond
- Certificates or keys become unreadable
- The Flask server stops responding

Manually trigger:

```
systemctl start pikachu-ca-healthcheck.service
```

## 14.7 Database Troubleshooting

Use:

```
python migrate_db.py
```

This script will:

- Create tables if missing
- Add missing columns
- Create admin user if not present

Idempotent — safe to run repeatedly.

# Appendixes

## A.1. Openssl commands

### Generate RSA Kaey + CSR

#### *Generate Key*

```
openssl genrsa -out client1.key 2048
```

#### *Generate request*

prepare configuration to be added to the certificate request

for instance,

```
cat > client1.cnf <<EOL
[ req ]
default_bits       = 2048
default_md         = sha256
distinguished_name = req_distinguished_name
attributes         = req_attrs
prompt             = no

[ req_distinguished_name ]
CN                 = client1.example.com
O                  = My Organization
C                  = US

[ req_attrs ]
challengePassword  = SecretChallenge
EOL
```

#### *Make the request*

```
openssl req -new -key client1.key -config client1.cnf -out client1.csr
```

this request can be used for either UI signing or API based SCEP  or EST

### Convert CSR to DER (for EST)

```
openssl req -in client.csr -outform DER -out client.csr.der
```

### Inspect Certificate

```
openssl x509 -in cert.pem -noout -text
```

## A.2. MQTTs

The MQTT broker **Mosquitto** is commonly used as a server that leverages the CRL file to deny access to clients presenting revoked certificates.

Configuration updates must be applied in **mosquitto.conf** (refer to Paragraph 2.2 *installation / Complementary / MQTT Broker*).

```
crlfile /mosquitto/certs/crl_client_2.pem
```

The MQTT server denies connection attempts made with revoked certificates.
To ensure proper enforcement, the server must be periodically updated with the latest CRL.

For example, a CRL updater service can be added to the **docker-compose** configuration to automatically fetch and refresh the CRL file.

example adding crl updater to the docker compose:

```
  crl-updater:
    image: alpine:latest
    container_name: crl_updater
    volumes:
      - ./mosquitto/certs:/mosquitto/
      - /var/run/docker.sock:/var/run/docker.sock  # So we can signal the
mosquitto container
    # Install curl and bash, then run a loop:
    command: >
      sh -c "apk add --no-cache curl bash &&
      while true; do
         echo 'Downloading new CRL file...';
         curl -k https://openxpki.iot-rad.com:4443/downloads/crl --output
/mosquitto/certs/crl.pem;
         echo 'Triggering Mosquitto to reload its configuration...';
         docker kill --signal=SIGHUP mosquitto;
         echo 'Sleeping for 24 hours before next update...';
         sleep 86400;
      done"
    restart: unless-stopped
```

Note: not tested yet

## A.3. APIs

The UI **API** tab includes:

- CA chain download
- CRL download
- Certificate endpoints
- CSR endpoints
- Enrollment endpoints
- OCSP endpoint
- SCEP, EST URLs & examples

Note all https commands are Server-only TLS unless stated mTLS (Mutual TLS) for Both server and client authenticate with certificates. (default mTLS port 4443)

For complete examples, see the *APIs* page in the Web UI or visit /api.

| | | | |
|---|---|---|---|
| CA General | Download CA Chain | curl -k https://pikachu-ca.iot-rad.com/downloads/chain | Returns full CA chain file |
| | Download CRL | curl -k https://pikachu-ca.iot-rad.com/downloads/crl | Generates and downloads latest Certificate Revocation List |
| | Certificate Status | curl -k https://pikachu-ca.iot-rad.com/status/0xc75f573d9cb2b581 | Returns status as valid, revoked, or not found in JSON |
| | Expired Certificates | curl -k https://pikachu-ca.iot-rad.com/expired | Returns list of certificate IDs that are expired |
| | Download CSR | curl -k https://pikachu-ca.iot-rad.com/requests/1/download | Serves saved CSR if available |
| SCEP | SCEP CRL | sscep getcrl -d -u http://pikachu-ca.iot-rad.com/scep -c ca_rsa.crt -w crl.pem -l local.crt -k local.key | |
| | SCEP Enrolment | sscep enroll -d -v -u http://pikachu-ca.iot-rad.com/scep -c rad_ca_sub_rsa.crt -k client1.key -r client1.csr -l client1.crt | |
| | SCEP CA Certs | sscep getcap -d -u http://pikachu-ca.iot-rad.com/scep -c cap.pem | |
| EST | EST Enrollment | curl -k -X POST --data-binary @etx.csr.der https://openxpki.iot-rad.com/.well-known/est/simpleenroll -H "Content-Type: application/pkcs10" --output etx.crt.p7<br><br>estclient -server openxpki.iot-rad.com -insecure -cacerts ca-bundle.pem -key device.key -csr device.csr -out device.crt enroll | Accepts DER CSR and returns signed certificate in PKCS#7 |
| | EST CA Certs | curl -k https://pikachu-ca.iot-rad.com/.well-known/est/cacerts --output chain.crt<br><br>estclient -server openxpki.iot-rad.com -insecure cacerts -out ca-bundle.pem | Returns CA chain in PKCS#7 format |
| EST mTLS | EST Enrollment | curl --cert https.crt --key https.key -X POST --data-binary @etx.csr.der https://openxpki.iot- | Config.ini trusted_port variable 4443 |

| | | | |
|---|---|---|---|
| | | rad.com:4443/.well-known/est/simpleenroll -H "Content-Type: application/pkcs10" --output etx.crt.p7<br><br>estclient -server openxpki.iot-rad.com -cert https.crt -key https.key -cacerts ca-bundle.pem -key device.key -csr device.csr -out device.crt enroll | |
| | EST CA Certs | curl --cert https.crt --key https.key https://pikachu-ca.iot-rad.com:4443/.well-known/est/cacerts --output chain.crt<br><br>estclient -server openxpki.iot-rad.com -cert https.crt -key https.key cacerts -out ca-bundle.pem | Config.ini trusted_port variable 4443 |
| OCSP | OCSP Responder | openssl ocsp -reqout ocsp_request.der -issuer rad_ca_sub.crt -cert valid.crt -url http://pikachu-ca.iot-rad.com/ocsp -resp_text -respout ocsp_response.der | Returns OCSP status for a given certificate in DER format |
| CA General | Download CA Chain | curl -k https://pikachu-ca.iot-rad.com:5443/downloads/chain | Returns full CA chain file |
| | Download CRL | curl -k https://pikachu-ca.iot-rad.com:4443/downloads/crl | Generates and downloads latest Certificate Revocation List |
| | Certificate Status | curl -k https://pikachu-ca.iot-rad.com:4443/status/0xc75f573d9cb2b581 | Returns status as valid, revoked, or not found in JSON |
| | Expired Certificates | curl -k https://pikachu-ca.iot-rad.com/expired | Returns list of certificate IDs that are expired |
| | Download CSR | curl -k https://pikachu-ca.iot-rad.com:5443///requests/1/download | Serves saved CSR if available |

## A.4. Vault Setup Summary

### Minimal configuration

```
[VAULT]
enabled = true
address = http://127.0.0.1:8200
pki_rsa_path = pki-subca-rsa
pki_ec_path = pki-subca-ec
```

### Environment variables

```
export VAULT_ROLE_ID=...
export VAULT_SECRET_ID=...
export VAULT_ADDR="http://127.0.0.1:8200"
```

### Testing Vault Integration

```
tail -f logs/server.log | grep vault
```

## A.5. Automation Tests

Two pytest suites live here:

- test_certificates_ui.py: 17-step UI-style certificate lifecycle with 5s pauses between steps.
- test_certificates_api.py: API/PowerShell flows (basic endpoints, EST, EST mTLS, SCEP, SCEP with challenge) plus artifact/DB cleanup.

### UI test

test_certificates_ui.py: 17-step UI-style certificate lifecycle with 5s pauses between steps.

### *Coverage*

| # | Action | What it does |
|---|---|---|
| 1 | Login as admin | Authenticate as admin for the session |
| 2 | Create key | Generate a keypair |
| 3 | Create req template | Create CSR template (REQ) |
| 4 | Create ext template | Create extension template (EXT) |
| 5 | Create CSR | Create a CSR using key + template |
| 6 | Create enrollment policy | Add enrollment policy with validity/EXT config |
| 7 | Create challenge password | Generate a challenge password (UI) |
| 8 | Sign CSR | Issue certificate from CSR + policy |
| 9 | Check certificate in list | Verify issued cert appears in UI list |
| 10 | Download certificate | Download issued certificate |
| 11 | Revoke certificate | Revoke the certificate |
| 12 | Delete certificate | Remove certificate record |
| 13 | Delete enrollment policy | Remove the policy |
| 14 | Delete challenge password | Remove the generated challenge password |
| 15 | Delete CSR | Delete CSR record |
| 16 | Delete templates | Delete REQ/EXT templates |
| 17 | Delete key | Delete keypair |

### *Command*

```
$env:PYTHONPATH=(Resolve-Path .); .venv\Scripts\pytest.exe
tests_repo/test_certificates_ui.py --capture=tee-sys --self-contained-html --
html=tests_repo/reports/pikachu_test_ui_full_$(Get-Date -Format 'yyyy-MM-
dd_HH-mm').html
```

### *Report*

**Summary**

17 tests took 00:01:28.

(Un)check the boxes to filter the results.

0 Failed, 17 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns — Show all details / Hide all details

| Result | Test | Duration | Links |
|---|---|---|---|
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_01_login_as_admin] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_02_create_key] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_03_create_req_template] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_04_create_ext_template] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_05_create_csr] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_06_create_enrollment_policy_1d] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_07_create_challenge_password] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_08_sign_csr_with_policy] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_09_check_certificate_in_list] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_10_download_certificate] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_11_revoke_certificate] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_12_delete_certificate] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_13_delete_enrollment_policy] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_14_delete_challenge_password] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_15_delete_certificate_request_csr] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_16_delete_certificate_templates_req_ext] | 00:00:05 | |
| Passed | tests_repo/test_certificates_ui.py::test_certificate_lifecycle_step[step_17_delete_key] | 00:00:05 | |

# API test

API/PowerShell flows (basic endpoints, EST, EST mTLS, SCEP, SCEP with challenge) plus artifact/DB cleanup.

### *Coverage*

| # | Action | What it does |
|---|---|---|
| 1 | test_api_basic_endpoints | Basic API reachability |
| 2 | test_est_enrollment_via_estclient_go | EST enrollment via estclient-go |
| 3 | test_est_enrollment_via_estclient_go_mtls | EST mTLS enrollment via estclient-go |
| 4 | test_sscep_core | SCEP core flow (no challenge password) |
| 5 | test_sscep_with_challenge_password | SCEP flow using challenge password via API token |

#### *Command*

```
$env:PYTHONPATH=(Resolve-Path .); .venv\Scripts\pytest.exe
tests_repo/test_certificates_api.py --capture=tee-sys --self-contained-html -
-html=tests_repo/reports/pikachu_test_api_full_$(Get-Date -Format 'yyyy-MM-
dd_HH-mm').html
```

#### *Report*

**Environment**

| | |
|---|---|
| Python | 3.12.7 |
| Platform | Windows-11-10.0.22631-SP0 |
| Packages | • pytest: 9.0.2<br>• pluggy: 1.6.0 |
| Plugins | • html: 4.1.1<br>• metadata: 3.1.1 |

**Summary**

5 tests took 00:00:27.

(Un)check the boxes to filter the results.

0 Failed, 5 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns — Show all details / Hide all details

| Result | Test | Duration | Links |
|---|---|---|---|
| Passed | tests_repo/test_certificates_api.py::test_api_basic_endpoints | 00:00:02 | |
| Passed | tests_repo/test_certificates_api.py::test_est_enrollment_via_estclient_go | 00:00:01 | |
| Passed | tests_repo/test_certificates_api.py::test_est_enrollment_via_estclient_go_mtls | 00:00:01 | |
| Passed | tests_repo/test_certificates_api.py::test_sscep_core | 431 ms | |
| Passed | tests_repo/test_certificates_api.py::test_sscep_with_challenge_password | 00:00:22 | |

# A.6. Unit Tests SCEP, EST, OCSP

## SCEP Test

```
.\tests\scripts\test_sscep.ps1
```

## SCEP Test w/ challenge password

Generate one from /challenge_passwords

```
.\tests\scripts\test_sscep_pass.ps1 B2D1BDD0A46BBF3D8755A63D5034DE86
```

## EST Test (insecure)

```
.\tests\scripts\test_estclient_curl.ps1
Or

.\tests\scripts\test_estclient_go.ps1
```

## EST Test (mTLS)

```
.\tests\scripts\test_estclient_curl_mtls.ps1

Or

.\tests\scripts\test_estclient_go_mtls.ps1
```

## OCSP Test

```
.\tests\scripts\test_ocsp.ps1
```

## Challenge Passord Test

```
.\tests\scripts\test_challenge_password_api.ps1
```

# Change History

| Date of Issue | Revision | Author | Responsible | Change Summary | Page Count |
|---|---|---|---|---|---|
| 8-Apr-2025 | 1.0.0 | Uzi G. | | Document baseline | 42 |
| 16-Apr-2025 | 1.0.1 | Uzi G. | | Update UI, CA details, Validity time manage | 43 |
| 23-Apr-2025 | 1.0.2 | Uzi G. | | Update UI, New Navigate tabs. configuration based | 52 |
| 18-Jun-2025 | 1.0.3 | Uzi G. | | Add manufacture certificate URL end point, Add generation of root and sub certificate procedure | 57 |
| 19-Nov-2025 | 1.0.4 | Uzi G. | | OCSP tested with mocana stack, added secret protection for various commands, added PFX format, create profiles without templates by editing | 64 |
| 12-Dec-2025 | 2.0.0 | Uzi G. | | Fix SCEP , add Multitenancy, Users management fix bugs | 144 |
| 2-Jan-2026 | 2.0.1 | Uzi G. | | Challenge password API , API Token Management , Automation tests | 150 |