Data Exchange Platform

Software Design Document

Tufan Küpeli

2023

Table of Contents

[1. Introduction 4](#_Toc147283053)

[1.1 Objective 4](#_Toc147283054)

[1.2 Scope 4](#_Toc147283055)

[1.3 Structure of the Document 4](#_Toc147283056)

[1.4 Design Goals 5](#_Toc147283057)

[2. System Overview 6](#_Toc147283058)

[2.1 System Architecture 6](#_Toc147283059)

[2.2 System Components 6](#_Toc147283060)

[2.2.1 Central Server 6](#_Toc147283061)

[2.2.2 Configuration Proxy 7](#_Toc147283062)

[2.2.3 Security Server 7](#_Toc147283063)

[2.2.4 Information System 8](#_Toc147283064)

[2.2.5 Time Stamping Authority (TSA) 8](#_Toc147283065)

[2.2.6 Certification Authority (CA) 8](#_Toc147283066)

[2.2.7 Monitoring 9](#_Toc147283067)

[2.2.7.1 Infrastructure 9](#_Toc147283068)

[2.2.7.2 Protocol 9](#_Toc147283069)

[2.3 System Interfaces 9](#_Toc147283070)

[2.3.1 Security Server Interface 9](#_Toc147283071)

[2.3.2 Central Server Interface 9](#_Toc147283072)

[2.3.3 Monitoring Interfaces 9](#_Toc147283073)

[2.3.3.1 Infrastructure Interface 9](#_Toc147283074)

[2.3.3.2 Protocol Interface 10](#_Toc147283075)

[2.4 System Protocols 10](#_Toc147283076)

[2.4.1 Message Protocol 10](#_Toc147283077)

[2.4.2 Configuration Downloading Protocol 10](#_Toc147283078)

[2.4.3 Metadata Protocol 11](#_Toc147283079)

[2.4.4 Transport Protocol 11](#_Toc147283080)

[2.4.5 Download Document Protocol 11](#_Toc147283081)

[2.4.6 Management Service Protocol 11](#_Toc147283082)

[2.4.7 Time Stamping Protocol 12](#_Toc147283083)

[2.4.8 Monitoring Protocol 12](#_Toc147283084)

[2.4.8.1 Infrastructure 12](#_Toc147283085)

[2.4.8.2 Protocol 12](#_Toc147283086)

[3. Architectural Design 13](#_Toc147283087)

[3.1 High-Level Design 13](#_Toc147283088)

[3.1.1 System Block Diagram 13](#_Toc147283089)

[3.1.2 Subsystem Descriptions 14](#_Toc147283090)

[3.1.2.1 Security Server Architecture 14](#_Toc147283091)

[3.1.2.2 Central Server Architecture 22](#_Toc147283092)

[3.1.2.3 Configuration Architecture 25](#_Toc147283093)

[3.1.2.4 Monitoring Architecture 28](#_Toc147283094)

[3.2 Detailed Design 32](#_Toc147283095)

[3.2.1 Interface Specifications 32](#_Toc147283096)

[3.2.2 Protocol Specifications 32](#_Toc147283097)

[3.3 Use Cases 32](#_Toc147283098)

[4. Database Design 33](#_Toc147283099)

[4.1.1 Central Server 33](#_Toc147283100)

[4.1.2 Security Server 34](#_Toc147283101)

[4.1.3 Log 34](#_Toc147283102)

[5. User Interface Design 35](#_Toc147283103)

[5.1 User Interface Mockups 35](#_Toc147283104)

[5.2 User Interaction Flow 35](#_Toc147283105)

[5.3 Screen Descriptions 35](#_Toc147283106)

[6. Software Development Environment 36](#_Toc147283107)

[6.1 Programming Languages 36](#_Toc147283108)

[6.2 Development Tools 36](#_Toc147283109)

[6.3 Version Control 36](#_Toc147283110)

[6.4 Build and Deployment Process 36](#_Toc147283111)

[6.5 Open Standards 36](#_Toc147283112)

[6.6 Data 36](#_Toc147283113)

[7. Quality Assurance and Testing 36](#_Toc147283114)

[7.1 Test Strategy 36](#_Toc147283115)

[7.1.1 Test Types 36](#_Toc147283116)

[7.1.2 Test Runs 36](#_Toc147283117)

[7.2 Test Cases 37](#_Toc147283118)

[7.3 Testing Tools 37](#_Toc147283119)

[7.4 Test Results 37](#_Toc147283120)

[8. Deployment and Maintenance 38](#_Toc147283121)

[8.1 Deployment Plan 38](#_Toc147283122)

[8.1.1 Software Deployment 38](#_Toc147283123)

[8.1.2 Security Settings and Controls 38](#_Toc147283124)

[8.1.3 Monitoring and Event Tracking 38](#_Toc147283125)

[8.1.4 End-User Testing 38](#_Toc147283126)

[8.1.5 Planned Downtime and Improvements 38](#_Toc147283127)

[8.2 Maintenance Procedures 39](#_Toc147283128)

[8.2.1 Bug Solving 39](#_Toc147283129)

[8.2.2 Change Management 39](#_Toc147283130)

[8.2.3 New Function 39](#_Toc147283131)

[9. Appendices 40](#_Toc147283132)

[9.1 Glossary 40](#_Toc147283133)

[9.2 Acronyms and Abbreviations 40](#_Toc147283134)

[9.3 References 40](#_Toc147283135)

# 1. Introduction

In today's interconnected digital landscape, the seamless and secure exchange of data among organizations stands as a cornerstone of collaborative success. As we navigate an era defined by dynamic data formats, diverse communication protocols, and stringent security requirements, the need for a robust and adaptable solution for data exchange has never been more critical. This System Design Document (SDD) outlines the architecture and design principles behind our Unified Data Exchange Platform, a cutting-edge software system engineered to facilitate the exchange of information while safeguarding data integrity, confidentiality, and availability.

## 1.1 Objective

The primary objective of this SDD is to provide a comprehensive insight into the architectural decisions, key components, security mechanisms, scalability considerations, and user and permissions models that underpin our Unified Data Exchange Platform. This document serves as a reference guide for architects, developers, and stakeholders involved in the creation and implementation of this sophisticated system.

## 1.2 Scope

Our platform aims to transcend the boundaries of traditional data exchange systems. It seeks to address the evolving needs of modern organizations by providing a unified solution capable of accommodating diverse data formats, seamlessly transitioning data between different systems, and meticulously managing permissions to ensure data remains accessible only to authorized users. Moreover, it places a paramount emphasis on the security and privacy of exchanged data, making it a dependable choice for organizations of all sizes and domains.

## 1.3 Structure of the Document

This SDD is structured into distinct sections, each of which delves into specific aspects of the system's design and functionality. The document begins with this introduction, followed by sections that explore the architectural rationale, key components, security mechanisms, scalability and performance considerations, and the user and permissions model. These sections provide a holistic view of the Unified Data Exchange Platform's inner workings, guiding the reader through its intricacies step by step.

As we embark on this journey to design a solution that bridges data exchange challenges, we invite you to explore this SDD and gain a comprehensive understanding of the thought processes, methodologies, and technologies that shape our Unified Data Exchange Platform. Together, we strive to elevate data exchange to a new standard, empowering organizations to collaborate, innovate, and thrive in the data-driven world of tomorrow.

## 1.4 Design Goals

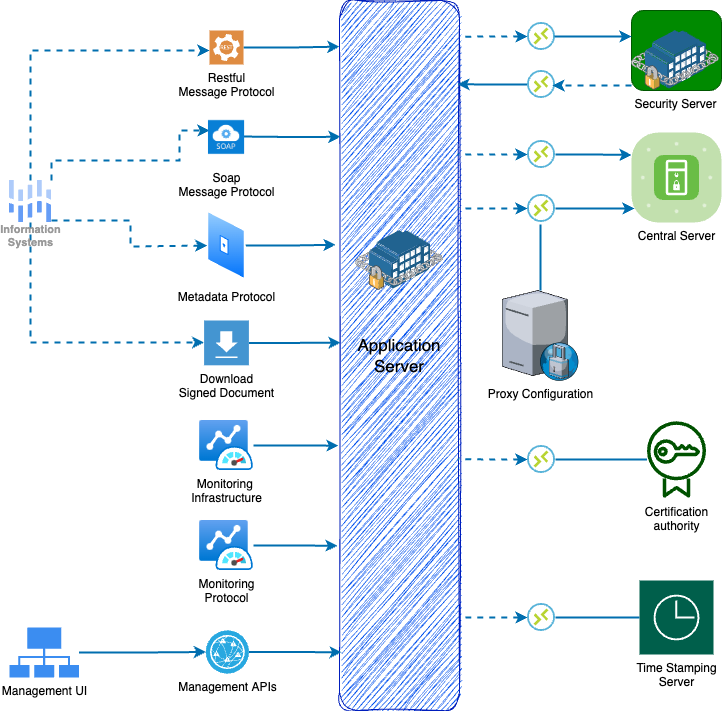
Our design goals encapsulate the essence of our vision for the Unified Data Exchange Platform:

* **Unified Data Handling:** Design a system capable of handling a wide array of data formats, communication protocols, and industry standards, ensuring compatibility with diverse organizational systems.
* **Data Integrity:** Prioritize data integrity by implementing robust data transformation mechanisms that guarantee the preservation of data quality and accuracy during exchanges.
* **Granular Permission Management:** Enforce granular permission controls, allowing organizations to define and manage user access rights with precision, including read-only, read-write, and administrative privileges.
* **Security and Privacy:** Integrate industry-leading security protocols and mechanisms, including encryption, secure data storage, and rigorous authentication and authorization mechanisms, to ensure data remains confidential and protected during transit and storage.
* **Scalability and Performance:** Architect the system for scalability, enabling it to seamlessly accommodate increasing volumes of data and user loads without compromising performance.
* **Interoperability:** Facilitate seamless interoperability with diverse organizational systems through well-defined APIs and connectors, promoting smooth data exchange.
* **Monitoring and Auditability:** Implement robust mechanisms for real-time data exchange monitoring and comprehensive logging to support audit and compliance requirements.
* **Error Handling:** Develop a fault-tolerant system with effective error handling and notification mechanisms to promptly address and resolve issues during data exchange.

# 2. System Overview

The Unified Data Exchange Platform is designed to revolutionize the way organizations share data by providing a unified solution that transcends the complexities of varying data formats, protocols, and security requirements. At its core, the system functions as a robust and intelligent mediator, facilitating the exchange of information while maintaining data integrity, confidentiality, and availability. It empowers organizations to collaborate seamlessly, innovate with agility, and harness the full potential of their data resources.

## 2.1 System Architecture



## 2.2 System Components

### 2.2.1 Central Server

The Central Server maintains a directory of system members and their associated Security Servers. It also houses the security protocol of the system, featuring:

* A roster of reputable certificate authorities
* A list of verified time-stamping services
* Various operational settings

Think of the Central Server as a master database that stores records of all system members and their Security Servers, along with security policies.

All this information is disseminated to the Security Servers through HTTP, forming the global settings that guide message processing. The System Operator oversees managing the Central Server.

### 2.2.2 Configuration Proxy

The Configuration Proxy serves as an optional medium to distribute the global settings to Security Servers. It first pulls this information from the Central Server and then securely passes it on.

The Configuration Proxy acts like an intermediary that fetches and shares configuration data, thereby reducing the load on the Central Server.

Using the Configuration Proxy enhances system reliability and decreases the Central Server's operational burden. Management of the Configuration Proxy falls under the system's purview.

### 2.2.3 Security Server

The Security Server acts as the system's gateway, serving as the crucial point of entry for both generating and utilizing services within the system. It plays a pivotal role in managing service calls and responses across various organizations. Within the system infrastructure, the Security Server is responsible for safeguarding security aspects, including the management of keys for authentication and digital signatures, secure message transmission, digital signature proof generation, time-stamping, and log maintenance.

A single Security Server is capable of hosting multiple organizations, following a multi-tenancy approach. The organization responsible for overseeing the Security Server is referred to as the server owner, while the hosted organizations are considered clients of the Security Server.

The Security Server administers two types of keys. Authentication keys are allocated to a specific Security Server and are used to establish secure communication channels with other Security Servers. On the other hand, signing keys are designated to clients of the Security Server and are employed for signing exchanged messages. Certificates for these keys are issued exclusively by a trusted certification authority,

To facilitate message mediation, the Security Server must continuously maintain an up-to-date global configuration. This configuration is routinely retrieved from the Central Server and is stored locally for use during message processing. The Security Server remains operational as long as it possesses a valid local copy of the global configuration.

The Security Server includes an internal client-side load balancer and also supports external load balancing.

### 2.2.4 Information System

Within the Information System, there are Producer and Consumer organizations that interact with the system by producing and/or consuming services. These organizations are owned by members of the system. The system offers support for both consuming and producing REST and SOAP services. However, it's important to note that the system doesn't facilitate automatic conversions between different message and service types.

For a service consumer organization, the Security Server serves as the primary entry point to access all the services offered by the system. Consumers can explore registered system members and the services they provide by utilizing the system's protocols.

On the other hand, service provider organizations are responsible for implementing REST and/or SOAP services and making them available within the system. Existing REST services can be published without the need for modifications; they can be used as-is. In contrast, SOAP services need to conform to the system's SOAP message protocol. Service descriptions for REST services are defined using the OpenAPI3 specification, while service descriptions for SOAP services are defined using WSDL. Service consumers can access these service descriptions through the system's protocol.

### 2.2.5 Time Stamping Authority (TSA)

Within the system, the Security Server plays a crucial role in time-stamping and logging all messages. The primary objective of time-stamping is to confirm the existence of data items at specific points in time.

This process relies on the Time-Stamping Authority (TSA), which provides the time-stamping service used by the Security Server to time-stamp both incoming and outgoing requests and responses. It's important to note that only trusted TSAs, as defined in the Central Server, are eligible for use.

The time-stamping authority is required to implement the time-stamping protocol supported by the system. The system employs batch time-stamping, which effectively manages the load on the time-stamping service.

Interestingly, this load isn't determined by the volume of messages exchanged within the system; instead, it depends on the number of Security Servers present in the system.

### 2.2.6 Certification Authority (CA)

The Certification Authority (CA) is responsible for issuing certificates to two key entities within the system:

1. Security Servers receive authentication certificates, and system member organizations receive signing certificates.
2. Authentication certificates are instrumental in securing the connection between two Security Servers, while signing certificates are utilized for digitally signing messages sent by system members.

It's important to note that only certificates issued by trusted certification authorities, as specified in the Central Server, are permissible for use.

To ensure the validity of signing and authentication certificates, the Security Server employs the OCSP. Each Security Server takes on the responsibility of verifying the validity of its certificates and subsequently shares this information with other Security Servers during the message exchange process. This ensures that only Security Servers equipped with valid signing and authentication certificates can engage in message exchanges with other Security Servers.

### 2.2.7 Monitoring

#### 2.2.7.1 Infrastructure

The infrastructure monitoring collects data regarding the operational surroundings of server and offers this information to external monitoring systems through suitable interfaces.

#### 2.2.7.2 Protocol

The purpose of the operational monitoring daemon is to gather and retain operational information from the security server of the Systems and provide access to this data for external monitoring systems through compatible interfaces.

## 2.3 System Interfaces

### 2.3.1 Security Server Interface

The user interface of the security server is employed by the security server administrator to set up and oversee the security server's configuration and administration.

### 2.3.2 Central Server Interface

The user interface for the central server is utilized by the central server administrator to perform configuration and administration tasks for the central server.

### 2.3.3 Monitoring Interfaces

#### 2.3.3.1 Infrastructure Interface

The infrastructure monitoring interface is designed to provide responses to inquiries regarding infrastructure monitoring data, which are initiated by the security server's server-proxy interface. The collection of infrastructure monitoring data is carried out by the infrastructure monitoring service.

#### 2.3.3.2 Protocol Interface

This interface serves as the conduit through which external monitoring systems retrieve operational data from the security server. The protocol employed here follows a **synchronous RPC**-style model, with initiation initiated by the external monitoring system.

## 2.4 System Protocols

### 2.4.1 Message Protocol

The Message Protocol serves as the communication medium between service client and service provider information systems in their interaction with the security server.

This protocol follows a **synchronous Remote Procedure Call** (RPC) style, which can be initiated either by the client information system or by the security server of the service provider.

System offers two distinct Message Protocols: one for **SOAP** and another for **REST**. Both protocols are built upon SOAP/REST over HTTP(S) and incorporate additional header fields to identify the service client and the invoked service.

These Message Protocols, in conjunction with the Message Transport Protocol, constitute the fundamental framework for data exchange. In situations where any of the involved components become unavailable, data exchange becomes impossible. To enhance component availability, the architecture incorporates redundancy mechanisms.

### 2.4.2 Configuration Downloading Protocol

Configuration clients retrieve the generated global configuration files from the central server.

The configuration download protocol operates synchronously and is provided by the central server. Configuration clients, including security servers and configuration proxies, utilize this protocol.

This protocol is built upon HTTP and MIME multipart messaging. To ensure the integrity of the configuration, it is digitally signed by the central server, guarding against any unauthorized modifications. Typically, the configuration comprises multiple components, and the protocol enables configuration clients to detect changes and download only the modified sections.

Security servers in the system store a local copy of the configuration, regularly refreshing it from their individual configuration sources. This cached data has a predefined validity period, typically longer than the update interval configured for the client-side configurations. During the validity period of the cached data, security servers operate smoothly without any interruptions. However, the presence of an outdated data significantly curtails the administrative control of security server administrators and blocks security servers from handling incoming requests. Consequently, brief interface downtime is considered acceptable, provided it falls within the duration specified by the configured configuration validity period.

### 2.4.3 Metadata Protocol

Clients of the Metadata Protocol, namely service client information systems, utilize it to retrieve vital system instance information. This encompasses the identification of system members, a comprehensive list of services offered by these members, and access to WSDL service descriptions. This protocol employs a **synchronous RPC** style.

To enhance simplicity in client service implementation, some of the information services are designed as straightforward HTTP(S) GET requests.

### 2.4.4 Transport Protocol

The Transport Protocol serves as the medium through which the security server conducts the exchange of service requests and service responses. This protocol adheres to a **synchronous RPC** style.

Built on the foundation of HTTPS, the Transport Protocol enforces mutual certificate-based TLS authentication. In practice, this means that both the client and the service provider information systems are required to present valid certificates for authentication. The SOAP/REST messages originating from the client and the service provider client are encapsulated within a MIME multipart message.

### 2.4.5 Download Document Protocol

The document service offers information systems the capability to retrieve containers from the security server's message log. Furthermore, it offers a streamlined method for acquiring the global configuration, which can be used to validate these containers.

This protocol follows a **synchronous RPC**-style approach and is instigated by the client information system. The service is realized through **HTTP(S) GET** requests. The Download protocol is specifically employed by the client for retrieving data stored within the security server.

### 2.4.6 Management Service Protocol

Tasks such as enrolling a security server client or revoking an authentication certificate are part of the functions handled by the management services. Protocol adheres to a **synchronous RPC**-style model and is furnished by the member management and registration web services.

These services are realized as standard services provided by the organization responsible for overseeing the system instance. One noteworthy exception is the authentication certificate registration service, which is directly incorporated into the registration web service.

Generally, the availability of the management services does not hold critical significance for the system's operation, and therefore, it is not an overriding concern. Should the management services become temporarily inaccessible, this does not disrupt the core functions of the system. It's worth noting that the operations conducted via the management services are not constrained by time sensitivity.

### 2.4.7 Time Stamping Protocol

Protocol serves a crucial role for security servers, ensuring the long-term validity and integrity of exchanged messages. Security servers meticulously log all messages along with their corresponding signatures. To establish long-term verifiability, these message logs undergo periodic time-stamping.

This protocol operates in a synchronous manner and is supplied by the time-stamp authority. However, security servers employ the time-stamping protocol in an asynchronous fashion. Here's how it works: Security servers log all messages involved in exchanges with other security servers. These messages are asynchronously time-stamped using batch time-stamping. This approach offers several advantages, including the decoupling of message exchange availability from time-stamping authority availability, reduced latency in message exchange, and a lighter load on the time-stamping authority.

Given this asynchronous usage, temporary unavailability of the service does not directly disrupt message exchange operations. However, if security servers fail to time-stamp accumulated messages over an extended period, it can potentially impede the ability to establish the precise timing of these message exchanges. To mitigate this risk, security servers take proactive measures. The maximum allowable time gap between message logging and acquiring a time stamp is defined by the central server owner and may vary between different instances of the system.

### 2.4.8 Monitoring Protocol

#### 2.4.8.1 Infrastructure

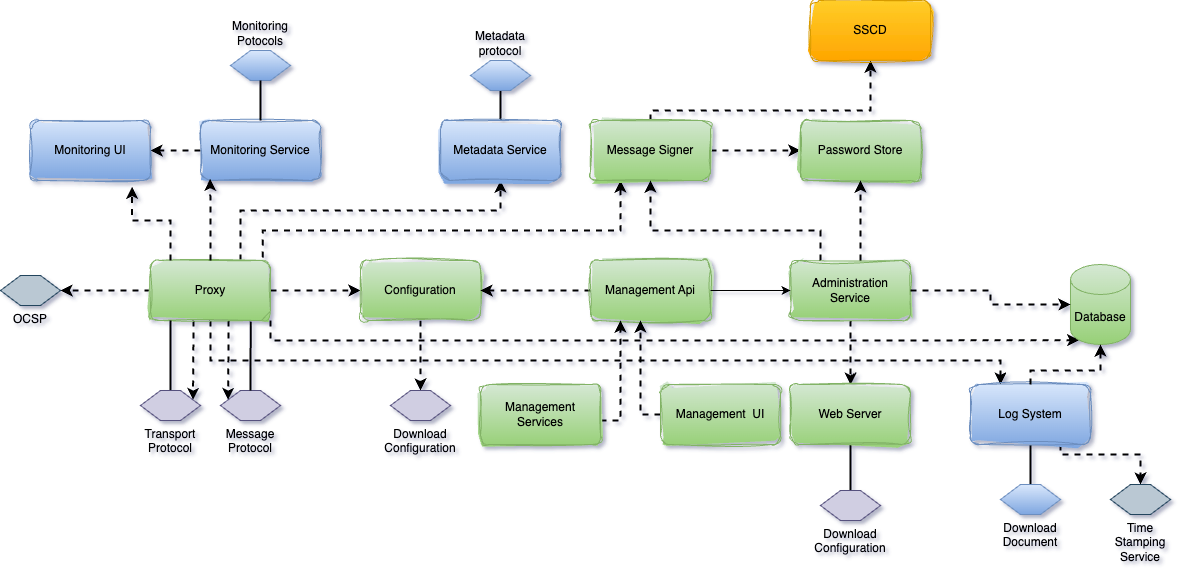
The infrastructure monitoring interface is designed to provide responses to inquiries regarding infrastructure monitoring data, which are initiated by the security server's server-proxy interface. The collection of infrastructure monitoring data is carried out by the infrastructure monitoring service.

#### 2.4.8.2 Protocol

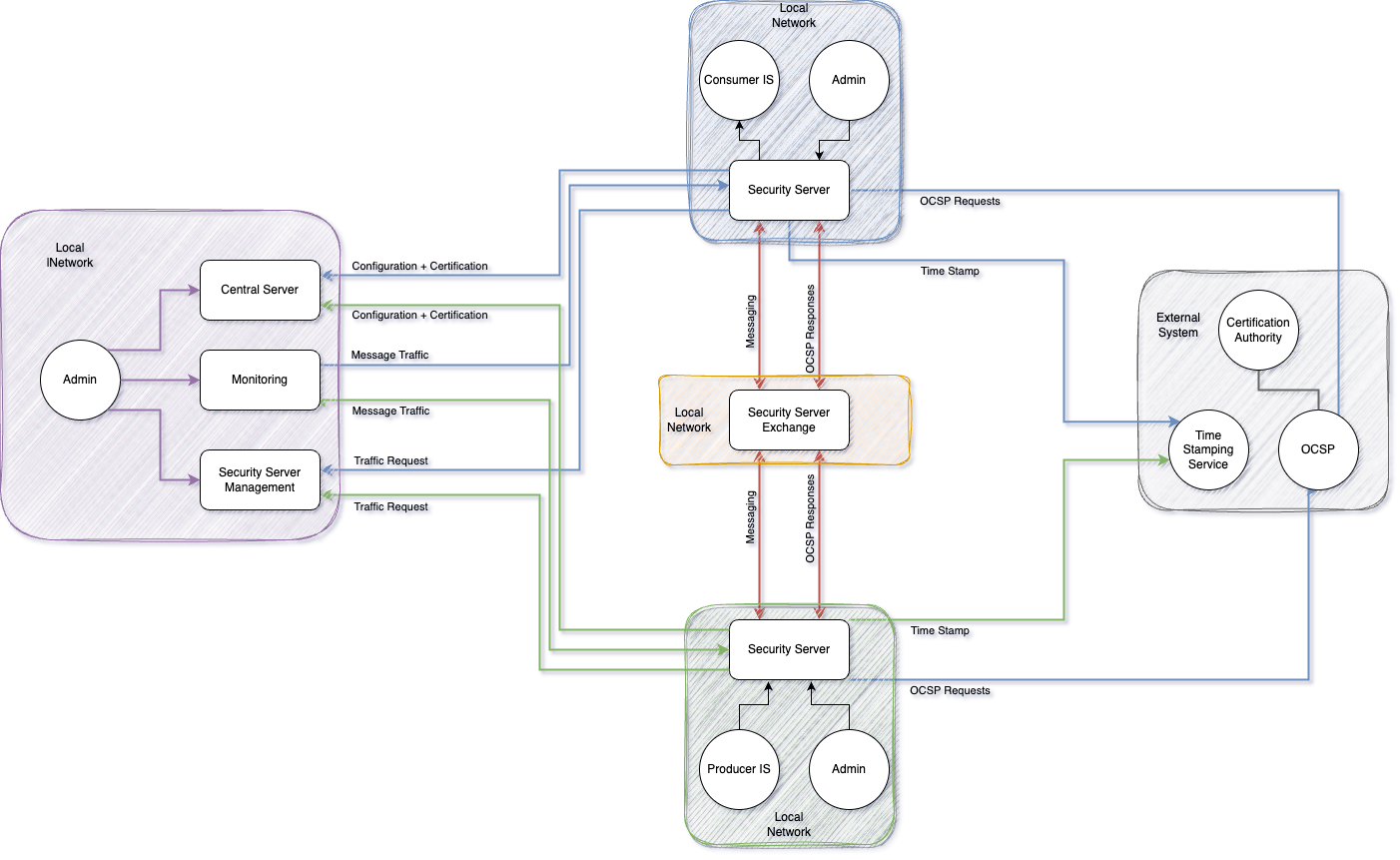
This protocol is employed by the security server to transfer its cached operational monitoring data into the operational monitoring daemon's database. It operates as a **synchronous RPC**-style protocol.

# 3. Architectural Design

## 3.1 High-Level Design



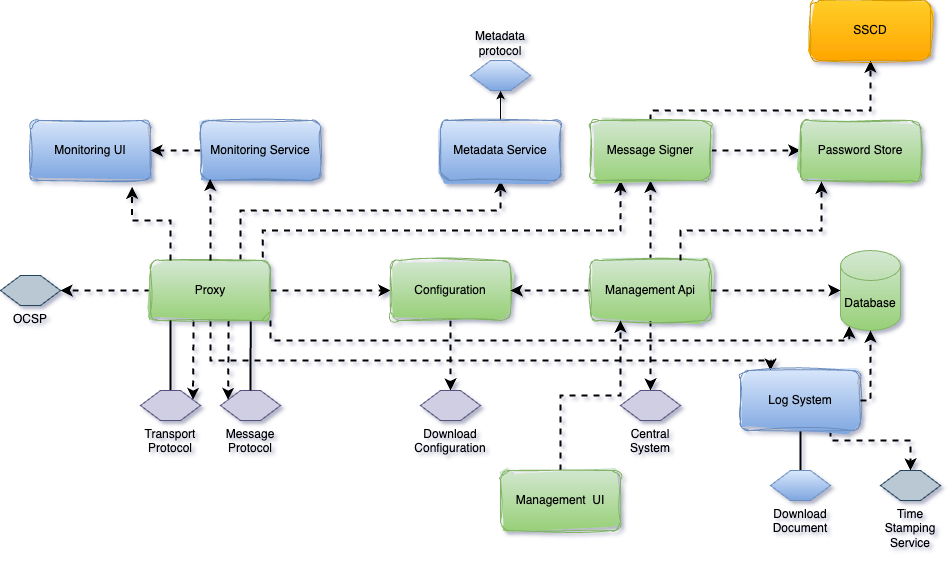
### 3.1.1 System Block Diagram



### 3.1.2 Subsystem Descriptions

#### 3.1.2.1 Security Server Architecture

Component of Security Server Architecture



##### 3.1.2.1.1 Components

###### 3.1.2.1.1.1 Proxy

The proxy has the duty of facilitating communication between service clients and service providers. Messages are sent across the public Internet, and the proxy guarantees the security of the communication through the utilization of digital signatures and encryption. This element represents an independent Java daemon application.

###### 3.1.2.1.1.2 Log System

Captures and stores all regular messages passing through the security server in a database. These messages are saved alongside their associated signatures, which are also timestamped. The primary purpose of the message log is to furnish a means of verifying the reception of a request/response message to a third party.

Periodically, the system archives log records from the database as signed documents on disk and removes archived log records from the database. By default, all signed documents on disk are organized together, although the option to group them by member or subsystem is also available.

Additionally, a service is provided to retrieve signed documents, containing the stored information, from the database.

###### 3.1.2.1.1.3 Metadata Service

Offers functions that can be utilized by participants to identify the services accessible through the security server.

###### 3.1.2.1.1.4 Monitor Services

3.1.2.1.1.4.1 Protocol Monitoring

Supplies functions that participants can utilize to retrieve operational monitoring information from the security server. These methods initiate requests for data from the local protocol service using SOAP XML requests and subsequently relay the SOAP XML responses to the caller.

3.1.2.1.1.4.2 Infrastructure Monitoring

Supplies a method that participants can employ to obtain environmental data from the security server. This method initiates a request for data from the local monitoring service through the interface and subsequently converts the received data into a SOAP XML response.

###### 3.1.2.1.1.5 Message Signer

It is activated by the proxy module for tasks like signing messages and validating their integrity. Moreover, the user interface interacts with the signer when generating authentication and signing keys or requesting certificates. This module operates independently as a Java daemon application.

###### 3.1.2.1.1.6 Database

Store configurations and serve this. PostgreSQL or MySQL. Detailed ERD in Database design section.

###### 3.1.2.1.1.7 UI Frontend

The user interface of the security server enables users to oversee the configuration of the security server. This user interface is implemented as a one-page web application, which communicates with the management REST API for the purpose of reading and adjusting the configuration. To obtain its resources, such as images, stylesheets, and JavaScript files, the user interface retrieves them from the web application hosting the management REST API.

###### 3.1.2.1.1.8 Management API

The management API provides endpoints that allow for the retrieval and modification of the security server's configuration. These endpoints are utilized by the user interface frontend and are also accessible for standalone API clients. (OpenAPI3 & Spring Boot & Tomcat)

###### 3.1.2.1.1.9 Configuration

The configuration client has the role of fetching global configuration files from remote sources. The location of these global configuration files is determined based on the anchor file uploaded through the security server user interface. This component operates independently as a Java daemon application.

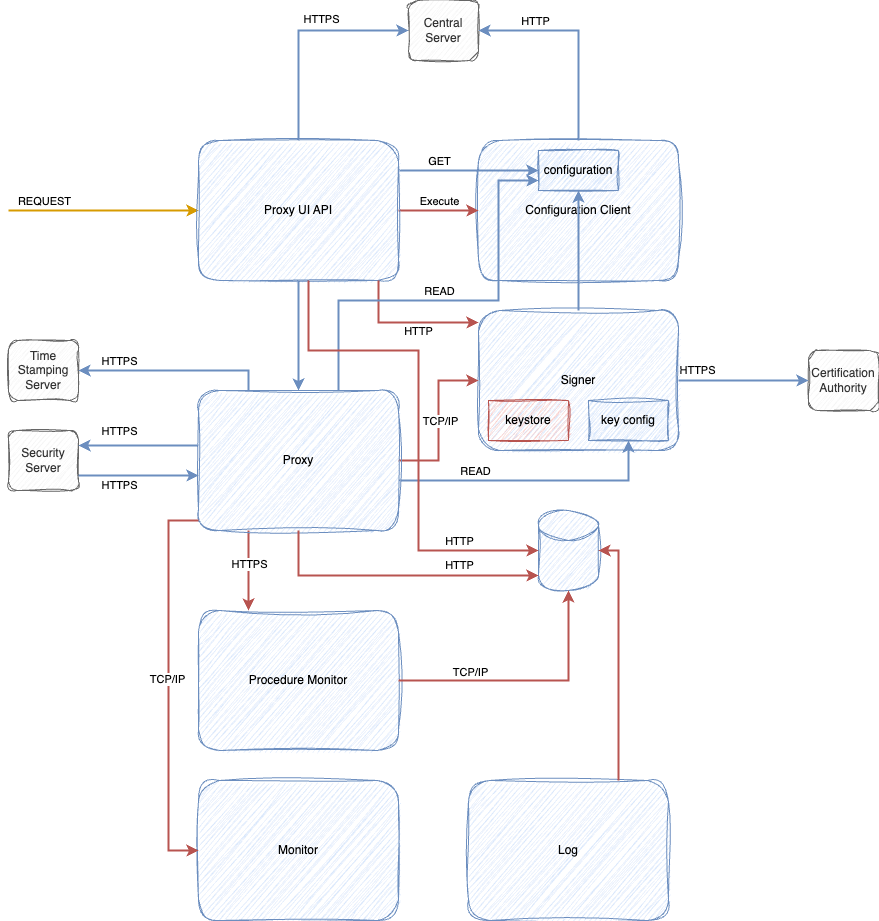
###### 3.1.2.1.1.10 Password Store

It stores security token passwords within a shared memory segment of the operating system, which is accessible to both the security server interface and the signer. This setup enables the persistence of security token logins even after the security server is restarted, all while ensuring the security of the stored passwords.

###### 3.1.2.1.1.11 SSCD

The SSCD (Secure Signature Creation Device) is an elective hardware component that delivers a secure cryptographic signature creation capability to the signer. Maybe **HSM**

##### 3.1.2.1.2 Process



###### 3.1.2.1.2.1 Proxy UI API

The Proxy UI API serves as the interface for the Security Server user interface and provides a REST API for management operations.

**Encapsulated Data**

* The Proxy UI API reads and writes various data from the filesystem, including the configuration anchor, configuration backups, and internal TLS certificate.
* It also reads and modifies the Security Server configuration using the PostgreSQL database server configuration, with the database model specified in section 4.1.2.
* Additionally, the Proxy UI API reads global configuration from the filesystem.
* Data related to token keys, certificates, and CSRs is accessed through the signer component.

**Messaging**

* The interface handles requests for serving content to the user interface as well as requests for REST API calls and is directly exposed to the outside world.
* It communicates with Central Server's management services interface.
* Global configuration is downloaded from the Central Server using the configuration client and stored on disk.
* It communicates with the configuration client in different scenarios, including launching global configuration fetching when a configuration anchor is uploaded, checking the status of global configuration fetching for diagnostics, and reading global configuration files on disk directly.
* It also communicates with the signer interface to manage token, key, and certificate information.
* It reads/writes data to the DB interface.

**Input/Output Ports**

* It has a listening port for incoming HTTPS traffic.
* It accesses the configuration client's admin port to command it to download global configuration.
* It accesses the signer's admin and signer protocol ports.
* It accesses the database using the port specified for DB properties.

**Persistent Data**

* It reads and writes various data from the filesystem, such as the configuration anchor, configuration backups, and internal TLS certificate.
* It reads and modifies the Security Server configuration using the database server configuration, with the database model specified in section 4.1.2.
* It also reads global configuration from the filesystem.
* Data related to token keys, certificates, and CSRs is accessed through the signer.

###### 3.1.2.1.2.2 Message Signer

The Message Signer component is tasked with the management of tokens, keys, and certificates.

**Encapsulated Data**

* It encapsulates a keystore where the Security Server's secret keys are securely stored.
* It also encapsulates configuration data, which tracks information related to tokens, keys, and certificates.

**Messaging**

* It fetches information from the certificate authority's OCSP (Online Certificate Status Protocol) responder.
* It offers an interface for signing requests, which is utilized by both the Proxy UI API and the Proxy component. Additionally, the Proxy component directly accesses the configuration encapsulated by Message Signer.
* It provides an admin interface for commanding Message Signer functionality, and this interface is used by the Proxy UI API and the Proxy component.

**Input/Output Ports**

* It has an output port for accessing the OCSP responder. The port number is configured in the Central Server, and Message Signer reads it from the global configuration.

**Persistent Data**

* It stores the configuration persistently in a configuration file.
* It also persists the secret keys in a keystore, which can be a file on the disk or stored inside a hardware security module (HSM) for enhanced security.

###### 3.1.2.1.2.3 Configuration

The Configuration Client is responsible for fetching the global configuration from a configuration source, which can be either the Central Server or Configuration Proxy.

**Encapsulated Data**

* It is responsible for downloading the global configuration and storing it locally on the disk. Other processes such as the Proxy, Proxy UI API, and Signer components access these files directly from the disk.

**Messaging**

* It downloads the global configuration from the configuration source, which can be the Central Server or Configuration Proxy, through interfaces.
* It offers an admin interface for commands and queries, which is utilized by the Proxy UI API to fetch diagnostic information.

**Persistent Data**

* It downloads and persists the global configuration on the local disk for later use.

###### 3.1.2.1.2.4 Proxy

The Proxy process is a critical component of the Security Server, primarily responsible for transmitting messages. Additionally, it handles message logging and timestamping, while message log archiving and cleaning are managed by the log process.

**Encapsulated Data**

Proxy configuration is stored in a database.

**Messaging**

* It accepts messages from the local trusted network through an interface.
* It also accepts messages from the untrusted network through another interface. Associated with this interface is another one that allows the downloading of OCSP (Online Certificate Status Protocol) responses from the Security Server.
* Furthermore, it offers an interface for admin commands and queries, which can be utilized by the Proxy UI API for diagnostics and can be used to set the Security Server in maintenance mode during a cluster rolling upgrade.

**Input/Output Ports**

* It has input ports for message exchange from both the internal and external networks, and one input port designated for uploading OCSP responses.
* Additionally, there is an input port for admin queries and commands.

**Persistent Data**

* It uses a database for the persistent storage of data, with the database model specified in section 4.1.2 of the Data Model Section.

###### 3.1.2.1.2.5 Database

The Database serves as the primary persistent data storage solution used by the Security Server.

* It stores databases, tables, and triggers, which are essential components for data management.
* It is utilized by the Proxy UI API, Proxy, and Procedure Monitor for persistent data storage, facilitating their operations.
* It typically has a single input port through which it receives queries and stores data.
* It is responsible for persisting databases, tables, and triggers based on the requirements and requests from its clients, ensuring the availability and integrity of stored data.

###### 3.1.2.1.2.6 Monitoring Infrastructure

The Monitor component is responsible for conducting environmental monitoring of the Security Server. It collects statistics related to various aspects, such as running processes, memory usage, and certificate statuses, using its sensors.

* The sensor data gathered by the monitor is stored in the memory of the monitor process.
* The sensor data from the monitor is queried by the Proxy using an interface.
* Additionally, the sensor data can be accessed via the JMX (Java Management Extensions) protocol through another interface.
* It does not persist the sensor data; it stores this data exclusively in the process memory, meaning it is not retained beyond the current session.

###### 3.1.2.1.2.7 Monitoring Protocol

The Procedure Monitor component is tasked with operational monitoring of the Security Server. This involves collecting statistics related to service calls, successes, failures, and other relevant operational data.

* It stores operational data in a database. Additionally, there is a buffer for this data in the Proxy process.
* It communicates with the Proxy through operational monitoring query and store interfaces, with detailed protocols described in section 3.2.
* Its data is also accessible via the JMX (Java Management Extensions) protocol through another interface.

This monitoring process plays a crucial role in tracking the performance and operational health of the Security Server.

###### 3.1.2.1.2.8 Log System

The Message Log is responsible for the crucial tasks of message log archiving and cleaning of the message logs from the database. However, it's important to note that message logging and timestamping are handled by the proxy process.

* It operates on shared data stored in the database.
* It plays a role in persisting data both to the disk and to the database. It handles the storage and maintenance of message logs, ensuring their availability and appropriate archival processes.

##### 3.1.2.1.3 Interfaces

###### 3.1.2.1.3.1 Management Service

The management services are utilized by security servers to execute various management tasks, including actions like registering a security server client or deleting an authentication certificate.

The management service interface is designed as a synchronous RPC-style interface, which is a mandatory requirement for the security server. These services are offered by central servers.

For a more detailed description of this interface section 3.2.2.5

###### 3.1.2.1.3.2 Download Configuration

The Security Server is responsible for retrieving the generated global configuration files from a configuration source.

The configuration download interface is designed as a synchronous interface, which is a necessary component for the proper functioning of the security server. This interface is made available by a configuration source, which can be a central server or a configuration proxy.

For a more comprehensive understanding of this interface, you can find detailed information in section 3.2.2.4.

###### 3.1.2.1.3.3 Message Protocol

The interface is described in more detail in 3.2.2.1 section.

###### 3.1.2.1.3.4 Transport Protocol

The interface is described in more detail in 3.2.2.2 section.

###### 3.1.2.1.3.5 Metadata Protocol

The Metadata Protocol serves as a means for service client information systems to collect information about the system instance. This protocol is particularly useful for identifying system members, the services they provide, and accessing WSDL service descriptions.

For a more thorough understanding of this protocol, you can refer to the detailed description provided in section 3.2.2.3 of the relevant documentation.

###### 3.1.2.1.3.6 Download Document

The service for downloading documents is a valuable resource for information systems, allowing them to retrieve signed containers from the security server's message log. These containers can then be transferred to third parties and verified offline, enhancing security and data integrity. Additionally, this service offers a convenient method for downloading the global configuration, which can be utilized to verify the signed containers effectively.

###### 3.1.2.1.3.7 Time Stamp Protocol

The Time-Stamp Protocol plays a vital role in the operations of security servers, ensuring the long-term proof of the exchanged messages. As part of this process, security servers log all messages along with their signatures. These logs are periodically time-stamped to establish enduring proof of their authenticity.

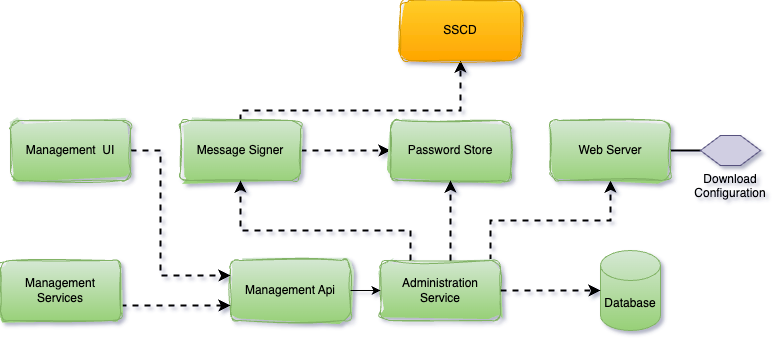
The Time-Stamp Protocol operates as a synchronous protocol and is provided by the timestamp authority. It serves as a crucial component in guaranteeing the long-term security and verifiability of the exchanged messages and their associated signatures.

###### 3.1.2.1.3.8 Online Certificate Status Protocol

The OCSP (Online Certificate Status Protocol) serves as a critical tool for security servers, enabling them to inquire about the validity status of signing and authentication certificates.

This protocol operates synchronously and is made available by the OCSP responder, which is associated with a certification authority. It plays a pivotal role in verifying the current validity status of certificates, ensuring the security and trustworthiness of certificate-based operations within the system.

#### 3.1.2.2 Central Server Architecture



##### 3.1.2.2.1 Components

###### 3.1.2.2.1.1 Administration Service

The Central Server Administration Service serves as a crucial tool for managing members and Security Servers, as well as defining global configuration parameters distributed to the Security Servers.

Notably, user actions that impact the system state or configuration are meticulously logged into the audit log. These actions are logged irrespective of whether the outcome was successful or resulted in a failure.

###### 3.1.2.2.1.2 Web Server

To make the global configuration files generated by the Central Server available to configuration clients, an HTTP-based protocol is used, as detailed in section 3.1.2.3. This protocol enables the distribution of a set of files, constituting the configuration, through a standard web server. The process of generating the global configuration is managed by the management services component, ensuring that clients can access the latest configuration data efficiently and reliably.

###### 3.1.2.2.1.3 Management Services

The collection of services used by Security Servers to execute management tasks, including actions like registering a Security Server client or deleting an authentication certificate, forms a crucial part of the system's administrative infrastructure.

These management services interact with the Administration Service REST API to initiate modifications to the configuration, ensuring that the Security Servers can efficiently and securely manage various aspects of their operations and configurations.

###### 3.1.2.2.1.4 Signer

The signer component plays a pivotal role in the system by overseeing the management of keys and certificates utilized for signing the global configuration. When the need arises to create a signature for the configuration, the management services component invokes the signer, which is responsible for generating the necessary signature, ensuring the security and integrity of the global configuration.

###### 3.1.2.2.1.5 Database

The Central Server serves as the repository for the system's configuration, storing it within a database. This configuration database encompasses critical components such as the security policy of the system instance, a list of members, Security Servers, global groups, and management services. For comprehensive insights into the specifics of the Central Server configuration, you can refer to the Database design section.

It's important to note that modifications to this configuration can be carried out through both the Central Server user interface and the management services, ensuring that the system's settings remain flexible and adaptable to changing requirements.

###### 3.1.2.2.1.6 Management UI

The Management UI is a user-friendly application designed to facilitate the management of members and Security Servers, as well as the definition of global configuration parameters distributed to the Security Servers.

This application serves as an interface through which users can interact with the system, enabling them to read and modify configuration settings. To achieve this, the Management UI communicates with the Administration Service REST API, ensuring seamless and secure management of the system's configuration and entities.

###### 3.1.2.2.1.7 Password Store

The system employs a secure mechanism for storing security token passwords. These passwords are stored within a shared memory segment of the operating system, which can be accessed by both the Central Server interface and the signer component.

This approach allows security token logins to remain persistent even when the Central Server is restarted, enhancing user convenience and system usability, all while maintaining the security and confidentiality of the stored passwords.

###### 3.1.2.2.1.8 Secure Signature Creation Device (SSCD)

The SSCD (Secure Signature Creation Device) is an optional hardware component designed to enhance the security of the system. Its primary function is to provide a secure and reliable capability for cryptographic signature creation to the signer component. This additional layer of security helps ensure the integrity and authenticity of cryptographic signatures, making it an essential option for organizations with stringent security requirements.

##### 3.1.2.2.2 Interfaces

###### 3.1.2.2.2.1 Management Services

The member management web service interface is a synchronous RPC-style interface made available by the member management web service component. This interface is responsible for validating incoming requests and subsequently forwarding them to the Central Server management REST API. Security Servers are the entities that make use of this service to perform various member management tasks, such as validating and updating member information within the system.

###### 3.1.2.2.2.2 Download Configuration

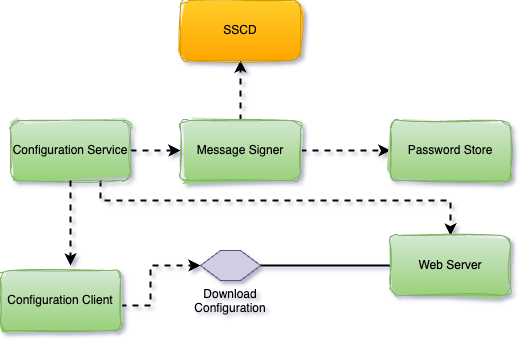
Configuration clients, including Security Servers and configuration proxies, retrieve the generated global configuration files from the Central Server.

This retrieval process is facilitated by the configuration download interface, which operates as a synchronous interface provided by the Central Server. It enables configuration clients to access and download the required global configuration files, ensuring that these clients are properly configured and up to date with the latest system settings and policies.

###### 3.1.2.2.2.3 Administration Web Service

The Administration Service REST API serves as an integral component within the system's architecture. Internally, it is utilized by the User Interface Frontend and Management Services components for various administrative tasks. Additionally, it is designed to be accessible to external components, providing a flexible and extensible interface that enables seamless integration with other systems or services as needed. This versatility allows for efficient management and configuration of the system, both internally and through external interactions. REST API use OpenAPI3 standards.

#### 3.1.2.3 Configuration Architecture



##### 3.1.2.3.1 Components

###### 3.1.2.3.1.1 Web Server

The global configuration files generated by the configuration proxy must be accessible to configuration clients. Technically, this configuration comprises a collection of files that are distributed using a standard web server. The configuration processor component plays a pivotal role in this process. It prepares and signs the configuration files, and then copies them to the web server's output directory. From there, they are made available to configuration clients through standard distribution methods. For more detailed information on how configuration distribution works, you can refer to Section 3.1.2.3.3. This process ensures that configuration clients can access and utilize the latest configuration data effectively.

###### 3.1.2.3.1.2 Configuration Service

The Configuration Service plays a crucial role in the system by overseeing the process of initiating the download of global configuration files for the configured system instance and preparing them for distribution to configuration clients.

To facilitate this, a scheduled job configured on the server is responsible for periodically triggering the Configuration Processor. This occurs at regular intervals, typically every minute, and the Configuration Processor then sequentially performs its functions for all configured configuration proxy instances. This systematic approach ensures that the latest configuration files are consistently prepared and distributed to configuration clients, maintaining the system's integrity and currency.

###### 3.1.2.3.1.3 Signer

The signer component plays a pivotal role in the system's security infrastructure by overseeing the management of keys and certificates used for signing the global configuration. When the need arises to create a signature for the configuration, the signer is called upon by the configuration processor to fulfill this important task. This ensures the authenticity and integrity of the global configuration, safeguarding it against unauthorized modifications and ensuring that it can be trusted by configuration clients.

###### 3.1.2.3.1.4 Configuration Client

The configuration client serves a vital role in the system by handling the task of downloading remote global configuration files. It obtains the source location of the global configuration from the anchor file, which is provided by the configuration processor when invoking the configuration client. This mechanism ensures that the configuration client knows where to retrieve the necessary global configuration files, enabling it to maintain an up-to-date and accurate configuration for the system instance.

###### 3.1.2.3.1.5 Password Store

The system employs a secure approach for storing security token passwords. These passwords are stored within a shared memory segment of the operating system, which can be accessed exclusively by the signer component. This design enables security token logins to persist, ensuring user convenience, while maintaining the security of the stored passwords. Importantly, this persistence remains effective until the configuration proxy is restarted, safeguarding the passwords against unauthorized access or compromise.

###### 3.1.2.3.1.5 SSCD

The SSCD (Secure Signature Creation Device) is an elective hardware component that delivers a secure cryptographic signature creation capability to the signer. Maybe HSM

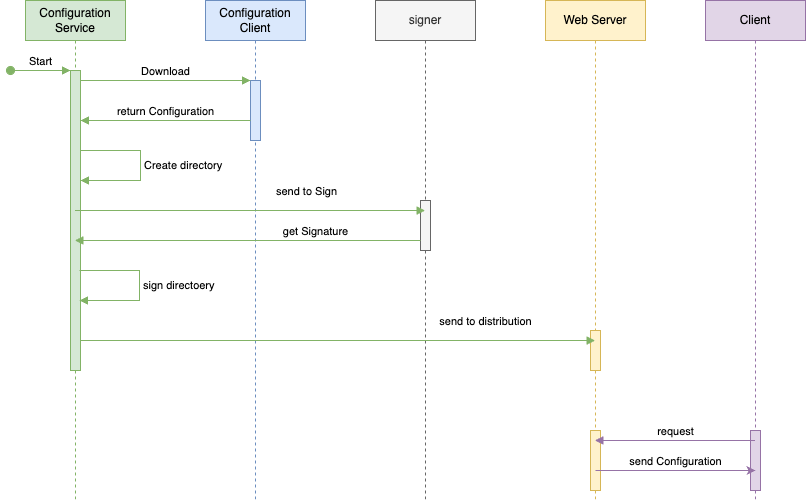
##### 3.1.2.3.2 Interfaces

###### 3.1.2.3.2.1 Download Configuration

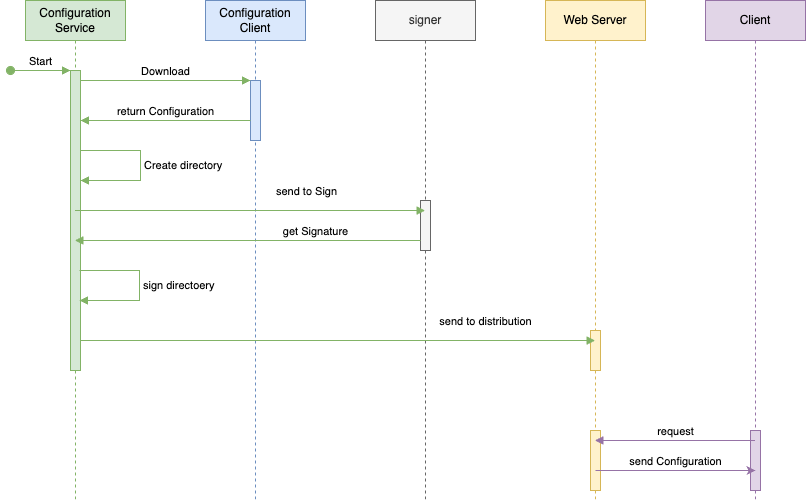
Configuration clients, including Security Servers and configuration proxies, retrieve the generated global configuration files from the Central Server.

This retrieval process is facilitated by the configuration download interface, which operates as a synchronous interface provided by the Central Server. It enables configuration clients to access and download the required global configuration files, ensuring that these clients are properly configured and up to date with the latest system settings and policies.

##### 3.1.2.3.3 Configuration Distribution

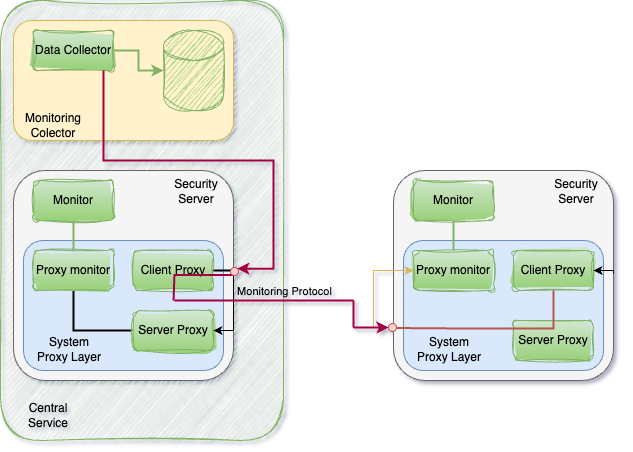


##### 3.1.2.3.4 Configuration Creation



#### 3.1.2.4 Monitoring Architecture

##### 3.1.2.4.1 Infrastructure Monitoring



###### 3.1.2.4.1.1 Components

3.1.2.4.1.1.1Meta Service (Proxy Monitor)

The Monitoring Metaservice plays a pivotal role in the system by responding to queries for monitoring data from the Security Server's Server Proxy interface. To fulfill these requests, the Metaservice communicates with the local monitoring service to obtain the most up-to-date monitoring data. Subsequently, the Monitoring Metaservice translates this monitoring data into a SOAP XML response, making it accessible and understandable for the requesting entity. This process ensures that real-time monitoring information is readily available in a standardized format, facilitating effective monitoring and analysis of the system's performance and health.

3.1.2.4.1.1.2 Monitoring Service

The Monitoring Service plays a crucial role within the system, responsible for collecting monitoring data from a specific Security Server instance. Once it has gathered this data, the Monitoring Service distributes it to monitoring clients. This process ensures that the performance and operational data of the Security Server are effectively collected and made available to clients for monitoring, analysis, and reporting purposes.

List of Sensor and listener: SystemMetricsSensor, DiskSpaceSensor, OsInfoLister, ProcessLister, PackageLister, CertificateInfoSensor.

3.1.2.4.1.1.3 Monitoring Client

The Central Monitoring Client is a specific Security Server instance that holds a special role within the system. It is granted the privilege to query monitoring data from other Security Servers, providing a centralized point for monitoring and data collection. The identity of this Security Server, acting as the Central Monitoring Client, is configured through the Central Server's admin user interface. This configuration allows the Central Monitoring Client to access and retrieve monitoring data from other Security Servers, facilitating comprehensive and centralized monitoring and analysis of the system's performance and health.

3.1.2.4.1.1.4 Data Collector

The Data Collector plays a crucial role in the system's monitoring infrastructure. Its primary responsibility is to gather monitoring data from all the Security Servers within the network. It accomplishes this by executing monitoring requests through the Central Monitoring Client, which communicates with all known Security Server instances.

Once the Data Collector has collected this monitoring data, it stores it in a permanent storage solution, where it can be securely and efficiently stored for future analysis and reporting. This data storage ensures that comprehensive monitoring information is readily available for assessing the performance and operational health of the entire system.

###### 3.1.2.4.1.2 Operations

3.1.2.4.1.2.1 Get Model

The Central Monitoring Data Collection operates using a "get" messaging model, which means that the Central Monitoring Client sends requests to individual Security Servers to obtain monitoring data. In this model, the client actively requests data from the servers when needed.

An alternative approach could involve a "push" model, where Security Servers periodically send monitoring data to the Central Monitoring Client without specific requests. This would involve the servers proactively sending their data at defined intervals.

To facilitate clustered configurations and ensure efficient communication, monitoring queries employ a message protocol. This protocol allows for effective data exchange and coordination between the Central Monitoring Client and the Security Servers, regardless of the chosen messaging model.

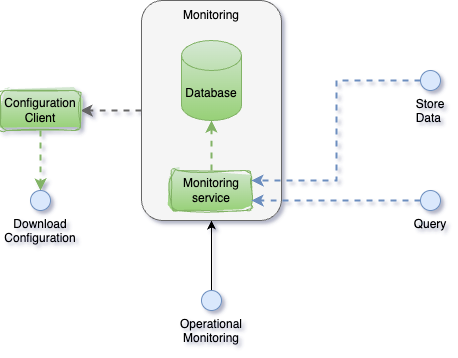
3.1.2.4.1.2.2 Authorization

The system has specific rules governing monitoring queries:

* Monitoring queries are allowed from the client that is the owner of the security server. This means that the Security Server itself, the entity that owns and operates it, can perform monitoring queries.
* Central Monitoring Client(s), if configured, are also permitted to execute monitoring queries. These Central Monitoring Clients are configured using the Central Server's admin user interface.

It's important to note that any attempts to query monitoring data from other clients that do not meet these criteria will result in an "access denied" response. This access control mechanism ensures that only authorized clients, including the owner of the Security Server and designated Central Monitoring Clients, can access and query monitoring data, maintaining data security and integrity.

##### 3.1.2.4.2 Protocol Monitoring



###### 3.1.2.4.2.1 Components

3.1.2.4.2.1.1 Monitoring Service

The Monitoring Service plays a crucial role in the system's monitoring capabilities. It receives and processes operational monitoring requests through the operational monitoring query interface. There are two types of requests used by the Security Server(s):

* Get Operational Monitoring Data: This request is used to retrieve operational monitoring data.
* Get Operational Health Data: This request is used to retrieve data related to the operational health of the system.

Here are some key aspects of how these requests are handled:

* When a regular client sends a "get monitoring data" request, only operational monitoring data records associated with that specific client are returned. In contrast, if the request sender is the Central Monitoring Client or the owner of the current Security Server, they have access to all records.
* To optimize performance and manage the size of responses, the Operational Monitoring Service limits the size of the "get operational monitoring data" response message. This maximum response size is configurable. If some queried records still do not fit into the response due to this size limit, the timestamp of the first excluded record is included in the response. This timestamp acts as an indicator of overflow, providing valuable information about potentially omitted data.

3.1.2.4.2.1.2 Configuration Client

The Configuration Client plays a critical role in the system by managing the download of remote global configuration files. To accomplish this task, the Configuration Client identifies the source location of the global configuration from an anchor file. This anchor file is manually copied to the configuration directory of the Operational Monitoring Daemon.

The Configuration Client operates as a standalone Java daemon application, ensuring that it can efficiently and independently perform its responsibilities related to configuration management and distribution.

###### 3.1.2.4.2.2 Interfaces

3.1.2.4.2.2.1 Store Monitoring Data

The protocol in question is utilized by the Security Server to store its cached monitoring data. This protocol operates as a synchronous RPC-style protocol based on JSON over HTTP(S). In cases where a secure connection is configured, the Security Server uses its internal self-signed TLS certificate, and the monitoring daemon utilizes its own internal self-signed TLS certificate. Importantly, both client-side and server-side certificate verification are carried out to ensure secure communication.

The availability of this service to the Security Server is not considered critical for the overall operation of the system. If this service becomes unavailable, the Security Server will continue to cache operational data records in its memory buffer. In the event of a buffer overflow, the oldest records are deleted as a means of managing memory usage.

It's worth noting that the storage of operational monitoring data is not time-critical. As a result, the Security Server performs asynchronous caching of the records on its side, providing flexibility in managing and storing this data efficiently.

3.1.2.4.2.2.2 Monitoring Query

The Monitoring Query Interface is employed by the Security Server to retrieve monitoring data. It operates using an asynchronous RPC-style monitoring protocol. When a secure connection (HTTPS) is configured, the Security Server utilizes its internal self-signed TLS certificate, while the monitoring daemon employs its own internal self-signed TLS certificate. Both client-side and server-side certificate verification processes are carried out to ensure the security of the communication.

It's important to note that monitoring of the Security Servers is not the primary functionality of the system. As a result, the availability and responsiveness of this service are not of paramount importance. Operational data records are stored in the database and remain available for a specified number of configured days, allowing for historical data analysis and reporting as needed.

## 3.2 Details

### 3.2.1 Specifications

#### 3.2.1.1 Message Protocol

##### 3.2.1.1.1 REST

<Time is not enough for this section>

+ System specific header parameters

+ Open API 3

##### 3.2.1.1.2 SOAP

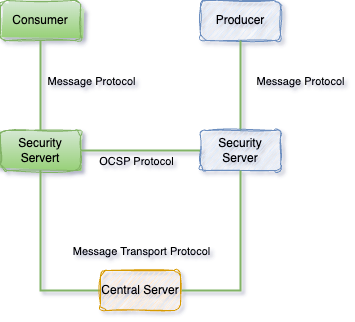
<Time is not enough for this section>

+ system specific parameters (align with REST)

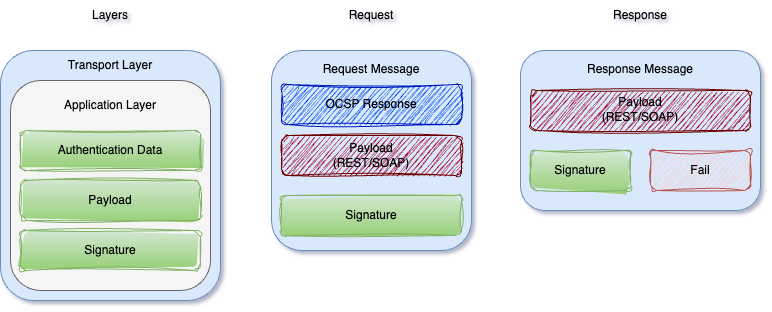
+ inner system differences for requests and responses

##### 3.2.1.1.3 Message Transport Protocol

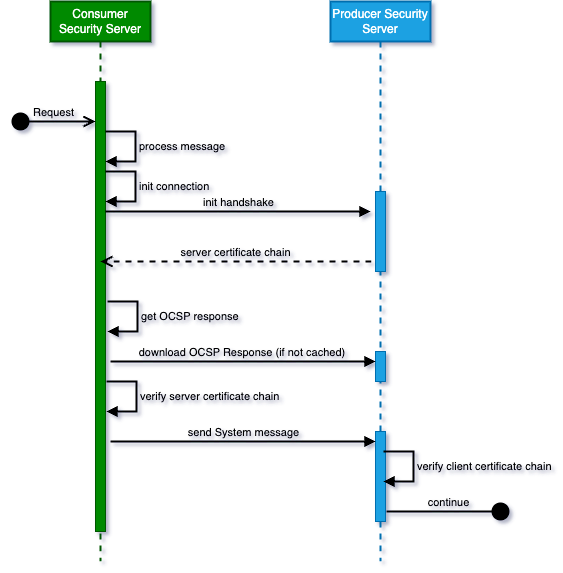
Generic flow



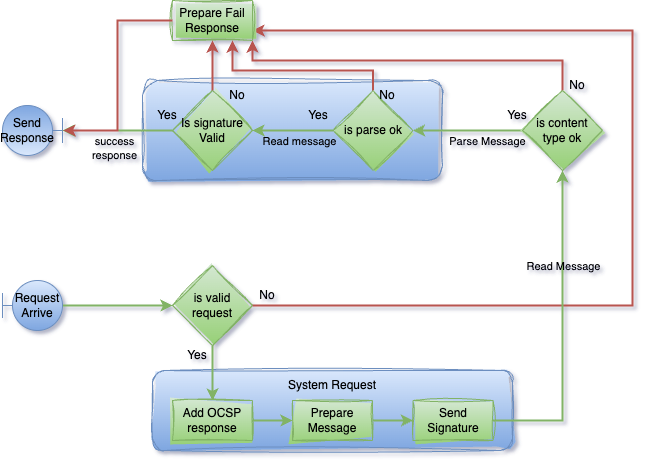
Structure of Messaging



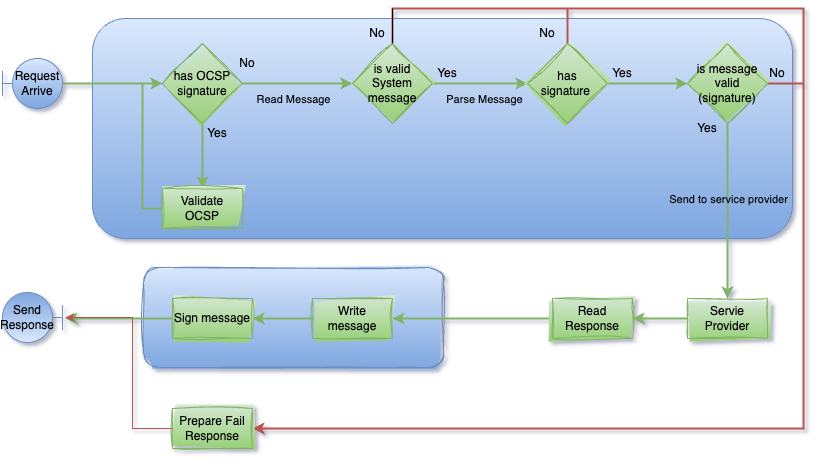
TLS Handshake



Client-Side flow



Server-Side flow



#### 3.2.1.2 Downloading Configuration Protocol

<Time is not enough for this section>

#### 3.2.1.3 Service Metadata Protocol

<Time is not enough for this section>

#### 3.2.1.4 Download Signed Document

<Time is not enough for this section>

#### 3.2.1.5 Management Services Protocol

<Time is not enough for this section>

#### 3.2.1.6 OCSP Protocol

<Time is not enough for this section>

#### 3.2.1.7 Time-Stamping Protocol

<Time is not enough for this section>

#### 3.2.1.8 Store Protocol Monitoring Data

<Time is not enough for this section>

#### 3.2.1.9 Protocol Monitoring Query

<Time is not enough for this section>

#### 3.2.1.10 Protocol Monitoring Protocol

<Time is not enough for this section>

#### 3.2.1.11 Protocol Monitoring JMX

<Time is not enough for this section>

#### 3.2.1.12 Infrastructure Monitoring Protocol

<Time is not enough for this section>

#### 3.2.1.13 Infrastructure Monitoring JMX

<Time is not enough for this section>

#### 3.2.1.14 Infrastructure Monitoring Query

<Time is not enough for this section>

#### 3.2.1.15 Security Server User Interface

<Time is not enough for this section>

#### 3.2.1.16 Central Server User Interface

<Time is not enough for this section>

#### 3.2.1.17 Audit Log

<Time is not enough for this section>

### 3.2.2 Authorization & Authentication & Consent

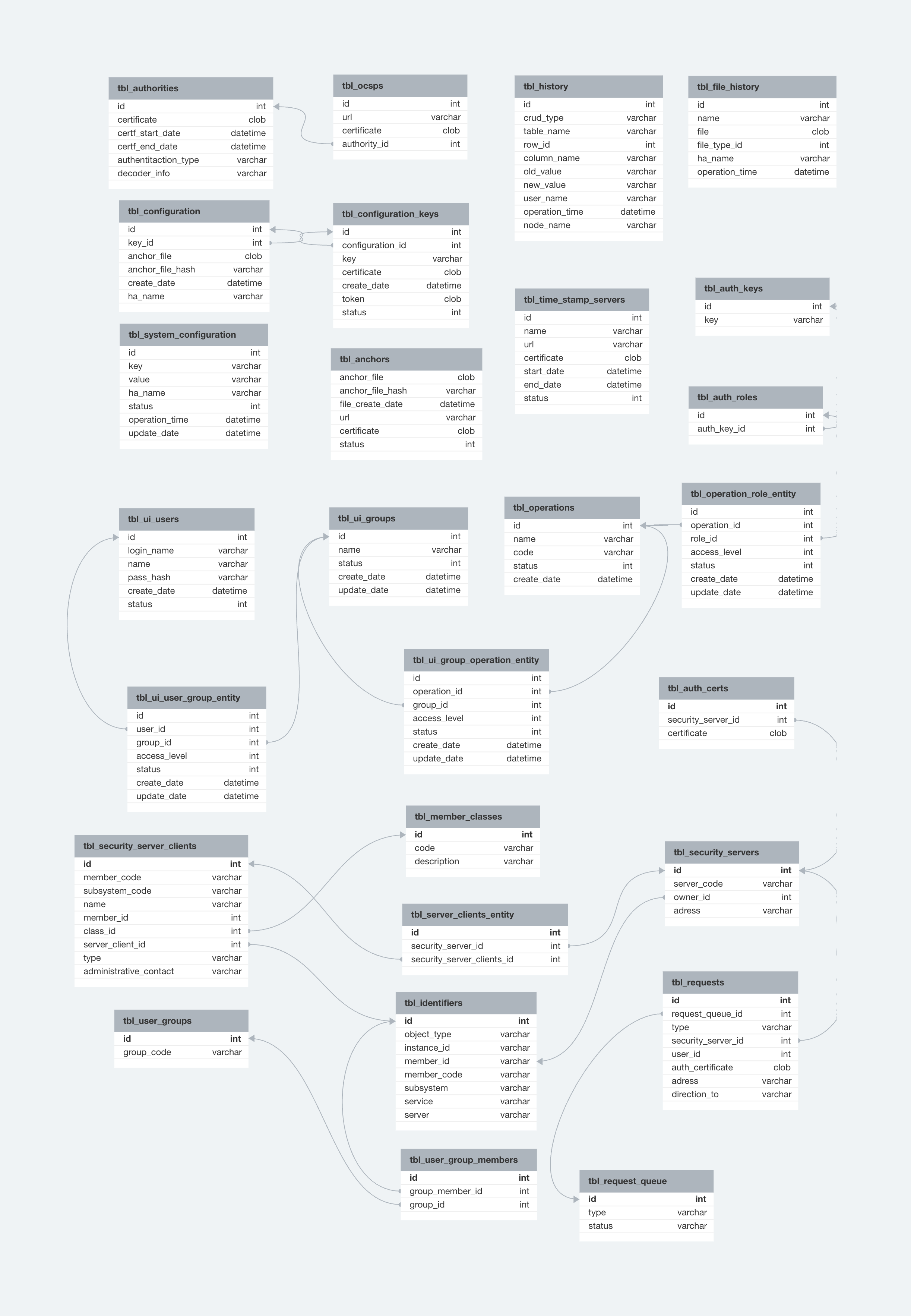
<Time is not enough for this section>

## 3.3 Use Cases

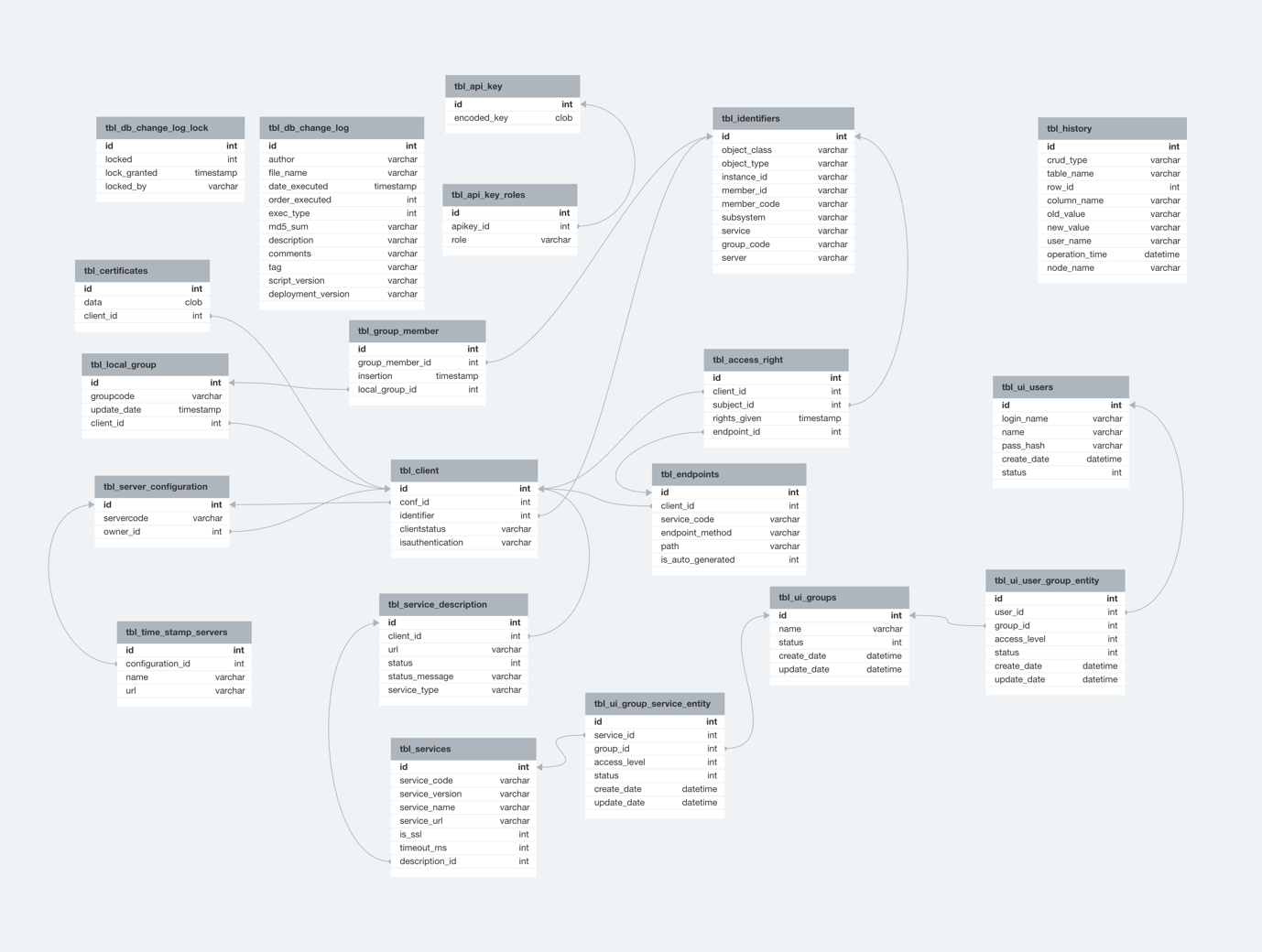
<Time is not enough for this section>

# 4. Database Design

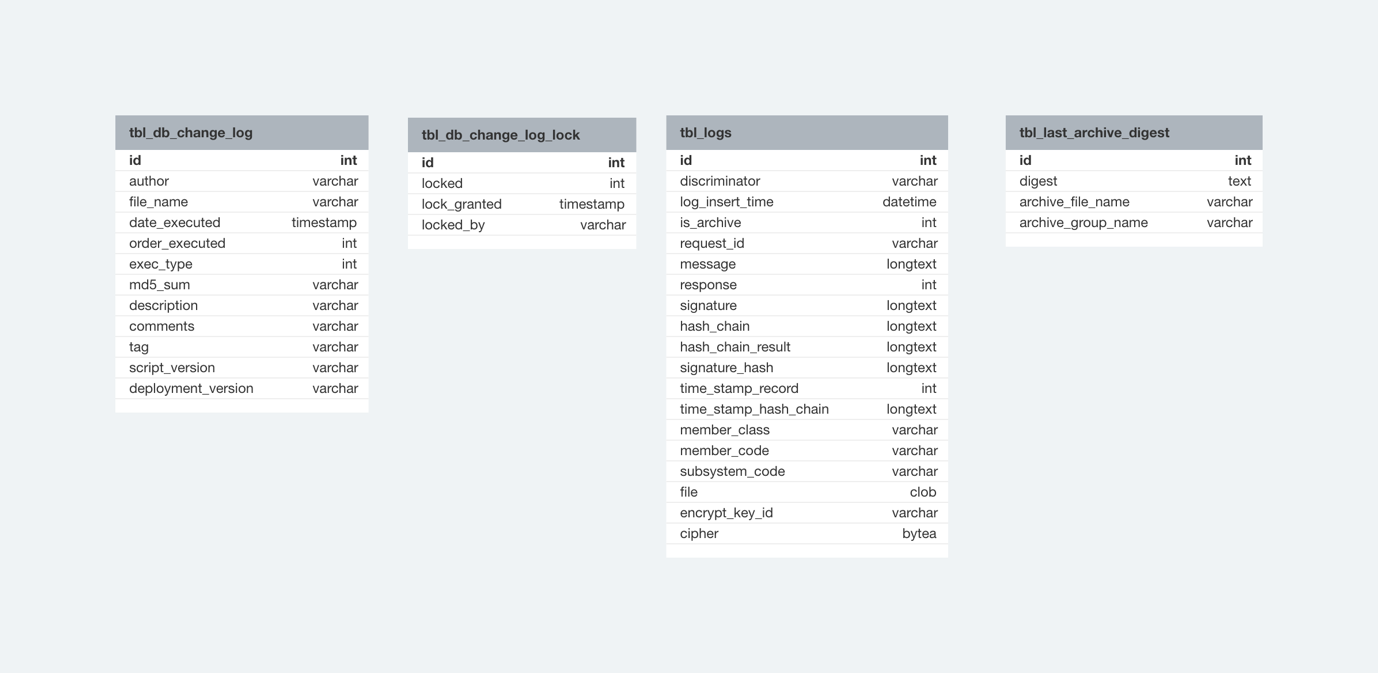
## 4.1.1 Central Server



## 4.1.2 Security Server



## 4.1.3 Log



# 5. User Interface Design

## 5.1 User Interface Mockups

<Time is not enough for this section>

## 5.2 User Interaction Flow

<Time is not enough for this section>

## 5.3 Screen Descriptions

<Time is not enough for this section>

# 6. Software Development Environment

## 6.1 Programming Languages

Java, C, typescript

## 6.2 Development Tools

IntelliJ, Visual Studio Code, Vim

## 6.3 Version Control

Git

## 6.4 Build and Deployment Process

Spring boot, Npm, Node, Kubernates, Jenkins, Liquabase, Lombok, Tomcat,

## 6.5 Open Standards

OpenAPI 3, OpenID, JWT, RSA, PKCS

## 6.6 Data

Transactional data Optional (MySQL or PostgreSQL)

Document Data – Mongo

Event Streaming (for Queue RabbitMQ, for notification Kafka)

Cache - Redis

# 7. Quality Assurance and Testing

## 7.1 Test Strategy

### 7.1.1 Test Types

* Unit Tests
* Load Tests
* Automated Tests
* Integration Tests
* Smoke Tests

### 7.1.2 Test Runs

* Every pipeline works (for Unit Tests)
* Every release Beta (Integration, Smoke, Automation)
* Every release (Automation, Load)

## 7.2 Test Cases

<Time is not enough for this section>

## 7.3 Testing Tools

JMeter, Selenium, Java Unit Test Automation

## 7.4 Test Results

<Time is not enough for this section>

# 8. Deployment and Maintenance

## 8.1 Deployment Plan

To successfully deploy the platform to the live environment, follow the steps below:

### 8.1.1 Software Deployment

* Prepare and thoroughly test the latest software version, ensuring it includes security updates and recent fixes.
* Verify that the live environment is ready. Ensure servers, network infrastructure, and other resources are operational and properly configured.
* Install the software on live servers, ensuring a backup and secure deployment process.
* Establish contingency plans to swiftly address any errors or issues that may arise during deployment.

### 8.1.2 Security Settings and Controls

* Configure data access and authorization, allowing only authorized users to access data.
* Activate security firewalls and enforce strict controls on incoming and outgoing traffic.
* Implement user authentication, encouraging strong password policies and multi-factor authentication.

### 8.1.3 Monitoring and Event Tracking

* Enable monitoring tools to track security events and system performance.
* Retain and archive log records to facilitate analysis and review of incidents.

### 8.1.4 End-User Testing

* Conduct final user testing before the platform goes live to identify functional glitches and performance issues.
* Take user feedback into consideration and promptly resolve issues.

### 8.1.5 Planned Downtime and Improvements

* Minimize planned downtime during the platform's deployment to live. Inform users in advance about any interruptions.
* Monitor performance after deployment to live and promptly address any issues.
* Evaluate feedback for continuous improvement and plan necessary updates.

These steps are crucial to ensure the secure and stable deployment of platform to the live environment.

## 8.2 Maintenance Procedures

### 8.2.1 Bug Solving

We will be implementing the existing SDLC. Modules that require version distribution, such as the security server, should be informed separately to relevant parties. Backward compatibility should always be considered. Testing for organizations that have not received updates should also be conducted separately.

### 8.2.2 Change Management

We will be implementing the existing SDLC. Modules that require version distribution, such as the security server, should be informed separately to relevant parties. Backward compatibility should always be considered. Testing for organizations that have not received updates should also be conducted separately.

### 8.2.3 New Function

We will be implementing the existing SDLC. Modules that require version distribution, such as the security server, should be informed separately to relevant parties. Backward compatibility should always be considered. Testing for organizations that have not received updates should also be conducted separately.

# 9. Appendices

## 9.1 Acronyms and Abbreviations

CA - Certification Authority

HSM – Hardware security module

OCSP – Online Certificate Status Protocol

SSH - Secure Shell

TLS - Transport Layer Security

TSA - Timestamping Authority

TSP - Time Stamp Provider

API - Application Programming Interface

CI - Continuous Integration

DSL - Domain Specific Language

GPG / GnuPG - The GNU Privacy Guard

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

JMX - The Java Management Extensions

JMXMP - Java Management Extensions Messaging Protocol

JSON - JavaScript Object Notation

MIME - Multipurpose Internet Mail Extensions

RPC – Remote Procedure Call

REST - Representational State Transfer

SDK - Software Development Kit

SOAP - Simple Object Access Protocol

## 9.2 References

<https://docs.airbyte.com/>

<https://github.com/GSA/data.gov>

<https://github.com/nordic-institute>

<https://x-road.global>

<https://www.nginx.com/>

<https://www.globalsign.com/en/resources/white-papers-ebooks/digital-signatures-made-easy-ebook-ppc>

<https://tools.ietf.org/html/rfc2616.html>

<https://tools.ietf.org/html/rfc6265>

<https://www.owasp.org/index.php/Server_Side_Request_Forgery>

<https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md>

<https://www.openapis.org/>

<https://tools.ietf.org/html/rfc3986>

<https://en.wikipedia.org/wiki/Representational_state_transfer>