Requirements Specification: Federated Trust Node

Document:	Requirements specification: Federated Trust Node
Version:	0.2.1/2024

FEDERATED TRUST NODE

REQUIREMENTS SPECIFICATION

1 BACKGROUND

The purpose of a Federated Trust Node is to provide mechanisms to create and publish verifiable information about members of a trust ecosystem as input to parties for making trust decisions.

Findynet Cooperative is building infrastructure for enhanced trust and verifiable data in Finland. We believe that many ecosystems will benefit from a federated trust model. One of Findynet's technical services will be a trust framework based on OpenID Federation specification (OIDF), see [1][1].

Our trust framework consists of independent nodes operated by different roles (Leaf Entities, Intermediate Entities, and Trust Anchors, as specified in OIDF. The organizations acting in the aforementioned roles need software to publish Entity Statements. We refer to this software as Federated Trust Node.

This document specifies the requrements for the implementation of a Federated Trust Node. Our intention is to release the Federated Trust Node implementation acquired by Findynet as an open-source software that other organizations can install and operate in their environments.

Findynet will also offer the Federated Trust Node functionality as a hosted service. When multiple verifiable data ecosystems are using Findynet's Federated Trust Node service, Findynet may operate multiple Federated Trust Nodes. Thus, the Federated Trust Node should support multi-tenant deployment where it is easy to upgrade multiple instances of the software to a newer version while isolating the data of each instance from other instances.

The OpenID Federation specification is still work in progress. The implementation of the Federated Trust Node should take the latest published draft as the baseline for the implementation and have flexible change management practices in place to follow the evolving specification.

2 ARCHITECTURE

This chapter describes the initial architectural design of the Federated Trust Node.

2.1 General

The Federated Trust Node shall be designed so that it can operate as a Trust Anchor, as an Intermediate Entity, or as a Leaf Entity. It shall support both single and multiple Trust Anchor models.

The Federated Trust Node can be deployed on cloud or on-premises. The design shall not make any assumptions on the operating environment and shall not e.g. rely on cloud provider specific technologies.

2.2 Interfaces and components

Diagram 1 is an initial design of the interfaces the Federated Trust Node exposes (on the right), components used to publish those interfaces (in the middle) and data stores (on the left).

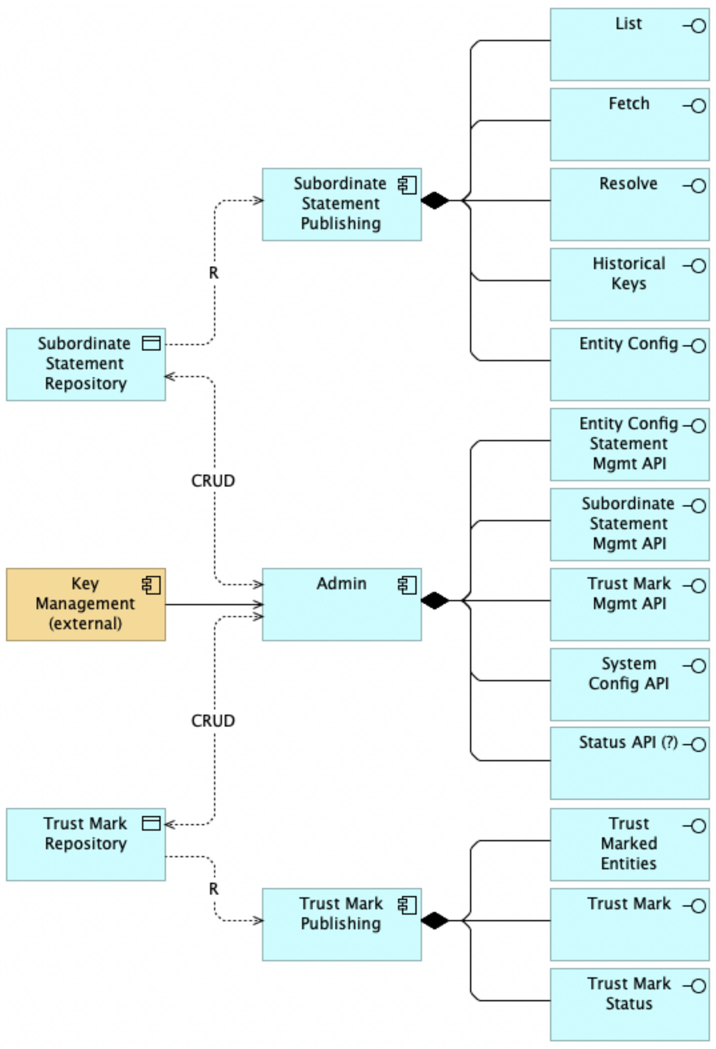

Diagram 1: Initial interface and component design (ArchiMate)

The interfaces (endpoints) offered by the **Subordinate Statement Publishing** component and the **Trust Mark Publishing** component are public and implement the functionality specified in OIDF. The operator of the Federated Trust Node may implement client authentication as specified in the chapter 8.8 of OIDF.

The interfaces (endpoints) offered by the **Admin** component shall be protected in the way described in the section 2.3 in this document.

2.3 Access Control

The Federated Trust Node shall support access control mechanisms to restrict access to all administrative functionality and ensure that only authorized users have permissions to perform aforementioned tasks. Access control shall be configurable to support fine-grained access control if required in the future.

Details of the initial access control mechanism will be designed together with the vendor implementing the Federated Trust Node.

2.4 Key Management and cryptographic operations

Federated Trust Node shall support JWT signing operations while providing flexibility to choose between built-in key management and signing functionality or integration with external

Hardware Security Modules (HSMs), Key Management Services (KMSs), or other trusted environments. Trust Node operator shall have the option to configure the software to perform cryptographic operations in the chosen environment based on federations security requirements and operational preferences.

2.5 REST APIs

Each component of the Federated Trust Node (Entity Statement Publishing, Admin, Trust Mark Publishing) shall have a RESTful API exposing its functionality. The APIs shall be documented using OpenAPI format.

All endpoints creating, updating, and deleting data shall have a dry-run variant which performs the operations and show the results without actually modifying any data.

2.6 Graphical User Interface

In addition to the REST APIs, the Federated Trust Node may offer a graphical, web browser based user interface for admin purposes. Details of the graphical user interface will be designed together with the vendor implementing the Federated Trust Node.

2.7 Logs

By default, the components of the Federated Trust Node shall output all audit log and system log records to their default output stream (`stdout`) and all errors log entries to the error stream (`stderr`). The Federated Trust Node may have a configurable logging component which allows the operator of the Federated Trust Node to direct the log entries to for example text files or to a database.

The logging level of the Federated Trust Node shall be configurable.

All log entries shall include at least

- the timestamp of the error
- error level (TRACE, DEBUG, INFO, NOTICE, WARN, ERROR, FATAL)
- the error code or type
- the error message (meaningful explanation of what happened)
- the name of the component logging the error
- the operation performed when the error occurred
- the source code line number.

2.7.1 Audit logs

The Federated Trust Node shall produce audit logs. All Statement lifecycle operations, Trust Mark creations, and all admin operations shall be recorded to audit log.

2.7.2 System logs

The Federated Trust Node shall produce system logs that can be used for monitoring system behavior, performance and stability.

2.7.3 Error logs

The Federated Trust Node shall produce a log entry for all error situations.

3 FUNCTIONAL REQUIREMENTS

This chapter describes the functional features of a Federated Trust Node. The requirements in this chapter are derived from the draft 33 of the OIDF specification [1][1].

3.1 Entity Configuration

3.1.1 Create

The Federated Trust Node shall have functionality for creating Entity Configuration statements. Entity Configuration generation shall support all mandatory parameters as defined in OIDF and the created statement shall be verified to be compliant with OIDF.

3.1.2 Publish

The Entity Configuration shall be made available at */.well-known/openid-federation* endpoint as defined in OIDF.

3.1.3 Update *or renew*

Federated Trust Node shall have functionality to update or renew its Entity Configuration. Entity Configuration shall be renewed before its expiration. All expired or otherwise invalidated Entity Configurations shall be kept in the repository.

3.2 Subordinate Statement Repository

Subordinate Statement Repository is the core function of the Federated Trust Node software. It provides functionality for creating and publishing Subordinate Statements and for managing their lifecycle.

The repository shall have the following publicly available endpoints ad defined in OIDF:

- federation_fetch_endpoint
- federation_list_endpoint
- federation_resolve_endpoint
- federation_historical_keys_endpoint

The functionality of the endpoints mentioned above shall comply with the OIDF specification.

3.2.1 Create and publish Subordinate Statements

The Federated Trust Node shall have functionality for creating Subordinate Statements.

When creating Subordinate Statements the Federated Trust Node shall verify all input parameters, and validate that the resulting Trust Chain is valid and terminates to one or more Root of Trust.

Created Subordinate Statements shall be made available for download from *federation_fetch_endpoint*.

3.2.2 Renew Subordinate Statements

The Federated Trust Node software shall automatically renew and publish Subordinate Statements before their expiration. When renewing the Subordinate Statements the system shall perform the same verifications as in when creating new Subordinate Statement. If the verification fails the system shall not renew the statement.

The Federated Trust Node shall have functionality for renewing one or more Subordinate Statements at any point in time. This functionality shall be available only for admin users.

Expired or replaced Subordinate Statements shall not be deleted from repository.

Subordinate statement renewal shall not have impact on Federated Trust Node availability.

3.2.3 Subordinate Statement deletion

The Federated Trust Nodeshall have functionality for deleting Subordinate Statements it has issued.

Once Subordinate Statement is deleted it shall not anymore be downloadable from federation_fetch_endpoint. However all deleted Subordinate Statements shall be kept in repository.

3.2.4 Subordinate Statement Revocation

Current version of OIDF does not specify Subordinate Statement revocation mechanism apart from deleting and unpublishing the Statement. Revocation mechanisms, such as Token Status List (see [1][2]) might be specified in the future and the Federated Trust Node shall allow such mechanisms to be used.

3.2.5 List subordinate statements

The Federated Trust Node shall have functionality to list active Subordinate Statements.

Active Subordinate Statements shall be available from *federation_list_endpoint*.

3.3 Trust Mark Publishing

The Federated Trust Node shall have functionality for creating and publishing Trust Marks as specified in OIDF specification. The created Trust Marks shall comply with the OIDF specification.

The Federated Trust Node shall have the following publicly available endpoints:

- federation_trust_mark_status_endpoint
- federation_trust_mark_list_endpoint
- federation_trust_mark_endpoint

The functionality of the endpoints mentioned above shall comply with the OIDF specification.

3.4 Federated Trust Node Admin API

All Subordinate Statement lifecycle operations, Trust Mark publishing and admin operations shall be available via REST API. The table below has example endpoints the Federated Trust Node should have.

Endpoint	Method	Purpose	Protected
/create	POST	Create new Subordinate Statement, should have also dry-run functionality	YES
/delete	DELETE	Delete existing subordinate statement, move to historical data	YES
/renew	POST	Renew existing statement(s)	YES
/trustmark	POST	Create or update a Trust Mark	YES
/config	GET/POST	Generic endpoint for viewing and modifying system configuration	YES
/status	GET	Status endpoint for monitoring system availability and functionality	NO
/stats	GET	System statistics endpoint	YES

Table 2: Example endpoints

The API shall be documented using OpenAPI format.

4 NON-FUNCTIONAL REQUIREMENTS

4.1 Availability

The Federated Trust Node shall be designed for high availability. Operations such as statement batch renewal should have only minimal impact on system availability and performance.

4.2 Performance

In a federated trust model, the check of the trustworthiness of an entity may require multiple queries to multiple Federated Trust Node.

The Federated Trust Node doesn't need to have an internal cache but it shall support the caching-related HTTP header values (e.g., `Cache-Control`, `Expires`, `ETag`, `If-Modified-Since`, `If-None-Match`) for each publishing endpoint and it shall have the possibility to configure the expiration time for each endpoint.

4.3 Scalability

The Federated Trust Node shall be designed to scale horizontally to accomodate an increasing number of Subordinate Statements and requests over time. For example, the Federated Trust Node Operator should be able to run multiple parallel instances of the Subordinate Statement Publishing component (see Diagram 1) using a load balancer in front of them.

4.4 Security

The Federated Trust Node shall have appropriate access controls to ensure that only authorized users can create, update, or delete Entity Configurations, Subordinate Statements and Trust Marks.

4.5 Configurability

The Federated Trust Node shall have mechanisms to configure its operational parameters, such as default statement lifetime. The Federated Trust Node shall read its operational parameters both from environment variables and a configuration file, the former having a higher priority.

Configuraton Statement creation shall be template based and should allow adding or removing mandatory or optional claim parameters.

5 REFERENCES

[1] OpenID Federation 1.0 - draft 33, openid.net/specs/openid-federation-1_0.html

[2] Token Status List – draft 02, ietf.org/id/draft-ietf-oauth-status-list-02.html