Software Requirements Specification

for

Gaia-X Federation Services

Trust Services API
IDM.TSA

Published by

eco – Association of the Internet Industry (eco – Verband der Internetwirtschaft e.V.)
Lichtstrasse 43h
50825 Cologne, Germany

Copyright

Table of Contents

List of Figures

List of Tables

1. Introduction

To get general information regarding Gaia-X and the Gaia-X Federation Services please refer to [TAD] and [PRD].

1.1. Document Purpose

The purpose of the document is to specify the requirements of the Identity Management and Trust Subcomponent "Trust Services API" with the intention of a European wide public tender for implementing this software. Main audience for this document is attendees of the public tender, which are able to supply an open-source software solution for the area of Signing/Validation, Secure Policy Management and Policy Execution with the purpose to provide trusted services around Decision Evaluations and Verifications.

1.2. Product Scope

The product scope covers the functionalities of the Trust Services API. The aim of the Trust Services API is to ensure a consistent level of trust between Gaia-X participants and components. The Trust Services API can be used by all components. The creation and validation of digital signatures plays a particularly important role here. The product scope includes signing and verifying of necessary data, enabling policy driven trust, ensuring trust-chains between participants and validating eIDAS compliant signatures.

The scope also includes necessary tools (e.g., Command Line Scripts) to operate and maintain the created software components in an enterprise environment with focus on high-availability, security and monitoring and logging based on common standards. Documentation for developer, operator and user MUST be written in markdown format which is public consumable over a publicly accessible source repository without access limitations.

1.3. Definitions, Acronyms and Abbreviations

The IDM and Trust Architecture Overview Document [IDM.AO] MUST be considered and applied as the core technical concept that includes also the Terminology and Glossary.

All requirements from other documents are referenced by [IDM.<document-id>.XXXXX] as defined in the chapter "Methodology" in the document [IDM.AO].

1.4. References

[Aries.RFC0047]	**(Community) (2020), Aries RFC 0047: JSON-LD Compatibility**
	https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0047-json-ld-compatibility/README.md (Status 03-17-2021)

[AMD.EPY]	AMD (2018), White Paper, AMD EPYC™ Hardware Memory Encryption
	https://www.amd.com/system/files/documents/cloud-security-epyc-hardware-memory-encryption.pdf (Status 03-18-2021)
[AMD.SEV-SNP]	AMD (2020), AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More
	https://www.amd.com/system/files/documents/cloud-security-epyc-hardware-memory-encryption.pdf (Status 03-18-2021)
[BDD]	Specflow (n.D.), Getting Started with Behavior Driven Development
	https://specflow.org/bdd/ (Status 03-18-2021)
[CryptoLen]	Damien Giry, Prof. Jean-Jacques Quisquater (2020), Cryptographic Key Length Recommendation
	https://www.keylength.com/en (Status 03-18-2021)
[CloudEvents]	**CloudEvents Authors, The Linux Foundation (2021), CloudEvents Specification**
	https://cloudevents.io/ (Status: 03-27-2021)
[EUCS]	**European Union Agency for Cybersecurity (ENISA) (2020), EUCS – Cloud Services Scheme**
	https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme (Status: 03-29-2021)
[Git.Tools]	**Scott Chacon and Ben Straub (2014), Pro Git (2nd Edition), 7.4 Git Tools - Signing Your Work**
	https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work (Status 03-04-2021)
[ISO25000]	**ISO 25000 Portal (n.d.), ISO/IEC 25010**
	https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (Status: 03-17-2021)
[IDM.AO]	**Gaia-X WP1[1] (2021), Architecture Overview**
	IDM.AO (Base of functional specification)
[Intel.SGX]	Intel (n.D.), Intel Software Guard Extension (SGX)
	https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/software-guard-extensions.html (Status 03-18-2021)
[JSON.LD]	**W3C Draft (2021), JSON-LD 1.1**
	https://w3c.github.io/json-ld-syntax/ (Status 03-04-2021)
[LD.Proofs]	**W3C Draft (2020), Linked Data Proofs 1.0**

[1] Please refer to appendix C for an overview and explanation of the Work Packages (WP).

	https://w3c-ccg.github.io/ld-proofs/ (Status 03-04-2021)
[NF.SPBD]	**Gaia-X Federation Service Non-functional Requirements Security & Privacy by Design**
	Please refer to annex "GXFS_Nonfunctional_Requirements_SPBD"
[PRD]	**Gaia-X, European Association for Data and Cloud, AISBL (2021): Gaia-X Policy Rules Document**
	Please refer to annex "Gaia-X_Policy Rules_Document_2104"
[RFC3161]	**C.Adams, P.Cain, D.Pinkas, R.Zuccherato (2001), Internet X.509 Public Key Infrastructure, Time-Stamp Protocol (TSP)**
	https://www.ietf.org/rfc/rfc3161.txt (Status: 03-17-2021)
[RFC6902]	**P.Bryan, M.Nottingham (2013), RFC6902 - JavaScript Object Notation (JSON) Patch**
	https://tools.ietf.org/html/rfc6902 (Status: 03-13-2021)
[Rego]	**(Unknown) (n.D.), Rego Policy Language**
	https://www.openpolicyagent.org/docs/v0.12.2/language-reference/ (Status: 03-16-2021)
[Sem.Vers]	**Tom Preston-Werner (n.D.), Semantic Versioning 2.0.0**
	semver.org (Status: 03-17-2021)
[SecVoc]	**W3C (2021), The Security Vocabulary**
	https://w3id.org/security (Status 03-17-2021)
[SOG-IS]	SOG-IS Crypto Working Group (2020), SOG-IS Crypto Evaluation Scheme - Agreed Cryptographic Mechanisms
	https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.2.pdf (Status 03-18-2021)
[TDR]	**Gaia-X Federation Services Technical Development Requirements**
	Please refer to annex "GXFS_Technical_Development_Requirements"
[TAD]	**Gaia-X, European Association for Data and Cloud, AISBL (2021): Gaia-X Architecture Document**
	Please refer to annex "Gaia-X_Architecture_Document_2103"
[TR02102-1]	**BSI (2020), Cryptographic Mechanisms: Recommendations and Key Lengths BSI TR-02102-1**
	https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=2 (Status 03-18-2021)
[TR02102-2]	**BSI (2020), Cryptographic Mechanisms: Recommendations and Key Lengths: Use of**

	Transport Layer Security (TLS) BSI TR-02102-2,
	https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=2 (Status 03-18-2021)
[Uni.Resolver]	**DIF (2020), Universal Resolver**
	https://github.com/decentralized-identity/universal-resolver#readme (Status 03-04-2021)
[URSA]	**Linux Foundation Project (2021), Hyperledger Ursa**
	https://www.hyperledger.org/use/ursa (Status: 03-15-2021)

***Table 1**: References*

1.5. Document Overview

The document describes the product perspective, functions and constraints. It furthermore lists the functional and non-functional requirements and defines the system features in detail. The listed requirements are binding. Requirements as an expression of normative specifications are identified by a unique ID in square brackets (e.g. **[IDM.ID.Number]**) and the keywords MUST, MUST NOT, SHOULD, SHOULD NOT, MAY, corresponding to RFC 2119 [RFC 2119], are written in capital letters (see also [IDM.AO] - Methodology).

2. Product Overview

2.1. Product Perspective

The origin of the product is the requirement to support other components with “Trusted Services” in the direction of policy evaluation, policy driven trust, trust anchor administration, DID services and signature and validation services. To reach this goal, the product MUST provide HTTP Endpoints which are consumable by components within the trusted security domain or provide the functions as a set of library-based components. Additionally, the product shall contain functionality to enhance the trusted services by policy built-in functions and be able to collect sufficient and validated data, e.g., for iterating evaluations, as well as signing and validation of trust chains and trust sets. All policies can be partly predefined and partly determined by the participants themselves. Policies SHOULD be manageable via GitOps principles and connections to a secure storage MUST be ensured.

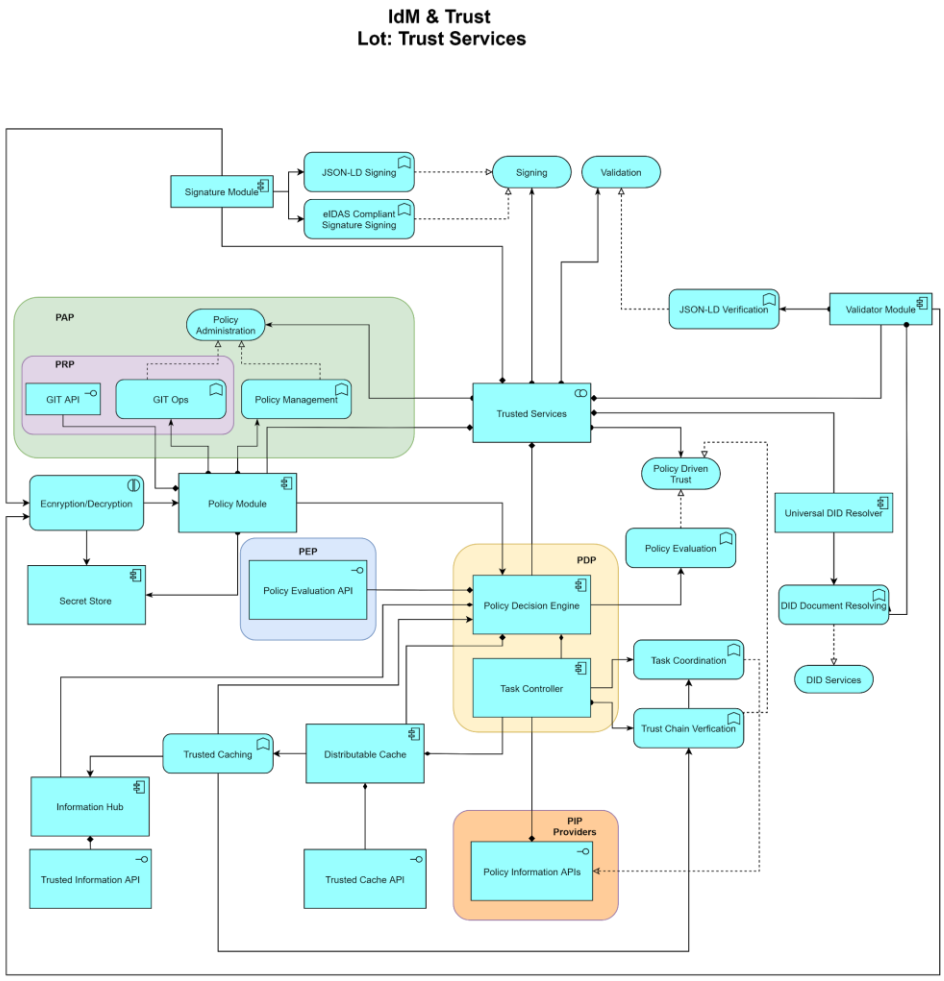


***Figure 1**: Functional Architecture*

2.2. Product Functions

The functions of the Trust Services component are provided case-dependent either as runtime or library components. Runtime components MUST expose endpoints as REST services. In addition, policy configuration with GitOps should be used to enable the provisioning and sharing of policies. The component is part of the Gaia-X Trust and identity management toolstack and not centrally hosted. To properly maintain and update, appropriate security measures MUST be in place. This includes role concepts, data storage protection and access control. The overall functionality of the product MUST be auditable (GDPR compliant).

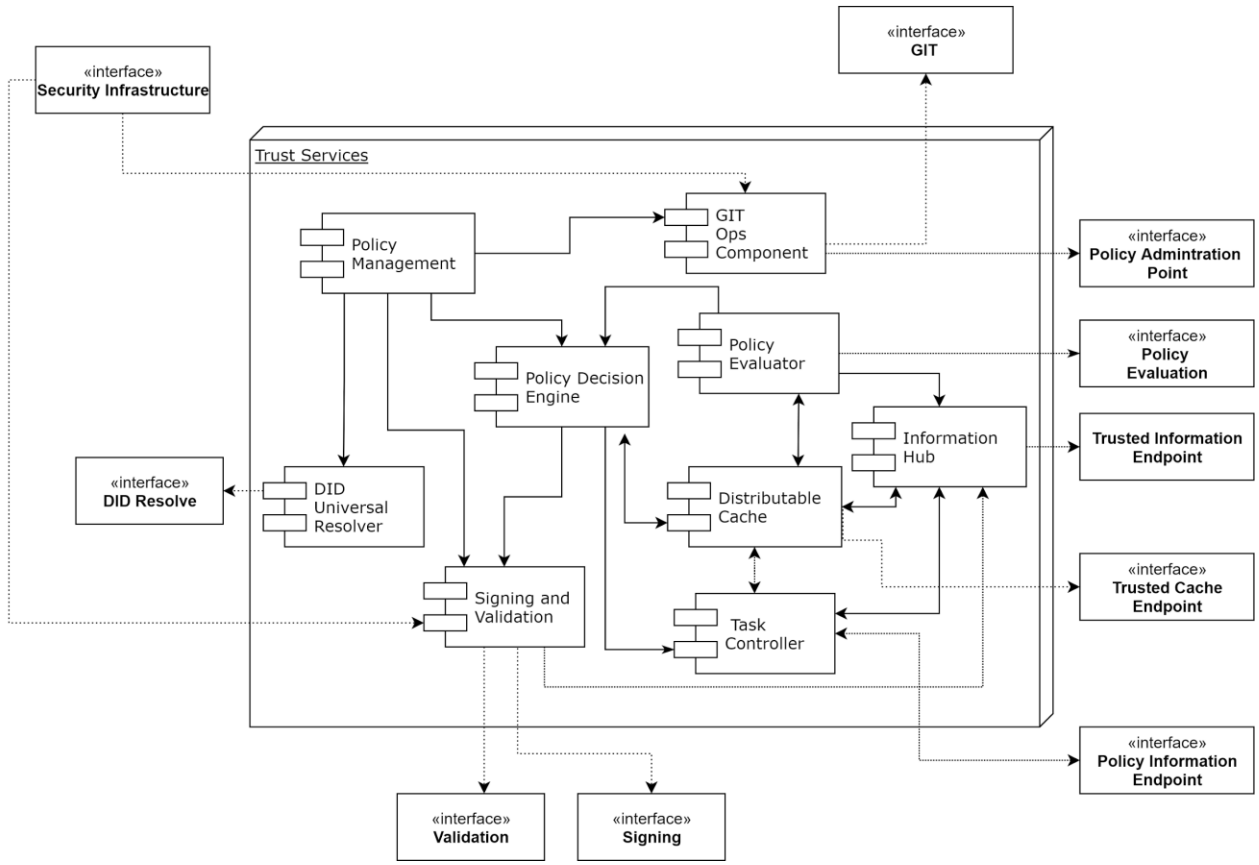


***Figure 2**: Sketch of the expected Software Architecture*

The core functions of the Trust Service API are:

- Verifying digital signatures of VCs
- Signing and verifying JSON-LD proof-chains and proof-sets
- Managing JSON-LD policies via GitOps
- Policy evaluation to ensure policy driven trust
- DID resolving endpoint to resolve DID documents
- Derived Verifiable Credentials SHOULD be supported in future releases

User interaction is expected at:

- Policy Administration Point
- Git Ops

2.3. Product Constraints

⏩ IDM.TSA.00001 **The document IDM.AO is the common basis for this functional specification**
The architecture document [IDM.AO] is an essential part of this specification and a prerequisite for understanding the context. The specifications and requirements from the Architecture Document MUST be considered during implementation. ⏪

⏩ IDM.TSA.00002 **Micro Service Architecture**
For a better scale out and decentralization, the product architecture MUST a micro service architecture. The modules MUST NOT be tightly integrated into the IAM solution, as Plugin or Extensions, rather should interact with the said system through standard APIs and Protocols. ⏪

⏩ IDM.TSA.00003 **Policy Language**
The policy language SHOULD be REGO [Rego] and MUST be uniform across within the trust services product. ⏪

⏩ IDM.TSA.00004 **Policy Decision**
The policy decision should be evaluated by a separate policy engine. ⏪

⏩ IDM.TSA.00005 **Policy Storage**
The policy storage is based on Git and MUST use the secure integrity methods of the tooling. ⏪

⏩ IDM.TSA.00006 **Universal DID Resolver**
The product MAY have a dependency to existing Universal DID Resolvers, so far, they can run locally. ⏪

⏩ IDM.TSA.00007 **Organization Key Pair(s)**
The product MUST have one or multiple key pairs for signing policies or data evaluation results. This key material has to be stored in a secure environment and the public key part MUST be published on the identity network next to the DID. ⏪

⏩ IDM.TSA.00008 **Open-Source Versions**
The Open-Source components SHOULD be used in the latest versions. ⏪

2.4. User Classes and Characteristics

User Class	*Description*	*Frequency*	*Expertise*	*Privilege Level*	*Product Usage*
Policy Developer	The policy developer develops and maintains Policies.	Low	High	Low	Commits Policy changes
Policy Administrator	The policy administrator develops and maintains policies	Low	High	High	Administration UX/Administration Endpoint Git

	within the product for usage in production.				Repositories
Administrator	Setup the system and maintain git repositories and the operations of the product included the network around it.	Low	High	Low	Maintenance
External Systems	Any API or component which accesses the system from outside.	High	High	Low	Evaluation Requests, Cache Users, Information imports/exports

***Table 2**: User Classes and Characteristics*

2.5. Operating Environment

Please refer to [TDR] for further binding requirements regarding the operating environment.

⏩ IDM.TSA.00009 **TLS Protected Endpoints**
To protect the product endpoint(s), it's necessary to support a network infrastructure e.g., load balancers/proxies which MUST support TLS encryption. The encryption MUST meet the requirements listed in the chapter for security requirements. ⏪

2.6. User Documentation

Please refer to [TDR] for further requirements regarding documentation.

⏩ IDM.TSA.00010 **Participant Administration Documentation**
The documentation MUST contain:

- Installation Manuals
- Cryptographic Initialization (if applicable)
- Description of Deployment/Compile Process
- Description of the Automatic Tests / Verification
- How to build the products from source code ⏪

⏩ IDM.TSA.00011 **Participant Documentation**

The documentation MUST contain:

- Short Software Description/Usage
- Usage Guide
- GDPR Design Decisions
- Security Concept
- Operations Concept
- FAQ
- Keyword Directory ⏪

2.7. Assumptions and Dependencies

An understanding of the overall Gaia-X architecture and philosophy is necessary. Please refer to [TAD] und [PDR].

2.8. Apportioning of Requirements

Feature	*Priority*
Policy Evaluation	*1*
DID Document Resolving	*1*
Trusted Caching	*1*
Trusted Information Exchange	*1*
JSON-LD Verification	*1*
JSON-LD Signing	*1*
Task Coordination	*2*
Trust Chain Verification	*2*
Policy Management	*3*
Git Ops	*3*
eIDAS compliant Signatures	*4*

***Table 3**: Apportioning of Requirements*

3. Requirements

Further binding requirements can be found in [TDR].

3.1. External Interfaces

3.1.1. User Interfaces

⏩ IDM.TSA.00012 **Policy Administration**
The policy administration is done over git access points and open-source development tools which are secured by an encrypted transport protocol. ⏪

3.1.2. Hardware Interfaces

⏩ IDM.TSA.00013 **Security Hardware**
Product does not include any hardware components but a connection to an HSM MUST be possible as described into the security requirements chapter. ⏪

3.1.3. Software Interfaces

⏩ IDM.TSA.00014 **Distributable Cache**
The product needs an interface to the distributable cache to query cached data. The protocol depends on the selected technology, but the interface MUST be supported by the policy language as built-in function. ⏪

⏩ IDM.TSA.00015 **DID Resolver**
The DID resolver MUST provide an HTTP interface to use the resolving functions inside of the trusted services. This interface MUST provide the functionality to use the DID resolver by provision of the DID. ⏪

3.1.4. Communications Interfaces

⏩ IDM.TSA.00016 **Trusted Information Endpoint**
The trusted information endpoint provides a HTTP API to get identity information, trust lists and other secured and signed information from the trusted services. ⏪

⏩ IDM.TSA.00017 **Trusted Cache Endpoint**
An endpoint to manage information which has to be securely cached into the cache system. ⏪

⏩ IDM.TSA.00018 **Policy Evaluation Endpoint**
This endpoint provides the functionality to evaluate policies, to obtain a result to a specific input. ⏪

⏩ IDM.TSA.00019 **Signing/Validation Interface**
This HTTP interface provides functionality around validation and signing of JSON LD files. ⏪

3.2. Functional

3.2.1. Git Ops

⏩ IDM.TSA.00020 **Secured Git Server**

The product MUST contain a git server which provides the storage for policies and other content. It MUST be secured with key material from the security infrastructure, and all developers need to check in content only signed to track back the commits.

Interface
Git Interface

⏪

⏩ IDM.TSA.00021 **Update policy git flow**

The product MUST have an automated policy update-flow, that will always take the corresponding main branch to a policy, sign it digitally for public or private repositories and publish the new policy. It MUST follow the pull and merge principles of git.

Interface
GitOps

Input
Policies

Output
Updated and signed policies for the according repository (private / public)

Acceptance criteria

1. A successful edited and signed policy
2. Successful tests of the policy
3. Activation of new policies

⏪

3.2.2. Policy Management Module

⏩ IDM.TSA.00022 **Authentication for Policy Administration**

The user MUST authenticate to get access to any action of the Policy Administration Point. The ways how to authenticate against the administration point MAY be chosen. The recommended communication channel MUST be encrypted and SHOULD use SSH or TLS.

Constraints
Use of internal authentication system to protect git in a standard way

Interface
Policy Administration Point

Input
Valid credentials

Output
Authenticated and secured connection to Administration Endpoint.

Acceptance criteria

1) Exception if no authenticated identity is present (401)
2) Access to the administration point, if an authenticated user is present.

⏮

⏭ IDM.TSA.00023 **UX Behavior for Policy Administration**
It SHOULD be possible to integrate the policy administration into standard IDE with standard capabilities of code highlighting, code suggestions and beautifying. It SHOULD be able to load, edit, copy, safe, test. The user interface MUST be able to generate policy groups and handle policy information within different groups. Changed external signed policies, imported from other participants, MUST be rejected from check-in.

Constraints
Depends on the Open-Source Tool

Interface
Policy Administration Point

⏮

⏭ IDM.TSA.00024 **Policy versioning**
Policy versioning MUST follow semantic versioning standard [Sem.Vers]. ⏮

⏭ IDM.TSA.00025 **Policy Bundles**
Policy bundles MUST contain further information, e.g., data input JSON files. They MUST contain meta-data and references like resolvable DIDs to be verifiable and trustworthy. The bundle MUST be signed in a productive state or before the export. It MAY be chosen, how the signing and verification is standardized. For instance, the metadata can be signed for the included files of the bundle, or the bundle itself can be signed as a compressed package. ⏮

IDM.TSA.00026 **Policy Creation and Structure**

The policies SHOULD be created in rego language [Rego] under a group stored in a git repository. The content format for storing MAY be chosen. The repository structure SHOULD be chosen in a way that the group policy name and version are given:

/{repo}/policies/{group}/{policyname}/{version}/evaluation

The exact format MAY be chosen, but all folders MUST map 1:1 to an URL which can be executed over HTTP with a versioning and a group. For instance:

http://localhost:8080/aisbl/policies/aisbl/trustedIssuerList/1.0/evaluation
http://localhost:8080/aisbl/policies/mycompany/loginpolicy/1.0/evaluation

The HTTP routes MUST be created and set to active, if a commit in the productive repository was successfully created or any additional repository was synchronized to the productive repository.

A policy flow can look like the following flow chart:

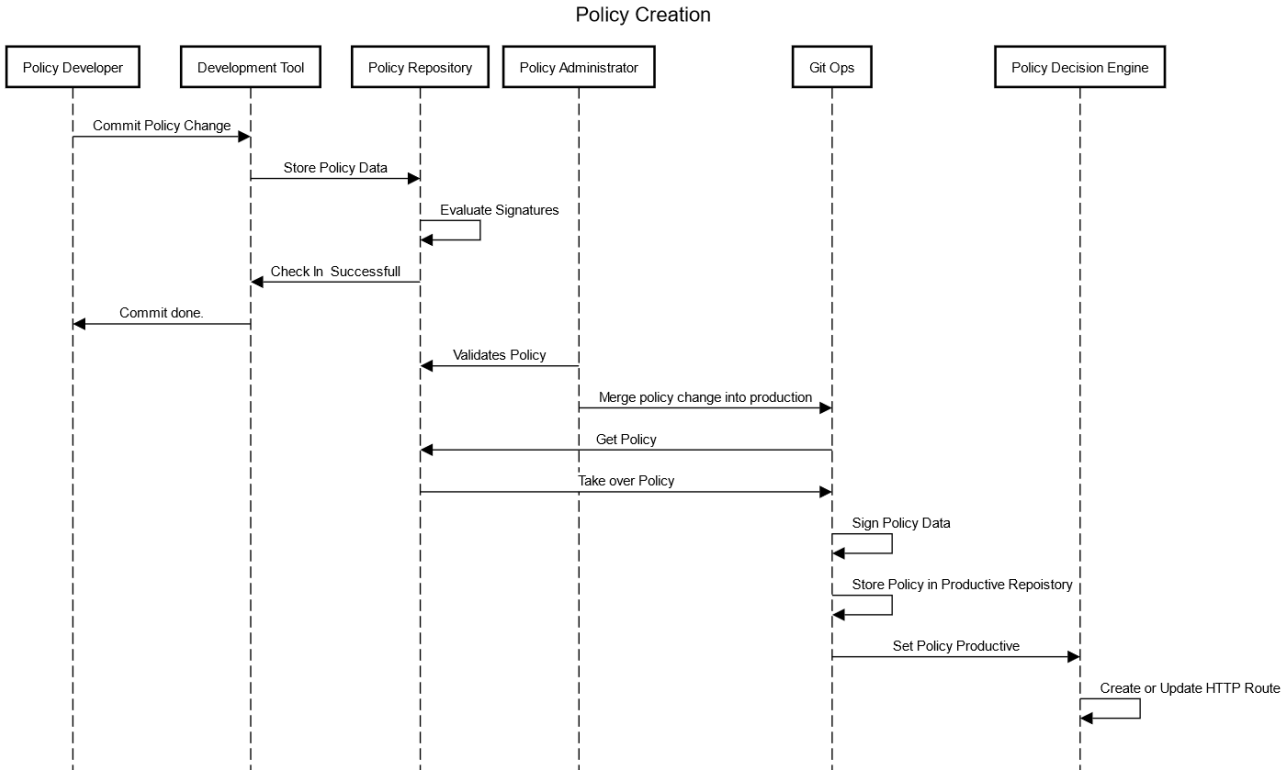


***Figure 3**: Policy Creation*

The content of the policies SHOULD be stored and signed as a bundle for an uncomplicated import and export of other policy bundles. The bundle MUST contain the owner DID of the policy. For signing a key pair MUST be used which is linked to the DID (public key inside of the DID document).

Acceptance Criteria

1) Available and responsive route after committing a policy and a takeover to production
2) No route available after deleting a policy from the productive repository.
3) Synchronized Policy Bundles are taking over to production and have active routes
4) Policy Bundles MUST be verifiable against DID public keys

IDM.TSA.00027 **Policy Actions**

Each policy route MUST have the capability to provide some static actions for future purposes in the following pattern:

/{repo}/policies/{group}/{policyname}/{version}/**{action}**

Possible actions SHOULD be “evaluation” and “lock”. It MAY be solved in a different way, but the actions MUST be statically in the URL.

Action	Input	Output	Verbs	Description
evaluation	JSON Object	JSON Object	POST	Evaluates a policy and delivers a result.
lock	-	-	POST, DELETE	Locks (POST) or unlocks (DELETE) the policy for evaluations.

***Table 4**: Policy Actions*

Acceptance Criteria

1) Evaluation action returns a policy result for an input
2) Policy is no longer evaluable, if lock for the policy was called
3) Policy is evaluable again, if unlock for the policy was called

IDM.TSA.00028 **Policy Import and Export**

The policy administrator MUST be able to configure policy import endpoints where the system downloads automatically new policy bundles. The policy export endpoint itself MUST be configurable in that way, that the policy administrator can decide which bundles from the productive repository

can be exported. Each import and export MUST be signed with a key registered in the DID document and verifiable against the public keys from the DID document defined in the bundle. For this purpose, the DID resolver MUST be used. Import and export MUST be observable for the configured repositories.

Acceptance Criteria

1) The export of a policy bundle MUST be signed with a keypair which is evaluable against the public keys registered in the DID document
2) Import and Exports are observable

⏮

3.2.3. Universal DID Resolver

⏭ IDM.TSA.00029 **DID Document Resolving**

A universal DID resolver takes a DID as an input, obtains the data and constructs a standardized DID document. Supported did methods MUST be DID:WEB, DID:IDU, DID:SOV, DID:IPID, DID:KEY. Others MAY be supported as well. It is RECOMMENDED to use an existing universal resolver.

Interface

DID Resolve

Input

The request must follow the standard the supported DID methods pattern specifications.

Output

The responding DID document.

⏮

⏭ IDM.TSA.00030 **DID Resolver HTTP Interface**

The resolver functionality MUST be usable by an HTTP interface, that other components can easily trigger and receive DID documents.

Interface

DID Resolve

Input

A DID.

Output

The responding DID document.

⏮

3.2.4. Policy Decision Engine

IDM.TSA.00031 **Call of external URLs**

The decision engine MUST have the capability to call HTTP URLs with free chosen query parameters, headers, and request bodies for any HTTP verb within the policy execution. This MUST be possible out of the defined policy to load external data dynamically into the execution context. It MAY be created also helper components e.g., built in functions, to support this feature outside of the policy policy execution context.

Constraints
Policy Language
Policy Execution

Input
HTTP URL with Parameters, Header and Request Body.

Output
JSON HTTP Response.

IDM.TSA.00032 **Support of Built-in Functions**

The policy decision engine MUST support built-in functions to enhance the policy execution language later with more complex functionality. (e.g., calculations, signings, hashing)

Constraints
Rego Policy Language [Rego]

Acceptance Criteria

1) Documentation how the policy decision engine can be enhanced with new custom built in functions

IDM.TSA.00033 **Use of distributable cache in policy**

The policy decision engine MUST support usage of the distributable cache. This SHOULD be realized over built in functions to make the policy creation easy and intuitive, but it MAY be realized with other protocols which MUST executable from the policy itself (e.g., HTTP).

Constraints
Rego Policy Language [Rego]

Acceptance Criteria

1) Cache content must be usable in the policy evaluation

IDM.TSA.00034 **Policy Execution**

The policy execution is a critical point in the product. The execution can reach millions of executions at the same time in a very big system. This means the execution MUST be high paralyzed and the architecture scalable. The execution MAY cache information like static data, policy rego content etc. in memory to reach a high responsive design. The same cache MAY be used for policies within the same repo, group and version to limit the memory usage. It MUST be able to support asynchronous tasks or long running operations.

3.2.5. Policy Evaluator

IDM.TSA.00035 **Consistent Policy Evaluation Routes**

The generated routes by the management system MUST be consistent during an update of the policy. It MAY be realized with additional open-source components within the product to fulfill this requirement. It's also allowed to set some requests temporarily on hold during the switch of policies and their related data. To simplify the process, the policy store MAY be immutable.

Acceptance Criteria

1) All routes have the same policy applied before any external call can get a wrong result during the switch process

IDM.TSA.00036 **High Available Policy Evaluation Routes**

The generated policy endpoints by the management system MUST be static, so that they can be stored in the load balancer or infrastructure components. It's not allowed to generate routes temporarily or alias it in a way that they can match temporary other targets. It MUST be guaranteed that the endpoints never execute different policies.

Acceptance Criteria

1) The endpoint MUST be mappable on a load balancer/API gateway.
2) The endpoint MUST have an availability up to 99,8%, also on policy change.

IDM.TSA.00037 **Task Controller Integration**

The policy decision engine MUST be able to use the task controller for the instantiation of tasks and tasks list. For this purpose, the engine SHOULD implement and build in function to support the usage of the task controller so easy as possible.

Input
A task(list) name with input.

Output
A task id.

Acceptance Criteria

1) The policy is able to instantiate tasks and task lists

3.2.6. Task Controller

IDM.TSA.00038 **Task Definition**

A task is a data structure which MUST contain at least:

- unique task identifier
- unique task name
- URL
- HTTP Action (GET, POST,)
- request header
- request body
- request policy
- response policy
- response header
- response body
- finalizer policy
- key metadata (key, namespace, scope)
- response code
- state

Each task SHOULD be held in memory during the processing. The distributable cache MAY be used. From a security perspective, the task definitions SHOULD NOT be created dynamically.

IDM.TSA.00039 **Task List Definition**

A task list is a data structure which MUST contain at least:

- unique TaskList Identifier
- Task Groups
- state
- finalizer policy
- execution metadata

Each task group MUST contain at least:

- a task name
- state
- finalizer policy
- execution metadata

The task groups SHOULD contain Metadata to support the execution of the tasklist.
Each task list SHOULD be held in memory. The distributable cache MAY be used. From a security perspective, the task list definitions SHOULD NOT be created dynamically.

IDM.TSA.00040 **Task(list) Instantiation and State Check**

To trigger the instantiation of the task(lists) the task controller MUST have an interface which is able to instantiate a task or tasklist by entering the unique name and other parameters like key metadata etc. Over this unique name the instance MUST be created. The result of the instantiation MUST return a unique task id to identify the created task later over the state check over the interface. This state check MUST return over the unique ID the state of the task(list). The states SHOULD be “created”, “pending”, “done” and “gone”. If a HTTP interface is used, the states SHOULD be represented over HTTP response codes. (201=created, 204=no content, 200=done,410=gone) Is the result “done”, and the task(list) result was successfully delivered to the state requestor, the task can be removed from the cache.

IDM.TSA.00041 **Instantiation/Execution of Tasks**

The task controller MUST be able to instantiate tasks in the described format of the task definition. Available task definitions MUST be pre-configured by the policy administrator. If the task is created, a background worker MUST handle the tasks asynchronously. The defined request policy in the task MUST be executed before the HTTP call to get additional headers and the request body (It MUST be also possible to push parameters into the task during the instantiation). All defined headers are put into the HTTP request next to the key metadata and the result of the task is stored into the task response body field, next to the response codes. If a response is received, the response policy SHOULD be executed with the response body and headers to evaluate the state of the task. This policy SHOULD decide then, if any further actions are necessary. When the final state of the task is

reached, the finalizer policy is executed, before the final response body and the key metadata is written to the distributable cache.

Acceptance Criteria

1) Task controller is able to instantiate and execute tasks
2) The result of the tasks is accessible over the distributable cache (key metadata)
3) Input parameters for the task request headers/request body/ key metadata are respected during the instantiation

⏮

⏭ IDM.TSA.00042 **Instantiation/Execution of Task Lists**

The task controller MUST be able to instantiate task lists in the described format of the task list definition. Available task list definitions MUST be pre-configured by the policy administrator. If the task list is created, a background worker MUST handle the tasks asynchronously. The execution metadata of the defined task lists controls the task processing by the task controller. Available processing options SHOULD be sequential, parallel, and grouped. Depending on the processing option, the task groups are executed, and the state is separately stored group by group. If each group has a "done" state, the entire task list is done, the result can be finalized over the policy and the result can be stored in the cache.

Acceptance Criteria

1) Task controller is able to instantiate and execute task lists
2) The result of the task lists is accessible over the distributable cache (key metadata)
3) Input parameters for the task list request headers/request body/key metadata are respected during the instantiation

⏮

⏭ IDM.TSA.00043 **Sequential processing**

If a task list or task group is sequentially processed, the task controller MUST push the final result from one task to the next one as input. If one item is failing the entire sequence is failing.

⏮

⏭ IDM.TSA.00044 **Parallel processing**

If a task list or task group is parallel processed, the task controller MUST execute everything in parallel. If one item is failing, the controller SHOULD ignore this results in the finalization.

⏮

⏭ IDM.TSA.00045 **Cache Event Subscription**

If a data update event occurs from the distributable cache, the task controller MUST execute a policy which evaluates whether any further tasks must be created. If any task is configured, the tasks MUST be created for execution. The task has the same metadata as the event. (key, namespace and scope) For instance: A proof data object is inserted from the OCM into the cache. This object contains an DID from the issuer. The policy evaluates the schema of the data object and returns Task "IssuerProof". The task controller inserts this task to the tasklist by adding the http URL of the OCM proof manager with the DID of the issuer as parameter. Some seconds later, the proof from the issuer arrives and the policy returns null (nothing to do).

Constraints
Distributable Cache Event
Policy Evaluation

Input
An Insert/Update Event

Output
A task result or null.

Acceptance Criteria

1) After an event, the task must be created as pre-configured (or dynamically by policy)

⏪

3.2.7. Secret Store

⏩ IDM.TSA.00046 **Policy Secret Store API**
The secret store MUST be able to deliver key-pairs to signature functions to support appropriate and secure handling of sensitive key material during these operations. It's also RECOMMENDED to align all crypto operations and the secret store with the [IDM.OCM] team. ⏪

3.2.8. JSON-LD signatures and validations

⏩ IDM.TSA.00047 **LD Proofs in JSON-LD**
JSON-LD proofs SHOULD be supported to sign verifiable credentials and verifiable presentations and verify all proofs within the JSON-LD data (including the embedded proofs of Verifiable Presentations). It MUST follow the W3C guidelines for LD proofs [LD.Proofs]

⏪

⏩ IDM.TSA.00048 **Proof Sets in JSON-LD**

JSON-LD proof sets SHOULD be supported to link multiple entities to the same data, when no order of proof is required. Therefore, multiple proofs are attached to the linked data document. This can be useful for contracts or shared policies. It MUST support signing and verifying the VCs.

Example:

```
{
 "@context": [
  "https://www.w3.org/2018/credentials/v1",
  "https://www.w3.org/2018/credentials/examples/v1"
 ],
 "title": "Hello World!",
 "proof": [{
  "type": "Ed25519Signature2018",
  "proofPurpose": "assertionMethod",
  "created": "2019-08-23T20:21:34Z",
  "verificationMethod": "did:example:123456#key1",
  "challenge": "2bbgh3dgjg2302d-d2b3gi423d42",
  "domain": "example.org",
  "jws": "eyJ0eXAiOiJK...gFWFOEjXk"
 },
 {
  "type": "RsaSignature2018",
  "proofPurpose": "assertionMethod",
  "created": "2017-09-23T20:21:34Z",
  "verificationMethod": "https://example.com/i/pat/keys/5",
  "challenge": "2bbgh3dgjg2302d-d2b3gi423d42",
  "domain": "example.org",
  "jws": "eyJ0eXAiOiJK...gFWFOEjXk"
 }]
}
```

IDM.TSA.00049 **Proof Chains in JSON-LD**

JSON-LD proof chains SHOULD be supported to link multiple entities to the same data, when proof sequences are required. This can be useful for notary counter-signing a proof that had been created on a document. It MUST support signing and verifying the VCs.

Example:

```
{
 "@context": [
```

```
  "https://www.w3.org/2018/credentials/v1",
  "https://www.w3.org/2018/credentials/examples/v1"
 ],
 "title": "Hello World!",
 "proofChain": [{
  "type": "Ed25519Signature2018",
  "proofPurpose": "assertionMethod",
  "created": "2019-08-23T20:21:34Z",
  "verificationMethod": "did:example:123456#key1",
  "domain": "example.org",
  "jws": "eyJ0eXAiOiJK...gFWFOEjXk"
 },
 {
  "type": "RsaSignature2018",
  "proofPurpose": "assertionMethod",
  "created": "2017-09-23T20:21:34Z",
  "verificationMethod": "https://example.com/i/pat/keys/5",
  "domain": "example.org",
  "jws": "eyJ0eXAiOiJK...gFWFOEjXk"
 }]
}
```


3.2.9. Information Hub

IDM.TSA.00050 **Trusted information export**

It MUST be possible to export data towards other HTTP services and components by using the information hub. The source of the data MUST be the distributable cache, any policy output or both. Signing the exported data SHOULD be done via a policy, but the format MUST be JSON LD content with LD-Proofs, signed by the participants DID key pair. The export output MUST be configurable for JSON transformations (e.g. modification of verifiable presentations), export policies and creation cache intervals. (the export files SHOULD not be created on each call). The export endpoint protection MUST be configurable by the policy administrator.

Constraints

Policy Evaluator

Distributable Cache

Interface

Trusted Information Endpoint

Input

An export HTTP request

Output

A signed JSON-LD response with LD-Proof, if export is allowed.

Acceptance criteria

1) The exported information MUST be signed
2) The exported information MUST be signed and verifiable by a resolvable DID or any public key
3) Policy export configurations MUST only be created and editable by policy administrators

IDM.TSA.00051 **Trusted information import**

It MUST be possible to import data towards the trust services from other HTTP services and components. Verifying the imported data SHOULD be done via a policy. The import sources MUST be configurable for JSON transformations, accept policies, import interval, http sources and allowed DIDs. The DID service endpoints MUST be respected in the configuration as well. The imported data MUST be pushed to the distributable cache after a successful validation against the DIDs public key. The import endpoint protection MUST be configurable by the policy administrator.

Interface

Trusted Information Endpoint

Policy Evaluator

Distributable Cache

Input

An import HTTP request

Output

A signed JSON-LD response with LD-Proof.

Acceptance criteria

1) The data import(er) MUST be trust-worthy
2) The imported information MUST be signed and verifiable by a resolvable DID or any public key.
3) Policy import policies MUST only be created and editable by policy administrators

IDM.TSA.00052 **Content Modifications**

The information hub MUST support content modifiers which deliver functionalities like flat JSON files, transform outputs or sign JSON-LD files during the import and exports. This can be supported by any policy for configuration purposes. For instance, a request maps to a policy evaluation result and to a cache content. One is XML and one is JSON. If the desired output is JSON, one file must be converted and the second merged to the first one, before the file is responded to by the requester.

IDM.TSA.00053 **Trusted Identity Information**

The endpoint MUST deliver to a key, namespace and scopes, information about an identity. This is a GET action which returns a flatten JSON file. This functionality maps to the description under "Content Access" in the distributable cache. If one scope is missing or the TTL acceptance value is not high enough (e.g., minimum TTL of 3 seconds), the reload MUST be triggered over the task controller, if configured. In this case a task id MUST be returned to indicate an asynchronous operation. The external caller MUST be able to send the TTL acceptance value within the call (e.g., TTL acceptance of 0 seconds).

3.2.10. Distributable Cache

IDM.TSA.00054 **In Memory Caching**

The distributable cache MUST be implemented as an in-memory solution. From a privacy and GDPR perspective, the cache MUST NOT implement any persistence or recovery.

Acceptance Criteria

1) The cache is empty when the environment is restarted
2) Given Input is hold in memory

IDM.TSA.00055 **Cache Distribution**

The cache content MUST be distributable over different servers, which can dynamically join or leave the cluster. The connections between the servers and/or the data distribution itself MUST be secured with cryptographic mechanisms. (e.g., TSL, Encryption, SPIFFE etc.)

Acceptance Criteria

1) New Server can join and the cache is synchronized, a call against the new server delivers a result
2) One Server can leave without interrupt the rest of the cluster

IDM.TSA.00056 **Internal Cache Interface**

The internal cache interface MUST use a technology, to reach the highest performance. The cache MUST be accessible over an internal interface of the programming language or other ways which increase the performance.

IDM.TSA.00057 **Key Metadata**

For the cache access, the following key metadata MUST be used:

- Key (string)
- Namespace (string)
- Scope (string)

The data types can be chosen differently, depending on the cache technology.

IDM.TSA.00058 **Content Format**

The content of the cache MUST support the storage of JSON or similar structures (e.g., CBOR, Protobuf). Other formats MAY be supported as well, but Memory footprint MUST be considered with the choice of the format. The content MUST be converted for input and output in the cache. For instance, if the choice is to use a binary format, JSON can be converted to binary and back, to support more efficient memory usage.

IDM.TSA.00059 **Content Access**

The cached content MUST be accessible over a key (e.g., a DID), a namespace and an array of scopes, which results in an array of JSON documents. For instance, the access to the cache can be the following:

Key: DID:sov:2358585
Namespace: Login
Scopes: administration, read, visitor

Result: {“name”:”userX”,”iss”:did:sov:33333},
{“membership”:”company”,”iss”:did:sov:1111}

To optimize the access, it MAY be optimized by flatten the access pattern like:

Key: DID:sov:2358585
Namespace: Login:administration

Independent from the format, the result for the accessor MUST be a flatten JSON structure. In the flattening step, it SHOULD namespace duplicated claims if they are semantically different (e.g., using JSON-LD context). If two JSON structures bring a duplicated claim which is semantically the same claim (say two JSON structures bring Name and Surname, but the values are different) then a policy shall decide whether to: take one of them or discard all of them.

Constraints
Supported data format of the cache technology
Policy Decision Engine

Acceptance Criteria

1) *Result of a flat JSON file to an DID, namespace and scopes*
2) *Duplicate Handling during the flattening of multiple documents*

IDM.TSA.00060 **Content TTL**
The content time to live time, SHOULD be set over a policy depending on the received data input.

Constraints
Policy Decision Engine

Input
A JSON Structure.

Output
A time to live value which is set during the cache store/update operation.

Acceptance Criteria

1) Policy decided which TTL is used
2) Object is removed after the expiration

IDM.TSA.00061 **External Cache Input**

The external cache input MUST be implemented over HTTP. The request body MUST contain the JSON payload which has to be cached. Within the headers the access keys have to be sent. (namespace, scope, key) The supported actions for the input API MUST be GET, PUT, POST and DELETE. PATCH is OPTIONAL, to get, create, update and delete cache content. The key metadata for the input MUST be evaluated by a policy before inserted in the cache.

Constraints

Key metadata is selected by the policy (key, namespace, scope)

Interfaces

Trusted Cache Endpoint

Input

A JSON structure, with a key, namespace and scopes.

Output

An appropriate HTTP response.

Acceptance Criteria

1) If successful, but not yet in the cache, the system returns 201
2) If successful, but just updated, the system returns 200
3) If not successful, the system returns 400

IDM.TSA.00062 **Input Observation**

The distributable cache MUST provide an event when new inputs are received over the trusted cache endpoint. This event MUST contain the access key to identify the created item in the cache.

Input

An JSON structure, and key metadata in the header.

Output

An event is fired to the subscribers with the key metadata (For instance the Task Controller or the Information Hub)

IDM.TSA.00063 **Input Subscriptions**

To subscribe on external changes, the Cloud Events Pattern MUST be supported. The incoming data MUST be stored in the cache.

⏪

3.2.11. eIDAS

⏩ IDM.TSA.00064 **eIDAS compliant Signature Creation / Validation**
Signatures must be generated/verified in compliance with eIDAS so that legally secure trust can be achieved. This should include the eIDAS signature types basic, advanced, and qualified. The implementation variant must be selected individually in coordination with the used technology.

⏪

3.3. Other Nonfunctional Requirements

3.3.1. HTTP Requirements

⏩ IDM.TSA.00065 **HTTPS**
All HTTP Endpoints MUST be protected by TLS 1.2 (all protocol version numbers SHOULD be superseded by upcoming standards) Each endpoint of the product MUST support TLS certificates which are configurable by the administrator of the system. ⏪

⏩ IDM.TSA.00066 **HTTP Protocol Definitions**
All HTTP Endpoints MUST follow RFC 7231[2] and RFC 5789[3], but it MAY be chosen what of the protocols is necessary to realize the functionality. For problem reports the RFC7807[4] MUST be used in combination with Standard HTTP Error Codes. ⏪

3.3.2. Configuration

⏩ IDM.TSA.00067 **Configuration**
All components MUST support one of the major configuration formats (yaml, json, ini, environment variables) wherever configuration is required. If environment variables are overwriting an actively set configuration, a warning SHOULD be logged. ⏪

3.3.3. Logging Requirements

[2] https://tools.ietf.org/html/rfc7231
[3] https://tools.ietf.org/html/rfc5789
[4] https://tools.ietf.org/html/rfc7807

⏩ IDM.TSA.00068 **Data Minimization**

From GDPR perspective the product MUST NOT log data which is related to personal information. (e.g., Usernames, Birth Dates etc.) The product MUST only log data, which is relevant to technical operations, except for the purpose that, in the event of an incident, enable reconstruction of the sequence of the message exchange for establishing the place and the nature of the incident. The data shall be stored for a period of time in accordance with national requirements and, as a minimum, shall consist of the following elements:

(a) node's identification

(b) message identification

(c) message data and time

All logged data/information MUST be documented in the GDPR design decisions for a GDPR review. ⏪

⏩ IDM.TSA.00069 **Logging Frameworks**

The product MUST support logging frameworks e.g., graylog, fluentD or logstash to support logging and analysis by enterprise infrastructures. The supported framework MAY be chosen for the first version, but it MUST support potentially the most common open-source logging solutions. The final solution MUST be aligned with the other subcomponents. It MUST be sketched in the operations concept how the support of multiple solutions is given in the future. ⏪

3.3.4. Monitoring Requirements

⏩ IDM.TSA.00070 **Monitoring Frameworks**

The product MUST support monitoring frameworks e.g., grafana to support the analysis of incoming data by the enterprise infrastructures. The supported framework MAY be chosen for the first version, but it MUST support potentially the most common monitoring solutions. (e.g., Zabbix) The final solution MUST be aligned with the other subcomponents. It MUST be sketched in the operations concept how the support of multiple solutions is given in the future. ⏪

⏩ IDM.TSA.00071 **Alerting Frameworks**

Additional to the Monitoring Frameworks an Alerting framework (e.g., Prometheus or Cloud Based) MUST/MAY be in place at least in the System nodes to promptly communicate to e.g., System Administrators or owners the occurrence of an event in form of a security incident or application/system malfunction or anomaly. ⏪

3.3.5. Performance Requirements

⏩ IDM.TSA.00072 **Performance Scalability**

The performance of the product MUST be scalable. This MUST be demonstrated in a load demonstration example. The optimal scalability SHOULD be in the best case a linear behavior of minimum 50% more performance by each additional instance. ⏪

⏩ IDM.TSA.00073 **Performance by Design**
The product SHOULD be designed and implemented in a way, that the implementation is non-blocking and performance oriented. It SHOULD be a microservice architecture, but it MAY follow other concepts. The decision MUST be documented. ⏪

3.3.6. Safety Requirements

⏩ IDM.TSA.00074 **Recovery Point Objective (RPO)**
The RPO for the product MUST be 0 for a single and multiple instance(s). It MAY be higher by configuration or deployment, decided by the user. ⏪

⏩ IDM.TSA.00075 **Recovery Time Objective (RTO)**
The RTO for the product MUST be one Minute for a single instance. For multiple instances the RTO MUST be 0. ⏪

⏩ IDM.TSA.00076 **Mitigation of Single Point of Failure threats**
Critical components in the Gaia-X Ecosystem MUST be identified and strategies to warranty their availability and scalability MUST be implemented. ⏪

3.3.7. Security Requirements

3.3.7.1. General Security Requirements

Each Gaia-X Federation Service SHALL meet the requirements stated in the document "Specification of non-functional Requirements Security and Privacy by Design" [NF.SPBD].

Federation Services specific requirements will be documented in the next chapter.

3.3.7.2. Service Specific Security Requirements

This chapter will describe the service specific requirements, which will extend the requirements defined in the chapter above.

⏩ IDM.TSA.00077 **Cryptographic Algorithms and Cipher Suites**
Cryptographic algorithms and TLS cipher suites SHALL be chosen based on the recommendation from the German Federal Office for Information Security (BSI) or SOG-IS. These recommendations and the recommendations of other institutions and standardization organization are quite similar[5] [CryptoLen]. The recommendations can be found in the technical guidelines[6] TR 02102-1 [TR02102-1] and TR 02102-2 [TR02102-2] or SOG-IS Agreed Cryptographic Mechanisms[7] [SOG-IS]. ⏪

[5] See https://www.keylength.com/en for a comparison
[6] See https://www.bsi.bund.de/EN/Service-Navi/Publications/TechnicalGuidelines/tr02102/tr02102_node.html
[7] See https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.2.pdf

IDM.TSA.00078 **Digital Certificates**

For digital certificates and cryptographic signatures in the context, the major requirements on cryptographic algorithms and key length MUST meet the definitions in the following table (as of 2020):

Signature Algorithm	Key size	Hash function
EC-DSA	Min. 250 Bit	SHA-2 with an output length ≥ 256 Bit or better
RSA-PSS (recommended) RSA-PKCS#1 v1.5 (legacy)	Min. 3000 Bit RSA Modulus (n) with a public exponent e > 2^16	SHA-2 with an output length ≥ 256 Bit or better
DSA	Min. 3000 Bit prime p 250 Bit key q	SHA-2 with an output length ≥ 256 Bit or better

***Table 5**:Requirements on cryptographic algorithms and key length*

Named curves SHALL be used for EC-DSA (e.g., NIST-p-256).

IDM.TSA.00079 **TLS Certificate Validity Periods**

In general, the recommended validity period for a certificate used in the system should be one year or less. Under some circumstances (for example RootCA) the certificate validity can be extended. Certificate owners MUST ensure that valid certificates are renewed and replaced before their expiration to prevent service outages.

IDM.TSA.00080 **Security by Design**

The software security MUST be from the beginning a design principle. Means separation of concerns, different administrative roles, especially for private key material and separate access to the data MUST be covered from the first second. It MUST be described in the security concept, what are the different security risks of the product and how they are mitigated (e.g., by Threat Modeling Protocols)

IDM.TSA.00081 **Installation of Critical Security Updates**

Node operators SHALL deploy security critical updates without undue delay.

IDM.TSA.00082 **Avoid HTTP Request Smuggling**

To avoid Request Smuggling attacks, the product MUST implement a standard which handles this kind of attack by design, because the attack vector results in an insufficient implementation of the header handling. The chosen way to handle it MUST be shared to the other implementers of all other subcomponents within IDM & Trust and MUST be described in the security concept.

IDM.TSA.00083 **HTTP Pentesting**

All HTTP parts of the product has to be pen tested, for the following criteria:

1) Unauthorized Access to the System MUST be tested
2) Unauthorized Actions MUST be triggered without a user action
3) Endpoints MUST be tested for HTTP smuggling attack vectors
4) If a datastore is present over HTTP, illegal data access MUST be tested

It's RECOMMENDED to test more attack vectors and document it for the purpose to mitigate it in later versions. ◂◂

▸▸ IDM.TSA.00084 **Storage of Secrets**

The storage of secret information such as private keys MUST take place in state-of-the-art secure environments to protect secret data confidentiality and integrity. Examples of this are Secure Enclaves, TPMs, HSM or Secure Vaults. In case (Personal) Agents are not equipped with a secure storage it MAY also be possible to store the secrets in a third party (e.g., Cloud) provider (e.g., Secure Wallet) that MUST provide overall the same level of security as the aforementioned methods. ◂◂

▸▸ IDM.TSA.00085 **Secret Distribution and Usage**

The product MUST ensure interoperability of cryptographic primitives and components by public standards and MUST use secure state of the art methods to create and import secrets into the secure storage, as well as performing cryptographic operations (e.g., encryption or digital signatures). For Key distribution, state of the art DKMS methods MUST be implemented. ◂◂

▸▸ IDM.TSA.00086 **Support for Potential Requirements for Secret Storages**

Devices that hold cryptographic information and perform cryptographic functions MUST be compliant with the standard PKCS #11. Moreover, the products MUST be potentially eligible for a FIPS-140-2 or ETSI/Common Criteria certification with the minimum-security level necessary to operate securely in the Gaia-X ecosystem. Security Levels in FIPS-140-2 range from 1 to 4. Current HSM Cloud Service offerings (AWS, Azure, GCP) are Level 3 (Source: https://en.wikipedia.org/wiki/FIPS_140-2). ◂◂

▸▸ IDM.TSA.00087 **Special Availability and Scalability Requirements for Secret Storage Components**

Secret Storage components play a central role in storage, encryption, and digital signing in the Gaia-X ecosystem, thus they can become a single point of failure for a Gaia-X participant, for example an organization. Therefore, methods and procedures to ensure the availability and scalability of the Secret Storage functionality MUST be implemented. ◂◂

▸▸ IDM.TSA.00088 **Authorization Concept**

Access rights to policies and rules must follow the principle of least privilege. Based on this principle, an authorization concept must be developed. ◂◂

⏩ IDM.TSA.00089 **Policy data signing**
The policy data MUST be signed digitally to ensure data integrity. ⏪

⏩ IDM.TSA.00090 **Secure Timestamps**
All timestamps MUST be issued according to RFC 3161[8]. ⏪

⏩ IDM.TSA.00091 **Security W3C Vocab**
The product SHOULD implement functionalities to enable stable security algorithms according to [SecVoc] from the W3C in compliance to [SecOps]. Unstable algorithms MAY be considered. [URSA] provides open-source algorithms and MUST be considered for the product. ⏪

⏩ IDM.TSA.00092 **Trusted Computing**
It's RECOMMENDED to consider from the beginning, trusted computing principles with secure enclave concepts, to ensure that security technologies can be integrated in the future releases, for instance Intel SGX [Intel.SGX], AMD SEV [AMD.SEV-SNP] or other technologies. ⏪

3.3.8. Software Quality Attributes

⏩ IDM.TSA.00093 **Quality Aspects**
The software MUST meet the following requirements:

- The quality standards MUST meet ISO 25010 [ISO25000] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
- Robustness / Reliability
- Performance
- Availability must be 24/7
- Interoperability with the other work packages[9]
- Security
- Adaptability / expandability
- Maintainability and Code Quality
- Scalability

Major security concerns regarding design and implementation MUST be documented and highlighted to the steering board. Minor security concerns SHALL be documented and mitigated. ⏪

3.4. Compliance

⏩ IDM.TSA.00094 **GDPR Audit Logging**

[8] https://tools.ietf.org/html/rfc3161
[9] Please refer to appendix C for an overview and explanation of the Work Packages (WP).

All GDPR relevant access to personal relevant data MUST be logged for a later audit. ◂◂

▸▸ IDM.TSA.00095 **GDPR Data Processing**
If it is necessary to process person-relevant data, it MUST be earmarked to a clearly defined business process, which has to be described in the GDPR design decisions. All person relevant data MUST be deleted after the processing, if applicable. ◂◂

3.5. Design and Implementation

Please also refer to [TDR] for further requirements.

3.5.1. Distribution

▸▸ IDM.TSA.00096 **Config Data Distribution**
The product SHOULD support a global data distribution of config data to synchronize configurations between multiple regions in the world. Built-in synchronization technology (asynchronous and synchronous) MAY be used. ◂◂

3.5.2. Maintainability

▸▸ IDM.TSA.00097 **Micro Service Architecture**
For a better scale out, maintainability and decentralization, the product architecture MUST have a micro service architecture. Each microservice MUST NOT be limited on the lines of code or number of days to implement it. The service “size” SHOULD be oriented on the fine granular business capabilities. (e.g., Order, ListMenu, Payment). ◂◂

▸▸ IDM.TSA.00098 **Domain Driven Design**
To support the micro service architecture within the maintainability, it MUST be declared a domain model before realization. The software description MUST explain which domain model was chosen, which services contain it and how it scales. This MUST be documented in the public code repository to support future enhancements for new developers. ◂◂

3.5.3. Operability

▸▸ IDM.TSA.00099 **FTE Estimation**
The product MUST be designed so that over scripts and tools one FTE within a Month SHOULD host and operate the product without any third-party help. It MUST be sketched in the operations concept how this can be achieved. If this target is not reachable it MUST be explained and described why the effort is higher and appropriate. ◂◂

3.5.4. Interoperability

▸▸ IDM.TSA.00100 **Interoperability of IT security features and algorithms**

The following interoperability requirements of the respective IT security features and algorithms MUST be ensured across the system components:

- Interoperability of crypto algorithms and protocols (including the novel peer-reviewed ones through the established bodies and communities)
- Interoperability of secure secret transfer protocols (such as the holistic usage of PKCS#11 for HSM communication, etc.)
- Format interoperability of crypto material (such as the holistic usage of PKCS#12 for relevant cases) ◂◂

4. System Features

4.1. Policy Evaluation

4.1.1. Description

The policy evaluation provides functionality around the execution of policies. This includes the provisioning of versioned HTTP routes to execute the policy, synchronous and asynchronous policy evaluation, usage of external HTTP resources within the execution and a policy decision engine to create JSON response for a given JSON request and helper functionality around it. For instance, the caching of results or the inclusion of static JSON documents into the policy decisions, to evaluate more complex policies. All policies and static documents for the execution must be loaded from the encrypted hard disk into memory to guarantee the maximum execution speed. Temporary JSON documents can be stored into an open-source database for caching and SHOULD be deleted when no longer required. Any process or user behaves as an actor if the policy route was called.

4.1.2. Stimulus/Response Sequences

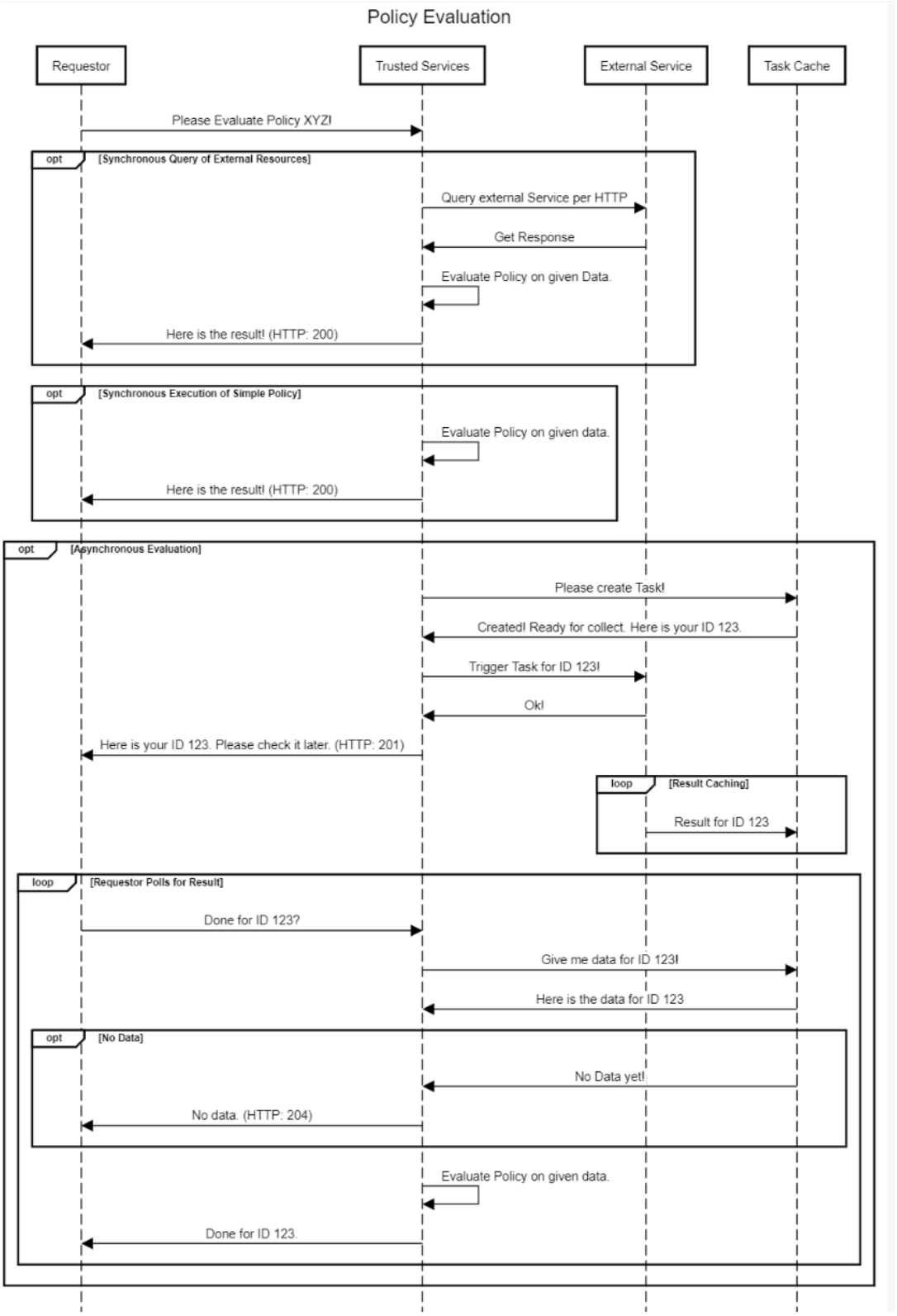


***Figure 4**: Policy Evaluation*

4.1.3. Functional Requirements

Functional Requirement
IDM.TSA.00034 Policy Execution
IDM.TSA.00035 Consistent Policy Evaluation Routes
IDM.TSA.00036 High Available Policy Evaluation Routes

***Table 6**: Functional Requirements Policy Evaluation*

4.2. Policy Management

4.2.1. Description

The policy management provides functionalities around the signing, validation, import, export, and merge of policies from internal and external policy repos. This is necessary to ensure that only trusted policies are imported from trusted resources. The feature must help and support the policy administrator in decisions to trust a policy repository from outside, trust imported policies, sign policies for export, export specific policies and merge changes into the productive repository. This feature can be automated by a continuous integration system (e.g., Jenkins), but it SHOULD include configurable manual reviewing steps to ensure that the signing of policies MUST be done by an authorized person.

The configuration from external policy repositories and the export of internal policies MUST be a manual step.

4.2.2. Stimulus/Response Sequences

Not applicable.

4.2.3. Functional Requirements

Functional Requirement
IDM.TSA.00021 Update policy git flow
IDM.TSA.00022 Authentication for Policy Administration
IDM.TSA.00023 UX Behavior for Policy Administration
IDM.TSA.00024 Policy versioning
IDM.TSA.00025 Policy Bundles
IDM.TSA.00026 Policy Creation and Structure

IDM.TSA.00027 Policy Actions
IDM.TSA.00028 Policy Import and Export

***Table 7**: Functional Requirements Policy Management*

4.3. Git Ops

4.3.1. Description

The feature provides a GIT server which is only reachable over a protected connection and configured to sign checked in data. The server must support file encryption. The server MUST only be reachable internally but MUST be configurable by the administrator to enable the access for employees from outside via secure remote access. Administration of policies MUST follow the “policy as code” and “branch model” principles by Git. The feature MUST allow the download of repositories. Policies MUST be assigned to according groups and signed before check-ins[10].

4.3.2. Stimulus/Response Sequences

Not applicable.

4.3.3. Functional Requirements

Functional Requirement
IDM.TSA.00020 Secured Git Server
IDM.TSA.00021 Update policy git flow
IDM.TSA.00022 Authentication for Policy Administration

***Table 8**: Functional Requirements Git Ops*

4.4. Task Controller

4.4.1. Description

The task controller feature provides an API which is able to handle asynchronous task lists. Each task represents one single action which executes an HTTP URL. Each task has a unique id and stores its result in the distributable cache for a later processing. Task lists can be preconfigured in the repository by a name, to create a new task subset more easily from a policy or any other component (e.g., a 1 to many task mapping). The task execution is asynchronous, and the result can be queried over the task(list) id, to query the state of the processing.

[10] https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work

4.4.2. Stimulus/Response Sequences

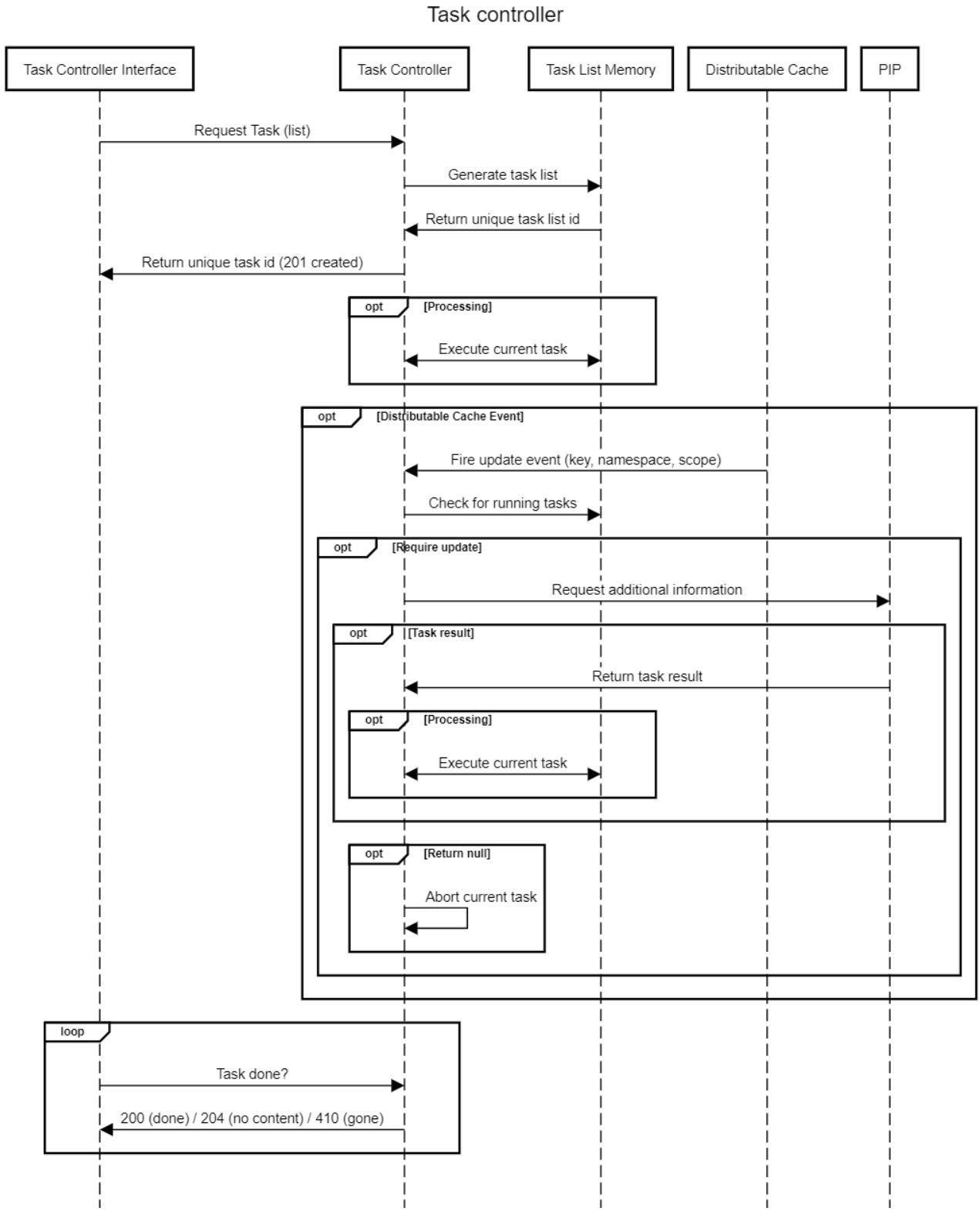


***Figure 5**: Policy Task Coordination*

4.4.3. Functional Requirements

Functional Requirement
IDM.TSA.00038 Task Definition
IDM.TSA.00039 Task List Definition
IDM.TSA.00040 Task(list) Instantiation and…
IDM.TSA.00041 Instantiation/Execution of Tasks
IDM.TSA.00042 Instantiation/Execution of Task Lists
IDM.TSA.00043 Sequential processing
IDM.TSA.00044 Parallel processing
IDM.TSA.00045 Cache Event Subscription

***Table 9**: Functional Requirements Task Controller*

4.5. Trust Chain Verification

4.5.1. Description

To ensure trust within multiple participants it is necessary to validate trust chains that could have multiple asynchronous verifications. Such behavior requires the need for long-running operations in the policy decision engine. Necessary information SHOULD be cached to improve the execution time and MUST be deleted after a reasonable time.

4.5.2. Stimulus/Response Sequences

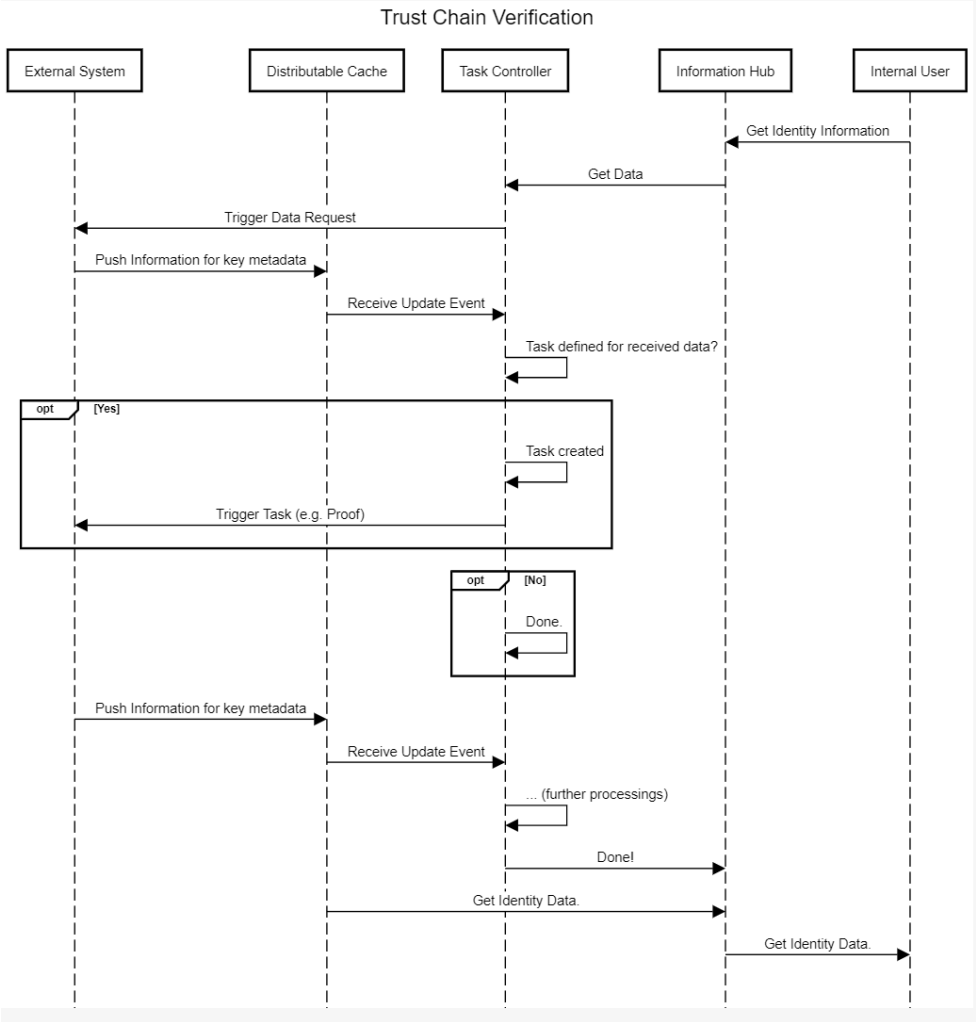


***Figure 6**: Trust Chain Verification*

4.5.3. Functional Requirements

Functional Requirement
IDM.TSA.00041 Instantiation/Execution of Tasks
IDM.TSA.00042 Instantiation/Execution of Task Lists
IDM.TSA.00043 Sequential processing
IDM.TSA.00045 Cache Event Subscription

***Table 10**: Functional Requirements Trust Chain Verification*

4.6. JSON-LD Signing and verification

4.6.1. Description

The feature provides verification and signature functionality of LD-Proofs embedded in JSON-LD files. The functionality is an internal HTTP API, but the core crypto functionality has to be provided as a separate library and SHOULD run within a secure environment.

4.6.2. Stimulus/Response Sequences

An input JSON-LD file.

4.6.3. Functional Requirements

Functional Requirement
IDM.TSA.00047 LD Proofs in JSON-LD
IDM.TSA.00048 Proof Sets in JSON-LD
IDM.TSA.00049 Proof Chains in JSON-LD

***Table 11**: Functional Requirements JSON-LD Signing and verification*

4.7. eIDAS compliant signatures

4.7.1. Description

To provide eIDAS compliant signatures the feature should be able to generate and validate eIDAS compliant signatures. In consideration of the different eIDAS types, legal signatures should be considered and a bridge functionality to sign the data should be implemented. A secure environment MUST be provided to store and execute the necessary functions (signature, validation) and SHOULD require at least two factor authentication.

4.7.2. Stimulus/Response Sequences

A signing or verification request of a JSON-LD structure.

4.7.3. Functional Requirements

Functional Requirement
IDM.TSA.00064 eIDAS compliant Signature Creation / Validation

***Table 12**: Functional Requirements eIDAS compliant signatures*

4.8. DID Document Resolving

4.8.1. Description

The did resolving feature provides capabilities to resolve a did document for different did methods and trusted DID document reading. For this feature we RECOMMEND the Universal DID resolver specified by DIF (2020), Universal Resolver.

4.8.2. Stimulus/Response Sequences

An incoming DID resolve request over a secure connection.

4.8.3. Functional Requirements

Functional Requirement
IDM.TSA.00029 DID Document Resolving
IDM.TSA.00030 DID Resolver HTTP Interface

***Table 13**: Functional Requirements DID Document Resolving*

4.9. Trusted Caching

4.9.1. Description

The trusted caching provides the functionality to store securely in memory data for identities and related information for trust evaluation.

4.9.2. Stimulus/Response Sequences

Request from outside or internal trigger.

4.9.3. Functional Requirements

Functional Requirement
IDM.TSA.00054 In Memory Caching
IDM.TSA.00055 Cache Distribution
IDM.TSA.00056 Internal Cache Interface
IDM.TSA.00057 Key Metadata
IDM.TSA.00058 Content Format

IDM.TSA.00059 Content Access

***Table 14**: Functional Requirements Trusted Caching*

4.10. Trusted Information Exchange

4.10.1. Description

To exchange trusted information a few functions are required. This covers the secure import and export of such trusted information data as well as the content.

4.10.2. Stimulus/Response Sequences

Request from outside or internal trigger.

4.10.3. Functional Requirements

Functional Requirement
IDM.TSA.00050 Trusted information export
IDM.TSA.00051 Trusted information import
IDM.TSA.00052 Content Modifications
IDM.TSA.00053 Trusted Identity Information

***Table 15**: Functional Requirements Trusted Information Exchange*

5. Other Requirements

Additionally, to the product functions, the product MUST contain the first policies as basis for the behavior of the system. The following policies have to be implemented within the rego policy language [Rego]. All inputs and outputs MUST be in JSON format. Necessary policies are mentioned in Appendix B: Policy List and rely on the functionalities mentioned in this chapter.

IDM.TSA.00120 **JWT Decode Policy**

The policy MUST be able to decode and verify a JWT to get the encoded contents for policy evaluations.

Input
A JWT Token.

Output

The decoded JWT Token and verification result.

Acceptance Criteria

1) Policy is able to decode the received JWT generated according to RFC7519
2) JWT signature MUST be verified
3) Decoded JWT content.

IDM.TSA.00121 **JWT Encode Policy**

The policy MUST be able to encode content to get valid and signed JWT token.

Input
Any content.

Output
The encoded and signed JWT Token.

Acceptance Criteria

1) Policy is able to encode content to a valid JWT according to RFC7519
2) JWT signature MUST be verifiable against public key
3) Valid JWT

IDM.TSA.00122 **Base64 Encode/Decode**

The policy MUST be able to encode content to BASE64 and decode content to raw string values.

Input
Any string content/base64 content

Output
The encoded content or the decoded content.

Acceptance Criteria

1) Policy is able to encode content to base64
2) Policy is able to decode base64 content

IDM.TSA.00123 **Hash Policy**

The policy MUST be able to hash content with SHA256/keccak-256 or better.

Input
Any content.

Output
A hash values.

Acceptance Criteria

1) Policy is able to generate SHA256/keccak-256 hashes over the content
2) Hash value must be valid

IDM.TSA.00124 **JSON Content Compare Policy**

This policy MUST evaluate the content of JSON data and verify whether the request is viable or not by a Boolean comparison of the fields. The comparison MUST demonstrate the standard Boolean compares like NOT,EQUAL, GREATER THAN, LESS THAN etc. for integer, timestamps, Boolean and string values.

Input

Field	**Type**	**Example**
updated_at	date	2020-10-09 04:44:47Z
their_label	string	Bob
invitation	JSON OBJECT	{ "accept": "auto", "alias": "Bob, providing quotes", "connection_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "created_at": "2020-11-09 04:44:47Z", "error_msg": "No DIDDoc provided; cannot connect to public DID", "inbound_connection_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "initiator": "self",

		"invitation_key": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV", "invitation_mode": "once", "my_did": "WgWxqztrNooG92RXvxSTWv", "request_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "routing_state": "active", "state": "active", "their_did": "WgWxqztrNooG92RXvxSTWv", "their_label": "Bob", "their_role": "Point of contact", "updated_at": "2020-11-09 04:44:47Z" }

Output
A list of compared values for each comparison type.

Acceptance Criteria

1) The policy is able to return if their_label is equal to “Bob”
2) The policy is able to return if updated_at is less than the given value

IDM.TSA.00125 **JSON Content Policy**

The policy MUST return a complex JSON content with the possibility of field value replacements inside of an JSON structure by a given input.

Input
“comment” = “Hello World”
“version” = “2.0”

Output

Field	**Type**	**Example**

proof	JSON Object	{
		"proof": {
		"comment": "**Hello World**",
		"proof_request": {
		"name": "GXEmployeeProof",

		"non_revoked": {
		"from": 346364,
		"to": 363636
		},
		"nonce": "1234567890",
		"requested_attributes": {
		"additionalProp1": {
		"name": "**Hello World**",
		},
		"restrictions": [
		{
		"schema_id": "WgWxqztrNooG92RXvxSTWv:2:employeeschema:1.0",
		"schema_issuer_did": "WgWxqztrNooG92RXvxSTWv",
		"schema_name": "**Hello World**",
		"schema_version": "**2.0**",
		}
		]
		
		}
		}
		}
		}}

Acceptance Criteria

1) The JSON document must be filled with the input values

IDM.TSA.00126 **Conditional statement policy**

This policy MUST be able to distinguish between different conditions. Hereby the output will differ depending on the input into the policy.

Input

"income":"15000",

"expense":"17000"

Output

valid_expense = "allow" {

```
   input.income == "15000" # allow

} else = "deny" {

   input.expense == "17000" # disallow

}
```

Acceptance Criteria

1) “Allow” Output, if income is 15000
2) “Deny” is true Output if income is not equal 15000 and expense is 17000

IDM.TSA.00127 **X.509 certificate policy**

The policy MUST be able to parse certificates to get the content.

Input

cert =

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

MIIF2zCCBM….

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

Output

Must be an array of X.509 certificates represented as JSON objects.

Acceptance Criteria

1) The policy must be able to decode PEM strings

IDM.TSA.00128 **External HTTP Request Policy**

The policy MUST be able to call external HTTP URLs. The execution MUST be feasible for single and multiple calls within one policy. The returned content MUST be processable with standard functionalities of the policy system.

Input
Any content.

Output
HTML content from any website.
HTML content encoded as base64

Acceptance Criteria

1) Policy is able get the content from another website and returns over the policy output.
2) The content from the returned website is encoded as base64

IDM.TSA.00129 **Regex Policy**

The policy MUST be able to interpret regexes.

Input
A single string.

Output
A Boolean value which indicates if the content contains lowercase signs.

Acceptance Criteria

1) Policy prints the correct result for A_b_C123
2) Policy Prints the correct result for ABC123

IDM.TSA.00130 **Distributable Cache Policy**

This policy demonstrates how the distributable cache interaction works.

Input
Key Metadata.

Output
The JSON result of the cache.

Acceptance Criteria

1) The policy returns content to a key metadata

IDM.TSA.00131 **Task Controller Policy**

The policy demonstrates how the task controller integration works. If the policy is called, the policy MUST return a task id. By calling the same policy with the task id, it must return a JSON content or a response code which indicates that the task is not yet done.

Input
Key Metadata.

Output
The JSON result of the cache.

Acceptance Criteria

1) The policy returns a task id from the task controller
2) The task controller triggers the task
3) The policy returns a pending state during the pending task
4) The policy returns a result to the task, if the task is done

IDM.TSA.00132 **Interaction Policies**

The policies are used to support other trust components and processes; therefore, the product MUST implement the basic policies described in the Appendix B. The functionality of these policies MUST be aligned with the different component vendors to finalize the input, output values and the exact behavior.

6. Verification

IDM.TSA.00133 **Behavior Driven Design**
Verification of fulfillment of the requirements and characteristics MUST be done using automated tests which are part of the deliverables. They SHOULD be done by patterns of the Behavior Driven Development (BDD) using the “Gherkin Syntax”.

IDM.TSA.00134 **Automated Test Environment**
All functionalities MUST be demonstrated in a complexer test environment within a sandbox, with the following infrastructure components:

- Load Balancer, e.g., HAProxy
- API Gateway, e.g., Kong
- Service Mesh, e.g., Linkerd/Istio
- DNS
- Multiple Servers
- Firewalls

All security tests MUST be passed in this test environment automatically. ◂◂

▸▸ IDM.TSA.00135 **Load Tests**
Scalability and Performance around the high workload scenarios MUST be demonstrated, by using any kind of Load Test Framework for HTTP APIs. e.g., Gatling[11]. ◂◂

[11] https://gatling.io/

Appendix A: Glossary

For the glossary refer to IDM.AO Glossary/Terminology [IDM.AO].

Appendix B: Policy list

Due to the complexity of the GXFS environment, a couple of policies are described below. These will be mentioned in the other IDM documents as well, especially in the OCM and AA documents.

To get a brief understanding of the necessary functionalities, these policies are mentioned here. They rely HEAVILY on the mentioned features in 5. Other Requirements.

Policy	**Description**	**Required by component**
PrincipalCredentialRequest	The policy MUST be able to decide whether an employee is allowed to receive a Gaia-X principal credential	IDM.OCM **Principal Manager Trust Service Interaction**
GetTrustedConnectionState	The policy MUST be able to decide whether a connection request from another OCM is accepted and build the necessary proof request.	IDM.OCM **Get Trusted Connection State Policy**
TrustedConnectionCredentials	This policy MUST decide whether a connection proof-request should be responded to.	IDM.OCM **Presentation Request to establish a Trusted Connection**
TrustedConnectionUpdate	This policy is being evaluated, once a connection trust state between two OCMs has been updated, e.g., when initiating a new connection or updating an existing one. It MUST validate the verifiable presentation and check whether it was valid or not.	IDM.OCM **Presentation Received for Trusted Connection**

CredentialIssueRequest	This policy MUST evaluate whether a credential should be issued or not. It will receive the credential schema id, the credential definition, and the necessary data.	IDM.OCM **Credential-Issue Request Received**
ProofRequestResponse	This policy MUST decide whether a proof request may be responded to.	IDM.OCM **Presentation Request Received**
PresentationFreshnessState	This policy validates if a valid verifiable presentation needs to be updated, because the presentation lifespan is running out soon. The output MAY be cached to avoid unnecessary calls.	IDM.OCM **Check Received Presentations**
PresentationRevokationState	This policy defines if a valid verifiable presentation needs to be checked for revocations. The output MAY be cached to avoid unnecessary calls	IDM.OCM **Check Received Presentations for Revocation State**
CredentialFreshnessState	This policy defines if a valid verifiable credential needs to be refreshed, because the credential lifespan is running out soon. The output MAY be cached to avoid unnecessary calls	IDM.OCM **Check Credential State**
ProofOfNonRevocation	This policy MUST decide whether an issued verifiable presentation shall be automatically renewed as soon as a request is received from the recipient	IDM.OCM **Proof of non-Revocation Received**

Participant Proof	Policy which provides a proof in JSON format for required GX employee attributes. E.g., Name, Email Address and issuer of the credential	IDM.OCM **Principal Manager Trust Service Interaction**
Principal Proof Policy	Policy which provides an Indy proof in JSON format for required participants GX membership attributes. E.g., ParticipantName, Onboarding Date etc.	IDM.OCM **Principal Manager Trust Service Interaction**
GetTrustedConnectionState	Policy which provides an Indy proof in JSON format for required credentials and schemas to decide whether a new connection will be accepted. This MAY include a proof-request for the Gaia-X participant credential and Gaia-X organizational credential.	IDM.OCM **Get Trusted Connection State Policy**
PublicProfileCredential	This policy MUST fulfill requests by the OCM to return if credentials are public SDs or not.	IDM.OCM **Self-Description Content**
GetLoginProofInvitation	This policy MUST respond to the AA request containing the scope and namespace for the authorization request. It returns the according presentationID and link from the OCM.	IDM.AA **QR Code Generation** IDM.AA.00001 **Session handling and scope elevation**
GetLoginProofResult	This policy MUST provide a result to the AA request initiated with GetLoginProofInvitation provided presentationID.	IDM.AA.00002 **Login State Background Polling** IDM.AA.00002 **Session handling and scope elevation**

	The result shall be a flattened list of claims related to the requested scopes of identity content (see IDM.TSA.00061 Trusted Identity Information**)**	
GetIatProofInvitation	This policy MUST evaluate whether a client is allowed to obtain an Initial Access Token. It will request an appropriate invitation from the corresponding OCM.	IDM.AA **Policy based authorization**
GetIatProofResult	This policy endpoint MUST return the IAT Proof Invitation result to the AA based on the according policy.	IDM.AA **Policy based authorization**

Appendix C: Overview GXFS Work Packages

The project “Gaia-X Federation Services” (GXFS) is an initiative funded by the German Federal Ministry of Economic Affairs and Energy (BMWi) to develop the first set of Gaia-X Federation Services, which form the technical basis for the operational implementation of Gaia-X.

The project is structured in five Working Groups, focusing on different functional areas as follows:

Work Package 1 (WP1): Identity & Trust
Identity &Trust covers authentication and authorization, credential management, decentral Identity management as well as the verification of analogue credentials.

Work Package 2 (WP2): Federated Catalogue
The Federated Catalogue constitutes the central repository for Gaia-X Self-Descriptions to enable the discovery and selection of Providers and their Service Offerings. The Self-Description as expression of properties and Claims of Participants and Assets represents a key element for transparency and trust in Gaia-X.

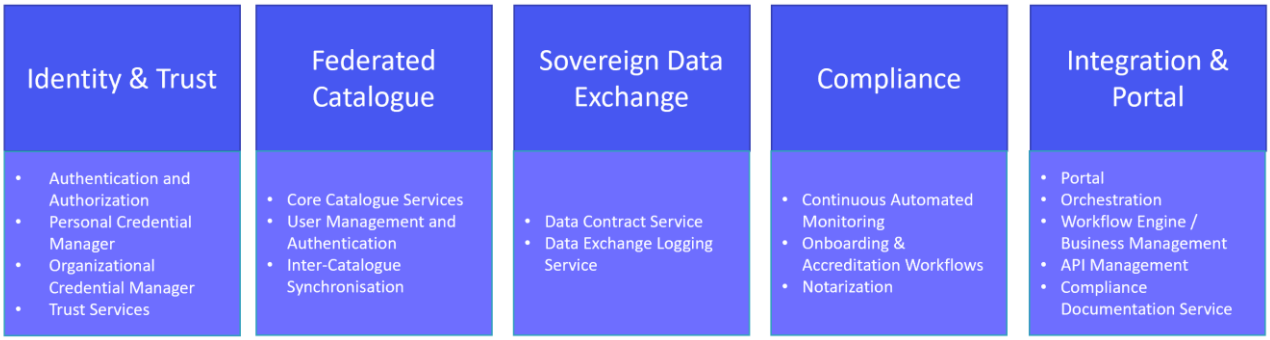
Work Package 3 (WP3): Sovereign Data Exchange
Data Sovereignty Services enable the sovereign data exchange of Participants by providing a Data Agreement Service and a Data Logging Service to enable the enforcement of Policies. Further, usage constraints for data exchange can be expressed by Provider Policies as part of the Self-Description

Work Package 4 (WP4): Compliance

Compliance includes mechanisms to ensure a Participant's adherence to the Policy Rules in areas such as security, privacy transparency and interoperability during onboarding and service delivery.

Work Package 5 (WP5): Portal & Integration
Gaia-X Portals and API will support onboarding and Accreditation of Participants, demonstrate service discovery, orchestration and provisioning of sample services.

All together the deliverables of the first GXFS project phase are specifications for 17 lots, that are being awarded in EU-wide tenders:



Further general information on the Federation Services can be found in [TAD].