

Software Requirements Specification

for

Federation Architecture for Composed Infrastructure Services (FACIS) Zero Trust Demonstrator

FACIS.ZTD

Table of Contents

1	Introduction	1
1.1	Document Purpose	1
1.2	Conformance Language	1
1.3	Product Scope	1
1.4	Definitions, Acronyms and Abbreviations	2
1.5	References	2
2	Product Overview	3
2.1	Product Perspective	3
2.2	Product Functions	5
2.3	Product Constraints	6
2.3.1	Component Dependencies	6
2.3.2	License	6
2.3.3	Technology Selection	6
2.4	Operating Environment	7
2.4.1	General	7
2.4.2	Requirements	8
2.4.3	Connectivity Target Architecture	9
2.5	User Documentation	12
2.6	Assumptions and Dependencies	13
2.6.1	Attestation Code Basis	13
2.6.2	Visualization	13
2.7	Apportioning of Requirements	13
3	Requirements	14
3.1	Functional Requirements	14
3.1.1	Zero Trust Connector	14
3.1.2	Zero-Trust Service Mesh	15
3.1.3	Inter-Cluster Communication Protocol	16
3.1.4	Zero Trust Execution Environment	19
3.1.5	Zero Trust Application Communication	20
3.1.6	Zero Trust Communication Visualization	21
3.2	Non-Functional Requirements	22
3.3	General Security Requirements	23
4	Design and Implementation	25
5	System Features	26
5.1	Zero Trust-based Service Mesh	26
5.1.1	Description	26
5.1.2	Architecture	26
5.1.3	Definition of Done	27
5.2	Bidirectional Remote Attestation	28
5.2.1	Description	28
5.2.2	Architecture	28
5.2.3	Definition of Done	29
5.3	Zero Trust Execution Environment	30
5.3.1	Description	30
5.3.2	Architecture	30
5.3.3	Definition of Done	30
5.4	Zero Trust Application Communication	31
5.4.1	Description	31
5.4.2	Architecture	31
5.4.3	Definition of Done	32
5.5	Zero Trust Communication Visualization	32
5.5.1	Description	32

5.5.2	Architecture	32
5.5.3	Definition of Done	33
6	Appendix.....	34

1 Introduction

1.1 Document Purpose

The purpose of this document is to specify the product perspective, functions, and constraints of the FACIS Zero Trust (ZT) Demonstrator in preparation for a Europe-wide public tender for its implementation. It also defines the system features and specifies the functional and non-functional requirements of the product. The primary audience of this document is prospective bidders participating in the public tender who can provide a software solution to be released as open-source software under the Apache License 2.0.

1.2 Conformance Language

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in [RFC 2119] and are written in capital letters.

1.3 Product Scope

In Scope

A software solution which can be installed within a Kubernetes cloud environment MUST be provided. The solution MUST be provided with all REQUIRED deployment artifacts and other REQUIRED software components and operational environment setups:

- Helm Charts
- Docker images
- CI/CD definitions on GitHub and ArgoCD
- Usage and integration of existing Eclipse XFSC¹ components, e.g., OCM W-Stack and PCM Cloud
- Software repositories within Eclipse XFSC GitHub repository
- Documentation for deployment instructions, user manual, and operation instructions
- Provision of BDD tests and test reports.

In the scope is a final acceptance presentation of the fully operational software solution on the two mentioned cloud environments.

Out of Scope

Out of scope is the provision of cloud environments and network connectivity.

¹ <https://github.com/eclipse-xfsc>

1.4 Definitions, Acronyms and Abbreviations

Table 1 – List of Acronyms and Abbreviations

Acronym or Abbreviation	Term	Definition
ABAC	Attribute-Based Access Control	An access control model that grants or denies access to resources based on attributes of the user, resource, action, and environment.
BDD	Behavior Driven Development	A design methodic to ensure that a system fulfills requirements on the level of “business requirements”, often implemented in Gherkin syntax in combination with Cucumber.
CrBAC	Credential Based Access Control	Credential based access control is a special version of attribute-based access control, where all attributes are derived from signed credentials.
DE-CIX	Deutscher Commercial Internet Exchange	A global operator of Internet Exchange Points (IXPs) that enables networks to directly exchange internet traffic.
ORCE	Orchestration Engine	XFSC orchestration engine based on Node-RED.
TEE	Trusted Execution Environment	Hardware-isolated secure area of a processor that protects sensitive code and data from unauthorized access or tampering, even if the main system is compromised.
TEEM	TEE Measurement	The launch digest of a trusted execution environment.
TRAIN	Trust Management Infrastructure	An XFSC component that enables Gaia-X participants to establish and verify trust using decentralized trust lists and DNS anchoring.
TRAIN TCR	Trust Management Infrastructure Trust Content Resolver	TRAIN feature is responsible for the Trust Discovery and Trust Validation functionalities based on the input from the Verifiable Credential / Verifiable Presentation.

1.5 References

Table 2 – List of References

Reference ID	Description	Link
[SHACL]	W3C Shapes Constraint Language	https://w3.org/TR/shacl/
[SPARQL]	W3C SPARQL 1.2 Query Language	https://w3.org/TR/sparql12-query/
[VC]	W3C Verifiable Credentials Data Model v2.0	https://w3.org/TR/vc-data-model-2.0/
[RFC2119]	Key words for use in RFCs (MUST/SHOULD/MAY)	https://rfc-editor.org/rfc/rfc2119
[NIST.800-207]	NIST SP 800-207 Zero Trust Architecture	https://doi.org/10.6028/NIST.SP.800-207

[BSI.TR-02102]	BSI TR-02102 Cryptographic Mechanisms	https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html
[DO-326A]	RTCA DO-326A / EUROCAE ED-202A Airworthiness Security Process Specification	https://rtca.org/security/
[RFC8915]	RFC 8915 Network Time Security (NTS)	https://rfc-editor.org/rfc/rfc8915

2 Product Overview

2.1 Product Perspective

The FACIS Zero Trust (ZT) Demonstrator SHALL showcase the interaction between two trust zones as individual federations, which share digital services between entitled participants and principals, based on policy rules. These policy rules define the trust relationship between these two zones, by building a group of loosely coupled actors which interact to each other. To realize this, the Demonstrator SHALL provide a two-party setup/ZT capabilities for establishing trusted connections between for accessing protected resources via a participant backends. Those capabilities can be collected within a ZT deployment which can communicate to other instances of itself for simplifying the ZT process by Trust Framework Manager from communication perspective. This connector SHALL later run within Kubernetes-based ZT service meshes on T-Systems Open Sovereign Cloud (OSC) and IONOS Cloud. TRAIN SHALL complete this by providing all relevant trust information to enable the final Demonstrator to orchestrate a Zero Trust visualization between the two environments. In summary, the ZT Demonstrator can be illustrated as a minimal viable federation of two participants and a ZT governance with TRAIN as federation technology.

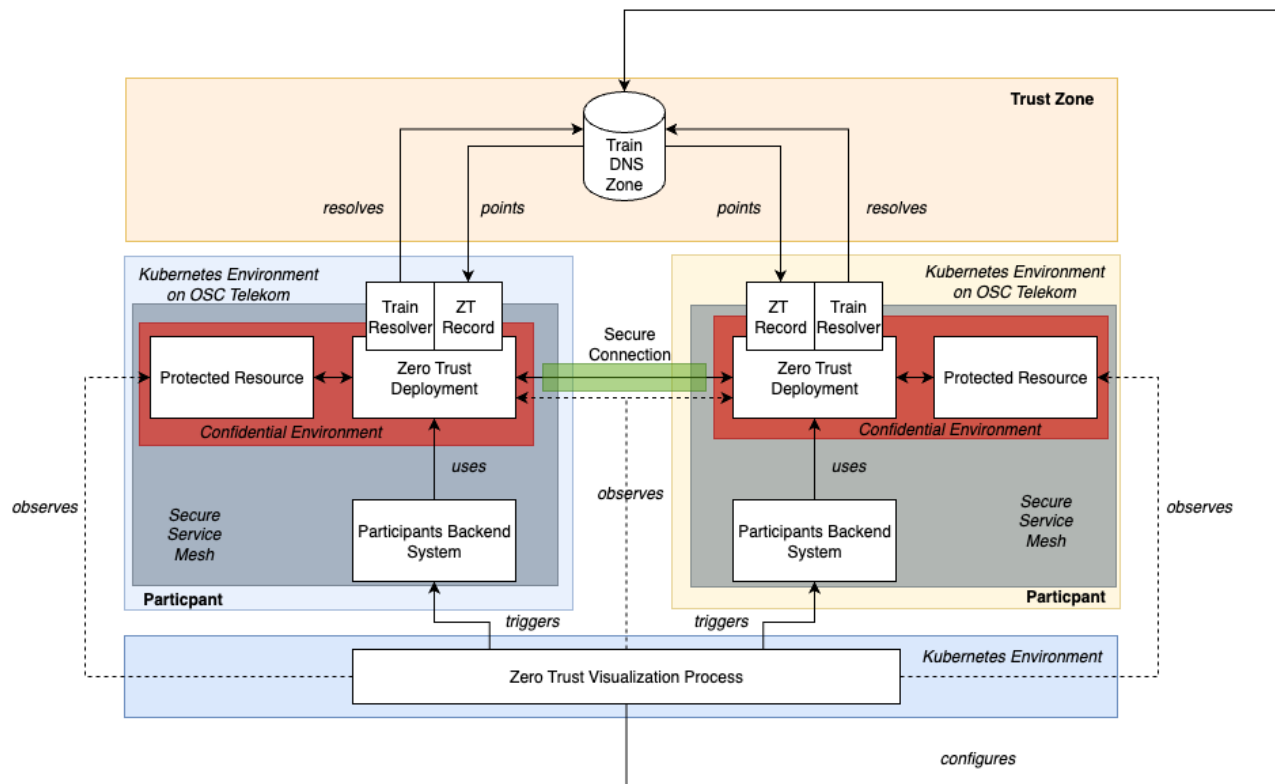


Figure 1 Zero Trust Demonstrator Overview

As shown in Figure 1, ZT deployment has a central role within this architecture. In fact, the deployment has a lot of tasks which can be split into the following areas:

- Secure Connection Establishment,
- Communication Guarding,
- Authorized Access Handling for Remote Protected Resources.

Each of those areas SHALL be realized via a set of components and procedures to simplify the usage of ZT principles by providing a Kubernetes deployment. This includes:

- Deployment scripts,
- Signing and attestation procedures,
- Component selection and configuration,
- Adoption pattern for existing backends.

This setup can be visualized as shown in Figure 2.

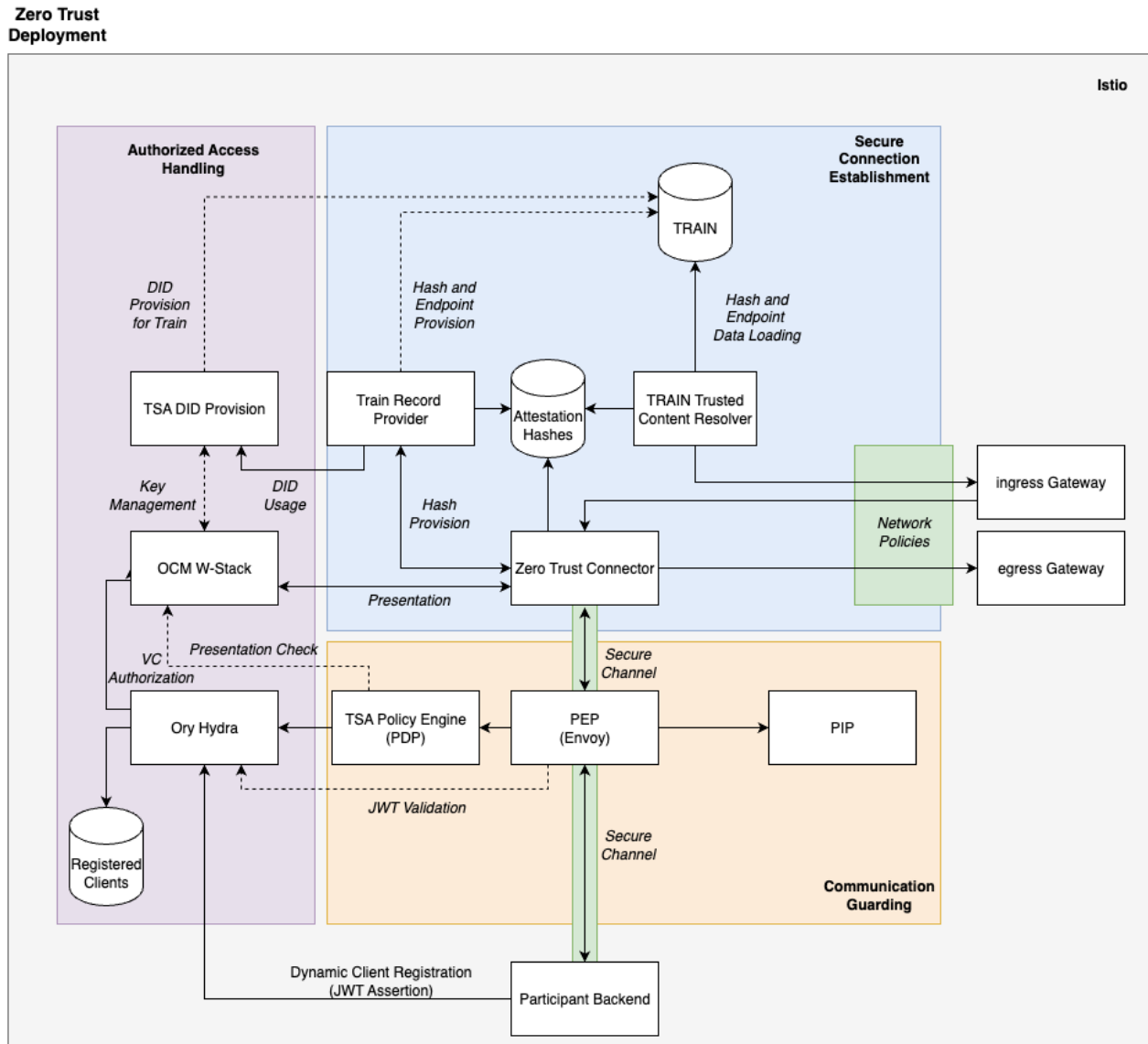


Figure 2 Zero Trust Setup

The entire setup SHALL be delivered and developed as out-of-the-box solution with a visualization part to demonstrate the working solution to an external audience.

2.2 Product Functions

The Zero Trust Demonstrator MUST provide the following core capabilities:

Table 3 – ZT Demonstrator Core Functions

No.	Name	Description
ZT-01	Secure Service Mesh	The secure service mesh SHALL protect communication between all services running for the Demonstrator.

ZT-02	Secure Execution Environment	For execution, the product MUST have a protected space within the cluster where the critical components can be securely executed with, signed images and others.
ZT-03	Trust Zone	The product SHALL contain a trust zone which features both clusters as “participant”.
ZT-04	Secure Connection	Between the ZT Connectors MUST be a secure connection.
ZT-05	Communication Visualization	For a better understanding, the product MUST visualize ZT communication via UX, e.g., with animations, etc.
ZT-06	Attribute/Credential Based Access Control	ABAC SHALL be applied for the inner communication channel from backend to protected resource,
ZT-07	API Dock	The Demonstrator MUST be able to dock APIs as protected resource and as participant backend.
ZT-08	Visualization	The Demonstrator MUST have an appropriate visualization of ZT functionality and the user journey behind it.
ZT-09	User Journey	The Demonstrator implements a user journey which is orchestrated by ORCE ² , and which illustrates how the entire ZT system works.

2.3 Product Constraints

2.3.1 Component Dependencies

The product requires the use of open-source components which are licensed under an Apache 2.0-compatible license.

2.3.2 License

All newly developed components of the Zero Trust Demonstrator MUST be Apache 2.0-licensed. Used components MUST have an Eclipse-compatible licensing, e.g., Apache 2.0.

2.3.3 Technology Selection

This subsection defines which core technologies SHOULD be used in development. If there is any abbreviation, e.g., additional or different components required, the abbreviation MUST be declared and aligned with the client.

Table 4 – Technology Selection

Technology	Constraint
Istio	For a secure service mesh, Istio SHOULD be the first choice. Ideally, it SHOULD be applied in ambient mode to reuse the ZT capabilities of Cilium. However, when this is not possible, it MUST be declared how this can be realized with other technologies.

² <https://github.com/eclipse-xfsc/orchestration-engine>

	Remark: Running Istio Ambient Mode with Cilium as CNI requires mandatory configuration steps: (1) Cilium MUST be configured with cni.exclusive=false, (2) L7 policies MUST NOT be enforced by both simultaneously to avoid a split-brain problem.
OCM W-Stack	OCM W-Stack SHALL be used for CrBAC, credential issuing and credential verification.
PCM Cloud	PCM Cloud MAY be used for issuing credentials and as part of the visualization.
ORCE	ORCE MUST be used for orchestration purposes
Ory Hydra	Ory Hydra offers a headless authorization flow which MAY be extended via Token Hooks to append claims from a presented credential. If other components fit better, Ory Hydra MAY be replaced.
Envoy	A proxy implementation is required for the ZT Connector. This MAY require modification/extension of existing components. Envoy in combination with external processors SHOULD be the first choice. If there is any other open-source choice, this MAY be used so long as it fulfills the goal of communication.

2.4 Operating Environment

2.4.1 General

The solution MUST be demonstrated in two cloud environments based on T-Systems OSC and IONOS Cloud to simulate Zero Trust capabilities. Although these environments are provided, the solutions MUST be installed and operated there. Moreover, the operating environment in both cloud providers MUST have the structure as shown in Figure 3.

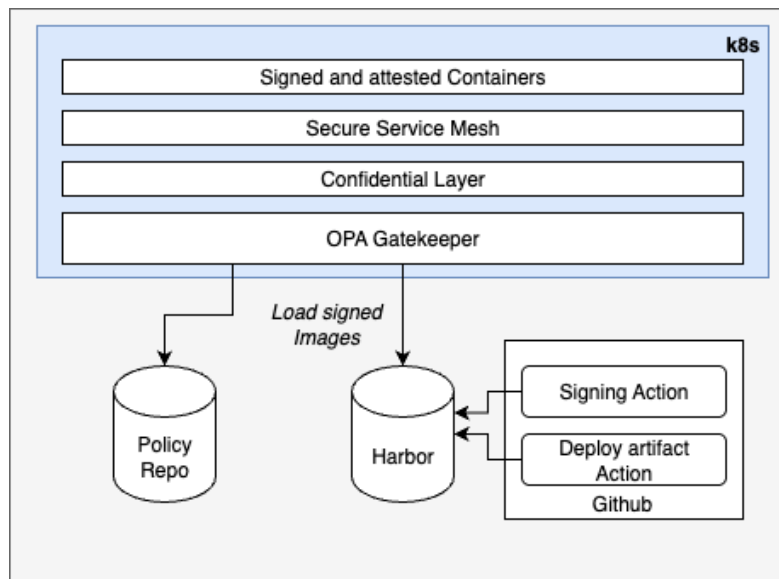


Figure 3 Basic Infrastructure

2.4.2 Requirements

[ZT-10] Image Signing and Attestation

Priority: MUST

Description: All images are currently built and deployed by GitHub Actions³ without signing. For a ZT environment, all used components require a signing for providing it properly to OPA Gatekeeper. This requires a workflow which MUST provide a signing for Docker images by using Cosign⁴ and an attestation flow which is valid for a mock TEE. The flow for the signing and for the attestation MUST be realized via GitHub Actions, but it MUST be ensured that the flows can be triggered by authorized maintainers and committers only. The flows MUST run on a custom GitHub runner which is prepared with key material, Cosign and other attestation tools. If there are other attestation pipelines available, they MAY be used after the client has approved it.

Acceptance Criteria:

- Custom GitHub runner up and running,
- Key material can be uploaded on the runner,
- Runner setup AMD64-based systems.

[ZT-11] OPA Gatekeeper

Priority: MUST

Description: The environment MUST start images based on OPA Gatekeeper policies only. In this case, a policy checks if the signature of the images is signed by the client. For this purpose, all relevant key material MUST be distributed via did:web over TSA and linked over TRAIN. The key material MUST be stored in ObenBao as key value entry (if a X509 is used). Otherwise, the keys MUST be used over the transit engine and TSA crypto provider service.

Acceptance Criteria:

- All Docker images signed by the client,
- Gatekeeper blocks unsigned images during pod creation,
- Trust Anchor is provided via did:web,
- Other Governance Rego Policies can be executed over Gatekeeper, e.g., allowed registries, naming conventions or similar.

[ZT-12] OPA Policy Repository

Priority: MUST

Description: The OPA Gatekeeper MUST pull its policies from Harbor over OCI. The policy packages MUST be built via GitHub Actions from a private repository to which authorized maintainers and committers have access.

Acceptance Criteria:

- Policy artifacts can be built and uploaded to Harbor,
- Gatekeeper loads and uses the policies from Harbor.

³ <https://github.com/eclipse-xfsc/dev-ops/blob/main/.github/workflows/dockerbuild.yml>

⁴ <https://github.com/sigstore/cosign>

[ZT-13] Linux-based Docker Images

Priority: MUST

Description: The solution MUST be developed for Linux-based operation environments and all Docker images MUST be Linux-based.

Acceptance Criteria:

- All Docker images are Linux-based,
- All Docker images up and running.

[ZT-14] T-Systems Open Sovereign Cloud Environment

Priority: MUST

Description: The solution MUST be fully operational on the T-Systems OSC⁵ environment

Acceptance Criteria:

- Cluster and the solution are reachable.
- All ZT components deployed and in Running state,
- OPA Gatekeeper demonstrably blocking unsigned images,
- End-to-end connection between ZT Connector and protected resource verified.

[ZT-15] IONOS Cloud Environment

Priority: MUST

Description: The visualization orchestration for the Demonstrator MUST be operated on the IONOS Cloud.

Acceptance Criteria:

Visualization of the Demonstrator runs on the IONOS Cloud.

2.4.3 Connectivity Target Architecture

Within the FACIS Zero Trust framework, two Kubernetes clusters are connected through a so-called Closed User Group (CUG). A CUG is a private, closed network that is exclusively available to defined and authorized participants. The objective is to establish a secure, stable, and controlled connection between two trust zones (federations) without using the public Internet.

This CUG communication concept is independent of all the other zero trust implementations and adds an additional layer of security for IP communication.

2.4.3.1 Technical Principle of the CUG

The CUG is based on two main technical components:

⁵ <https://github.com/eclipse-xfsc/osc-devops-docs>

- Layer 2 Virtual Private LAN Service (VPLS)
- Layer 3 Route Reflector architecture

This design follows the IX peering services with quality-of-service monitoring. Global peering enables a customer to exchange IP traffic with many networks connected to the IX via a single BGP session. It is designed to optimize Internet reachability, reduce transit costs, and improve performance by providing scalable, many-to-many connectivity to the global Internet ecosystem.

A CUG, in contrast, is a private, controlled connectivity environment that allows a defined set of participants to communicate exclusively with each other. It is typically used for enterprise, government, or partner networks that require secure, predictable, and isolated communication, independent of public Internet routing.

2.4.3.2 Technical Architecture Layer-2 / Layer-3

The CUG is implemented as a private Layer 3 (IP-based) interconnection service running on the IX network fabric (e.g., DE-CIX Fabric).

This means:

- The connection does not traverse the public Internet.
- No public IP addresses are required.
- Traffic remains within an isolated routing domain (e.g., VLAN or VRF).
- Routing decisions are controlled and managed within the CUG environment.
- The Layer 2 VPLS component provides the private transport domain across the IX fabric, while the Layer 3 Route Reflector ensures scalable and controlled routing between participants.

The CUG therefore provides a secure and deterministic transport layer between locations, forming the foundation for higher-layer services such as encrypted application communication or Zero Trust connectivity.

2.4.3.3 Connectivity Model

The CUG service instance is accessible across the IX network via a CUG Membership based on IX access port.

Participants connect to the IX network via their geographically closest entry point. Connectivity can be achieved through:

- Metro Connect or Cross Connect (if the participant has local data center presence),
or
- Third-party Ethernet transport from an on-premises location to the nearest IX site.

2.4.3.4 Architecture for Connecting the Kubernetes Clusters

Each participant operates their own Kubernetes cluster (e.g., IONOS Cloud and T-Systems OSC). The connection is implemented as follows:

- Each cluster includes a dedicated ZT Connector.
- The connector is the only permitted communication endpoint between the clusters.
- The connectors communicate with each other via the private CUG connection.
- Within each cluster, communication is handled through a service mesh (e.g., Istio or Cilium).

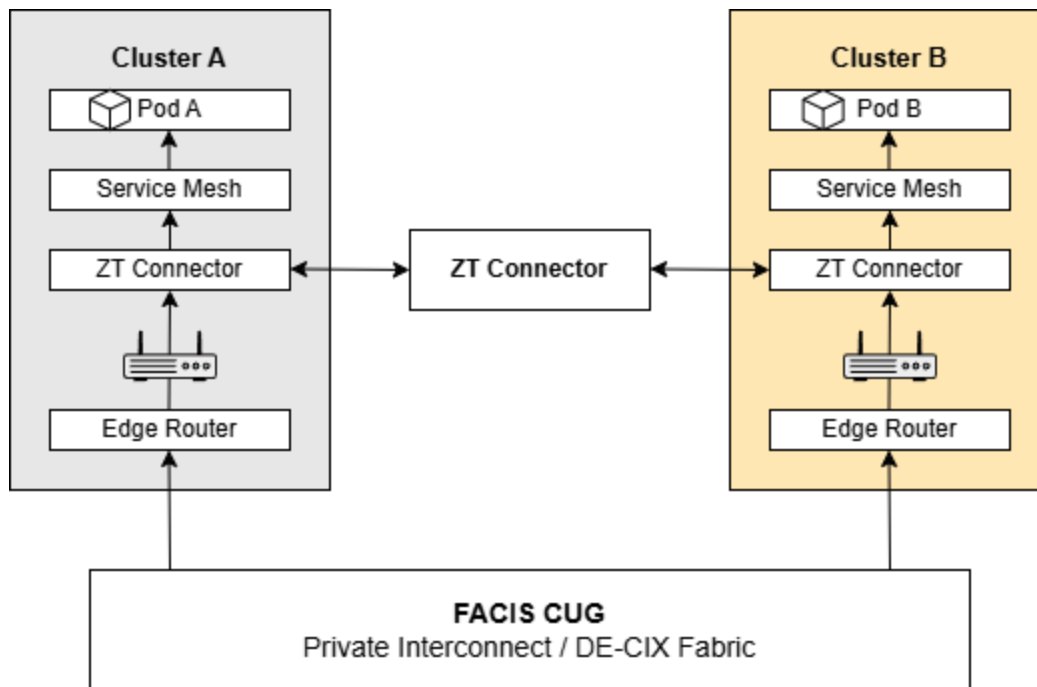


Figure 4 Simplified Traffic Flow

2.4.3.5 Benefits of the CUG in FACIS ZT Framework

The CUG provides the following advantages:

- No traffic over the public Internet
- Clearly defined participants
- Deterministic routing
- Support for guaranteed latency
- Scalability for additional federation members
- Alignment with zero trust principles.

2.4.3.6 FACIS CUG Setup Communication Process

Within each FACIS Kubernetes cluster, a ZT Connector acts as the controlled exit and entry point for external communication. When two participants are connected via a CUG, communication does not occur directly from pod to pod. Instead, it always passes through clearly defined transition points.

The process works as follows:

1. An application in Cluster A needs to communicate with an application in Cluster B.
2. The traffic is first handled internally within Cluster A through the service mesh.
3. The traffic is then forwarded to the ZT Connector of Cluster A.
4. The connector sends the traffic to the site's edge router.
5. The edge router determines, based on its routing table (for example via BGP), that the destination network is reachable through the private CUG.
6. The traffic is transmitted over the dedicated connection toward the IX infrastructure.
7. The IX infrastructure transparently forwards the Ethernet frames to the router of Cluster B.
8. The edge router of Cluster B receives the traffic and forwards it into the internal network.
9. The ZT Connector of Cluster B processes the incoming connection.

10. Only after successful verification (TLS, attestation, policy enforcement) is the traffic forwarded into the internal service mesh.

2.4.3.7 End-to-End Flow of a Single Data Packet

Assume that a single data packet is sent from an application in Cluster A to an application in Cluster B.

The logical path of the packet is as follows:

1. First, the application running inside a pod in Cluster A generates the data packet. The packet leaves the pod and is processed by the Kubernetes networking stack.
2. If the destination is outside the local cluster, the node recognizes that the destination IP address is not local. The packet is therefore forwarded to the node's default gateway.
3. This default gateway is typically the edge router or a virtual router within the cloud infrastructure.
4. The edge router checks its routing table. It identifies that the destination network of Cluster B is reachable via the private CUG connection.
5. The packet is then forwarded over the dedicated connection toward IX.
6. The router of Cluster B determines that the destination network is locally reachable and forwards the packet into the internal subnet.
7. From there, the packet reaches the Kubernetes node in Cluster B, is processed by the Zero Trust Connector for verification, and is then delivered through the service mesh to the target pod.

The entire physical path is determined by routing tables and BGP decisions and not by Kubernetes itself.

2.5 User Documentation

[ZT-16] Cloud Environments

Priority: MUST

Description: The user documentation MUST contain documentation about the operations in both environment and the documentation about the setup. Each documentation MUST be provided in MD file format on GitHub.

Acceptance Criteria:

Documentation can be demonstrated for reproducing the setup of each test cluster step-by-step.

[ZT-17] Demonstrator Usage

Priority: MUST

Description: The documentation and usage of the Demonstrator MUST be documented in a way that a manual for the Demonstrator can be generated automatically over GitHub actions.

Acceptance Criteria:

Documentation can be demonstrated for reproducing the setup of each test cluster step-by-step.

2.6 Assumptions and Dependencies

2.6.1 Attestation Code Basis

The Bidirectional Remote Attestation feature relies on the Apache 2.0-licensed CMC aTLS protocol⁶ (see Appendix for implementation reference). If this dependency cannot be fulfilled, an alternative **MUST** be documented and approved by the client prior to implementation. See Section 3.1.3 for protocol requirements.

2.6.2 Visualization

For the visualization is the assumption that grafana plugins based on Open Telemetry logs can be used to visualize the functionality of the Demonstrator. If this is not the case, another technology **MAY** be used.

2.7 Apportioning of Requirements

The product **SHALL** be set up in multiple phases to ensure the integrity of the product's functionality. Each phase **SHALL** be reflected in the milestone planning of the project plan:

Table 5 – ZT Demonstrator's Development Phases

Phase	Target Description
Demonstrator UX Concept	- Concept created and aligned with the client - Concept documented on GitHub
Cluster Setup	- Two managed clusters in OSC MUST be fully up and running, OPA Gatekeeper + signed image policies, secure service mesh, CI/CD and image signing and attestation pipeline to deploy images. - A third cluster MUST be up and running, including CI/CD on the IONOS Cluster. - DNS Zone for TRAIN is reachable and created. - XFSC Stack is deployed.
Zero Trust Connector Development	- Zero Trust Connector deployed in dedicated environment - Connection can be established - Measurement Values are available
Zero Trust Visualization	- Protected Resource Mockup(s) up and running - Participant Backend(s) deployed and connected to Zero Trust Connector - Communication between two Clusters is visualized and can be managed via a UI

⁶ <https://github.com/Fraunhofer-AISEC/cmc>

3 Requirements

3.1 Functional Requirements

3.1.1 Zero Trust Connector

[ZT-18] TRAIN Allow List

Priority: MUST

Description: The Connector MUST consume TRAIN lists to allow over policies in which counter-connector connections are allowed.

Acceptance Criteria:

Connector reads TRAIN trust lists to allow counter-connector connections.

[ZT-19] Upstream / Downstream Proxy Functionality

Priority: MUST

Description: The connector MUST allow bidirectional communication to another connector by utilizing the attested TLS channel. This includes HTTPS headers and other relevant call information.

Acceptance Criteria:

Connector uses and establishes the TLS channel.

[ZT-20] Guarding Functionality

Priority: MUST

Description: The connector MUST provide an application connection guard such as OAuth2 Provider, PIP, PEP or PAP. The enforcement MUST be based on Rego Policies and TSA Policy Engine. The guarding MUST be deployable for each protected resource on REST/GRPC basis. If any unauthorized access occurs, the guard SHOULD respond with OID4VP links.

Acceptance Criteria:

- Connector can guard a protected resource,
- No access without matching policies and authorizations.

[ZT-21] OAuth2 Server

Priority: MUST

Description: The OAuth2 Server of the connector **MUST** be able to dynamically register new participant backends via Dynamic Client Registrations to allow participant backends to use the connector. On the other hand, the server **MUST** also be able to issue access tokens based on OID4VP presentations to grant access for a counter connector. The server **MUST** utilize DPOP for access tokens.

Acceptance Criteria:

- Server issues access tokens based on OID4VP presentations,
- Server issues access tokens to participant backends for accessing the connector itself.

[ZT-22] Token Storage

Priority: MUST

Description: The Connector **MUST** store the issued access tokens of the counter connector after a credential presentation. If a participant backend call arrives, the used access token of the participant backend **MUST** be checked for DPOP headers, and if valid, the access token of the call **MUST** be upstreamed with the stored access token for the protected resource of the counter connector. The storage itself **MUST** keep caring about token renewal and OID4VP tasks.

Acceptance Criteria:

- Participant backend can call the protected resource over the connector,
- Upstream authorization is replaced via the token storage on DPOP basis.

3.1.2 Zero-Trust Service Mesh

[ZT-23] Service Mesh Implementation

Priority: MUST

Description: A service mesh **MUST** be used for inter-service communication within each Kubernetes cluster. This service mesh **MUST** either be Istio- or Cilium-based.

Acceptance Criteria:

- A service mesh based on Istio or Cilium is implemented in each Kubernetes cluster,
- Communication between services is routed through the service mesh.

[ZT-24] SPIFFE/SPIRE Deployment

Priority: MUST

Description: Each Kubernetes cluster **MUST** contain a deployment of SPIRE⁷. The service mesh **MUST** use SPIRE deployment to provide service identities. Identity verification **MUST** be based on SPIRE as well.

⁷ <https://spiffe.io/docs/latest/spire-about/>

Acceptance Criteria:

- SPIRE is deployed in each Kubernetes cluster,
- The service mesh is configured to use SPIRE-provided identities,
- Appropriate SPIRE node- and workload attestation is used,
- All services participating in the service mesh are registered as SPIRE workloads.

[ZT-25] Routing Policies

Priority: MUST

Description: Interactions between services in the service mesh **MUST** be governed by network policies. These policies **MUST** prevent interactions between unaffiliated services. These policies **MUST NOT** use insecure and unstable identifiers like an IP address.

Acceptance Criteria:

- The service mesh is configured to enforce network policies,
- Network policies are deployed to allow related services to communicate,
- A default deny policy is in place to prevent unaffiliated service communication.

[ZT-26] Ingress and Egress

Priority: MUST

Description: To ensure that the inter-cluster communication protocol is adhered to, appropriate ingress and egress rules **MUST** be configured to direct traffic to the ZT connector. Protocols (e.g. DNS) and endpoints not related to inter-connector communication (e.g. admin or observability APIs) **MUST** be able to bypass the ZT connector.

Acceptance Criteria:

- Ingress and egress for all inter-service communication is configured to use the ZT connector,
- Other APIs use a different appropriate ingress and egress configuration.

[ZT-27] Observability

Priority: MUST

Description: Observability **MUST** be considered when configuring the service mesh. Events **MUST** be exported from the cluster in a standard format such as Open Telemetry (Otel) traces and metrics:

- (1) OpenTelemetry Collector as the primary aggregation and export tool (OTEL standard),
- (2) Prometheus for metrics (retain),
- (3) OTEL-compatible tracing backend (e.g. Jaeger)

Acceptance Criteria:

Otel logs available.

3.1.3 Inter-Cluster Communication Protocol

[ZT-28] Attested TLS

Priority: MUST

Description: All communication between Kubernetes clusters **MUST** be performed over a mutually attested TLS channel. A high-level description of the desired protocol can be found in Section 5.2. The implementation of the TLS protocol **MUST** use TLSv1.3 or later and **MUST** be configured to reject any earlier version.

Acceptance Criteria:

- All inter-cluster communication uses the attested TLS protocol,
- An appropriate TLS version is used,
- TLS certificates are correctly checked against the cluster's identities.

[ZT-29] Attestation Request

Priority: MUST

Description: The attestation request message **MUST** be transmitted in a machine-readable format (e.g. JSON, CBOR). It **MUST** include the attestation type for the connection **REQUIRED** by the sender. Additional information **MAY** be included in the message.

Acceptance Criteria:

- The attestation request message is sent in a machine-readable format,
- The information contained in the message includes the desired attestation type, including at least:
 - Server-Side Attestation: Only the server provides an attestation report,
 - Client-Side Attestation: Only the client provides an attestation report,
 - Mutual Attestation: Both server and client provide an attestation report.
- The size of any additional information contained in the message does not exceed the size of the attestation report.

[ZT-30] Attestation Response

Priority: MUST

Description: The attestation response **MUST** be transmitted in a machine-readable format. The message **MUST** contain at least the attestation report. If the peer did not request an attestation report, the attestation report part of the message **SHOULD** be empty or null.

Acceptance Criteria:

- The attestation response message is sent in a machine-readable format,
- The attestation response contains an attestation report,
- The size of any additional information contained in the message does not exceed the size of the attestation report.

[ZT-31] Attestation Report Format

Priority: MUST

Description: To ensure that the sender's attestation report is valid, it **MUST** be cryptographically verifiable. A TEE format **SHOULD** be used for demo purposes. The report **MUST** contain the TLS-exporter channel binding specified in RFC 9266⁸.

Acceptance Criteria:

- All attestation report signatures can be verified using common cryptography libraries (e.g. OpenSSL),
- All attestation reports can be verified by the receiver without special knowledge; if additional information besides the attestation report is needed (e.g. an intermediary certificate), it is provided in TRAIN or as part of the attestation response message,
- All attestation reports contain the TLS-exporter channel binding for the TLS channel through which they are sent.

[ZT-32] Verification Result Message

Priority: MUST

Description: The verification result message **MUST** clearly communicate in a machine-readable format if the provided attestation report is acceptable. If the verification fails, the sender **SHOULD** provide a human-readable error message to assist in debugging.

Acceptance Criteria:

- The verification result message is sent in a machine-readable format,
- An affirmative verification result message is sent only if the attestation report sent by the peer could be verified successfully,
- The size of any additional information contained in the message does not exceed the size of the attestation report.

[ZT-33] Error Message

Priority: MUST

Description: If a peer encounters an error during the attestation protocol, it **MUST** terminate the protocol flow with a machine-readable error message. The error message **MUST** contain a human readable error message to assist in debugging. After sending or receiving an error message, either peer **MUST** close the TLS connection.

Acceptance Criteria:

- For any error case in each peer implementation, an error message is sent,
- The error message is machine-readable,
- The size of any additional information contained in the message does not exceed the size of the attestation report,
- Error messages are correctly received and lead to connection termination.

[ZT-34] Proxy Integration

Priority: SHALL

⁸ <https://doi.org/10.17487/RFC9266>

Description: The attested TLS protocol SHALL be integrated into a reverse proxy to accept attested upstream connections. The reverse proxy SHALL accept Kubernetes Ingress or Gateway API resources as configuration, possibly through an additional operator. It further MUST integrate with the service mesh to accept configuration and route inbound traffic to other members. The implementation SHOULD be implemented via the Go programming language.

[ZT-35] Hash Management/TRAIN Integration

Priority: MUST

Description: Any endpoint capable of establishing attested TLS connections (e.g. the Ingress gateway) MUST be able to resolve trusted hashes for its peers from TRAIN. This hash MUST reflect the launch digest value or equivalent of the endpoint's attestation report. For each endpoint, its CI/CD pipeline MUST pre-generate its own hash and upload it to TRAIN on deployment.

Acceptance Criteria:

- Upon connection establishment, the attested TLS endpoint fetches the peer's hash from TRAIN,
- Connection establishment proceeds only if the hash matches the value in the attestation report.

3.1.4 Zero Trust Execution Environment

[ZT-36] OPA Gatekeeper

Priority: MUST

Description: The Environment(s) MUST install OPA Gatekeeper and combine it with XFSC TSA Policy Service⁹ to load Rego policy packages for secure container admission.

Acceptance Criteria:

Gatekeeper consumes the policies and enforces them.

[ZT-37] OPA Gatekeeper Policies

Priority: MUST

Description: The Rego policies MUST be distributed as Tarball packages within the XFSC Harbor over the OCI interface. To publish the package, a GitHub action MUST be provided.

Acceptance Criteria:

- Policies are available,
- GitHub push action is in place.

⁹ [https://github.com/eclipse-xfsc/smartdeployment/tree/main/Easy%20Stack%20Builder%20\(ESB\)/TSA](https://github.com/eclipse-xfsc/smartdeployment/tree/main/Easy%20Stack%20Builder%20(ESB)/TSA)

[ZT-38] Container Signing Pipeline

Priority: MUST

Description: To allow containers for OPA Gatekeeper, a signing pipeline **MUST** be provided over GitHub actions and custom runner setups to sign Docker images. The key material **SHALL** be integrated in the runner setup, but the public keys **MUST** be published for verification.

Acceptance Criteria:

OPA Gatekeeper accepts Docker images which are signed by the client.

3.1.5 Zero Trust Application Communication

[ZT-39] Participant Backend Mockup

Priority: MUST

Description: The participant backend mockup **SHALL** dynamically register itself via dynamic client registration against the OAuth2 server of the ZT connector to obtain access tokens for the usage of the trusted channel. All actions **MUST** be provided for the visualization.

Acceptance Criteria:

- Participant backend can register itself,
- Participant backend can obtain access tokens,
- Participant backend can successfully pass PEP proxy to connect Zero Trust connector and its channel,
- Participant backend gets an answer from protected resource mock,
- Visualization reflects this interaction.

[ZT-40] Credential-Based Access Control (CrBAC)

Priority: MUST

Description: The Zero Trust Connector **MUST** be able to get access tokens from another zero-trust connector over credentials from OCM W-Stack. Each of those access tokens belongs to a registered participant backend and can be used only over DPOP extensions. The Zero Trust Connector **MUST** take care of the expiration and renewal.

Acceptance Criteria:

- A participant backend can use the Zero Trust Channel of the Connector to call protected resource, and the Connector will add the obtained Access Tokens of the counter connector to each call,
- DPOP Verification of each participant backend call and matching it against internal JWT storage,
- OCM W-Stack Presentations can be triggered by using OID4VP Verifier requests from counter connector.

[ZT-41] Protected Resource Mockup

Priority: MUST

Description: The protected resource mock simulates various responses based on access token roles. All actions MUST be visualized.

Acceptance Criteria:

Protected resource responds different results based on roles.

3.1.6 Zero Trust Communication Visualization

[ZT-42] UX Concept

Priority: MUST

Description: A visualization concept which explains how the Demonstrator works MUST be created on GitHub. The concept MUST be aligned beforehand with the client and designed via Wireframes or similar tooling. Any UI Design has to be implemented with the ORCE UI Builder.

Acceptance Criteria:

- GitHub documentation and GitHub Page ready,
- Concept aligned with the client.

[ZT-43] ORCE Usage

Priority: MUST

Description: The ORCE MUST orchestrate the visualization demonstration.

Acceptance Criteria:

ORCE is used.

[ZT-44] Value Visualization

Priority: MUST

Description: The Demonstrator MUST visualize important values like TEE measurements, credentials, TRAIN zones, communication decisions, messages etc.

Acceptance Criteria:

- Concept for value visualization aligned with the client,
- Values presented in Demonstrator.

[ZT-45] Demonstrator Configuration

Priority: MUST

Description: The Demonstrator MUST provide user interfaces for configuring TRAIN in the scope of the Zerto Trust Demonstrator (zones, policies).

Acceptance Criteria:

- Configuration documented,
- Configuration within the Demonstrator for certain parts can be demonstrated.

[ZT-46] User Journey Scenario

Priority: MUST

Description: The Demonstrator MUST demonstrate a scenario where an issuing process is providing a credential to OCM W-Stack to enable the protected resource access for a participant backend. This process can involve the user directly (e.g., by mobile app), or indirectly by providing user interfaces which unlock permissions. The scenario for this journey MUST be aligned with the client and MUST be described and documented on GitHub.

Acceptance Criteria:

- Scenario is aligned and documented on GitHub,
- Scenario is integrated into the Demonstrator and orchestrated by ORCE step-by-step,
- Scenario is illustrated with animations and images.

3.2 Non-Functional Requirements

[ZT-47] Continuous Authentication Enforcement

Priority: MUST

Description: The system SHALL continuously verify user and device identity for every access request to protected resources, regardless of network location.

Acceptance Criteria:

- All access requests require authentication and authorization,
- Session re-validation occurs at configurable intervals (≤ 15 minutes),
- Access is revoked automatically if session becomes non-compliant.

[ZT-48] Least Privilege Access Control

Priority: MUST

Description:

The system SHALL enforce least-privilege access by granting users and services only the minimum permissions REQUIRED to perform their tasks.

Acceptance Criteria:

- RBAC or ABAC is implemented,
- Privileged access rights are time-bound and require explicit approval,

[ZT-49] Micro-Segmentation

Priority: MUST

Description: Network architecture SHALL implement logical micro-segmentation to prevent lateral movement between workloads and services.

Acceptance Criteria:

- Workloads are segmented by policy-defined trust zones,
- East-west traffic is inspected and authorized,
- Unauthorized lateral communication attempts are blocked and logged.

3.3 General Security Requirements

[ZT-50] Cryptographic Standards (BSI TR-02102)

Priority: MUST

Description: All cryptographic algorithms and key lengths used within the demonstrator MUST comply with BSI Technical Guideline TR-02102 (Cryptographic Mechanisms). Specifically: symmetric encryption MUST use AES-256 or equivalent, asymmetric encryption MUST use RSA-4096 or ECC P-256/P-384, hash functions MUST use SHA-256 or stronger, and TLS configurations MUST follow BSI TR-02102-2. Deprecated algorithms (MD5, SHA-1, RSA < 2048) MUST NOT be used. Reference: BSI TR-02102-1 and TR-02102-2.

Acceptance Criteria:

- No deprecated algorithms present in any component.
- Key lengths documented and auditable.

[ZT-51] Credential Revocation

Priority: MUST

Description: The system MUST support credential revocation for Verifiable Credentials issued within the federation. Revocation MUST be implemented using a standards-compliant mechanism such as W3C Bitstring Status List v1.0 (W3C Recommendation, 15 May 2025) or IETF Token Status List (draft-ietf-oauth-status-list, active Internet Draft). Verifiers MUST check revocation status at credential verification time. Revoked credentials MUST NOT grant access to any protected resource. The revocation infrastructure MUST be available, as its unavailability MUST result in fail-closed behavior.

Acceptance Criteria:

- Revocation check performed at every credential verification.
- Revocation endpoint availability is monitored.

[ZT-52] Fail-Secure (Fail-Closed) Behavior

Priority: MUST

Description: In the event of unavailability of any Zero Trust control plane component (Policy Engine, TRAIN, SPIRE, OCM), the system MUST default to denying access rather than granting it (fail-closed). No access to protected resources SHALL be possible without a valid, verified policy decision. The system MUST detect control plane failures within 30 seconds and emit alerts via the observability stack. Partial degradation scenarios (e.g. one cluster unreachable) MUST be documented with defined behavior. Reference: NIST SP 800-207, Section 3.3.

Acceptance Criteria:

- Access denied when Policy Engine is unreachable (verified in test).
- Control plane failure detected and alerted within 30 seconds.
- Degradation scenarios documented with defined fail-closed behavior.

[ZT-53] Multi-Tenancy and Tenant Isolation

Priority: SHALL

Description: The system SHALL support logical multi-tenancy, allowing strict isolation between different organizational participants (tenants) within the same federation zone. Data, credentials, and policy configurations of one tenant SHALL NOT be accessible to another tenant under any circumstances. Tenant isolation SHALL be enforced at the Kubernetes namespace layer.

[ZT-54] Software Bill of Materials (SBOM)

Priority: MUST

Description: All software components of the demonstrator MUST have a documented Software Bill of Materials (SBOM) in CycloneDX or SPDX format. The SBOM MUST be generated automatically as part of the CI/CD pipeline and MUST be published alongside each release. The SBOM MUST include all direct and transitive dependencies, their versions, known vulnerabilities (CVE references), and license information. This is REQUIRED for NIS2 supply chain security compliance (Article 21) and enables rapid response to zero-day vulnerabilities across the dependency tree. The SBOM MUST be signed with the client's key material to ensure integrity.

Acceptance Criteria:

- SBOM generated in CycloneDX or SPDX format for every release.
- SBOM published in GitHub repository alongside release artefacts.

[ZT-55] Management Plane / Data Plane Separation

Priority: MUST

Description: The system MUST enforce strict separation between the management plane (Policy Engine, TRAIN, SPIRE control plane, OCM, observability stack) and the data plane (application traffic, participant backend communication, protected resource access). Management plane components MUST be accessible only via dedicated control channels and MUST NOT be reachable from the data plane network. This separation MUST be enforced at both the Kubernetes network policy layer and the service mesh layer. Compromise of the data plane MUST NOT grant any access to management plane components. Reference: NIST SP 800-207 Section 3 - Zero Trust Architecture Components.

Acceptance Criteria:

- Management plane components not reachable from data plane (verified in network test).
- Separation enforced at both network policy and service mesh layers.
- Architecture diagram clearly shows plane separation.

[ZT-56] CI/CD Pipeline Security (Zero Trust for the Pipeline)

Priority: MUST

Description: The CI/CD pipeline itself **MUST** be treated as a critical security boundary and **MUST** apply Zero Trust principles. This includes: (1) All GitHub Actions workflows **MUST** run with minimum necessary permissions (least privilege on repository secrets and tokens); (2) Third-party GitHub Actions **MUST** be pinned to exact commit SHAs, not mutable version tags; (3) Pipeline-generated artefacts (Docker images, policy packages, SBOMs) **MUST** be signed before use; (4) Secrets **MUST** never appear in pipeline logs; (5) All pipeline runs **MUST** be logged (6) Branch protection rules **MUST** enforce code review before merge to main.

Acceptance Criteria:

- All third-party GitHub Actions pinned to commit SHAs.
- Pipeline runs with minimum **REQUIRED** permissions (no wildcard write access).
- No secrets visible in any pipeline log output.

4 Design and Implementation

The system **SHALL** be designed and implemented according to Zero Trust principles, assuming that no user, device, workload, or network segment is inherently trustworthy. The architecture **SHALL** align with the guidelines defined by National Institute of Standards and Technology Special Publication 800-207¹⁰, including continuous authentication and authorization, policy decision and enforcement points, least-privilege access, and comprehensive monitoring. All access to resources **MUST** be explicitly authenticated, authorized, and continuously validated based on verified identity, device posture, contextual risk, and dynamic access policies.

Furthermore, the implementation **SHALL** follow applicable technical recommendations and baseline protection requirements published by the Bundesamt für Sicherheit in der Informationstechnik (BSI), including relevant technical guidelines (BSI-TR)¹¹, IT-Grundschutz controls and requirements for secure identity management, network segmentation, logging, cryptographic protection, and secure configuration management¹². The system **SHALL** enforce multi-factor authentication, strong cryptography compliant with recognized standards, encrypted communication channels, and tamper-resistant logging mechanisms.

The ZT Demonstrator **MUST** ensure auditability, traceability of access decisions, high availability of identity and policy components, scalability for enterprise workloads, and fail-secure behaviour in case of component failure. Security controls **SHALL** operate independently of network location and support on-premise, cloud, and hybrid environments while maintaining compliance with both international best practices and applicable national cybersecurity regulations.

The end-to-end integrity of the ZT setup **SHALL** be ensured through automated, behavior-driven validation mechanisms. All security-relevant requirements, policies, and trust assumptions **MUST** be formally specified using BDD methodologies (e.g., Gherkin syntax)

¹⁰ <https://doi.org/10.6028/NIST.SP.800-207>

¹¹ https://bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/technische-richtlinien_node.html

¹² https://bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/it-grundschutz_node.html

and translated into executable acceptance tests. These tests SHALL cover identity verification flows, policy decision logic, micro-segmentation rules, encryption enforcement, logging behavior, and fail-secure scenarios. CI/CD pipelines MUST automatically execute these test suites to validate that security controls remain effective after configuration changes, software updates, or infrastructure modifications. Any deviation from defined ZT policies SHALL result in automated build or deployment rejection. This approach ensures traceability from security requirements to implementation, continuous compliance verification, and verifiable end-to-end integrity of the ZT architecture.

5 System Features

5.1 Zero Trust-based Service Mesh

5.1.1 Description

The purpose of the service mesh is to establish trust between the different microservice within a cluster by establishing an identity-based microservice communication. The mesh is based on Cilium or Istio and uses SPIFFE/SPIRE to manage the identities of the microservices. To retrieve an identity, a microservice MUST prove itself against the SPIRE agent. Afterwards, the service mesh ensures that the identity is used for all communications. This enables restricting communications to connections between identities permitted via the configuration and using basic policies to restrict communication within the cluster.

The feature consists of the following functionalities to establish the service mesh:

SPIFFE/SPIRE Setup: The feature MUST use SPIFFE/SPIRE to perform basic checks against each microservice and provide it with an identity on success.

Service Mesh: The feature MUST utilize Cilium or Istio to ensure that all communication on the node goes through the identity check and restricts the communication between the microservices to those permitted by the configuration.

5.1.2 Architecture

A typical service mesh consists of a control plane that handles service discovery and configuration, as well as a data plane that forwards data between services within a Kubernetes cluster. Figure 5 provides an exemplary overview of the desired architecture and interactions between the specified components. In case of Istio's ambient mode, the control plane consists of the istiod¹³ while the data plane is implemented as one Ztunnel¹⁴ daemon per node. In addition to the service mesh, the use of SPIRE is REQUIRED to assert the identity of each node and pod as well as provision of service mesh certificates for each pod. The service mesh MUST be configured to fetch the certificate on behalf of each pod (e.g., using the SPIRE-delegated identities API¹⁵) and use it in mutual TLS authentication for communication in the service mesh.

Data from external sources enters and exits the service mesh through the ZT Connector, thus eliminating the need for an external gateway service supplied by the service mesh. Data entering the service mesh from the ZT Connector MAY be routed to any service within

¹³ <https://istio.io/latest/blog/2020/istiod/>

¹⁴ <https://istio.io/latest/blog/2023/rust-based-ztunnel/>

¹⁵ https://spiffe.io/docs/latest/deploying/spire_agent/#delegated-identity-api

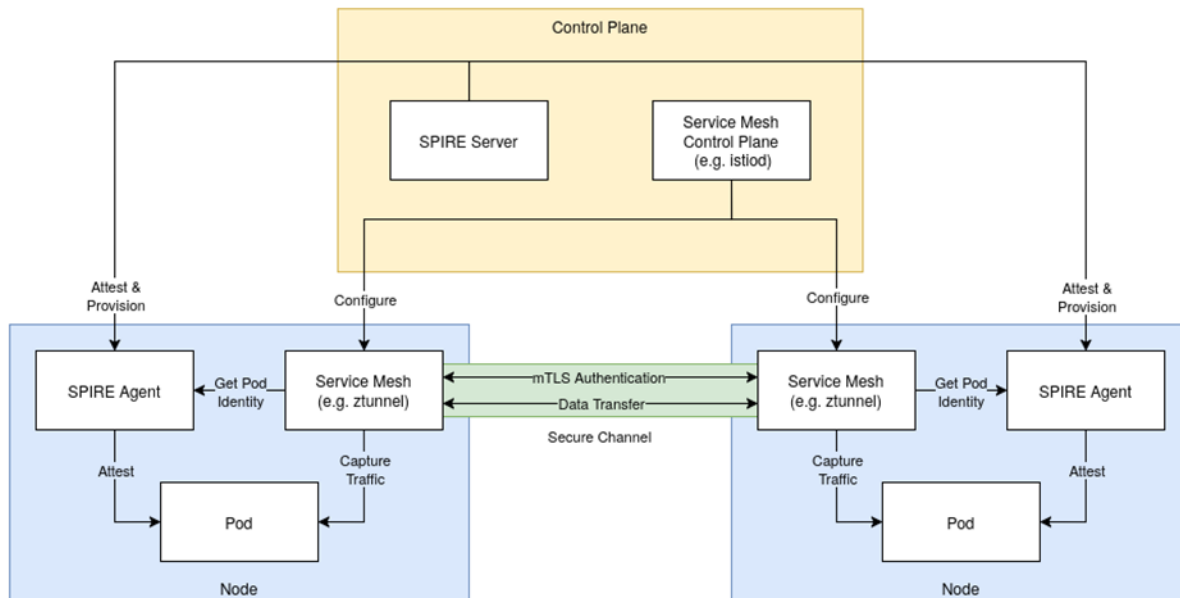


Figure 5 Architecture Service Mesh and Interaction

the service mesh but **MUST** pass an additional proxy acting as a policy enforcement point. Interactions between different services are restricted based on network policies that are enforced by the service mesh itself.

5.1.3 Definition of Done

Feature is complete when a pod fetches its identity from a SPIRE Agent and the identity is afterwards used in all communication within the mesh.

Table 6 – Requirements for a Zero Trust-based Service Mesh

Number	Name	Description
ZT-57	SPIFFE/SPIRE Setup	SPIFFE/SPIRE is set up and each microservice has access to a SPIRE Agent on the same node.
ZT-58	Service Mesh Setup	The service mesh is enabled, and the service mesh control plane is set up. An entity that captures traffic (such as Ztunnel daemon) and uses the pod's certificate for mutual TLS is available on each node.
ZT-59	Successful Identification	The microservice successfully attests itself against the SPIRE Agent and receives an identity.
ZT-60	Unsuccessful Identification	The microservice unsuccessfully attests itself against the SPIRE Agent and does not receive an identity. The microservice cannot connect to any other service in the mesh.
ZT-61	Traffic routed via Service Mesh	All communication of the microservice is routed via the service mesh.
ZT-62	Limited Communication within Mesh	The microservice is only able to communicate with identities permitted in the configuration.

5.2 Bidirectional Remote Attestation

5.2.1 Description

The purpose of the remote attestation is to establish a TLS connection between two ZT Connectors bound on the successful remote attestation of each other. The procedure includes the verification of an attestation report, which among others contains the measurement of the mocked TEE. To be able to verify this measurement, correct measurements are stored in TRAIN in the respective TRAIN Trust Content Resolver (TCR). This allows the verifier to fetch expected measurements for the remote ZT Connector, which it can then match against the measurement received in the attestation report.

To establish this connection, the feature consists of the following functionalities:

Trusted Environment Setup:

The setup of a real TEE environment is not REQUIRED, but the attestation file formats MUST be mocked.

TRAIN Setup: The feature MUST utilize a shared TRAIN infrastructure to exchange information between ZT Connectors.

TLS Setup: The feature MUST utilize TLS to ensure the integrity and confidentiality of the exchanged data.

5.2.2 Architecture

REQUIRED is the use of an attestation protocol that is performed after the TLS handshake is completed. Figure 6 shows a high-level representation of the protocol flow for one peer, which is described in more detail below. While no specific protocol is REQUIRED to be used, Appendix includes an example of the usage of an acceptable protocol. This protocol flow MUST be performed by the client and server of the TLS connection to ensure that both peers are attested. As the client's and server's protocol flow are independent, they MUST be performed in parallel to avoid high handshake delays. In the following, we will refer to the two peers as initiator and responder.

After the TLS handshake has finished, the initiator begins the protocol flow by sending an attestation request message containing the type of attestation the initiator requests. This can be server-only, client-only or mutual attestation. The responder MUST terminate the protocol with an error if it requires a different type of attestation.

The responder then generates an attestation report using the mocked TEE that contains the TLS-exporter secret associated with the underlying TLS connection. This value MUST be included so that its integrity is ensured by the attestation report, e.g. by including it in the report's user data field. The responder then sends an attestation response message to the initiator containing that report.

When evaluating the attestation report sent by the responder, the initiator MUST ensure that the correct TLS-exporter secret is contained within and that the mocked TEE's launch digest matches the expected value found in the responder's TCR. If this verification fails, the initiator MUST terminate the protocol. If the attestation report is acceptable, the initiator

sends a verification result message to the responder, indicating that the initiator is ready to proceed to the application layer protocol.

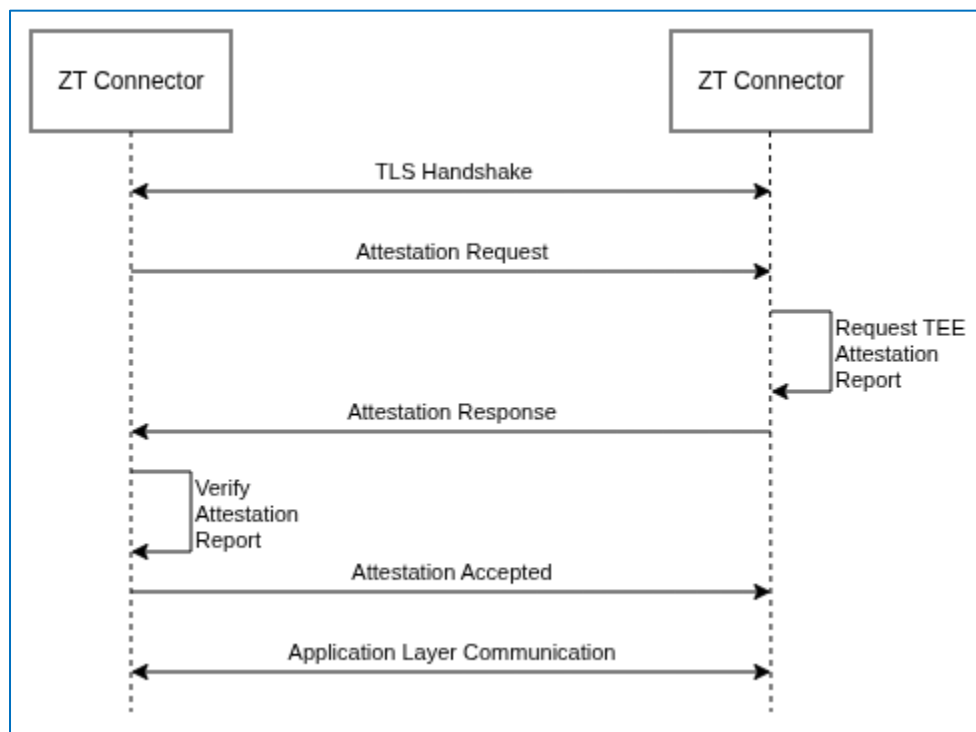


Figure 6 Protocol Flow for One Peer

5.2.3 Definition of Done

A feature is complete when two ZT Connectors establish a connection between each other by performing the attested TLS protocol handshake and verifying the information in the received attestation report against the information in the respective TCR.

Table 7 – Requirements for Bidirectional Remote Attestation

Number	Name	Description
ZT-63	TEE Mock Setup	The TEE mock setup is enabled and generates Demo Attestations.
ZT-64	Expected Measurement Retrieval	A ZT Connector can fetch expected mocked TEEM values from other connectors via TRAIN.
ZT-65	Connection Start	ZT Connector 1 starts a connection by sending an attestation request.
ZT-66	Remote Attestation	ZT Connector 2 responds to the attestation request by creating an attestation report and sending it to ZT Connector 1.
ZT-67	Attestation Report Verification	ZT Connector 1 verifies the attestation report by comparing the expected mocked TEEM against the mocked TEEM values in the TCR.
ZT-68	Verification Result	ZT Connector 1 informs ZT Connector 2 about the result of the verification.

ZT-69	Established Connection	The handshake was successful, and application layer communication can take place via established connection.
ZT-70	Refused Connection	An error occurred during the handshake or verification of the attestation report failed. It MUST be proven that no application layer communication can take place

5.3 Zero Trust Execution Environment

5.3.1 Description

The ZT execution environment feature of the solution SHALL provide a mocked TEE together with secure Kubernetes configurations which is the basis for remote attestation and security relevant actions like authentication and authorization. To establish this, the feature consists of the following functionalities:

Trusted Environment Setup: The feature MUST mock the behavior of a TEE environment by creating simulated attestation reports without any real enclave technology.

Admission Policies: The feature MUST ensure that only verifiable image according to an OPA Policy can be started. For this purpose, the OPA Gatekeeper MUST be used in combination with the TSA Policy Engine and TRAIN.

Signed Harbor Images: To sign the images which are used for the ZT environment, the used Harbor images MUST be signed in a way that the OPA Gatekeeper policies can verify from which source the images were created. For this purpose, the feature MUST provide a Cosign solution for Harbor which can be triggered via GitHub CI to sign a set of images for the ZT Demonstrator.

5.3.2 Architecture

Architecture as described in Chapter 2.4 Operating Environment.

5.3.3 Definition of Done

Table 8 – Requirements for Zero Trust Execution Environment

Number	Name	Description
ZT-71	Image Signing and mock Attestation	A defined set of images from Harbor can be defined for ZT environment, can be signed via CI/CD, and the signature is uploaded to harbor. A mock attestation is generated as JSON format for any TEE vendor.
ZT-72	Secure Pod Startup	OPA Gatekeeper is configured together with a policy, and the cluster starts only the images which are properly signed.

5.4 Zero Trust Application Communication

5.4.1 Description

The application of communication is the part of the Demonstrator for the usage of the secure channel between participant backend and protected resource over the ZT Connector. Within this communication channel, Zero Trust is enforced on application level and consists of the following parts:

- Credential verification to obtain access tokens of counter ZT Connector,
- JWT-based access to protected resources based on credentials,
- Policy enforcement and policy decision making by ZT Connector guarding,
- Routing of incoming ZT Connector calls together with access credentials to protected resources.

These parts SHALL be combined into a guarding solution which bi-directionally guards the REST/GRPC communication upstream/downstream.

5.4.2 Architecture

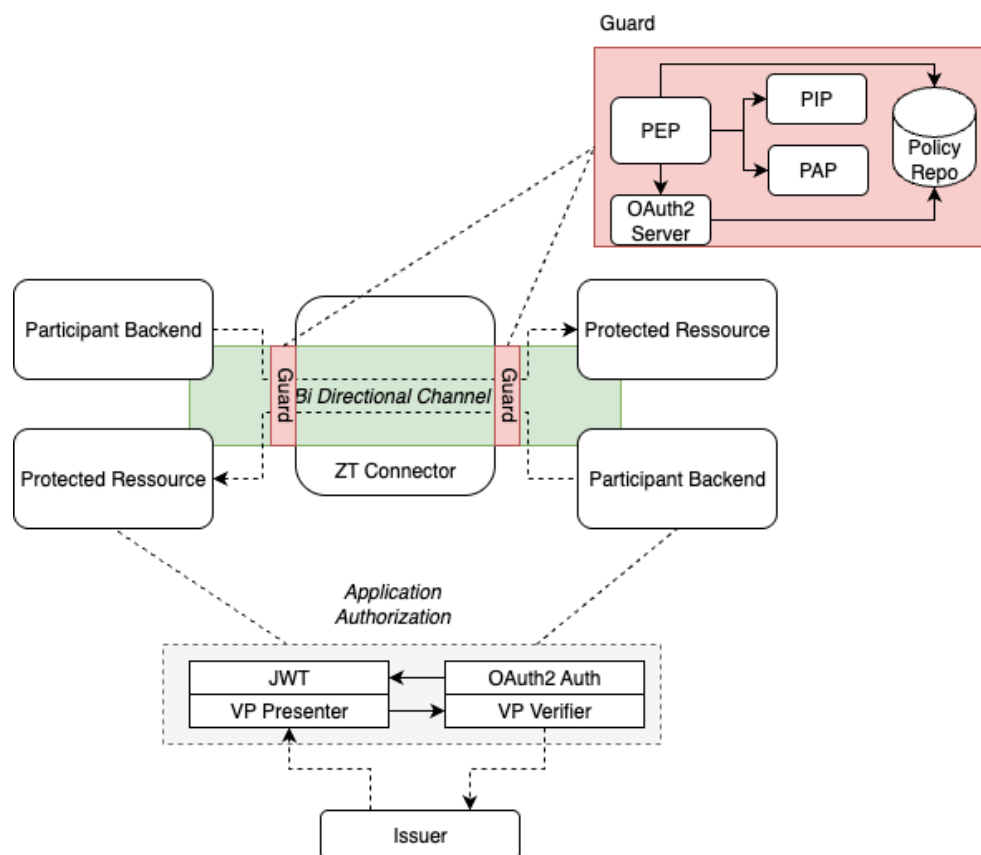


Figure 7 Zero Trust Application Communication

The architecture contains on both sides a guard setup with PIP, PEP, PAP, OAuth2 Server and Policies. This setup has the task to verify and authorize incoming requests for their validity, rate limits, and authorization.

5.4.3 Definition of Done

Table 9 – Requirements for Zero Trust Application Communication

Number	Name	Description
ZT-73	Authorized Upstreaming over the Zero Trust Connector	The participant backend can authorize and connect the ZT Connector and can successfully call over the trusted channel the protected resource. The protected resource verifies successfully the incoming call according to policies and responds with an answer.
ZT-74	Participant Backend Mock	The participant backend mock simulates and participant backend which uses the trusted channel to the protected resource and pushes status data to the visualization.
ZT-75	Protected Resource Mock	The protected resource mock simulates a protected resource and reacts on various combinations of access rights. All accesses are visualized in the visualization.
ZT-76	Credential Based Access	Based on credentials, the protected resources are returning various responses. All credentials MUST be exchanged beforehand against a JWT token and mapped into any kind of roles.
ZT-77	Mock Replaceable	All mocks are replaceable via demonstrating a real-world API.

5.5 Zero Trust Communication Visualization

5.5.1 Description

The Demonstrator SHALL provide a visualization for the Zero Trust activities, which shows, for instance, the measurement values of the connection, IP addresses, used policies or their results. Furthermore, the visualization SHALL allow to manage used policies, manage the TRAIN zone and connect to a second cluster via TRAIN discoveries (Figure 8). The process itself SHALL be visualized as a scenario which demonstrates a user journey. In summary, the workflow of the running Demonstrator setup SHALL be comprehensible to a wider audience with less knowledge about Zero Trust.

5.5.2 Architecture

The architecture for the visualization SHALL be designed as a workflow-driven UI by using ORCE, which demonstrates step-by-step communication and a guided configuration to the user. The entire feature can be understood as presentation UI for the Demonstrator.

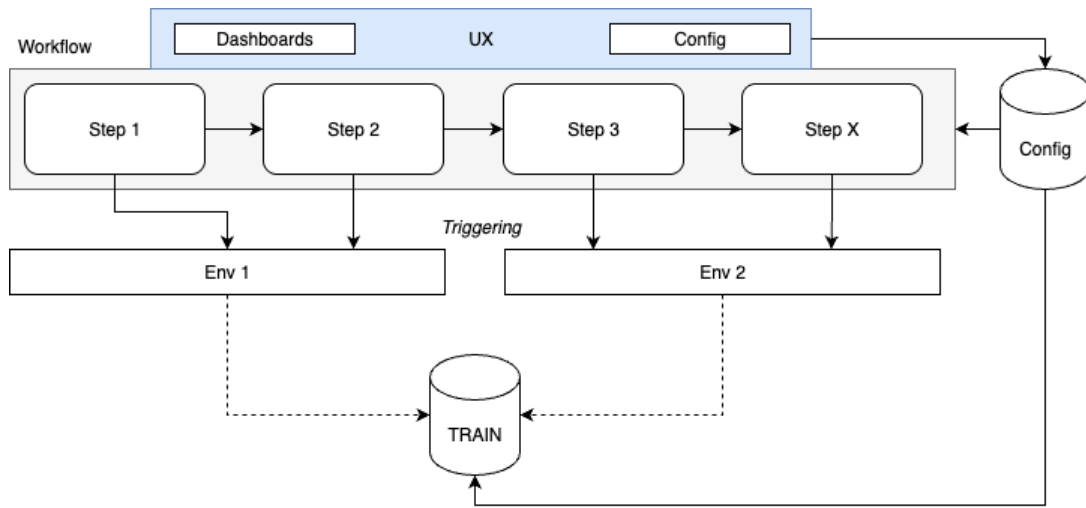


Figure 8 Workflow-driven UX

5.5.3 Definition of Done

Table 10 – Requirements for Zero Trust Communication Visualization

Number	Name	Description
ZT-78	Visualization UI	A visualization UI presents the core values of Zero Trust, like connection status, measurement values of the mocked TEE, sent messages, blocked messages etc.
ZT-79	Visualization Concept	The visualization concept is documented in GitHub and aligned with the client.
ZT-80	Zero Trust Configuration	The setup of the Zero Trust can be configured, e.g., protected resources, participant backends, mock replacement, attestation procedures and DNS records.
ZT-81	User Journey Scenario	The visualization demonstrates a user journey scenario orchestrated via ORCE. The scenario is aligned with the client and explained in documentation and visualization step by step.

6 Appendix

Example of an Attested TLS Protocol

An example of an attested TLS protocol as specified in Section 5.2 is the attested TLS protocol implemented at github.com/Fraunhofer-AISEC/cmc as an Apache 2.0-licensed project. Below is an example of how to implement a simplistic reverse proxy with the protocol. Note that this example would not fit the requirements for an ingress gateway laid out in this specification. Instructions on how to deploy the REQUIRED CMC daemon can be found in the linked repository.

```
package main

import (
    "crypto/tls"
    "fmt"
    "net/http"
    "net/http/httputil"
    "net/url"
    "strings"

    atls "github.com/Fraunhofer-AISEC/cmc/attestedtls"
)

type Endpoint struct {
    Prefix string
    ForwardTo *url.URL
}

// Obtaining the TLS certificate is out of scope for this example
func GetTlsConfig() *tls.Config

func ReverseProxy(endpoints []Endpoint, addr, cmcAddress string, policy string) error {
    // There are no special requirements for the TLS certificates for the attestation protocol
    tlsConf := GetTlsConfig().Clone()
    // We require at least TLS version 1.3
    tlsConf.MinVersion = tls.VersionTLS13

    handler := http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        key := r.Host + r.URL.Path

        var destination *url.URL
        for _, endpoint := range endpoints {
```

```
if strings.HasPrefix(key, endpoint.Prefix) {
    destination = endpoint.ForwardTo
}
}

if destination == nil {
    w.WriteHeader(http.StatusNotFound)
}

httputil.NewSingleHostReverseProxy(destination).ServeHTTP(w, r)
})

listener, err := atls.Listen("tcp", addr, tlsConf,
// The address the cmc is listening on
atls.WithCmcAddr(cmcAddress),
// Specify that we require mutual TLS
atls.WithMtls(true),
// Set the attestation type to mutual attestation.
atls.WithAttest(atls.Attest_Mutual),
// Include a custom javascript policy to evaluate the attestation.
atls.WithCmcPolicies([]byte(policy)),
)

if err != nil {
    return fmt.Errorf("failed to listen for incoming connections")
}

server := &http.Server{
    Addr: addr,
    Handler: handler,
}

return server.Serve(listener)
}
```