A Zero Trust Single Sign-On Framework with Attribute-Based Access Control

Daniel Kaltenböck, Ilir Murturi, and Schahram Dustdar
Distributed Systems Group, TU Wien, Vienna, Austria.

***Abstract*—Authentication, authorization, and access control are fundamental functionalities that are crucial for network infrastructures and software applications. These functionalities work together to create a fundamental security layer that allows administrative entities to control user actions. Implementing a security layer may be simple for basic applications, but as modern digital infrastructures become more complex, more advanced security systems are needed. Traditional perimeter-based security models, long relied upon for securing large networks, exhibit vulnerabilities and lack adaptability to modern architectures. As technology advances, there is a growing demand for new authentication and authorization systems to keep up with the changes. Zero Trust (ZT) emerges as a paradigm embodying such principles and concepts for constructing contemporary security systems. This paper introduces a ZT-based Single Sign-On (SSO) framework to demonstrate how ZT can be realized in multi-service environments using Attribute-Based Access Control (ABAC). A prototype is developed to show the feasibility and applicability of the proposed framework in a smart city context.**

***Index Terms*—Zero Trust, IoT, Computing Continuum, ABAC, Security**

I. Introduction

In recent years, network infrastructures have been rapidly evolving due to the increased number of Internet of Things (IoT) and distributed computing devices (e.g., fog and edge devices [1]). This shift from centralized resources to more distributed computing resources brings new challenges, such as (i) an increasing demand to tailor application functionality to new environments [2], [3] and (ii) a need for novel security mechanisms to secure and protect resources distributed across the computing continuum [4], just to name a few. In order to grasp the complex security challenges, it is important to explore past authentication and authorization methods and comprehend why these solutions demand supplementary mechanisms.

Traditionally, the perimeter-based security model has been widely used for authentication and authorization [5]. It assigns permissions across various applications based on network attributes like IP addresses or subnet affiliations. Meeting basic security requirements in modern business IT infrastructures necessitates a more expansive perimeter. This involves establishing additional network structures, such as firewalls and demilitarized zones (DMZs) to control inbound and outbound communications. Moreover, such security models focus on securing resources using perimeter-based security or encryption techniques to ensure that only reliable and authenticated users may enter a secured domain [6]. However, the perimeter-based solutions may not fully apply in broad scenarios, for instance, within the Smart City context. This is because managing and protecting resources becomes even more complex when computing devices are distributed geographically and across the computing continuum. Such limitations underscore the importance of implementing new mechanisms and systems to address future architectural and security challenges. In this regard, Zero Trust (ZT) has emerged as a comprehensive architecture comprising guidelines and concepts for modern security systems to address these evolving challenges [7].

In a ZT environment, strict controls are enforced to monitor and authenticate all access to data and resources, regardless of their origin inside or outside the network [6]. Utilizing these tools makes it possible to identify and respond to suspicious or unauthorized activities swiftly. If such behavior is detected, specialized techniques and tools are used to conduct thorough investigations and assess any associated risks. Various tools and methodologies are utilized to monitor and manage activities within a ZT environment effectively. Upon analyzing logs and network traffic, investigations are conducted to identify a specific activity's source and potential consequences. Based on the findings, appropriate measures are taken to effectively address the situation [8]. These measures may include limiting access to data or resources, revoking access privileges, stopping lateral movement, or preventing further unauthorized access or damage. Furthermore, ZT requirements demand a robust security model to protect against unauthorized access to resources and sensitive data.

In this paper, we introduce a ZT-based Single Sign-On (SSO) framework as an architectural solution applicable to multi-service environments such as smart cities. The primary focus is demonstrating how policy evaluation, attributes, and data collection can be utilized to form an authorization gateway within such complex environments (e.g., computing continuum systems [9]). Attribute-Based Access Control (ABAC) is utilized to derive solutions meeting ZT requirements [10], [11]. ABAC is based on a set of rules defining permissions based on user, resource, and environment attributes. This approach demands descriptive data structuring regarding the entity that has to be controlled. The entity's attributes are evaluated against rules to determine whether access should be allowed or denied. ABAC offers a flexible and dynamic approach to access control, making it a suitable choice for implementing a ZT security model. The proposed architecture provides a tangible framework and illustrates the interaction between various components. Therefore, we developed a prototype to present the applicability and feasibility of the framework in the smart city context. Nevertheless, the proposed solution is still in its early development stage and bypasses many critical aspects.

The remaining sections are structured as follows. An overview, related work, and motivation are presented in *Section* II. *Section* III introduces and explains the architecture and concepts of the ZT-based SSO framework. *Section* IV explains authentication and authorization processes within the framework. In *Section* V, we outline implementation details, the prototype, and the limitations of the framework. Finally, we conclude our discussion in *Section* VI.

II. An Overview & Related Work

A. Zero Trust

Zero Trust (ZT) is a term for modern cybersecurity principles targeting modern authentication and authorization mechanisms, suitable for large enterprise infrastructures to counteract network vulnerabilities. The fundamental goal of ZT is to detach security measurements from an underlying network structure (i.e., helps to identify and prevent potential threats easily). By using advanced technology, ZT can detect and respond to potential threats quickly and effectively. ZT itself can be seen as a collection of specifications an actual implementation should obey. The following list comprises an overview of ZT architecture: [7]:

- *E*very access and operation within a ZT environment has to be verified. Any implicit access must not be granted.
- *C*onnections must be secure in terms of basics like confidentiality, integrity, and availability. Additionally, all communications must be monitored to track down malicious behavior and where it originates.
- *E*very entity in the system that executes operations must only possess the rights it needs to complete its task. No more and no less.
- *A*ll digital services or entities that allow other entities to access them are considered resources and must be treated accordingly.
- *E*very entity that can execute operations possesses attributes that describe them and their current operation. Based on these attributes, policies determine if an entity is allowed to execute an operation or not. Attributes are very important in a ZT environment as they describe the events within the system. Consequently, the more accurate an access is described by its attributes, the easier it will be to grant the least privilege and improve threat detection.

The ZT architecture presented in Figure 1 shows a logical proposal for a componentized system suitable for implementing ZT fundamentals. Nevertheless, only implemented components are explained, and peripheral components are addressed separately.

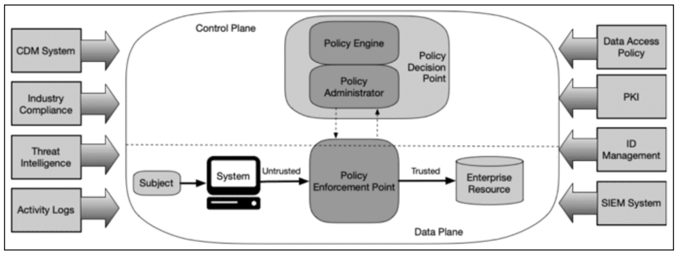


Fig. 1: Zero Trust Architecture [7].

1) Subjects: Subjects are entities that interact with the system. More specifically, their task is to access infrastructure/system resources. Subjects can be human users, machines, or anything similar capable of interacting with the system. Therefore, ZT focuses on important monitoring entities (i.e., subjects). Furthermore, to distinguish between good and malicious behavior, the system needs to identify and describe subjects exhibiting such behavior. In order to identify and distinguish between good or authorized behavior and malicious behavior, subjects must possess both static and dynamic attributes that can be used to evaluate their actions. Static attributes refer to the inherent characteristics of the subject, whereas dynamic attributes are properties that may change depending on the type of operations or access performed. By monitoring dynamic attributes, the system can learn and detect changes in a subject's behavior over time.

2) Objects / Resources: Resources are the targets that a subject wants to access. More practically, a resource can refer to any action a subject can perform within a digital infrastructure, such as accessing services, APIs, filesystems, and similar entities. To grant access to subjects, resources must also hold attributes that enable the evaluation of a subject's rights for a specific object.

3) Policy Enforcement Point (PEP): The PEP is a crucial component that decides whether a subject is authorized to perform a particular operation or not. Any request that fails to pass through the PEP is considered untrusted. The PEP gathers information about the current request, including the attributes of the subject and object involved, and sends it to the Policy Decision Point (PDP) to evaluate the request. Based on the evaluation results, the PEP either allows or denies the request. It can also intercept a request if necessary.

4) Policy Decision Point (PDP): The PDP evaluates access decisions based on available information. A ZT architecture requires all relevant data to be available to the PDP in time for decision-making. This component forms the foundation for an interface to modern technologies like machine learning to learn from the provided data and create trust evaluations based on subject behavior. Furthermore, the policy component is important to ensure that the Privacy Data Protection (PDP) system can adapt to changing circumstances. Both static and dynamic adaptation of the system's evaluation results can be achieved by defining policies that guide the decision-making process. For instance, a policy can set a minimum trust level output by a machine learning algorithm to accept a request

[12]. Similarly, authorization policies can enable access for subjects based on certain attributes. When combined, these evaluation approaches deliver the benefits of a ZT system.

5) Peripheral Components: In the ZT architecture illustrated in Figure 1, several peripheral components support the control plane. Peripheral components include services for subject management, monitoring, and security. This paper focuses on the components of threat intelligence and activity logging. Specifically, the proposed architecture provides communication flows and data formats suitable for activity tracking as well as connection points for threat intelligence solutions. This provides an interface for threat intelligence to access system data and enforce resulting decisions. Additionally, a simple public key infrastructure is used for subject authentication. In the next section, we will explore the role of these components in more detail within the proposed implementation architecture.

B. Related Work

Most existing research on ZT centers focuses on broad functionalities, such as analyzing behaviors, detecting threats, and assessing trust levels. These topics make up one of the front lines in ZT research because of their potential in combination with machine learning and similar state-of-the-art technologies. However, what is usually neglected is the specific structure and information flow of a system that is suitable for such analysis methods. Zhang et al. [13] propose a solution for trust evaluation based on tags using logical architectures, leaving room for interpreting an actual system design. Another work representing a typical ZT topic is [14]. This work focuses on trust evaluation and is, therefore, less precise in how data is gathered and carried to a potential component for executing trust evaluation. Further findings on work that refer more to ZT's architectural topics have similar characteristics.

The authors in [15] proposed a novel approach leveraging ZT extended with blockchain to enhance security. The approach utilizes blockchain for immutable user request storage and malicious activity identification, with experiments validating its feasibility and applicability in the context of smart cities. Qazi et al. [16] discuss architectural problems based on the logical structure proposed by NIST [7], though it does not go into detail about an actual architecture design. This raises the question of what a suitable system for advanced ZT features may look like. The basics of a ZT architecture presented in [7] is a foundation for the implementations and structures proposed in this paper. Several platforms such as Google BeyondCorp[1], Microsoft Azure Active Directory[2], Duo Beyond (Cisco)[3], and Cloudflare Zero Trust[4] are well-established and provide ZT solutions. However, such solutions are primarily focused on enterprise environments, which might not address the unique challenges within smart city and IoT contexts.

SSO allows users to access multiple related but independent software systems with a single authentication action, enhancing user productivity and simplifying central user management. Radha et al. [17] examine various SSO methods, implementation strategies, and the associated protocols, highlighting the benefits of adopting SSO within different network environments. Wang et al. [18] present a study on popular web SSO systems, uncovering eight serious logic flaws in high-profile providers and highlighting the need for larger-scale studies to better understand and address the security issues in SSO implementations. Witkowski et al. [19] proposed a solution using dual key-based schemes demonstrating the feasibility of Integrating Identity Management (IdM) to provide SSO in IoT environments without significant overhead for systems with up to 50 appliances. Nevertheless, none of these works address challenges within the ZT architecture adapted to the IoT context.

C. Motivation Scenario

Consider a smart city environment with hundreds of resources such as IoT resources, security cameras, and other monitoring facilities connected to the city's network (e.g., as shown in [20]. Including vehicle to infrastructure systems, the range of digital services and components can be spread among the different layers of the city's network structure, starting from edge nodes to data centers. These resources are accessed by existing clients, like apps and websites that monitor the data provided by those facilities. Humans are assumed to be using those clients to access the data. Conventional solutions like role-based access control or perimeter-based security make managing the city's digital infrastructure difficult. For instance, defining permission groups for every scope of duty in the city or configuring network hardware and services like firewalls are required. This setup has the disadvantage of allowing attackers to access multiple resources if they bypass single perimeter-based security measures. Furthermore, suppose an attacker can gain access to the account of an authorized person. In that case, it is hard to detect malicious behavior in the large infrastructure of a city that runs services in different resources of the computing continuum [21]. Therefore, an architectural approach is required to enable manageable monitoring and security mechanisms independent from network positions and interfaces to apply modern technologies. A ZT-based SSO integrated into the different applications can solve these challenges. Furthermore, by gathering information about access events in an infrastructure through an SSO service, the city's network security can be improved with threat intelligence to detect malicious events. For a more detailed understanding of the term multi-service used in this context, consider the various aspects of smart city operations, such as watering systems, emission monitoring, and other essential services. Each domain might have unique implementations developed by different companies or managed by various authorities. The SSO framework must offer a uniform solution for handling

[1]https://cloud.google.com/beyondcorp
[2]https://learn.microsoft.com/en-us/azure/security/fundamentals/overview
[3]https://www.cisco.com/site/us/en/solutions/security/zero-trust/index.html
[4]https://www.cloudflare.com/zero-trust/

authentication and authorization across these diverse applications. This ensures consistent and secure access management throughout the smart city's digital ecosystem.

III. ZT-based Single Sign-On Framework

A. Architecture and Technologies

The ZT-based Single Sign-On provides an architecture for an implementable system that shows how policy evaluation, attributes, and data collecting work together as an authorization gateway in multi-service environments like smart cities. For the proposed architecture and its subsequent implementation, it is important to note that the implementation only covers static attributes for simplicity and to demonstrate the way it operates. Nevertheless, it has been designed to directly integrate with dynamic attributes and threat intelligence, which is covered by system flow descriptions (see Section IV).

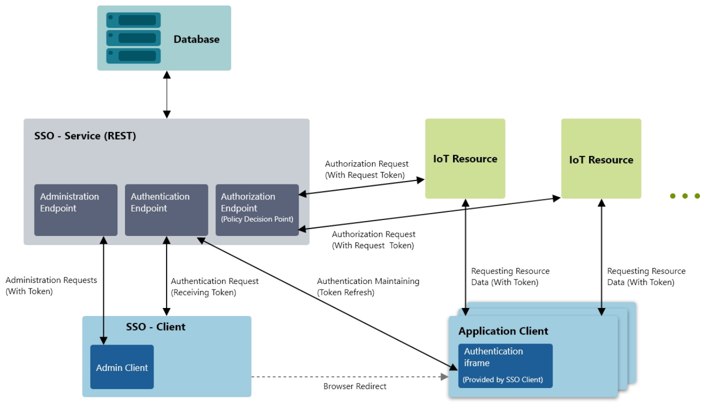

Database
SSO - Service (REST)
Administration Endpoint
Authentication Endpoint
Authorization Endpoint (Policy Decision Point)
Authorization Request (With Request Token)
IoT Resource
IoT Resource
Authorization Request (With Request Token)
Administration Requests (With Token)
Authentication Request (Receiving Token)
Authentication Maintaining (Token Refresh)
Requesting Resource Data (With Token)
Requesting Resource Data (With Token)
SSO - Client
Admin Client
Application Client
Authentication iframe
(Provided by SSO Client)
Browser Redirect

Fig. 2: An overview of ZT-based SSO architecture.

Figure 2 shows the fundamental components of the ZT-based SSO architecture. We assume that a ZT-based system is architecturally composed of microservice processes [22], [23]. Such microservices, once invoked, yield resources (e.g., IoT resources). The proposed architecture presents a simplified version in which some infrastructure components have been excluded. Each rectangular shape in Figure 2 represents an application that can be executed. The smaller rectangles inside them represent important parts and features of these applications. The solid arrows indicate the network communication that occurs between these components. It is important to note that applications connected by a solid arrow must communicate over the network. The dashed arrow at the bottom of the architecture indicates a redirection within a browser since web clients are used in this architecture demonstration. In the following sections, we describe each of these components in detail.

1) SSO-Service: The SSO-Service is the core part of the system and is responsible for user authentication, authorization, and administrative operations. The authentication endpoint handles the authentication by issuing signed JWT-Tokens[5] using the SSO-Service's key pair. The authentication endpoint makes its public key accessible for its applications. The administration endpoint allows for the management of SSO resources, such as policies and users. The authorization endpoint acts as the policy decision point and core of the system where the available information comes together for evaluation.

2) SSO-Client: The SSO-Client builds the bridge for web-clients to integrate the SSO with a uniform authentication page. The authentication flow for the SSO-Client is described in section IV. Additionally, a route is provided to be loaded into the application clients to retrieve the information about the authenticated user. This is required to integrate well with browser security for cross-domain authentication since different applications are assumed to run under different domains. In the implementation the SSO-Client holds the features for administration as shown in the architecture.

3) Database: The database serves as the persistence layer for authentication and authorization data. This data contains the defined policies, registered users, and associated attributes. These policies determine who can access specific resources, under what conditions, and with what privilege level.

4) IoT Resources: IoT devices are software components (i.e., generally as microservices) deployed in various environments and contain resources. Such microservices, once invoked (i.e., receive requests from a client), yield resources (i.e., measured values). The most important property of the IoT resources is the architectural decision to integrate the policy enforcement point as a gate into the resource itself. On one hand, this removes the problem of deploying the PEP as a separate application requiring load balancing. On the other hand, decision-making for handling cases that might not require the heavy SSO-Service evaluation can be integrated into the IoT resources to reduce network overhead. This presupposes that the resource has enough computing power to do so. The IoT resources produce simulated continuous measurements with random values as part of the demonstration.

5) Application client: The application client represents a web-client as a smart city application. For the sake of simplicity, let's assume that this application includes a user interface that enables end-users to interact with IoT resources.

IV. Authentication and Authorization

Web clients can take advantage of the SSO-client's uniform authentication form. This form can be used to redirect the user to the appropriate authentication process. Transferring the ID-token from the SSO-client to the web-client is known as cross-domain authentication (as discussed in Section IV-B). This is particularly helpful when the web-client is located under a different domain.

A. Retrieving the authentication/ID-token

The process starts with the application from where we want to start the authentication. As shown in Figure 3, the application redirects the user to the SSO-client and includes its URL as a query parameter to tell the client from where the

[5]JSON Web Tokens, https://jwt.io/

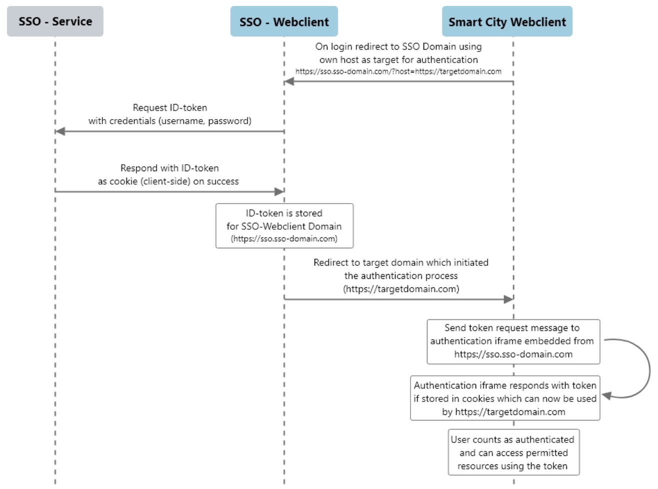

SSO - Service
SSO - Webclient
Smart City Webclient
On login redirect to SSO Domain using
own host as target for authentication
https://sso.sso-domain.com/?host=https://targetdomain.com
Request ID-token
with credentials (username, password)
Respond with ID-token
as cookie (client-side) on success
ID-token is stored
for SSO-Webclient Domain
(https://sso.sso-domain.com)
Redirect to target domain which initiated
the authentication process
(https://targetdomain.com)
Send token request message to
authentication iframe embedded from
https://sso.sso-domain.com
Authentication iframe responds with token
if stored in cookies which can now be used
by https://targetdomain.com
User counts as authenticated
and can access permitted
resources using the token

Fig. 3: Sequence diagram of the authentication flow.

redirect originates. After entering the credentials, the SSO-Client requests a token from the SSO-Service. The returned token is a JWT-token, which the SSO-Service signs to prove by whom it has been issued. The SSO-Service uses asymmetric key pairs to sign tokens. In the implementation, the keys are stored as files and will only change if the service cannot find them on restart. The current public key for the token can be retrieved from the service's authentication endpoint. Due to simplification, no client in the system does token verification since it has to be verified during the authorization process for resource access.

During the authentication process, the SSO-Client checks if the provided credentials match. If they do, we receive an authentication token, also known as an ID-token. If the credentials don't match, the SSO-Client will send an error message explaining the issue. To make use of the ID-token later in the process for cross-domain authentication, it needs to be set as a client-side cookie. However, doing so may pose security risks, such as cross-site scripting. If the authentication process is successful, the SSO-Client will redirect us back to the starting point of the authentication. If the SSO-Client can't find a redirect target, it redirects users to the client's administration UI.

B. Cross-Domain Authentication

After redirection, the solution for cross-domain authentication starts by loading a prepared part of the SSO client into an iframe within the application client. The client communicates with the iframe via post-messaging to share the ID-token. Furthermore, the iframe maintains a simple protocol for communicating with the application. Two different requests are accepted:

- **Maintain:** Validates the current ID-token and returns a refreshed one if possible.
- **Logout:** This message tells the iframe to delete all authentication information available in the browser for user logout.

Application clients can handle the response waiting times as they are required to provide a good user experience. After receiving a valid ID-token from the iframe, it can be stored as desired by the application. In the implementation case, it is stored as a client-side cookie to start requesting values from the IoT resources. Furthermore, a vital part of cross-domain authentication is the prevention of security vulnerabilities that can occur during the procedure. Passing the ID-token to another application requires some validation by using the origin of the post-messaging message. This topic is addressed in subsection V-D. Regarding the implementation, no security measures have been taken for this step to focus on the authorization part of the system.

C. Token Expiration

Tokens play a major role in SSO solutions and their security aspects We use asymmetrical cryptography to sign ID-tokens and no metadata about its state is stored on the server side. The signed tokens allow us to store immutable information inside them. This property can be used to determine an expiration time. Furthermore, we integrate a refresh mechanism that allows users to be as secure as possible while reducing the count of fresh authentications. The implementation uses an extend-on-use approach by which an expired one can retrieve a fresh ID-token within a specific time range after its expiration. The time range can be adjusted depending on the security and usability requirements.

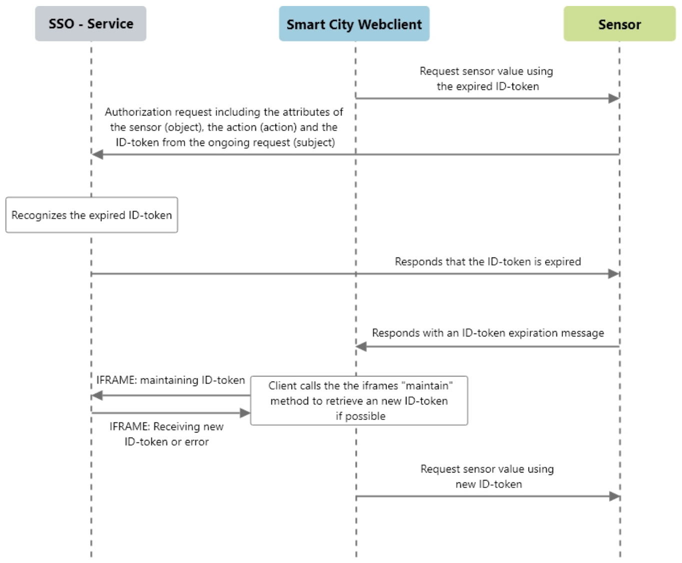

SSO - Service
Smart City Webclient
Sensor
Request sensor value using
the expired ID-token
Authorization request including the attributes of
the sensor (object), the action (action) and the
ID-token from the ongoing request (subject)
Recognizes the expired ID-token
Responds that the ID-token is expired
Responds with an ID-token expiration message
IFRAME: maintaining ID-token
Client calls the the iframes "maintain"
method to retrieve an new ID-token
if possible
IFRAME: Receiving new
ID-token or error
Request sensor value using
new ID-token

Fig. 4: Sequence diagram of the token refresh flow (e.g., a sensor as an IoT resource).

In Figure 4, we show the communication flow that happens in the system if an IoT resource responds with an expiration error. The client is informed of an ID-token's expiration only if it receives a message from the SSO-Service or an IoT resource during authorization. The token validation in the system only happens in the SSO-Service for simplifications. The client can fetch the token's public key from the SSO-Service to validate it and check its expiration date if required. Looking at Figure 4 the process starts with a normal IoT resource value request using an expired ID-token that is still renewable. The process during the authorization determining the token expiration is explained in subsection IV-D. When the client knows that its ID-token has expired, it will perform a maintain request to the iframe as already discussed in section IV. The maintain functionality of the iframe sends a maintain request to the SSO-Service for the ID-token, which responds with a renewed one. On success, the new ID-token is forwarded to the client to move on with requesting values.

D. *Authorization*

Authorization features enable the system to verify the identity and permissions of each user attempting to access its resources [24]. The authorization process is performed in the authorization endpoint of the SSO-Service. When a user or a subject requests a resource (e.g., accessing an IoT resource), all the data required to make a decision is collected along the way - from dynamic information available to the resource / PEP to threat intelligence systems that extend the data using their output. The structuring of data and policies is addressed in section V, where a more detailed explanation of implementing the system-based ABAC is given. This section provides a general flow of the data during an authorization process (i.e., the basic authorization flow used in the system for gathering the subject-, object-, and action-attributes). Subjects and objects (resources) should be known from subsection II-A. Action-attributes are a collection of data describing how a subject attempts to access an object. This helps to keep the data describing the operation separate from the data describing the object itself. Just like object-attributes, action-attributes can be defined by the resource. A common example of action-attributes is read or write.

The authorization flow in the architecture typically starts with access to a resource. The PEP located inside the IoT resource uses the ID-token and the object/action attributes to send them together as authorization requests to the service. At this point, we append all the available dynamic data, like request properties, to the request as a new attribute collection. At the SSO-Service the evaluation process starts by evaluating the attributes against policies that contain boolean conditions referencing the contained attributes as detailed in section V. Before the evaluation starts threat intelligence can be used to extend the attribute collections by adding the outputs of advanced learning algorithms. The subject attributes are gathered by using the ID-token as a placeholder and reference to load them from the SSO database. This procedure also includes token validation to inform the resource if it sent an invalid token directly. If one policy exists with all of its rules evaluated to be true, access will be granted by sending a successful response. A negative response will deny access if no such policy has been found. Last but not least, the resource processes the authorization response by performing the requested operation or informing the client about the reason for the rejection. In the case of IoT resources, a measured value is returned on success. The whole procedure is displayed in Figure 5.

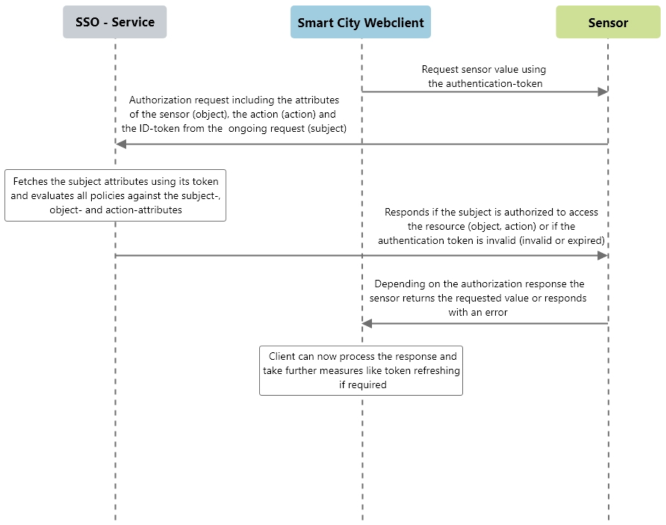

SSO - Service
Smart City Webclient
Sensor
Request sensor value using the authentication-token
Authorization request including the attributes of the sensor (object), the action (action) and the ID-token from the ongoing request (subject)
Fetches the subject attributes using its token and evaluates all policies against the subject-, object- and action-attributes
Responds if the subject is authorized to access the resource (object, action) or if the authentication token is invalid (invalid or expired)
Depending on the authorization response the sensor returns the requested value or responds with an error
Client can now process the response and take further measures like token refreshing if required

Fig. 5: Sequence diagram of the authorization flow (e.g., a sensor as an IoT resource).

V. Implementation

The well-known approach of attribute-based access control (ABAC) [11], [25] serves as the fundamental concept for building the requirements of the proposed system. In the following, we discuss authorization features that align with ZT principles and provide implementation details.

A. *Attributes*

The ABAC concept is based on the idea that every entity capable of performing tasks has attributes that accurately or sufficiently describe that entity. These attributes can be static (i.e., the department of an employee, etc.) or dynamic (i.e., the browser an employee uses for the current request). To determine whether an entity can access a resource, we require information about that resource. Therefore, we assign attributes to resources as well. It is a common approach to additionally include attributes for the type of access an entity performs, as already discussed in subsection IV-D. Regarding the representation, a collection of attributes can be expressed as a simple JSON-object:

```
// Example subject attributes of an
// employee
{
    "department": "development"
    "securityLevel": 5
    "hasTopEtageAccess": true
}
```

We use an attribute collection where each entry key represents an attribute name, and each entry's value represents the attribute value. However, the SSO prototype's attribute values are limited to string, number, and boolean data types. This attribute collection does not specify how to determine whether an entity is authorized to perform an action based on this information. Therefore, we need to create policies.

B. Policies

Policies are rules that define whether an entity is allowed to perform a specific request or not. This decision is based on the information, or in the case of ABAC, the attributes we know from the request. In the case of the ZT-based SSO implementation, a policy is a list of unary or binary boolean expressions called rules, which can include static values (string, number, boolean) and, most importantly, references to attributes. The following example shows a single policy consisting of two rules; each rule is a binary expression, including an attribute reference.

```
//Policy with 2 rules checking subject
//attributes.
  [
   "#subject_department == ’development’",
   "#subject_securityLevel <= 5"
  ]
```

For the prototype implementation, a simple policy language has been defined. Using this language, attributes of a collection are addressed using the ”#” for the collection name and the ”_” for the attribute name. In the example above, the first rule inserts the value of the attribute ”department” from the ”subject” collection on the left side of the equation during evaluation. The language definition is detailed in section V. A policy has a list of rules that can be evaluated as true or false if applied to a specific set of attribute collections. The whole policy is evaluated by combining all rule results using a logical AND operation. By applying this procedure, a policy is considered true if and only if all rules are true. This way, policies act as positive logical gates. The example below represents a complete authorization scenario.

```
// Attributes of the request:
// Subject attributes
  { "department": "development",
    "secLevel": 5
  }
// Object attributes
  { "type": "smartcity_measures",
    "secLevel": 4
  }
// Action attributes
  { "type": "read" }
// Policies:
  1:[
  "#subject_department == ’development’",
  "#subject_secLevel >= #object_secLevel",
  "#object_type == "smartcity_measures",
  "#action_type == "read"
    ] // Evaluates to true
  2:[
  "#subject_department == ’development’",
  "#object_type == "smartcity_measures",
  "#action_type == "read"
    ] // Evaluates to false
```

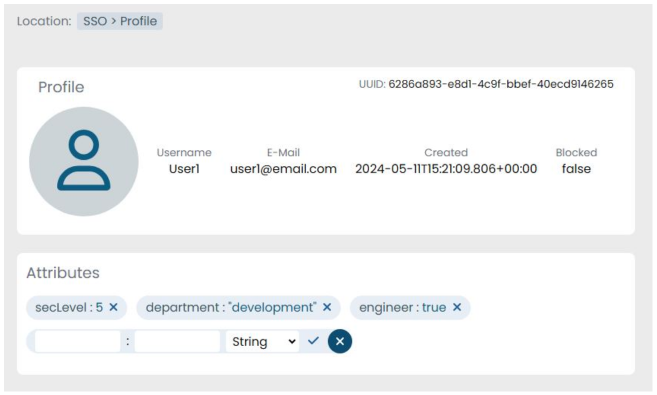
The first policy result grants the subject the privilege to operate. It is crucial to keep this definition in mind as a policy must describe an authorization scenario accurately enough so that not too many permissions are granted. If no further measures are taken before a new policy is added to the system, this problem can lead to a critical scenario, as a wrong policy can make all other policies ineffective. This can be shown by adding the following policy.

```
// Policy_03
 [
   "#subject_department == ’development’"
 ]
```

The above policy (i.e., Policy_03) evaluates to true when the subject has the department ”development” no matter which other attributes are included. This gives all subjects with department ”development” full access to the system. Nevertheless, such a problem belongs to the pitfalls that must be mitigated by further checks when considering production scenarios.

C. Prototype Implementation

A prototype[6] is developed to show the feasibility and applicability of the proposed approach. Figure 6 shows an example to allow admins to configure the user’s static attributes. It is important to specify the value type for each attribute. This helps to maintain consistency and accuracy in the evaluation, and ensures that all attributes are appropriately assessed.


Location: SSO > Profile

Profile

UUID: 6286a893-e8d1-4c9f-bbef-40ecd9146265

Username	E-Mail	Created	Blocked
User1	user1@email.com	2024-05-11T15:21:09.806+00:00	false

Attributes

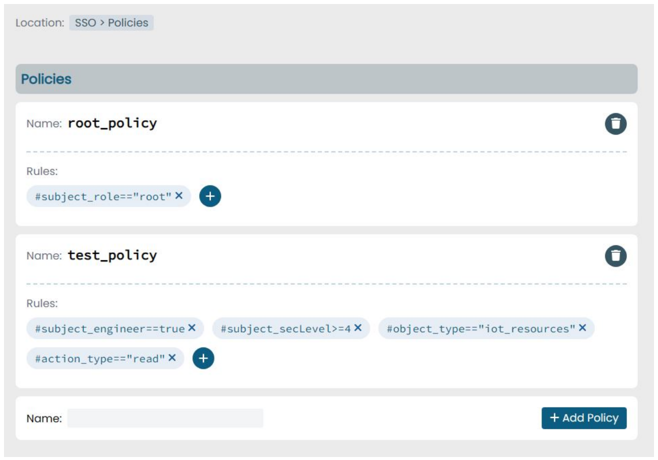
secLevel : 5 ✕ department : "development" ✕ engineer : true ✕

: String

Fig. 6: User attribute management.

Figure 7 shows the policy tool of the SSO-client. The figure provides a visual representation of two distinct policies that define the access control mechanism for a system. The first policy, called *root_policy*, is designed to act as a superadmin rule that allows users with the attribute role and root value unlimited access to the system features. This policy is particularly useful when a few users need complete control over the

[6]Zero Trust Single-Sign-On Framework using Attribute-Based Access Control, https://github.com/Fruit3r/zt_abac_sso_framework


Location: SSO > Policies

Policies

Name: root_policy

Rules:

#subject_role=="root" ×

Name: test_policy

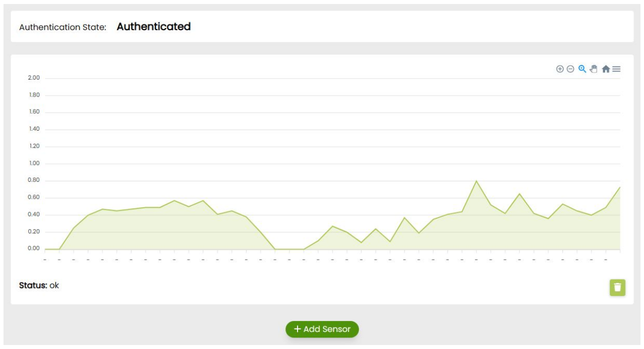
Rules:

#subject_engineer==true × #subject_secLevel>=4 × #object_type=="iot_resources" × #action_type=="read" ×

Name:

+ Add Policy

Fig. 7: Administration tools for policies.


Authentication State: Authenticated

Status: ok

Fig. 8: Accessing sensor data.

system, while others have limited access to specific features. The second policy, called *test_policy*, is a typical example of a policy configuration that includes subject, object, and action rules. This policy defines the specific actions that users with different roles can perform on various objects in the system. For example, it may allow users with the manager role to create and modify records in a database, while users with the viewer role can only view the records. Both policies have an input field that allows users to add new rules, making them flexible and adaptable to different scenarios. Overall, these policies play an essential role in ensuring that the system is secure and that users can only access the features that they are authorized to use. Furthermore, the syntax required to define basic rules can be described using a regular expression. This regular expression is used to specify a set of characters and patterns that determine the rules. In the following, the available syntax to define basic rules is described by a regular expression, and it is given below:

```
// Left side
  ^((#(subject|object|action))_[a-zA-Z]
  [a-zA-Z0-9]*)( )*
// Operators
  (==|!=|<=|>=|<|>)( )*
// Right side
  (((#(subject|object|action))_[a-zA-Z]
  [a-zA-Z0-9]*)|\"[a-zA-Z0-9_@./#&+-]*\"
  |[0-9]+|(true|false))$
```

The smart city client implements the application client as a smart city feature as discussed in subsection III-A. The implementation contains a UI to monitor sensors representing IoT resources. The sensors are implemented as standalone software in the prototype system. The interface is built to add and remove running sensors for monitoring. The sensors generate random values to simulate real measurements. The smart city client displays the received values in time series graphs. To illustrate this feature, an example of the user interface is shown in Figure 8.

D. Limitations and Future Work

The proposed framework and implemented prototype have overlooked several aspects of building an experimental environment. The system is still in the early stages of development, and a few remaining topics for future work need to be discussed. In the following, we discuss such challenges:

1) Authentication Host Validation: The current client module is vulnerable to security breaches as it responds to every maintain and logout message, regardless of its origin. Any website can load the module into an iframe and steal a user's ID-token. Only trusted application domains registered by the SSO should be allowed to communicate with the iframe to prevent such an issue.

2) ID-token Scoping / Flow: The ID-token in the current implementation poses a risk of universal authentication. If an attacker steals one ID-token, they can access all the victim's applications. The solution is to add scopes to ID-tokens, allowing only certain combinations of ID-token scopes and secrets. This approach provides better control over the services accessing the SSO.

3) Forced ID-token Invalidation: An attacker who steals an ID-token can use it indefinitely under current systems. Even a maximum refresh count can provide enough time for significant damage. To prevent this, a timestamp of the user's last logout can be kept in the database. During authorization, the issuing time of the ID-token can be compared to the last logout time. Tokens issued before that time are rejected as authentication proof.

4) Policy Model Enhancement: The current policy model and rule syntax enable administrators to define authorization scenarios only in a specific "AND"-based logic manner. Consequently, scenarios can occur where the current model makes achieving a complex policy structure cumbersome. Therefore, extending the possibilities of policy construction plays a significant role in further development.

5) Threat Intelligence: The presented system is a prototype and a proposal for a functional ZT environment. Learning-based assessment using clearly defined dynamic attributes is one of the most important parts that is up to future work for creating a modern ZT ecosystem. This should be followed by assessment strategies that enable effective policy definition to detect malicious actors. ZT focuses on assessing users and entities acting as the smallest entity that can be controlled individually. Therefore, we must build the information required for authorization evaluation and behavior analysis around the smallest individual in the system. This comes with an advantage that is crucial for future implementations. An attribute-based approach generates descriptive data structures that can be adapted for learning techniques to connect modern technologies to the system. To give an example, this architecture allows us to add another attribute collection called environment to an authorization request, holding information like the subject's operating system. The SSO-Service can pass this collection to a machine learning algorithm to add another collection called rating, holding the algorithm outputs as attributes. This collection can then be evaluated against the known policy structure.

6) System Evaluation: Considering the possibilities for the proposed architecture, evaluating the system's applicability and how well it performs compared to current approaches in use is crucial. As this system is still in its infancy, applicability tests and performance measurements are considered for future development. The current architecture policy evaluation remains the most intensive computing task considering the architecture's applicability. When enhancing the system with threat intelligence components, it is crucial to perform performance investigations of state-of-the-art technologies.

VI. Conclusion

Digital environments are becoming more complex revealing the limitations of traditional security models. Meanwhile, the ZT paradigm provides a more adaptable and robust security framework compared to traditional security models. Therefore, our paper presented a ZT-based Single Sign-On (SSO) framework with Attribute-Based Access Control (ABAC). The developed prototype demonstrates the proposed framework's feasibility and potential applicability within a smart city context. Nevertheless, this paper represents only a small step toward operationalizing the framework. In future work, we plan to provide a comprehensive technical framework, encompassing both technical and architectural aspects.

Acknowledgment

Research has partially received funding from grant agreement No. 101079214 (AIoTwin) and grant agreement 101135576 (INTEND).

References

[1] S. Dustdar and I. Murturi, "Towards distributed edge-based systems," in *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, pp. 1–9, IEEE, 2020.

[2] P. K. Donta, I. Murturi, V. Casamayor Pujol, B. Sedlak, and S. Dustdar, "Exploring the potential of distributed computing continuum systems," *Computers*, vol. 12, no. 10, p. 198, 2023.

[3] I. Murturi and S. Dustdar, "Decent: A decentralized configurator for controlling elasticity in dynamic edge networks," *ACM Transactions on Internet Technology (TOIT)*, vol. 22, no. 3, pp. 1–21, 2022.

[4] C. Buck, C. Olenberger, A. Schweizer, F. Völter, and T. Eymann, "Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust," *Computers & Security*, vol. 110, p. 102436, 2021.

[5] L. Golightly, P. Modesti, R. Garcia, and V. Chang, "Securing distributed systems: A survey on access control techniques for cloud, blockchain, iot and sdn," *Cyber Security and Applications*, p. 100015, 2023.

[6] N. F. Syed, S. W. Shah, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss, "Zero trust architecture (zta): A comprehensive survey," *IEEE Access*, 2022.

[7] V. Stafford, "Zero trust architecture," *NIST special publication*, vol. 800, p. 207, 2020.

[8] X. Yan and H. Wang, "Survey on zero-trust network security," in *Artificial Intelligence and Security: 6th International Conference, ICAIS 2020, Hohhot, China, July 17–20, 2020, Proceedings, Part I 6*, pp. 50–60, Springer, 2020.

[9] V. Casamayor Pujol, P. K. Donta, A. Morichetta, I. Murturi, and S. Dustdar, "Distributed computing continuum systems–opportunities and research challenges," in *International Conference on Service-Oriented Computing*, pp. 405–407, Springer, 2022.

[10] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, pp. 1–54, 2013.

[11] P. Sun, S. Shen, Y. Wan, Z. Wu, Z. Fang, and X.-z. Gao, "A survey of iot privacy security: Architecture, technology, challenges, and trends," *IEEE Internet of Things Journal*, 2024.

[12] I. Murturi, P. K. Donta, V. C. Pujol, A. Morichetta, and S. Dustdar, "Learning-driven zero trust in distributed computing continuum systems," in *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 0044–0049, IEEE, 2023.

[13] C. Zhang, J. He, B. Fan, Y. Gong, S. Li, B. Yin, and Y. Lin, "Tag-based trust evaluation in zero trust architecture," in *2022 4th International Academic Exchange Conference on Science and Technology Innovation (IAECST)*, pp. 772–776, IEEE, 2022.

[14] W. Yunanto and H.-K. Pao, "User behaviour risk evaluation in zero trust architecture environment," in *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, pp. 1–6, IEEE, 2022.

[15] C. Bicer, I. Murturi, P. K. Donta, and S. Dustdar, "Blockchain-based zero trust on the edge," *arXiv preprint arXiv:2311.16744*, 2023.

[16] F. A. Qazi, "Study of zero trust architecture for applications and network security," in *2022 IEEE 19th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*, pp. 111–116, IEEE, 2022.

[17] V. Radha and D. H. Reddy, "A survey on single sign-on techniques," *Procedia Technology*, vol. 4, pp. 134–139, 2012.

[18] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services," in *2012 IEEE Symposium on Security and Privacy*, pp. 365–379, IEEE, 2012.

[19] A. Witkovski, A. Santin, V. Abreu, and J. Marynowski, "An idm and key-based authentication method for providing single sign-on in iot," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2015.

[20] I. Murturi, A. Egyed, and S. Dustdar, "Utilizing ai planning on the edge," *IEEE Internet Computing*, vol. 26, no. 2, pp. 28–35, 2022.

[21] B. Sedlak, I. Murturi, P. K. Donta, and S. Dustdar, "A privacy enforcing framework for data streams on the edge," *IEEE Transactions on Emerging Topics in Computing*, 2023.

[22] A. Bouguettaya, M. Singh, M. Huhns, Q. Z. Sheng, H. Dong, Q. Yu, A. G. Neiat, S. Mistry, B. Benatallah, B. Medjahed, *et al.*, "A service computing manifesto: the next 10 years," *Communications of the ACM*, vol. 60, no. 4, pp. 64–72, 2017.

[23] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, 2010.

[24] K. Ragothaman, Y. Wang, B. Rimal, and M. Lawrence, "Access control for iot: A survey of existing research, dynamic policies and future directions," *Sensors*, vol. 23, no. 4, p. 1805, 2023.

[25] J. Qiu, Z. Tian, C. Du, Q. Zuo, S. Su, and B. Fang, "A survey on access control in the age of internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4682–4696, 2020.