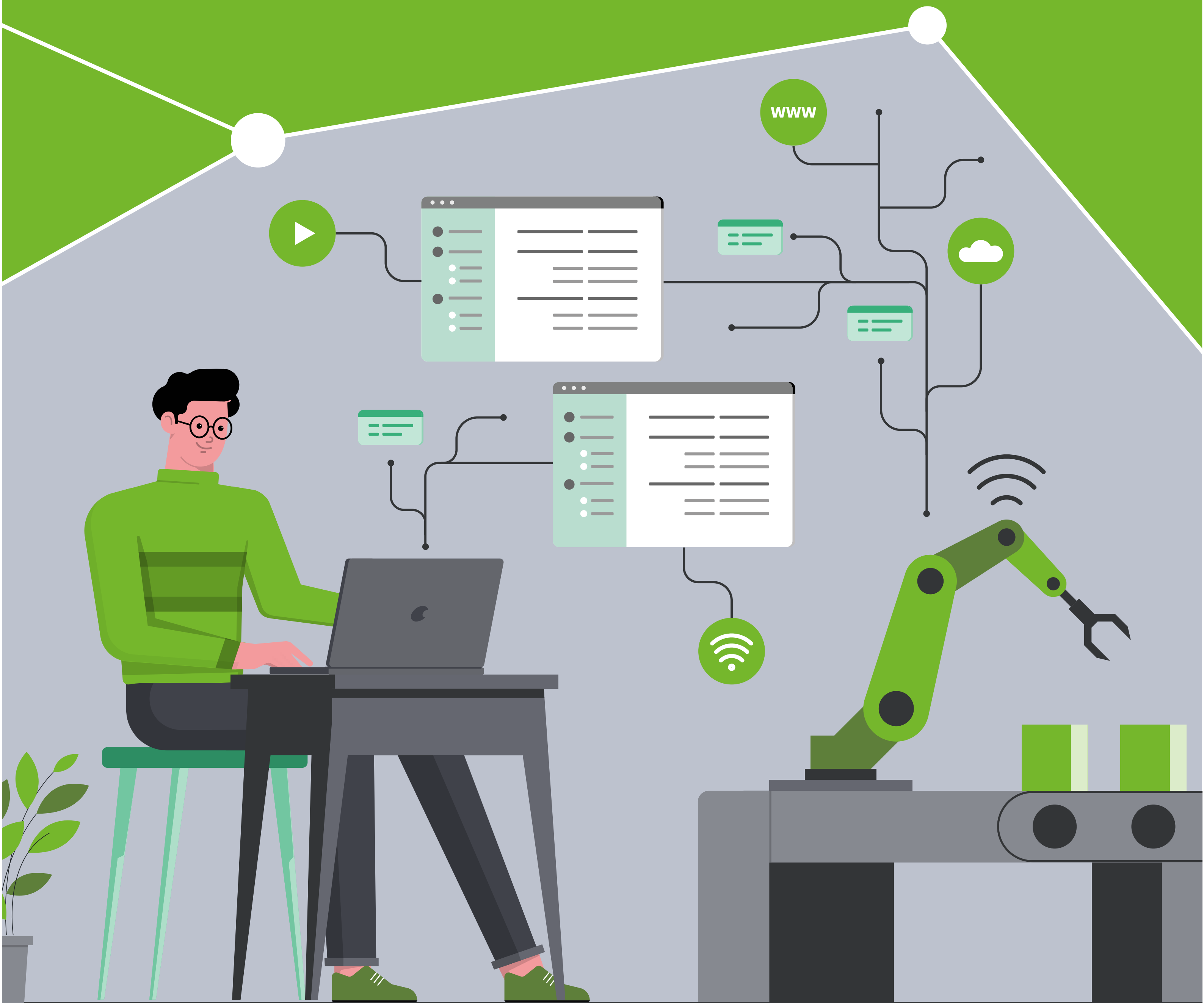
OPEN INDUSTRY 4.0 ALLIANCE

Best Practice Guide

AAS Security Specifications & Beyond



IMPRINT

Publisher

Open Industry 4.0 Alliance

Christoph Merian-Ring 12, 4153 Reinach, Switzerland

https://openindustry4.com

info@openindustry4.com

Status

File: AAS Security Specifications & Beyond

August 18th, 2025 – Version 1

Editors

Lucas Wolf (Open Industry 4.0 Alliance)

Autors

Alain Tenkeu (Murrelektronik GmbH)
Christina Borntraeger (Meta-Level Software AG)
Johanna Kiess (DXC Technology Deutschland GmbH)
Markus Rentschler (ARENA2036 e.V.)
Michael Hofmann (cdmm GmbH)
Philip Sehr (TH-OWL)
Vitas Kling (AMETEK GmbH)

Abstract

This document describes a standardized approach to secure access to the Asset Administration Shell (AAS) in accordance with IEC 63278. The AAS is a machine-readable representation of a product that enables the exchange of structured product-related information across systems and organizational boundaries. As AAS usage expands into environments with many participants, a consistent security concept is required to ensure interoperability, data integrity, and controlled access.

The paper focuses on the mechanisms defined in IDTA-01004 for implementing confidentiality and integrity protection without modifying the AAS itself. Access policies are stored externally in dedicated repositories and applied through a technology-neutral rule model. Attribute-Based Access Control (ABAC) provides the basis for context-dependent decisions, complemented by support for federated and organization-specific identity providers.

The content is structured around four main aspects: reference scenarios that define the application context; the concepts of access levels and access rules; authentication and identity provisioning based on established protocols; and infrastructure options for cross-company identity management. Implementation considerations, open challenges, and recommendations for adoption are also discussed.

The target audience includes system architects, product developers, and service providers involved in AAS-based data exchange. This document aims to provide both the conceptual framework and practical guidance needed to deploy secure AAS infrastructures in industrial environments.

Content

Introduction

The Asset Administration Shell (AAS) is a standardized and machine-readable virtual representation of a physical or digital asset in accordance with IEC 63278. It is used for the interoperability of Industry 4.0 products and systems. With the AAS, all asset-related information, such as the digital nameplate, manuals, certificates or technical data, can be stored, updated and exchanged across stakeholder boundaries throughout the entire product life cycle.

Security considerations in the context of AAS have become increasingly central to their deployment, especially within data spaces consisting of thousands of participants. Consistent implementation across systems, organizations, and technologies requires a uniform security concept. Without it, access decisions remain fragmented, leading to interoperability issues and reduced trust in exchanged data. To address this, access control must follow a standardized approach based on well-established security principles which define how sensitive information is protected and how the integrity of data is maintained. The main goals of this approach are the protection of confidentiality and the assurance of integrity in the handling of AAS content.

Confidentiality protection means that only authorized parties shall be able to access sensitive information managed by the AAS. This applies both to actual values (properties) and to the structure and metadata represented by the AAS, including the topology described by Submodels, SubmodelElements. Integrity protection requires that only authorized parties shall be able to enter or modify data in an AAS. Mechanisms should enable the detection of unintentional changes and allow the originator of the data to be clearly traced.

To enable confidentiality and integrity protection, IDTA-01004 defines a concept in which user access restrictions can be configured without modifying the AAS itself. Access policies are stored externally in dedicated policy repositories, separate from the AAS instances to which they apply. This separation ensures that data structures remain unchanged while access logic can be maintained and updated independently. The concept relies on a technology-neutral model expressed in Backus-Naur Form (BNF), together with a corresponding JSON schema. These elements are used for both access rule definitions and the query language in HTTP API version 3.1. The approach supports attribute-based access control (ABAC) and is compatible with both federated and organization-specific identity providers. Access decisions are evaluated through data filtering before information is returned to the requester. While this mechanism primarily targets AAS Type 2 implementations, it can also be applied during the creation of AAS Type 1 files. The chapters that follow describe how these elements are applied in practice. Section 1 defines the application scenario context. Section 2 introduces the basic concepts of access levels and the structure of access rules. Section 3 focuses on authentication, token formats, and identity provisioning based on established protocols. Section 4 discusses scenarios for cross-company identity handling, including API gateway integration.

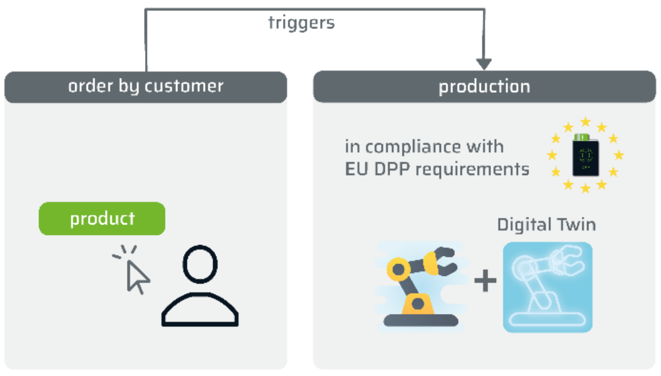
In summary, the following topics are addressed in this paper:

1) The access levels and rules related to a certain successful authentication (see chapter Access Control Concepts for the AAS).
2) The authentication and authorization of users or systems. This raises the question of who can authenticate themselves and how, and what additional information needs to be provided for authentication (see chapter Authentication).
3) The infrastructure used. This involves how authentication is regulated (centralized or decentralized) and which combinations are possible (see chapter Identity Provider (IdP)).

1 Reference Scenarios: Defining the Context for Access Management

To define the scope and direction of this paper, we also describe the following representative B2B & B2C scenarios that serve as a reference for the security requirements addressed in the following chapters.

1.1 B2B Use Case



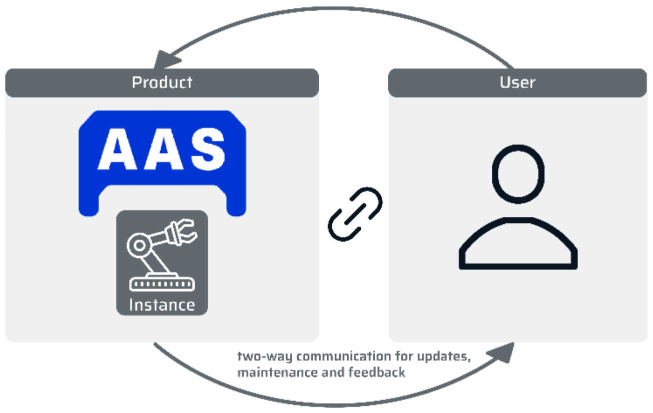
A customer orders a product directly from the manufacturer, for example via an online shop. During this process, all relevant customer data is collected, including name, address, product configuration, and, where applicable, individual customization requests. Once the order is completed, production begins. In parallel to manufacturing, a digital twin of the product is created. The digital twin is a structured digital representation of the physical product and includes information such as material composition, CO_2 footprint, maintenance instructions, and additional technical details.

When the product is shipped, the digital twin is published on an AAS platform. This process follows the requirements of the digital product passport (DPP), which is mandated as part of European sustainability initiatives. The DPP requires certain product data to be made publicly available to ensure transparency along the supply chain and to support circular economic goals.

Since not all information is intended for public access, a differentiated access control concept is applied. Public data such as the digital nameplate, environmental information, or instructions for disposal and recycling is accessible to everyone. In contrast, customer-specific or sensitive data, including maintenance history or individual configurations, is restricted to authorized users.

Access rights are managed through a central authorization system. This ensures that each user group can only view the information relevant to their role. The scenario described here forms the starting point for the following technical sections.

1.2 B2C Use Case



In a B2C context, the use of an AAS infrastructure creates new possibilities for structured interaction between end users and manufacturers. One example is product registration, which can serve as the basis for a feedback channel from the customer to the supplier. Traditionally, this form of interaction is difficult to establish due to the lack of standardized product identifiers, inconsistent data availability after the point of sale, and disconnected communication channels.

Through registration, the end user links themselves to a specific AAS instance of the product. This allows the supplier to provide usage-specific updates, maintenance information, or technical notices via the AAS. At the same time, users can submit feedback or error reports in a structured format that aligns with the AAS model. This enables consistent data exchange tied directly to the product configuration and lifecycle.

To ensure secure operation, appropriate access control mechanisms are required. While general information may remain publicly accessible, any exchange of usage data or personal information must be limited to authorized participants. This scenario illustrates how AAS infrastructures can enable data-driven services even in decentralized consumer environments.

2 Access Control Concepts for the AAS

Access control in the Asset Administration Shell is structured through a combination of layered access levels, attribute-based decision-making, and a formalized rule model. The concept of access levels defines the granularity at which access can be regulated, from the entire asset to individual attributes. ABAC provides the underlying mechanism for making context-based decisions. These are expressed using the AAS access rule model, which applies a standardized syntax for defining and evaluating access rules across different system environments.

2.1 Type AAS & Instance AAS

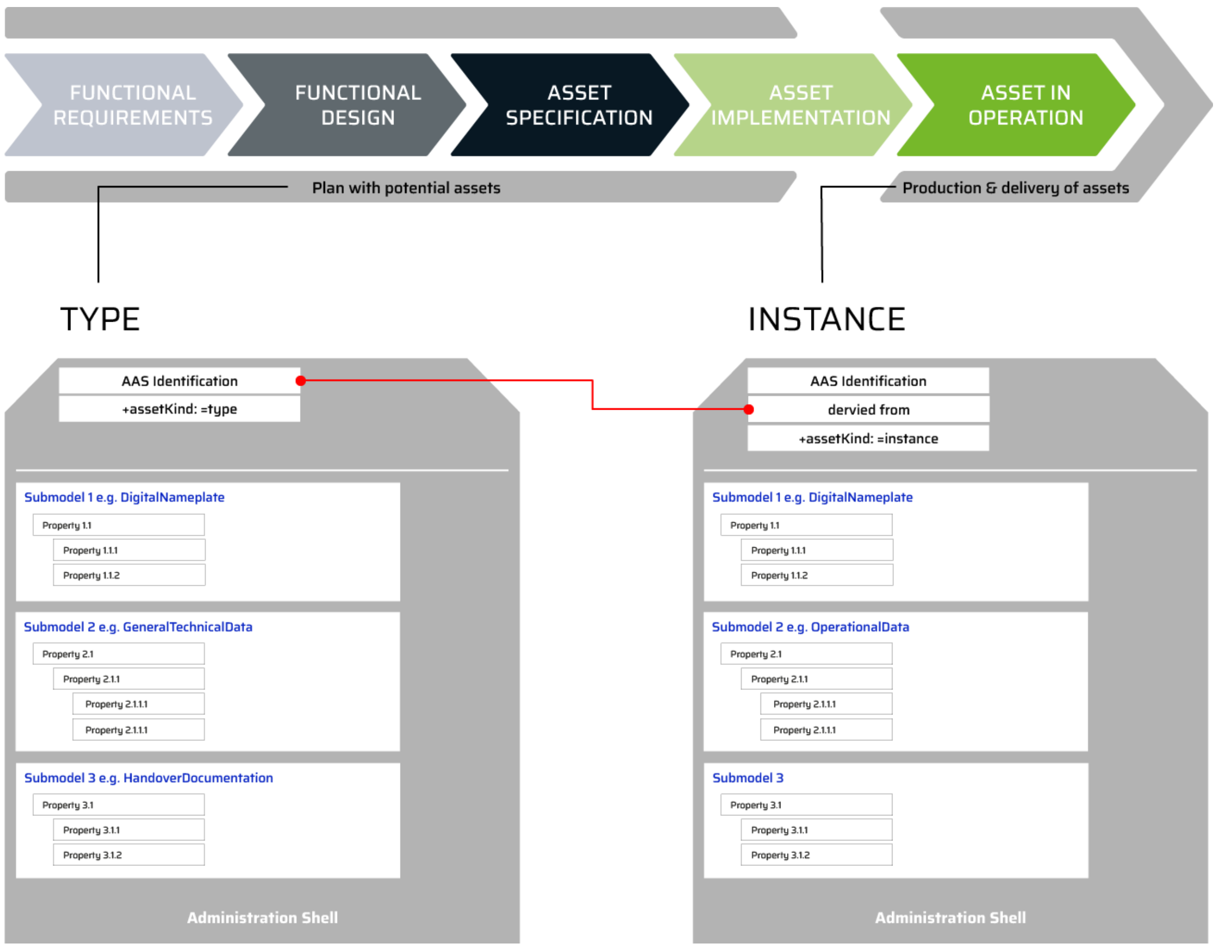
In practice, Asset Administration Shells can be distinguished into type-level and instance-level representations. This separation reflects the difference between generic product information and data specific to an individual asset. Each of these representations follows the same structural principles but differs in scope, data content, and access requirements.

2.1.1 Type AAS

The Type AAS represents general, reusable information about a product type or class, thus can be valid for all serial numbers of a product. It contains static data that applies across multiple instances of the same product, such as technical specifications, standard materials, or design parameters. Within a company, this may also include internal submodels or submodel elements that are not intended for external sharing. These private elements are relevant for internal development, production planning, or variant management and are typically subject to restricted access control.

2.1.2 Instance AAS

The Instance AAS refers to the specific digital representation of a single, manufactured asset with a specific single serial number. It references a Type AAS and includes instance-specific data such as serial numbers, individual configurations, production history, and operating parameters. It can contain private submodels or submodel elements, for example, maintenance logs or customer-specific settings, which are only made available to authorized users.



2.1.3 Public Type and Product Information

A Type AAS can be valid for all serial numbers of a product. An AAS created for a single serial number is referred to as an Instance AAS. The overlapping area between Type and Instance contains public product information that is relevant to both representations. This includes submodels or submodel elements intended for open access, such as digital nameplates, environmental impact data, or instructions for recycling and disposal. These shared elements enable transparency and interoperability across systems and stakeholders. For example, the submodel DigitalNameplate contains both Type and Instance information.

2.2 Access Levels

The AAS plays a central role in Industry 4.0 by enabling a standardized digital representation of assets. To ensure secure and controlled application of this concept, a differentiated access control mechanism is required. This mechanism is structured into distinct levels: the asset level, the submodel level, and the attribute level (AAS property).

This layered structure supports a precise authorization process and contributes directly to the protection of AAS-managed data.

2.2.1 Asset Level

At the asset level, access to the entire AAS is regulated. This level defines whether a user or system can interact with an AAS at all and therefore represents the highest level of access control. Mechanisms such as authentication and authorization are applied to enforce this restriction. Only verified users or systems are granted access. Attribute-based Access control (ABAC), and in simpler cases role-based access control (RBAC), can be used to assign permissions. For example, a maintenance technician may require access to the AAS of a machine, while an external service provider may not receive such access.

2.2.2 Submodel Level

The submodel level enables more detailed access control to specific data segments within an AAS. AAS instances can contain various submodels, each representing a functional or informational aspect of the asset, such as technical parameters, operational data, or maintenance records. Access to these submodels can be assigned selectively, depending on the user role or task. Not every user should be able to view or modify every submodel. Digital signatures and hashing techniques can be used to ensure integrity. For instance, a production manager may be authorized to access the "operational parameters" submodel but not the "maintenance history", which is restricted to technical personnel.

2.2.3 Attribute Level

The attribute level, also referred to as the submodel element level, defines access control at the most granular resolution. It determines whether individual properties or collections within a submodel are visible or editable. Applying access rules to submodel element collections allows partial restriction within a single submodel. This reduces configuration complexity and helps avoid errors. Depending on the sensitivity of the data, access can be differentiated into read, write, or restricted modes, especially for external users. For example, a worker may be allowed to read machine temperature and speed but not the serial number or license information, which are reserved for administrators.

In addition to access control, encryption can be applied at all levels to protect sensitive content. Only authorized users can decrypt and use this data. The combination of asset level, submodel level, and attribute level access structures allows implementation of robust security mechanisms for the AAS. When supported by access control methods such as ABAC or RBAC, and complemented with encryption and digital signatures, a consistent and high level of protection for integrity, availability, and confidentiality in industrial systems can be achieved.

2.3 Attribute-Based Access Control

Attribute-Based Access Control (ABAC) is an access control method that grants or restricts permissions based on attributes of the user, resource, action, and context, rather than fixed roles. Part 4 of the IDTA specification defines an access rule model that is based on the principle of ABAC. In this model, access requests are evaluated based on attributes associated with the subject, the object, environmental conditions, and a set of defined policies. Authorization is granted or denied depending on whether these attributes meet the specified conditions. This approach allows for flexible and context-aware access control.

The ABAC model follows the structure outlined in the OASIS XACML specification It includes the following system components:

- Policy Administration Point (PAP): defines and manages the set of access control policies.
- Policy Decision Point (PDP): evaluates access requests against applicable policies and issues authorization decisions.
- Policy Enforcement Point (PEP): enforces authorization decisions by controlling access to protected resources.
- Policy Information Point (PIP): provides the attribute values required for policy evaluation.

ABAC also encompasses RBAC, since roles can be treated as one type of attribute. Other attributes may include time of day, geographic location, or origin of the request. These attributes can be expressed as claims and used directly within access rules. Access policies can be applied to API routes defined in IDTA-01002 or to specific AAS elements such as Identifiables and Referables, either by direct reference or through their semantic identifiers.

2.4 AAS Access Rule Model

The AAS access rule model uses technology-neutral grammar based on Backus-Naur Form (BNF) to represent ABAC, where attributes are used in access rules. RBAC can also be implemented by defining role attributes. Subject attributes and roles may be provided as claims in access tokens.

The implementation of access control is performed in the resource server, which evaluates the transferred attributes (claims). This mechanism checks access permission rules that include constraints related to subject attributes, object attributes, and environment conditions. After evaluation, a decision is taken and enforced upon the object, for example, access to a submodel element is either permitted or declined.

If the claims match the data provider's expectation for a particular policy constraint, the system returns information for accessing the AAS resources. This typically includes the URL, a short-living access token, and a refresh token.

3 Authentication

Authentication refers to the process of verifying the identity of a user and forms the initial step for enforcing controlled access to data via web applications and services. With the increasing number of web-based applications, authentication procedures are executed frequently and often in parallel. This leads to operational inefficiencies and security challenges, particularly when users manage multiple credentials across different platforms. In many cases, users reuse or slightly modify passwords, which introduces risks. The use of more secure authentication methods, such as two-factor authentication, further increases user effort and can reduce efficiency. Additional risks arise when user credentials are processed or stored across multiple systems, increasing the potential for breaches.

Single sign-on (SSO) addresses these challenges by allowing users to access multiple applications with a single set of credentials. Once authenticated, users can interact with all authorized services without repeated login actions. SSO thereby improves usability and mitigates some of the risks associated with credential reuse and distributed identity handling.

Depending on the application context, different authentication protocols are used. In enterprise environments, Security Assertion Markup Language (SAML) is commonly applied to meet high security and flexibility requirements. For web-based applications, OpenID Connect (OIDC) is more widely adopted. OIDC extends OAuth 2.0 with an authentication layer and is suited for web scenarios. In the context of industrial data exchange along supply chains, secure and reliable user authentication is essential for protecting data integrity and confidentiality.

OIDC defines a process involving three actors: an Identity Provider (IdP), a User Agent (UA), and a Client Application (CA). The steps involved are as follows:

1. **User initiates login**: The user attempts to access a protected resource, such as an AAS, and selects an external IdP (e.g., Google or Microsoft) for login.

2. **Authorization request**: The CA redirects the UA to the IdP with parameters including response type, redirect URL, requested scope, and client ID. The scope defines which parts of the user's profile are requested.

3. **User authentication**: The user authenticates using credentials and consents to share the specified profile information with the CA.

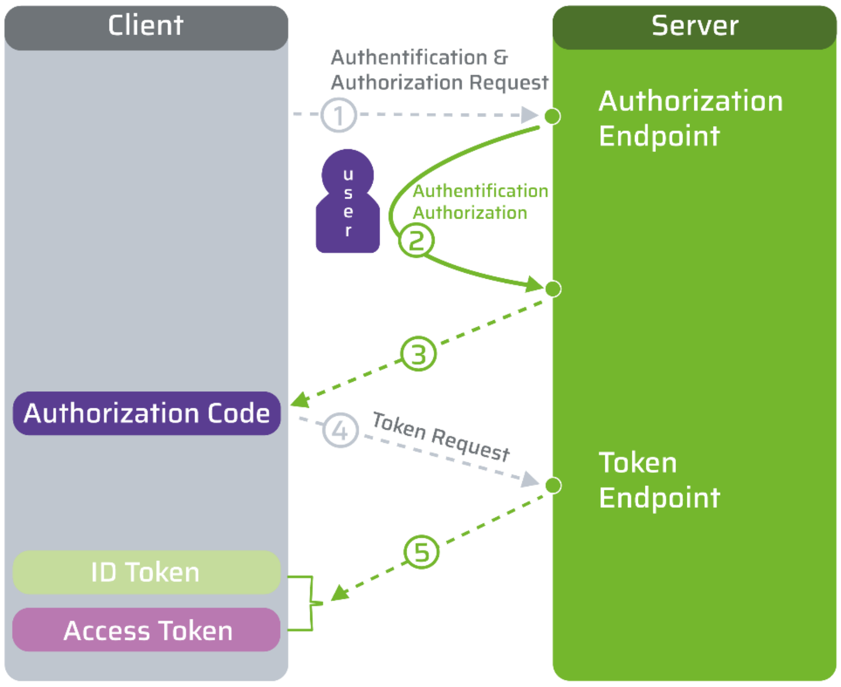
4. **IdP authorizes and redirects**: Upon successful authentication, the IdP authorizes the user and redirects the UA back to the CA.

5. **Token exchange**: The CA uses the received authorization code to request tokens from the IdP's token endpoint.

6. **Token response**: The IdP responds with an ID token (JWT) and optionally an access token. The CA then validates the token structure, IdP signature, and claim values.

This token-based authentication mechanism ensures that the identity of the user is reliably established before granting access to AAS resources.

response_type=code (scope includes openid)



In this flow, the authorization request specifies the parameter response_type=code. This indicates that the client is using the Authorization Code Flow, in which an authorization code is returned and later exchanged for tokens. By including scope=openid, the request explicitly follows the OpenID Connect standard, enabling the issuance of both ID tokens and access tokens.

The ID token is issued as a JSON Web Token (JWT) and contains verified claims about the user, such as name or e-mail address, as specified during the authorization request. It also includes information about the issuer and the Client Application. The access token, also issued as a JWT, is used to access protected resources.

OpenID Connect supports multiple flows for the token exchange. The authorization code flow, as shown above, is the most widely used due to its security properties and general suitability for web applications. In contrast, the implicit flow allows direct issuance of access and ID tokens from the Identity Provider without client authentication. The hybrid flow supports a combination of both approaches.

3.1 Claims, Scopes & Roles

When securing access, the following relationships and attributes are generally considered:

- Comparisons with other attributes
- Comparison with set values (e.g. a date "released from")
- Comparison with claims from the access tokens (e.g. a specific role)

All three aspects are used by policies and in the newer IDTA Security Specification. The use of attributes is quite self-explanatory. The use of claims, however, requires closer examination.

The necessary data for verification is typically provided through the Identity Provider in the form of the Json Web Token, which usually consists of a header, content, and a signature. While the header and signature of a JWT are largely standardized, the content contains some mandatory fields but can also be extended individually. The key terms are 'claims', 'scope', and optionally 'roles':

3.1.1 Claims

Claims are statements that drive authorization decisions in the form of key-value pairs that describe a specific attribute, property, or right associated with a subject. To be more precise, a claim is a statement by the issuer of a certificate that the value represents a valid property assigned to the token holder. For example, an attribute 'name=Mustermann' is a claim. The definition of roles (e.g. 'role = {Admin}') is also considered a claim.

3.1.2 Scopes

Scopes are groups of claims. OpenID Connect, for instance, defines the scope 'email', which specifies that the attributes 'email' and 'email_verified' must be included in the issued token. However, scopes can also be a type of permission. A 'read' scope could mean that the user has access to all APIs that read data. Therefore, the application logic also directly affects the scopes, as these must implement the corresponding security rules.

3.1.3 Roles

Roles is an optional term. They can be included as a claim in a JWT.

Examples of properties/claims available in an access token:

- **iss (Issuer):** Identifies the authorization server that issued the token.
- **sub (Subject):** Identifies the entity that the token refers to (e.g. a user, a client or a device).
- **aud (Audience):** Identifies the resource server for which the token is intended.
- **exp (Expiration Time):** Specifies the expiration time of the token in seconds since the Unix epoch (January 1, 1970 00:00:00).
- **nbf (Not Before):** Specifies the time before which the token is not valid, in seconds since the Unix epoch.
- **iat (Issued At):** Specifies the time at which the token was issued, in seconds since the Unix epoch.
- **jti (JWT ID):** A unique identifier for the token, typically generated as a unique string.

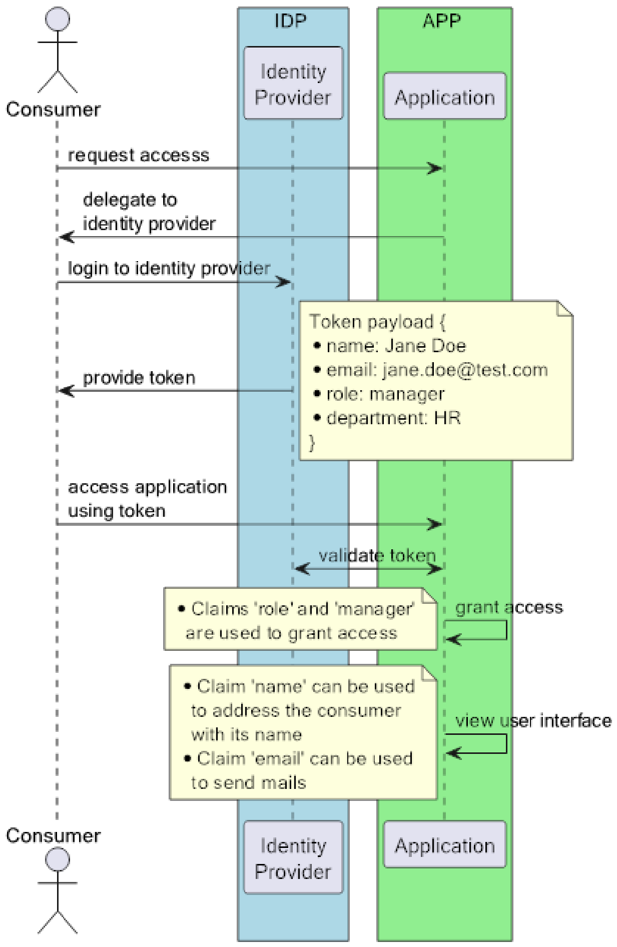
An Identity Provider can define different conditions for a scope; the most important ones are:

- **Optional:** The user must explicitly select this option when logging in. If the user does not make an explicit selection, they have no access to the associated data or functions. The advantage is higher data security, as the user has control over their access rights.
- **Default:** These scopes are automatically granted when a user logs in. The user does not need to explicitly select them. Default scopes are often required for basic functions, such as "profile" for displaying the user profile. The advantage is a simpler login process and the immediate availability of basic functions.

- **Required**: If a scope is defined as required, the inclusion of this scope in the request is mandatory to access the service at all. If this scope is not included in the request, no access token (or the required permissions) will be issued.

As an example in the following chapters, we assume the Identity Provider sends a token with the following payload to consumer:

- name: <the user name>
- email: <the user email>
- role: <the access role of the user>
- department: <the department the user is located>



Consumer
IDP
Identity Provider
APP
Application
request accesss
delegate to identity provider
login to identity provider
Token payload {
• name: Jane Doe
• email: jane.doe@test.com
• role: manager
• department: HR
}
provide token
access application using token
validate token
• Claims 'role' and 'manager' are used to grant access
grant access
• Claim 'name' can be used to address the consumer with its name
• Claim 'email' can be used to send mails
view user interface
Consumer
Identity Provider
Application

This diagram shows a claims-based authentication process in which a user is redirected to an Identity Provider (IdP), receives a token with verified claims (e.g., name, email, role, department), and presents it to the application. The application validates the token, uses claims like role and department for access control, and may use others for personalization or communication, avoiding additional user data queries.

See also examples of access rules in JSON serialization.

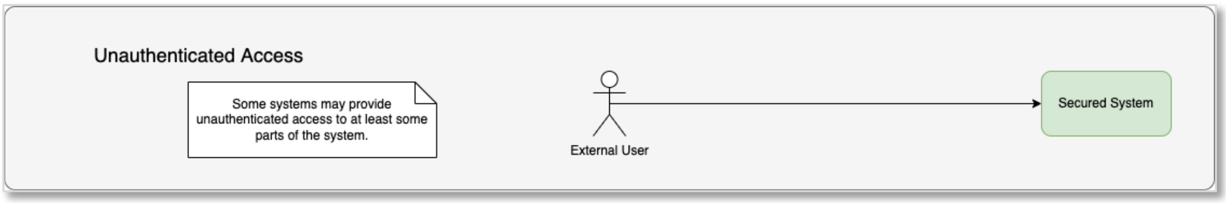
4 Identity Provider (IdP)

An Identity Provider (IdP) is responsible for issuing authentication information in the form of an access token, typically structured as a JSON Web Token (JWT). Access tokens are applied to all AAS HTTP/REST APIs, with specific relevance for both repository and registry services to enable access to resources such as models and submodels.

There are different variants of centralized and decentralized identity providers possible. Some options are considered below.

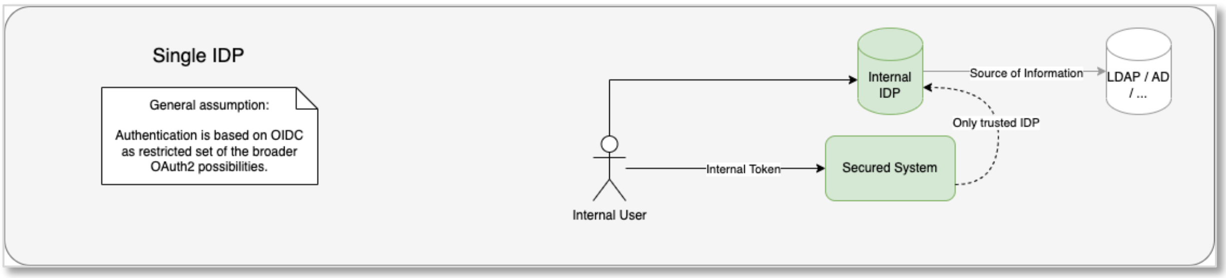
4.1 No Authentication/Authorization

If there is no authentication, the data marked as public by the data provider can still be retrieved.


Unauthenticated Access
Some systems may provide unauthenticated access to at least some parts of the system.
External User
Secured System

4.2 Internal Authentication/Authorization

The identity provider is aligned with the resources of the data provider.



4.3 External and Internal Identity Providers

In this setup, a central external identity provider is responsible for user authentication in collaboration with the internal IdPs.

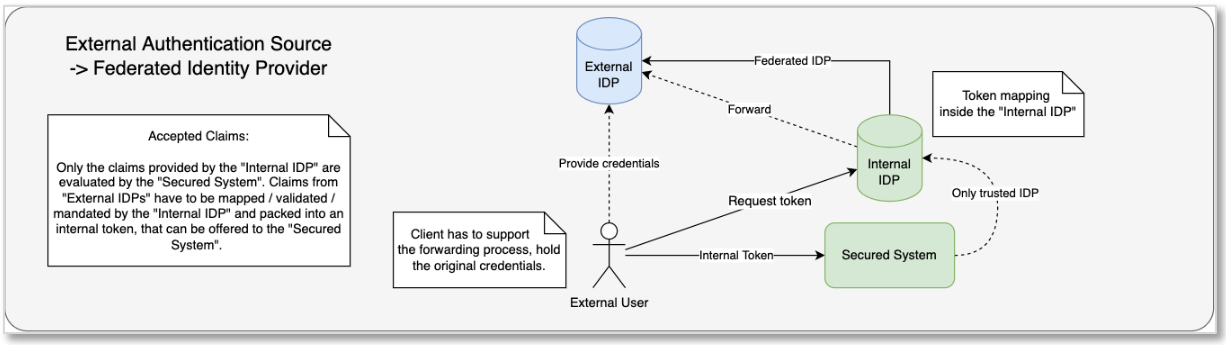
The following steps are taken:

1. The external user obtains a token from the external identity provider.

2. The external user uses the token from the external identity provider to identify himself to the internal identity provider of the data provider.

3. The internal identity provider of the data provider checks the token from the external identity provider and issues the internal access token.

4. The external user then uses the internal access token to identify with the data provider.



Accepted Claims:

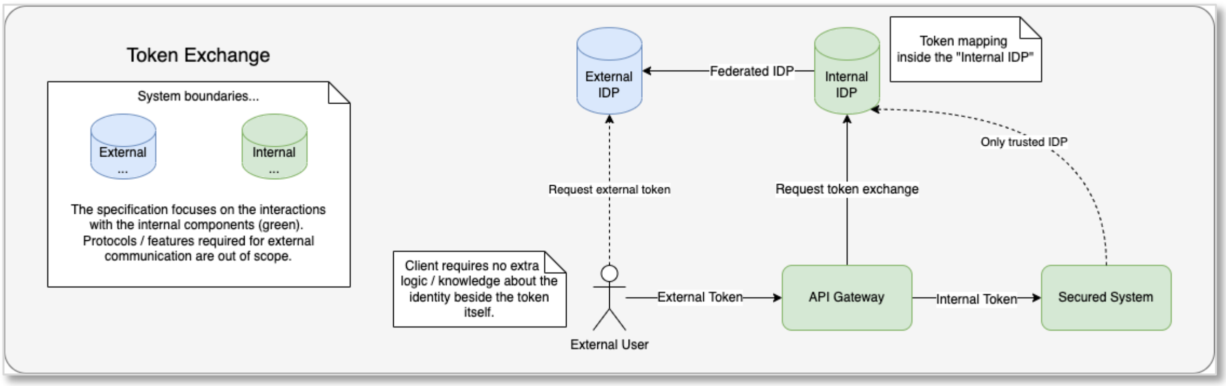
Only the claims provided by the "Internal IDP" are evaluated by the "Secured System". Claims from "External IDPs" have to be mapped / validated / mandated by the "Internal IDP" and packed into an internal token, that can be offered to the "Secured System".

Client has to support the forwarding process, hold the original credentials.

One disadvantage is that external users require a separate token for each data provider and must log in to each data provider again. However, the option of continuing to use the token received from the external identity provider eliminates the need for re-authentication.

4.4 Gateway

To counteract the flood of possible access tokens, the use of a gateway can be useful. The external user still requires the token of the external identity provider, which is used to log in to the gateway of the data provider. The gateway contacts the internal identity provider and hands over the external token. If the external token could be validated, the internal identity provider generates an internal token that is used to access the data provider's resources.



The specification focuses on the interactions with the internal components (green). Protocols / features required for external communication are out of scope.

Client requires no extra logic / knowledge about the identity beside the token itself.

For each further access, the system checks whether an internal token already exists for the external token. In this case, it is no longer necessary to generate a new token via the internal identity provider.

The advantage of this approach is that the external user only needs to know the token that was received during authentication with the external identity provider.

5 Conclusion

This document outlined how access to Asset Administration Shells can be managed through standardized mechanisms for authentication, identity handling, and access control. The concept of IDTA-01004 is based on separating access policies from the AAS content itself and applying them consistently through structured rule evaluation.

When applying security rules, required elements within submodels must remain accessible. In case such fields are inaccessible, the AAS becomes unusable, and interoperability is lost, since automated systems cannot validate incomplete submodels. To avoid this, all required elements within AAS should either be openly accessible or consistently excluded. Mixed configurations create uncertainty and should be avoided.

Access activities should be logged wherever possible. This includes both successful and rejected requests. Such data can help evaluate whether the current access configuration is appropriate and support later adjustments if needed.

Access control in AAS environments must be reliable and predictable. Overcomplicated configurations reduce transparency and increase the risk of incorrect behaviour. A clear and well-documented rule set helps maintain system integrity and ensures that AAS remains usable in industrial applications.

Optional features such as time-based access should only be used with caution. The increase in complexity for rule evaluation may introduce dependencies that are difficult to control in distributed or international environments. In most cases, they are probably not even necessary.

6 Open Challenges

Several unresolved issues remain regarding the practical implementation of access control in AAS-based environments. One recurring challenge is the use of ABAC policies in scenarios where necessary attributes for decision-making are not part of the submodel itself. For example, the “globalAssetId” is often used to determine access rights, but this attribute is defined at the AAS level. In distributed system setups, where AAS and submodels are hosted in different systems or components, this separation complicates access control logic. It requires synchronized availability of contextual information across components, which increases implementation effort and the need for standardization.

Another challenge is the performance of authorization checks. While evaluating access rules for single AAS instances or submodels is manageable, processing large lists such as search results can create significant overhead. Each result may require separate rule evaluation, especially in setups with complex access structures or high-frequency queries. Without optimized handling, this can reduce system responsiveness and limit scalability.

7 Outlook

The broader adoption of AAS security depends on how product and service providers implement and integrate the specified mechanisms into their existing environments. In many cases, identity and access management is already handled through internal systems such as LDAP or Active Directory. A practical security solution must be able to interface with these existing structures. Maintaining separate and isolated access control systems in parallel is not realistic for most companies.

Therefore, it is essential that providers of AAS-enabled solutions are directly involved in the design and refinement of security mechanisms. This includes understanding the technical consequences of implementing policy repositories, supporting token validation, or integrating federated identity systems. Especially in brownfield scenarios, where existing infrastructure cannot be replaced easily, the effort for integration must be considered early on. Otherwise, there is a risk that potential adopters disengage due to the perceived complexity or lack of compatibility.

To ensure a balanced progression, standardization must be developed in parallel with implementation experience. A rigid specification that is not validated by real systems will likely face resistance. Instead, feedback from those building and deploying AAS infrastructure must remain part of the process.

We, the Open Industry 4.0 Alliance, actively support the implementation of AAS security concepts in industrial environments. As these mechanisms move from specification to real systems, practical experience becomes essential. If you are working on similar challenges or have expertise in secure identity handling, access control, or system integration, we invite you to contribute. Join the ongoing work, share your perspective, and help ensure that these solutions remain applicable, efficient, and aligned with operational needs!