Grant Agreement No.: 101070473
Call: HORIZON-CL4-2021-DATA-01
Topic: HORIZON-CL4-2021-DATA-01-05
Type of action: HORIZON-RIA

Final Report

Revision: v.1.0

Subproject Name	XADATU
Authors	Rafael Marín Pérez (ODINS), Alejandro Arias Jiménez (ODINS)
Submission date	31/01/2025
Reviewers	FLUIDOS Internal

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	03/12/2024	1st version of the template for comments	Rafael Marín Pérez (ODINS), Alejandro Arias Jiménez (ODINS)
V0.2	02/01/2025	2nd version of the Final Report	Rafael Marín Pérez (ODINS), Alejandro Arias Jiménez (ODINS)
V0.3	23/01/2025	3rd version of the Final Report after UMU review	Rafael Marín Pérez (ODINS), Alejandro Arias Jiménez (ODINS)

Disclaimer

The information, documentation and figures available in this deliverable are written by the Beneficiaries of the "Flexible, scaLable and secUre decentralIzeD Operationg" (FLUIDOS) project's Financial Support to Third Parties scheme under EC grant agreement 101070473 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

Copyright notice

Funded by Horizon Europe Framework Programme of the European Union

Table of Contents

Funded by Horizon Europe Framework Programme of the European Union

LIST OF FIGURES

Funded by Horizon Europe Framework Programme of the European Union

LIST OF TABLES

Funded by Horizon Europe Framework Programme of the European Union

ABBREVIATIONS

ABAC	Attribute-Based Access Control
DLT	Distributed Ledger Technology
EdDSA	Edwards-Curve Digital Signature Algorithm
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
PAP	Policy Administration Point
PEP	Policy Enforcement Point
PDP	Policy Decision Point
PSM	Privacy and Security Manager
REAR	Resource Acquisition Manager
UMU	University of Murcia
VC	Verifiable Credential
VDR	Verifiable Data Registry
VP	Verifiable Presentation
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

Funded by Horizon Europe Framework Programme of the European Union

1 Introduction

This final report compiles the contributions and results of T.3 'Developing and Evaluating' of the XADATU Project for the FLUIDOS Open Call. The document serves as a self-contained overview of the progress made in deploying and validating the MVPs developed during T.2, emphasizing their alignment with FLUIDOS' needs, open-source principles, and performance requirements. Furthermore, T.3 has focused on the evaluation of authorization and accounting mechanisms (PAP, PEP, PDP) developed using XACML, ABAC, and DLT technologies within the GAIA-X framework and open-source stacks (e.g., Linux). The task has relied on defined KPIs and validation factors to assess the MVPs in real-world testbed sites, including OdinS and University of Murcia (UMU) (Spain). Feedback from UMU experts and validation outcomes have been critical in refining the T.2 designs.

The report also highlights the integration of the validated components into the FLUIDOS architecture, presenting the achieved TRL5 demonstration. This phase has marked a significant step toward achieving the project's strategic goals and ensuring the practical applicability of the developed solutions.

1.1 Overview and objectives of XADATU project

XADATU is a project that aims to create, execute, and include decentralized authorization and accounting for resources dispersed geographically within the GAIA-X federated architecture of the FLUIDOS project. Moreover, to address a priority issue in the FLUIDOS Open Call, the XADATU Project is a Technology Extension (TE) that will improve the FLUIDOS architecture, shown in *Figure 1*, by introducing decentralized authorization and accounting functionality. Therefore, in a highly dynamic and dispersed Edge/Cloud situation, XADATU seeks to provide a safe and scalable solution for resource access control and accounting by imposing domain-specific access control policies by the FLUIDOS Project's current implementation of the zero-trust paradigm. To summarise, ODINS will specifically make use of the Verifiable Data Registry (VDR) to expand the present features with the accountability of the access control activities via the storage in distributed ledgers, e.g., Hyperledger Fabric. A distributed authorization system based on standards such as the Attribute-Based Access Control (ABAC) [1] model and Extensible Access Control Markup Language (XACML) [2] policy language will also be implemented by ODINS by extending the Privacy and Security Manager (PSM).

Funded by Horizon Europe Framework Programme of the European Union

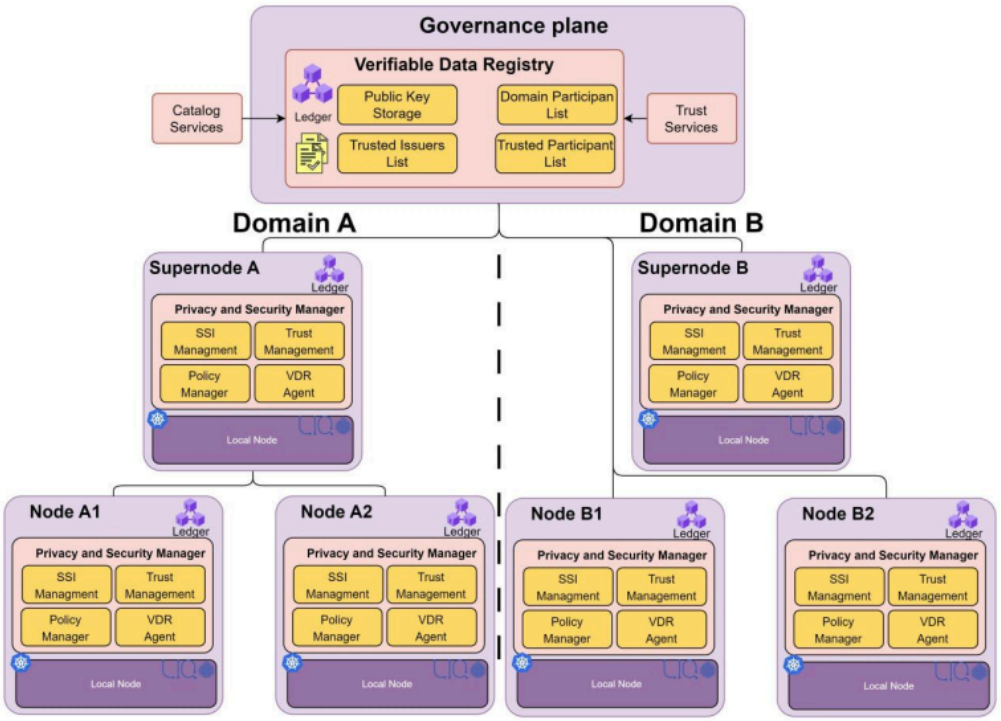

Figure 1 FLUIDOS Overall architecture

1.2 Relevance and benefits to FLUIDOS

The technical expansion of the FLUIDOS architecture with distributed authorization and accounting within the GAIA-X federated framework is the primary advantage of adopting the XADATU proposal. In practical terms, XADATU will offer fine-grained access control to improve security by limiting authorised user's access to dispersed resources, reducing risks, and guaranteeing adherence to laws like the General Data Protection Regulation (GDPR). Additionally, ABAC optimises resource use by enabling the accurate assignment of permissions according to roles and data sensitivity. Also, in accordance with the FLUIDOS project's present implementation of the zero-trust paradigm, domain-specific access control rules will be made possible using XACML-based policy access management. Furthermore, the XADATU proposal allows seamless interoperability with the current identity management mechanism, such as Decentralized Identifiers [3], Verifiable Credential (VC) [4], and OpenID Connect [5] implemented in the FLUIDOS ecosystem to extend the zero-trust paradigm to control efficiently and securely the authorization and accounting of highly distributed resources of dynamic Edge/Cloud deployments in Distributed Ledger Technologies (DLT) [6] systems, such as Hyperledger Fabric, which enables forensic analysis of resource utilisation and tracking of access events.

Funded by Horizon Europe Framework Programme of the European Union

1.3 Proposed solution

This XADATU project will design, develop, and integrate the required components within the FLUIDOS architecture responsible for providing access control policies management, decentralized fine-grained security policy decisions, and distributed ledger accounting of highly distributed resources access.

The new features will be created and incorporated into the VDR, Resource Acquisition Manager (REAR), Privacy/Security Manager, and other FLUIDOS components. By extending the REAR, the FLUIDOS module in charge of the negotiation process, XADATU will provide access to resources and services from distant FLUIDOS nodes. Specifically, XADATU will feature a Distributed Authorization Engine that will act as a lightweight Policy Enforcement Point (PEP) to provide authorization in accordance with XACML and ABAC standards. Also, decisions made by the Security Manager will be translated into standards-based Extensible Markup Language (XML) [7] documents that can be consumed and transported across many different FLUIDOS architecture components, such as Edge nodes and Cloud supernodes. This enables a PEP to request a highly distributed authorization decision-making process from the Policy Decision Point (PDP). Furthermore, Policy Administration Points (PAP) for the management of security policies that can be generated independently in various FLUIDOS domains will be designed and integrated by XADATU. Moreover, the XADATU solution will make use of the decentralized storage of the distributed ledger, or Hyperledger Fabric, which is a component of the Governance plane of the FLUIDOS ecosystem, to store and account for the security policies and authorization decisions.

The XADATU solution allows extending the decentralized FLUIDOS architecture to ensure that distributed entities (i.e. nodes and supernodes) have access control over their own resources. The solution will enable evaluating access requests against predefined policies and generating any session-specific authorisation token used by a concrete client to access a specific resource. XADATU will use distributed ledgers as storage infrastructure to avoid using centralised systems in the cloud that would generate unwanted dependencies.
By leveraging the XADATU solution, the security user administrator will use the XADATU extension of VDR with the PAP to define the security authorisation policies linked with the distributed identifiers for the security policy management with DTL storage. XADATU will implement and integrate the PEP in the REAR. The PEP will be an access control system responsible for enforcing access decisions made by the PDP. Moreover, the PDP will be integrated into the Privacy/Security Manager to enable evaluating access requests against predefined policies and generating any session-specific authentication token or credential used by a client to access a resource.

Funded by Horizon Europe Framework Programme of the European Union

2 Technical Approach

This section depicts the technological framework and developments under the FLUIDOS Open Call project. It includes a detailed examination of the XACML for defining and managing access control policies and the DLT used for secured and distributed storage of policies and authorization requests. An overview of the developments, technologies in use, and their implementations will be provided, illustrating how these components are integrated and contribute to the advancement of the project's objectives.

2.1 Technology Selection

This subsection provides the State of the Art of the technologies used for the developments in the XADATU Project. Also, it outlines how these technologies are employed to address specific requirements of the project, highlighting their roles in enhancing access control and ensuring secure, transparent policy management.

2.1.1 Extensible Access Control Markup Language

XACML is a standard, declarative, and XML-based language to define access control policies, which allows specifying the set of subjects that can perform certain actions on a specific set of resources based on their attributes. As shown in *Figure 2*, the XACML architecture comprises:

- The PAP manages/configures access control policies and stores them, following the XACMLv2 specification, in the XACML Policies file. For this goal, this component offers a web frontend to the system administrator.
- The PDP will access the XACML Policies file when an authorization request is received and decide if it must respond with a positive or negative verdict.

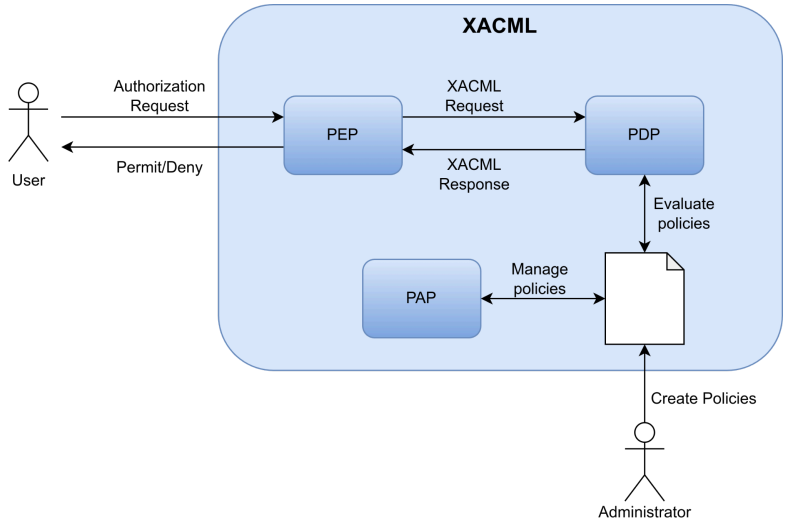

Figure 2 XACML architecture

Funded by Horizon Europe Framework Programme of the European Union

The definition of access control policies is mainly based on three elements in the XACML data model: PolicySet, Policy, and Rule (*Figure 3*). A PolicySet may contain other PolicySets and Policies, whereas a Policy includes a set of Rules, specifying an Effect (Permit, Deny, or Not Applicable), because of applying that Rule for a particular request.

- Rule: contains a Boolean expression that can be evaluated in isolation, but that is not intended to be accessed in isolation by a PDP. So, it is not intended to form the basis of an authorization decision by itself. It is intended to exist in isolation only within an XACML PAP, where it may form the basic unit of management.
- Policy: contains a set of *Rule* elements and a specified procedure for combining the results of their evaluation. It is the basic unit of the policy used by the PDP, and so it is intended to form the basis of an authorization decision.
- PolicySet: contains a set of *Policy* or other *PolicySet* elements and a specified procedure for combining the results of their evaluation.

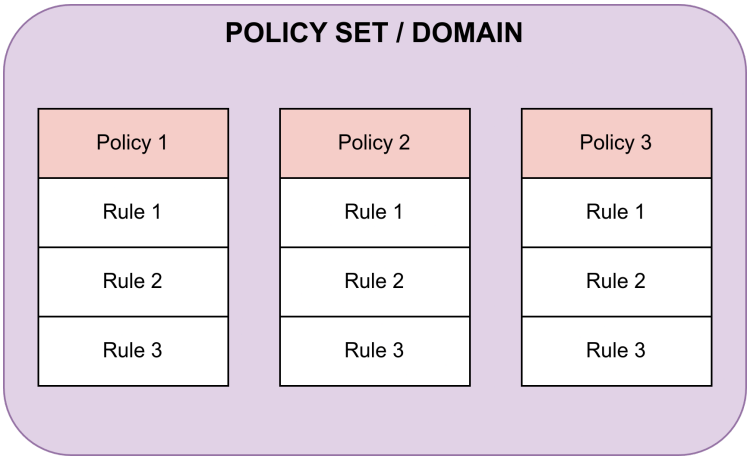

Figure 3 XACML Policy elements structure

The Target sections of these elements define the set of attributes from resources, subjects, actions, and environment to which the PolicySet, Policy, or Rule are applicable. Moreover, since different Rules might be applicable under a specific request, XACML defines Combining Algorithms to reconcile multiple decisions. In addition, a set of obligations (Obligations class) can be used to notify a set of actions to be performed related to an authorization decision.

In the XACML data model, rules are the fundamental building blocks used to determine whether a particular access request should be allowed or denied. Each rule has one of three possible effects: *Permit*, *Deny*, or *Not Applicable*.

The *Permit* effect is applied when an authorization request matches a rule that explicitly allows the action. This means that all the conditions specified within the rule are met, such as the attributes of the subject, the attributes of the resource being accessed, and the specific action being performed. When a rule's effect is *Permit*, the PDP concludes that the requested action is authorised and should be permitted. On the other hand, the Deny effect

Funded by Horizon Europe Framework Programme of the European Union

is triggered when an authorization request matches a rule that explicitly disallows or prohibits the action. This effect indicates that the conditions of the rule are met but the policy dictates that the action should not be allowed. A *Deny* decision prevents the requested action from taking place, ensuring that access is restricted according to the policy definitions.

The third possible effect, *Not Applicable*, occurs when an authorization request does not match any rule in the policy set. In this situation, the PDP cannot apply any of the rules because none of them meet the conditions specified in the request. Consequently, no explicit authorization decision is made, which can lead to a default behaviour being applied, such as denying access by default if no applicable rule is found.

In addition to these effects, XACML also supports combining algorithms, which are used to reconcile multiple rules' outcomes within a single policy. For instance, if multiple rules match a request with conflicting outcomes (some permitting and others denying), a combining algorithm will determine how these conflicts should be resolved — whether by giving precedence to *Permit* decisions, *Deny* decisions, or some other method (such as "deny-overrides" or "permit-overrides"). At this moment, the only algorithm that is implemented in the XACML component is the 'first-applicable' algorithm, which determines which rule to apply by selecting the first match that occurs while checking the rules. With this algorithm, the rules that you define first will have more priority than other rules defined later, so it is crucial to organise the order of the rules from more important to less important, so in the case there is a conflict between two or more rules, the rule that was defined first will be selected.

Together, these effects and combining algorithms make XACML a powerful and flexible framework for defining fine-grained access control policies, enabling organisations to precisely manage who can perform what actions on which resources under specific conditions.

2.1.2 Hyperledger Fabric

Hyperledger Fabric[1] is an open-source enterprise-grade permissioned DLT platform designed for use in enterprise contexts that delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms. One of the main features of Hyperledger Fabric is its modular and configurable architecture, enabling innovation, versatility, and optimization for a broad range of industry use cases. Another key feature is that Fabric is the first distributed ledger platform to support smart contracts authored in general-purpose programming languages such as Java, Go, and Node.js. The Fabric platform is permissioned, meaning that the participants are known to each other, rather than anonymous, and therefore fully untrusted as with Bitcoin or Ethereum. Another platform differentiator is its support for pluggable consensus protocols, such as crash fault-tolerant (CFT) when Fabric is deployed within a single enterprise or byzantine fault tolerant (BFT) when Fabric is deployed in a multi-party, decentralized use case. Also, Fabric can leverage consensus protocols that do not require a native cryptocurrency, thus reducing significant processing and transaction confirmation latency and the absence of cryptographic mining operations. In *Figure 4* the following elements can be observed:

[1] Hyperledger Fabric documentation | https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatis.html

Funded by Horizon Europe Framework Programme of the European Union

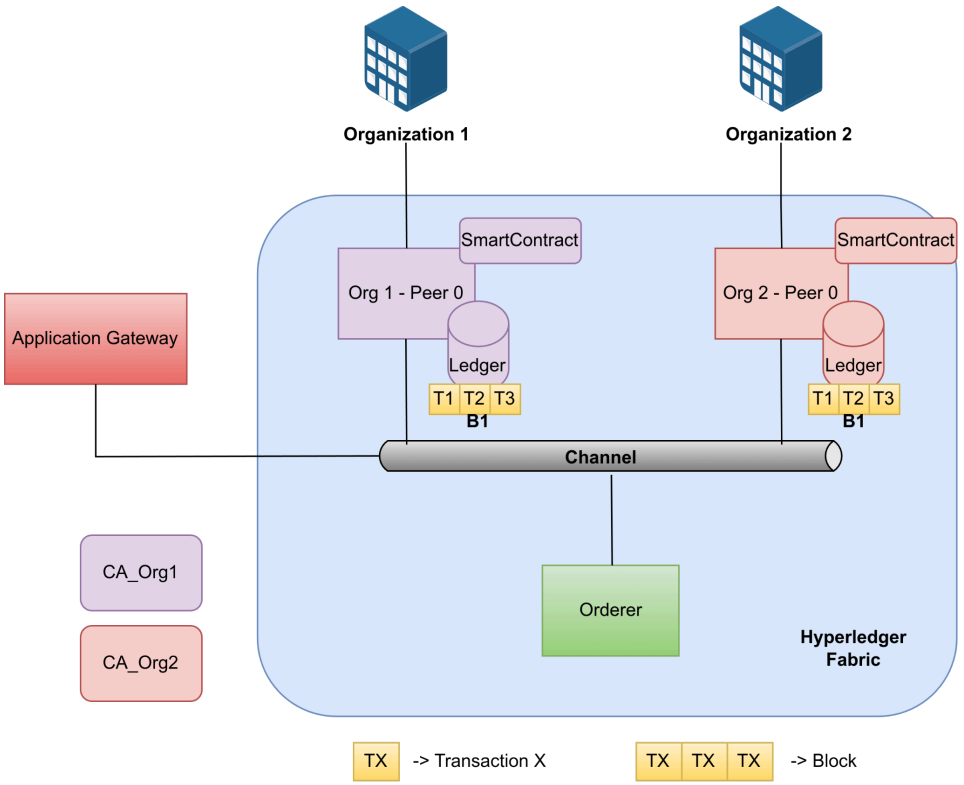

Figure 4 Hyperledger Fabric

- Organisation: An organisation is a member that has been added to the blockchain network invited by a blockchain network provider. An organisation is joined to the network by adding its Membership Service Provider (MSP) to the network. The MSP defines how other members of the network may verify that signatures were generated by a valid identity. The CAs are the entities responsible for issuing and managing cryptographic certificates for an organisation.
- SmartContract/Chaincode: A smart contract is code — invoked by a client application external to the blockchain network — that manages access and modifications to a set of key-value pairs in the World State via a Transaction. A smart contract defines the transaction logic that controls the lifecycle of a business object contained in the world state. In Hyperledger Fabric, smart contracts are packaged as chaincode. Then, the chaincode is installed on peers and then defined and used on one or more channels. Also, note that the smart contract can be defined within the same chaincode. When a chaincode is deployed, all smart contracts within it are made available for applications.
- Channel: A channel is a private blockchain overlay that allows for data isolation and confidentiality. Each channel has a completely separate world state, but applications and smart contracts can communicate between channels so that ledger information can be accessed between them. A channel-specific ledger is shared across the peers in the channel, and transacting parties must be authenticated to interact with it.

Funded by Horizon Europe Framework Programme of the European Union

- Ledger: A ledger is formed by two different but related parts - a blockchain and the "state database", also known as "world state". Blockchains are immutable, e.g., once a block is added to the chain, it cannot be changed. In contrast, the "world state" is a database containing the current value of the set key-value pairs that have been added, modified, or deleted by the set of validated and committed transactions in the blockchain. Smart contracts primarily put, get, and delete states in the world state, and can also query the immutable blockchain record of transactions.
- Peer: Peers are one of the main elements of Hyperledger Fabric networks. This is because peers are network entities that maintain a ledger and run chaincode containers in order to perform read/write operations to the ledger. Also, peers are flexible and redundant elements that can be created, started, stopped, reconfigured, and deleted. Another key characteristic is the Fabric Gateway service, which allows peers to expose a set of APIs that enable client applications to interact with the services that peers provide.
- Orderer: The orderer is a node that receives and processes transactions, creating blocks, and broadcasting them to the network. Also, these nodes enforce basic access control for channels, restricting who can read and write data to them, and who can reconfigure them. Along with other orderer nodes, they form an ordering service.
- Application: An application is an external element of the Hyperledger Fabric network that can interact with the network by submitting transactions to a ledger or querying ledger content using the Fabric Gateway. The process that involves the interaction between a client application, the gateway service running on a peer, orderer nodes, and additional peers is divided into 3 phases: Phase 1 - Transaction Proposal and Endorsement, Phase 2 - Transaction Submission and Ordering, and Phase 3 - Transaction Validation and Commitment.
- Transaction: A transaction is created when a chaincode is invoked from a client application to read or write data from the ledger. The transaction represents an operation or a set of operations that are executed on the blockchain network. Transactions are very important to understanding how clients' applications interact with the blockchain ledger and how they can update the state of the ledger. Hyperledger Fabric uses the transaction model, which ensures that only valid transactions endorsed by the required parties are added to the blockchain. This allows the integrity and consistency of the ledger across the network while also providing privacy.

Funded by Horizon Europe Framework Programme of the European Union

2.2 Developments and integrations

This subsection provides an overview of the developments and integrations within the FLUIDOS Open Call project. It highlights the implementation and status of critical components such as the XACML and its interaction with the DLT. The integration of XACML authorization processes and blockchain storage not only enhances security and transparency but also supports efficient management and querying of authorization requests. This comprehensive update outlines how these elements contribute to the project's overall functionality and effectiveness, reflecting the advancements made and their impact on system performance and compliance. All developments have been made in the FLUIDOS WP5 GitHub repository[2]. The green box of *Figure 5* shows the new integrations added by ODINS to the credential verification flow.

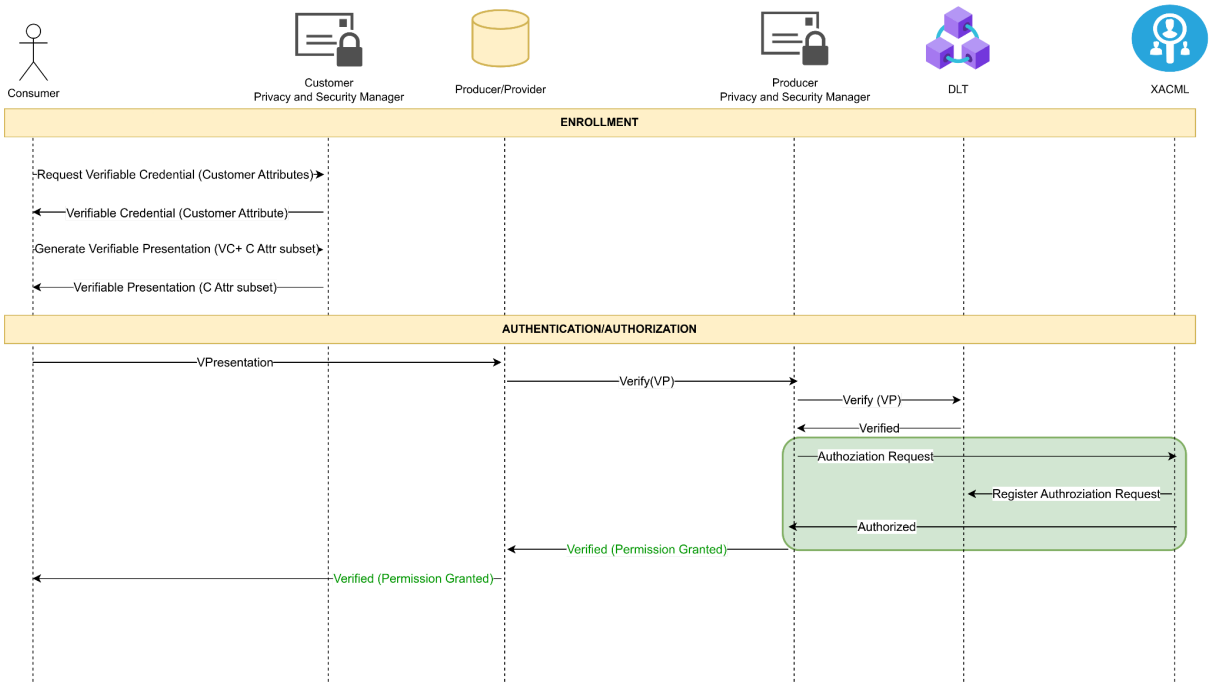

Figure 5 Verification and Authorization flow with ODINS updates

2.2.1 XACML Policies and Attributes

The XACML PAP provides a very simple web application from which it is possible to create, modify, and delete policies and attributes. The person responsible for interacting with the PAP should be the system administrator, and no one else should have access to manage the XACML policies and attributes. The frontend of the PAP can be divided into three parts or views.

[2] https://github.com/fluidos-project/idm-fluidos-aries-framework-go

Funded by Horizon Europe Framework Programme of the European Union

The first part is the main page, as shown in *Figure 6*. In this view, domains can be created to separate groups of policies and attributes for different contexts and use cases, allowing for independent policies for each scenario. Additionally, creating a domain is the initial step required before attributes and policies can be defined. From this page, it is also possible to access the other two sections of the web application: attribute management and policy management.

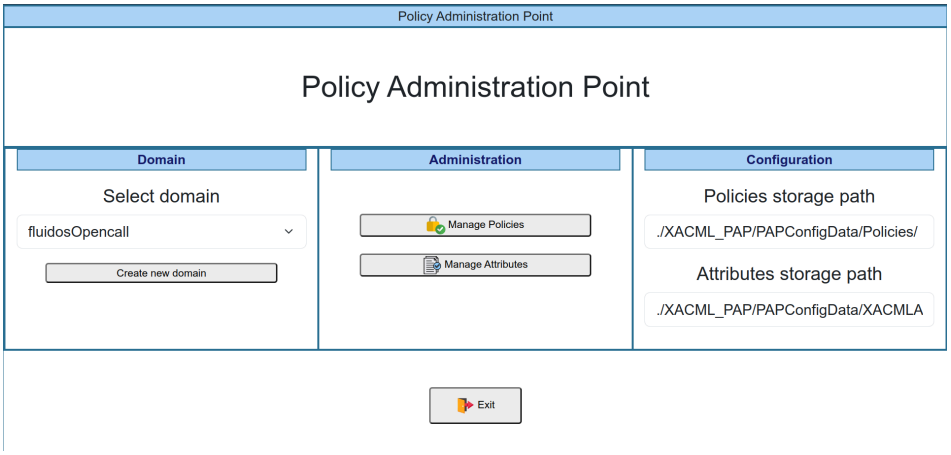

Figure 6 PAP main web page view

The second part of the web application is the attribute creation view. As shown in *Figure 7*, this section is divided into three parts, corresponding to each type of attribute supported by XACML. In this view, attributes to be used by the policies and rules can be created, modified, and deleted. All attributes are defined within a specific domain, as previously mentioned. Attribute creation is the second step in the process of defining policies.

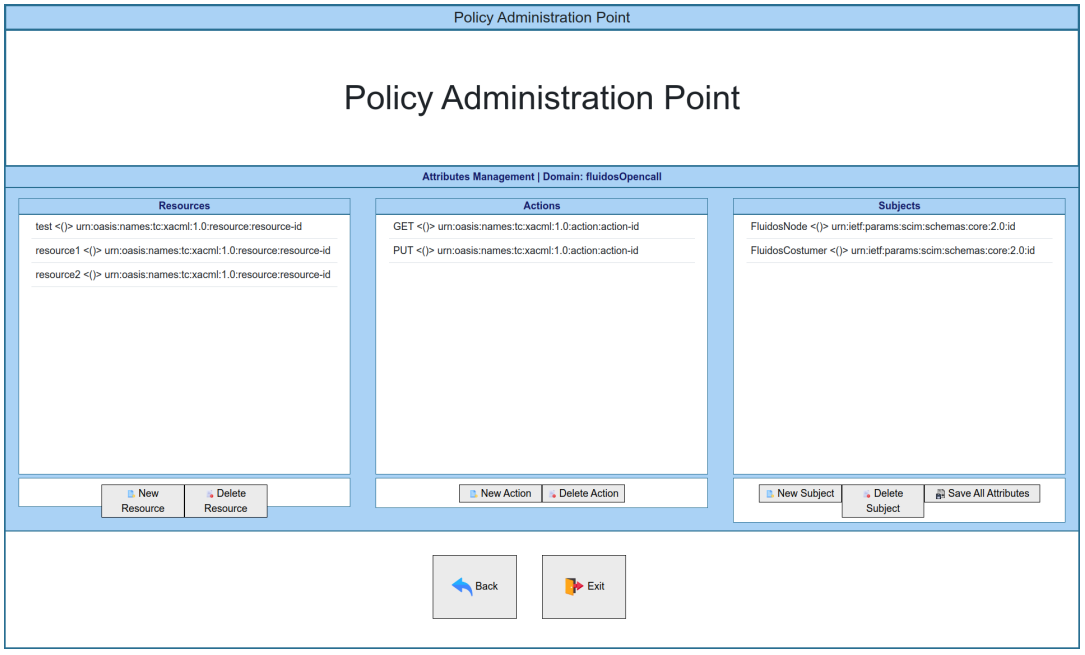

Figure 7 PAP attributes management web page view

Funded by Horizon Europe Framework Programme of the European Union

Finally, the third part of the PAP web application is the view for managing policies and rules. As previously mentioned, a policy consists of a group of rules, and multiple policies can be defined under a specific domain. The creation of policies is the final step in the process of defining them. *Figure 8* illustrates this part of the web application. To define rules, a policy must first be established, after which rules can be created by selecting one attribute for each group of attributes and specifying the policy decision, which indicates the verdict in the case there is a match with that rule, i.e., whether it is 'Permit' or 'Deny'. If there is no match with any of the defined rules, XACML will respond with 'Not Applicable,' which is neither permitted nor denied but is generally treated as denied.

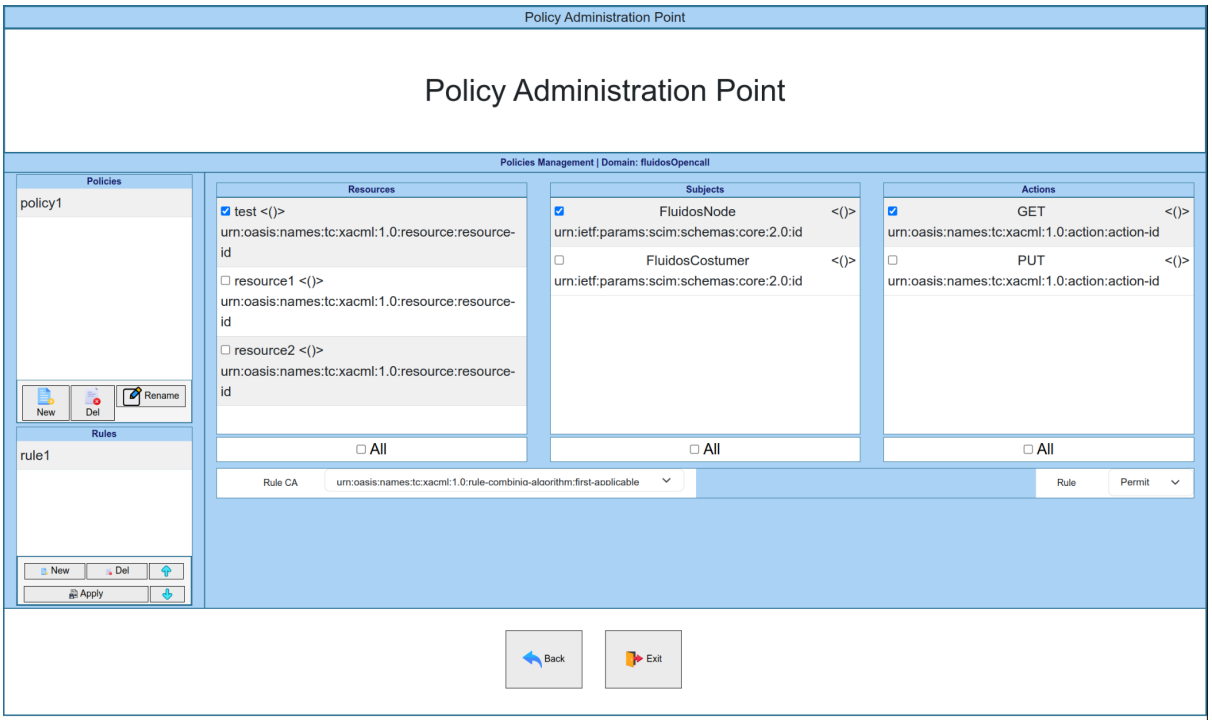

Figure 8 PAP policies management web page view

The XACML leverages a DLT, Hyperledger Fabric, for the storage of policies and attributes, which ensures enhanced security and transparency, as Hyperledger Fabric's decentralized and immutable nature prevents unauthorised modifications and provides a verifiable, tamper-proof record of all policy changes. To be able to store policies and attributes in the Hyperledger Fabric network, a Smart Contract with the functionality to store these kinds of values has been developed and deployed in Hyperledger Fabric. *Figure 9* shows an example of the JavaScript Object Notation (JSON) structure that contains all the information about the policies defined for one domain, which is the JSON that is stored in the Blockchain. Also, the XACML checks every two minutes if the policies have been updated by checking the 'timestamp' value, and if this value does not match with the value stored in the blockchain, the XACML will update the policies by uploading the new JSON to the blockchain. It is important to note that after updating the policies in the PAP web

Funded by Horizon Europe Framework Programme of the European Union

application, the old policies may continue to be effective for up to two minutes, so this is something to be taken into account.


```
"id": "urn:ngsi-ld:xacml:fluidosOpencall",
"type": "xacml",
"version": {
    "type": "Property",
    "value": "2"
},
"xacml": {
    "type": "Property",
    "value": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><PolicySet xmlns=\"urn:oasis:names:tc:xacml:2.
        0:policy:schema:os\" PolicyCombiningAlgId=\"urn:oasis:names:tc:xacml:1.
        0:policy-combining-algorithm:first-applicable\" PolicySetId=\"POLICY_SET\">  <Target/>  <Policy
        PolicyId=\"policy1\" RuleCombiningAlgId=\"urn:oasis:names:tc:xacml:1.
        0:rule-combining-algorithm:first-applicable\">    <Target/>    <Rule Effect=\"Permit\" RuleId=\"rule1\">
        <Target>        <Subjects>          <Subject>            <SubjectMatch MatchId=\"urn:oasis:names:tc:xacml:1.
        0:function:string-equal\">              <AttributeValue DataType=\"http://www.w3.org/2001/
        XMLSchema#string\">FluidosNode</AttributeValue>              <SubjectAttributeDesignator
        AttributeId=\"urn:ietf:params:scim:schemas:core:2.0:id\" DataType=\"http://www.w3.org/2001/XMLSchema#string\"/
        >            </SubjectMatch>          </Subject>        </Subjects>        <Resources>
        <Resource>            <ResourceMatch MatchId=\"urn:oasis:names:tc:xacml:1.
        0:function:string-equal\">              <AttributeValue DataType=\"http://www.w3.org/2001/
        XMLSchema#string\">test</AttributeValue>              <ResourceAttributeDesignator
        AttributeId=\"urn:oasis:names:tc:xacml:1.0:resource:resource-id\" DataType=\"http://www.w3.org/2001/
        XMLSchema#string\"/>            </ResourceMatch>          </Resource>        </Resources>
        <Actions>          <Action>            <ActionMatch MatchId=\"urn:oasis:names:tc:xacml:1.
        0:function:string-equal\">              <AttributeValue DataType=\"http://www.w3.org/2001/
        XMLSchema#string\">GET</AttributeValue>              <ActionAttributeDesignator
        AttributeId=\"urn:oasis:names:tc:xacml:1.0:action:action-id\" DataType=\"http://www.w3.org/2001/
        XMLSchema#string\"/>            </ActionMatch>          </Action>        </Actions>      </Target>    </Rule>
        </Policy></PolicySet>"
},
"timestamp": {
    "type": "Property",
    "value": "2024-09-11 08:17:17"
}
```


Figure 9 JSON structure of policies stored in Blockchain

2.2.2 XACML Authorization Process

The XACML authorization process is a process that consists of obtaining a verdict when trying to access a specific resource with a specific action. To obtain the verdict from the XACML, it is needed to send a specific request to the XACML component, which is responsible for making verdicts, the PDP. This functionality has been developed and integrated into the PSM. It has been integrated right after the verification of a Verifiable Presentation (VP), as shown in *Figure 5*. This introduces an additional layer of security. This step ensures that not only the VP is authentic and valid, but also that access permissions are strictly evaluated against defined policies before granting authorization. By incorporating XACML-based authorization, the system benefits from fine-grained access control, allowing for dynamic decision-making based on context and user attributes. This added measure significantly reduces the risk of unauthorised access, providing a more robust and flexible security framework. The next box shows an example of the Hypertext Transfer Protocol (HTTP) request that is sent to the PDP to obtain a verdict:

Funded by Horizon Europe Framework Programme of the European Union

Method: POST
URL: http://xacml:8883/pdp/veredict

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
        <Subject
SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
                <Attribute AttributeId="urn:ietf:params:scim:schemas:core:2.0:id"
                DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>FluidosNode</AttributeValue>
                </Attribute>
        </Subject>

        <Resource>
                <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>test</AttributeValue>
                </Attribute>
        </Resource>

        <Action>
                <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>GET</AttributeValue>
                </Attribute>
        </Action>

        <Environment/>
</Request>
```

Table 1 Example of a request to the XACML to obtain a Verdict

After receiving the request, the PDP will check if there is a match with any of the rules defined under the domain specified in the 'domain' header of the request. As the only algorithm defined is the 'first-applicable', the PDP will iterate through every policy and rule defined for the specified domain until there is a match. If there is no match, it will reply with 'Not Applicable', but if there is a match, the PDP will reply with the decision included in the rule, which can be 'Permit' or 'Deny'. Following the example above from *Figure 9*, this request will match with 'rule1' and the PDP will reply this:

RESPONSE CODE: 200 OK

```
<Response>
  <Result ResourceID="test">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Table 2 Example of a successful Verdict response from XACML

2.2.3 XACML Accounting

Funded by Horizon Europe Framework Programme of the European Union

Another functionality that has been developed and integrated into the FLUIDOS architecture is the registration of the XACML authorization request into the Blockchain. This process has been developed and integrated as a new XACML functionality, as the XACML has all the necessary information needed to register all the authorization requests it receives. Furthermore, a first version of the Smart Contract that allows storing authorization requests has been developed and deployed in the Hyperledger Fabric network of FLUIDOS. This first version of the Smart Contract offers the possibility to register, query-by-ID, and query-by-date authorization requests. The query-by-date functionality returns all the authorization requests between two dates. *Figure 10* shows the structure of the JSON that is stored in the Blockchain, which is formed by the following field:

- Timestamp: indicates the date in ISO 8601 format when the XACML receives the authorization request.
- Action: indicates the HTTP Method that the 'Subject' is attempting to use to access the resource indicated in 'Resource'.
- Resource: indicates the resource that the 'Subject' is trying to access.
- ID: indicates the unique identifier for each authorization request. It is also used as the key to store the authorization request in the Blockchain.
- DID: indicates the DID of the consumer node that requested the Access Token.
- Subject: indicates the identity of the requester of the authorization process.
- Decision: indicates the verdict of the XACML for this authorization request.


```
{
  "Timestamp": "2024-12-23T12:04:02.861456",
  "Action": "GET",
  "Resource": "https://172.16.10.118:1027/producer/flavors",
  "ID": "2681363a943bf7c1bdacb2d232075391",
  "DID": "did:fabric:iecsNmDhJQV2HBFNqG0W6aAkLPENTvAL2S1cZzABFNY",
  "Subject": "Customer",
  "Decision": "Permit"
}
```


Figure 10 JSON structure of the authorization request stored in Blockchain for accounting

The storage of XACML authorization requests on a blockchain offers several benefits. As Hyperledger Fabric is decentralized and immutable by nature, it ensures that authorization requests are securely recorded and cannot be altered. This provides a transparent and verifiable register of access control decisions, enhancing accountability and traceability. Moreover, the structured and permanent record on the blockchain facilitates efficient querying and consultation of authorization requests for accounting and compliance

Funded by Horizon Europe Framework Programme of the European Union

purposes, enabling organisations to easily track and verify historical access decisions and ensure adherence to regulatory requirements.

2.2.4 Access Token

Once the verification process is completed, and provided that the authorization has been granted, the next step is the creation of an Access Token as shown in *Figure 11*, which is a compact digital entity that provides a user or application with the necessary permissions to access specific resources. These tokens function as a digital key, verifying that the user has the appropriate authorization to access the requested data. In the XADATU project, the Access Tokens implemented are in the form of JSON Web Tokens (JWT) [8], which is an open standard used to create compact, self-contained tokens used for securely transmitting information between different applications or services. These tokens are typically used for authentication and authorization, as they can contain information that verifies the identity of a user, and their permissions.

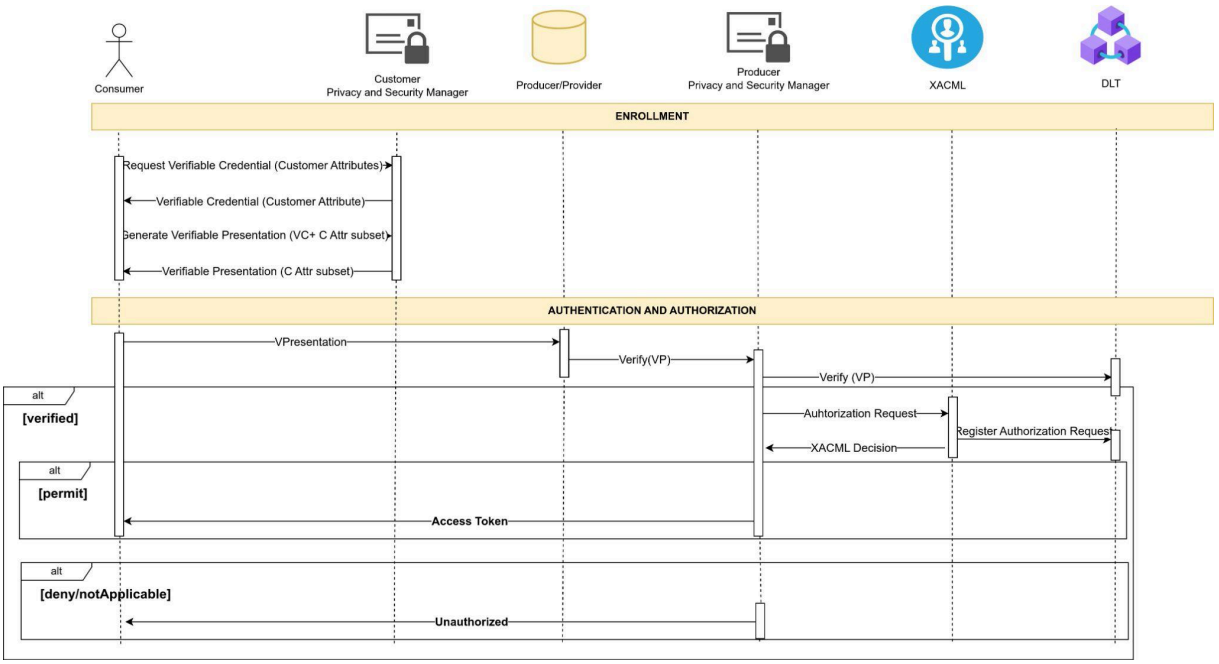

Figure 11 Sequence diagram of the process for obtaining an Access Token

JWTs are composed of three main components: the header, the payload, and the signature. The header is a JSON object that typically contains different properties, such as the token type and the encryption algorithm. Then, the payload is another JSON object that includes claims, which are pieces of information or data that represent information about a user or an entity, such as authorization permissions, session data, expiration time, etc. Finally, the signature is created by signing the Base64Url encoded header and payload with a secret key. In this case, the secret key used to sign the Access Token is the private key of the node acting as a 'Producer' and assigned to its DID. *Figure 12* shows an example of an Access Token obtained during the execution of the demo:

Funded by Horizon Europe Framework Programme of the European Union

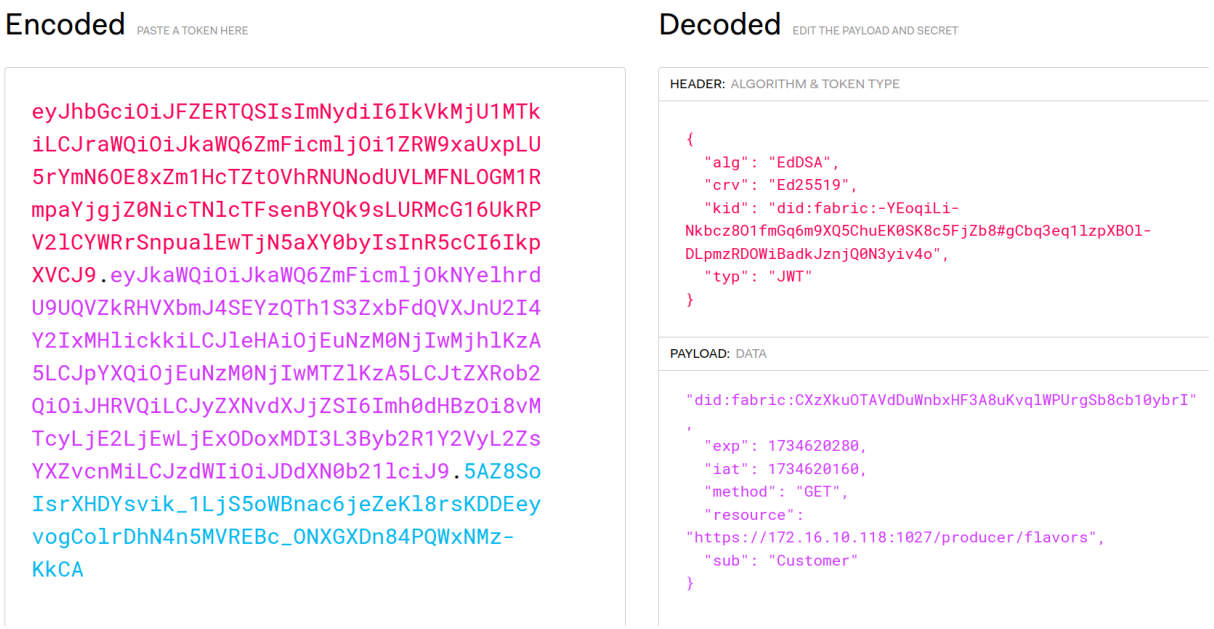

Figure 12 Access token obtained during the execution of the demo

The decoded header of the JWT in the image above has the following fields:

- alg: specifies the algorithm used to sign the JWT. In this case, it is the Edwards-Curve Digital Signature Algorithm (EdDSA) [9].
- crv: indicates the elliptic curve used when signing the algorithm requires one. In this case, it is Ed25519, a popular elliptic curve associated with EdDSA, known for its high speed and strong security properties.
- kid: provides a unique identifier for the cryptographic key used to sign the token, which helps the recipient of the JWT to quickly look up and verify the corresponding public key. In this case, the value is the DID of the producer node that signs the Access Token.
- type: declares the type of the token. In this case, the type is JWT.

Then, the next part of the JWT Access Token is the payload, which has the following fields:

- did: the unique identifier of the node that requests the Access Token.
- exp: specifies the time in Unix timestamp format when the token will expire and no longer be valid.
- iat: indicates the time in Unix timestamp format when the token was issued.
- method: indicates the HTTP method that can be used to access the resource indicated in 'resource'. In this case, the value is 'GET', which means that the owner of the token is authorized to access the resource exclusively using this HTTP method.
- resource: indicates the target resource URL that the token grants access to. In this case, the resource is the '/producers/flavors' endpoint, which indicates that the owner of the token is authorized to list the flavors of the producer node.
- sub: indicates the role or identity of the user/entity interacting with the system.

These fields in the header and the payload are not static or fixed, meaning you can add as many fields as you want. Furthermore, JWT has standardized fields for the header and the

Funded by Horizon Europe Framework Programme of the European Union

payload as defined in the RFC specification. Moreover, there is also the possibility of including custom fields in both the header and the payload to meet specific application requirements. Therefore, using standardized fields ensures interoperability and consistency across systems, while the use of custom fields allows flexibility in the definition of claims.

The use of Access Tokens provides several benefits. For instance, JWT Access Tokens enhance scalability by reducing the load on systems, authentication servers, and databases, making them more efficient. Additionally, Access Tokens enable fine-grained access control, allowing precise regulation of what resources a user or application can access, ensuring greater flexibility and security in resource management. Another key feature of JWT Access Tokens is its expiration time, which is crucial for enhancing security and mitigating potential risks. By having a defined expiration time, tokens minimize the window of opportunity for attackers to take advantage of them if stolen. The recommended expiration time for JWT Access Tokens is typically short, ranging from 5 to 15 minutes, depending on the sensitivity of the application. This short lifespan ensures that even if a token is compromised, the potential damage is limited. However, shorter expiration times may require implementing token refresh mechanisms to maintain a seamless user experience. The Access Token was implemented during the period between the midterm report and the final report.

2.2.5 PEP-Proxy

The PEP-Proxy is a component within the XACML framework that operates independently of the other XACML components, as it is distributed and deployed on each node in the system. Its primary role is to act as a gatekeeper between clients (requesters) and protected resources, i.e., to secure requests directed to the REAR by verifying whether requests include a valid Access Token in the 'x-auth-token' header. To validate the Access Token, the PEP-Proxy performs several checks (shown in *Figure 13*):

1. Token expiration: PEP-Proxy decodes the Access Token and checks if it has expired by obtaining the date in Unix timestamp format from the 'exp' field of the Access Token payload.

2. Token signature: PEP-Proxy validates the signature of the Access Token using the information of the Access Token header. As the token is signed with the private key associated with the DID of the producer node, the PEP-Proxy verifies the token using the public key associated with the DID of the node that signed the token.

3. Request matching: the PEP-Proxy verifies that the HTTP method and the endpoint being accessed correspond to those specified in the token's fields. For example, if the Access Token allows the user to access the endpoint '/producer/flavors' with the 'GET' HTTP method, but the request is a POST to that endpoint, the PEP-Proxy will reject the request.

Funded by Horizon Europe Framework Programme of the European Union

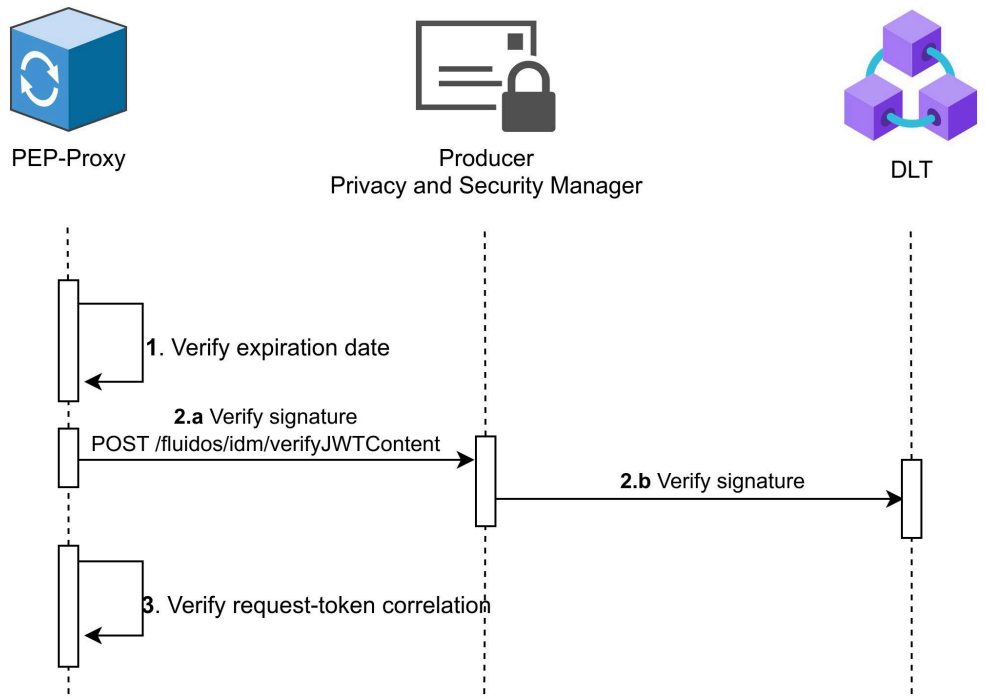

Figure 13 Sequence diagram of how the PEP-Proxy verifies an Access Token

Figure 14 depicts how the PEP-Proxy operates, granting or denying access to the REAR based on the validity of the Access Token it receives. If the token is valid — meaning it has not expired, the signature is valid, and its claims match the request being made — the PEP-Proxy forwards the request to the REAR. On the other hand, if the token is invalid, the PEP-Proxy denies the requests and responds to the consumer with a '401: Unauthorized' error to the consumer. A request can be rejected for various reasons:

- Token is missing: the consumer did not include an Access Token in the 'x-auth-header'.
- Token has expired: the Access token included in the request has expired, which means that the token is no longer valid, even if the signature is valid.
- Invalid the signature: the PEP-Proxy cannot verify the signature of the token, which means that it may have been signed by an untrusted party. Therefore, the request must be rejected.
- Request-Token mismatch: the request's characteristics do not match the claims of the Access Token. For example, when you have an Access Token to access the '/listFlavor' endpoint but instead of accessing that endpoint, you use that Access Token to access the '/purchaseFlavor' endpoint. In this case, the token is completely valid, but it is not being used as it should and the request must be rejected.

Funded by Horizon Europe Framework Programme of the European Union

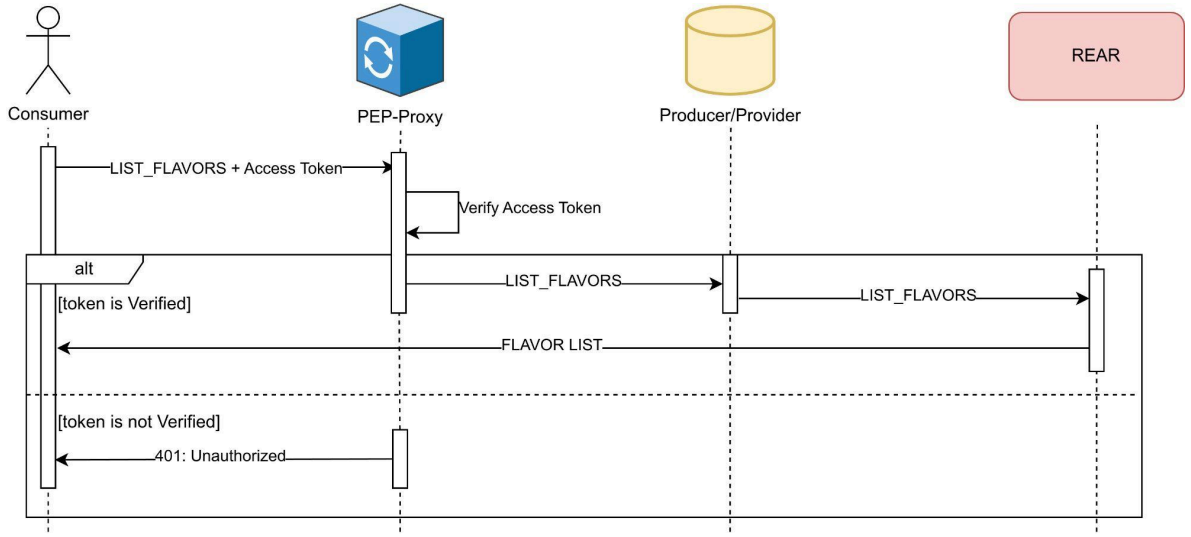

Figure 14 Sequence diagram of the PEP-Proxy functionality

The Access Token is stored in the node that requests it. Following the example above, the consumer is responsible for storing the Access Tokens it requested to the Producer PSM. Finally, *Table 3* shows the response sent by the PEP-Proxy when the Access Token is missing or invalid.

Method: GET URL: https://<<producerIP>>/producer/flavors
RESPONSE CODE: 401 Unauthorized

{
"code": 401,
"error": "Unauthorized",
"details": "The token is missing or invalid."
}

Table 3 Example of invalid Access Token response

Implementing a PEP-Proxy offers several key characteristics that enhance security and access control in distributed systems. It acts as a mediator between clients and resources, ensuring requests are authenticated and authorized before granting access. Additionally, validating access tokens — including their signatures, expiration, and scope — prevents unauthorized or tampered requests. Moreover, a PEP-Proxy enforces fine-grained policies by matching token claims with the requested HTTP methods and endpoints. Furthermore, its distributed deployment capability allows it to be close to the protected resources, reducing latency and enabling scalability while maintaining robust access control mechanisms. The PEP-Proxy was implemented during the period between the midterm report and the final report.

Funded by Horizon Europe Framework Programme of the European Union

3 Results, Key Performance Indicators, and Milestones

3.1 Results - Demo integration

During the final stage of the opencall, and after all the XADATU security components have been developed, collaboration has taken place with the colleagues from the University of Murcia (UMU) to create a demo that integrates all the security components developed in the XADATU project with the FLUIDOS architecture. This demo focuses on validating the security mechanisms and components developed by UMU, which were already part of the FLUIDOS project, alongside the security mechanisms and components from XADATU. This integration adds an additional layer of security, including distributed authorization through XACML components (PEP-Proxy, PDP, PAP) and the DLT. Moreover, XADATU has also integrated the accounting of authorization requests made to the XACML, enabling transparency, auditing, and compliance. This feature tracks who made each request, when, and with what outcome, allowing the detection of unauthorized access attempts, ensuring correct policy enforcement, and meeting regulatory requirements.

Figure 15 depicts the FLUIDOS architecture deployed for the demo. As mentioned before, the demo is focused on demonstrating the functionality of the security components. The red numbers indicate that those steps were already implemented by the colleagues of UMU. On the other hand, the green numbers indicate the additional steps OdinS has developed and implemented to include the XADATU security components and functionalities. Moreover, ODINS has added the XACML and PEP-Proxy to FLUIDOS's initial architecture and deployed Smart Contracts for storing XACML policies and accounting for authorization requests in the DLT.

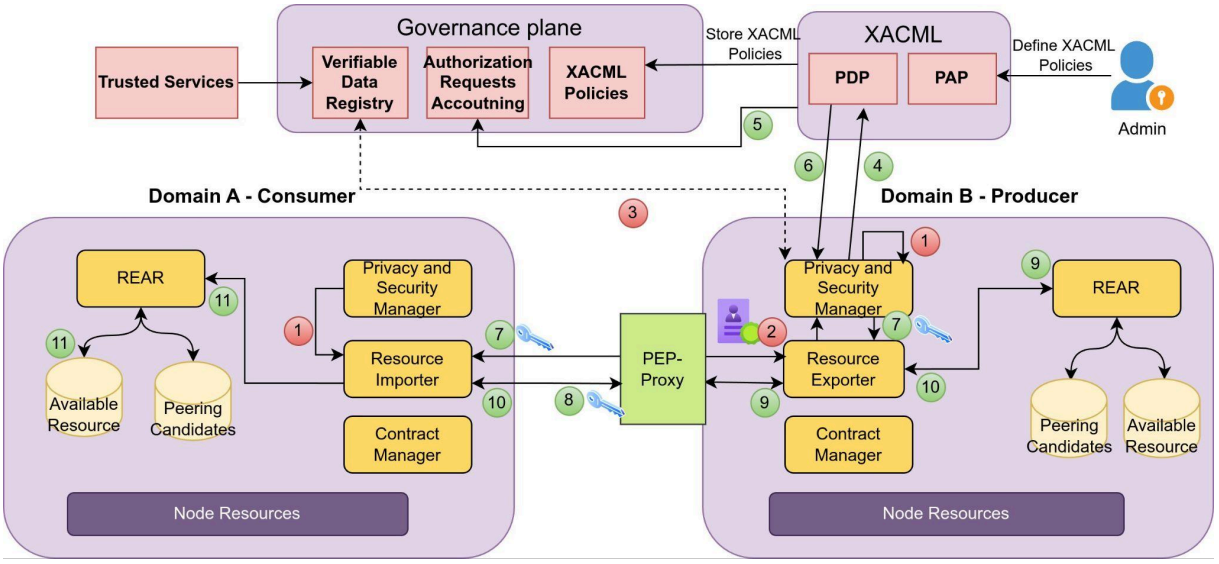

Figure 15 XADATU security components integration in demo

The following are the steps illustrated in the figure above, explained in detail:

Funded by Horizon Europe Framework Programme of the European Union

1. The PSM of each domain creates its DIDs, and the PSM from Domain A generates its VC. *Figure 16* shows how the Consumer obtains its DID in the demo, while *Figure 17* shows how the Producer obtains its DID. The generated DID document is different for each party because the Producer has two verification methods: one with the key pair generated using the Ed25519 algorithm and the other one with the key pair generated using the Bls12381G1 algorithm. After this, the Consumer requests a Verifiable Credential for its PSM, as seen in *Figure 18*. This VC includes key information about the Consumer, such as claims with information about the Consumer, the expiration date of the VC, or how to verify the credential.

Figure 16 DID generation for Consumer

Figure 17 DID generation for Producer

Funded by Horizon Europe Framework Programme of the European Union

Figure 18 Verifiable Credential for Consumer

2. The Resource Importer of the Consumer creates a VP and sends it to the Resource Exporter of the Producer to obtain an Access Token. In this step, after the creation of the VP, the Consumer checks if there is any Access Token stored to access the endpoint it wants to consume. If there is no Access Token stored for that endpoint, the Consumer then starts the process of obtaining the Access Token by sending the VP to the Producer. As it is the first time accessing any endpoints, there are no Access Tokens stored, so the Consumer will have to obtain the Access Tokens needed to consume all of the endpoints from the Producer. *Figure 19* shows how the VP is created in the demo.

Figure 19 Verifiable Presentation creation for Consumer

Funded by Horizon Europe Framework Programme of the European Union

3. The PSM of the Producer verifies the VP by querying the VDR for the DID public key associated to that Verifiable Presentation, and the defined credential schemes.
4. If the VP is valid, the PSM of the Producer makes a request to the PDP component of the XACML to check if the consumer has authorisation.
5. The PDP stores the resource access event in the Blockchain for accounting of the XACML requests.
6. If the PDP validates and grants approval to the request according to the security policies stored in the Blockchain, the PDP sends a confirmation to the PSM of the Producer.
7. If the PSM of the Producer receives that the VP has been verified, and the Consumer is authorised, it will create the Access Token and send it to the Consumer. On the other hand, the Consumer will receive and store the Access Token to use it to make requests to the REAR of the producer.
8. The consumer sends a request to the REAR of the Producer, including the Access Token in the request. *Figure 20* shows the process of accessing one endpoint. As mentioned before, the first step when trying to access an endpoint is to check if there is already an Access Token for that endpoint. If there is no Access Token, or the Access Token has expired, the Consumer will have to do steps 2-7. Once the Access Token is obtained, for future requests, the Consumer can skip those steps and send the request directly to the endpoint, so the process of obtaining the Access Token is executed once in a while, it is not executed every time the Consumer wants to access an endpoint.

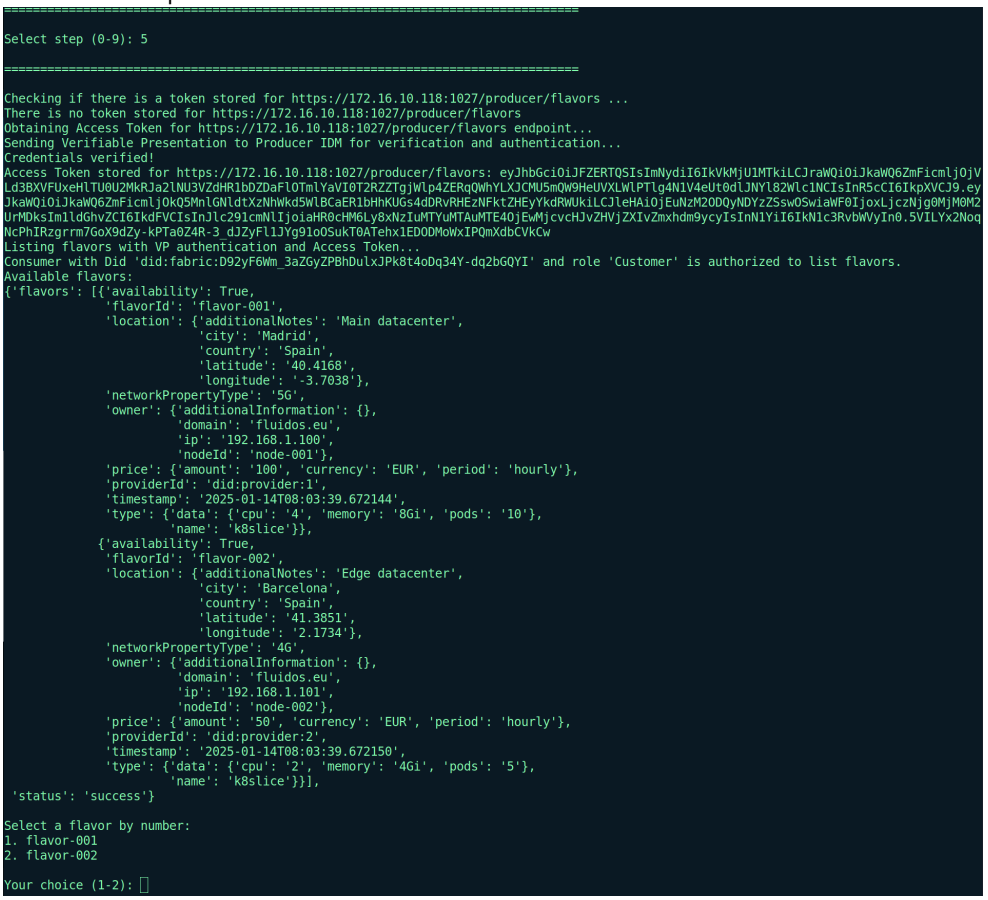

Figure 20 Consumer consuming the *LIST_FLAVORS* endpoint of Producer

Funded by Horizon Europe Framework Programme of the European Union

9. The PEP-Proxy receives the request and verifies that the Access Token is valid and has not expired. If this is the case, the PEP-Proxy forwards the request to the REAR.
10. The REAR processes the request and sends a response to the Consumer, as can be seen in *Figure 20*.
11. The Consumer receives and processes the response. In this example, the response is the list of flavors available at the Producer, which is stored in the consumer for future use.

The integration of the security components and functionalities introduced by XADATU (XACML, accounting of authorization requests, Access Tokens…) with the security components and functionalities already developed and implemented in FLUIDOS (DIDs, VCs, and VPs) provides a series of benefits to the architecture and security of the project. Integrating the XACML enforces fine-grained access control policies, ensuring that authorisation decisions are more accurate. Then, the blockchain recording of authorisation requests in XACML. In addition, Access Tokens facilitate secure and efficient communication, while the PEP-Proxy is in charge of validating them and redirecting valid requests to the necessary backend services. These enhancements not only improve the system's security and accountability but also ensure scalability and interoperability across distributed environments.

3.2 Key Performance Indicators

The XADATU project established a series of Key Performance Indicators (KPIs) and milestones to measure its success and progress. The KPIs were carefully selected and defined to align closely with the project's objectives. They are focused on evaluating compatibility, system availability, performance, and the impact of the developed solution. The monitoring of KPIs provides actionable insights into system compatibility, availability, performance, and community engagement. By tracking these indicators, the XADATU project evaluates its achievements and ensures the delivery of a robust, scalable, and secure solution that integrates seamlessly with the FLUIDOS architecture. *Table 4* shows the results obtained based on the defined KPIs.

KPI	Description	Target Value	Achieved Value	Comments
XADATU Integration	Assess the system's compatibility and integration of the XADATU tools	100%	100%	The integration of XADATU in the FLUIDOS architecture was fully integrated.
Distributed authorization and blockchain account systems availability	Monitor the uptime and accessibility of distributed authorization and blockchain accounting systems	99.9%	100%	During the tests of the solution, the distributed authorization and blockchain account systems were available the whole time.
Transaction Throughput	Evaluate the number of transactions processed per unit of time.	-	5.7 Transactions per Minute	Number of transactions per minute achieved when executing the demo.

Funded by Horizon Europe Framework Programme of the European Union

Open Source	Publish the open source libraries developed. At least 2 open source repositories published in Github with the XADATU development for distributed authorisation and accounting	2	1*	All open source libraries developed were uploaded to the same Github repository[3] to facilitate deployment for testing.

Table 4 Evaluation of KPIs and achieved results

Regarding the integration of XADATU KPI, as explained throughout this document, all components have been seamlessly integrated into the FLUIDOS architecture, fulfilling the KPI that estimated 100% of XADATU's integration with FLUIDOS. It is important to note that this KPI was measured using the FLUIDOS components available during the execution of the Open Call. In other words, not all system components of FLUIDOS were available or final, meaning that the integration and compatibility of XADATU were evaluated based on the FLUIDOS components available during the implementation of the Open Call components. On the other hand, the KPI for the availability of the distributed authorization and blockchain accounting systems has been measured based on the tests conducted. Since the FLUIDOS project is in an implementation and integration phase, certain components of the architecture are not yet fully deployed. As a result, the components were repeatedly launched, stopped, and restarted for various tests. During the periods when all components were deployed and tests were being performed, the distributed authorization and blockchain accounting systems remained operational 100% of the time. Regarding the KPI of evaluating the number of transactions per unit of time, during the demo execution, an average of 5.7 transactions per minute was achieved. It is worth noting that at the beginning of the execution, the number of transactions with the blockchain is higher since it involves interactions for the registration of DIDs and XACML policies/attributes of the XACML. However, once everything has stabilized, the only transactions executed are for verifying VCs and for storing the requests made to the XACML. Additionally, using the Access Token reduces the number of interactions with the blockchain, as it eliminates the need to verify the VCs every time a request is made to the Producer. Finally, regarding the KPI that measures the number of open-source repositories published, the target value has not been achieved because all open-source components have been published in a single repository to facilitate the deployment of the architecture for functional testing. It is worth noting that all components developed and integrated by XADATU are open source, which includes the XACML, the PEP-Proxy, the REST API server that acts as an intermediary between the Blockchain and the applications, and the Smart Contracts developed both for storing XACML policies and for accounting the requests made to the XACML. Therefore, in the future, all these components can be divided into individual repositories and published as open source.

3.3 Milestones

The milestones selected for the XADATU project establish key checkpoints reflecting progress and achievements throughout the project's lifecycle. These milestones are directly related to critical phases of planning, development, and evaluation, ensuring that the project stays on track and aligned with its objectives within the FLUIDOS framework.

[3] https://github.com/fluidos-project/idm-fluidos-aries-framework-go/tree/opencall-XADATU

Funded by Horizon Europe Framework Programme of the European Union

Furthermore, each milestone is associated with a set of reports, activities, software development and implementation, and demonstrations that showcase the project's progress. By achieving these milestones, the XADATU project demonstrates its commitment to delivering a well-integrated, validated solution while addressing the requirements of FLUIDOS. *Table 5* shows the milestones proposed for the XADATU project and their current status.

Milestone	Description	Date	Status
Initial Plan	Reported activities and outcomes of T1.1.	End of M1	Completed
Midterm Report	Reported the XADATU design and developments of T.2 until M4	End of M4	Completed
Final Report	Reported the final design, development and evaluation of T.2 and T.3	End of M8	Completed

Table 5 Evaluation of milestones and current status

Each milestone in the table above represents a significant point in the progress of the XADATU project. Furthermore, they are closely tied to the tasks proposed for the project's execution, as shown in *Table 6*. These milestones are divided into the Initial Plan, the Midterm Report, and the Final Report:

1. Initial Plan: This milestone indicates the initial planning of the XADATU project, focusing primarily on Task T.1, which involves the preparation and structuring of all activities related to the integration of the XADATU solution with the requirements of FLUIDOS. This milestone was completed at the end of the first month, establishing a solid foundation that enabled further progress in the project's development.
2. Midterm Report: This milestone corresponds to the intermediate report documenting the design and developments carried out during Task T.2 up to the fourth month of the project. It is a critical milestone as it showcases some of the components developed for the XADATU project and their functionality. Additionally, this milestone was completed at the end of the fourth month, ensuring that the project progresses as planned, evaluates technical advancements, and provides valuable feedback for the subsequent steps.
3. Final Report: This milestone represents the final stage of the project, where the designs, developments, and evaluations carried out during tasks T.2 and T.3 are documented. This report includes the final outcomes, such as the development of the MVPs and their validation in test environments.

Funded by Horizon Europe Framework Programme of the European Union

Task Name	Activity Description	Plan Duration	Expected Outcome
T.1 XADATU Planning	T.1 will plan the XADATU solution according to the requirements of FLUIDOS architecture and its key components (i.e., Security Manager, Resource Acquisition Manager, Verifiable Data Registry). To do that, ODINS will take most of the requirement analysis defined in FLUIDOS, as well as active cooperation with UMU and the consortium. Moreover, T.1 will consider aspects like the integration in FLUIDOS architecture and open-source sharing. This task will define SCRUM software backlog, APIs, and open-source aspects for T.2 as validation factors for T.3.	M1 - 1 month	D1 Initial report of all T1 activities
T.2 Design & Developing	T.2 will design and develop the distributed authorisation and accounting mechanisms (i.e., PAP, PEP, PDP) based on standardized XACML, ABAC, and DTL technologies for GAIA-X framework of the FLUIDOS architecture and open source stacks (e.g., Linux). This development will be performed in iterative development cycles considering the feedback from FLUIDOS partners and the validation performed in task T.3.	M2-M7 - 6 months	D2 Midterm report; all SW tools developed; open source libraries
T.3 Deploying & Evaluating	T.3 will deploy and evaluate the developed MVPs of T.2 according to the factors defined in T.1 for FLUIDOS needs, open-source aspects, authorisation/Accounting functionalities, and key performance indicators (KPIs). T.3 will validate the MVP development in testbed sites of OdinS and UMU (Spain). This task will provide the validation results and the opinions of the UMU experts to feed the task of T.2 design and development. The result of this task will be a final TRL5 demo integrated into FLUIDOS architecture and KPIs obtained.	M5-M8 - 4 months	D3 Final report; TRL5 Demo; KPIs monitored

Table 6 Proposed activities for the XADATU project

Funded by Horizon Europe Framework Programme of the European Union

4 Conclusions

The final report summarizes the progress of the XADATU Project, marking its completion as part of the FLUIDOS Open Call. The report highlights the project's contributions, focusing on the technological choices and recent developments. Additionally, decentralized authorization and accounting functionalities have been integrated into the current FLUIDOS architecture, which are crucial for access control in dynamic and geographically distributed Edge/Cloud environments. XADATU enhances the security and scalability of the FLUIDOS ecosystem through Distributed Ledger Technologies (DLT) and advanced access control models such as ABAC and XACML. The integration of these technologies addresses critical challenges within the FLUIDOS Open Call, supports interoperability with existing identity management mechanisms, and ensures alignment with the zero-trust paradigm. The methods presented in this report aim to optimize resource utilization, provide precise access control, and ensure open and verifiable accounting of authorization operations.

Working with the technology developed in the FLUIDOS project has brought both benefits and challenges. Regarding the benefits, the technology developed and implemented in XADATU aligns very well with the previous work carried out in the project's privacy and security task. This has allowed the FLUIDOS technology to maximize the performance of the components integrated by XADATU. On the other hand, no significant difficulties have been encountered beyond the challenge of integrating new security components into an ongoing project. Since the open call started midway through the FLUIDOS project, some components have evolved over time, which has led to XADATU components also evolving and adapting to the project's changing needs. As a result, the experience with FLUIDOS technology has been very valuable.

In conclusion, this final report showcases the achievements of the XADATU Project, which integrates decentralized authorization and accounting into the FLUIDOS architecture. This improves security, scalability, and interoperability, aligning with the zero-trust paradigm and ensuring access control and operational accounting.

Funded by Horizon Europe Framework Programme of the European Union

5 References

[1] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, 'Attribute-Based Access Control', Computer, vol. 48, no. 2, pp. 85–88, Feb. 2015, doi: 10.1109/MC.2015.33.

[2] eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[3] M. Sporny, D. Longley, S. Markus, D. Reed, O. Steele, and C. Allen, 'Decentralized Identifiers (DIDs) v1.0'. Accessed: Jan. 20, 2025. [Online]. Available: https://www.w3.org/TR/did-core/

[4] M. Sporny, D. Longley, and D. Chadwick, 'Verifiable Credentials Data Model v1.1'. Accessed: Jan. 20, 2025. [Online]. Available: https://www.w3.org/TR/vc-data-model/

[5] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <httpp://openid.net/specs/openid-connect-core-1_0.html>

[6] R. Soltani, M. Zaman, R. Joshi, and S. Sampalli, 'Distributed Ledger Technologies and Their Applications: A Review', Applied Sciences, vol. 12, no. 15, Art. no. 15, Jan. 2022, doi: 10.3390/app12157898.

[7] M. Garcia-Martin and G. Camarillo, 'Extensible Markup Language (XML) Format Extension for Representing Copy Control Attributes in Resource Lists', Art. no. rfc5364, Oct. 2008, doi: 10.17487/RFC5364.

[8] M. Jones, J. Bradley, and N. Sakimura, 'JSON Web Token (JWT)', RFC Editor, RFC7519, May 2015. doi: 10.17487/RFC7519.

[9] S. Josefsson and I. Liusvaara, 'Edwards-Curve Digital Signature Algorithm (EdDSA)', RFC Editor, RFC8032, Jan. 2017. doi: 10.17487/RFC8032.

Funded by Horizon Europe Framework Programme of the European Union

APPENDIX A

This appendix will extend this final report with information about the Smart Contracts developed and deployed in the Fabric Network, the main characteristics of the new Smart Contracts, and how the XACML interacts with the Hyperledger Fabric to execute the Smart Contracts. Specifically, two new Smart Contracts have been developed and integrated into the Blockchain network, which will help to extend the functionality of the latest components integrated into the FLUIDOS architecture.

First of all, it is worth mentioning that a REST API has been developed and deployed with the functionality to interact with the Blockchain. This REST API offers different endpoints that allow the XACML to make requests to register/retrieve/update policies or attributes, which removes the need to implement and run a Gateway that connects to the Blockchain network and interacts with Smart Contracts for each security component added to the FLUIDOS architecture that requires interaction with the Blockchain. In this way, the REST API would be responsible for processing the request and executing the Gateway that connects to the Hyperledger Fabric network, which allows the REST API to interact with Smart Contracts.

A.1 XACML POLICIES AND ATTRIBUTES SMART CONTRACT

The first Smart Contract deployed allows XACML to store policies and attributes defined in XACML. This chaincode has been developed in Go and includes an index that helps to query the database to obtain attributes or policies. The format in which the XACML stores policies and attributes is very similar, as both share most of the fields of the JSON structure stored in the Blockchain. This JSON structure can be shown in *Figure 21* for attributes and in *Figure 22* for policies, and includes the following fields:

- Id: indicates the domain in which these attributes/policies were defined. It also indicates if the JSON is for attributes by including 'attributes' before the domain name, or policies by including 'xacml' before the domain name.
- Type: indicates which information is included in the JSON, 'attributes' for attributes, and 'xacml' for policies.
- Timestamp: indicates the last time the attributes/policies were modified. This is useful to check if the attributes/policies JSON stored in the Blockchain is older than the local version, which means that the XACML will have to update the policies/attributes in the Blockchain.
- Version: indicates the actual version of the policies/attributes. It increases by one every time the policies/attributes are updated.
- XACML/Attributes: this is the only difference between the two elements. For the attributes, this field name is 'attributes' and indicates the attributes defined for a specific domain in XML format. Then, for policies, this field name is 'xacml' and it indicates the policies defined for a specific domain in XML format. This field is updated every time the policies/attributes are modified, so for each domain, there

Funded by Horizon Europe Framework Programme of the European Union

will be only two entries in the Fabric database, one that stores all policies defined, and the other one that stores all the attributes defined.


```
{
  "attributes": {
    "type": "Property",
    "value": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<attributes storingDate=\"2024-09-24T07:02:29.822449Z\">\n
      <attributes>\n    <attribute name=\"test\" xacml_id=\"urn:oasis:names:tc:xacml:1.0:resource:resource-id\"
      sortedValue=\"resource\" xacml_DataType=\"#string\"/>\n    <attribute name=\"GET\" xacml_id=\"urn:oasis:names:tc
      :xacml:1.0:action:action-id\" sortedValue=\"action\" xacml_DataType=\"#string\"/>\n    <attribute name
      =\"FluidosNode\" xacml_id=\"urn:ietf:params:scim:schemas:core:2.0:id\" sortedValue=\"subject\" xacml_DataType
      =\"#string\"/>\n  </attributes>\n</attributes>\n"
  },
  "id": "urn:ngsi-ld:attributes:fluidosOpencall",
  "timestamp": {
    "type": "Property",
    "value": "2024-09-24 07:02:29"
  },
  "type": "attributes",
  "version": {
    "type": "Property",
    "value": "2"
  }
}
```


Figure 21 JSON structure for storing XACML Attributes in Hyperledger Fabric


```
{
  "id": "urn:ngsi-ld:xacml:fluidosOpencall",
  "timestamp": {
    "type": "Property",
    "value": "2024-09-24 07:03:03"
  },
  "type": "xacml",
  "version": {
    "type": "Property",
    "value": "2"
  },
  "xacml": {
    "type": "Property",
    "value": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><PolicySet xmlns=\"urn:oasis:names:tc:xacml:2.0:policy:schema
      :os\" PolicyCombiningAlgId=\"urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable\"
      PolicySetId=\"POLICY_SET\">  <Target/>  <Policy PolicyId=\"Policy1\" RuleCombiningAlgId=\"urn:oasis:names:tc:xacml
      :1.0:rule-combining-algorithm:first-applicable\">    <Target/>    <Rule Effect=\"Permit\" RuleId=\"rul1\">
      <Target>        <Subjects>          <Subject>            <SubjectMatch MatchId=\"urn:oasis:names:tc:xacml:1.0
      :function:string-equal\">              <AttributeValue DataType=\"http://www.w3.org/2001/XMLSchema#string\"
      >FluidosNode</AttributeValue>              <SubjectAttributeDesignator AttributeId=\"urn:ietf:params:scim:schemas
      :core:2.0:id\" DataType=\"http://www.w3.org/2001/XMLSchema#string\"/>            </SubjectMatch>
      </Subject>        </Subjects>        <Resources>          <Resource>            <ResourceMatch MatchId=\"urn:oasis
      :names:tc:xacml:1.0:function:string-equal\">              <AttributeValue DataType=\"http://www.w3.org/2001
      /XMLSchema#string\">test</AttributeValue>              <ResourceAttributeDesignator AttributeId=\"urn:oasis:names
      :tc:xacml:1.0:resource:resource-id\" DataType=\"http://www.w3.org/2001/XMLSchema#string\"/>
      </ResourceMatch>          </Resource>        </Resources>        <Actions>          <Action>
      <ActionMatch MatchId=\"urn:oasis:names:tc:xacml:1.0:function:string-equal\">              <AttributeValue DataType
      =\"http://www.w3.org/2001/XMLSchema#string\">GET</AttributeValue>              <ActionAttributeDesignator
      AttributeId=\"urn:oasis:names:tc:xacml:1.0:action:action-id\" DataType=\"http://www.w3.org/2001/XMLSchema#string\"
      />            </ActionMatch>          </Action>        </Actions>      </Target>    </Rule>  </Policy></PolicySet
      >"
  }
}
```


Figure 22 JSON structure for storing XACML Policies in Hyperledger Fabric

A.1.1. XACML Policies/Attribute registration

In terms of functionality, this Smart Contract also offers simple methods to manage policies and attributes. For registration, the 'id' field of the JSON mentioned before is used as the

Funded by Horizon Europe Framework Programme of the European Union

key for the storage in the Blockchain, and the value would be the JSON from the figures above. To do this, XACML will send a POST request to the specific endpoint of the REST API that offers this functionality, including in the request body the JSON the XACML wants to register in the Blockchain. *Figure 23* shows the process to register the attributes and policies in the Blockchain.

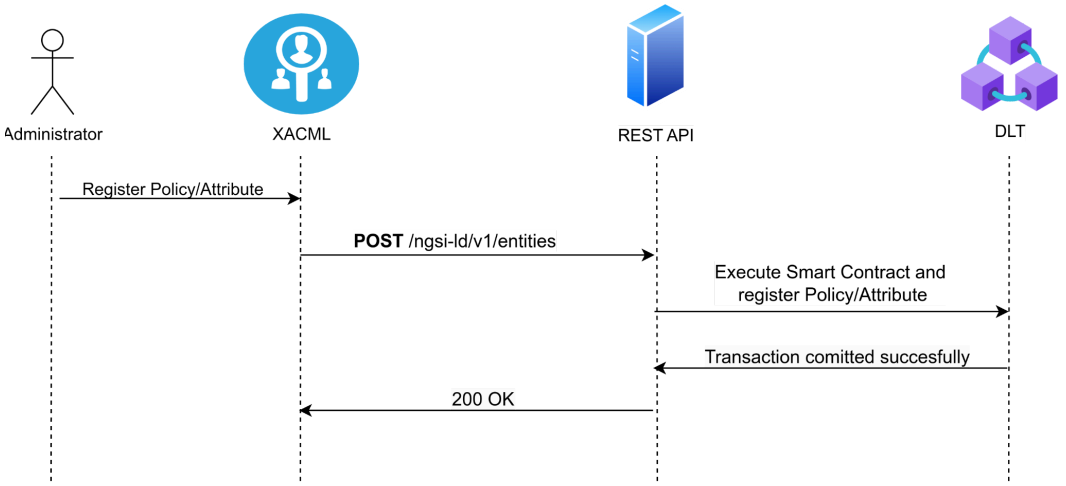

Figure 23 Sequence diagram for registering XACML Policies/Attributes in Hyperledger Fabric

A.1.2. XACML Policies/Attributes update

Once the first entry of the policies and attributes for a specific domain has been registered for the first time, it can be modified as the Smart Contract implements the functionality to update values stored in the Blockchain, whether to add a policy, delete an attribute, or update a rule. In this case, the XACML sends a PATCH request to the specific endpoint of the REST API that provides this functionality, including in the request body the 'version', 'timestamp', and 'xacml/attributes' JSON fields with the updated values. The 'id' and 'type' fields are not included in the request body as they are embedded in the request's URL. The process and steps to update policies and attributes can be shown in *Figure 24*.

Funded by Horizon Europe Framework Programme of the European Union

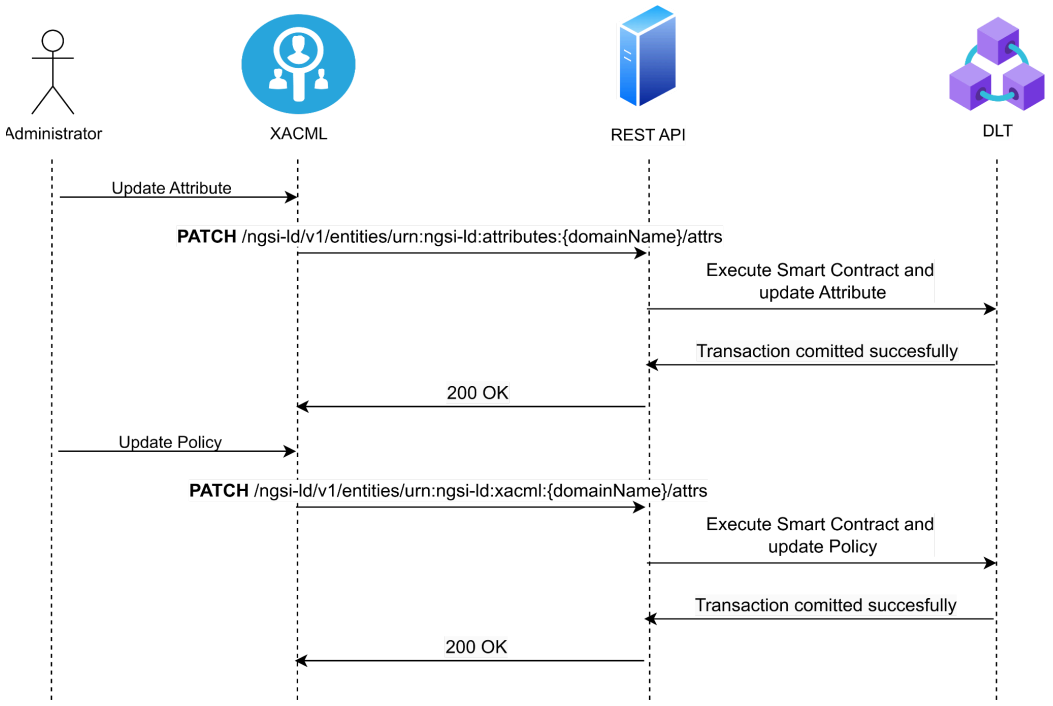

Figure 24 Sequence diagram for updating XACML Policies/Attributes in Hyperledger Fabric

A.1.3. XACML Policies/Attributes Query

There are two methods defined to retrieve policies and attributes stored in Blockchain, one for retrieving policies/attributes from a specific domain, and another one for retrieving all policies/attributes defined in the XACML. For the first case, the XACML will make a GET request to the specific endpoint in the REST API that offers that functionality, including in the URL the specific ID that the XACML wants to obtain, e.g., to retrieve the attributes from the 'fluidosOpencall' domain the Id would be 'urn:ngsi-ld:attributes:fluidosOpencall'. *Figure 25* shows the whole process of obtaining specific attributes or policies.

Funded by Horizon Europe Framework Programme of the European Union

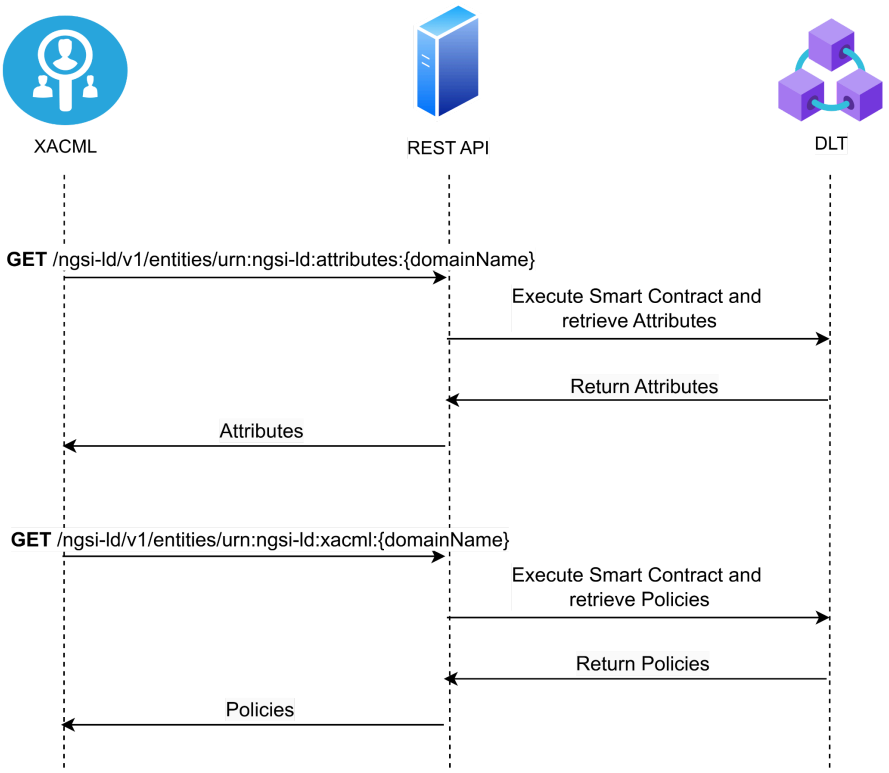

Figure 25 Sequence diagram for querying XACML Policies/Attributes for a specific domain from Hyperledger Fabric

For the second case, where the XACML wants to obtain all policies and attributes for all domains registered in XACMI, it will make a GET request to the specific endpoint in the REST API that offers that functionality, including as query parameter in the URL the 'type' it wants to retrieve, where the possible values for 'type' are 'attributes' to retrieve the attributes and 'xacml' to retrieve the policies. The whole flow to retrieve all policies and attributes from all domains can be shown in *Figure 26*.

Funded by Horizon Europe Framework Programme of the European Union

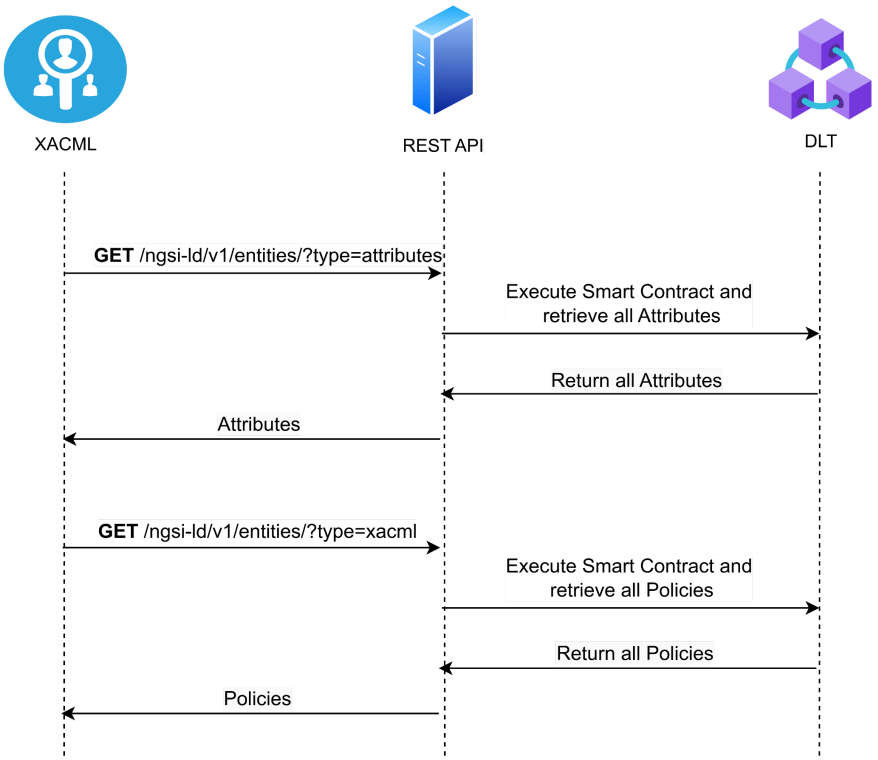

Figure 26 Sequence diagram for querying all XACML Policies/Attributes for all domains from Hyperledger Fabric

A.2. XACML AUTHORIZATION REQUESTS SMART CONTRACT

The second Smart Contract that has been deployed in the Hyperledger Fabric network allows XACML to register the authorization requests it receives from the PSM on the Blockchain, which allows for the accounting of authorization requests. This Smart Contract also allows retrieving the stored requests, so it is possible to observe and analyse the historical record of the authorization requests. Furthermore, an index has also been deployed in this Smart Contract with the functionality to retrieve all authorization requests between two dates. This retrieval method has been added as an example of the possibilities the Blockchain and indexes offer for retrieving data, so it is possible to extend this by adding more retrieval methods in the future. The JSON structure that is stored in the Blockchain and contains all the information about the authorization request can be seen in *Figure 10*. Then, in terms of functionality, this Smart Contract also offers simple methods for the management of authorization requests, which are the registry and retrieval of requests. In this case, unlike what happened with the Smart Contract for attributes and policies, this Smart Contract does not offer the functionality to update stored values, as this would violate the integrity of the registered data, thus eliminating any validity of the authorization requests registered on the Blockchain.

Funded by Horizon Europe Framework Programme of the European Union

A.2.1. XACML Authorization requests registration

In the case of the registry, the XACML will send a POST request to the specific endpoint in the REST API, including in the request body the JSON that will be registered in the Blockchain. *Figure 27* shows the whole process of registering authorization requests in the Blockchain.

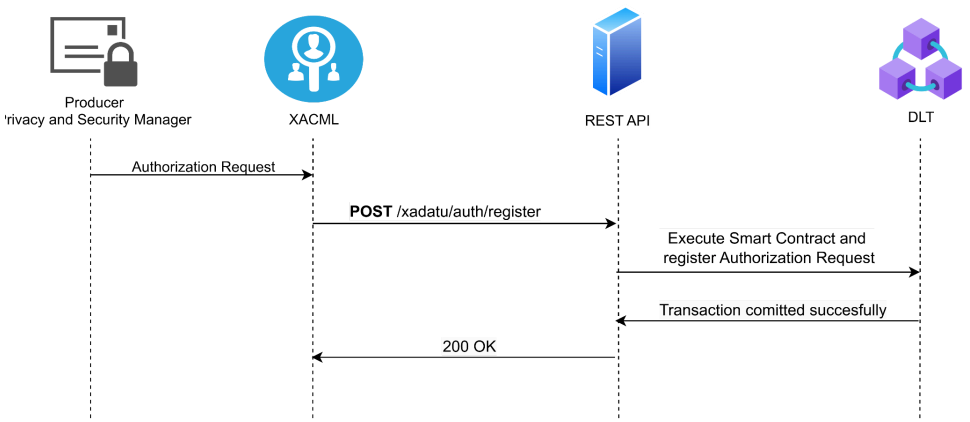

Figure 27 Sequence diagram for registering XACML authorization requests in Hyperledger Fabric

A.2.2. XACML Authorization requests query by id

Next, for retrieving authorization requests stored in Blockchain, the functionality implemented includes two different methods, as mentioned before: one for retrieving a specific authorization request, and another one for retrieving all authorization requests between two dates. For the first case, the user must send a GET request to the specific endpoint in the REST API, including the ID of the specific authorization request to be retrieved in the URL as shown in *Figure 28*.

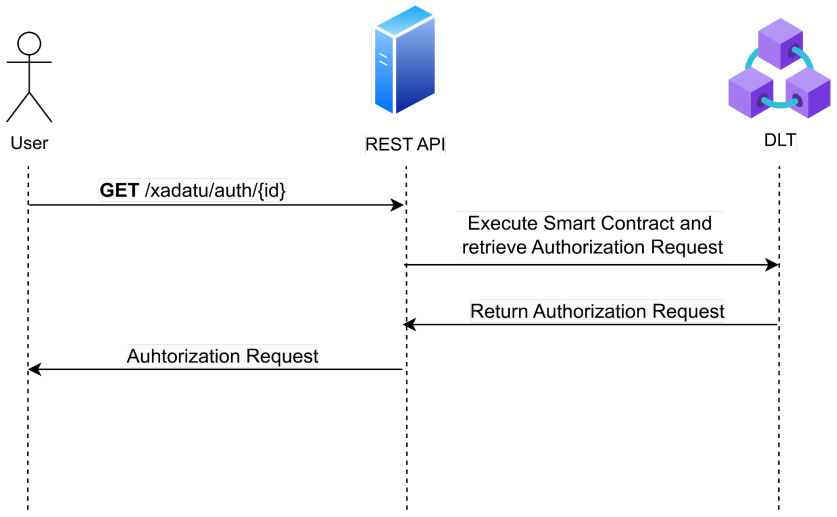

Figure 28 Sequence diagram for querying a specific XACML authorization request from Hyperledger Fabric

Funded by Horizon Europe
Framework Programme of the European Union

A.2.3. XACML Authorization requests query by date

Moreover, to retrieve all authorization requests between two dates, a GET request must be sent to the specific endpoint of the REST API that offers this functionality. The request body should include a JSON with two fields: 'startDate' containing the start date and time in ISO 8601 format, and 'endDate' with the end date and time in ISO 8601 format for the query. *Figure 29* shows the flow diagram of the process to retrieve all authorization requests between two specified dates.

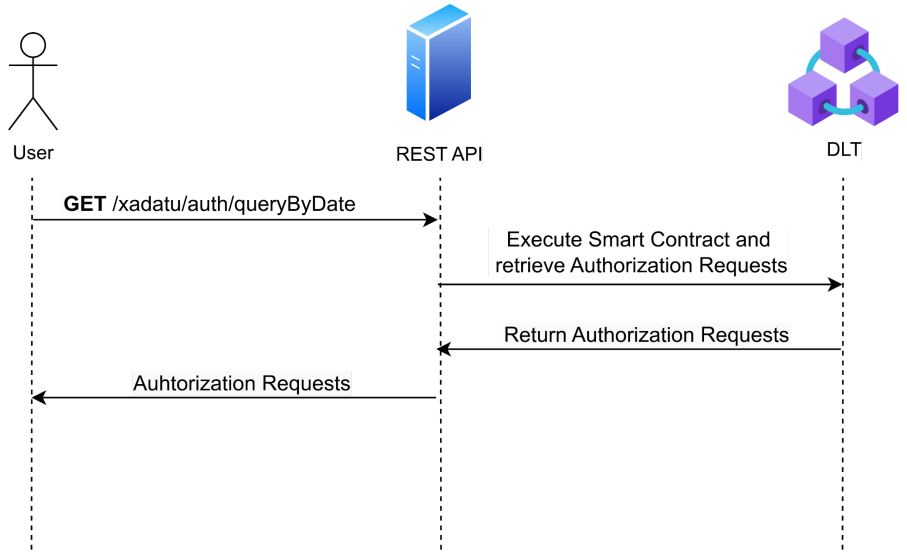

Figure 29 Sequence diagram for querying all XACML authorization requests between two dates from Hyperledger Fabric

A.2.4. XACML Authorization Requests Query By DID

This query returns all authorization requests made by a specific DID. To do that, a GET request must be sent to the 'xadatu/auth/queryByDid' endpoint of the REST API server. The requests body must include a JSON with a single field, *did*, which specifies the DID of the authorization requests to be retrieved. This query process can be shown in *Figure 30*.

Funded by Horizon Europe Framework Programme of the European Union

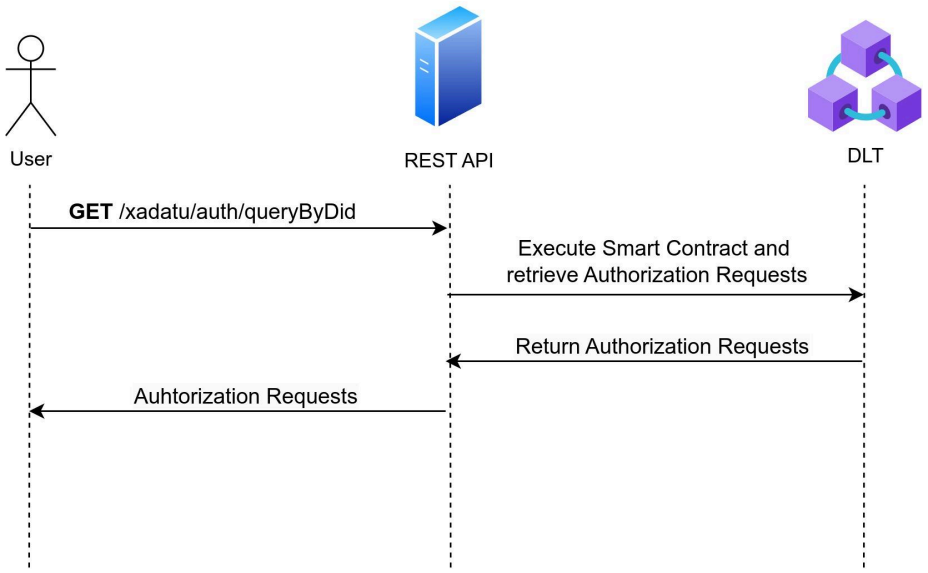

Figure 30 Sequence diagram for querying all XACML authorization requests by DID

A.2.5. XACML Authorization Requests Custom Query

Finally, there is the possibility to make custom queries to retrieve authorization requests from the Blockchain to provide maximum flexibility. This endpoint enables users to perform any type of query they need by directly specifying the desired query structure in the request body. This query structure is a mango query, and an example of this query can be shown in *Figure 31*. This query retrieves authorization requests where the timestamp is greater than the specified date, the *Action* is 'GET', the *Decision* is 'Permit', and the *Subject* is 'FluidosNode', i.e., all authorization requests from the specified subject that were accepted starting from the specified date. Then, the query process can be shown in *Figure 32.*

```
{
    "selector": {
        "Timestamp": {
            "$gt": "2024-09-05T15:30:00Z"
        },
        "Action": "GET",
        "Decision": "Permit",
        "Subject": "FluidosNode"
    }
}
```

Figure 31 Example of a custom query structure to retrieve authorization requests

Funded by Horizon Europe Framework Programme of the European Union

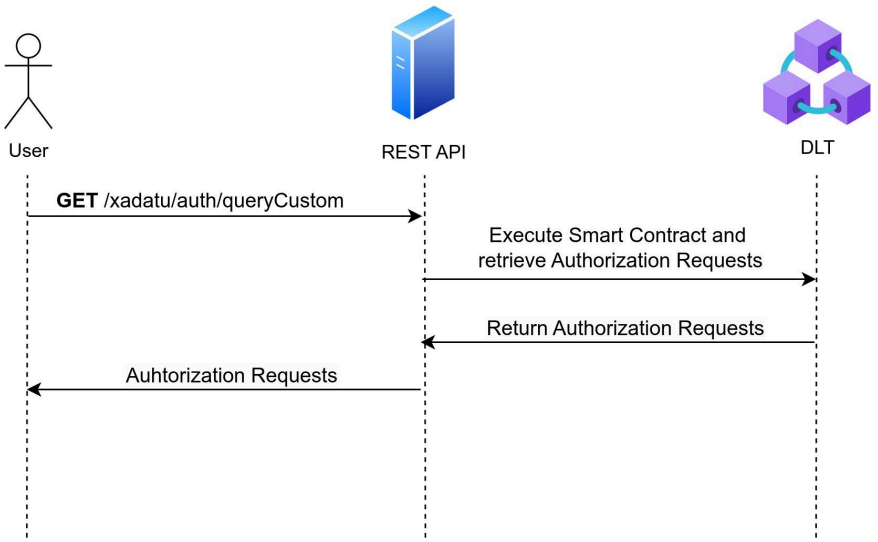

Figure 32 Sequence diagram for querying authorization requests using a custom query

Funded by Horizon Europe
Framework Programme of the European Union