HORIZONS 2020 PROGRAMME

Research and Innovation Action – FIRE Initiative

Call Identifier:	H2020–ICT–2014–1
Project Number:	643943
Project Acronym:	FIESTA-IoT
Project Title:	Federated Interoperable Semantic IoT/cloud Testbeds and Applications

D4.2.1 Techniques for Secure Access and Reservation of Resources

Document Id:	FIESTAIoT-WP4-D4.2.1-SecureAccessAndReservation
File Name:	FIESTAIoT-WP4-D4.2.1-SecureAccessAndReservation-140716-V10.docx
Document reference:	Deliverable 4.2.1
Version:	V10
Editor:	ITINNOV
AUTHORS:	ITINNOV, UNICAN
Date:	22 / 08 / 2016
Document type:	Deliverable
Dissemination level:	PU

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Paul Grace, Vadims Krivcovs	ITINNOV	2015/07/22	Initial content: security analysis, federation styles, and reservation.
V02	Vadims Krivcovs	ITINNOV	2015/08/11	Updating core document sections
V03	Paul Grace, Vadims Krivcovs	ITINNOV	2016/02/2	Description of FIESTA-IoT Security Framework based on 1st Integration Meeting in Santander
V04	David Gómez Fernández	UNICAN		Material about securing and protecting FIESTA-IoT Testbeds
V05	Paul Grace	ITINNOV	25/07	Update to Security Framework architecture description and progress after 2nd Integration Meeting in Paris
V06	Michael Boniface	ITINNOV	29/07	Update to use cases. This is the version sent for internal review
V07	Rachit Agarwal	INRIA	05/08/2016	Technical Review 1
V08	Amelie Gyrard	NUIG	09/08/2016	Technical Review 2
V09	Konstantinos Bountouris	COM4INNOV	09/08/2016	Quality Review
V10	Paul Grace, David Gómez Fernández	ITINNOV, UNICAN	24/08/2016	Revisions to address Technical Review 1 and 2; and Quality Review

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND ACRONYMS

AAT	Authorization API Token
ABAC	Attribute-Based Access Control
AM	Authorization Manager
Authn	Authentication
Authz	Authorization
CAS	Central Authorization Service
CMD	Command Line
EaaS	Experiment as a Service
FIRE	Future Internet Research and Experimentation
FITeagle	Semantic Resource Management Framework
FQDN	Fully Qualified Domain Name
FRCP	Federated Resource Control Protocol
GENI	Global Environment for Network Innovations
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IdP	Identity Provider
JOSE	JSON Object Signing and Encryption
JWT	JSON Web Token
J2EE	Java 2 Platform Enterprise Edition
LDAP	Lightweight Directory Access Protocol
NEPI	Network Experiment Programming Interface
OAuth	Open standard for authorization (Open Authorization)
OEDL	OMF Experiment Description Language
OMF	Orbit Management Framework – a reference implementation of FRCP
OpenID Connect	Simple identity layer on top of the OAuth 2.0 protocol
OpenIG	Open Identity Gateway
PAP	Policy Administration Point
PAT	Protection API Token
PEP	Policy Enforcement Point
PDP	Policy Decision Point

REST	Representational State Transfer
RPT	Requesting Party Token
RS	Resource Server
SA	Slice Authority
SAML	Security Assertion Markup Language
SaaS	Software as a Service
SFA	Slice Federation Architecture
SSO	Single Sign On
TLS	Transport Layer Security
UC	Use Case
UMA	User-Managed Access
XACML	eXtensible Access Control Markup Language

EXECUTIVE SUMMARY

This document presents the first version of the security architecture that will secure access to and usage of the FIESTA-IoT applications, services and testbeds.

An analysis of security technologies that can applied to the federation of IoT testbeds is first carried out; this covers single-sign on authentication; authorization; and access control functionality. We then make recommendations for the technologies that should be investigated in greater depth. Further, there is an investigation of federation examples that are relevant to FIESTA-IoT, e.g. the Fed4FIRE, and XIFI federations. We analyse each of these technologies and solutions against the FIESTA-IoT security requirements, and choose a particular security technology that best meets these needs. This technology is the OpenAM platform.

Subsequently, we present the design of the FIESTA-IoT security architecture. This describes how OpenAM and its related technologies (e.g. the OpenIG gateway) are deployed in order to secure the FIESTA-IoT platform, i.e., securing the FIESTA-IoT portal, securing the FIESTA-IoT tools, and finally, securing the testbeds providing the IoT data resources.

Finally, this deliverable looks at the deployment and implementation of the security architecture. It describes how various resources have been secured using OpenAM and OpenIG in the first version of the FIESTA-IoT platform. Concluding the document, we then look to the future versions of the FIESTA-IoT security architecture and what additional features and functionality will be provided.

1 INTRODUCTION

1.1 Overview of WP4

FIESTA-IoT WP4 implements an infrastructure for accessing data and services from multiple distributed diverse testbeds in a secure and testbed agnostic way. To this end, it will rely on the semantic interoperability of the various testbeds (realized in WP3) and implement a single entry point for accessing the FIESTA-IoT data and resources seamlessly and according to an on-demand EaaS model. The infrastructure to be implemented will be deployed in a cloud environment and will be accessible through a unified portal infrastructure.

The objectives of WP4 are:

- To specify and implement the testbed agnostic operations (e.g., data access, service execution, data/service management) that comprise the EaaS model of the project, along with tools and techniques for combining them into experiments/workflows.
- To specify and implement tools and techniques for accessing resources from the various testbeds in a secure way.
- To specify and implement tools and techniques for testbed agnostic access to data sets stemming from multiple heterogeneous IoT platforms.
- To specify and implement a portal infrastructure enabling the submission of experiments over semantically interoperable testbeds.
- To design and implement tools and techniques for managing experiments, including management of information the sensors and data streams that the experiments use, the volume of data that they consume/use, the timing of the execution of the various experiments and more.

This deliverable describes the technology solutions created in Task **T4.2 Techniques for Secure Access and Reservation of Resources**:

> *"This task will ensure that experimenters will have secure access to the resources of the federated testbeds. It will deal with the provision of a solution for authentication and authorization of resources (e.g., devices, data streams), which will be operational independently of the number and type of underlying IoT platforms and testbeds. This solution will enable experimenters to gain access to all the FIESTA testbeds through a single set of credentials. Furthermore, the task will research and provide techniques for reserving resources through keeping track of the various experiments, the data streams and IoT resources that they use, the lifetime of the experiments, logging information about the experimenters and the experiments that they execute and more. By keeping track of the resources reserved/used (at the level of experiments and experimenters) this task will provide a foundation for managing experiments and the IoT/cloud resources that they use. In terms of the implementation of the low-level security and resource reservation functionalities, results for related projects (e.g., Fed4FIRE) will be studied and appropriately reused".*

1.2 FIESTA-IoT Security Introduction

1.2.1 Security Requirements

The work to secure the FIESTA-IoT architecture is closely related to the majority of architectural components and testbed resources; that is, such elements must be secured in order that only authorized FIESTA-IoT users can access and utilize them. The security requirements identified in the initial FIESTA-IoT requirements specification (from Deliverable D2.1 **[1]**), and listed in Table 1 define the features and functionality that the FIESTA-IOT security framework must achieve. Note, from this point forward we define the ***FIESTA-IoT security framework*** to be the components and technologies that work together to secure the FIESTA-IoT architecture.

Table 1 – FIESTA-IoT Security requirements

UNIQUE ID	Requirement Type	Priority	Description
18_FR_SEC_Testbed_authentication_mechanisms	Functional Requirements	MUST	Testbed providers must provide authentication mechanisms to (secure) access services/resources
19_FR_SEC_Testbed_manage_priviledges	Functional Requirements	MUST	Testbed providers must provide authorization and access control mechanisms to (secure) access services/resources
20_FR_SEC_Experimenter_single-sign-on	Functional Requirements	MUST	Single-sign-on mechanism has to be in place
21_FR_SEC_Tool_manage_users	Functional Requirements	MUST	FIESTA-IoT must have a tool to manage users and their respective access rights
68_NFR_SEC_Support_certification_authority	Non-Functional Requirements	SHOULD	FIESTA-IoT Platform should support federated identity management
69_NFR_SEC_Verify_authorise_user_actions	Non-Functional Requirements	MUST	When a user wants to execute any action FIESTA-IoT has to verify that he is authorized to do this action
70_NFR_SEC_Different_profile_types	Non-Functional Requirements	MUST	FIESTA-IoT must have different profile types (e.g. different kind of experimenters, researchers, etc.). Thereby grant different permissions
71_NFR_SEC_Privacy_collected_data	Non-Functional Requirements	COULD	Privacy of collected data

1.2.2 Security Roles

Within the FIESTA-IoT platform there is a set of core roles (these may be extended to more fine-grained roles in the future) that define which user gets access to which resources:

- **FIESTA-IoT Security Manager:** This role is in charge of admin of security configuration and requests. When a testbed or experimenter sends a request to participate they are responsible for authenticating their request.

- **Observer:** This is the most restricted role in FIESTA-IoT. This user has access to some of the portal resources and applications e.g. discovering what

testbeds and data is available, what the experimental tools look like etc. However, they are not able to execute experiments or extract any detailed data. This role is essentially a taster session for the user to evaluate if FIESTA-IoT provides what they wish for. In terms of registration and sign-on—this role uses self-registration; the user fills out a form asking to be an observer, FIESTA-IoT mails them a validation link to their e-mail address. Subsequently, they can log onto the FIESTA-IoT portal as an observer.

- **Experimenter:** This role allows the user to create and execute experiments within the FIESTA-IoT platform. The experimenter registers for FIESTA-IoT via the portal; the FIESTA-IoT Security Manager then reviews their request, authenticates the user as a valid experimenter and assigns them the role.

- **Testbed Admin:** A user from the testbed administration team, whose role is to configure their integration with FIESTA-IoT. This includes the registration of the testbed resources, and the definition of any security policies. The default policy is to allow FIESTA-IoT experimenters to access each testbed resource. However, the testbed themselves can add further more fine-grained policies.

1.2.3 FIESTA-IoT Logical Security Architecture and Use Cases

The FIESTA-IoT architecture defined in Deliverable D2.4 **[2]** describes the logical components that form the security framework of FIESTA-IoT, and the use cases related to secure access and usage of FIESTA-IoT resources. In addition D2.4 **[2]** provides a set of use cases (UC) to drive the design and specification of the security framework. These are summarized below (the full descriptions and diagrams can be found in deliverable D2.4 **[2]**).

Figure 1 highlights the key components in the use cases (where experimenters attempt to access FIESTA-IoT services and testbed resources):

- The **AuthN** component which is the Identity Manager in the FIESTA-IoT security framework. It deals with user registration, user account management, and user authentication.
- The **AuthZ** component which is the authorization and access management component. It deals with the creation and update of access policies for all FIESTA-IoT resources—for both FIESTA-IoT services (e.g. the portal) and the individual testbeds (e.g. the data APIs to retrieve testbed data). It is also the Policy Decision Point (PDP)—when access to a resource is attempted the AuthZ component evaluates the request against the polices and makes a decision whether to grant or deny the request.
- The **Meta-Cloud Data Endpoint**, which is the FIESTA-IoT endpoint to allow an experimenter to access the data resources from the testbed. Hence, the policy enforcement point (PEP), i.e. where policies are enforced, is deployed here.

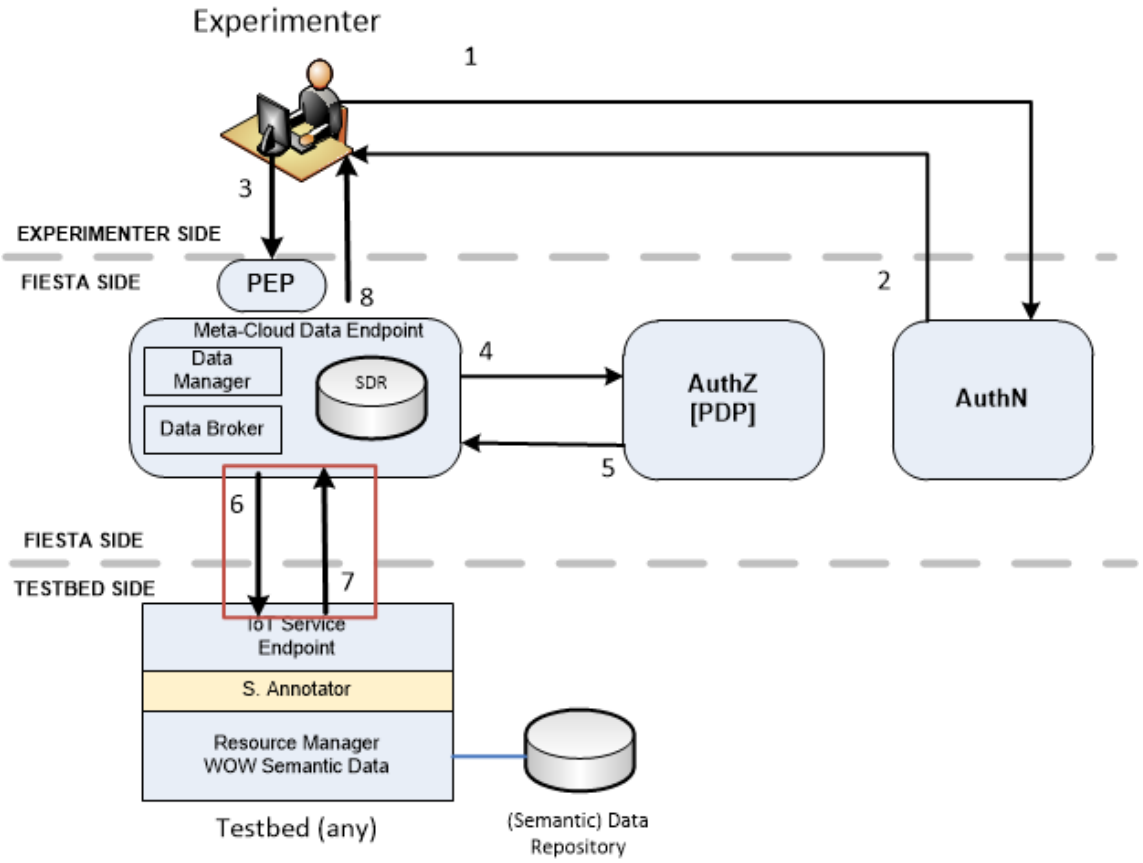

Figure 1: Secure Access to Protected FIESTA-IoT Resources

UC1 - Protected Resource Access

Resources in the FIESTA-IoT federation (e.g. the FIESTA-IoT Meta-Cloud services and applications, and the testbed resources), are protected and require authorization in order to be accessed by only FIESTA-IoT experimenters. The logical security architecture and sequence flow is illustrated in **Error! Reference source not found.**. Note, the resource presented here is the IoT Service Endpoint (i.e. the API to access the testbed data), but the security pattern can be equally applied to the other types of FIESTA-IoT resources.

Each request is checked in order to ensure that the request is from an authenticated user (e.g. experimenter), and that they are authorized to perform the request. Resource access in FIESTA-IoT follows a traditional Policy Enforcement Point (PEP) Pattern – requests are intercepted by the PEP (at FIESTA-IoT endpoints) and these are sent to a Policy Decision Point (PDP) component, which forms part of the logical AuthZ component). The PDP implements the grant/deny decision when evaluating the request against the access policy.

Resource Access can be managed in two ways in FIESTA-IoT.

In the first version of the FIESTA-IoT security framework (described in this deliverable) – access to resources is fully managed by FIESTA-IoT based upon inputs from the testbed admin:

- **Fully managed by FIESTA-IoT:** The testbed trusts FIESTA-IoT to forward only authorized requests to the testbed (enforcing the policies described by the testbed in use case 3). That is, the testbed registers its policies with FIESTA-IoT, and then trusts FIESTA-IoT to enforce these policies. A secure trusted channel from FIESTA-IoT to the testbed API is managed and implemented by the testbed. For example, using digital certificates or API keys (i.e. their own security choice).

In future versions of the FIESTA-IoT security framework we plan to allow more decentralized control of access. That is, the testbed may manage the access themselves.

- **The testbed manages access on their endpoints*:*** Instead of the AuthZ component operating at the FIESTA-IoT side it operates at the testbed (i.e. it does not need to trust FIESTA-IoT to execute access control). Hence, the PDP executes on site. This case is suited to testbeds who may want greater control over FIESTA-IoT access—for example, where the resources are shared beyond FIESTA-IoT and there is a need to ensure FIESTA-IoT does not starve other users of resources.

UC2 - Experimenter registration/ Identity Management

An experimenter signs up to use FIESTA-IoT (i.e. they register a new user account and username and password credentials). In this use case, FIESTA-IoT is the central identity provider manager in the testbed federation—when a FIESTA-IoT user wants to access either a FIESTA-IoT service, or a testbed's data API they must first authenticate with the central identity manager (part of the FIESTA-IOT security framework). The steps of this use case are as follows:

1. The experimenter selects a sign-up link on the FIESTA-IoT portal web page;
2. The experimenter fills in his/her information including e-mail address and password (security credentials selects the **role** they wish to register for (**Experimenter**, **Testbed Admin**, or **Observer**). Roles are discussed further in Section 1.2.2.
3. Where the user signs up as an observer, a self-service approach is followed. For experimenter and testbed admin, the request is passed to a FIESTA-IoT member to verify. The experimenter will then be sent an email at his/her registered email address for verification of his/her identity. When the user verifies via the link, registration is complete and the use case continues.

The experimenter is now free to authenticate, i.e., they can sign-in to use services. Whenever, a FIESTA-IoT user tries to access a service (e.g. using a browser) they will be redirected to a log-in page to authenticate.

The other sub use-cases of the experimenter identity management are:

- The experimenter can update their account info any time using the FIESTA-IoT portal, i.e. change the information FIESTA-IoT has stored about them e.g. name and contact details;
- The experimenter can delete his/her account from the system at any time using the FIESTA-IoT portal;

- The FIESTA-IoT administrator can delete an experimenter's account from the system at any time using the FIESTA-IoT administration portal.

UC3 – Testbed Resource Management

Each testbed controls access to their resources using access control policies. Each incoming request to use testbed resources is checked against these policies to determine if the request will be granted or denied.

- The access policies are stored centrally in the FIESTA-IoT security framework.
- Only a testbed administrator of a given testbed can manage (change, add, delete, etc.) the policies of a given testbed (even though it is stored centrally).

Therefore, testbed owners, manage their existing resources at the testbed level. Each testbed is a set of static resources to be protected. Testbed owners can add to these over time by registering new resources in the testbed. The testbed owners, when registering resources, will add control policies for these resources. The testbed administrator can directly alter both specific resource policies and default policies by interacting with the AuthZ component via the Access Policy Administration Tool. A default policy may be:

- "All authenticated FIESTA-IoT experimenters can access and use all operations on the testbed API."

Example specific access policies may be:

- Experimenter ID1 can perform GET operations on testbed interface A.
- Experimenter ID2 can perform GET & PUT operations on testbed interface A.

1.2.4 Resource Reservation

In the first version of the FIESTA-IoT architecture, the testbed resources made available are data centric. That is, an experimenter can query for data and subscribe to data streams. Such resources are therefore inherently shareable – multiple experimenters can access and utilize the resources at the same time without conflict. Compare this with a testbed providing computational resources (e.g. virtual machines)—only so many resources can be provided and therefore to ensure that demands are met – the resources must be reserved in advance. Without such limitations in FIESTA-IoT a dedicated framework to reserve resources is not required—and hence in the first version of the FIESTA-IoT security and reservation framework we provide no functionality beyond the ability to use access control to manage the usage of resources. For example, if a testbed wishes to allow individual experimenters access to a resource at a time—they can define a policy for this.

In the future, FIESTA-IoT may encompass IoT testbeds with the need for advance reservation, e.g. a set of actuators that can be controlled by an experimenter during a particular time period. We envisage that access policies can again be used to control such behavior—however, this would need to be integrated with a calendar service (where reservations for time periods are agreed) to provide input to the policy decision point. We will investigate this further in future version of the FIESTA-IoT security and reservation framework.

1.3 Document overview

In this document we focus on the security framework and functionality required across the FIESTA-IoT functional architecture. For this purpose, we present the following:

- An analysis of security technologies that can be applied to the FIESTA-IoT federation; this covers single-sign on authentication; authorization and access control functionality. We make recommendations for the technologies that should be investigated further.
- An analysis of federation examples that are relevant to FIESTA-IoT, e.g. Fed4FIRE[1], and XIFI[2]. We provide a discussion of what FIESTA-IoT can learn from the services and tools provided in these initiatives. This is taken from the security and reservation viewpoint.
- Presentation of the initial security framework within FIESTA-IoT.
- A description of the OpenAM and OpenIG technologies that have been chosen to implement the security framework. We also identify how these two technologies are deployed and used to secure specific FIESTA-IoT services and resources.
- A discussion of the future directions of the FIESTA-IoT security framework.

[1] http://www.fed4fire.eu

[2] https://fi-xifi.eu/home.html

2 BACKGROUND & RELATED WORK: SECURITY TECHNOLOGIES

This section provides an overview of security technologies relevant to secure access to the FIESTA-IoT experiment-as-a-service architecture, and the data resources provided by the testbeds in the FIESTA-IoT federation. Authentication and Authorization are the most important security technologies to consider for this purpose; hence, in turn we describe technologies in these domains, focusing in particular on approaches that have been employed to secure experimental facilities and IoT platforms. We then analyse these solutions with respect to meeting the FIESTA-IoT security requirements. Finally, the section concludes with the security choices made for FIESTA-IoT—based upon this analysis.

2.1 Authentication and Single Sign On (SSO)

Authentication is the process of identifying that someone is who they say they are. SSO (typically using a username and password) is the ability to authenticate using a single set of credentials in order to access multiple services, and in multiple domains. Currently the most common SSO techniques are based on username/password, Security Assertion Markup Language (SAML), and more modern OpenID Connect that was built on top of the new OAuth 2.0 standard.

2.1.1 SAML 2.0

Any conversation about web authentication standards and Web SSO must begin with the Security Assertion Markup Language (SAML) **[3]**. This is the current leading standard for enterprise inter-domain authentication. It is widely supported by off-the-shelf software, and major Software-as-a-Service (SaaS) vendors like Google, SalesForce, WorkDay, Box, Amazon, and many others. SAML is the basis for extensive Business-to-Business (B2B), government and educational networks around the globe.

Analysis

Table 2 provides a brief analysis of the key benefits and weaknesses of SAML. SAML can be seen as a reliable solution, but the switch towards devices (sensors, things) in the future (as opposed to Web technologies) means its weaknesses in this area will become apparent. This is seen in Gluu's prediction **[4]**: providing SAML endpoints and services will be critical for domains for years to come. However, in the next 15 years or so, organizations will look to consolidate on OAuth2 based trust networks, and will look to end-of-life and de-commission SAML relationships.

Table 2: SAML 2.0 analysis

+	SAML tokens provide information about identity of entity and widely used in SSO systems
+	Dominant protocol for achieving secure attribute exchange and SSO *today*
+	Mature and well defined standard
+	Widely adopted and provides good security based on signed XML documents
-	Not really suitable for mobile devices (and most likely sensor devices) without additions effort to provide broker services etc. (was primary designed for Web-

	based applications)

2.1.2 OpenID Connect

OpenID Connect **[5]** is a simple identity layer on top of the OAuth 2.0 **[6]** protocol, which allows computing clients to verify the identity of an end-user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end-user in an interoperable and REST-like manner.

OpenID Connect allows a range of clients, including Web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users. The specification suite is extensible, supporting optional features such as encryption of identity data, discovery of OpenID Providers, and session management.

Analysis

Table 3 provides a brief analysis of the key benefits and weaknesses of Open ID Connect. Its greatest advantage being that is a simple web based standard that is suited to mobile devices; hence, it is well suited to the IoT domain. However, it remains less established than other solutions; but usage is likely to grow—and many companies have already started to use OpenID Connect. These include: StruSoft, Google, Microsoft, Ping Identity, Deutsche Telekom and many more.

Table 3: OpenID Connect analysis

+	Built on top of OAuth 2.0 standard
+	Suitable for mobile devices (and possibly for sensor devices)
+	OpenID Connect tokens are designed for today's REST-based application development practices
+	Lightweight, hence suitable to mobile devices (and possible to sensor devices)
+	Simpler protocol to realise (great support for open libraries as well)
+	Potentially safer (XML-DSIG changed to JWS what eliminates a range of possible attacks)
+	Foundation for a far more efficient and scalable **enterprise federated SSO solution**
+	Compare to SAML: OpenID Connect can satisfy these same use cases but with a simpler, JSON/REST based protocol
+	Allows to choice of trust provider
-	So far not widely adopted, but the situation is rapidly changing (many companies are currently in a process of adopting it)

2.1.3 Authentication and Single Sign On technology comparison

SAML is not sufficiently interoperable to be the future standard for identity management federation. SAML is limited in its ability to support mobile & smart-TV

applications and requires the implementation of a complex broker Service **[7]** in order to support multi-service provider & multi-Identity provider use cases.

OpenID Connect is a lightweight identity verification protocol built on top of modern web standards (OAuth 2.0, REST and JSON) superseding OpenID 2.0. OpenID Connect allows a service provider (Relying Party) to select between varieties of registered or discovered identity providers. OpenID Connect can satisfy all of the SAML use cases but with a simpler, JSON/REST based protocol.

Based on the initial review of SAML 2.0 and OpenID Connect (which are currently the most widely used technologies for SSO) for Single Sign-On Authentication; OpenID Connect has the following benefits compared to SAML 2.0 which are summarized in Table 4 below and need to be considered as a strong candidate for archiving Single Sign-On Authentication in FIESTA-IoT.

Table 4: SAML and OpenID Connect comparison

Feature	SAML 2.0	OpenID Connect
Discovery Service	No (Requires pre-agreed metadata)	Single discovery service for Requesting Parties (RP) allowing sites and applications to “validate” your users
Mobile Apps	SSO not supported	SSO Supported
Support for SSO	Only web based SSO provided	SSO available from multiple device types e.g. mobiles, sensors, etc. that can implement the standard.

2.2 Technology Comparison for Authorization

This section provides an overview of access control solutions for IoT. It is assumed that data communication to/from web and IoT devices is encrypted using standards such as TLS, as well as the identity of requesting party is sufficiently verified and trusted.

Unfortunately many proposed Authorization frameworks do not provide complete and efficient mechanisms that allow scalable and effective access control solutions for IoT; that is, unbound by the number of IoT devices and users in a dynamic environment. We now discuss a subset of different technologies that are currently used (some of them are during early stages of adoption) in web and IoT access control systems.

2.2.1 OAuth 2.0

OAuth **[6]** is an open standard for Authorization that provides client applications with a “secure delegated access” to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. OAuth is built upon the following key party roles:

- **Resource owner**: an entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user. For example, a user who has stored their pictures on Picasa[1].
- **Resource server**: the server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens. For example, the Picasa server.
- **Client**: an application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g. whether the application executes on a server, a desktop, or other devices). For example, this could be a printing client – that prints photographs stored on Picasa.
- **Authorization server**: the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The interaction sequence for the OAuth 2.0 protocol between the above roles is shown in Figure 2:

A) **Authorization request phase:** the client (print application) requests Authorization from the resource owner (User with Photos). The Authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the Authorization server (OAuth server) as an intermediary.
B) **Authorization grant:** the client receives an Authorization grant, which is a credential representing the resource owner's Authorization, expressed using one of four grant types defined in this specification **[6]** or using an extension grant type. The Authorization grant type depends on the method used by the client to request Authorization and the types supported by the authorization server.
C) **Request an access token:** the client requests an access token by authenticating with the Authorization server and presenting the Authorization grant.
D) **Return an access token:** the Authorization server authenticates the client and validates the Authorization grant, and if valid, issues an access token.
E) **Request protected resource:** the client requests the protected resource from the resource server (Picasa) and authenticates by presenting the access token.
F) **Use protected resource:** the resource server validates the access token, and if valid, serves the request.

[1] https://picasa.google.co.uk/

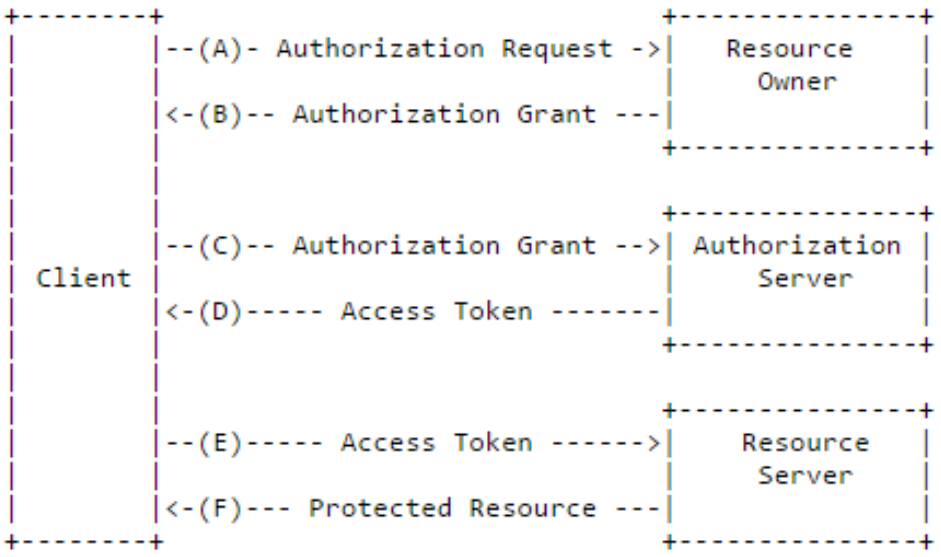

Figure 2: OAuth 2.0 abstract protocol flow[1]

Table 5: OAuth 2.0 analysis

+	Open standard for Authorization, i.e., publicly available, and developed, approved and maintained via a collaborative and consensus driven process.
+	The only framework in its genre and is widely used for similar applications.
+	More like a framework (not a defined protocol), which leaves a lot of implementation freedom that we need because of non-standard requirements.
+	Easy to configure and deploy which saves time, effort and ease development process.
+	Organisations do not need support for password renewal, forgotten password, authentication of users, and support to let users remove themselves from the service, etc.
+	Low-risk for ID theft, etc. The service already has good support to prevent this. Authentication takes place at provider, the OAuth tokens are encrypted and not in our application.
+	Gives great possibility to add new services (many well-known industry and education organisations already deployed OAuth 2.0 which allows clients to use resources using their identity).
+	User can prevent access to the application from the OAuth provider
-	No notion of identity, therefore additional effort is needed to validate an identity of requesting parties
-	No fine grained authorization (based on scopes only)
-	Requires logic to allow the user to log in with multiple OAuth providers.

[1] https://tools.ietf.org/html/rfc6749

2.2.2 User-Managed Access (UMA)

The User-Managed Access (UMA) **[8]** protocol is designed to give a web user a unified control point for authorizing who and what can get access to their online personal data (such as identity attributes), content (such as data or data streams that can contain private and sensitive information about users, entities etc.), and services, no matter where all those things live on the web. Quite importantly UMA allows a user to make demands of the requesting side in order to test their suitability for receiving Authorization. The following roles compose the UMA protocols:

- **Resource owner:** an OAuth resource owner is a "user" in UMA. This is typically an end-user (a natural person) but it can also be a corporation or other legal person.
- **Resource set:** one or more protected resources that the resource server manages as a set, abstractly. In authorization policy terminology, a resource set is the "object" being protected.
- **Requesting party**: an end-user, or a corporation or other legal person that uses a client to seek access to a protected resource. The requesting party may or may not be the same party as the resource owner.
- **Client**: An application making protected resource requests with the resource owner's authorization and on the requesting party's behalf.
- **Authorization server**: centralized authorization server that governs access based on resource owner policies. Resource owners configure authorization servers with access policies that serve as asynchronous authorization grants.

The UMA protocol has three phases, as shown in Figure 3:

1. Protect a resource(s): the resource owner, who manages online resources at the resource server, introduces it to the authorization server for protection. To accomplish this, the authorization server presents a protection API to the resource server. This API is protected by OAuth (or an OAuth-based authentication protocol) and requires a protection API token (PAT) for access. Out of band, the resource owner configures the authorization server with policies associated with the resource sets.
2. Get Authorization: the client asks the resource server for access to an UMA-protected resource. In order to access, the client must use the authorization server's authorization API to obtain authorization data and a requesting party token (RPT) on behalf of its requesting party, and the requesting party may need to supply identity claims. The API is protected by OAuth (or an OAuth-based authentication protocol) and requires an authorization API token (AAT).
3. Access a resource(s): The client successfully presents to the resource server an RPT that has sufficient authorization data associated with it.

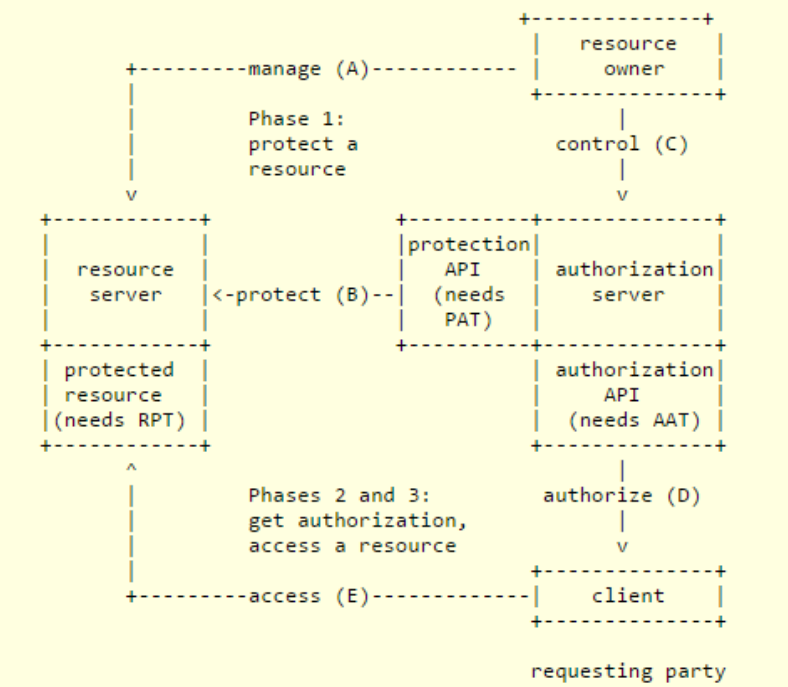

Figure 3: UMA abstract protocol flow [8]

UMA provides the following important Authorization capabilities that are highly relevant for IoT technologies as detailed in Table 6

Table 6: User-Managed Access (UMA) analysis

+	Provides dedicated access relationship service that allows resource owners to control access to their services/resources/etc. that are residing on different domains, in one central place (no need for expensive and error prone access policy management throughout different domains).
+	UMA allows resource owners to configure their own policies that are required for appropriate access control to their resources without any restrictions (i.e. starting with Access Control Lists, simple OAuth 2.0 scopes and finishing with complex Authorization policies that will be driven by compound access policy engines).
+	Allow resource owners to provide claims based Authorization mechanisms for more fine grained access control decisions to the resources and services that they provide on the web.
+	UMA does not define specific format of claims that need to be specified during Authorization decision which gives a flexibility to use various security claim based technologies
+	Provides automatic access policy enforcement mechanisms that will allow Authorization Manager to ask a requesting party without a presence of a resource owner (request will be based on specified resource owner access policy)
+	Full support for OpenID Connect (authentication) and OAuth 2.0 (Authorization) that are both modern and well adopted by industry protocols (in fact some of the core parts of UMA were build based on OAuth 2.0 and OpenID Connect is an identity protocol build on top of OAuth 2.0).

-	So far not widely adopted, but the situation is rapidly changing (many companies, universities, health organizations, security software suites are currently in a process of adopting it)

2.2.3 Authorization technology comparison

Based on reviewing the latest industry security technologies for Authorization – both OAuth 2.0 and UMA have their great benefits that should suit many web and IoT frameworks. Making a direct comparison is difficult as they share an underlying technology. However, in Table 7 we provide an analysis of how well suited the approaches are for handling security use cases relevant to IoT e.g. support for user-driven protection of resources, privacy, flexibility, etc.

The analysis indicates that UMA introduces some additional complexity beyond the simple OAuth protocol. However, there are numerous benefits in terms of user control, fine-grained Authorization and flexibility in deployment (to heterogeneous configurations that OAuth may be difficult to apply to).

Table 7: OAuth 2.0 and UMA comparison

	OAuth 2.0	User-Managed Access (UMA)
User-driven policies	Simple policies (based on scope)	Possibility to specify own policies that are required for appropriate access control to their resources without any restrictions
Support for claims-based access control	No (scopes only)	Yes (full support)
Flexible claims format approach	No (scopes only)	Yes (for any claims format)
Automatic access policy enforcement mechanisms	No	Yes (UMA can ask requesting party for additional claims in order to grant access to the resources)
Support for modern Authentication and Authorization protocols	No (OAuth 2.0 is an authorization framework)	Yes (fully support for OpenID Connect)
Privacy	Some (OAuth 2.0 is a delegation protocol that simply limit access to the resources)	Yes (UMA provides ways to specify and control fine grained access to the resources hence provides better privacy)
Resource Server (RS)	Not stated	RS interface is fully defined (allows more flexible

		configuration and deployment)
Authorization Manager (AM)	Not stated	AM interface is fully defined (allows more flexible configuration and deployment)

2.3 Other Security Solutions: FIRE and IoT Projects

The FIESTA-IoT project crosses the domain of experimental Internet testbeds (as traditionally found in FIRE-Future Internet Research and Experimentation projects[1]), and IoT facilities. Hence, we now examine the security solutions applied in other FP7 and H2020 projects to protect resources across these two domains.

2.3.1 Fed4FIRE

Fed4FIRE[2] is a general framework to federate experimental testbeds across a broad range of Internet Testbeds, i.e., cloud computing, Wireless radio, Software Defined networking, etc. The emphasis is on the controlling of access to shared virtualised resources e.g. a Virtual Machine on a compute or network host, rather than data being provided by an IoT testbed. In terms of security, FedFIRE employs the following technologies:

Authentication

- Username and password for login to Fed4FIRE portal
- X.509v3 technology is used in various Fed4FIRE tools (i.e. jFed, OMF6 etc.)

Authorization

- Policy Decision Point (PDP) provides federated Authorization using SFA Slice Credentials
- Fine grained Authorization using Attribute-Based Access Control (ABAC) model
- Variety of proprietary Authorization software on testbed sides

Table 8 provides an analysis of the technology choices of Fed4FIRE. While Fed4FIRE has created a mature and secure platform based upon traditional X.509 authentication certificates, and a project developed PDP/PEP solution for experimental testbeds—these technologies are not well suited to a data-driven facility built upon Web interfaces and protocols (which are central to the FIESTA-IoT testbeds and semantic data services).

Table 8: Fed4FIRE FP7 project security analysis

+	X.509v3 is a standards-based technology (X.509v3 certificates as identity provider and Single Sign-On)
+	Single credential authentication for cross FIRE testbeds

[1] https://www.ict-fire.eu/

[2] http://www.fed4fire.eu

+	Single credential for authorized access on cross FIRE resource controllers
+	Policy Decision Point provides federated Authorization using SFA Slice Credentials
+	Fine grained Authorization using ABAC model
+	Allows to choice of trust provider
-	Heavyweight technologies to configure and utilise; today's web-based approaches offer more flexible lightweight solutions

2.3.2 OpenIoT

OpenIoT[1] manages cloud environments for IoT resources (such as sensors, actuators and smart devices) and offers a set of utility-based (i.e. pay-as-you-go) IoT services. This enables the concept of "Sensing-as-a-Service". Hence, this is similar in concept to FIESTA-IoT's collection and annotation of data from multiple experimental IoT testbeds. In terms of security solutions, OpenIoT employs the following:

Authentication

- Uses Centralised Authentication Server (CAS) **[11]** for centralised authentication between services (username / password approach)

Authorization

- Based on OAuth tokens – short lived tokens are issued after successful authentication with CAS.
- Very simple (i.e. basic) Authorization policy rules based on lattice-based access control model **[9]** that does not provide fine grained access control to resources/services.

Table 9: OpenIoT FP7 project security analysis

+	The use of CAS APIs simplified development. These are well established APIs with mature documentation and community usage.
+	Centralised Authentication. Simplified the complexity of managing authentication.
+	Authorization using OAuth 2.0 tokens (implemented in addition to CAS)
-	Basic username / password authentication; this is limited in comparison to new standards such as OpenID Connect.
-	No fine grain Authorization. Using standard capability of OAuth 2.0 i.e. scopes etc. (for example to specify that a resource owner would like his/her resource to be used between 18:00 and 19:00 during weekdays would be impossible!)

2.3.3 IoT-A

IoT-A is an architectural reference model **[10]** and the definition of an initial set of key building blocks that are envisioned as the foundations for fostering the emerging Internet of Things. Concrete IoT systems can be developed following the proposed principles in order to simply the task, and promote interoperability. In terms of security building blocks, IoT-A proposes:

[1] http://www.openiot.eu/

Authentication

- Proposes to use SAML for SSO

Authorization

- Proposed to use the eXtensible Access Control Markup Language (XACML) policy engine based on Attribute-Based Access Control. **XACML**[1] is an OASIS standard that describes both a policy language and an access control decision request/response language (both written in XML). The policy language is used to describe general access control requirements, and has standard extension points for defining new functions, data types, combining logic, etc. The request/response language lets you form a query to ask whether or not a given action should be allowed, and interpret the result. The response always includes an answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (the request cannot be answered by this service).

Table 10: IoT-A FP7 project security analysis

+	SAML is widely adopted in SSO systems
+	XACML gives possibility to specify fine grained access control policies (the only widely recognised standard that defines declarative access control policy language)
-	SAML is not really suitable for mobile devices (and most likely sensor devices) without additions effort to provide broker services etc. (was primary designed for Web-based applications)
-	XACML does not solve access policy and privacy issues on its own – need defined mechanisms for Resource Owners to manage access control to their resources

2.3.4 Authentication and Authorization technology comparison of FIESTA-IoT related FP7 projects

Table 11 and Table 12 provide a side to side comparison of the authentication and Authorization technologies of reviewed above FIESTA-IoT related FR7 projects.

[1] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

Table 11: Authentication review summary of FIESTA-IoT related FP7 projects

	OpenIoT	IoT-A	Fed4FIRE
Authentication	**Pros**: • Central Authorization service provides an easy developer API • Centralised solution for federated SSO • Widely adopted • Support for OpenID Connect and OAuth 2.0 (SAML support was native) in new CAS versions **Cons**: • Username and password authentication • CAS server does not provide any notion for Authorization (additional solution(s) for Authorization is/are needed)	**Pros**: • Information about identity of entity is provided using SAML tokens (SAML tokens are also used for SSO) • SAML is widely used in SSO systems **Cons**: • SAML is mainly designed for Web-based applications • SAML is not really suitable for mobile devices without broker services etc. (i.e. extra effort/overhead is required)	**Pros**: • X.509v3 is a standards-based technology (X.509v3 certificates as identity provider and Single Sign-On) • Single credential authentication for cross FIRE testbeds • Single credential for authorized access on cross FIRE resource controllers • Allows to choice of trust provider **Cons**: • Can be expensive to setup up and maintain

Table 12: Authorization review summary of FIESTA-IoT related FP7 projects

	OpenIoT	IoT-A	Fed4FIRE
Authorization	**Pros**: • OAuth 2.0 is well defined modern open standard for Authorization (both Web and mobile devices) • Provides secure and robust access control delegation mechanisms using short lived API tokens **Cons**: • No fine grained Authorization capability (instead OAuth 2.0 scopes etc.)	**Pros**: • XACML is de facto standard for authorization • Consistent access control • Fine-grained and risk-aware **Cons**: • XACML does not solve access policy and privacy issues on its own (need defined mechanisms for Resource Owners to manage access control to their resources)	**Pros**: • Policy Decision Point provides federated Authorization using SFA Slice Credentials • Fine grained Authorization using ABAC model **Cons**: • Depends on OMF 6.0 – not widely applied. • SFA is not a global standard • Variety of proprietary testbed Authorization software is in place.

2.4 Security Implementation Technologies

There are several technologies that provide security solutions “out of the box”. It is important from various perspectives (resources, time etc.) to reuse well defined, proven and well tested security technologies that are available nowadays and not “reinvent the wheel”, especially during implementation and deployment of security solutions. This is because, many software errors and bugs will have been eliminated by community evaluation of the software. Furthermore, the use of Open Source solutions is equally important, as the security community can verify the implementation to assess its security.

2.4.1 Central Authentication Service (CAS)

The “Central Authentication Service” (CAS) **[11]** defined one of the first Web SSO protocols. It’s has a simple to use API, and is supported by several open CMS platforms. Backed by LDAP, it was a good choice for many organizations to centralize username / password authentication. It also allows access control based on network address, to restrict which servers can use the enterprise web authentication service. With the availability of newer, more functional authentication standards, like SAML and OpenID Connect, new applications should be directed away from CAS. An analysis of the CAS is given in Table 13.

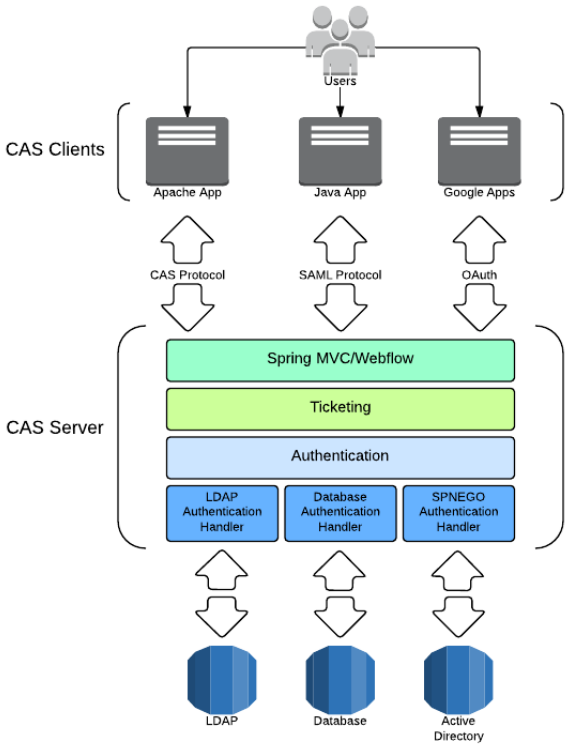

Figure 4: CAS (4.0.x) architecture[1]

[1] http://jasig.github.io/cas/4.0.x/planning/Architecture.html

Table 13: Analysis of CAS

+	Mature developer API (as a result faster development and less prone to errors)
+	Centralised solution for federated SSO
+	Widely adopted, mainly because it provides centralised authentication mechanism
+	New versions of CAS provide support for OpenID Connect and OAuth 2.0 (SAML support was native)
-	CAS server does not provide any concept of Authorization therefore other technologies in line with CAS need to be implemented (additional and potentially very expensive overhead)!

2.4.2 OpenAM

OpenAM **[12]** is an open source access management, entitlements and federation server platform. OpenAM provides core identity services to simplify the implementation of transparent SSO as a security component in a network infrastructure. The software stack is illustrated in Figure 5; it can be seen that OpenAM provides implementations of multiple security technologies e.g., OAuth, OpenID Connect and XACML. However, the most relevant technologies provided by OpenAM are the security agents to protect resources (top left in Figure 5).

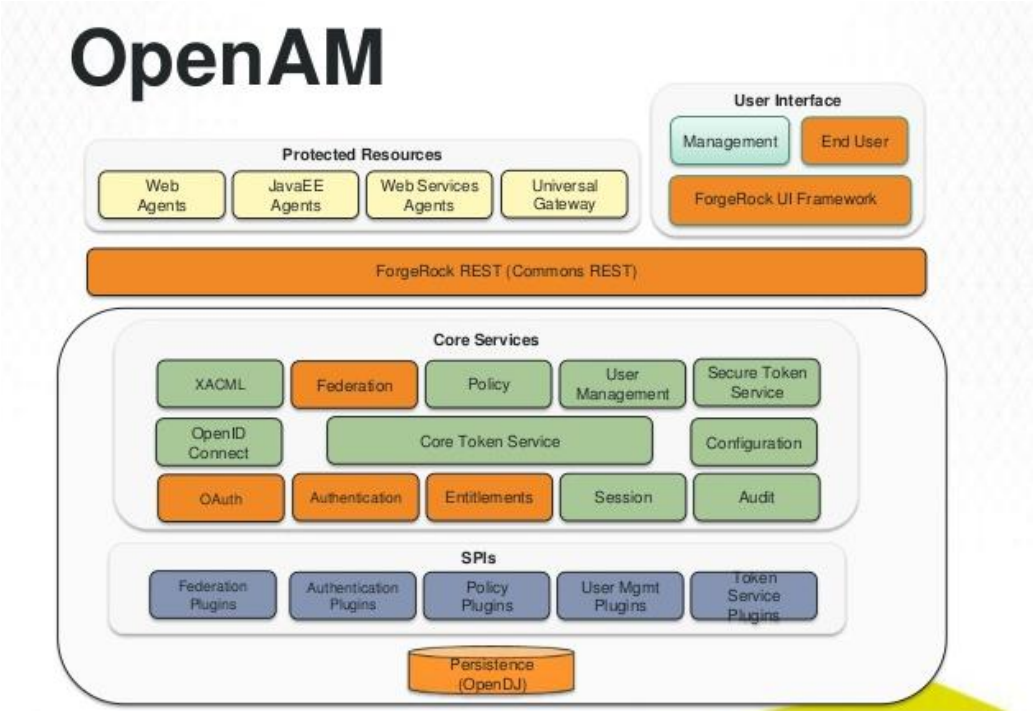

Figure 5: OpenAM server stack[1]

[1] http://www.slideshare.net/ForgeRock/ois-architecture-review

Policy Agents. A web policy agent is a software library installed at the resource server and configured to intercept all client requests to access resources. The interception sequence is as follows:

1. A client requests access to a protected resource
2. The policy agents intercepts all requests to a protected resource API. Hence, the policy agent acts as Policy Enforcement Point (PEP).
3. The policy agent communicates with OpenAM in order to enforce a policy decision based on the request from a client. A very simple policy might enforce users to be authenticated. Hence, OpenAM acts as the Policy Decision Point (PDP).
4. OpenAM enforces the policy based on set of claims that are provided by a client (if applicable) and returns an authorization result back to the policy agent
5. Based on the returned authorization result from OpenAM the policy agent grants or denies access to the protected resource

Policy agents are available for multiple web server platforms such as Apache, Tomcat, Microsoft IIS, Sun Web server, Glassfish, JBoss and others. *The downside of web policy agents are that it is necessary to provide a specific configuration for a resource server in order to install a web policy agent; this is a complicated task that requires modification of existing server configurations.*

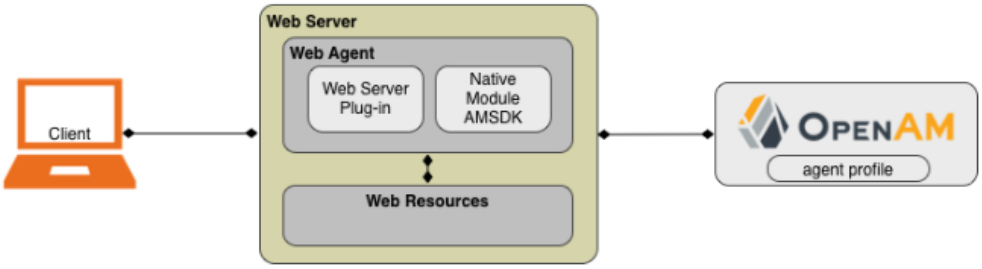

Figure 6: OpenAM Policy Agent architecture[1]

Open Identity Gateway (OpenIG) offers an alternative and transparent solution compared to policy agents to allow legacy infrastructure to be easily integrated with OpenAM. Open Identity Gateway (OpenIG) works as an HTTP gateway, also known as a reverse proxy. OpenIG is usually deployed on a network so that it intercepts both client requests and also server responses (as illustrated in Figure 7). When a client requests a protected resource it is intercepted by the Open Identify Gateway (OpenIG) - this means that there is be no need to configure either servers or clients; it is handled transparently. The only configuration needed is of OpenIG in order to add new capabilities to existing services, and to ensure it communicates between the parties.

An analysis of OpenAM is provided in Table 14

[1] http://openam.forgerock.org/doc/bootstrap/web-users-guide/images/thumb_web-policy-agent.png

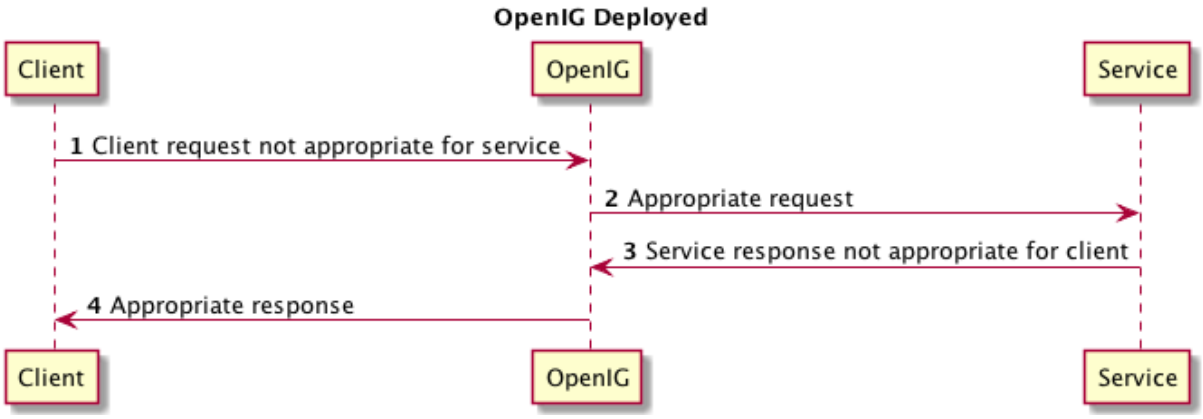

Figure 7: deployed Open Identity Gateway in existing infrastructure[1]

Table 14: Analysis of OpenAM

+	"All-in-one" open source access management solution
+	Provides single and federated Single Sign-On (SSO) solutions out of the box (username/password, SAML and OpenID Connect)
+	Easily integrates into existing systems
+	Allows you to build a federated identity and entitlement solution easily
+	Entitlements engine that supports the latest XACML protocol
+	Full support for OAuth 2.0 and OpenID Connect
+	Support for User-Managed Access (UMA). In the latest release of OpenAM 13.5 (2016), UMA is available and supported.
+	Variety of Web/J2EE Agent software that enforces Authorization policy decisions for OpenAM on protected resources (sometimes maybe not suitable for existing infrastructure configurations)
+	Gateway/reverse proxy complimentary software (OpenIG) that allows to protect resources without modifying existing server configurations
+	Easy configurable using GUI and CMD tools
-	Combining with CDDL 1.0 license may rise issues (if the software needs to be extended).

2.4.3 Gluu Server

The Gluu Server **[13]** is a direct competitor of OpenAM; it offers a free open source software stack that enables an organization to deliver a self-hosted, standards-based authentication and authorization service. As illustrated in Figure 8, Gluu provides multiple authentication and authorization technologies e.g. OpenID Connect, SAML, OAuth, etc. Gluu, like OpenAM provides agents to protect resources. However, like OpenAM these require reconfiguration on the server infrastructure, which is again a

[1] https://backstage.forgerock.com/static/docs/openig/3.1.0/gateway-guide/images/gateway-deployed.png

complicated task. Unlike OpenAM, Gluu does not provide a reverse proxy solution to counter this concern.

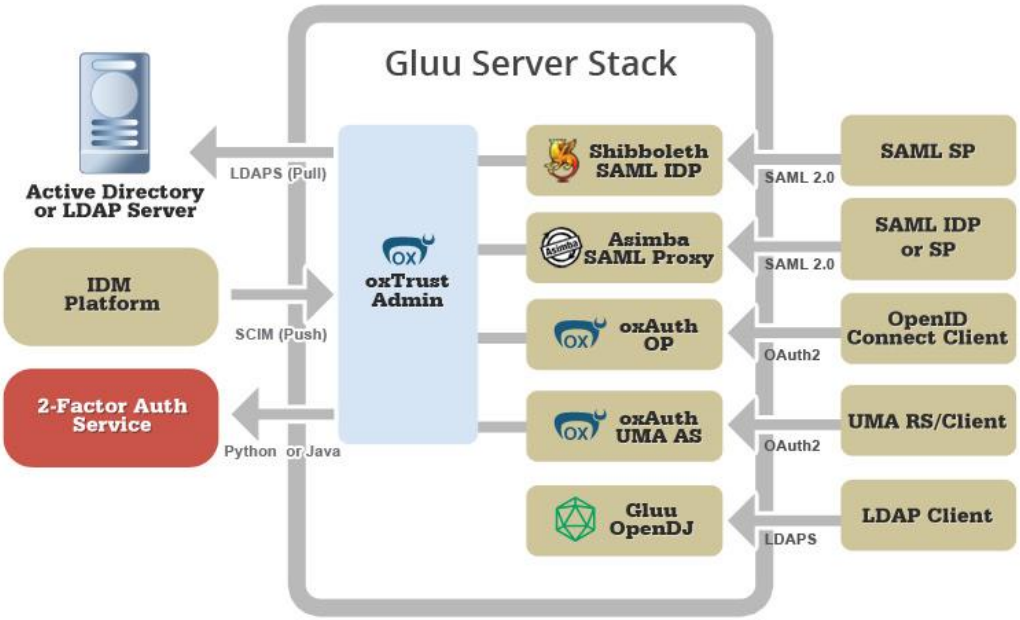

Figure 8: Gluu server stack[1]

An analysis of Gluu is provided in Table 15.

Table 15: Analysis of Gluu

+	A simba SAML Proxy enables an organization to consolidate inbound SAML authentication from the IDPs of partners to a website or an application
+	Provides single and federated SSO solutions out of the box
+	Allows you to build a federated identity and entitlement solution easily
+	Entitlements engine that supports the latest XACML protocol
+	Full support for Oauth 2.0 and OpenID Connect
+	Full support for User-Managed Access (UMA) (currently resources can be protected using Apache 2.0 containers)
+	Web Agent software that enforces Authorization policy decisions for OpenAM on protected resources (not a very wide choice)
+	Easy configurable using GUI and CMD tools
-	Lack of some administration, development and tutorial documentation that makes training/integration more complicated, time consuming and more likely more error prone
-	Currently no solutions for gateways/reverse proxies that will allow to protect resources without modifying existing server configurations (editing existing server configuration is not the best option!)

[1] http://www.gluu.org/open-source/open-source-vs-on-demand/

2.4.4 Comparison of Authentication and Authorization Technology Suites

Based on the reviewed technologies it is important to note that both OpenAM and Gluu server provide support for UMA therefore both technologies were considered for further investigation and comparison (see Table 16). Based on the key points in the comparison table (highlighted in bold) it can be seen that OpenAM is a more beneficial technology to choose and will be considered as the technology choice.

Table 16: CAS, OpenAM and Gluu comparison

	CAS	OpenAM	Gluu
Open Standards	Yes	Yes	Yes
Single Sign-On (SSO)	Yes	Yes	Yes
Support for modern Authentication mechanisms (e.g. SAML, OpenID Connect etc.)	Yes	Yes	Yes
Support for various Authorization mechanisms (e.g. custom user policies, XACML etc.)	Not applicable	Yes	Yes
User-Managed Access (UMA) support	Not applicable	Yes (currently under development/integration)	Yes (currently full support)
Variety of Web Policy Agents to serve as a container for protected resources	Not applicable	**Yes (various)**	Yes (some)
Ready and easy configurable Gateways/Reverse Proxy solutions to serve as a container for protected resources	Not applicable	**Yes (well-defined and mature OpenIG software)**	Yes (some)
Admin and development documentation	Yes (not applicable for AuthZ)	**Yes (detailed)**	Yes (limited in some cases)
License	Open source (Apache 2.0[1])	Open source (CDDL-1.0. Common Development and Distribution License (CDDL)[2])	Open source (MIT[3])

[1] http://opensource.org/licenses/Apache-2.0

[2] http://opensource.org/licenses/CDDL-1.0

[3] http://opensource.org/licenses/MIT

2.5 Proposed Security Technologies for FIESTA-IoT

Based upon the analysis provided in this section, the following technology decisions were made to implement the security requirements listed in the introduction to this document:

- Open ID Connect is chosen as the authentication solution.
- UMA is chosen as the Authorization solution.
- OpenAM is chosen as the security implementation suite.

The following sections in turn highlight the benefits afforded by each of these three decisions.

2.5.1 Benefits of using OpenID Connect for Authentication and Single-Sign On

Analysis of FIESTA-IoT related FP7 projects showed that projects deployed and used various Authentication and Single-Sign On technologies:

- Fed4FIRE uses username/passwords to login users to the main portal, as well as X.509v3 certificates for SSO to experimenter tools such as jFed and OMF. The username/password approach is a standard practice to provide login services to users and can potentially be used in FIESTA-IoT (for example to provide login services to FIESTA-IoT portal). X.509v3 based technology is more heavyweight, can be expensive to setup up and maintain, and hence potentially not well suited to IoT devices. In contrast OpenID Connect can substitute username/password approach (if necessary), it's lightweight, easy to use and maintain, as well as suitable for mobile and possible IoT devices (communication data is based on lightweight JWT tokens).

- OpenIoT used username/password approach to login users to the main portal and kept session tokens that allowed users to SSO to various OpenIoT services etc. Using username/password requires more work to provide a SSO to different domains; but with OpenID Connect it is quite easy -- you simply specify what identity providers you trust – if 2 domains trust the same identity provider then they can automatically allow each other to use their services.

- IoT-A proposed to use SAML 2.0 tokens to authenticate users and SSO. SAML 2.0 is mainly designed for Web-based applications and can potentially be not suitable with growing IoT device industry (many IoT devices are migrating to OAuth 2.0 based authentication technologies). It is possible to use SAML 2.0 in IoT, but extra effort/overhead will be required.

Table 17 highlights the benefits of using OpenID Connect as the core authentication and SSO mechanism for FIESTA-IoT.

Table 17: OpenID Connect benefits for FIESTA-IoT

Technology/Feature	Benefits
Build on top of OAuth 2.0 standard	Based on an open standard. Well-known industry and education organisations already deploy OAuth 2.0. FIESTA-IoT can quickly integrate testbeds and

	applications that employ the standard.
Lightweight, hence suitable to mobile devices (and possible to sensor devices)	FIESTA-IoT is built around the integration of sensor and data technologies. Existing technologies in this domain are already switching to OpenID connect.
Simpler protocol to realise (great support for open libraries as well)	Greatly simplifies development and integration processes hence save effort, time and resources.
Potentially safer (XML-DSIG changed to JWS what eliminates a whole range of possible attacks)	OpenID Connect is proven to be potentially more secure than signed XML documents therefore providing direct benefit to build/use more security services/communications in FIESTA-IoT
Become widely adopted standard by large organisations such as Google, Microsoft etc.	Improved user experience (SSO using a Google account for example), as well as potentially bring users from multiple sites (i.e. Google, Microsoft etc. users using their credentials to access FIESTA-IoT services).
Foundation for a far more efficient and scalable **enterprise federated single sign-on solution and allows to choice of trust provider**	OpenID Connect allows configuring and using multiple identity providers with a minimal effort. An example would be allowing users to login, authorise and use FIESTA-IoT services using Google and Microsoft accounts (here we will trust 2 identity providers). Ease the increase of more users to FIESTA-IoT.

2.5.2 Benefits of using UMA for Authorization and Access Control

Analysis of FIESTA-IoT related FP7 projects showed that projects deployed and used various Authorization and Access Control technologies/mechanisms:

- Fed4FIRE supports testbeds via an OMF 6.0 based Policy Decision Point (PDP), as well as permitted testbed providers to use their own Authorization and access control mechanisms that are perfectly suitable. The restrictions here are that: the project consisted of wide range of propriety Authorization and access control mechanisms that can be hard to maintain if something changes in the future, as it can potentially require maintenance and development work and as a result it will cause delays/expenses.
- OpenIoT used modern OAuth 2.0 tokens for Authorization and access control. The only downside of using purely OAuth 2.0 is – it does not provide fine-grained Authorization and access control to protected resources since it is based on scopes only. In contrast UMA is based on OAuth 2.0, which allows users to protect resources using fine-grained Authorization mechanisms based on a flexible claims format.
- IoT-A proposed to use XACML for fine-grained Authorization and access control. Even though some experts claim that XACML is hard to use and maintain it is currently the only Authorization standard and its perfectly suitable for fine grained Authorization. Unfortunately XACML does not solve access policy and privacy issues on its own (we need defined mechanisms for Resource Owners to manage access control to their resources etc.). UMA provides exactly that – it allows resource owners to manage wide range resources on any domains, specify and control XACML (if needed)

policies/rules, as well as help them to maintain privacy and access control issued by providing all security services (i.e. audit, access control etc.) in a centralised manner.

The UMA protocol **[8]** is a new technology (its draft was approved by the IETF early in 2015), is believed to be the future Authorization control protocol. Table 18 provides the benefits that UMA can provide for FIESTA-IoT—this illustrates that it goes beyond the previously described authorization and access control technologies and is best suited to the security requirements of FIESTA-IoT.

Table 18: UMA benefits for FIESTA-IoT

Technology/Feature	Benefits
Provides a dedicated access relationship service that allows resource owners to control access to their services/resources/etc. that are residing on different domains, in one central place (no need for expensive and error prone access policy management throughout different domains)	This eliminates a resource owner to control access to his/her resources by managing different sites that quite often is expensive, complex, time consuming, cumbersome as well as quite importantly prone to errors. FIESTA-IoT will greatly benefit here, i.e. it will allow different stakeholders to apply Authorization decisions, e.g. testbed owners, knowledge providers, users (of their submitted data).
Allows resource owners to provide claims based Authorization mechanisms for more fine grained access control decisions to the resources and services that they provide on the web	Since we are trying to achieve fine-grained access control to protected resources in FIESTA-IoT. UMA allows a resource owner to provide claims based Authorization mechanisms for more fine-grained access control decisions. It will allow FIESTA-IoT administrators to create, configure and deploy any suitable access control policies that will protect FIESTA-IoT services (i.e. portal etc.), as well as will give freedom to specify control policies for FIESTA-IoT testbed administrators (e.g. maybe they will decide to re-use existing policies which UMA will allow without any problem).
UMA does not define specific format of claims that need to be specified during Authorization decision which gives a flexibility to use various security claim based technologies	Claims format is quite important during Authorization flow (it might affect Authorization sequence flow, enforce deployment of various policy engines based on used claims etc.), therefore FIESTA-IoT will greatly benefit by allowing to use flexible claims format during Authorization control flow (single format or a combination of them) and therefore will allow to find better technology/technique to (re)use. It is especially important for FIESTA-IoT testbeds since they might need specific access control decisions/flows based on their heterogeneous resources and therefore most likely will require to use flexible/various access control claim formats.
Provides automatic access policy enforcement mechanisms that will allow an Authorization Manager to ask a requesting party without a presence of a resource owner	UMA introduces the notions of claims where a sub-protocol can ask a requesting party for additional claims in order to grant access to the resources. This will highly benefit FIESTA-IoT resource owners (i.e. testbed providers), as well as FIESTA-IoT

(request will be based on specified resource owner access policy)	administrators because Authorization Server/Manager will automatically enforce provided policies and notify requesting parties (whether it would be users trying to authorise to FIESTA-IoT or use protected FIESTA-IoT resources) that claims are missing in order to grant access to protected resources.

2.5.3 Benefits of using OpenAM for FIESTA-IoT

It is important from various perspectives (resources, time etc.) to reuse well-defined, proven and well-tested security technologies that are available (i.e. considering open source technologies only) and not "reinvent the wheel". During the analysis of security software suites (CAS, Gluu and OpenAM) OpenAM was identified as the best technology to choose.

Table 19 highlights the benefits using of OpenAM as a core security software suit for implementing the security framework of FIESTA-IoT.

Table 19: OpenAM security software suite benefits for FIESTA-IoT

Technology/Feature	Benefits
OpenAM is "all-in-one" open source identity, audit, and access control management solutions and much more	No need to implement and deploy multiple identity, audit, and access control management solutions. OpenAM provides core services as one software suite and that significantly improves integration, maintenance, configuration and sustainability.
Customizable to different environments	Customised to fit different environments which makes it easier to introduce security and solutions into existing infrastructures. This includes integration into existing testbeds as well as core FIESTA-IoT services (i.e. portal)
Provides single and federated Single Sign-On (SSO) solutions out of the box (username/password, SAML and OpenID Connect)	Allows FIESTA-IoT to ease user experience by providing SSO solutions (for example allowing users to login FIESTA-IoT using their well-known and trusted identity provider), allows to bring new users to FIESTA-IoT by integrating trusted OpenID Connect identity providers, as well as create and configure federated Single Sign-On solutions (valuable integration point with other frameworks, systems and projects)
Allows you to build a federated identity and entitlement solution easily	Building a federation can be a challenging task, but OpenAM provides all needed technologies and tools to build/adopt federated identity solutions easier and faster.
Entitlements engine that supports the latest XACML protocol	It is vital from UMA point of view to allow users to protect their resources using fine-grained policies such as XACML.
Full support for OAuth 2.0 and OpenID Connect	Full implementation of the chosen authentication solution.
Support for User-Managed Access	Implementation of the chosen Authorization solution.

(UMA) (under development)	
Gateway/reverse proxy complimentary software (OpenIG) that allows to protect resources without modifying existing server configurations	One of the great benefits of OpenIG is complete avoidance of editing protected resource host services. By using OpenIG it is possible to route all requests to protected APIs in any required way (without a need to edit any infrastructure/service configurations).
Multiple authentication mechanisms	OpenAM allows users to be authenticated by using various authentication mechanisms such as username/password, OpenID Connect and SAML 2.0. There is also a possibility to configure X.509v3 certificate authentication that can potentially allow linking Fed4FIRE users to use FIESTA-IoT platform.
Easy configurable using GUI and CMD tools	OpenAM deployment is shipped with Graphical User Interface (GUI) that is clear, easy to use and allows configuring any feature/service without much hassle.

3 FIESTA-IOT SECURITY ARCHITECTURE

3.1 Introduction

In this section, we describe how OpenAM is employed within the FIESTA-IoT platform. In this respect, there are two important dimensions to consider:

1. Protecting the FIESTA-IoT Meta-cloud services. That is, ensuring that authenticated users (e.g. experimenters, testbed owners, administrators) have secure access to the applications and services that are deployed and available via the FIESTA-IoT Web Portal. Hence, securing the usage of the EaaS services and GUIs by users.

2. Protecting access to the individual testbeds by the FIESTA-IoT services and applications.

Underpinning these are two core elements—the user roles (section 1.2.2), and the security building blocks (in the following section) that will be applied to the FIESTA-IoT architecture. In Section 5, we conclude this work by tracing this architecture implementation against the requirements and use cases documented in Section 1.

3.2 Security Framework Building Blocks

Figure 9 illustrates the software components that are used to implement and configure the security within the FIESTA-IoT architecture. These components (in green) are directly provided by the OpenAM software. The core components for Authorization and Authentication are deployed on a server as a running OpenAM instance. Each resource (i.e. REST API to resources deployed on a Web Server) is then protected behind an OpenIG component deployed on the resource server itself (acting as a PEP that communicates directly with the central OpenAM server). Hence, the OpenAM components (left) are deployed centrally in FIESTA-IoT, and the OpenIG components (right) are deployed multiply across the resource servers to be protected.

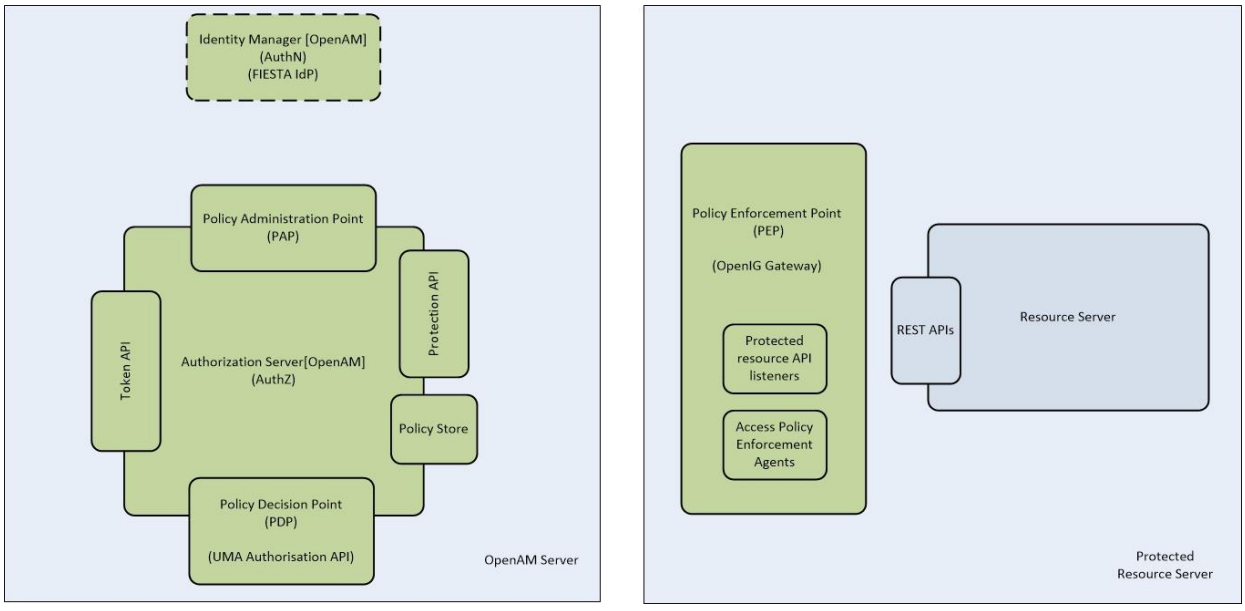

Figure 9: Security Framework Components

We now briefly summarize the functionality of these building blocks; subsequent sections will then describe how these are configured and integrated together to achieve the security requirements of FIESTA-IoT:

- **Identity Manager [OpenAM]:** This is the concrete instance of the AuthN logical component within the FIESTA-IoT architecture. The identity manager is the implementation of the functionality to register and manage user accounts and credentials. The component also deals with identifying individuals in a system and control their access to resources within that system by associating user rights and restrictions with the established identity.
- **Authorization Server [OpenAM]**: This is the concrete instance of the AuthZ logical component within the FIESTA-IoT architecture. The authorization server is the implementation of the policy decision point. Its purpose is to make decisions about whether to grant or deny access to particular client requests. OpenAM was chosen as the software component to provide this building block. Some of the core components of OpenAM are:
    - *Admin Tool (acts as Policy Administration Point)*: here policies are defined, stored and managed. OpenAM uses the configuration directory to store entitlements, whereas profiles are stored in the identity repository (user data store).
    - *Protection API*: the main API endpoint that allows resource owners to register their resources
    - *Authorization API (acts as Policy Decision Point)*: evaluates policies and issues authorization decisions, and as a policy information point, providing the information needed for authorization decisions.
    - *Token API*: the centralized location that is responsible for issuing UMA OAuth 2.0 based tokens (i.e. Protection API Tokens, Authorization API tokens etc.).
    - *Policy Store*: the centralized place where authorization policies are stored (profiles are stored in the identity repository i.e. user data store)
- **Resource server**: the server hosting the protected resources (or reference to protected resource APIs), capable of accepting and responding to protected resource requests using access tokens.
- **Gateway / reverse proxy [OpenIG]**: works as an HTTP gateway, also known as a reverse proxy. It is deployed on a network so that it intercepts both client requests and server responses. Core components of the are:
    - *Protected resource API listeners*: intercepts HTTP requests to/from protected resources, calls the authorization manager for authentication and authorization decisions etc. It can be thought of as the middleware component that is responsible for communication between experimenters and protected resources.
    - *Access Policy Enforcement Agents*: called by the protected resource API listeners (described above) when a client requests access to a protected resource. Access Policy Enforcement Agents communicate with OpenAM in order to enforce a policy decision based on the request from a client. OpenAM in turn enforces the policy based on set of claims that were provided by a client and return Authorization decision back to Access Policy Enforcement Agents.

3.3 Securing the FIESTA-IoT Portal and Services

This section describes how the previous building blocks are used to protect access to the various FIESTA-IoT portal services and applications—that is, the usage of UI and tools rather than accessing the data on testbeds directly (securing the testbed data resources is described in the next section). A high-level visualization of this objective is shown in Figure 10. Here, an experimenter tries to access the FIESTA-IoT portal (1). The request is intercepted by OpenIG (2) and redirected to login to FIESTA-IoT (using their previously registered credentials) via the browser. This input is passed to OpenAM (3) to authenticate the user as a FIESTA-IoT experimenter. Once authenticated, OpenAM redirects the initial request to OpenIG again (4). At this point, the request is checked against authorization policies (e.g. All authenticated FIESTA-IoT users can access the portal), i.e. OpenAM acts as the PDP to decide if the request is granted or not (5). Finally, OpenAM sends the Grant result to OpenIG that allows the request to proceed (6).

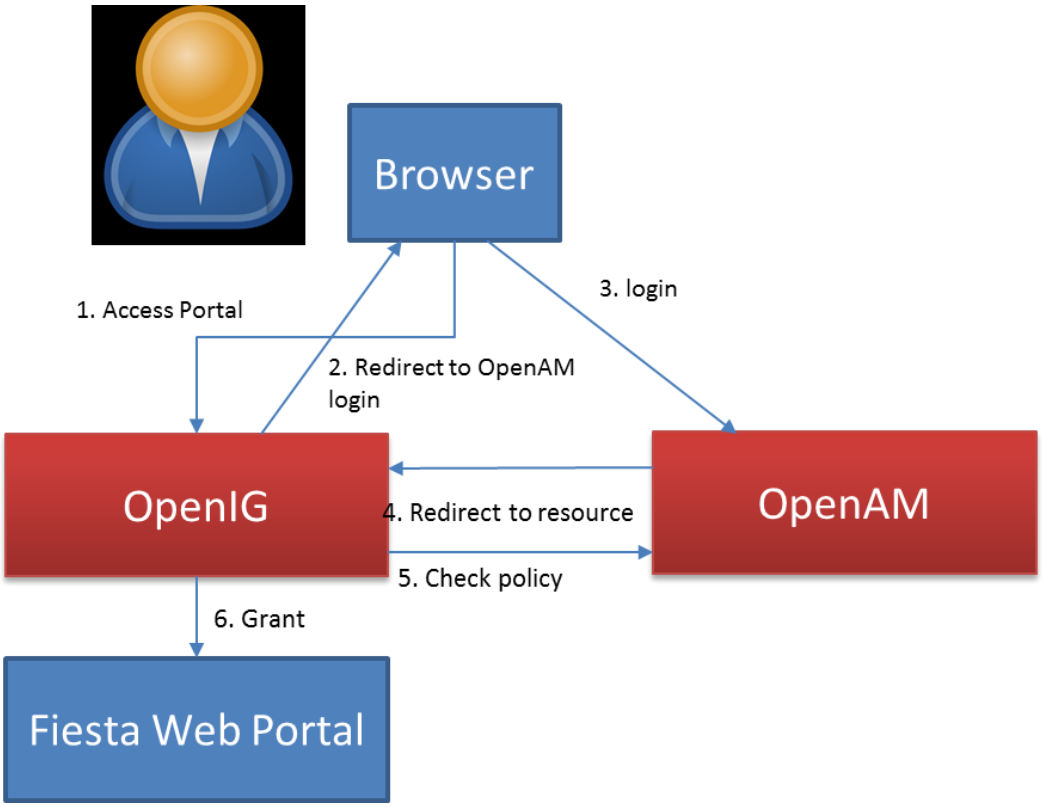

Figure 10: High-level view of protecting portal services

Detailed architecture. We now explain how this high-level procedure is carried out in detail (using the protocols and standards chosen in the previous section). This is illustrated in Figure 11 where the components interact via carrying out the numbered steps:

1. **Obtain tokens for authorization to use FIESTA-IoT portal and services**: The FIESTA-IoT Security Manager will obtain a special Protection API token that is needed to register resources (or set of resources). In this case the resources are the FIESTA-IoT portal and service APIs. A request to obtain Protection API token(s) is

made by an admin tool using a HTTP REST call to the OpenAM UMA "Token API" endpoint.

2. **Registering users/experimenters**: The FIESTA-IoT Security Manager will register users and experimenters using the admin GUI tool to interact with the Identity Manager API.
3. **Registering services** The FIESTA-IoT Security Manager will register FIESTA-IoT services (some services will need to be defined as entities since they will be making requests to protected APIs hence they must be authenticated and authorized to do so). Protected resources will be the FIESTA-IoT portal and service APIs. He/she will specify URLs of protected resources.
4. **Set policies for protected resources**: at this point the FIESTA-IoT Security Manager will be ready to specify fine grained access control policies that will be used to enforce access to protected resources (i.e. FIESTA-IoT portal and services). Different users, experimenters and services will require different access rights to protected FIESTA-IoT resources, hence access to that resources will need to be enforced by different access control policies. UMA will provide all necessary tools and options to achieve that task.
5. **Attempt to use FIESTA-IoT portal or service**: A FIESTA-IoT user, experimenter or service will attempt to use protected FIESTA-IoT portal or service by issuing a HTTP request to the protected portal or service API. HTTP request to protected API will be intercepted by OpenIG *Gateway / Reverse Proxy* configured on the Portal's web server. Since FIESTA-IoT requesting party (i.e. user, experimenter or service) will not have special UMA RPT that is needed to use protected resource *Access Policy Enforcement Agent* will redirect a requesting party to OpenAM Token API in order to get needed access request token (i.e. RPT token).
6. **Experimenter/User/Service requests an Access Token**: as was already mentioned in the previous step since a requesting party did not provide a UMA RPT token that is needed to access protected resources it will be redirected to OpenAM Token API. UMA RPT generation involves exchange of several UMA OAuth 2.0 tokens (for simplicity not covered here). It can be done explicitly by a requesting party contacting OpenAM UMA Token API before accessing protected resource (not covered here) or implicitly by a protected resource wrapper redirecting experimenters/users/services to the Token API (shown in this example).
7. **Check initial experimenter/user/service authentication**: before issuing UMA Requesting Party Token that will be used to access protected resources it is necessary to make sure that user, experimenter or service is actually authenticated as well as authorized to perform such action (usually simple/default policies will be setup by a FIESTA-IoT admin that will allow requesting parties to acquire UMA Requesting Party Tokens that will be used to access protected resources i.e. an example of simple/default policy might state that a requesting party need to be authenticated in order to access OpenAM UMA Token API). Final result of several interactions between requesting party and OpenAM UMA Token API (though Gateway / Reverse Proxy) would be a return of short lived (for security purposes) UMA Requesting Party Token (RPT) to a requesting.
8. **Check experimenter/user/service authentication/Authorization**: as soon as UMA RPT token will be returned to a requesting party Gateway/Reverse Proxy will issue an HTTP request to OpenAM UMA Authorization API endpoint where the identity of the requesting party and access rights will be checked.
9. **Check experimenter/user authentication**: the identity of a requesting party will be checked and verified at this point (need to the done before any authorization checks).

10. **Authorization Policy Check**: an authorization check will be then be performed against policies specified by a resource owner (i.e. FIESTA-IoT administrator or its authorized delegates in this case). At this stage OpenAM UMA Authorization Server might request additional claims from a requesting party in order to fulfill access control policy requirements (e.g. for example extra claims about an identity of a requesting party).
11. **Use protected resource**: if a requesting party is authorized to use a resource (i.e. passed all authentication and authorization checks) then access to a protected resource (i.e. FIESTA-IoT portal or services) will be granted.

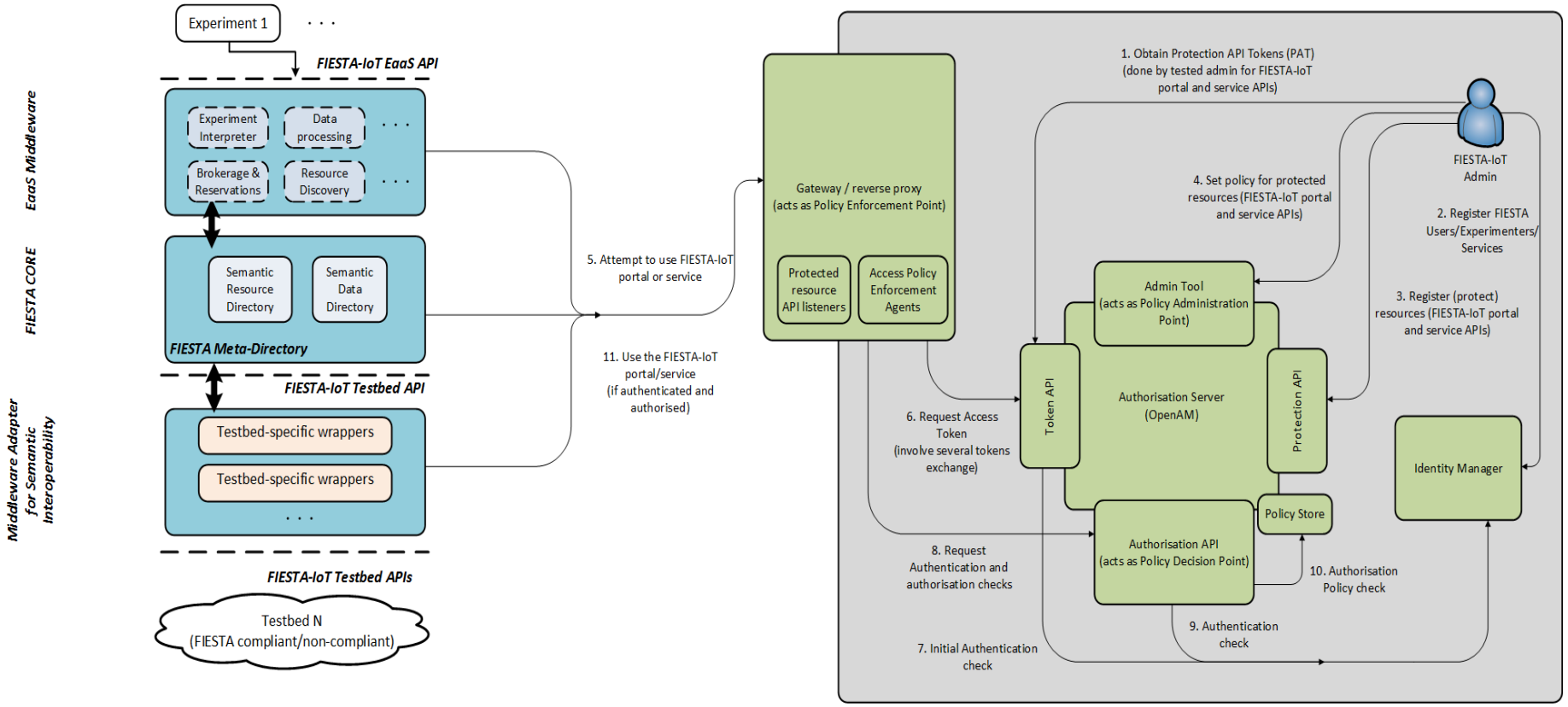

Figure 11: Securing the FIESTA-IoT Portal and Services

3.4 Securing FIESTA-IoT testbeds

To secure testbed resources, the same PEP/PDP model is applied. However, there are certain assumptions that must be made, and concessions made by the testbed in joining the FIESTA-IoT platform:

- Testbeds trust FIESTA-IoT to authenticate users.
- Testbeds trust FIESTA-IoT to enforce authorization decisions (the policies they have registered.
- Testbeds must secure the channel from the FIESTA-IoT platform technologies to their resource API. That is, if their API uses digital certificates, or API keys—it is up to the testbed to integrate the security implementation.

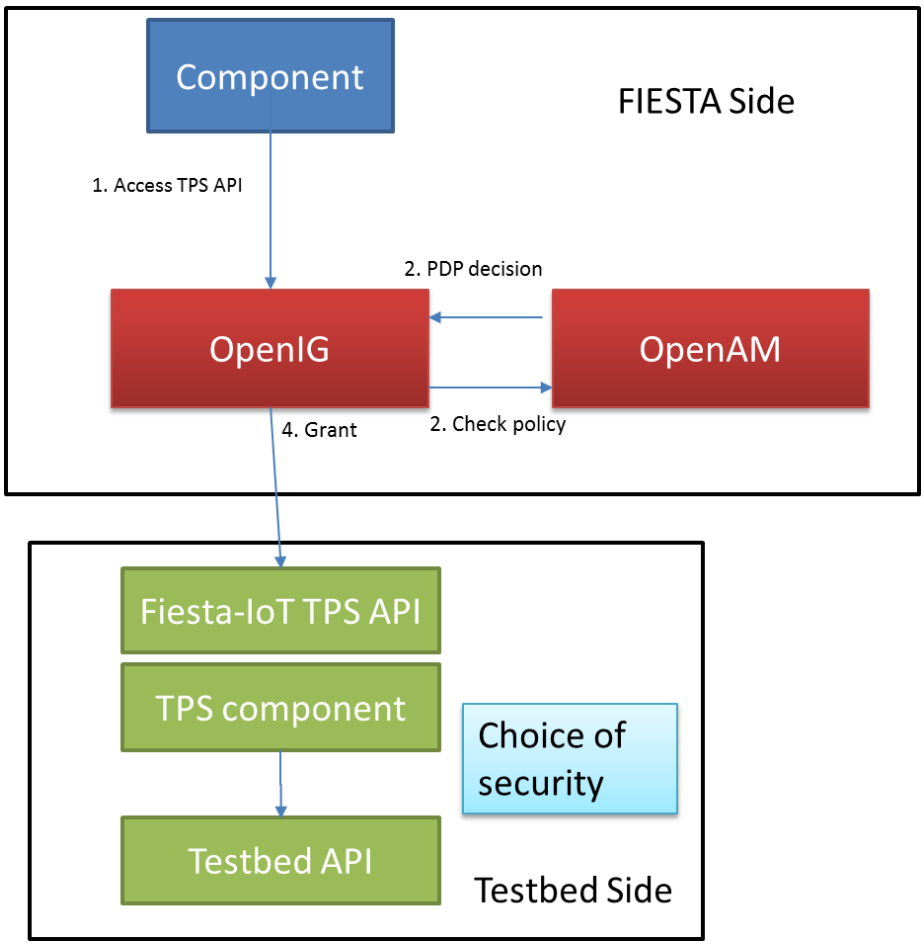

Figure 12: High-level view of FIESTA-IoT securing of testbed resources

A high-level view of the testbed security is shown in Figure 12. Here it can be seen that as per the FIESTA-IoT platform approach to connect testbeds—the testbed implements the TPS API to connect and interoperate with FIESTA-IoT. This TPS interface is the one that is protected by the authentication and authorization methods described earlier. It is then up to the testbed to secure the implementation of this component when it talks to their own testbed API.

When a request is invoked on the TPS API, the PEP (OpenIG) intercepts the request; takes the authorization token from the request (this will have been generated earlier e.g. as described in the portal protection); and checks the user has authorization to access this testbed API. If granted, the request is passed on to the TPS component implementation.

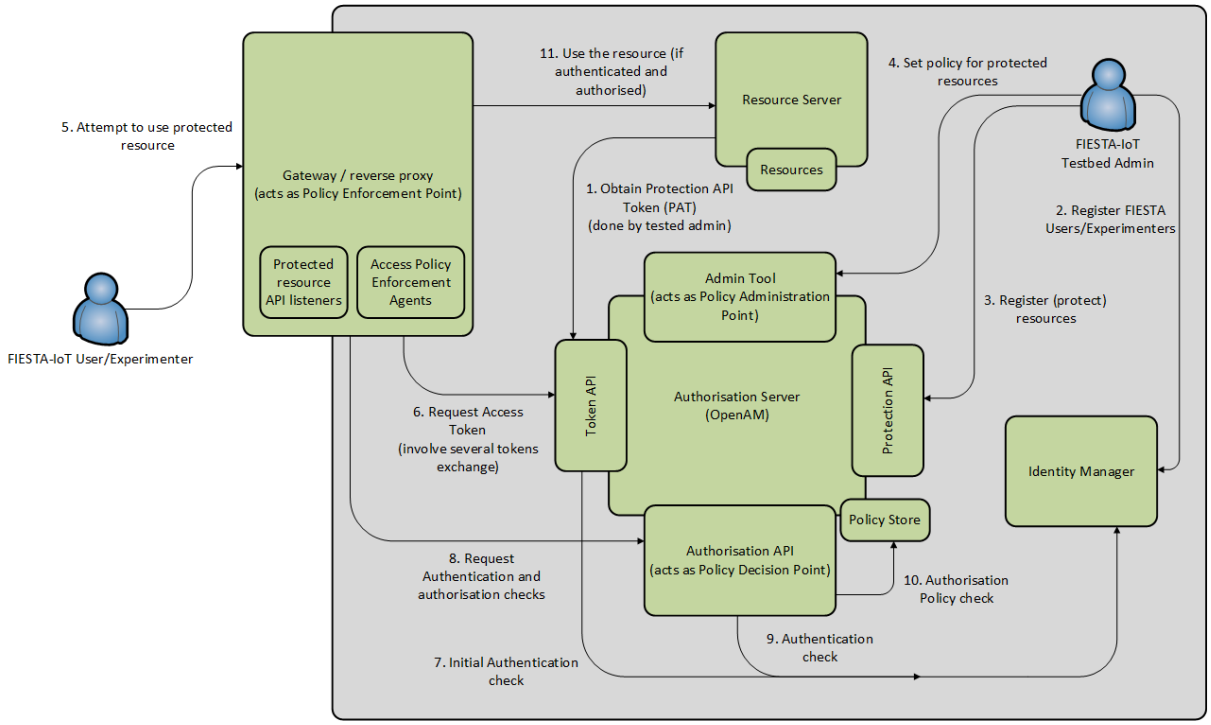

Figure 13: Securing the FIESTA-IoT testbed security architecture

Detailed architecture. We now explain how this high-level procedure is carried out in detail (using the protocols and standards chosen in the previous section). This is illustrated in Figure 13 where the components interact via carrying out the numbered steps:

1. **Obtain UMA tokens for protected resources**: first of all FIESTA-IoT testbed admin will obtain special UMA OAuth 2.0 based "Protection API token" that is needed for registering resources (or set of resources). A request to obtain Protection API token(s) will be made using HTTP REST call to OpenAM UMA "Token API" endpoint[12]. NOTE: testbed admin must be authenticated and authorized to make that call (testbed admin privilege setup will be usually done during initial OpenAM UMA setup). In this case, the API is invoked by the web-based admin tool; however, it can also be invoked programmatically using the referenced APIs in the footnotes.
2. **Registering users**: FIESTA-IoT testbed admin will register users/experimenters. Alternatively testbed admin can point them to the URL for self-registration (e.g. as an example would be allowing experimenters to register and login using OpenID Connect if they have account on trusted testbed identity server such as Google or

[1] https://forgerock.org/openam/doc/bootstrap/dev-guide/index.html#sec-rest-uma

[2] https://forgerock.org/openam/doc/bootstrap/dev-guide/index.html#rest-api-tokens

use standard username/password approach if it would be more preferable by a testbed).

3. **Register protected resources**: testbed admin will then register details of protected resources at OpenAM (using UMA resource set Protection API). He/she will specify UMA Protection API OAuth 2.0 based tokens, URL of protected resource (or set of URLs of protected resources) as well as additional information such as ID, name and even URL to a simple (i.e. default) access policy that will be usually stored at OpenAM.
4. **Set policies for protected resources**: Testbed admin will specify fine-grained access control policies that will be used to enforce access to protected resources. There are no restrictions for resource owners on the choice of access policies – it might be a simple access policy based on OAuth 2.0 scopes or a complex XACML access control policy logic.
5. **Attempt to use protected resource**: FIESTA-IoT user/experimenter will attempt to use protected FIESTA-IoT testbed resource (or a set of resources) by issuing a HTTP request. The request to protected API will be intercepted by OpenIG *Gateway / Reverse Proxy* by one of the protected resource API listeners. Since FIESTA-IoT requesting party (i.e. user or experimenter will not have special UMA "Requesting Party Token (RPT)" that is needed to use protected resource *Access Policy Enforcement Agent* will redirect a requesting party to OpenAM Token API in order to get the needed access request token (i.e. RPT token)
6. **Experimenter/User requests an Access Token**: before using UMA protected resource experimenters/users would acquire special UMA "Requesting Party Token" (RPT) from OpenAM UMA "Token API" which involves exchange of several UMA OAuth 2.0 tokens (for simplicity not covered here). It can be done explicitly by a requesting party contacting OpenAM UMA Token API before accessing protected resource (not covered here or implicitly by a protected resource wrapper redirecting users/experimenters to the OpenAM UMA Token API (show in the example in Figure 13).
7. **Check initial experimenter/user authentication**: before issuing UMA Requesting Party Token that will be used to access protected resources it is necessary to make sure that the user/experimenter is authenticated as well as authorized to perform such action (usually simple/default policies will be setup by a FIESTA-IoT testbed admin that will allow requesting parties to acquire UMA Requesting Party Tokens that will be used to access protected resources i.e. an example of simple/default policy might state that a requesting party need to be authenticated in order to access OpenAM UMA Token API). Final result of several interactions between requesting party and OpenAM UMA Token API (though Gateway / Reverse Proxy) would be the return of short lived (for security purposes) UMA Requesting Party Token (RPT) to a requesting part.
8. **Check experimenter/user authentication/Authorization**: Gateway/Reverse Proxy will issue an HTTP request to OpenAM UMA Authorization API endpoint where identity of the requesting party and access rights to a protected resource (or set of protected resources) will be checked.
9. **Check experimenter/user authentication**: identity of a requesting party will be checked and verified at this point (need to the done before any authorization checks).
10. **Authorization Policy Check**: after identity of a requesting party will be checked and verified authorization check will be performed against specified by a resource owner (i.e. FIESTA-IoT testbed administrator in this case) access control policy. At this stage OpenAM UMA Authorization Server might request additional claims from a

requesting party in order to fulfill access control policy requirements (e.g. for example extra claims about location of experimenter/user).

11. **Use protected resource**: if a requesting party will be authorized to use the resource (i.e. passed all authentication and authorization checks) then he/she will be allowed to use a protected resource (or a set of protected resources).

3.5 Example Application of the Security Framework to FIESTA-IoT Functionality

The following UML sequence diagram (see Figure 14) accentuates how steps depicted on the FIESTA-IoT architecture (Figure 13) can be mapped to the FIESTA-IoT functionality—in this case the execution of a defined experiment.

Please note that for simplicity purposes some of the steps are not shown on the UML sequence diagram (such as setup of OpenAM, User-Managed Access, Gateways, Enforcement Policy Agents etc.), but instead some are described (or briefly mentioned) during UML sequence diagram description steps. Each step of the flow diagram (marked from 1 to 16) matched the description sequence (Figure 14):

Preparation (protecting resources)

1. First of all resource owner(s) (i.e. in this case it will be FIESTA-IoT testbed administrator of their authorized delegate) registers their protected resources (as it is mentioned earlier, it is assumed that appropriate protection container *Open Identity Gateway* was installed and properly configured i.e. protected resource API listeners, Policy Enforcement agents etc.). At this state a resource owner will provision the location of OpenAM UMA Authorization Manager (i.e. simply specify URL of authorization server), UMA Protection API tokens, and UMA tokens validation URLs etc.

2. After a resource owner registers protected resources with OpenAM UMA Authorization Server he/she will define and setup fine-grained access control policies, then store them on the OpenAM UMA Authorization Server. As was already mentioned UMA does not restrict resource owners on defining access policy formats (which is very valuable feature from the usability, maintenance, management and implementation points of view) – from simple policy matrices as Access Control Lists, OAuth 2.0 scope based or fine grained complex XACML powered policies). **NOTE: this is access Policy Administration Point (PAP).**

Experiment execution

3. FIESTA-IoT experimenter/user will define an experiment using a specific set of tools (not described here) and execute it.

4. Actual experiment execution will be performed and controlled by a Client tool (i.e. FIESTA-IoT experimenter tool); or the experiment execution engine. The Client tool will issue appropriate HTTP request(s) to available experiment resources (i.e. to their protected APIs). HTTP request will be intercepted by a Web/J2EE or Gateway/Reverse Proxy (for simplicity it is not shown on the diagram) and will act as a **Policy Enforcement Point (PEP).**

5. User identity will be checked at this point. We need to make sure that a user is authenticated to the system before processing actual experiment request. If

experimenter/user will not be authenticated (i.e. was not previously logged in to the system and session ticked was not registered) then he/she will be redirected to the OpenAM for authentication.

6. OpenAM will return authentication decision back to the protected resource (i.e. to the service that is protecting it). If authentication was successful then the request execution will continue, otherwise it will stop here.

7. Resource Server that will be protected by container with Open Identity Gateway installed (mentioned in step 4 above) will register a permission request with OpenAM on behalf of a client. The permission request will typically contain the ID of a protected resource (or set of resources) that was requested, request scoped (i.e. usually scoped will be requested methods/actions), details of identity that requested access etc.

8. OpenAM will return permission ticket (usually short lived). This ticket will be needed later for acquiring OAuth 2.0 based UMA Requesting Party Token (RPT) that in turn will be used to access a protected resource (or set of resources).

9. Since a request did not contain a special UMA Requesting Party Token (RPT) that must be used to access UMA protected resources an HTTP 403 (forbidden) response will be send back to a Client. HTTP response will contain the following information:

    a. OpenAM Authorization Server details (i.e. location of the Authorization Server, how to contact it etc.) that is used to protect a resource (or a set of resources)

    b. Protected resource access permission ticket that was previously mentioned

10. Now a client should have all necessary information in order to acquire UMA Requesting Party Token (RPT) in order to finally get access to a protected resource (or set of protected resources). A Client will issue a request to Authorization Server and include the following information:

    c. UMA request permission ticket that was occurred in the previous step

    d. Special Authorization API Token (AAT) what was occurred when a Requesting Party (i.e. FIESTA-IoT experimenter) registered its Client application. AAT token is needed by Client application to access protected APIs of the Authorization Server when acquiring Requesting Party Token (RPT)

    e. Set of claims (depending on Resource Server authorization policy specification) such as claims about the identity, scopes, locality etc.

11. OpenAM UMA Authorization Server will parse a request from the previous step and check it against the Resource Owner access control policy in order to determine whether a Client provided enough information (i.e. valid permission ticket, valid Authorization API Token, valid set of claims etc.) in order for the Authorization Server to issue a Requesting Party Token (RPT). It is also possible that the Authorization Server will request additional claims from a Client in order to fulfil access request requirements (i.e. for example additional claims about location of a Client etc.). **NOTE: this is the main Policy Decision Point (PDP)**.

12. If the Authorization Server validates the permission ticket, then the Authorization Server will grant and issue a Requesting Party Token (RPT) back to a Client.

13. At this point a Client will have all necessary information to access a protected resource (or set of resources). A Client will issue an HTTP request back to a protected experiment resource (or a set of resources), but this time include the UMA Requesting Party Token (RPT) that is needed in order to access protected UMA resource.

14. The OpenIG Gateway/Reverse Proxy will intercept an HTTP request from a Client (same as was done in step 4 above), will detect a presence of a UMA Requesting Party Token and contact OpenAM UMA Authorization Server for RPT token introspection. The Authorization Server will examine the Requesting Party Token (RPT) and check its validity.

15. If Requesting Party Token (RPT) is valid the Authorization Server returns a success status code.

16. If the Authorization Server returns a status code indicating that the Requesting Party Token (RPT) was valid then the Resource Server returns HTTP 20X back to the Client (i.e. Client is authorized to use the requested resources).

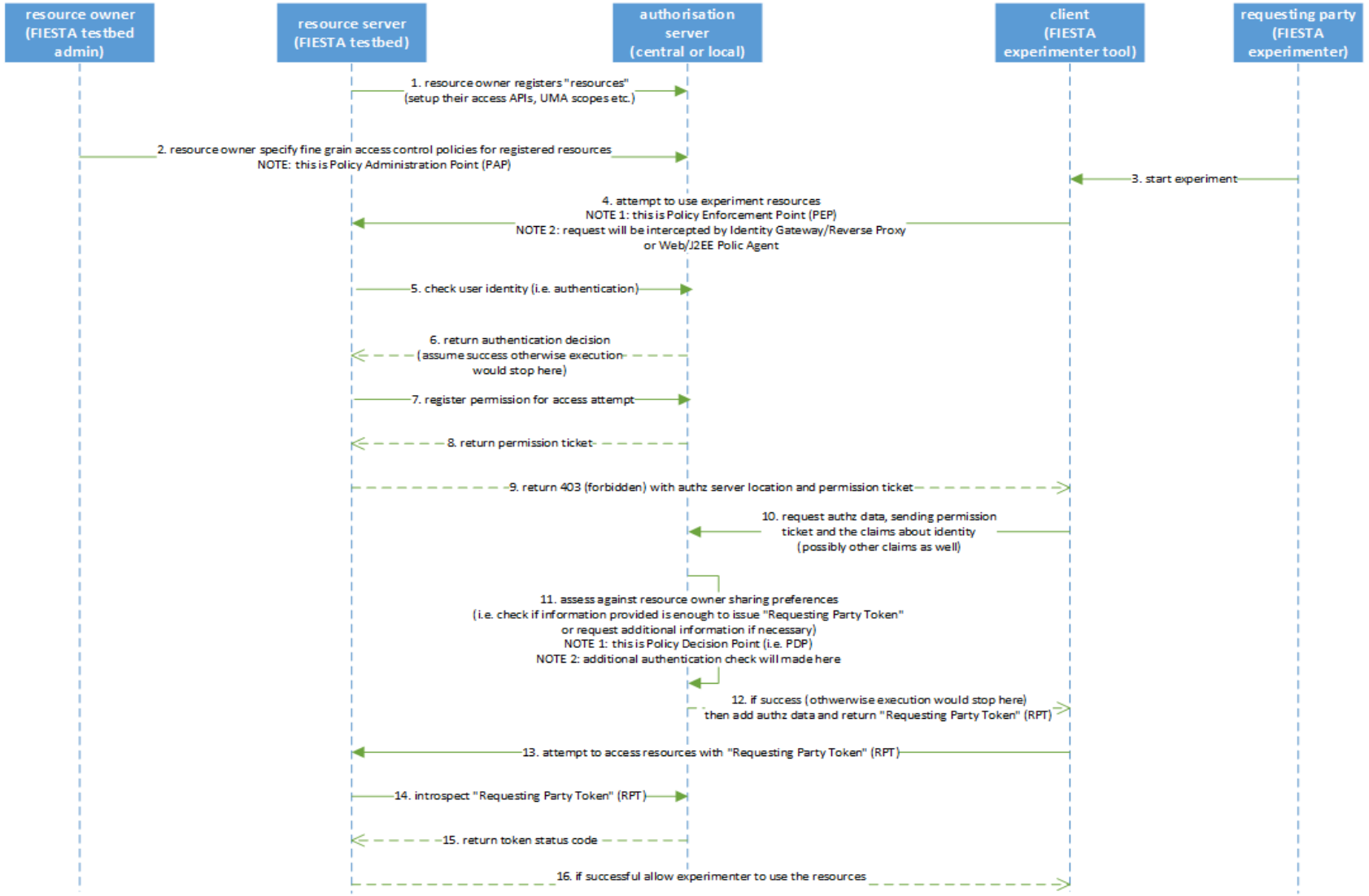

Figure 14: UMA flow diagram (FIESTA-IoT experiment lifecycle use case)

4 OPENAM/OPENIG COMPONENTS

In this section we describe in more detail how the OpenAM and OpenIG components are utilized to configure the security setup.

4.1 User Registration

The OpenAM identity manager components allow the registration and management of FIESTA-IoT users via a web browser. To illustrate this we demonstrate via the FIESTA-IoT portal, how new users can register and authenticate to use FIESTA-IoT.

When the user accesses the portal, they are presented with a login screen. There is a link to sign up to FIESTA-IoT portal. The registration screen (in Figure 15) is displayed and the user submits the listed details (this is the registration screen for the Observer role). A validation link is then sent to the user's email address. Once, they select this link their account is activated and the user can log into the FIESTA-IoT system with their provided credentials.

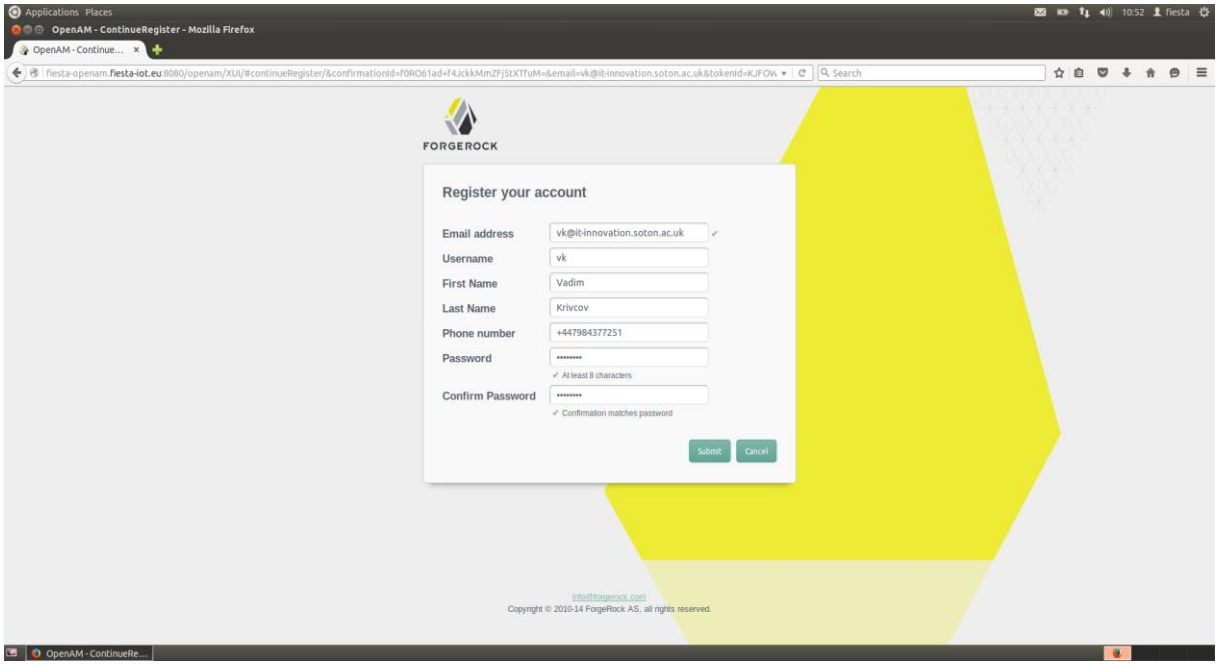

Figure 15: User Registration

4.2 Policy Creation

Policies are specified using the web policy tool available to FIESTA-IoT service/tool owners and testbed admin (i.e. FIESTA-IoT admin role and testbed admin role).

- The admin create a new policy for a given REST resource (see Figure 16). Here, the URL and the wildcard pattern is entered for the URL resource or resources to be protected by the policy. For example: http://www.example.com/* means to apply the policy to all resources in this domain.

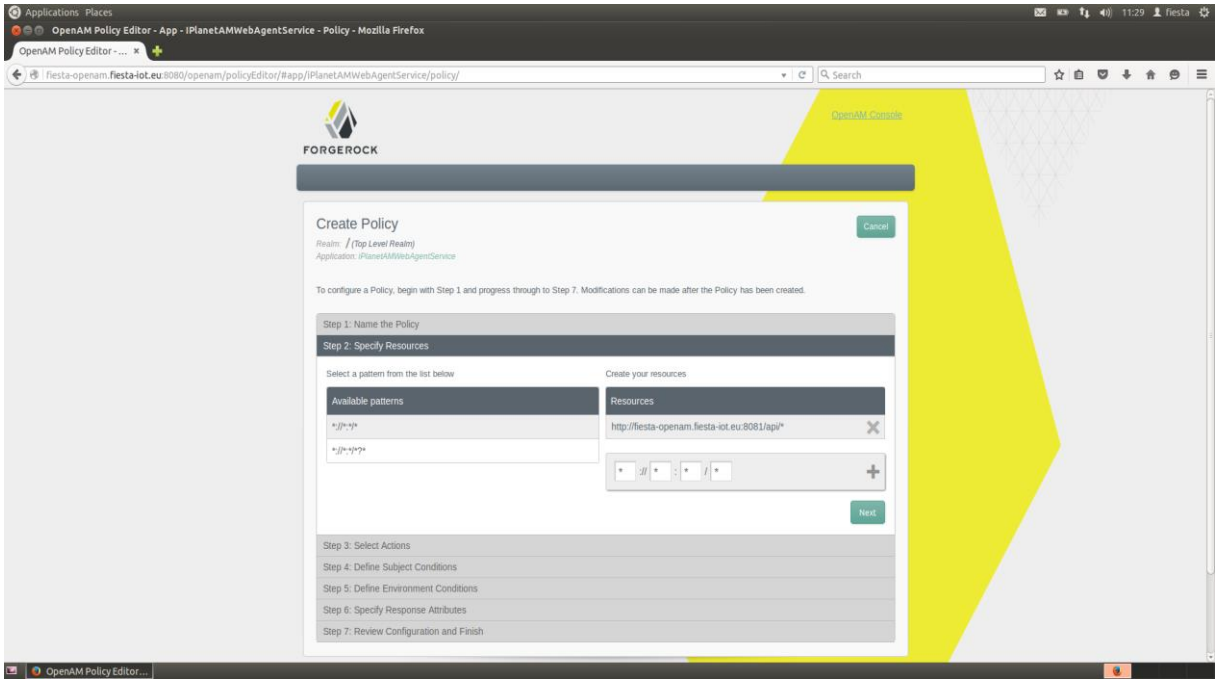

Figure 16: Policy Creation

- The next step is to set the access permissions for actions (HTTP actions e.g. GET, POST, PUT, DELETE, PATCH) on the specified protected URL. Permission can be granted or denied to the users who match the conditions.

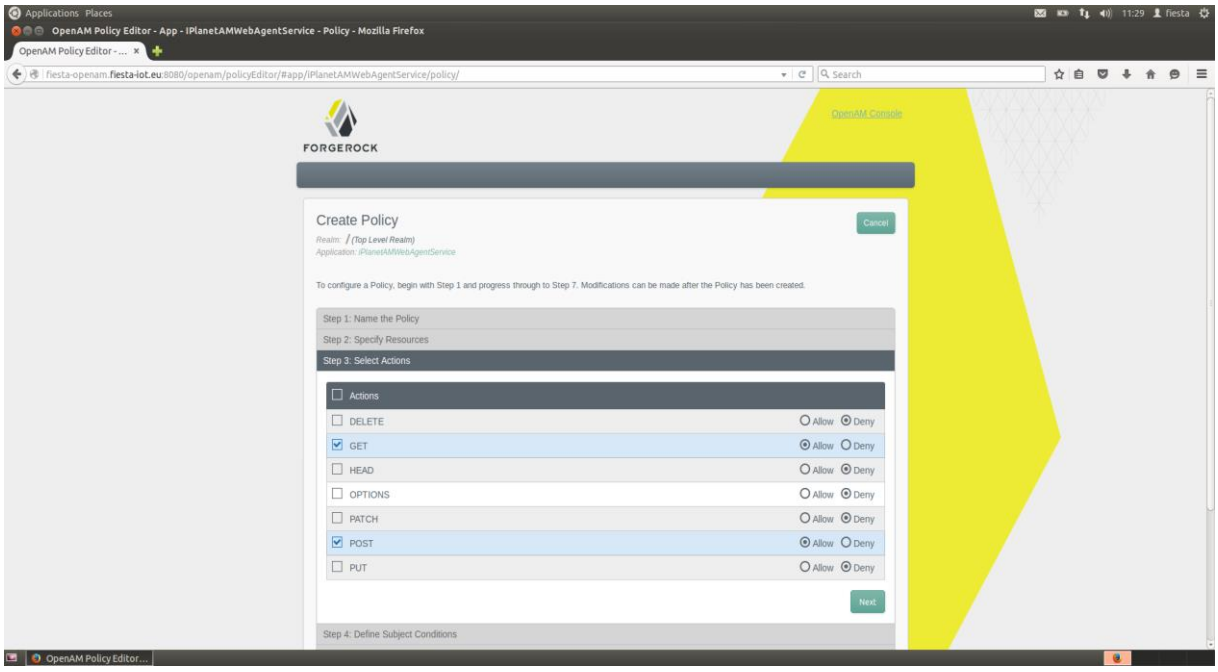

Figure 17: Policy Action setup

- The final step is to define the user roles and conditions that the policy applies to (Figure 18)

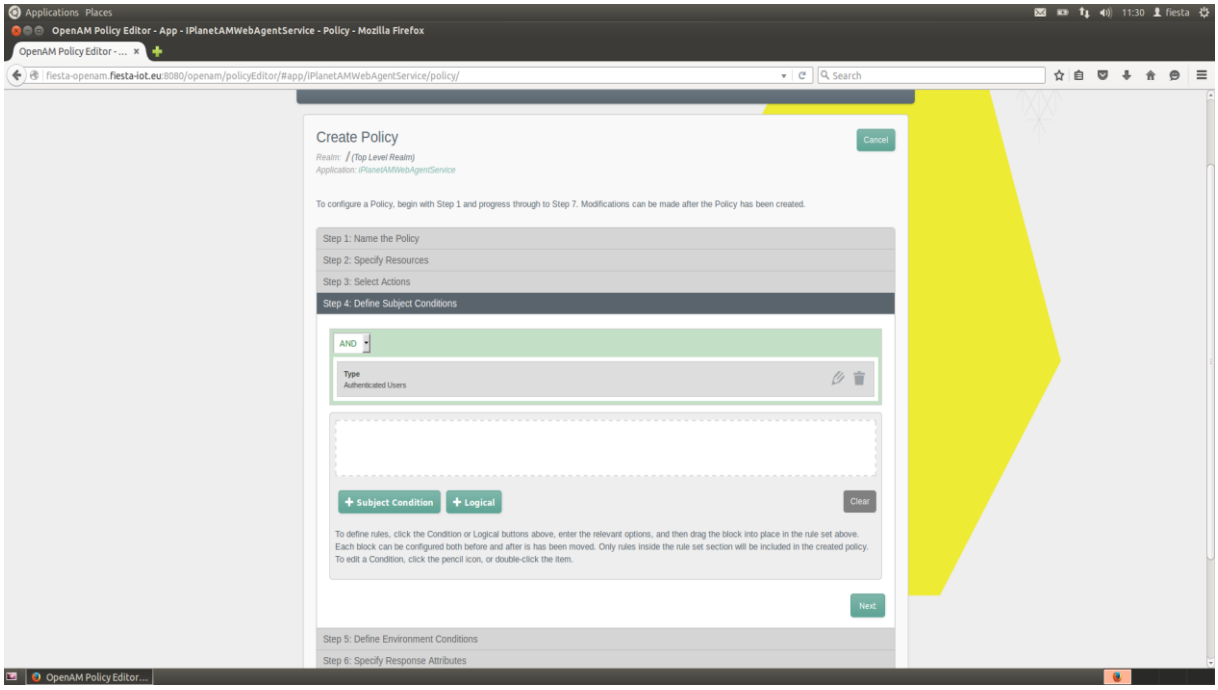

Figure 18: Policy Conditions

An extension of this use-case that will be considered in the final iteration of the architecture, is that of *federated identity management*. Experimenters who have a user account at a testbed or 3rd party in the FIESTA-IoT federation can authenticate with FIESTA-IoT using their testbed credentials. When the experimenter signs in to FIESTA-IoT, they are redirected to authenticate with their own identity provider (trusted by FIESTA-IoT)—once authenticated they are free to leverage the FIESTA-IoT services.

4.3 User Identity Database and API

OpenAM uses OpenDJ as the underlying directory for storing user information and credentials. This is a traditional LDAP based directory, with a REST API atop for access to information about users.

For example:

- Information about a user stored in OpenDJ

GET http://opendj.fiesta-iot.eu:8080/api/users/newuser


```
{
  "_id": "newuser",
  "_rev": "0000000023257469",
  "_schema": "frapi:opendj:rest2ldap:user:1.0",
  "_meta": {
    "created": "2016-06-24T12:20:45Z"
  },
  "userName": "newuser@example.com",
  "displayName": ["New User"],
  "name": {
    "givenName": "User",
```



```
    "familyName": "New"
  },
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
}
```

- To get the information about a user based on the bearer token attached to a http request. This returns the user id which can be used for further queries

POST –header "token: dsfsdjhfsdjh84758437*" http://vm1.fiesta-iot.eu:8080/openam/json/users?_action=idFromSession

```
{
  "id":"demo",
  "realm":"/",
  "dn":"id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
  "successURL":"/openam/console",
  "fullLoginURL":null
}
```

The important fields are explained as follows:

- “id” is the user’s unique identifier in the FIESTA-IoT domain.
- “realm” labels the security realm that the user is registered in; “/” means they are in the top level realm i.e. FIESTA-IoT.
- “dn” is the fully qualified domain name of the user.

5 CONCLUSIONS

5.1 Implementation of Use Cases

In this document we have presented the first version of the FIESTA-IoT security framework to secure and protect resources in the FIESTA-IoT architecture. The solution is based upon the usage of a well-established, open source, and community-based framework i.e. OpenAM. This reduces development effort because re-implementation of existing standards is not required. Further, such mature implementation has undergone significant scrutiny in the community and is there likely to have fewer security bugs and issues.

We now document the extent to which the use cases that were illustrated in the introduction have been realised by the configuration of the security technologies. It can be seen from Table 20 that the three use cases have been fully realised by OpenAM and OpenIG solutions presented in this document.

Table 20: Use Case Solutions

Use Case	Solution applied	Complete
UC2	OpenAM Identity Manager. User sign up and registration pages on FIESTA-IoT portal. Identity API for programmatic usage.	Y
UC3	OpenIG PEP to secure resource. GUI and APIs to enter policies. Appendix A describes how to deploy the PEP. Section 4 describes how policies can be created.	Y
UC1	OPENAM with OpenIG to authenticate and authorize access	Y

5.2 Next Steps

In the second part of the work in Task 4.2—the work will focus on the security of data relevant to IoT data. In particular, this will concentrate on the requirement that data privacy can be maintained during the experimentation carried out on the FIESTA-IoT platform.

For this purpose, three key paths will be followed and implemented:

- *User driven policies*. Users may submit data e.g. from their own resources within a testbed (smart home), or as part of a crowdsourcing platform. Rather than treat all resources in a testbed the same (i.e. the testbed sets the access policies), the user is able to set the policy for their data within the broader testbed. The use of UMA makes the implementation of this case feasible and we will investigate deploying it for the final release of the FIESTA-IoT platform.
- *Fine-grained access control*. FIESTA-IoT focuses on data. It may be beneficial for finer grained sub data to be made available e.g. part of the data may be

available to the experimenter, but not all of it. It might also be that the content itself informs who is allowed access to it. Again, the use of UMA and XACML will help allow these features to be implemented in the next version.

- *Controlled delegated access*. Users should be able to delegate access to other users (or groups). For example, an experimenter may have produced a set of results and wants to share these with a subset of experimenters, and/or allow one experimenter to execute the experiment that created them. We will seek to support delegated authorization in the FIESTA-IoT platform. The usage of OAuth 2.0 will provide a first step in reaching this implementation.

6 REFERENCES

[1] FIESTA-IoT Consortium, Bruno Almeida, Tiago Teixeira (Eds.), "Stakeholder Requirements, FIESTA-IoT Deliverable D2.1, July 2015.

[2] FIESTA-IoT Consortium, Francois Carrez (ed.), "FIESTA-IoT Meta-Cloud Architecture", FIESTA-IoT Deliverable D2.4, December 2015.

[3] OASIS Standards, "SAML Specifications: SAML v2.0", http://saml.xml.org/saml-specifications, last viewed August 2016.

[4] Gluu Inc. "Gluu Web Authentication / SSO Protocol Adoption Predictions",https://www.gluu.org/blog/gluu-web-authentication-sso-protocol-adoption-predictions, last viewed August 2016.

[5] N. Sakimura et al., "OpenID Connect Core 1.0 incorporating errata set 1", http://openid.net/specs/openid-connect-core-1_0.html

[6] D. Hardt (ed.), "The OAuth 2.0 Authorization Framework", http://oauth.net/2/

[7] APICrazy. "Identity Broker Service in SAML: Supporting Multiple Identity Providers & Service Providers", https://apicrazy.com/2014/08/04/identity-broker-service-in-saml-supporting-multiple-identity-providers-service-providers

[8] Thomas Hardjono (ed.), "User-Managed Access (UMA) Profile of OAuth 2.0", http://kantarainitiative.org/confluence/display/uma/Home, December 2015.

[9] Wikipedia, "Lattice-based access control", http://en.wikipedia.org/wiki/Lattice-based_access_control, Last viewed August 2016.

[10] M. Bauer et al. "Introduction to the Architectural Reference Model for the Internet of Things", http://www.iot-a.eu/public/public-documents/copy_of_d1.2, 2014.

[11] Jasig, "Central Authentication Service", https://wiki.jasig.org/display/CAS/Home, last viewed August 2016.

[12] Forgerock, "OpenAM – Access Management", http://openam.forgerock.org/, 2016.

[13] Gluu Inc, "Gluu Server Overview", https://www.gluu.org/gluu-server, 2016.

APPENDIX A –SECURING A SERVICE DEPLOYED IN A WILDFLY CONTAINER

This document provides the sequence of instructions to secure a web application deployed in a Wildfly container. This security will ensure that only authenticated FIESTA-IoT users are authorized to access the web application.

Pre conditions

- The service must be a web application utilised by a web browser client. For a web service/application that a client tool/service interacts with, see the documentation (Security using a FIESTA-IoT client tool).
- Java 8 is installed
- The following installation is to be carried out in a virtual machine running Ubuntu Server 14.04. While this installation and configuration is likely to work with other Operating System flavours - there may be minor changes required. Please seek support from the FIESTA-IoT team if you encounter configuration difficulties.
- The FIESTA-IoT OpenAM server is deployed at a Fully Qualified Domain Name, i.e., FQDN (URL: http:// vm1.fiesta-iot.eu:8080/openam)
- The VM must be on the same domain as the OpenAM server (using this configuration) e.g. vm2.fiesta-iot.eu
- Edit /etc/hosts to point change all IP addresses to the domain name

This document goes through the following steps:

- Install Wildfly 10
- Install an example Wildfly application
- Deploy the OpenIG gateway as a root application in Wildfly
- Configure the OpenIG gateway to protect the resource
- Demonstration of protected resource

INSTALL WILDFLY 10

1. Set to root: Sudo -i
1. sudo unzip wildfly-10.0.0.Final.zip -d /opt/
2. sudo ln -s /opt/wildfly-10.0.0.Final /opt/wildfly
3. sudo cp /opt/wildfly/bin/init.d/wildfly.conf /etc/default/wildfly
4. Edit variables in */etc/default/wildfly*:

```
##Location of JDK
JAVA_HOME="/usr/lib/jvm/java-8-oracle"
## Location of WildFly
JBOSS_HOME="/opt/wildfly"
## The username who should own the process.
```



```
JBOSS_USER=wildfly
## The mode WildFly should start, standalone or domain
JBOSS_MODE=standalone
## Configuration for standalone mode
JBOSS_CONFIG=standalone.xml
## Configuration for domain mode
# JBOSS_DOMAIN_CONFIG=domain.xml
# JBOSS_HOST_CONFIG=host-master.xml
## The amount of time to wait for startup
STARTUP_WAIT=60
## The amount of time to wait for shutdown
SHUTDOWN_WAIT=60
## Location to keep the console log
JBOSS_CONSOLE_LOG="/var/log/wildfly/console.log"
## Additionals args to include in startup
# JBOSS_OPTS="--admin-only -b 172.0.0.1"
```

```
6.  sudo cp /opt/wildfly/bin/init.d/wildfly-init-debian.sh
/etc/init.d/wildfly
7.  sudo chown root:root /etc/init.d/wildfly
8.  sudo chmod +X /etc/init.d/wildfly
9.  sudo update-rc.d wildfly defaults
10. sudo update-rc.d wildfly enable
```

Create directory for logs:

```
11. sudo mkdir -p /var/log/wildfly
```

Add system user to run WildFly:

```
12. sudo useradd --system --shell /bin/false wildfly
```

Change the owner of WildFly directories:

```
13. sudo chown -R wildfly:wildfly /opt/wildfly-9.0.2.Final/
14. sudo chown -R wildfly:wildfly /opt/wildfly
15. sudo chown -R wildfly:wildfly /var/log/wildfly
```

Next, edit the `standalone.xml` file using your preferred file editor and do the changes:

```
cd /opt/wildfly-8.2.0.Final/standalone/configuration
```

```
1       <interface name="management">
2           <any-address/>
3       </interface>
4       <interface name="public">
5           <any-address/>
6       </interface>
```

Start WildFly:


```
sudo service wildfly start
```

INSTALL WILDFLY APPLICATION

Check out the wildfly examples

1. `git clone https://github.com/wildfly/quickstart.git`
2. `cd /quickstart/helloworld-ws`
3. `mvn clean package wildfly:deploy`

Check at http://vm2.fiesta-iot.eu:8080/wildfly-helloworld-ws

INSTALL OPENIG AS WILDFLY ROOT

Get OpenIG and rename to ROOT.war

Create the OpenIG configurations:

Config.json

```
{
  "handler": {
    "type": "Router",
    "audit": "global",
    "baseURI": "http://fiesta-iot-portal.eu:8080/example",
    "capture": "all"
  },
  "heap": [
    {
      "name": "LogSink",
      "type": "ConsoleLogSink",
      "config": {
        "level": "DEBUG"
      }
    },
    {
      "name": "JwtSession",
      "type": "JwtSession"
    },
    {
      "name": "capture",
      "type": "CaptureDecorator",
      "config": {
        "captureEntity": true,
        "_captureContext": true
      }
    }
  ]
}
```

- `cp config.json /home/wildfly/.openig/config/config.json`
- `mkdir /home/wildfly/.openig/config/routes`


```
{
  "handler": {
    "type": "DispatchHandler",
    "config": {
      "bindings": [
        {
          "condition": "${request.cookies['iPlanetDirectoryPro'] == null}",
          "handler": {
            "type": "StaticResponseHandler",
            "config": {
              "status": 302,
              "reason": "Found",
              "headers": {
                "Location": [
                  "http://vm1.fiesta-
iot.eu:8088/openam/XUI/#login/&goto=${urlEncode(contexts.router.originalUri)}"
                ]
              },
              "entity": "Redirecting to OpenAM..."
            }
          }
        },
        {
          "comment": "This condition is optional, but included for clarity.",
          "condition": "${request.cookies['iPlanetDirectoryPro'] != null}",
          "handler": {
            "type": "Chain",
            "config": {
              "filters": [
                {
                  "type": "PolicyEnforcementFilter",
                  "config": {
                    "openamUrl": "http://vm1.fiesta-iot.eu:8088/openam/",
                    "pepUsername": "policyAdmin",
                    "pepPassword": "password",
                    "ssoTokenSubject": "${request.cookies['iPlanetDirectoryPro'][0].value}"
                  }
                }
              ],
              "handler": "ClientHandler"
            }
          }
        }
      ]
    }
  },
  "condition": "${matches(request.uri.path, '^/pep') and not contains(request.uri.path, 'not-
enforced')}"
}
```

- `cp 04-pep.json /home/wildfly/.openig/config/routes/04-pep.json`

Restart Wildfly

```
1. sudo service wildfly restart
```

CONFIGURE THE POLICIES

Create a Policy in OpenAM

Follow these steps:

1. Log in to OpenAM console as administrator (amadmin).
2. In the top-level realm, create an authorization policy in the iPlanetAMWebAgentService policy set called Policy for OpenIG as PEP.
3. Configure the policy with the following characteristics:
    4. **Resources**
        5. Protect the URL for the minimal HTTP server app.example.com:8081/pep.
        6. This policy applies to resources served by the minimal HTTP server as they are accessed through OpenIG.
    7. **Actions**
        8. Allow HTTP GET.
    9. **Subjects**
        10. Add a subject condition of type Authenticated Users.
11. Make sure all the changes are saved.

Create a Policy Administrator in OpenAM

Follow these steps:

1. In the top-level realm, create a subject with ID policyAdmin and password password.
2. Create a policyAdmins group and add the user you created.
3. In the privileges configuration, add the REST calls for policy evaluation privilege for the policyAdminsgroup.
4. This allows the user to request policy decisions.
5. Make sure all the changes are saved.

