Leveraging eID in the Private Sector

D4.2 eIDAS Interconnection Supporting Service

Document Identification			
Status	Final	**Due Date**	28/02/2018
Version	2.0	**Submission Date**	28/02/2018

Related Activities	Activity 4 - Activity 5	**Document Reference**	D4.2/5.2
Related Deliverable(s)	D4.1, D5.1	**Dissemination Level (*)**	PU
Lead Participant	UAegean	**Lead Author**	Petros Kavassalis Harris Papadakis Nikos Triantafyllou Katerina Ksystra
Contributors	UAegean	**Reviewers**	Nuria Ituarte, ATOS
			Elena M. Torroglosa Garcia, UMU

Keywords:
eIDAS API Adaptor, eIDAS Node, Service Provider, SAML 2.0, JWT, Systems Interconnection, Interoperability

This document is issued within the frame and for the purpose of the *LEPS* project. This project has received funding from the European Union's Innovation and Networks Executive Agency – Connecting Europe Facility (CEF) under Grant Agreement No.INEA/CEF/ICT/A2016/1271348; Action No 2016-EU-IA-0059 The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

List of Contributors	
Name	Partner
Petros Kavassalis	UAegean
Petros Kavassalis	UAegean
Harris Papadakis	UAegean
Nikos Triantafyllou	UAegean
Elena M. Torroglosa Garcia	UMU
Nuria Ituarte	ATOS

Document History			
Version	Date	Change editors	Changes
0.1	15/02/2018		Initial ToC and document draft
0.2	20/02/2018		Edited document draft
0.3-0.9	23/02/2018		Updated by UAegean
1.0	27/02/2018		Pre-Final version integrating the comments of reviewers and UAegean updates
1.1	28/02/2018	UAegean	Final updates by UAegean
1.2	28/02/2018	ATOS	Quality Check
2.0			FINAL VERSION TO BE SUBMITTED

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
EC	European Commission
eID	Electronic Identity
eIDAS	electronic IDentification, Authentication and trust Services
eID_EU	Brand name for the authentication via national eIDs and by using the eIDAS Network
SAML	Security Assertion Markup Language
JWT	JSON Web Token
SP	Service Provider under the definition of eIDAS SPs

1 Executive Summary

This deliverable describes in technical detail the functionality and the operation of an intermediary connection facility for the integration of a Service Provider (SP) to eIDAS Network, i.e. an eIDAS API Connector[1] that allows SPs to easily and transparently interoperate with an eIDAS Node (this applying to "proxy countries"). Through this API Connector (initially named Interconnection Supporting Service), a SP can formulate authentication requests to be sent to the eIDAS-Node and receive "pre-processed" authentication responses, while investing minimal technical and organizational effort. The API Connector developed in this project re-uses the essential of the functionality of eIDAS Demo SP package (part of the sample eIDAS implementation provided by CEF).

This document describes also the technical details of the deployment of LEPS API Connectors within ATHEX and Hellenic Post (ELTA) IT premises, as requested by the Grant Agreement.

[1] The name "eIDAS API connector(s)" replaces what is described in the GA as "Interconnection Supporting Service". A taxonomy of API Connectors developed in LEPS is provided in the subsequent chapters of this document.

2 Introduction

2.1 Purpose of the document

The objective of this document is to describe, in technical detail, an intermediary connection facility for the integration of a Service Provider (SP) to eIDAS Network, i.e. an eIDAS API Connector[2] that allows SPs to easily and transparently interoperate with an eIDAS Node (this applying to "proxy countries")[3]. Through this API Connector, a SP can formulate authentication requests to be sent to the eIDAS-Node and receive "pre-processed" authentication responses, while investing minimal technical and organizational effort.

Initially named "Interconnection Supporting Service", the API Connector developed in this project re-uses the essential of the functionality of eIDAS Demo SP package (part of the sample eIDAS implementation provided by CEF)[4]. It also benefits from the development of similar tool built in the context of STORK 2.0 project, and progressively re-eng after the end of STORK 2.0[5].

In addition, this document describes the technical details of the deployment of the API Connectors within ATHEX and Hellenic Post (ELTA) premises, in the objective to facilitate the integration of the e-Services provided by these operators with the eIDAS Network.

2.2 Relation to other project work

The document related to work undertaken in Tasks 4.2 and 5.2 (Activities 4 and 5, respectively), described as follows: *"Develop an elDAS Interconnection Supporting Service (extending and augmenting a similar tool built in the context of STORK 2.0 project). Under this task, the elD infrastructure specific protocols will be translated to common enterprise technologies (XSD, JSON, SharePoint etc.)"*. The successful deployment of the API Connector, as described in this document, is of critical importance for the completion of Tasks 4.3 and 5.3 (Integration to Greek PEPS/eIDAS-Node Connector) / Activities 4 and 5, respectively.

2.3 Structure of the document

This document presents in detail the proposed eIDAS Connection Modules for simplifying the integration of Service Providers (SPs) to the eIDAS infrastructure. The eIDAS Connection Modules described in this document cover a wide range of types of Service Providers and aim at reducing any type of SP connection to the eIDAS infrastructure effort as much as possible. The document is structured as follows:

[2] The name "eIDAS API connector(s)" replaces what is described in the GA as "Interconnection Supporting Service". A taxonomy of API Connectors developed in LEPS is provided in the subsequent chapters of this document.

[3] For a short presentation of the operational modes of eIDAS Network and the architecture of an eIDAS (proxy) Node, see: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/How+does+it+work+-+eIDAS+solution

[4] https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eIDAS+Node+integration+package#

[5] https://www.eid-stork2.eu/

Chapter 2 presents the need for eIDAS Connector Modules and their purpose, presenting a brief overview of each one, as well as the characteristics of the use cases for which each Module applies.

Chapter 3 presents each eIDAS Connector Module in detail. For each one, it provides an overview of the Module (its purpose, the functionality it offers to the SP, etc), as well as detailed steps for the deployment of the Module as well as its integration with the SP.

Chapter 4 describes the usage of the eIDAS Connector Modules within the context of specific LEPS SPs

The provided **Annexes** offer some technical details regarding certain mechanisms employed in the eIDAS Connector Modules.

3 "eIDAS Connection Modules for a Service Provider (SP)"

3.1 Single Sign-On, eIDAS Network and LEPS

E-commerce and online service providers, either private or public (such as e-government service providers) increasingly rely on third parties to identify and authenticate their users; Facebook, Google. Microsoft and others gradually become *de facto* providers of a personal e-identity[6] that is, after a User's request, is transmitted to e-commerce and online services providers, and stored locally by them. In this perspective, they act as Relying Parties (or Service Providers - SPs)[7] who delegate web login to Identity Providers (IdPs), in the context of a web SSO (Single Sign-on) System. Internet users and online consumers, the customers of e-commerce and online services providers, are pleased to use Internet Social Networks and other Internet mega-Providers, as Single Identity Providers for logins with the different online commerce and service providers they frequently visit and use. Users clearly benefit from managing access to multiple web sites with single login credentials.

On the other side, with third-party web Single Sign-On (SSO) services, ecommerce and online services offer to their customers the possibility to quickly and easily create personalized user accounts (which, of course, should be consolidated with existing or future accounts that use "local" credentials), and to login to their sites through the credentials users already have to access their favourite IdP. While third-party login systems may deploy now more efficient, more secure and less complex protocols, than in the past, to streamline the login process, they may not provide to SPs an adequate level of assurance and, in addition, they significantly increase risks for these organizations. In fact, current web SSO systems contain natural inefficiencies as far as information reliability and process security are concerned[8]. Normally, an Identity Management System should be able to answer two fundamental questions: who is the requested of access and how trustily this user can prove her identity. On both issues, web SSO through mega-Internet Providers fail short of the SP expectations. First, Social Networks and other popular web sites are not, by definition, authoritative sources of personal data. Second, the authentication level they provide does not guarantee strong security and privacy with sensitive data[9]. However, the popularity of mega-Internet Providers and the simplicity of the current web SSO process makes it a *de facto* solution in the third-part authentication landscape.

Nevertheless, given this set of problems with authentication based on current web SSO, other solutions may appear more attractive to SPs. The success of eduGAIN as Identity Federation, for the needs of

[6] A. Vapen et al, 2016, A Look at the Third-Party Identity Management Landscape, IEEE Internet Computing, available at http://ieeexplore.ieee.org/document/7420509/?reload=true

[7] A Relying party (RP) is a service, site, or entity that depends on a third party Identity Provider (IdP) to identify and authenticate a User who is requesting access to a digital resource owned by the R

[8] A. Dei and S. Weis, 2010, PseudoID: Enhancing Privacy for Federated Login, HotPETS, available at https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36553.pdf

[9] R. Horbe, 2014, A Model for Privacy-enhanced Federated Identity Management, eprint arXiv, available at file:///C:/Users/pkava/Documents/Projects/LEPS/Library/1401.4726.pdf

the academic community, is an interesting case to be carefully studied[10]. Similar efforts to build federated identity systems to access e-government systems are now deployed by various countries. Specific regulations are also introduced to promote strong authentication and encourage SPs to re-organize the management of electronic identities of their users. eIDAS in Europe is perhaps the most complete regulatory framework for electronic identification and trusted service provision, as well as a best practice for cross-border authentication. Besides regulation, eIDAS is a network of national proxy nodes and trusted IdPs providing different sets of personal a and legal identity attributes obtained from authoritative resources[11]. Service Providers (SP) may be willing to connect their services to eIDAS network to make their offerings accessible across EU countries borders and enjoy the legal coherence and (high) online safety requirements of the eIDAS framework.

eIDAS Network proposes an architecture based on STORK 2.0 project achievements. The architecture and the entities, which are essential part of the eIDAS Network architecture, are presented in the Figure below.

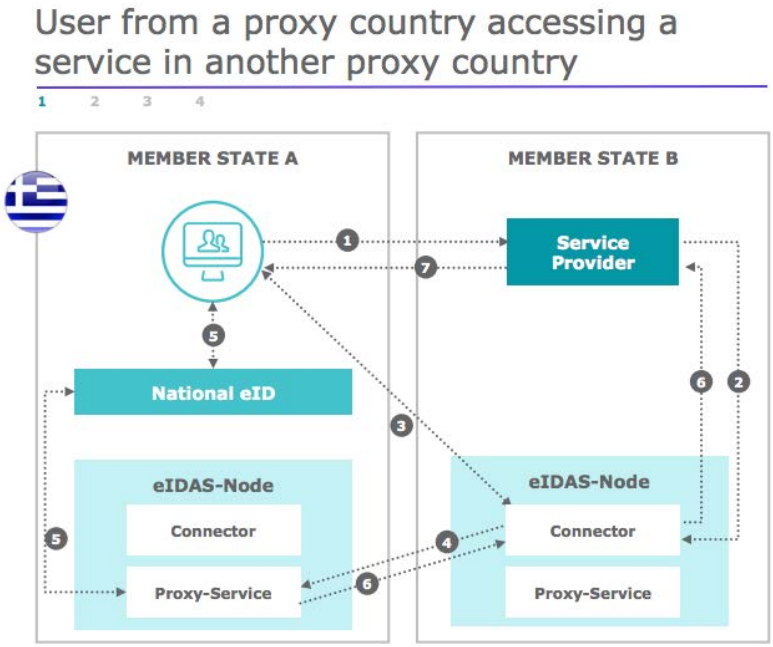


Figure 1: eIDAS Network functional architecture

According to a recent study, the eIDAS framework creates new opportunities for e-service providers[12]. However, the adoption of the eIDAS Network authentication services remains relatively low. Lack of information but also procedural and technical complexity are the most significant barriers to take-up of the (strong) authentication services provided by the eIDAS Network, especially in the private sector. LEPS aims precisely at bridging this gap, and the API Connector described in this document may

[10] E. Torroglosa-Garcia and Skarmeta-Gomez, 2017, Towards Interoperabilty in Identity Federation Systems, JoWUA, available at http://isyou.info/jowua/papers/jowua-v8n2-2.pdf

[11] See https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/Implement+or+operate+an+eIDAS+Node+-+Overview

[12] ENISA, 2017, eIDAS: Overview on the implementation and uptake of Trust Services One year after the switch over, available at https://www.enisa.europa.eu/publications/eidas-overview-on-the-implementation-and-uptake-of-trust-services/at_download/fullReport

significantly ease the integration of a SP with the eIDAS Network and, consequently, increase the adoption of eIDAS Network.

3.2 Supporting Service Providers, the essential adopters of eIDAS Network for authentication

eIDAS Network provides a pan-European infrastructure to connect national IdPs and eiD schemes based on SAML 2.0. Online Service Providers integrating with this infrastructure can transparently use a trusted federated environment for cross-border user authentication, based on technical specifications which have emerged through STORK 2.0 pilots experience and from requirements laid down in the eIDAS Implementing Act. But this opportunity comes also with a cost that is recognized by them[13].

In broader terms, when it is to identify the key factors influencing the further development of trust services market, the main stakeholders (Relying Parties are among them) cite "the evolution of business models" and a "change in the mind-set" believing that it will positively affect market development, while also mentioning "ease of use, mobility and user experience" and "accessibility of free/open source libraries and open specifications" as critical factors[14]. The deployment of trusted services needs as a result, to become more streamlined, while usability and easy integration with existing operational modes should be considered critical drivers for the market uptake. In such a context, we define in this document, SP-level optimal (technical) strategies for integrating the eIDAS Network. SPs are by definition the essential consumers of eIDAS-based authentication services. The adoption of the eIDAS infrastructure for authentication will depend on gaining their "buy-in" decision and commitment, especially of private sector SPs. But to increase SP adoption in the private sector, "of-the-shelf" eIDAS Network integration solutions should be made available to them, since such solutions are generally considered more cost-effective and "mind-set" active changers.

LEPS introduces the concept of a stand-alone API Connector that can be easily deployed within SP's premises and interoperate with existing applications and operations modes, thus making integration with a (proxy-based) eIDAS Node a standardized and lean process. An illustration of this is presented in the following Figure.

[13] See footnote 12
[14] See footnote 12

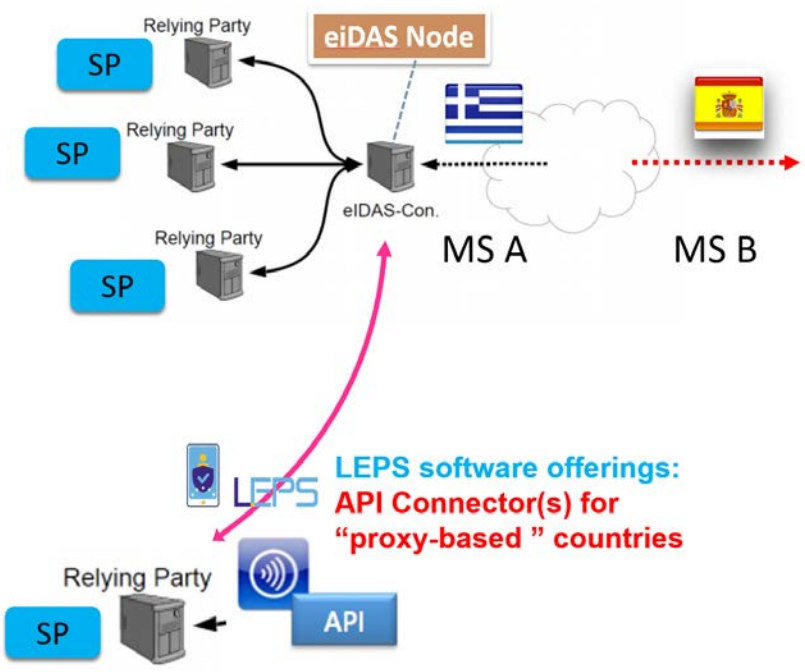


Figure 2: The role of the LEPS API Connector(s)

3.3 SP perspective: eIDAS Network integration tasks and API Connector

Successful connection of a Service Provider to the eIDAS Network is a multi-step process, which is comprised of five main requirements on the SP side. These requirements must be implemented carefully in order to ensure the correct operation of the authentication process, as well as the protection of the personal information of the user who requires a login through the eIDAS infrastructure (eID_EU).

These five high level requirements can be summarized in the following list:

1. Familiarization with SAML communication (protocol understanding and implementation).
2. Implementation of the required web interface (UI) for User interaction with the eIDAS-enabled services.
3. Formulation and proper preparation of an eIDAS SAML Authentication Request.
4. Processing of an eIDAS Node SAML Authentication Response and provision of the appropriate authentication process end events for success or failure.
5. Publishing of the SP's metadata, as is required by the CEF eIDAS Specifications.

To implement these requirements, an SP can deploy two "primitive" strategies: a) a custom "from scratch" eIDAS Specifications Implementation and, b) a custom but based on "Demo SP" package[15] eIDAS Specifications Implementation. The first strategy provides of course a great flexibility but it is requires a significant effort and time to invest and may be bug prone (if not well maintained and regularly updated). The second strategy may us a stable/tested implementation, but it still requires a substantial development effort, a good familiarization with the SAML 2.0 protocol and the functioning of an eIDAS Node, the design and development of the necessary UIs, testing etc. In addition, the available "Demo SP" package supports only JVM systems and applications.

[15]https://ec.europa.eu/cefdigital/artifact/content/repositories/eid/eu/eIDAS-node/1.4.0/eIDAS-node-1.4.0.zip

LEPS provides the ingredients of an alternative cost-effective strategy that consists of using a standardized, open source, of-the-shelf Connection Facility to make SP applications *de facto* interoperable with an eIDAS Node (proxy-based). The following figure illustrates their usage within the infrastructure.

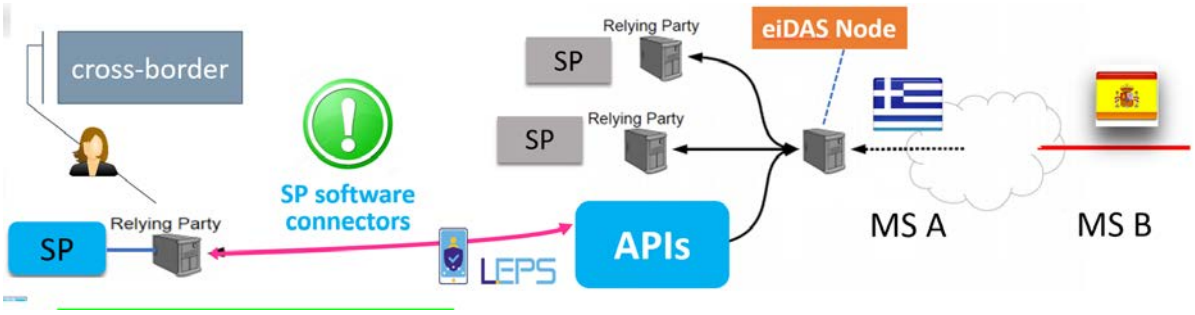


Figure 3: SP Integration with eIDAS Node using Connector(s)

Essentially, it is about using a REST-like API Connector that enables automated User Registration/Authentication to SP facilities via eID_EU by efficiently streamlining the whole authentication process from the SP application login to eIDAS Node and back.

It provides support for managing both authentication requests to and authentication responses from an eIDAS Node, while redirecting the User to the appropriate web pages. An overview of the functionalities provided by the Modules is presented in the following Table:

Table 1: Overview of functionalities of the eIDAS Connector Modules

Manage requests to eIDAS Node
Provide next-to-loginRequest-UI (includes country e-Form and SP-requested attributes list)
Construct and propagate SAML 2.0 authRequest
Publish SP metadata
Manage responses from eIDAS node
Process authResponse (as received from eIDAS Node)
(On success) Provide authResponse translated from SAML 2.0 to common enterprise standards (JSON, SOAP), and redirect user to SP
eIDAS authentication failure report
Support SP authentication policy (optional)

Basically, LEPS API connector acts as intermediary between an SP application/service and the eIDAS Node, as seen in the Figure below. They require only the deployment of the API Connector and the organization of a Service Point between the SP application/service and the API.

We define a Service Point as:
The minimal SP configuration that redirects a user's login request to an API (Connector), when the user selects "register/login via eID_EU"
The sum of endpoints where the API (Connector) will forward to SP application/service the authResponse data received from the eIDAS Node (auth success or failure report)

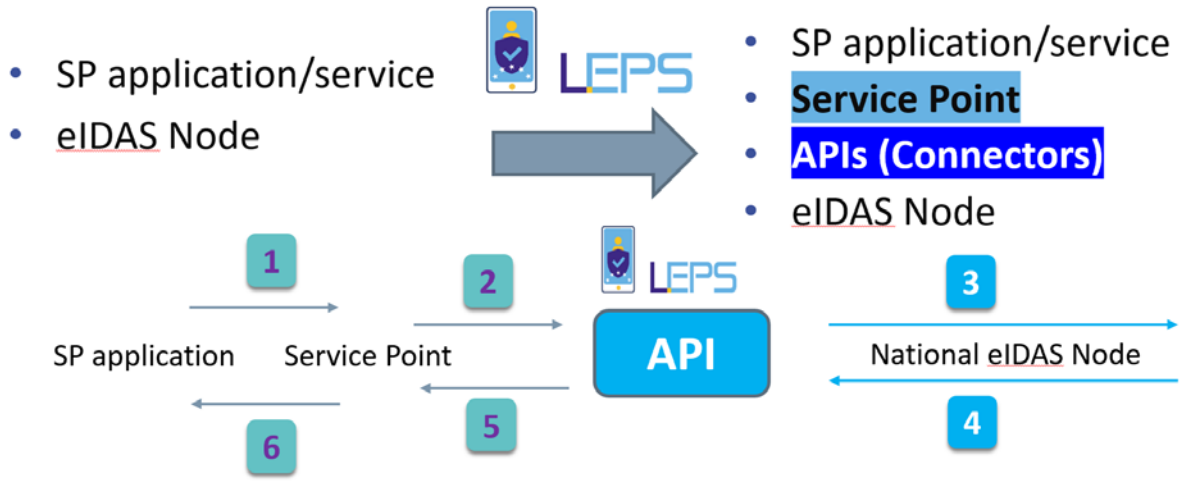


Figure 4: Illustration of the role of the Service Points within the architecture

3.4 Adapting to specific use-cases

LEPS API Connector provides a method to connect to an eIDAS (proxy-based) Node depending on specific SP requirements. Three efficient solutions (presented in Table 2) with different characteristics are provided, in order to cater for any usage and requirements scenario:

Table 2: List of offered LEPS API Connectors

LEPS API Connector(s)	
1	eIDAS SAML Tools Lib
2	eIDAS SP WebApp 2.0
3	eIDAS Inteconnection Supporting Service 2.0 (eIDAS ISS 2.0)

Table 3 summarizes the case scenarios where each of the above solutions can be used:

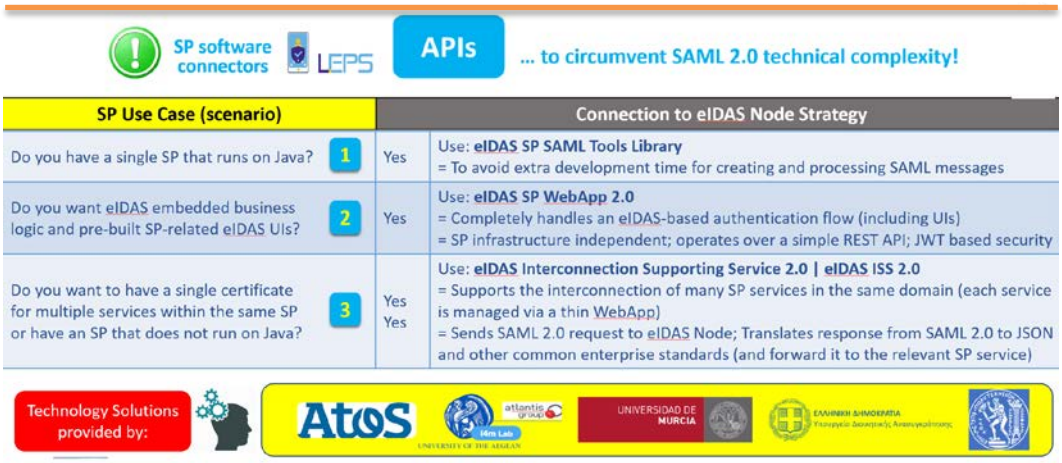
Table 3: API Connectors in context

API Connector	SP specific conditions	Why/When
1. eIDAS SP SAML Tools Library	- Java-based SP (developed from scratch) - No need for one certificate for many services within SP - No need for pre-built Uis	a. Avoid extra development time for creating and processing SAML messages
2. eIDAS SP WebApp 2.0	- Java or non-Java-based SP - No need for one certificate for many services within SP	a. Avoid development time for processing SAML messages b. Completely handles an eIDAS-based

	- Need for built-in UIs	authentication flow (including UIs) c. SP infrastructure independent; operates over a simple REST API d. Increased security, JWT based security
3. eIDAS ISS 2.0	- Java or non-Java-based SP (developed from scratch) - One certificate for many services within SP (*) Comes with or without SP e-Forms / thin WebApp	a. Avoid development time for processing SAML messages b. Supports the interconnection of many SP services in the same domain (each service is managed via a thin WebApp) c. Sends SAML 2.0 request to eIDAS Node; Translates response from SAML 2.0 to JSON and other common enterprise standards(WSDL) and forwards it to the relevant SP service d. Multiple services with the same SPs sharing one certificate

The available choices for a Service Provider can be explained on the basis of the following Table 4 and visualized as a BPMN diagram.

Table 4. Available SP choices and use-cases



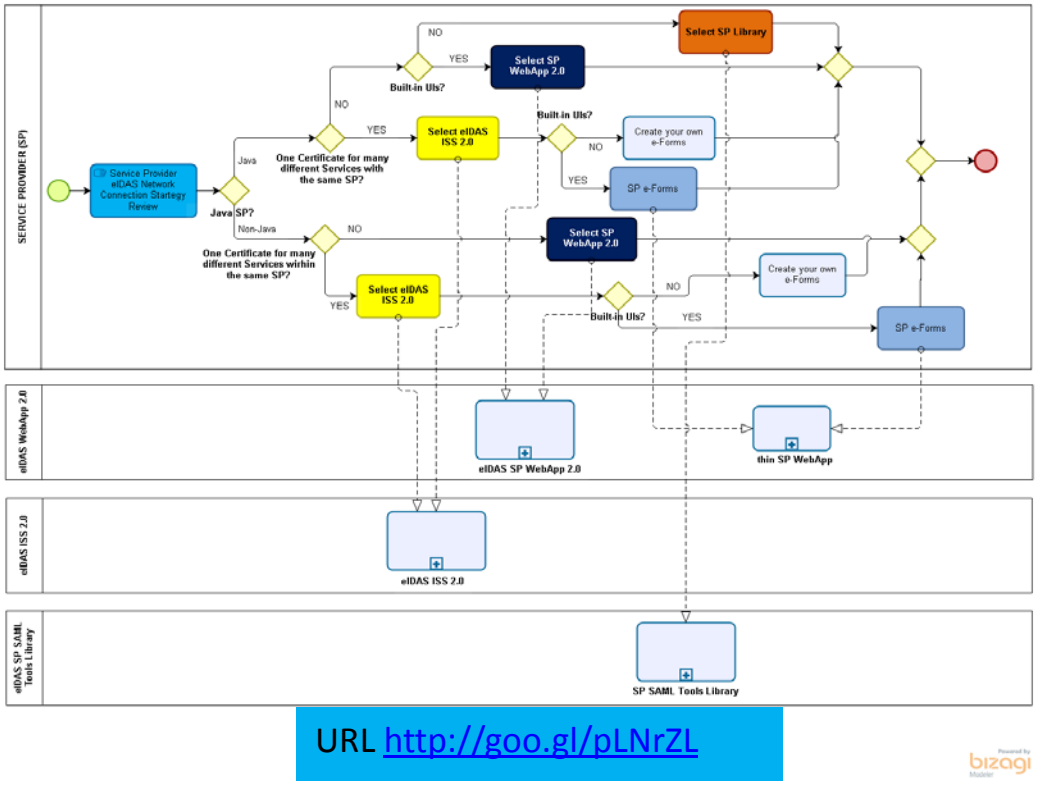


Figure 5: Connection Options for a Service Provider | SP (BPMN diagram)

4 Detailed description of LEPS eIDAS API connectors

4.1 eIDAS SP SAML Tools library

4.1.1 Overview

The eIDAS SP SAML Tools library can be used to simplify SP-eIDAS Node communication development, on the SP side. It is offered in the form of a Java library, which can be easily integrated into the development of any Java-based SP. The library itself is based on the CEF provided SP implementation (demo SP).

The library provides methods that a Java-based SP implementation can call to

- create SAML Requests (format, encode, encrypt),
- parse SAML Responses (decrypt, decode, parse)
- create the SP metadata xml, as required by the eIDAS Specs

It is designed to be easily integrated in any Java-based SP development. The SP developer can specify, on a per request basis:

- The list of attributes requested
- The Nationality of the SP
- The Nationality of the Citizen/User to be authenticated
- The requested Level of Assurance (LoA) for this Service

4.1.2 Deployment

The eIDAS SP SAML Tools Library package contains the following items to facilitate the integration of a new SP to the eIDAS Infrastructure:

Table 5: Contents of the eIDAS SP SAML Tools package

Filename	Purpose
Libs.zip	The Java libraries which facilitate integration and communication with the Greek eIDAS Node
configEidas	Folder containing the required configuration files
Login.jsp ReturnPage.jsp metadata.jsp	Sample JSP files containing examples for integrating and using the eIDAS SP SAML Tools Library

The contents of the configEidas folder must be copied to a directory in the local file system. After deployment is complete, the following environmental variable needs to be set in the tomcat execution environment (either as OS/AS environment variable or command-line parameter): SP_CONFIG_REPOSITORY. The variable must point to the location of the file system where the contents of the configEidas.zip file were copied.

The aforementioned directory contains the file eidasKeystore.jks, which must contain all the necessary certificates for secure and trusted communication with the eIDAS Node. The following steps need to be executed in order to prepare the keystore for operation.

1. Obtain a certificate that identifies the SP. The certificate must satisfy the criteria described in the eIDAS - Cryptographic requirements for the Interoperability Framework document[16], regarding SAML signing certificates.
2. Import the certificate in the keystore as a PrivateKeyEntry.
3. Provide the Greek eIDAS Node team with the public certificate of the SP, to be added to the Greek eIDAS Node list of trusted SPs.

The eIDAS SP SAML Tools Library requires the modification of a few configuration files. In each of the following configuration files, replace the existing entries with the following information (bold-faced values need to be replaced with the corresponding value):

SignModule_SP.xml

```
<entry key="response.sign.assertions">true</entry>
<entry key="keyStorePath"> eidasKeystore.jks</entry>
<entry key="keyStorePassword">keystore_password</entry>
<entry key="keyPassword">SP_certificate_password</entry>
<entry key="issuer">SP_certificate_issuer</entry>
<entry key="serialNumber">SP_certificate_serial_number</entry>
<entry key="keyStoreType">JKS</entry>

<entry key="metadata.keyStorePath"> eidasKeystore.jks</entry>
<entry key="metadata.keyStorePassword">keystore_password</entry>
<entry key="metadata.keyPassword">SP_certificate_password</entry>
<entry key="metadata.issuer">SP_certificate_issuer</entry>
<entry key="metadata.serialNumber">SP_certificate_serial_number</entry>
<entry key="metadata.keyStoreType">JKS</entry>
```

[16] https://joinup.ec.europa.eu/sites/default/files/document/2015-11/eidas_-_crypto_requirements_for_the_eidas_interoperability_framework_v1.0.pdf

EncryptModule_SP.xml

```
<!-- Key Encryption algorithm -->
<entry key="key.encryption.algorithm">http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p</entry>
<entry key="keyStorePath"> eidasKeystore.jks</entry>
<entry key="keyStorePassword">keystore_password</entry>
<entry key="keyPassword">SP_certificate_password</entry>

...

<!-- If not present then no decryption will be applied on response -->
<!-- Certificate containing instance private key-->
<entry key="responseDecryptionIssuer">SP_certificate_issuer</entry>
<entry key="serialNumber">SP_certificate_serial_number</entry>
<entry key="keyStoreType">JKS</entry>
```

sp.properties

sp.return=**URL of the return page** (e.g.: http://84.205.248.180/ReturnPage.jsp)**.**
This page receives and processes the authentication response data from the eIDAS Infrastructure.
….
sp.metadata.url=**URL of the metadata page** (e.g.:http://84.205.248.180:80/metadata.jsp). This is the URL under which the already provided metadata.jsp page is located
…..
sp.qaalevel=#
The level of Assurance required by this SP for the provided authentication data.
1=Non-existent
2=Low
3=Substantial
4=High

…

#Available connector nodes for this Sp
country.number=1
country1.name=**{Nationality code for the SP}**
country1.url=**{National eIDAS Node Service URL}**
country1.metadata.url=**{ National eIDAS Node Metadata URL }**
country1.countrySelector=**{ National eIDAS Node Coutry Selector Page URL }**

4.1.3 Integration

The final step in the integration procedure is the use of the eIDAS SP SAML Tools Library in order to be able to request and receive Authentication services. For this reason, the Library provides the following API classes and methods (examples of their use can be found in Section 5.1):

eu.eidas.sp.SpAuthenticationRequestData
Class containing the bundle of information fully describing an Authentication Request. Provides the following methods: • String **getSaml**(): Returns the encrypted, signed SAML document representing the request, to be sent to the eIDAS Node. **Parameters**: none **Returns**: the string containing the SAML document • String **getID**() Returns the ID representing this request. This information is mainly used to match the request with its corresponding response. **Parameters**: none **Returns**: the string representing the Request ID

eu.eidas.sp.SpAuthenticationResponseData
Class containing the bundle of information fully describing an Authentication Response. Provides the following methods: • ArrayList<String[]> **getAttributes**() Returns a list with the requested authentication attributes and their values. **Parameters**: none **Returns**: the list of Attributes and their values. Each entry in the list represents a single Attribute and contains an array of Strings, which in turn contains 3 cells: Cell 0: Attribute name Cell 1: true/false. Depending on whether the attribute is mandatory or optional. Cell 2: Attribute value • String **getID**() Returns the ID representing this response. **Parameters**: none **Returns**: the string representing the Response ID • String **getResponseToID**() Returns the ID of the request that triggered this response. This information is mainly used to match the request with its corresponding response. **Parameters**: none **Returns**: the string representing the corresponding Request ID • String **getResponseXML**() Returns the entire decrypted, response data, in XML form. **Parameters**: none **Returns**: the string representing the XML representing the Response data.

- String **getStatusCode**()
  Returns the Status Code of the response.
  Parameters: none
  Returns: the string representing the Status Code of the response.
  “urn:oasis:names:tc:SAML:2.0:status:Success”: Authentication success
  “urn:oasis:names:tc:SAML:2.0:status:Requester”: Error occurred
  “urn:oasis:names:tc:SAML:2.0:status:Responder”: IdP Authentication error
- String **getStatusMessage**()
  Returns a Human readable message representing the status (and reason of) of the response.
  Parameters: none
  Returns: the string representing a Human readable message representing the status (and the reason for that status) of the response.
- String **getLevelOfAssurance**() (**deprecated**)
  Returns the Level of Assurance (Low, Substantial, High) provided in this response.
  Parameters: none
  Returns: the string representing the LOA of this response.

eu.eidas.sp.SpEidasSamlTools

Class that provides the main Library methods for processing Authentication Requests and Responses. Provides the following methods:

- SpAuthenticationRequestData **generateEIDASRequest**(ArrayList<String> pal, String citizenCountry, String serviceProviderCountry, int qaaLvl)
  Constructs a new eIDAS SAML Authentication Request.
  Parameters:
  pal: the list of eIDAS Attribute names to be requested
  citizenCountry: a String representing the Country Code of the citizen to be authenticated
  serviceProviderCountry: a String representing the Country Code the SP operated in. In this case, this value should always be “GR”.
  qaaLvl: the Level of Assurance requested.
  Returns: a SpAuthenticationRequestData object representing the constructed Request.
- SpAuthenticationResponseData **processResponse**(String SAMLResponse, String remoteHost)
  Processes the Authentication Response.
  Parameters:
  SAMLResponse: The String containing the eIDAS Node provided Authentication Response. The String is contained as a parameter in the HTTP request used by the eIDAS Node to redirect the response to the SP’s Return Page.
  remoteHost: the URL representing the host which provided the response. This is also contained as a parameter in the HTTP request.
  Returns: a SpAuthenticationResponseData object representing the Response data.
- String **getNodeUrl**()
  Provides the SP with the URL of the eIDAS Node. This is the URL the Authentication Request is sent to, via HTTP POST or GET.
  Parameters: None
  Returns: the URL of the eIDAS Node.

4.2 eIDAS SP WebApp 2.0

4.2.1 Overview

The eIDAS WebApp 2.0 facilitates the integration of a SP with the eIDAS node by:

- Allowing the SP to avoid development time for processing SAML messages
- Completely handling an eIDAS-based authentication flow (including UIs)
- Being SP infrastructure independent; operates over a simple REST API
- Providing strong security assertions in the form of JWT (RFC 7519)

Important Note: In order to use the WebApp 2.0 for integration with the eIDAS GR node, it needs to be deployed in the same domain as the SP.

4.2.2 Deployment

Internally the eIDAS WebApp 2.0 uses the eIDAS SAML library presented in section 3.1. Thus, the same steps presented in the deployment of the SAML library are required. These can be summed up to:

- The copying of the configEidas folder
- Configure the keystore (eidasKeystore.jks)
- Edit the configuration files (SignModule_SP.xml, EncryptModule_SP.xml, sp.properties)

After configuring the SAML library we can proceed with the configuration of the eIDAS WebApp 2.0. This WebApp is offered as a Docker image. Thus, in order to deploy the WebApp 2.0 the hosting machine must have a functional Docker engine. For instructions of how to setup Docker please refer to: https://docs.docker.com/install/ and follow the installation instructions depending on your hosting system (linux, windows, mac). Additionally, for easier configuration of the container it is assumed that Docker compose is also installed (for instructions on installing docker compose please also refer to: https://docs.docker.com/compose/install/) although it is not a requirement.

Having configured the SAML library and having a working Docker engine, the only thing required is the creation/editing of a configuration file (yml) that will deploy a container from the image of the WebApp 2.0 endimion13/eidas-gr-loginwebapp/:latest. Please be advised that at times (depending on updates) no image might be tagged as “latest”. To this end please browse the above repository and select the latest image tag available (at the moment of writing of this document, the most recent tag is 3.2). Assuming that the WebApp 2.0 will be deployed at http://www.host.com then such a configuration file is given below:

Table 6: Example of WebApp 2.0 Configuration file

Docker Compose File (e.g. loginService.yml)
version: '3' services: loginWebApp2: image: endimion13/eidas-gr-loginwebapp:3.1 ports: - 9080:8090 - 9090:8090 environment: -EIDAS_PROPERTIES=http://eidas.europa.eu/attributes/naturalperson/CurrentFamilyName, http://eidas.europa.eu/attributes/naturalperson/CurrentGivenName, http://eidas.europa.eu/attributes/naturalperson/DateOfBirth, http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier - SP_FAIL_PAGE= http://www.host.com/authfail - SP_SUCCESS_PAGE=http://www.host.com/loginSuccess - SP_METADATA_URL=http://www.host.com:9090/metadata - SP_RETURN_URL=http://www.host.com:9090/eidasResponse - SP_LOGO=http://www.host.com/university-of-the-aegean.png - SP_CONFIG_REPOSITORY=/configEidas/ - SP_SECRET=secret - AUTH_DURATION=1800 volumes: - /configEidas:/configEidas

The various parts of this file are explained below:

- Version: denotes the syntax version of the composer file (up to the user for compatibility with the examples provided in this document please use 2 or 3.
- Services: denotes the start of the service sector of the compose file.
- loginWebApp2: denotes the name of the Thin WebApp 2.0 service name (up to the user to pick a friendly name). Format; hostmachineport:containerport.
- Image: the image to use to build the container.
- Ports: list of host machine ports that will be mapped to container ports (note that you should always map to container port 8090).
- environment: list of environmental variables that will be added to the container:
  - EIDAS_PROPERTIES: comma separated properties the SP requires from eIDAS.
  - SP_FAIL_PAGE: SP url to redirect to in case of authentication failure.
  - SP_SUCCESS_PAGE: SP url to redirect to in case of authentication success.
  - SP_METADATA_URL: WebApp 2.0 endpoint that the eIDAS node will use to retrieve the required communication metadata (do not change).
  - SP_RETURN_URL: WebApp 2.0 endpoint that the eIDAS node will redirect to after the authentication.

- SP_LOGO: a url containing the logo of the SP for UI customization.
- SP_CONFIG_REPOSITORY: path to the /configEidas directory (either do not edit this or edit it so that it matches the mounting of the /configEidas directory in the volumes section)
- SP_SECRET: string that will be used on HS256 signature of the generated assertions that will be propagated to the SP.
- AUTH_DURATION: integer denoting the duration for which the authentication cookie will be stored

- volumes: list of native directories that will be mounted as container directories. Here the mounting of the /configEidas directory should take place. Format: nativeDir:containerDir

Deployment now is simply a matter of starting the services defined in such a compose file, for example: docker-compose -f loginService.yml up

4.2.3 Integration

In order for the SP to interact with the WebApp 2.0 it needs to follow the next steps (in the rest of this section we assume that the WebApp 2.0 is deployed at http://www.host.com:8090):

- Upon authentication request, redirect the user to http://www.host.com:8090/login
- Build an endpoint (e.g. http://www.host.com/authsuccess) that the WebApp 2.0 will redirect to in case of authentication success
- (optionally) Build an endpoint (e.g. http://www.host.com/authfail) that the WebApp 2.0 will redirect to in case of authentication failure

The success and fail endpoints will consume a JSON Web Token (JWT) generated by the WebApp 2.0 delivered to the SP in the form of a cookie. JSON Web Tokens are an open, industry standard RFC 7519[17] method for representing claims securely between two parties. For this reason, the WebApp 2.0 needs to be deployed in the same domain as the SP. In the case of a successful authentication this token will contain the retrieved eIDAS identification attributes of the user. In case of an authentication failure this token will contain the authentication error message.

The generated authentication token is signed using HS256 (HMAC with SHA-256) with a secret shared between the SP and the WebApp 2.0. Upon receiving the token, the SP should validate it and read the identification attributes from its payload. The format of the generated JWT token payload is presented in the following table:

Table 7: Authentication Token Payload Format

Authentication Token Payload
{ "sub": "{ \"eid\":\eidValue\", \"personIdentifier\":\"eidValue\", ... \"eIDASAttribute_friendlyName\":\"attributeValue\""}"

[17] https://tools.ietf.org/html/rfc7519

```
}
```

An example of such a token is shown in the next table:

Table 8: Authentication Token Example

Authentication Token Example
{"sub":"{\"eid\":\"GR/GR/12345\",\"personIdentifier\":\"GR/GR/12345\",\"dateOfBirth\":\"1965-01-01\",\"currentFamilyName\":\"Garcia\",\"currentGivenName\":\"javier\"}" }

Libraries for the consumption of such a token exist in all major programming languages. For example, consuming a token in node.js (using the library jsonwebtoken) might can look like the following:

Table 9: Authentication Token Consumption

Consumption of Authentication Token

```
let  token = req.cookies.access_token;
 return new Promise( (resolve,reject) =>{
   jwt.verify(token,secretKey,{ algorithms: ['HS256']},function(err,token){
    if(err){
      reject(err);
      return {"message":"User not authorized"};
    }else{
      let result = JSON.parse(token.sub);
      result.eid = stripchar.RSExceptUnsAlpNum(result.eid);
      if(!result.firstName){
        result.firstName = result.currentGivenName;
      }
      if(!result.familyName){
        result.familyName = result.currentFamilyName;
      }
      result.userName = result.firstName+"_"+result.familyName;
      resolve(result);
    } });   });
```

4.3 eIDAS Interconnection Supporting Service (eIDAS ISS 2.0)

4.3.1 Overview

The eIDAS ISS 2.0 solution further simplifies the connection of any SP (regardless whether it is Java-based or not) further. It provides the following list of functionalities to the SP.

1. Enables SPs to communicate with the eIDAS Node without using SAML 2.0
2. One ISS 2.0 installation can support multiple services within the same SPs

3. Simple public web access to the log file of all requests served by the ISS 2.0
4. Endpoint communication method: JSON
5. Each request can be parameterized (explained below) by:
    a. the Attributes requested
    b. the Citizen/User Nationality
    c. the Level of Assurance Requested
6. On Authentication failure, two "Attributes" are provided ("StatusCode" and "StatusMessage"), providing the SP with information on the cause. These are provided and defined by the eIDAS infrastructure

4.3.2 Deployment

The ISS 2.0 package contains 2 files to facilitate the integration of a new SP to the Greek eIDAS Infrastructure:

Table 10: Contents of the ISS 2.0 package

Filename	Purpose
ISS2.war	The Java Web application package containing the ISS 2.0 executables
eidasKeystore.jks	ISS 2.0 keystore including eIDAS Node certificate

The aforementioned package contains the file eidasKeystore.jks, which must contain all the necessary certificates for secure and trusted communication with the eIDAS Node. The following steps need to be executed in order to prepare the keystore for operation.

1. Obtain a certificate that identifies the SP. The certificate must satisfy the criteria described in the eIDAS - Cryptographic requirements for the Interoperability Framework document, regarding SAML signing certificates.
2. Import the certificate in the keystore as a PrivateKeyEntry.
3. Provide the SP's National eIDAS Node team with the public certificate of the SP, to be added to the Greek eIDAS Node list of trusted SPs.

The ISS 2.0 app is provided as a pre-built .war archive, which can be easily deployed on a tomcat 7+ web server. After deployment is complete, the provided eIDAS keystore file needs to be copied to the hosting machine. We shall refer to this folder as "KSLOC".

In addition, the following information needs to be modified in the following configuration files (located under $TOMCAT_HOME/webapps/ISS2/WEB-INF/classes):

SignModule_SP.xml
<entry key="response.sign.assertions">true</entry> <entry key="keyStorePath">**KSLOC/keystore/eIDASKeystore.jks**</entry> <entry key="keyStorePassword">**keystore_password**</entry>

```
<entry key="keyPassword">SP_certificate_password</entry>
<entry key="issuer">SP_certificate_issuer</entry>
<entry key="serialNumber">SP_certificate_serial_number</entry>
<entry key="keyStoreType">JKS</entry>

<entry key="metadata.keyStorePath">KSLOC/keystore/eIDASKeystore.jks</entry>
<entry key="metadata.keyStorePassword">keystore_password</entry>
<entry key="metadata.keyPassword">SP_certificate_password</entry>
<entry key="metadata.issuer">SP_certificate_issuer</entry>
<entry key="metadata.serialNumber">SP_certificate_serial_number</entry>
<entry key="metadata.keyStoreType">JKS</entry>
```

EncryptModule_SP.xml

```
<!-- Key Encryption algorithm -->
<entry key="key.encryption.algorithm">http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p</entry>
<entry key="keyStorePath">KSLOC/keystore/eIDASKeystore.jks</entry>
<entry key="keyStorePassword">keystore_password</entry>
<entry key="keyPassword">SP_certificate_password</entry>
<!-- Management of the encryption activation -->
<entry key="encryptionActivation">KSLOC/tomcat/encryptionConf.xml</entry>
...

<!-- If not present then no decryption will be applied on response -->
<!-- Certificate containing instance private key-->
<entry key="responseDecryptionIssuer">SP_certificate_issuer</entry>
<entry key="serialNumber">SP_certificate_serial_number</entry>
<entry key="keyStoreType">JKS</entry>
```

sp.properties

```
#Sp Country
sp.country=Code of the SP's Country

#Sp return url
sp.return= Return URL of the SP (eg: https://ip:port/ISS2.0/ServiceRedirect)
sp.metadata.url= Metadata URL of the ISS 2.0 (eg: https://ip:port/ISS2.0/metadata)
….
```

sp.qaalevel=#
The level of Assurance required by this SP for the provided authentication data.
1=Non-existent
2=Low
3=Substantial
4=High
….

#The Service URLs (that we provide support)
#For more information see the Integration Section
spID.ds.url=**SP Retrieve Requested Attributes Endpoint URL** (SP Endpoint 1)
spID.ss.url=**SP Receive Requested Values Endpoint URL** (SP Endpoint 2)
spID.sr.url=**SP Authentication success redirect URL** (SP Endpoint 3)
spID.sf.url=**SP Authentication failure/error redirect URL** (SP Endpoint 4)
spID.mode=(**json|ws) communication mode.**
…..

#Available connector nodes for this Sp
country.number=1
country1.name=**{Nationality code for the SP}**
country1.url=**{National eIDAS Node Service URL}**
country1.metadata.url=**{ National eIDAS Node Metadata URL }**

In addition, after deployment is complete, the following environmental variable needs to be set in the tomcat execution environment (either as OS/AS environment variable or command-line parameter): SP_CONFIG_REPOSITORY. The variable must point to the location $TOMCAT_HOME/webapps/ISS2/WEB-INF/classes.

4.3.3 Integration

The ISS 2.0 requires the following steps for integration with each SP:

1. Each SP must redirect each user to be authenticated to a specific endpoint of the ISS 2.0 app.
2. Each SP must provide to the ISS 2.0, two endpoints (Endpoints 1 and 2) for the ISS 2.0 to contact during the authentication of the user.
3. Each SP must also provide to the ISS 2.0, two SP URLs (Endpoints 3 and 4) for the user to be redirected to, after the conclusion of the authentication process.

Endpoint 1 is contacted by the ISS 2.0 and should provide the list of the eIDAS Attributes to be requested during this Authentication.

Endpoint 2 is contacted by the ISS 2.0, providing the results of the Authentication process. The SP implementing the Endpoint should respond with whether the user described by the results is accepted or not.

Endpoint 3 will be used by the ISS 2.0 to redirect the user to, in the case Endpoint 2 responds with acceptance.

Endpoint 4 will be used by the ISS 2.0 to redirect the user to, in the case Endpoint 2 responds with non-acceptance.

Both Endpoints 1 and 2 provide a JSON formatted document. The format of the document is explained in Section 5.2.

Figure 6 illustrates the flow of the Authentication process, using ISS 2.0:

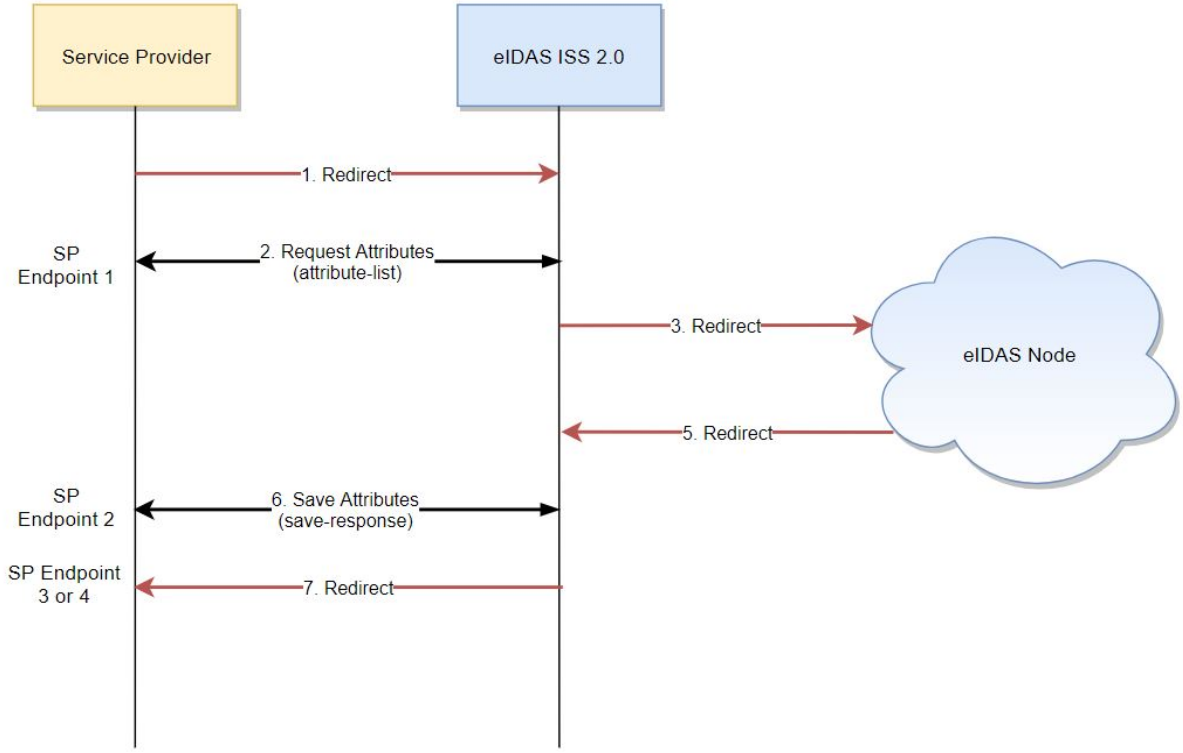


Figure 6: Authentication process flow

1. The SP redirects the user to be authenticated to a specific endpoint of the ISS 2.0 app
2. The ISS 2.0 app contacts the SP Endpoint 1 and receives a list of eIDAS Attribute Names.
3. The ISS 2.0 creates the corresponding eIDAS SAML Authentication Request and redirects it and the user to the eIDAS Infrastructure
4. The user authenticates in the eIDAS Infrastructure.
5. The eIDAS Infrastructure redirects the eIDAS Authentication Response as well as the user, back to the ISS 2.0 app.
6. The ISS 2.0 app contacts the SP Endpoint 2 and provides the SP with the results of the Authentication process. The SP Endpoints responds with “OK” or “NOK” (acceptance/non-acceptance).
7. The user is redirected back to the SP, using the appropriate redirection Endpoint (3 or 4).

In order to differentiate between several concurrent Authentication efforts (for the same SP), the SP must generate a unique identifier for each effort. Thus, the first step for the SP is to create a unique

random identifier. This identifier will be unique and different for each user request, much like a session ID. We call this unique identifier, a “token” and it will be used by the ISS 2.0 and the SP in order to differentiate between several, concurrent user authentication requests. After the creation of the token the SP will redirect the user to the ISS 2.0 entry point. For demonstration purposes we will assume that the unique token that was generated is: 1234567890, so the entry point of the ISS 2.0 is:

https://localhost/ISS2/ValidateToken?t=1234567890

In addition to the “t” parameter (which represents the token), some other parameters (some mandatory and some optional) must be included in the user redirection URL to the ISS 2.0. The following table enumerates the parameters of the HTTP request that redirects the user to the ISS 2.0:

Table 11: SP to ISS 2.0 redirection parameters

Parameter Name	Parameter Purpose	Mandatory?
t	SP-generated random Token. Used to identify authentication requests	yes
sp	ID of the requesting SP as defined in the ISS 2.0 sp.properties configuration file	yes
cc	Citizen Country Code. If not provided, the user will be requested to choose between the supported countries	no
saml	Generated SAML version definition. While communicating with the eIDAS infrastructure, the value of this parameter should always be “eIDAS”.	yes
qaa	The level of Assurance required for this request for the provided authentication data (1=Non-existent,2=Low,3=Substantial,4=High) If not provided, the qaa defined in sp.properties will be used.	No

(Entry point example: https://localhost/ISS2/ValidateToken?t=0f985470-3adb-4b33-8bb6-d6f515ab8bce&sp=sp1&cc=GR&saml=eIDAS)

The following table summarizes the specifications for the interaction between the SP and the ISS 2.0.

Table 12: SP-ISS 2.0 interaction specifications

SP Endpoint	Request (Parameters-Purpose)	SP Reply
1	t=token	A JSON describing the requested attributes (see Appendix I. Attribute List Retrieval API)
2	t=token, r=json message	User accepted: “OK”

	Pushes the attribute values to the SP (see Appendix I. Attribute Values Storage API)	User NOT accepted: “NOK” (see Section 3.2. Attribute Values Storage API)
3, 4	Redirection of user back to SP to a) SP Endpoint 3: if reply of Endpoint 2 was “OK” b) SP Endpoint 4: if reply of Endpoint 2 was “NOK”	N/A

In case of authentication failure, only two “Attributes” are provided in the “r” parameter of the ISS 2.0 request to Endpoint 2. These “Attributes” are called “StatusCode” and “StatusMessage” providing the SP with information on the cause of failure. These messages are provided and defined by the eIDAS infrastructure.

Example:

```
{
"StatusCode"     :{"value": "urn:oasis:names:tc:SAML:2.0:status:Requester",}
"StatusMessage":{"value": "202007 - Consent not given for a mandatory attribute.",}
}
```

4.3.4 SP e-Forms / Thin WebApp

4.3.4.1 Overview

The Thin WebApp complements the ISS 2.0 functionality by containing a pre built UI that the SPs can use and also require from the SPs to build fewer end points in order to retrieve the identification attributes from the eIDAS node. The retrieved attributes get bundled together as a JWT token that arrives to the SP in the form of a cookie (auth_token).

Important Note: In order to use the Thin WebApp for integration with the eIDAS GR node, a fully functional instance of the ISS 2.0 service must be deployed.

4.3.4.2 Deployment

First let us emphasise that the Thin WebApp needs to be deployed in the same domain as the SP. This is required because the SP needs be able to retrieve the authentication token (that the Thin WebApp generates) which is transferred as a cookie.

The Thin WebApp is offered as a Docker image thus providing platform agnostic deployment. In order to deploy the ThinWebApp as a docker container, the hosting machine must have a functional Docker engine. For instructions of how to setup Docker please refer to: https://docs.docker.com/install/ and follow the installation instructions depending on your hosting system (linux, windows, mac). Additionally, for easier configuration of the container it is assumed that Docker compose is also installed (for instructions on installing docker compose please also refer to: https://docs.docker.com/compose/install/) although it is not a requirement. With Docker and

Docker Compose set up the only thing required is a configuration file (yml) that will deploy a container from the image **endimion13/eidas-gr-isswebapp:latest**. Please be advised that at times (depending on updates) no image might be tagged as "latest". To this end please browse to the repository endimion13/eidas-gr-isswebapp and select the latest image tag available (at the moment of writing this document, the most recent tag is 1.3).

For the rest of this document we assume that the Thin WebApp is to be deployed at http://www.host/com. Also, we assume that the Thin WebApp will be available through the port 8090 of the hosting machine. Finally, we assume that the ISS 2.0 instance is deployed at http://www.host.com/ISSPlus

4.3.4.3 ISS 2.0 integration

The Thin WebApp exposes the following endpoints that need to be inserted as configuration parameters to the ISS 2.0 instance (assuming the deployment parameters of the previous paragraph).

- http://www.host.com:8090/attributeList. This endpoint returns the list of eIDAS attributes required by the SP which are queried by the ISS 2.0
- http://www.host.com:8090/issResponse. This endpoint parses and handles the response form the ISS 2.0
- http://www.host.com:8090/authsuccess. This endpoint handles the success redirection from ISS 2.0
- http://www.host.com:8090/authfail/. This endpoint handles the error redirection from ISS 2.0

Finally, upon setting up the integration with the ISS 2.0 instance an SP identifier should be created for the Thin WebApp 2.0. For the rest of this document we assume that this identifier will be sp2. For instructions on how these integration parameters are added to the ISS 2.0 configuration please refer to section 3.3.2 Deployment.

4.3.4.4 Docker Container Configuration

After configuring the ISS 2.0 instance we are ready for the deployment of the Thin WebApp. The easiest way is by defining a Docker Compose file. This file should follow the structure of the following snippet. We will go over the specific parameters one by one next.

Table 13: ThinWebApp 2.0 Docker Compose File example

Docker Compose File (e.g. loginService.yml)

version: '2'
services:
  issLoginWebApp:
  container_name: issLoginWebApp
  image: endimion13/eidas-gr-isswebapp:1.0
  expose:
  - 8090

```
    ports:
      - 8090:8090
    environment:
      -EIDAS_PROPERTIES=http://eidas.europa.eu/attributes/naturalperson/CurrentFamilyName,
http://eidas.europa.eu/attributes/naturalperson/CurrentGivenName,
http://eidas.europa.eu/attributes/naturalperson/DateOfBirth,
http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier
      - SP_FAIL_PAGE=https://www.google.com
      - SP_SUCCESS_PAGE=http://138.68.103.237/loginSuccess
      - SP_LOGO=http://trc.aiest.org/wp-content/uploads/2013/04/university-of-the-aegean.png
      - ISS_URL=http://84.205.248.180/ISSPlus/ValidateToken
      - SP_ID=sp2
      - SP_SECRET=secret
      - AUTH_DURATION=1800
```

- Version: denotes the syntax version of the composer file (up to the user for compatibility with the examples provided in this document please use 2 or 3.
- Services: denotes the start of the service sector of the compose file.
- issLoginWebApp: denotes the name of the Thin WebApp 2.0 service name (up to the user to pick a friendly name).
- Container_name: the name of the container (pick a friendly name again).
- Image: the image to use to load the container.
- expose: list of ports that the container will expose to other container even though they might not be in the same docker network.
- Ports: list of host machine ports that will be mapped to container ports (not that you should always map to container port 8090).
- Environment: list of environmental variables that will be added to the container.
- EIDAS_PROPERTIES: comma separated properties the SP requires from eIDAS.
- SP_FAIL_PAGE: SP url to redirect to in case of authentication failure.
- SP_SUCCESS_PAGE: SP url to redirect to in case of authentication success.
- SP_URL: url of the instance of ISS 2.0 that the Thin WebApp module is connected to pointing to the endpoint ValidateToken of ISS 2.0.
- SP_ID: the id of the SP as that was configured in ISS 2.0.
- SP_SECRET: string that will be used on HS256 signature of the generated assertions that will be propagated to the SP.
- AUTH_DURATION: integer denoting the duration for which the authentication cookie will be stored.

Deployment with such a file is simply a matter of starting the services defined in such a compose file, for example: docker-compose -f loginService.yml up

4.3.4.5 Integration

In order for the SP to interact with the Thin WebApp, it needs to follow the next steps (in the rest of this section we assume that the Thin WebApp is deployed at http://www.host.com:8090):

- Upon authentication request, redirect the user to
- Build an endpoint (e.g.) that the Thin WebApp will redirect to in case of authentication success
- (optionally) Build an endpoint (e.g.) that the Thin WebApp will redirect to in case of authentication failure

The success and fail endpoints will consume a JSON Web Token (JWT) generated by the Thin WebApp delivered to the SP in the form of a cookie. JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. For this reason the Thin WebApp needs to be deployed in the same domain as the SP. In the case of a successful authentication, this token will contain the retrieved eIDAS identification attributes of the user. In case of an authentication failure, this token will contain the authentication error as that was returned by the ISS 2.0.

The generated authentication token is signed using HS256 (HMAC with SHA-256) with a secret shared between the SP and the Thin WebApp. Upon receiving the token, the SP should validate it and read the identification attributes from its payload. The format of the generated JWT token payload is presented is identical to the one presented in section 3.2.3.

4.4 API Connectors distribution via GitHub as open source software

We provide the links to the Github repositories, where the source code for each Module is stored:

eIDAS SAML Tools Library:
https://github.com/ellak-monades-aristeias/eIDAS-SP-SAML-Tools-Library

eIDAS SP WebApp 2.0
https://github.com/ellak-monades-aristeias/eIDAS-SP-WebApp/

eIDAS Inteconnection Supporting Service 2.0
https://github.com/ellak-monades-aristeias/eIDAS-ISS

e-Forms / Thin Web App
https://github.com/ellak-monades-aristeias/eIDAS-SP-WebApp/

5 Use of LEPS eIDAS API Connectors in Activities 4 & 5

ATHEX and ELTA (Hellenic Post) will deploy and use LEPS eIDAS API Connectors for the integration of their applications/services with the Greek eIDAS Node, in the context of Activities 4 & 5 respectively.

5.1 ATHEX – Activity 4

In the case of the ATHEX Services, the ability of a single eIDAS ISS 2.0 to support multiple Services, makes it a prime candidate for its adoption in this scenario, as seen in the following Table.

Table 14: Athex Services and corresponding Module

Application/Service	eIDAS API Connector
ATHEX Sign (Remote eSignature Service) Customer registration (pre-activation procedure)	eIDAS ISS 2.0
ATHEX AXIAweb (Receive electronic information on an Investor's positions in Greek Central Securities Repository) Service login	
ATHEX AXIAweb (Receive electronic information on an Investor's positions in Greek Central Securities Repository) Customer registration (pre-activation procedure)	

Figure 7 illustrates the architecture that will be used in the ATHEX case.

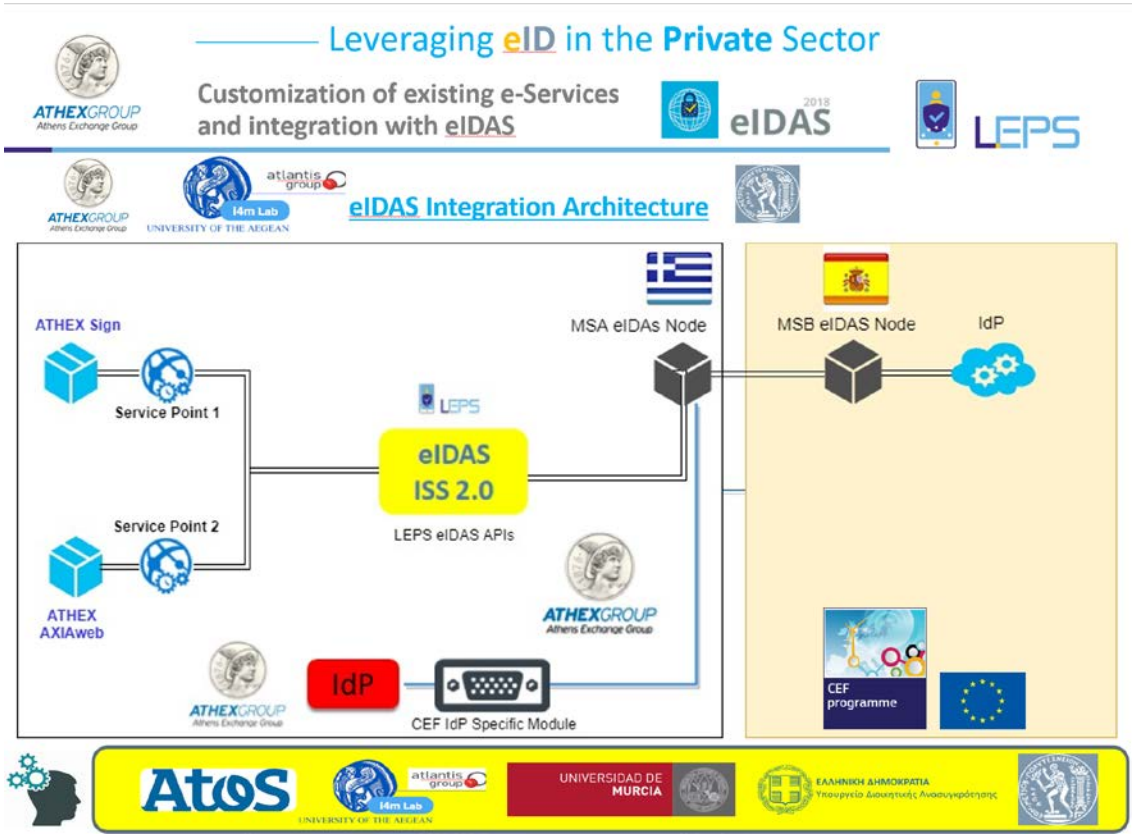


Figure 7: Athex Services Architecture

5.2 Hellenic Post (ELTA) – Activity 5

As in the case of the ATHEX Services, the ability of a single eIDAS ISS 2.0 to support multiple Services, makes it a prime candidate for its adoption in the services provided by the Hellenic Post, as seen in the following Table:

Table 15: Hellenic Post Services and corresponding Module

Applicatiion/Service	eIDAS API Connector
ELTA eDelivery Hybrid Service (cross-border exchange of electronic documents)	eIDAS SP WebApp 2.0
ELTA Online Postal Services ELTA portal/e-shop	eIDAS ISS 2.0
ELTA Online Postal Services Parcel Delivery Voucher	
ELTA Online Postal Services Online Zip Codes for Business Users	

The eDelivery Hybrid Service however required additional Bussiness Logic implemented in the Module to be used, which requires the use of the eIDAS SP WebApp 2.0. The architecture is presented in Figure 8.

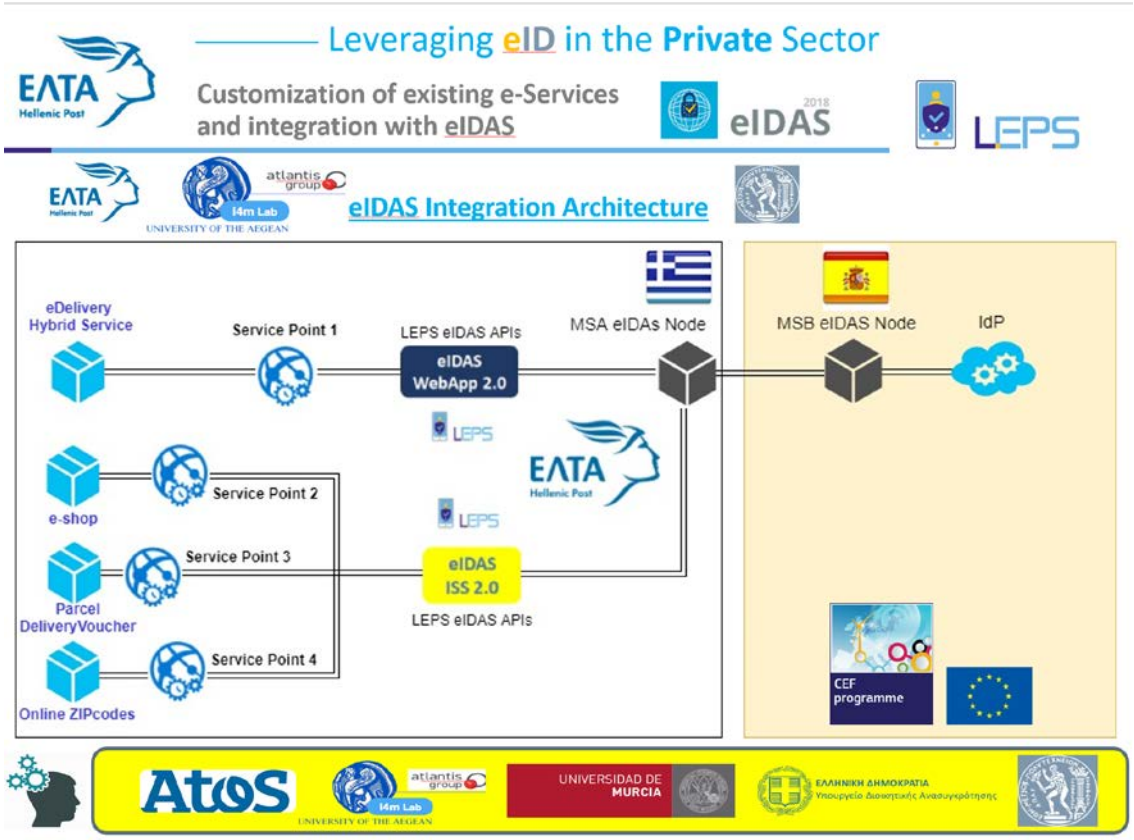


Figure 8: Hellenic Post Services Architecture

5.3 SP (ATHEX/ELTA) – LEPS eIDAS API Connector – eIDAS Node: The integration architecture in detail

The Figure below displays the complete flow for the integration process. This flow is based on the registration use case and login use case as described in section 2, The screens displayed to the user can be seen in section 3.2.1 and section 3.2.2.

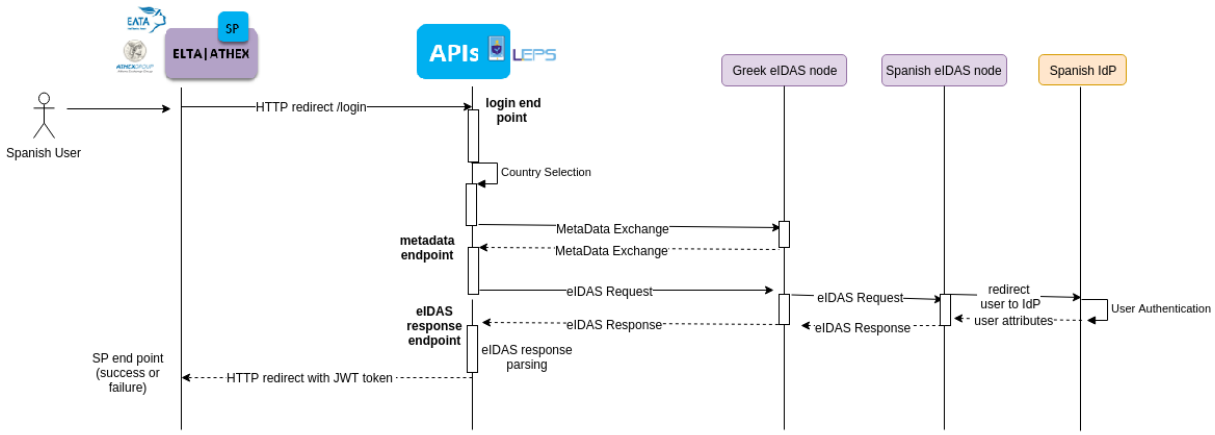


Figure 9: Integration Flow (SP -eIDAS API connector - eIDAS Node)

- The **Service Provider** configures and deploys a (API Connector) UI instance (SP e-Forms)
    - The configuration requires the definition of which attributes will be requested from the eIDAS Network
- The **Service Provider** redirects authentication requests from the application login page (Login with eID_EU) to the deployed **API Connector/UI**.
    - Redirection is a simple browser redirection to the "Country e-Form" (no parameters are required)
    - The User selects "Country"
- The **API Connector/UI** transfer to the **API Connector**: <listOfRequestedAttributes> and the <selectedCountty> with a freshly generated UUID string (that will be used as a unique identifier of the eIDAS authentication session).
- The **API Connector** takes as input what it receives and formulates as output an appropriate eIDAS SAML authentication request

User's Country

- The SAML authentication request is next transmitted to the near (proxy) **eIDAS Node** (GR in occurrence) and, then to the eIDAS infrastructure of the Country the User has dispatched the authentication process (ES in occurrence)
- The User is authenticated using the eIDAS flow (eIDAS Node – IdP – back to eIDAS Node); the output of the eIDAS authentication flow an eIDAS SAML authentication response
- The eIDAS Node of the Country the User has dispatched the process (ES in occurrence) forwards the eIDAS SAML authentication response to the eIDAS Node that has initiated the authentication request (GR in occurrence).

Back to SP's Country

- The **eIDAS Node** which receives the authentication response (GR in occurrence) forwards it to the **API Connector**

- The **API Connector** process and decrypts the authentication response and asks from the **API Connector/UI** to generate a JWT token that contains all the retrieved eIDAS attributes (plus the UUID of the session, generated in the beginning of the process). Essentially, the API Connector/UI creates the appropriate identification assertions for the eIDAS authenticated User, so that the SP service can easily consume them.
    - If an "error" occurred during authentication, such as a failure in the authentication of the User to the IdP, the JWT token contains the retrieved error codes from eIDAS.
- The JWT token is forwarded to **Service Provider's** Service Point[18], to the Service Point "success endpoint" if the authentication was successful, or to the Service Point "failure endpoint" in case an error occurred. This is again a simple redirection; no parameters are required (the JWT token is added as an "httpOnly" cookie). The Service Provider retrieves the JWT cookie from the http request and acts accordingly (verifies its signature, and authenticates the User or handles the error).
    - The User is redirected to the SP application/service, where she had submitted a "Login with eID_EU" request (or to an "Authentication Terminate" page, in the case of an authentication error).

[18] We define a Service Point as: a) the minimal SP configuration that redirects a user's login request to an API (Connector), when the user selects "register/login via eID_EU" and, b) the sum of endpoints where the API (Connector) will forward to SP application/service the authResponse data received from the eIDAS Node (auth success or failure report).

6 Annexes

6.1 Sample eIDAS SP SAML Tools Library Integration Code

metadata.jsp

```
<%@ page language="java" contentType="application/xml; charset=US-ASCII" pageEncoding="US-ASCII"%>
<%@ page import="eu.eidas.sp.metadata.GenerateMetadataAction" %>
<%
  out.clear();
  out.println(new GenerateMetadataAction().generateMetadata().trim());
%>
```

Login.jsp

```
<%@ page language="java" contentType="text/html; charset=US-ASCII" pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>eIDAS SP Login</title>
</head>
<%@ page import="eu.eidas.sp.SpEidasSamlTools" %>
<%@ page import="eu.eidas.sp.SpAuthenticationRequestData" %>
<%@ page import="java.util.ArrayList" %>

<body>
 <%
 String nodeUrl="";
 String samlToken = "";
 String citizenCountry = "GR";
 String serviceProviderCountry = "GR";
```

```
ArrayList<String> pal = new ArrayList<String>();
pal.add("http://eidas.europa.eu/attributes/naturalperson/CurrentFamilyName");
pal.add("http://eidas.europa.eu/attributes/naturalperson/CurrentGivenName");
pal.add("http://eidas.europa.eu/attributes/naturalperson/DateOfBirth");
pal.add("http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier");
SpAuthenticationRequestData data = SpEidasSamlTools.generateEIDASRequest(pal, citizenCountry, serviceProviderCountry, 2);
samlToken = data.getSaml();
nodeUrl = SpEidasSamlTools.getNodeUrl();
out.println("Request ID: "+data.getID());
%>

<form name="redirectForm" method="post" action="<%=nodeUrl%>">
    <input type="hidden" name="SAMLRequest" value="<%=samlToken%>" />
    <input type="hidden" id="country" name="country" value="<%=citizenCountry%>" />
    <input type="submit" value="Authenticate" name="Login" />
</form>

</body>
</html>
```

ReturnPage.jsp

```
<%@ page language="java" contentType="text/html; charset=US-ASCII" pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>eIDAS SP Return</title>
</head>
<%@ page import="eu.eidas.sp.SpEidasSamlTools" %>
<%@ page import="eu.eidas.sp.SpAuthenticationResponseData" %>
<%@ page import="java.util.ArrayList" %>

<body>
<%
out.clear();
SpAuthenticationResponseData data =
```

```
SpEidasSamlTools.processResponse(request.getParameter("SAMLResponse"),
request.getRemoteHost());
ArrayList<String []> pal = data.getAttributes();
out.println("Reponse ID: "+data.getID());
out.println("ReponseToID: "+data.getResponseToID());
out.println("Status Code: "+data.getStatusCode());
out.println("Status Message: "+data.getStatusMessage());
for (int i = 0; i < pal.size(); i++) {
        String[] t = pal.get(i);
        out.print("Attribute Name: "+t[0]);
        out.print("Mandatory: "+t[1]);
        out.print("Attribute Value: "+t[2]);
}
out.println("Response XML: "+data.getResponseXML());
%>

</body>
</html>
```

6.2 ISS 2.0 JSON API

6.2.1 Attribute List Retrieval API

The most important part when interfacing with the ISS 2.0 are the two API calls that are used by the ISS 2.0 and the SP in order to exchange information securely (out-of-band communication). As we mentioned earlier, the ISS 2.0 has an open design and can be extended to support custom SP needs. The current version of the ISS 2.0 supports JSON formatted communication.

The first step for the ISS 2.0 is to retrieve the list of Attributes. So the ISS 2.0 will communicate with a URL provided by the SP presenting the token. If everything is correct (i.e. valid token) the data returned by the URL must be in the following format:

```
{
 "status":"OK",
 "list":{
   "attribute name ":{
     "value":attribute value or null,
     "complex":0 or 1 depending on the value above,
     "required":0 if optional or 1 if mandatory
   },
...
```

```
  }
}
```

Each field is described in the table below. Please note that the list may contain one or more attributes and in some cases the value field may contain information, but in most cases it will be *null*.

Field Name Details:

status: Informs us if the outcome of the request was successful (OK literal) or not (NOT literal)

list: Contains one or more Attributes attribute name The attribute name is the name that identifies this attribute.

value: Contains the attribute value that is a string or null. The attribute value may be normal or complex. In the normal case it contains just one or a list of values separated by comma. So we may have: <value> or we may have: <value #1>,<value #2>...In the complex case we have a single or multiple key-value pairs. So we may have: <key>=<value> or we may have: <key# 1>=<value#1>,<key #2>=<value #2>... The key-value pairs are separated with the equals sign (=).

complex: In order to identify the contents of the value above, we set this variable to 1 if the value is complex and to 0 in any other case.

required: Identifies if the attribute is required in order to provide access to the SP service or not (0 if it is optional and 1 if mandatory).

For clarity we also provide a sample output that the DA Attribute List Retrieval API must return:

```
{
  "status":"OK",
    "list":{
      "http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier":{
      "value":null,
      "complex":"0",
      "required":"1"
    },
...
  }
}
```

6.2.2 Attribute Values Storage API

Once the ISS 2.0 has received the populated Attribute List it must deliver these values to the SP. This is performed via JSON again. The only difference is that the ISS 2.0 is not retrieving data from the SP, but sending data. Using the HTTP POST method the SS send a JSON object with the following format:

```
{
 "attribute name ":{
   "value":attribute value or null,
   "complex":0 or 1 depending on the value above,
   "required":0 if optional or 1 if mandatory
 },
...
}
```

As we can see this object is identical with the list object we described in the previous section. The SP will respond with a JSON message stating if the save procedure was completed successful or not:

```
{
 "status":"OK"
}
```

If the response is OK, then the SS can redirect the user back to the SP using the appropriate point.