SAML and eIDAS

Laboratory for the class "Cybersecurity" (01UDR)
Politecnico di Torino – AA 2021/22
Prof. Antonio Lioy

prepared by:
Diana Berbecaru (diana.berbecaru@polito.it)
Andrea Atzeni (andrea.atzeni@polito.it)

v. 2.0 (17/11/2021)

Contents

1 Purpose of this laboratory

In this laboratory, you will perform exercises aimed to experiment with SAML and eIDAS.

To perform the proposed exercises, you will need to use a PC with connection to Internet and a browser, such as Google Chrome or Mozilla Firefox.

To analyse the SAML message content, you will need to install in your browser some add-ons that capture and visualise SAML messages embedded in HTTP.

For example, you can use the `SAML-tracer` add-on, which is available at the following links (respectively for Firefox and Chrome):

`https://addons.mozilla.org/it/firefox/addon/saml-tracer/`

`https://chrome.google.com/webstore/detail/saml-tracer/mpdajninpobndbfcldcmbpnnbhibjmch`

Once you have installed the SAML-tracer add-on, you will see a dedicated icon in upright position of the tool bar, as shown in Fig. 1.

If you click on the icon, you will see a window as shown in Fig. 2.

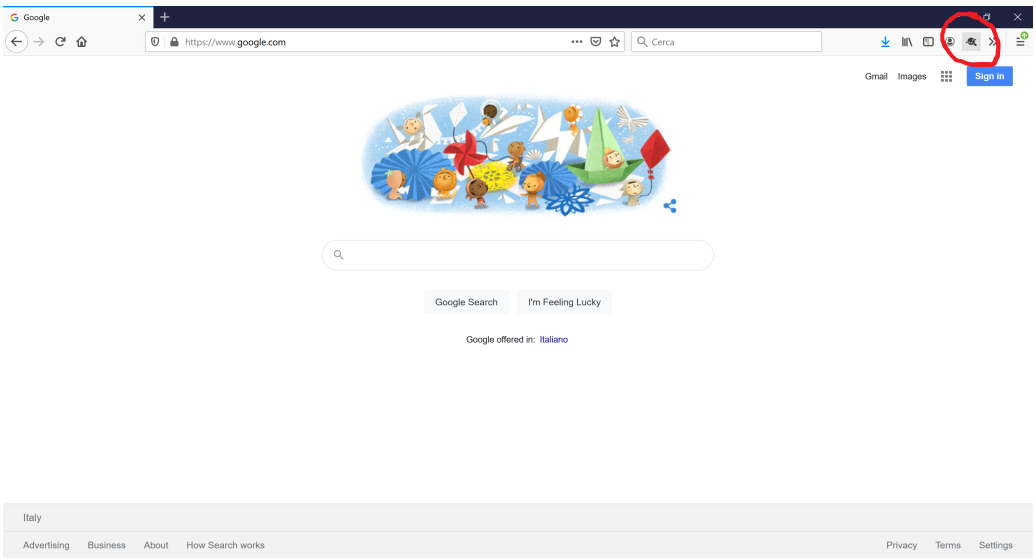

Figure 1: SAML-tracer add on icon in Firefox browser toolbar.

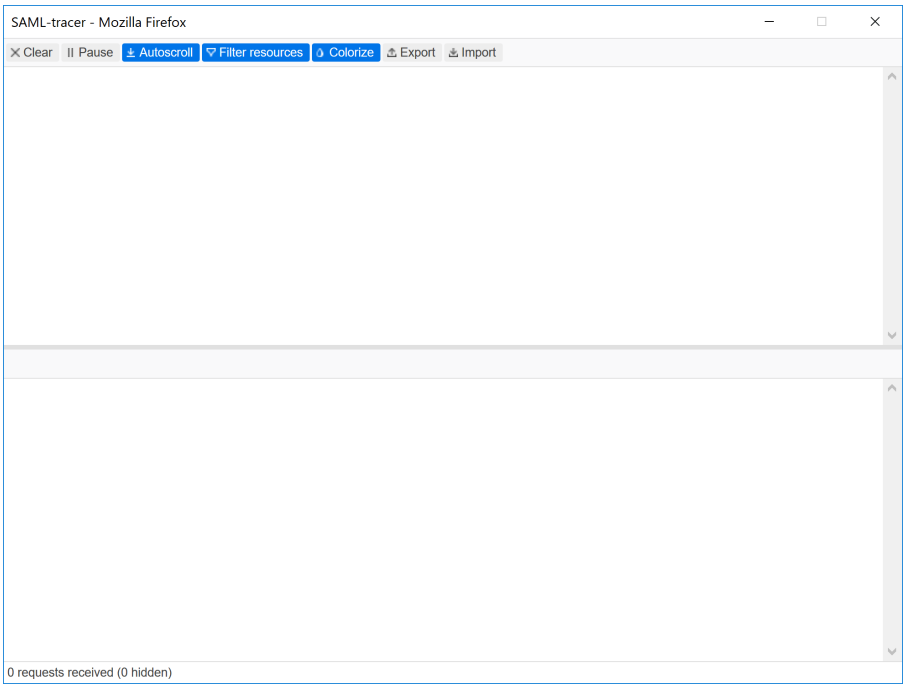

Figure 2: SAML-tracer window.

2 SAML

To experiment with SAML messages, we will start with the Login service at our university, which actually uses a SAML-based solution to authenticate students and academic staff to provide them access to the services.

Start from `https://www.polito.it`, start the `SAML-tracer` add-on in the browser, and click on the "Login" button in the page (the button is located in the up right area of the page).

In the Login page, perform the login with the solution that you normally use, e.g. with your username and password, or with a digital certificate.

Next, go to the `SAML-tracer` window and analyse the content of the messages you have just captured. You should see two orange "SAML" labels indicating that your browser has conveyed two SAML messages (a Request and a Response). The SAML request is shown in Fig. 3, while the SAML response is shown in Fig. 4.

How are the SAML Request and Response messages related to each other (which field indicates that the re-

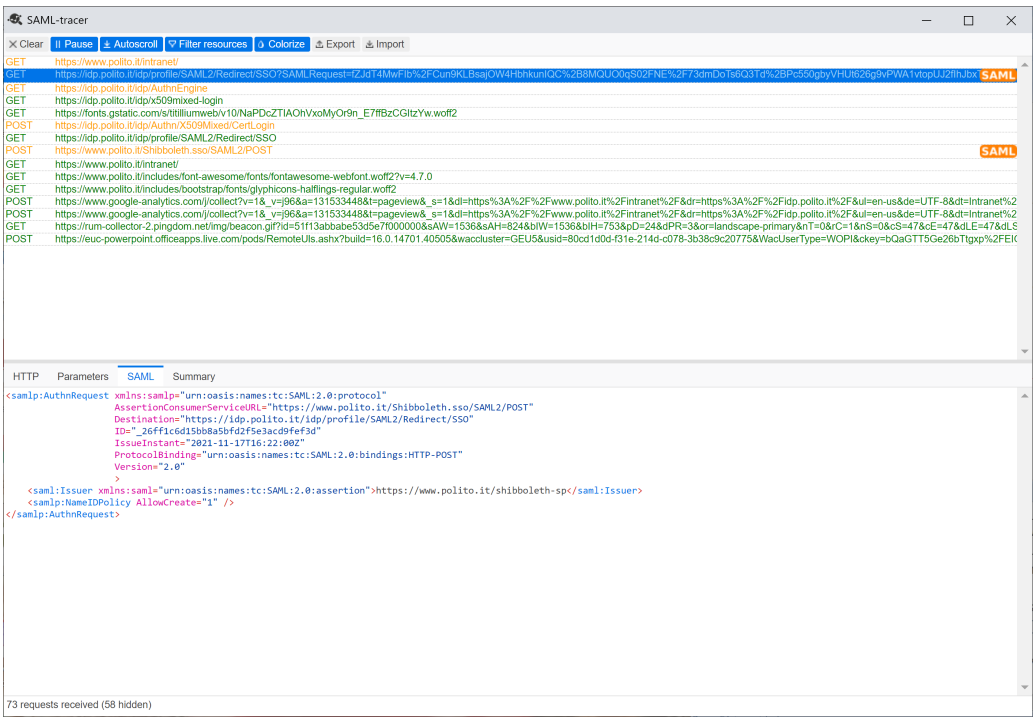

Figure 3: SAML Request message generated when performing Login at Politecnico di Torino.

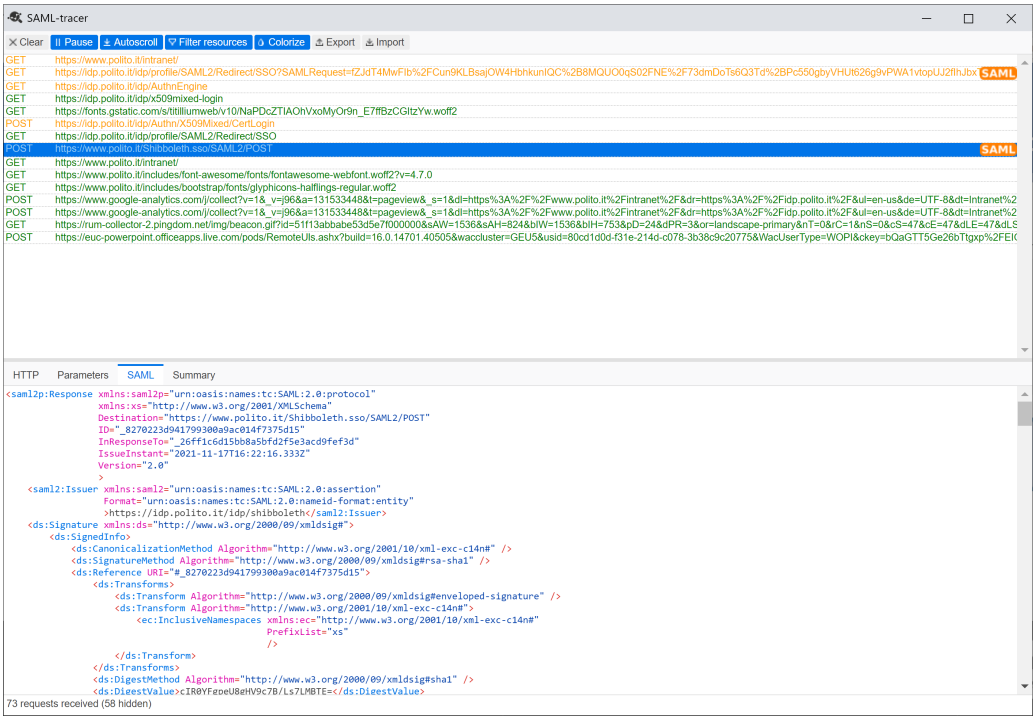

Figure 4: SAML Response message generated when performing Login at Politecnico di Torino.

sponse corresponds to a particular request)?

→

Check out the content of the SAML Response message and answer the following questions:

Which certificate is used (by the IdP) for signing the SAML Response?

→

To which URL is sent the SAML Response? Which field in the SAML Request has been used by the SP to indicate this URL (to the IdP)?

→

3 SPID

The Sistema Pubblico di Identità Digitale (SPID) addresses e-ID interoperability at the national level. It is composed by a set of trusted private and public services that can handle authentication of Italian citizens and companies for the public administration. SPID credentials are required to access public services, simplifying the interaction between entities and increasing security of the user authentication. Various credentials can be used, ranging from traditional ones based on smart-cards (e.g. the citizens' service card, in short CNS) to modern systems (e.g. one-time password generators, implemented as smartphone applications or via a hardware device). SPID components interact with each other using the SAML 2.0 language, for whom a specific profile has been defined.

3.1 SPID exchange

In the supporting files for this lab, you can find four SAML packets belonging to a full round of authentication with SPID (captured with SAML tracer as shown in Fig. 5).

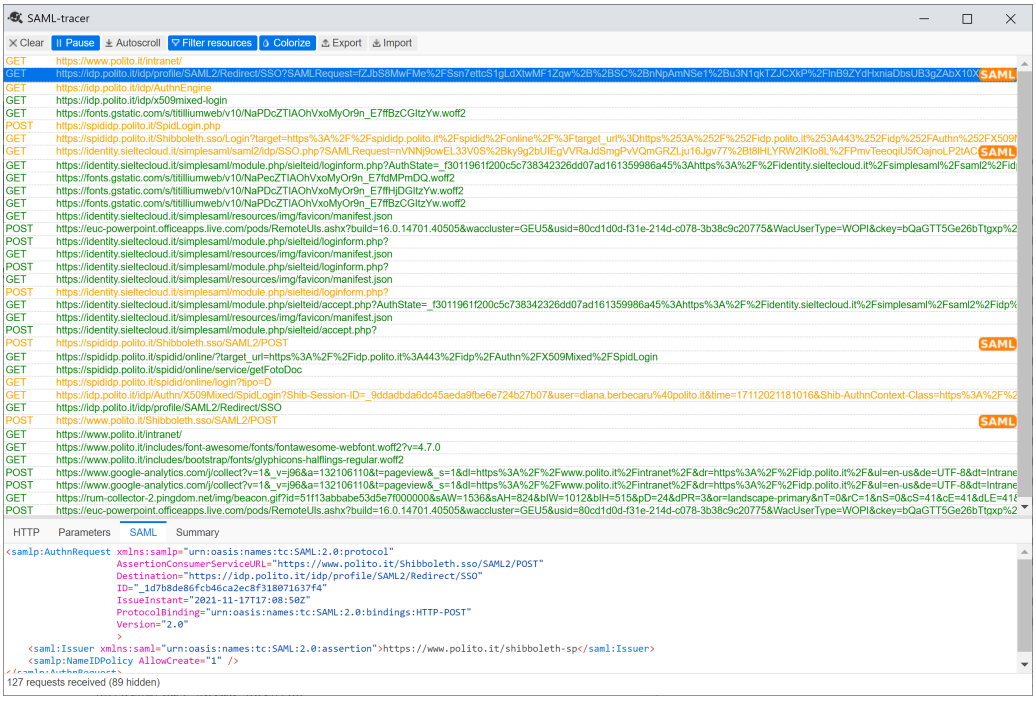

Figure 5: Capture of authentication with SPID on Politecnico di Torino portal.

By examining the packets answer to the following points:

What is the chronological sequence of the packets?

→

What is the protocol version?

→

What is the adopted binding?

→

Identify the relying party (from where the workflow has started).

→

Identify the involved identity providers.

→

What is the validity period of the SAML answers?

→

Who is the subject of the SAML-C certificates?

→

Where do you find information about the user who has been authenticated?

→

What do you think is the “high-level” process?

→

Do you think the involvment of SPID improves the security?

→

The authentication is finally successful?

→

3.2 SPID student authentication (optional)

If you have a SPID digital identity, try to authenticate to "portale della didattica" with SPID and identify the SAML packets exchanged by your browser. Do you notice any particular difference with the packets exchanged in the previous point?

→

4 eIDAS

4.1 eIDAS messages

In this exercise you will experiment with the messages exchanged by the eIDAS protocol, which are SAML-based messages extended to support the attributes defined in eIDAS, such as the ones for the natural person (FirstName, FamilyName, DateOfBirth, PersonIdentifier, ...).

We suppose to start from a so-called `Demo SP`, which will redirect the user to a so-called `Demo IdP` for authentication. Since we are using the eIDAS infrastructure (in test environment), the messages are redirected from the Demo SP to the Demo IdP via two additional elements: the `eIDAS Connector` and the `eIDAS Proxy`.

Start from the link: (`http://demo-sp-test-eid4u.polito.it/`. You should see the window shown in Fig. 6.

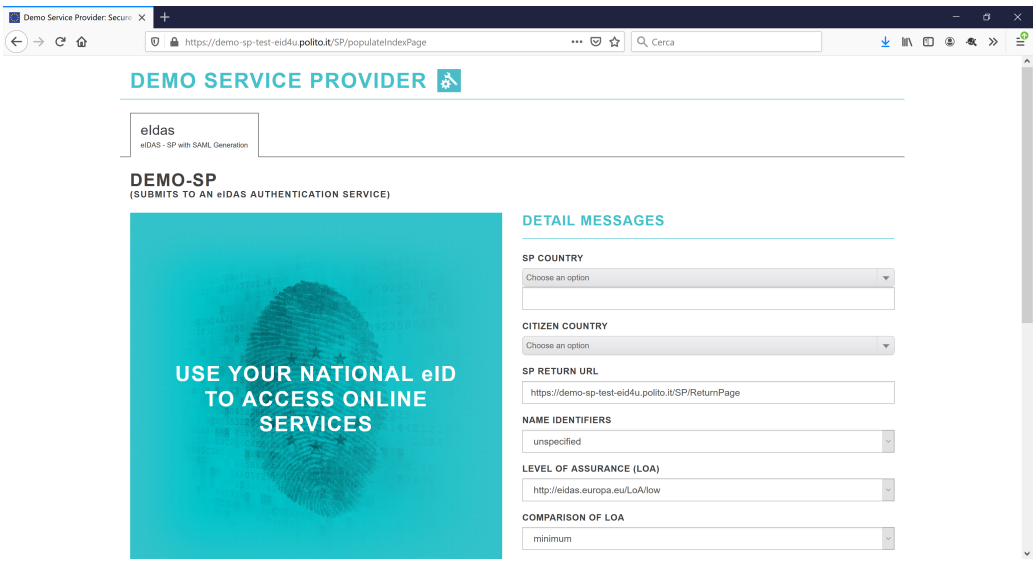

Figure 6: Initial page of the eIDAS-enabled Demo Service Provider (SP).

Select “IT” for the “SP COUNTRY”, then “IT” for the “CITIZEN COUNTRY”. In the part “REQUESTED CORE ATTRIBUTES”, select “DO NOT REQUEST”. Next, click on “SHOW” close to “NATURAL PERSON ATTRIBUTES”. You will see a list of attributes. Select as “MANDATORY” the 4 ones indicated with a (*), that is CurrentFamilyName, CurrentGivenName, DateOfBirth, PersonIdentifier.

Start the SAML-tracer, by clicking on the dedicated icon in the tool bar.

Then, in the Demo-SP page, press “SUBMIT”. You should see a window as illustrated in Fig. 7. In this page you can see the SAML Request in eIDAS format that has been generated.

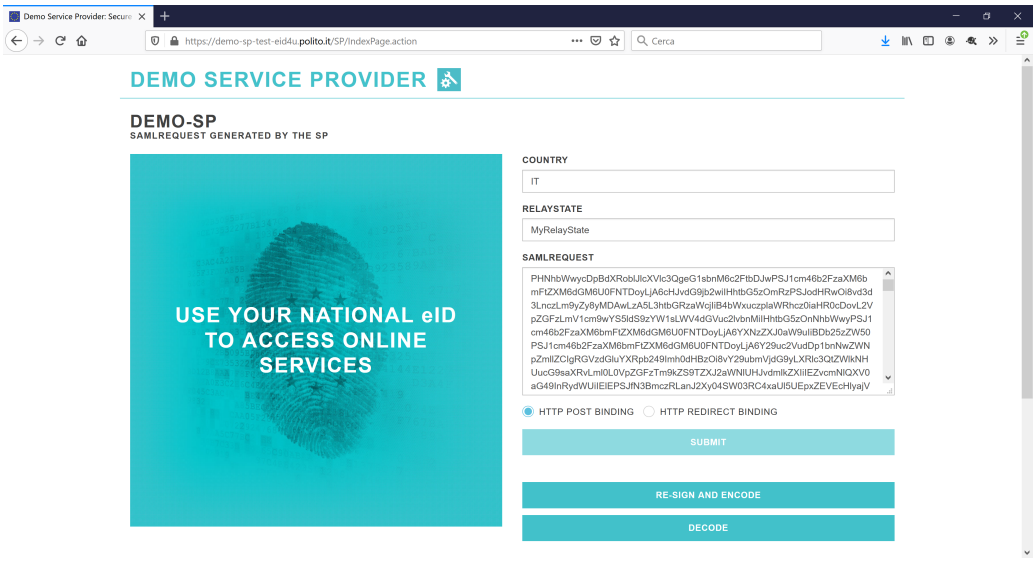

Figure 7: SAML Request (in eIDAS format) viewed in the Demo SP.

Next, press “SUBMIT”. You should see a window as shown in Fig. 8. At the next step you should see a window as shown in Fig. 9.

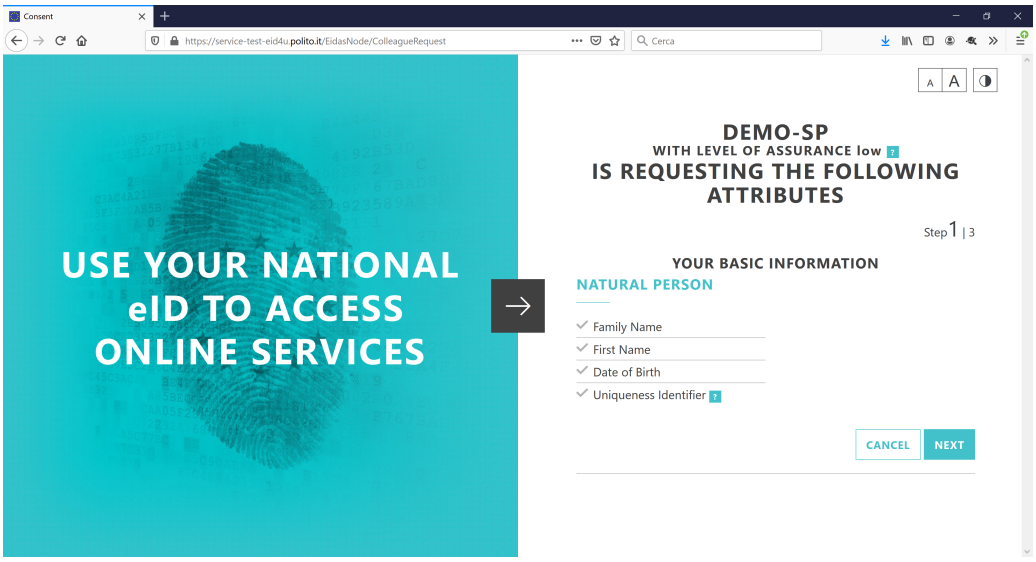

Figure 8: eIDAS attributes requested.

Then, click “NEXT”. At this point you land on the authentication page, as shown in Fig. 10. Click on “Entra con SPID”, then click on the button “Torsec - Polito SPID Demo IdP” in the page as shown in Fig. 11.

Use the following authentication credentials:

- the username: `test`

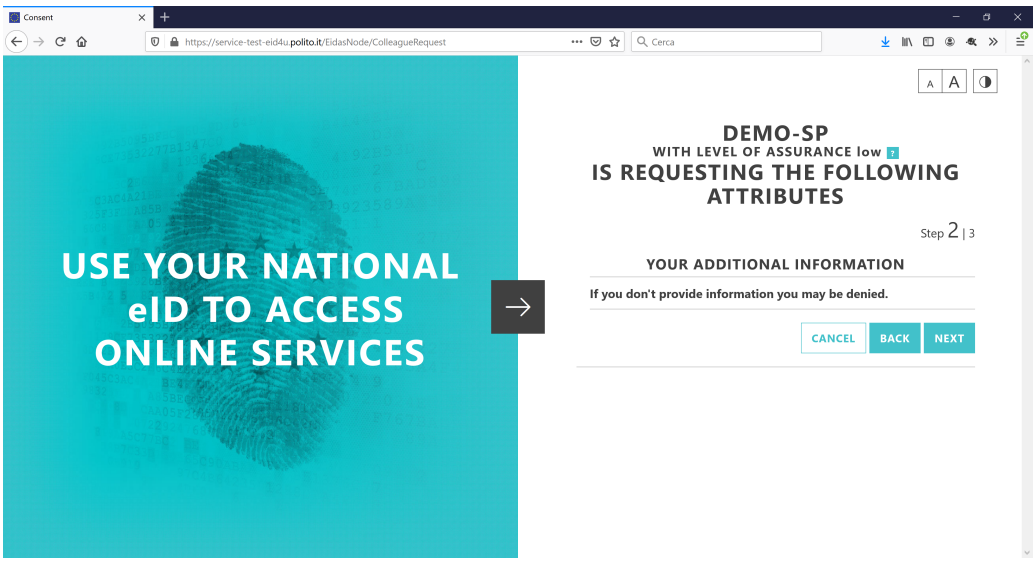

Figure 9: User consent page for the eIDAS attributes requested.

- the password: `test`

You should see a page as shown in Fig. 12. In this page you see already the SAML Response that has been generated, but we will analyze it better with the SAML-tracer later on. Then click "Invia". You should see a page with the valued attributes as shown in Fig. 13. Next, click on "SUBMIT". You will see a page as shown in Fig. 14, where you can see (already) the SAML Response (encoded in Base64), the encrypted response and the decrypted SAML Assertion. Then click "SUBMIT, and you should see the last page (as shown in Fig. 15) illustrating the valued attributes.

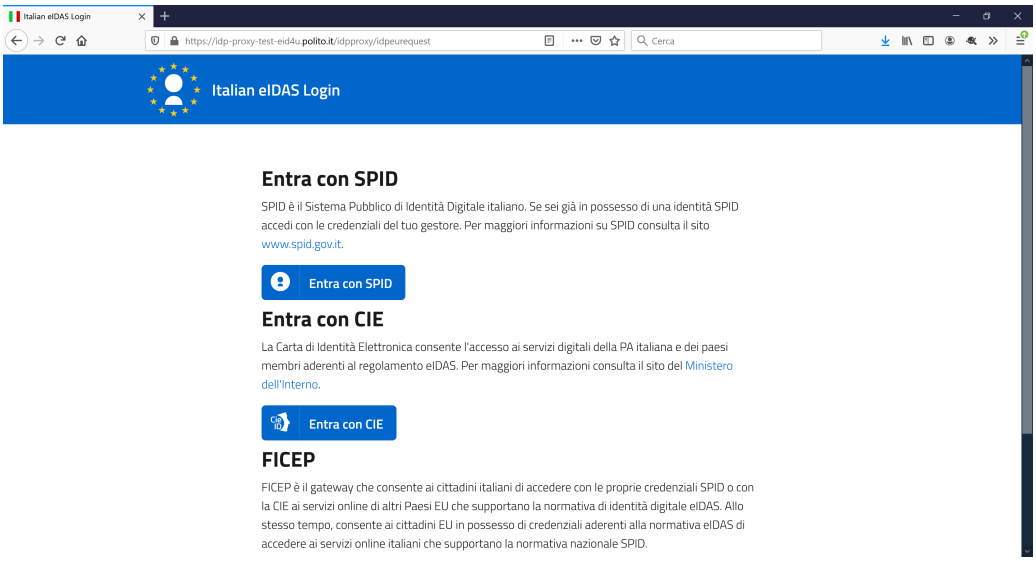

Figure 10: Authentication page allowing the selection of the Demo IdP.

At this point, we will start to analyze all the SAML messages captured in the above workflow with the SAML-tracer.

If you check the `SAML-tracer` window, you should observe the SAML messages that have been captured by the browser while executing the above steps (as shown in Fig. 16).

If you select the first row with an orange "SAML", then click on the "SAML" tab (in the lower part of the window): you can see the details of the SAML Request sent by the Demo SP to the element called eIDAS Connector (at the url `https://connector-test-eid4u.polito.it/EidasNode/ServiceProvider`) as shown in Fig. 17.

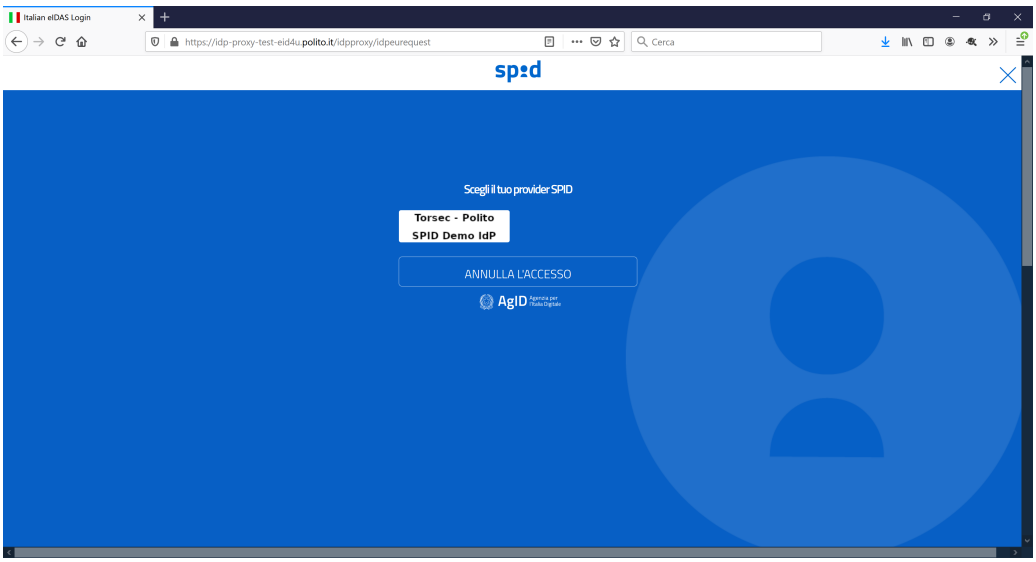

Figure 11: Authentication with SPID (in Demo IdP).

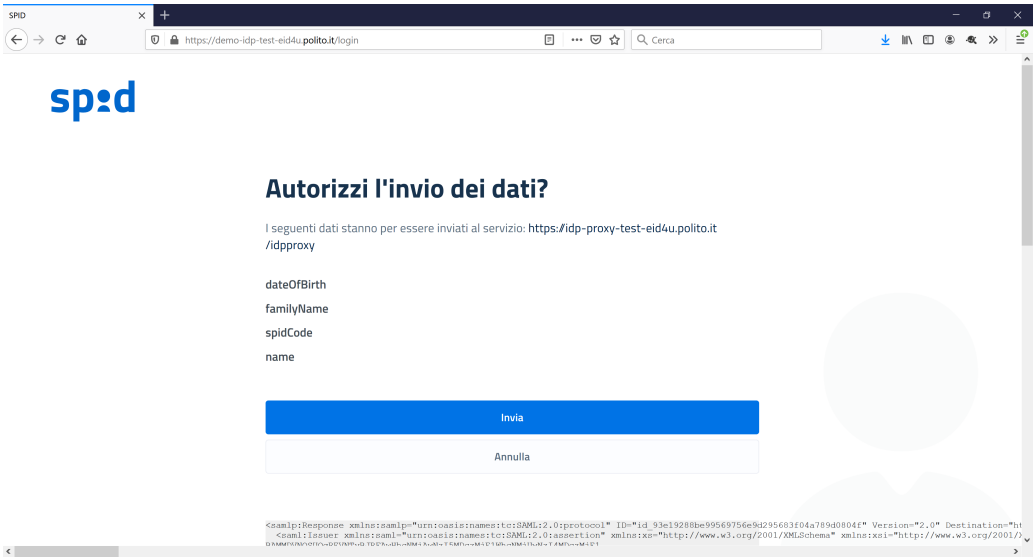

Figure 12: User consent on the valued eIDAS attributes.

Respond at the question:

Where is the indication of the URL (of the eIDAS Connector) where the SAML Request will be sent?

→

Where are placed in the SAML Request the (natural person) attributes requested?

→

Which algorithm has been used to digitally sign the SAML Request?

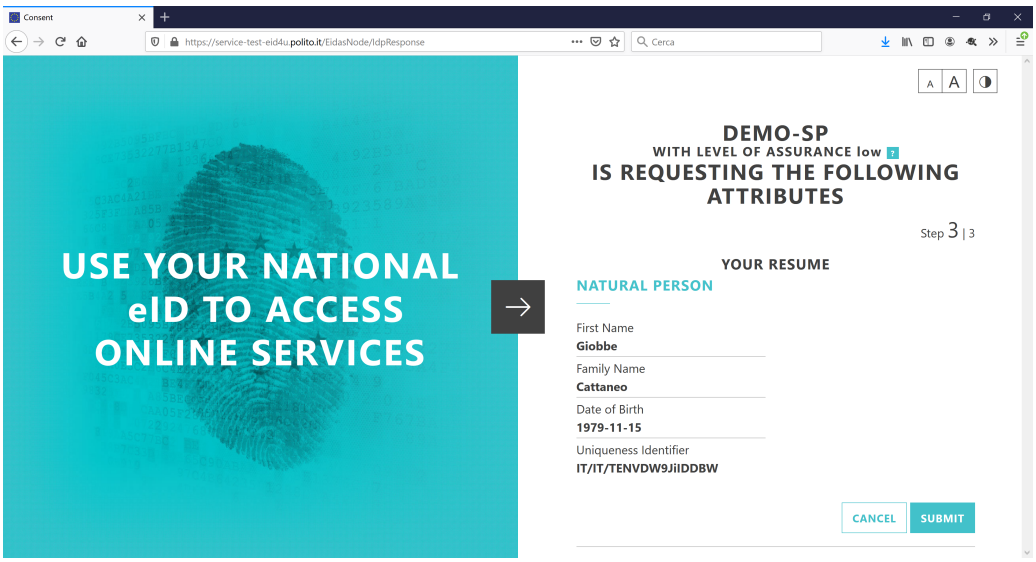

Figure 13: Valued eIDAS attributes.

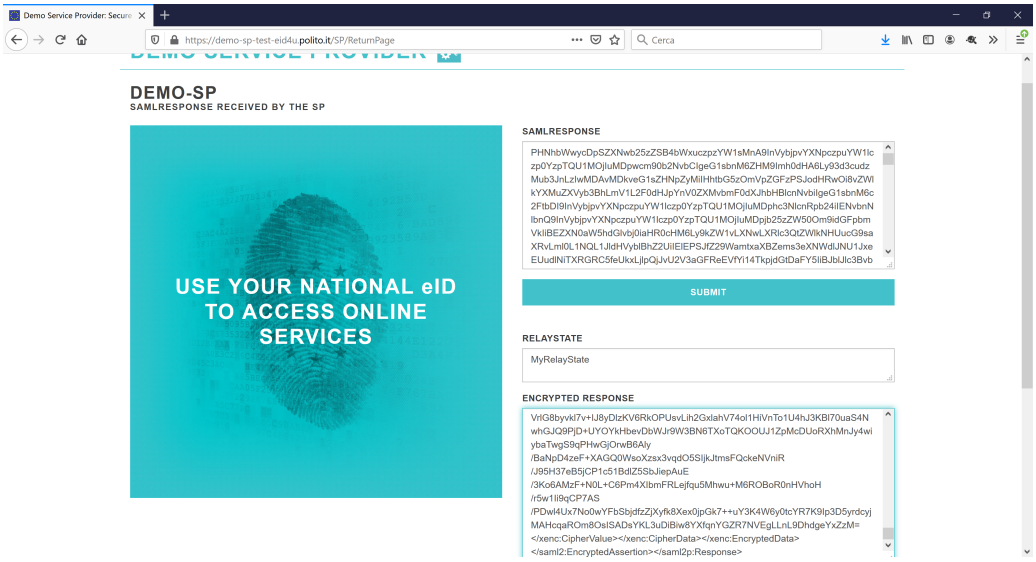

Figure 14: SAML Response (in eIDAS format) viewed in the Demo SP.

→

If you select the last row with an orange "SAML", the click on the SAML tab (in the lower part of the window): you can see the details of the SAML Response sent by the eIDAS Connector to the Demo SP, as shown in Fig. 18.

The SAML Response from the eIDAS Connector to the demo SP conveys an `<saml2:EncryptedAssertion>` element. In this element, there is an `<ds:X509Certificate>`.

Respond to the question: In your opinion, for what purpose is used the above certificate?

→

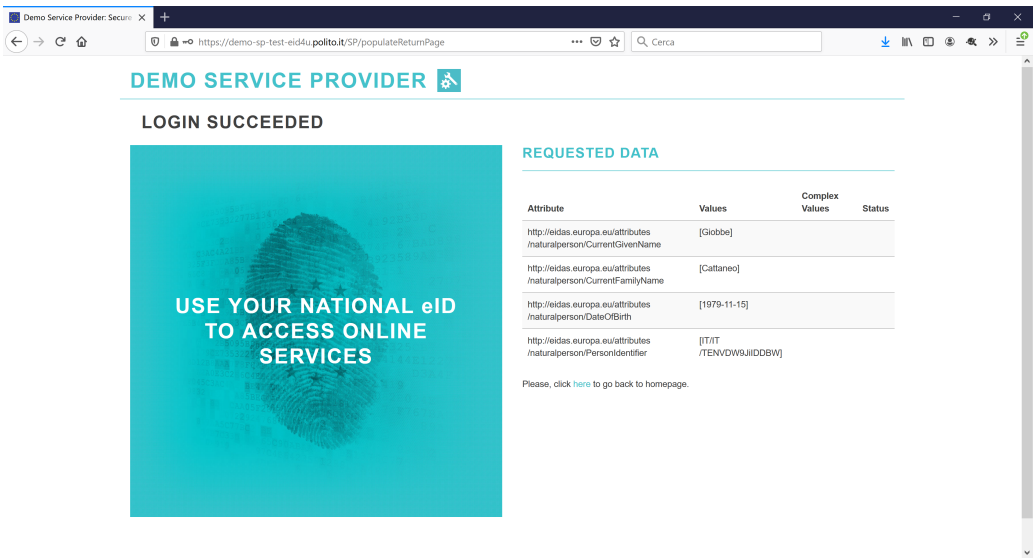

Figure 15: Final page of eIDAS-enabled Demo SP.

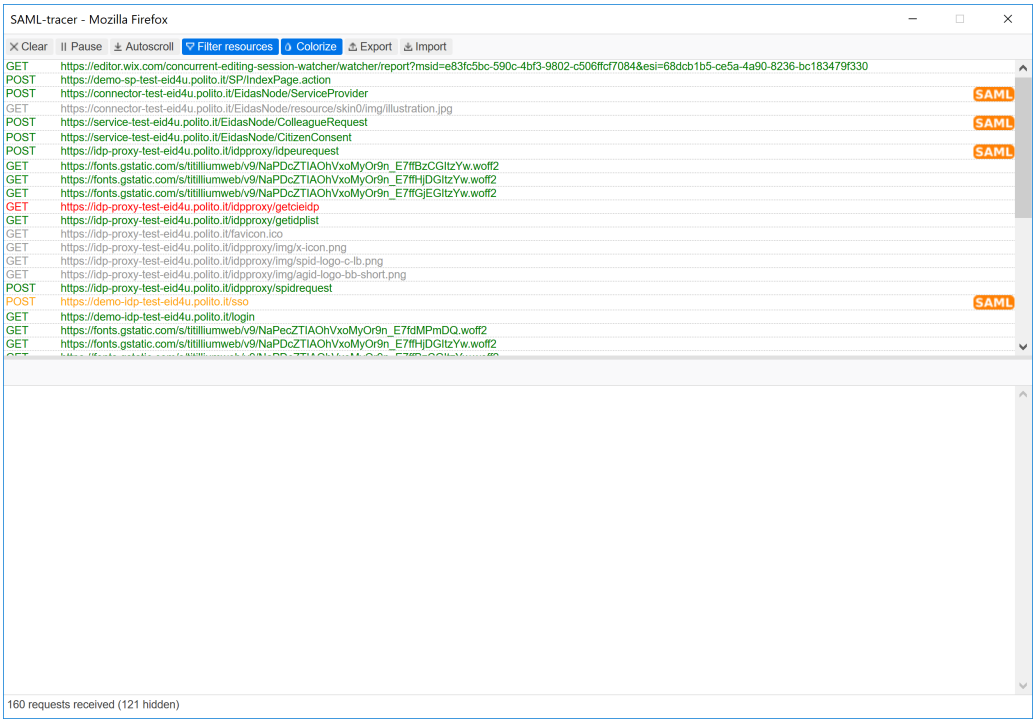

Figure 16: SAML messages captured in the SAML-tracer.

4.2 eIDAS (SAML) Metadata

To securely interoperate, all the actors in the above workflow (Demo SP, eIDAS Connector, eIDAS Proxy and the demo IdP) must share SAML metadata.

For simplicity, we will analyze only the SAML metadata for the communication between the demo SP and the eIDAS Connector, where the demo SP acts as an SP (in federation) and the eIDAS Connector acts an IdP (in a federation). Note however that SAML metadata must exist also for the communication between the eIDAS Connector and the eIDAS Proxy, and for the communication between the eIDAS Proxy and the Demo IdP.

The SAML Metadata of the Demo SP is available at the URL:

https://demo-sp-test-eid4u.polito.it/SP/metadata

The SAML Metadata of the eIDAS Connector necessary for the communication with the Demo SP is available

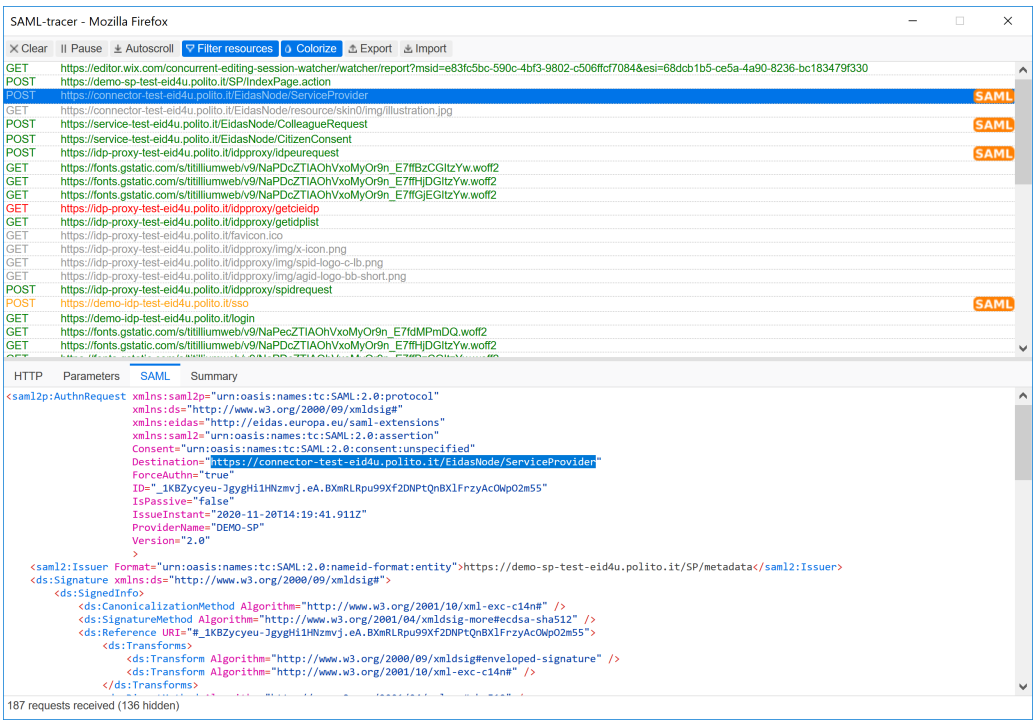

Figure 17: SAML Request captured in the SAML-tracer, sent from the Demo SP to the eIDAS Connector.

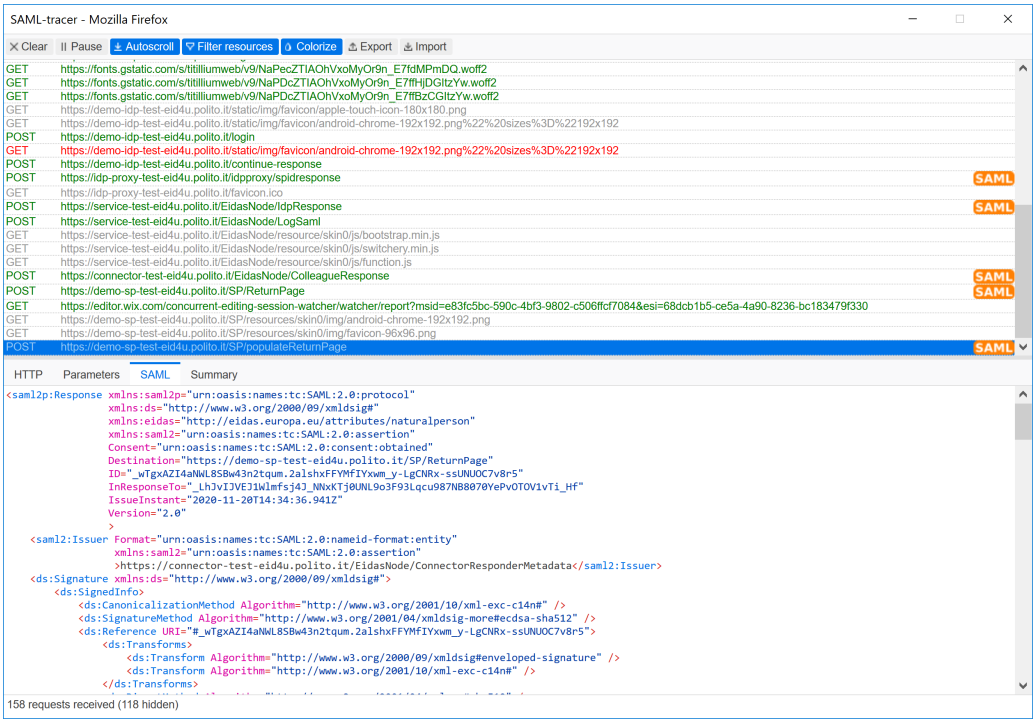

Figure 18: SAML Response captured in the SAML-tracer, sent from the eIDAS Connector to the Demo SP.

at the URL:

https://connector-test-eid4u.polito.it/EidasNode/ConnectorResponderMetadata.

We will analyze next which part(s) of these two SAML Metadata files are used:

1. when the Demo SP sends the SAML Request to the eIDAS Connector.
2. when the eIDAS Connector sends back the SAML Response to the demo SP.

Open the SAML Metadata of the Demo SP and analyze its content:

1. You should observe that the file starts with an `EntityDescriptor` element, having the entityID="https://demo-sp-test-eid4u.polito.it/SP/metadata". The entityID is like an identifier for the demo SP. In fact, you can see it in the <saml2:Issuer> element of the SAML Request, which is sent from the Demo SP to the eIDAS Connector. Note also that there is a `validUntil`="..."' which indicates the time limit until the SAML metadata is valid.

2. Next, you should see a <ds:Signature> element. This indicates that the SAML Metadata file (of the Demo SP) is digitally signed. Moreover, it indicates also which X.509 certificate needs to be used to verify the signature on the SAML Metadata file. The certificate is found in <ds:X509Certificate> element. Note that the eIDAS Connector must obtain this certificate through an OOB manner and configure it in a (local) trust store.

3. In the <md:SPSSODescriptor> you should note the element <md:KeyDescriptor use="signing">. This element contains an X.509 certificate which must be used by the eIDAS Connector to verify the signature on the SAML Requests received from the demo SP.

In fact, if you check (in the SAML-tracer) the SAML Request sent by the demo SP to the eIDAS Connector (that is the first raw tagged with an orange "SAML"), you should note that there is a <ds:X509Certificate> element which has the same value with the one indicated in the <ds:X509Certificate> in the <md:KeyDescriptor use="signing"> of the Demo-SP SAML metadata file, i.e "MIICVTCCAdqgAwIB...". Note that the certificate is Base 64 encoded.

4. In the <md:SPSSODescriptor> you should note the element <md:KeyDescriptor use="encryption">. This element contains an X.509certificate which must be used by the eIDAS Connector to protect the assertion sent in the SAML Response to the demo SP.

In fact, if you check in the `SAML-tracer` the SAML Response sent by the eIDAS Connector to the demo SP (that is the last raw tagged with an orange "SAML"), you should see an <saml2:EncryptedAssertion> element. In this element, the <ds:X509Certificate> contains a certificate whose value is equal to the one in the element <md:KeyDescriptor use="encryption"> of the demo SP SAML Metadata, i.e. "MIIDpTCCAo2g..". Note that the certificate is Base 64 encoded.

Respond to the following questions.

Which algorithms are supported by the Demo SP for the decryption of the encryption assertion sent to him by the eIDAS Connector? Which element in the Demo SP's SAML Metadata contains such indication?

→

Which encryption algorithms has been actually used by the eIDAS Connector for the protection of the eIDAS Response sent to the demo-SP?

→

5. Note also the `AssertionConsumerService` element:

```
<md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="https://demo-sp-test-eid4u.polito.it/SP/ReturnPage">
```

What is its meaning?

→

Now open the SAML Metadata file of the eIDAS Connector (at the url mentioned above) and respond to the questions:

What is meaning of the X.509 certificate in the <md:KeyDescriptor use="signing”>? Indicate in which message (captured with SAML-tracer you find this certificate.

→

What is the meaning of the X.509 certificate in the <md:KeyDescriptor use="encryption”>? Indicate in which message (captured with SAML-tracer you find this certificate.

→

Refresh the web browser page in which you see the SAML Metadata of the Demo SP (or of the eIDAS Connector). Do you note any difference? Can you figure out why ?

→

5 OpenID Connect

OpenID Connect (OIDC) is a protocol built on top of OAuth2 to provide a complete solution for both authentication and authorization. It allows OAuth2 clients (usually, a service provider from end-user/resource owner perspective) to check the identity of an end-user entity thanks to an authentication performed on a third component, in order to authorize the OAuth2 consent. A nice security outcome, is that the OAuth2 client does not need to store or manage any password. In fact, the user delegates authority for the OAuth2 client application (i.e. application server from end user perspective) to access some protected resource on their behalf. Such a resource might be anything (e.g. Google profile information, Facebook friend list). Delegating access to this resource gives the client application the means of verifying the end-user's identity without ever seeing his credentials

The OAuth2 standard allows for a number of different flows (i.e. message exchange schemas). The most used, and suggested for its security properties, is the AuthZ Code Grant one. In this flow the response to the Relaying party is not in the form of the actual OAuth2 ticket (i.e., the credentials in OAuth2 terminology), but in the form of a “code”, not containing the user credentials but allowing the relying party to check if the end-user has been successfully authenticated. In fact, the presence of a OAuth2 value of type “tocken” or “code” allows to distinguish between the implicit flow (a simpler but more insecure one, which allow the end-user credential to flow among the actors, which should be always avoided) and the preferred Authz Code Grant flow. In the following scenario, the AuthZ code is delivered by an AuthZ Server. The AuthZ server acts like a broker between the client and the resource owner. The server authenticates the resource owner and obtains authorization before redirecting it back to the client with the AuthZ code. As pointed out, the resource owner only authenticates versus the authorization server.

A brief description of this flow follows:

1. The flow starts with the client that builds the URI with the necessary parameters and redirects the user on the AuthZ Server dedicated endpoint. The user logs in and accepts/denies to give authorization to the client. Peculiar to OIDC scenario, a subset of the following OpenID Connect scopes is requested:

   - `openid`: it means that the client is making an OIDC request
   - `profile`: this requests access to the user's profile information
   - `email`: this requests access to the user's e-mail address
   - `address`: this requests access to the user's address information
   - `phone`: this requests access to the user's phone number

   the presence of them (in particular `openid` is an indication of an ongoing OIDC exchange

2. If the user accepts, the AuthZ Server redirects the user back to the client with an AuthZ code. OIDC defines optional parameters (e.g. `nonce`, `prompt`) for managing the authentication. The server check the parameters and in case of a valid login request, the user is presented with the usual consent screen. The user MAY authorizes the relying party (OAuth2 client) to authenticate it. In this case, he will be redirected back to the client application via the redirection endpoint, along with the authorization code.

3. the client (without user intervention) requests the access token by sending the token request with the AuthZ code to the AuthZ Server. The server can verify the correct relation of the AuthZ Code and the OIDC authentication request by checking the client_id . It builds up the `id_token` that is a JWT which contains private claims about the authenticated user (e.g. `name`) and other claims like, for example `iss` that represents the issuer and `sub` that represents a unique identifier of the authenticated user.

4. The AuthZ Server respond with the access token, as the property `access_token`). However, it will contain the additional token known as an `ID token` (as the property `id_token`). The client MUST decrypt the token by using the specified cryptographic operations contained in the JOSE Header, validate the JWT ID Token, and retrieve the claims (e.g. the user and the issuer. Finally, the user identity provided by the AuthZ Server is verified. Now the client can use the access token for accessing resources on a dedicated resource server.

5.1 Example of an OpenID Connect flow

In the following is described a simple example of the flow. The scenario is composed of a user (web) application listening on `http://localhost:9090` (Fig. 19), and an application server (also known as `relying party` and `OAuth2 client`) listening on `http://localhost:8080`.

In this example the goal is just to retrieve (only read) basic user information (name and email, that in the OAuth2 flow can be identified in the `scope` parameter) from the Google profile by using the OIDC on top of the Authorization Grant flow.

1. From the "Login" page (on `http://localhost:9090`) (Fig. 20), the user clicks on "Login With Google" and a first redirection starts with the requested parameters. The capture of the HTTP header is present in the `OIDC-1.http` file. Analising the capture, are you able to identify where the redirection will go, i.e. the application server endpoint?

   →

2. the application server creates the state parameter, takes the info from the configuration and redirects the user to the Google OAuth2 page. The capture of the HTTP header is present in the `OIDC-2.http` file. Are you able to identify the google OAuth2 page endpoint?

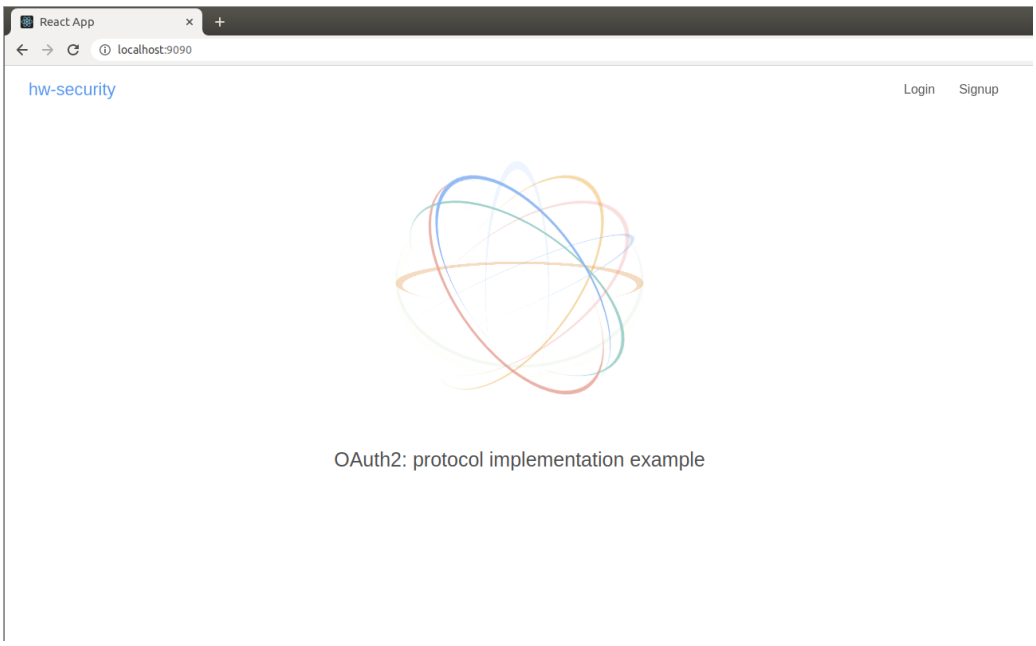

Figure 19: Starting page of the end-user app for the sample OIDC flow.

→

3. The user logs exploiting Google as AuthZ server (Fig. 21) and accepts or denies the AuthZs scopes. After the user acceptance, Google redirects it back on the app server with the AuthZ Code
4. the app server checks that the state is the same and prepares a HTTP GET to the Google Token endpoint (`OIDC-3.http`)
5. Google checks code and secrets, and gives a JWT access token to the app server
6. the app server sends an HTTP GET to the Google endpoint by inserting the `access_token` in the authorization header. The Google API server checks the token and prepares the response as JSON, ending the OAuth2 exchange (`OIDC-4.http`).
7. finally, the user app displays the information of the logged user (Fig. 20).

Answer to the following questions:

Analysing the first message (i.e. `OIDC-1.http`) can you identify the authorization server?

→

Analysing the second message, (i.e. `OIDC-2.http`) can you identify the type of OAuth2 adopted flow?

→

Can you identify the objects of the authorization requests?

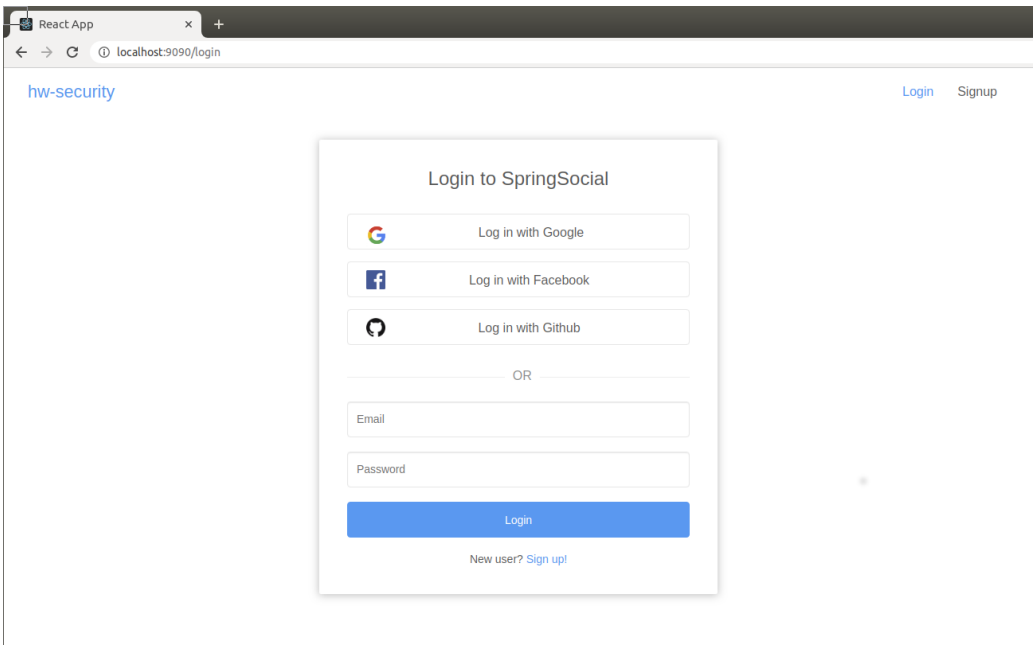

Figure 20: Login page of the end-user app for the sample OIDC flow.

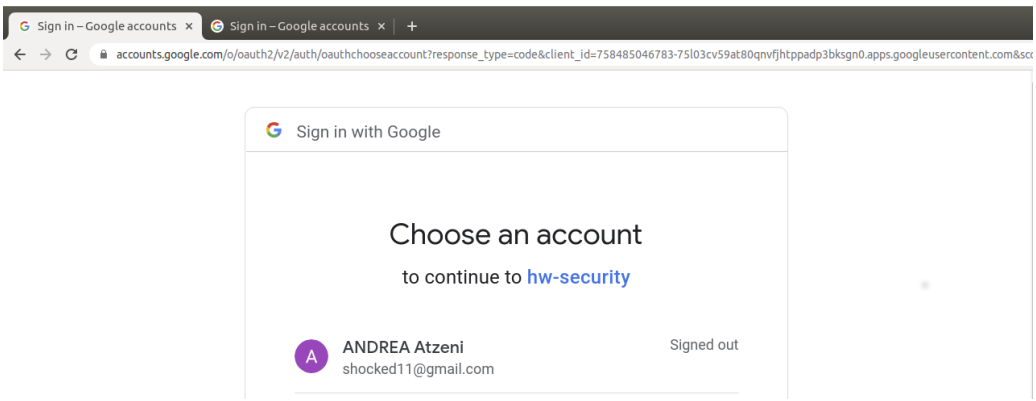

Figure 21: Google login page

→

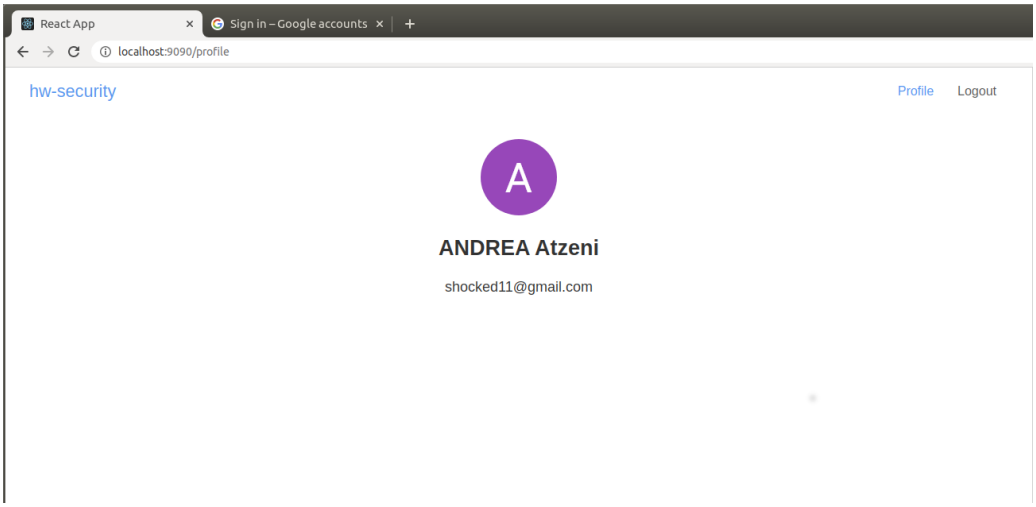

Figure 22: Data shown at the end of the sample OIDC flow.