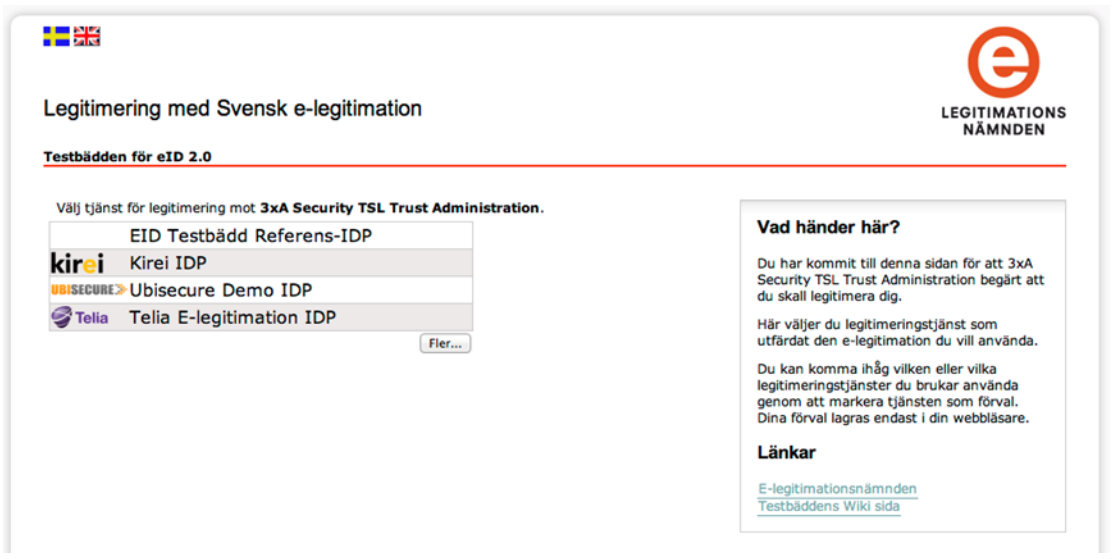
LEGITIMATIONS
NÄMNDEN

Discovery Service Infrastructure for "Test-bädden för EID 2.0"

Implementation guidelines

Version 0.70
2013-04-24



Legitimering med Svensk e-legitimation

Testbädden för eID 2.0

Välj tjänst för legitimering mot **3xA Security TSL Trust Administration**.

- EID Testbädd Referens-IDP
- Kirei IDP
- Ubisecure Demo IDP
- Telia E-legitimation IDP

Fler...

Vad händer här?

Du har kommit till denna sidan för att 3xA Security TSL Trust Administration begärt att du skall legitimera dig.

Här väljer du legitimeringstjänst som utfärdat den e-legitimation du vill använda.

Du kan komma ihåg vilken eller vilka legitimeringstjänster du brukar använda genom att markera tjänsten som förval. Dina förval lagras endast i din webbläsare.

Länkar

E-legitimationsnämnden
Testbäddens Wiki sida

This document describes the discovery service infrastructure for "testbädden for EID 2.0 and how to implement it at a Service Provider.

LEGITIMATIONS
NÄMNDEN

1 Overview

The present discovery service infrastructure can be utilized by service providers in “Testbädden for Eid 2.0” to handle users selection of the Identity Provider they want to use when authenticating (logging in) to the service.

This discovery service infrastructure supports two main implementation alternatives:

1. Discovery using a central Discovery Service.
2. Integration of the discovery UI into the Service Provider web page.

The advantage with using the central Discovery Service is that it is slightly easier than integrating the UI in a service webpage.

The main advantage with integrating discovery into the service webpage is that it avoids an extra webpage the user need to visit in order to log into the service.

These two modes of operation is demonstrated at the login page of the central signing service when logging into the test authority for signing (SP Myndigheten).

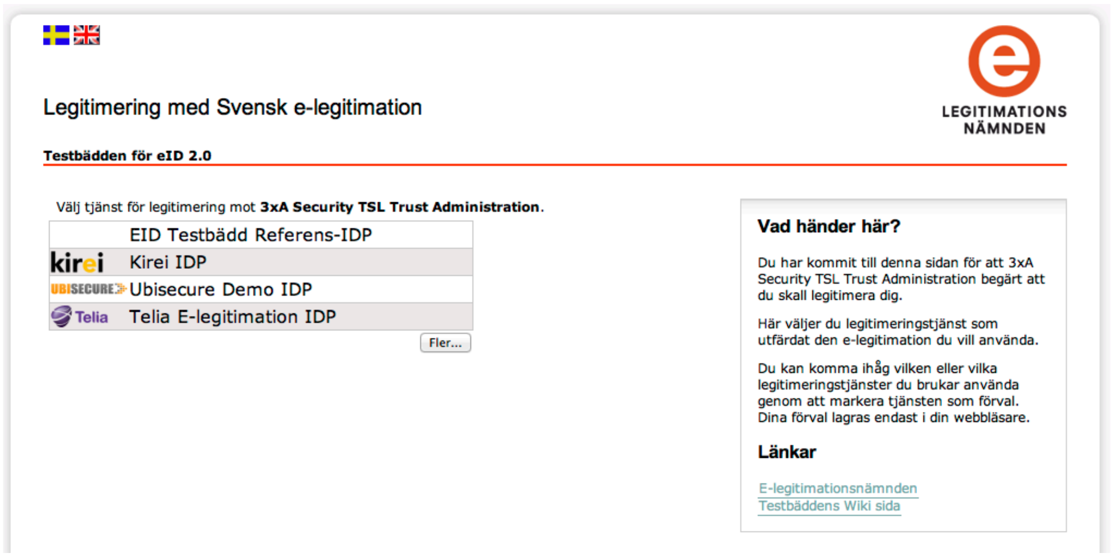
See: https://eid2cssp.3xasecurity.com/login/index.jsp

In the red banner to the right, there is a choice between central discovery (denoted “Central Anvisning“) or integrated discovery (denoted “Integrerad Anvisning)”

The simple difference is that using central discovery, the page just displays a “Login” button that takes you to central discovery. When selecting integrated discovery the page instead displays the discovery UI directly in login page.

Implementation instructions for the two alternatives are outlined below:

2 Implementation of Central Discovery Service


Legitimering med Svensk e-legitimation

Testbädden för eID 2.0

Välj tjänst för legitimering mot **3xA Security TSL Trust Administration**.

EID Testbädd Referens-IDP
Kirei IDP
Ubisecure Demo IDP
Telia E-legitimation IDP

Vad händer här?

Du har kommit till denna sidan för att 3xA Security TSL Trust Administration begärt att du skall legitimera dig.

Här väljer du legitimeringstjänst som utfärdat den e-legitimation du vill använda.

Du kan komma ihåg vilken eller vilka legitimeringstjänster du brukar använda genom att markera tjänsten som förval. Dina förval lagras endast i din webbläsare.

Länkar

E-legitimationsnämnden
Testbäddens Wiki sida

Implementation of integrated discovery only involves two steps:

1. Configure your SAML service provider software (here referred to as the authentication server) to use a discovery service.
2. Create a login button that points the user directly to the URL of the protected service (The service requiring login).

What then happens is that the user being pointed to the URL of the protected service will be stopped by the authentication server unless the user has an active ongoing session with the service. The user will then automatically be transferred to the Discovery Service to select an Identity Provider. Information about the selected Identity Provider will be returned to the authentication server who initiates an authentication request to the selected service.

If the authentication server is a Shibboleth SP implementation, the use of Discovery Service is configured by including the following element in the <Session> element of your <ApplicationDefaults> element (if using default settings) or of your <ApplicationOverride> element (if using an application override profile):

```
<SSO discoveryProtocol="SAMLDS"
       discoveryURL="https://eid2.3xasecurity.com/disco/idpdisco">
   SAML2 SAML1
</SSO>
```

The URL to the Discovery Service described here is (as the example suggests):

https://eid2.3xasecurity.com/disco/idpdisco

Once the authentication server is setup the step to include a link or a button for login is straightforward. The easiest way is to just provide a hyperlink to the target service.

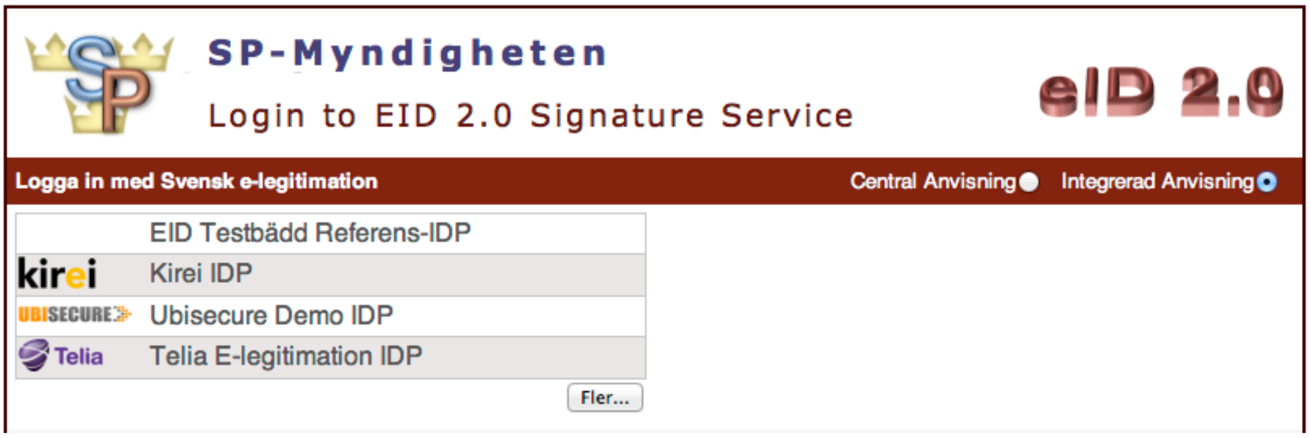
LEGITIMATIONS NÄMNDEN

But just for illustrative purpose and completeness, the following JavaScript example (based on jQuery) adds a login function to a html element with id=”loginElement” when the variable urlToProtectedService holds the URL to the target service[1]:

```
$('#loginElement').css('cursor','pointer').click(function(){
   window.location= urlToProtectedService;
});
```

[1] The (.css('cursor','pointer')) makes sure that mouse pointer has the form of a pointer when hovering over the element.

3 Implementation of Integrated Discovery



Implementation of integrated discovery is almost as easy as implementing central discovery. In a way it is easier since it in its simplest form doesn't involve any server configuration at all as integrated discovery can be fully implemented just in the login webpage sent to the user.

Implementing integrated discovery requires the following modifications to the login web page:

1. Import necessary JavaScripts and Stylesheets in the header of the page.
2. Add a container (e.g. a '<div id=discoUI></div>' to the page body where the UI components shall appear on the page.
3. Add a JavaScript to the page that executes on page load, which invokes the discovery JavaScript function ($.eid2Disco).
4. Add a login function that should be executed when the user has selected an Identity Provider, that initiates login with that identity provider.
5. Optional: Add an error function that can capture any errors from the discovery JavaScript.

Step 5 is really not necessary since errors only can be caused by misconfiguration, not by user error or failure to select an IdP. Failure to select an IdP only means that the login function is never called.

JavaScripts and Stylesheets

The required JavaScripts and stylesheets are:

Script or stylesheet	Location
jquery_eid2Disco.js	https://eid2.3xasecurity.com/disco/jquery_eid2Disco.js https://eid2.3xasecurity.com/disco/jquery-1.8.3.js
jquery-1.8.3.js	http://code.jquery.com/jquery-1.8.3.js or https://eid2.3xasecurity.com/disco/jquery-1.8.3.js
eid2Disco.css	https://eid2.3xasecurity.com/disco/eid2Disco.css

The discovery functions are provided by the jquery_eid2Disco.js script. This script is written as a jquery plugin, thus it also requires the jQuery JavaScript. It is not necessary to use version 1.8.3 of JQuery. The script has been tested with versions down to 1.4 without any problems. Version 1.8.3 is just the latest version at the time of this implementation.

The eid2Disco.css stylesheet defines the styles for the discovery UI.

LEGITIMATIONS
NÄMNDEN

Example implementation:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Discovery Test SP</title>
    <link rel="stylesheet" type="text/css" href="https://eid2.3xasecurity.com/disco/eid2Disco.css" />
    <script type="text/javascript" src="https://eid2.3xasecurity.com/disco/jquery-1.8.3.js"></script>
    <script type="text/javascript" src="https://eid2.3xasecurity.com/disco/jquery_eid2Disco.js"></script>
    <script type="text/javascript">
      var targetServie = "https://eid2cssp.3xasecurity.com/sign/";
      var loginUrl = "https://eid2cssp.3xasecurity.com/Shibboleth.sso/Login";
      $(document).ready(function() {
        /**
         * Initiates the eid2Disco UI
         *
         * Supported Parameters
         * storage: URL to storage service providing iframes for reading and setting local storage
         * discofeed: URL to json discovery data feed
         * storagefeed: URL for reading local storage data
         * discodiv: the id attribute value of the div element where the disco UI is placed
         * lang: the preferred UI language (Default "en")
         * jsonp: boolean default true. If set to true, all feed data is retrieved using JSONP javascript import
         * selectHeight: maximum height of select box. Need to be provided in the form of '{number}px'.
         * Default 300px
         * logoMaxHeight: default 20, specifies the max pixel height of Identity Provider logotypes
         * logoMaxWidth: default 70, specifies the max pixel height of Identity Provider logotypes
         * success: a function that is called when the choice of IdP is completed successfully
         * error: a function that is called then this disco service encounters an error
         */
        $.eid2Disco({
          discodiv:"discodiv",
          success : function(entityId){
            idpLogin(entityId);
          },
          error : function(message){
            alert(message);
          }
        });
      });

      function idpLogin(entityId){
        window.location = loginUrl
        + "?entityID=" + encodeURIComponent(entityId)
        +"&target="+encodeURIComponent(targetServie);
      }
    </script>
  </head>
  <body>
    <h1>Discovery Test SP</h1>
    <br />
    <div id="discodiv"></div>
  </body>
</html>
```

Notes about the example implementation

The discovery UI is invoked through the **$.eid2Disco(**parameter**)** function.
The parameters are provided as an object holding "name: value" pairs:

```
var parameter = {name1: value1, name2: value2}
```

This function only needs the parameter **discodiv** and **success**. All other parameters are optional (See further the comments in the example implementation).

The **discodiv** parameter must specify the name of the html container where the discovery UI is to be placed.
The **success** parameter must provide a function to be called when the user has selected an Identity Provider. The **entityID** of the selected Identity Provider is returned as a string parameter to this function.

The login function **idpLogin(entityId)** in this example is called on success and executes a login call to a Shibboleth SP implementation, where the login URL and the URL to the target service are specified at the start of the page JavaScript.

Other types of SAML SP implementations will have their own style of consuming login information, so the login function has to be customized to fit the type of authentication server in use.

Advanced implementations

As suggested by the comments in the implementation example, this discovery script has a number of optional parameters that can be set for customization.

The most important is the **selectHeight** parameter (Default "300px").
This parameter specifies the maximum height of the select window where the user may choose from all IdPs in the federation. If the present IdPs exceeds the maximum height, a vertical scrollbar is added. This gives control to how much space the login page can afford to give the discovery UI.

The second most important feature is the ability to configure local URIs for the discover data feed and the storage data feed.

These feeds default to:

Discovery metadata feed (**discofeed**): https://eid2.3xasecurity.com/disco/json
Storage/Preference data feed (**storagefeed**): https://eid2.3xasecurity.com/disco/pref

These data feeds can be redirected through a local proxy in the service provider domain. If this is the case, then it is advisable to also the parameter **jsonp** to false. This because if these both feeds are provided through the same domain as the login page, then the data does not have to be transmitted as JSONP data in order to pass the same origin policy of the browser. It can now be transmitted as plan JSON data. Setting **jsonp** to false makes sure that not data is imported as JavaScript (which is the operation of JSONP).

The **discofeed** data feed can be cached to protect the service from single point of failure. This data is fairly static and does only change on metadata updates in the federation. This by keeping a local cache of this data at the Service Provider, in case the central service is unavailable; the user is ensured the possibility to login even if the central data source is unreachable.

4 Architecture

The discovery service is composed of 4 separate components:

- A JavaScript component that provides the discovery dialogue in the users Browser and communicate with the associated data feeds.
- A JSON Discovery Data Feed that provides necessary information about the federation metadata to the JavaScript.
- A Preference Storage Service that receives preference information stored in the user's browser and feed them back to the JavaScript through a JSON data feed.
- A Discovery Service that implements the SAML Identity Provider Discovery Service Protocol and Profile (OASIS Committee Specification 01, 27 March 2008).

The component relationships are outlined in the following table:

Component	Relationships
JavaScript	• Reads metadata from the JSON discovery data feed • Loads temporary iframes from the Preference Storage Service that reads user preferences stored under the context of the common domain of the Preference Storage Service. • Sends storage information under the context of the common domain to the Preference Storage Service and reads it back under the main window domain context from the same Preference Storage Service. (Thus making this information available to the domain context of the current service provider),
JSON Discovery Data Feed	Provides metadata information to the JavaScript
Preference Storage Service	Only communicates with the JavaScript
Discovery Service	Receives request for discovery by the Service Provider and loads the JavaScript to generate the discovery UI.
Service provider	Implements discovery by either: • Loading and running the discovery JavaScript in its own login web page, or • Transfers the user to the discovery service from its login page.

LEGITIMATIONS
NÄMNDEN

5 Appendix A – Revision History

Rev	Date	Update	Author
0.10	2013-01-30	First draft with revision history.	SS
0.70	2013-04-24	Updated draft with new document structure.	SS