Developing a registration application for UniVote with multiple identity provider access

Scalzi Giuseppe

Master of Science in Engineering, Project 1, Bern University of Applied Science, Biel/Bienne BE 2500, CH,
scalg1@bfh.ch

Abstract. This project aims to provide the study of web application that can be used in front of the UniVote system, in order to have at the end a registration application that can be used with many identity provider (a server that stores the identity of a user, we will call it IdP) instead that only one. Different security aspects must be evaluated and taken in account when building such systems. Attacks like cross-site request forgery or SQL-injections can easily compromise the integrity of a web application. During this report we will explain the development of the application and the different countermeasures used in order to prevent web-based threats and provide a working application.

Table of Contents

1 Introduction

In this chapter we will introduce the project by giving some background information. A little introduction about e-voting and UniVote is given at the beginning because our project is related with them. Then we explain how the actual situation looks like and at the end we can find the goals our project must achieve.

1.1 E-Voting

An e-voting platform is a system that lets users to participate to an election that is completely digital. This means that we can cast our vote from every part of the world. E-voting brings benefits such as the fact that everything is digitized and that an election can be created and managed more easily than a paper based one. Along with these advantages, there are also some problematic that are always present: is it really possible to vote and be sure that no one has cheated, modified or altered any vote during the whole process? To answer these problems modern e-voting systems make strong use of cryptography to fulfill requirements like the integrity of the data and the anonymity of involved parties. Next section will introduce the e-voting system related to this project: UniVote.

1.2 UniVote

UniVote is an e-voting platform developed at the Bern University of Applied Science (BUAS) by the e-voting group. This platform is used to perform electronic voting and it offers a great level of security in order to meet the modern security standards in term of e-voting. UniVote has been successfully used for some elections in the Swiss universities and it is always under development (the latest version is UniVote 1.7) to assure to voters the maximal functionality without losing anything in term of security. As it stands today, a user must first register to the system in order to cast a vote during an election. This is done by a registration application that at the end of the registration process generate a digital certificate to identify the user. After this registration phase the user can vote using the UniVote platform.

1.3 Actual situation

Here we want to give an overview of the existing registration application that is being used. The actual the registration system works only with an IdP, that is the SWITCHaai system used in the Swiss universities. So only users from Swiss schools can take part to an election held with the UniVote system.

The following image illustrates better the situation:

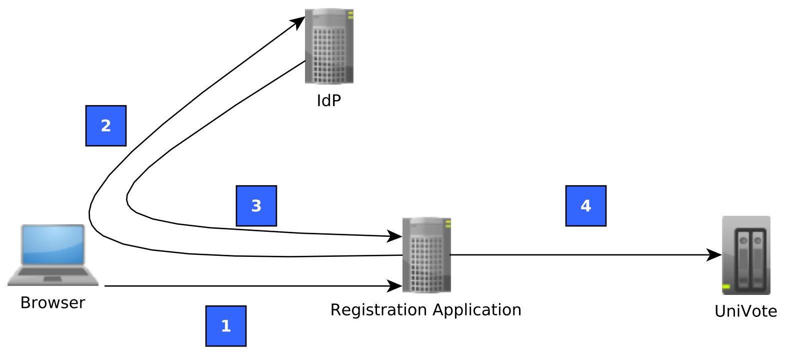


Fig. 1. “The actual registration flow”

The explanation of the figure is the following:

1. During step one, the client wants to register to UniVote so he surfs to the address of the registration application.
2. In step two, the registration application redirects the user to the IdP, in this case the SWITCHaai. At this point the user can insert log-in data.
3. Step three is where after checking the validity of the log-in data, the IdP redirects the user back to the registration application. The registration application knows that the user has an identity.
4. During the last step, the registration application creates a digital certificate with the data obtained from the IdP. This certificate is then stored in the UniVote system and sent back to the client.

After this little explanation of the registration application, in the next section we want to describe the goals of our project in relation to actual situation.

1.4 Goal

During this project we have to provide a study and a possible implementation of a new registration application with the following properties:

– give the possibility to use other IdPs
– get cryptographical proofs from the IdP stating that identities really come from the right IdP
– generate a digital certificate after that the user has been identify correctly

We now give a description of each goal.

1.4.1 Use other IdPs

The first goal of our project is to extend to others IdPs the authentication phase. So if at today there is only the possibility to use the SwitchAAI with the new registration application it will be possible at the end to use:

– Shibboleth (SwitchAAI)
– Facebook
– Google
– a mail server

For the first one we will use a Shibboleth installation to simulate the SwitchAAI platform because we don't have access to it. So a user will be redirected to a page where he has to insert a user name and a password to prove his identity. For the second and the third the behavior is quite the same as the first one, log-in with user name and password to Facebook and respectively Google servers.

Generally the first three IdPs will communicate with the registration application after a successful authentication so that we are able to identify users.

For the last one, the mail server things are different. Instead of redirecting the user to the IdP log-in page, the registration application will send an e-mail with an activation code to the specified recipient. If the user is able to get the activation code from the e-mail then we can say that is authenticated to a mail server and has an identity.

1.4.2 Cryptographical Proof

The second goal is two get a cryptographical proof from the listed IdP so that we can verify that the identities really come from them. This proof can be a digital signature that we can verify prior the digital certificate generation.

1.4.3 Digital certificates

At the end of the authentication phase the registration application must generate a digital certificate for the authenticated user. The application will get data needed to generate the certificate from the IdP used to log-in.

To conclude this paragraph we can say that the time budget for this project is a semester with Dr. Benoist as supervisors. No particular development method has been used.

1.5 Structure of the Report

The following sections of the report are organized as follows: the next one is about the security aspects involved during this project, after that we will talk about the technical implementation of the application. We will give details about the technologies involved and how the application works. The "Results" section will give an idea of the final result using some screen shots. At the end we give the conclusions. In the "Annexes" section you will find the code of the application.

2 Web Security Aspects

Before going into the technical aspects we would like to give some insight about the web security aspects involved. As reference we can use the OWASP Top 10, a chart of the ten most dangerous web security threats.

In particular we focus our attention to these threats:

– SQL Injections
– Cross-Site request forgery attacks

2.1 SQL Injections

An SQL injection occurs when a malicious user pass to a form a string of data that can change the behavior of the SQL query behind the form. This kind of attacks are nowadays the most frequent and we want to be sure that our application is not affected by them.

In general we have to escape any eventual parameters that comes from a GET or POST request inside a query. This can be done manually but there are library that can do this automatically leading to a better escape of an eventual malicious string. In our application we use the PDO library to interface with the MySQL database.

So we developed a class (DB.php) that can automatically use the PDO library when we query the database, A little code snippet can show this:

```
public function query($queryString, $values = null) {
  $stmt = $this->conn->prepare($queryString);

  if ($values != null) {
    $stmt->execute($values);
  } else {
    $stmt->execute();
  }

  return $stmt->fetch(PDO::FETCH_ASSOC);
}
```

The "prepare" statement is used optimize and prepare the query to be executed . Then we pass the values as an associative array in the "execute" function. In this way PDO escapes automatically any parameter in the query avoiding problems with possibles injections of unwanted character.

2.2 Cross-Site Request Forgery

A Cross-Site Request Forgery (CSRF) is a type of attack that occurs when the browser of a logged-on victim sends forged HTTP requests to a web application, including authentication information. The vulnerable web application thinks that these requests come from a legitimate user and it executes the request.

An example scenario can be:

- A is logged-on on his e-banking web site and receives an e-mail from B with a link to a low-cost flight site.
- This low-cost flight web site contains a <img> tag with source `http://a-ebanking.com/mymoney.php?withdraw=100`, A doesn't see this and she sends an unintentional request to her e-banking withdrawing 100$.
- The e-banking web site thinks that the request really comes from A because it also receives the authentication information and proceeds to the withdrawal.

To avoid this kind of automation, the e-banking has to include in every sensitive request a token that is used to identify the request.

In our application we use two techniques: captchas and token. In particular, with the OAuth protocol (explained in the "Technologies" subsection) we can use a parameter called "state" during the authenticate request to identify the flow of information. Captcha can also help to prevent these attacks because one has to first confirm the code shown in an image before proceeding.

3 Technical Implementation

In this chapter we explain the technologies used during this project as well as the implementation of the registration application. We begin with the different technologies involved.

3.1 Technologies

Here we give a brief description of each technology used in this project. This because we want to give to the reader an introduction before going into the real implementation and without this introduction it will be difficult to understand the rest of the report.

3.1.1 OAuth OAuth is an open standard used for authentication purposes. With this protocol we are able to access users' data without knowing the user name and the password they used to authenticate to a particular service like Facebook. So our application will be capable to obtain the access to user data (like e-mail or user ID) after that he is successfully authenticated to the chosen service. We do not explicitly use OAuth rather we use the software development kit (SDK) of Facebook and Google that make use of OAuth. The actual version of OAuth is the 2.0 and we can see how it works:

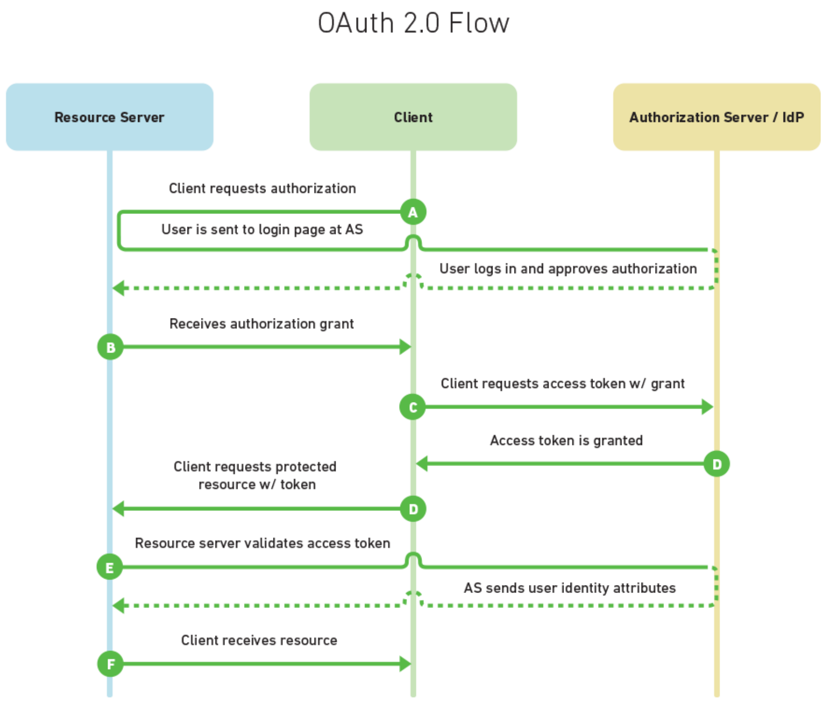

OAuth 2.0 Flow
Resource Server
Client
Authorization Server / IdP
Client requests authorization
A
User is sent to login page at AS
User logs in and approves authorization
Receives authorization grant
B
Client requests access token w/ grant
C
Access token is granted
D
Client requests protected resource w/ token
D
Resource server validates access token
E
AS sends user identity attributes
Client receives resource
F

Fig. 2. "The OAuth authentication process. Source `http://www.mutuallyhuman.com/blog/2013/05/09/choosing-an-sso-strategy-saml-vs-oauth2/`."

In this illustration our application plays the role of "Resource Server" while the IdP can be Facebook or Google. In OAuth there are some security measure that can be used: for example to protect against CSRF attacks we can use a parameter called "state" to guarantee. This make OAuth a valid choice.

3.1.2 SAML The Security Assertion Markup Language (SAML) is an XML-based open standard that is used to exchange authentication and authorization data between parties[2]. With SAML we have quite the same situation as OAuth: the user wants to access a resource on the service provider (SP) so he authenticates to an identity provider (IdP). The IdP will then send an assertion to the service provider to prove the successful authentication so that there is no need to provide user name and password. SAML is used by Shibboleth.

Here is the authentication process:

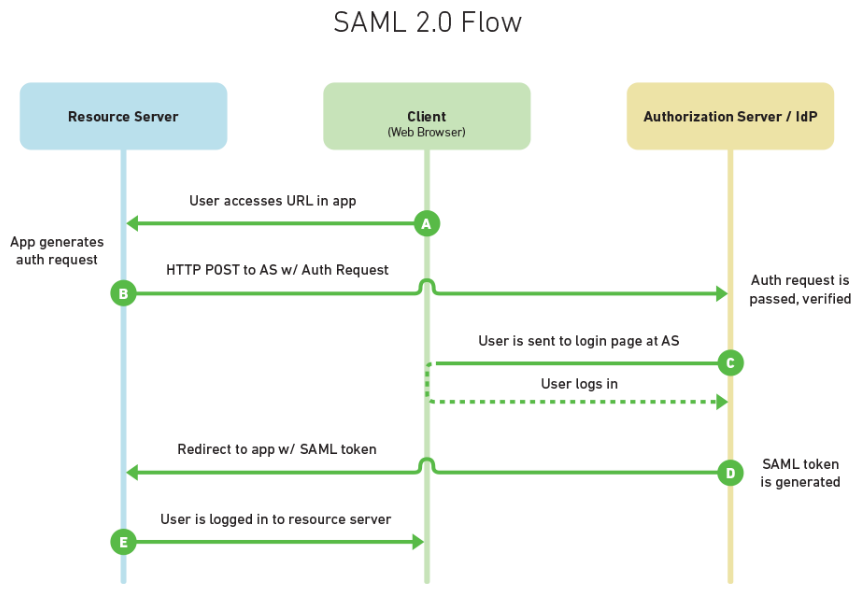

SAML 2.0 Flow
Resource Server
Client
(Web Browser)
Authorization Server / IdP
User accesses URL in app
A
App generates auth request
HTTP POST to AS w/ Auth Request
B
Auth request is passed, verified
User is sent to login page at AS
C
User logs in
Redirect to app w/ SAML token
D
SAML token is generated
User is logged in to resource server
E

Fig. 3. "The SAML authentication process. Source `http://www.mutuallyhuman.com/blog/2013/05/09/choosing-an-sso-strategy-saml-vs-oauth2/`."

As the previous OAuth illustration, our server is the resource server.

3.1.3 PHP The programming language chose for the development is PHP. With PHP we can easily include the different SDK of Google and Facebook so that we are able to communicate and get the different data we need. We can also include the PHP library we used:

– xmlseclibs: an XML security library . The main purpose of this library is to deal with XML signatures and encryption and we use it to verify the SAML assertion generated by the IdP. `http://code.google.com/p/xmlseclibs/`
– secureImage: library used to manage CAPTCHA. We use them in order to avoid that an attacker can automatize the registration with for example a script. (`http://www.phpcaptcha.org/`)

3.1.4 SPKAC The Signed Public Key and Challenge is a particular format for sending certification singing request to a server in order to create a client-side certificate. With this technology we send an encoded public key that can be manipulated by OpenSSL in order to create a digital certificate [3].

3.1.5 OpenSSL OpenSSL is the implementation of the SSL protocol and it is used to manage X.509 certificates. We use OpenSSL to generate RSA keys and users' certificates.

3.1.6 Digital Time-stamping Service A digital time-stamping service (DTS) is used to sign a document at a given time by a third party that we can identify as notary. We use a DTS because with Facebook, Google+ and the mail authentication we don't obtain a proof that we can later verify. So we generate our proof and we send it to the DTS and we will obtain a signature that we can later verify with the public key of the notary. On the counter part, with Shibboleth we obtain a SAML assertion digitally signed with the public key of the IdP that has generated it so we can easily check its validity. During this project we have build our DTS and you can find the code in the "Annexes" section (the implementation of the DTS is very simple and must be replaced by a real DTS).

3.1.7 Google Developers Console and Facebook Developer The Google Developers Console is a Google service used to give the access to the API to a specified application. When we enable a particular application in this console (the application can be for example an URL on the web) we are telling to Google that this application has access to a set of API that we can chose. These API can include for example the access to user data like e-mail or user ID.

On the other side the Facebook Developer works in quite the same way.

We will obtain a shared secret that we have to include in the server side of our application so that when we want to communicate with Google or Facebook, they will recognize our application as a trusted one, letting us to get useful information.

3.1.8 RSA, ElGamal and DSA These three elements are asymmetric algorithms used to encrypt/decrypt or sign data. RSA (Rivest, Shamir, Adleman) is known because it employs as security measure the fact that is hard to factorize a given number into prime numbers. This algorithm is very used when we speak about X.509 certificates. ElGamal, from the creator Taher Elgamal, uses the discrete log problem as security measure and the last crypto-system, DSA (Digital Signature Algorithm) is a standard for digital signatures. DSA is very similar to ElGamal when we compare for example the parameters that has to be chosen or the fact that is also based on discrete log problems.

3.2 Application Overview

In this section of the report we would like to give an overview of the new registration application by explaining in a general way how the application works.

3.2.1 Index Page For the duration of the project the address of the application is https://localhost/ so when the user types this address the index page will be displayed.

In the index page we can find the four authentications mechanisms. For Facebook, Google+ and Shibboleth we have to click on the provided link and the user will be redirected to the respective server where he can authenticate himself.

3.2.2 Redirection Page When the user has successfully authenticated to an IdP it will be redirected to our registration application. Each authentication system has it own page, this because each of them works in a different way. For Google+ and Facebook we cannot use their authentication system if we don't create an "application" on their platform and when we create this application we have to specify a "redirection URI" used to redirect the user after a successful authentication. For the mail authentication system we simply send the URI via e-mail and for Shibboleth we can also chose the URI we want so we have chosen to have four different URIs in order to keep things separated from each other:

- Facebook: https://localhost/fb/
- Google+: https://localhost/gplus/
- Mail: https://localhost/mail/
- Shibboleth: https://localhost/shib/

In these four pages, the cryptographical proof will be checked and stored in order to be sure that the identity really comes from the IdP. For Facebook, Google+ and the mail system we create the proof and then we send it to a DTS that can sign this proof so that we can later verify it. The DTS will send the signature back to our application so we will be capable to check the validity. In the case of Shibboleth we don't use the DTS because we have a SAML assertion that can be verified using the public key of the IdP that has generated it.

3.2.3 Generation Page After the verification of the proof we show a form for generating a digital certificate (X.509) so that the authenticated user can obtain one. The form will be submitted to the "gen.php" page that after the various security checks will generate and send back to the client a certificate. We explain in details the generation of the certificate in the dedicated section "Certificates Generation".

We can resume the different pages involved using this illustration:

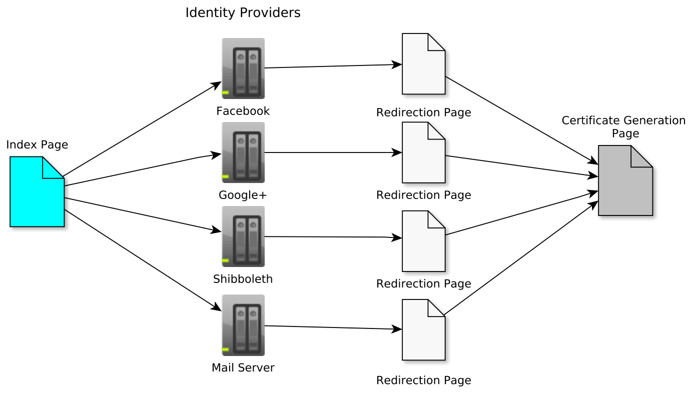


Fig. 4. "The pages of the application."

3.2.4 Directory Tree

Here we have the directory tree of the application:

```
|-- project1
|   |-- class
|   |   |-- DB.php
|   |   |-- Verificator.php
|   |   `-- Welcome.php
|   |-- etc
|   |   `-- Config.php
|   |-- fb
|   |   |-- channel.html
|   |   `-- index.php
|   |-- gplus
|   |   `-- index.php
|   |-- library
|   |   |-- PHPMailer
|   |   |-- facebook-php-sdk
|   |   |-- google-api-php-client
|   |   |-- securimage
|   |   `-- xmlseclibs
|   |-- mail
|   |   `-- index.php
|   |-- pages
|   |   |-- gen.php
|   |   `-- index.php
|   `-- shib
|       `-- index.php
```

The directory tree has been organized to be as simple as possible. So we have a "class" folder for PHP classes, an "etc" folder with the constants file. Then the four folders "fb", "gplus", "mail" and "shib" explained in the "Redirection Page" the "pages" folder where we can find the index and the page used to generate the certificate. The last one is used to keep all the library.

3.3 Authentications

This section is dedicated to the explanation of the four authentications system more in details. We will include some portions of code to better explain how the system works.

3.3.1 Facebook

We will begin by explaining the link “Register with Facebook” in the index page. This is generated with the Facebook SDK call “getLoginUrl” so we don’t have to worry about the additional parameters needed.

Here is the code if the “facebookLogin” method in the “Welcome.php” class:

```
public function facebookLogin() {
  $config = array();
  $config['appId'] = Config::FACEBOOK_APP_ID;
  $config['secret'] = Config::FACEBOOK_SECRET;
  $facebook = new Facebook($config);
  $loginUrl = $facebook->getLoginUrl(array('scope' => 'email', 'redirect_uri' =>
      Config::FACEBOOK_REDIRECT_URL));
  $values = explode("?", $loginUrl);
  $fields = explode("&", $values[1]);
  echo "<form action='https://www.facebook.com/dialog/oauth' method='post'>";
    foreach ($fields as $value) {
      $expVal = explode("=", $value);
      echo "<input type='hidden' name='$expVal[0]' value='" . urldecode($expVal[1])
          . "' />";
    }
  echo "<input type='hidden' name='method' value='post' />";
  echo "<input type='submit' value='Register with Facebook' />";
  echo "</form>";
}
```

The first lines are used to initialize the SDK where we pass the application id and secret in order to create a “Facebook” object. The “scope” parameters in “getLoginUrl“ is used to tell Facebook that we want to have access to the email and “redirect_uri” is the URI to where redirect the user after he is authenticated. The result is this link:

```
 https://www.facebook.com/dialog/oauth?client_id=543670079021930
 &redirect_uri=https%3A%2F%2Flocalhost%2Ffb%2F
 &state=9587ac1d3d50d15f861fc6f6515f2f45
 &sdk=php-sdk-3.2.2
 &scope=email
```

The “client_id” parameter is the ID of our application registered with Facebook and “state” is used to protect against CSFR attacks. This parameter will be passed to Facebook and if we will obtain after the redirection, we will be able to say if the request comes from the origin and not from a possible attack. When the user clicks it will be redirect to the Facebook log-in page and after the authentication he will be redirected to “https://localhost/fb/”. Note that we build a form in order to avoid to expose the CSRF token, this is a practice that is suggested by OWASP. The code relative to this page is the following:

```
...
try {
  if($facebook->getAccessToken()){
    $verificator = new Verificator();
    $user = $facebook->api('/me');
    $email = $user['email'];
    $userid = $user['id'];
    $userDB = base64_encode($userid . "." . $email);
    $hashInfo = hash('sha256', $userDB);
    $proofVer = $verificator->verifiyFacebook($hashInfo, $userDB, $user['id'],
        $email);
  }
} catch (Exception $e) {
  echo $e;
  error_log($e);
  exit;
}
...
```

With “getAccessToken” the SDK check if after the authentication a valid access token exists. The access token state that the user has authenticated himself and if this token is invalid or doesn’t exist the SDK will complain exiting immediately. This call also check if the CSRF token “state” match with the previous one. After this step we get the user profile with “$facebook->api('/me')” and we extract the user ID and email because we will use them to generate our proof. The “$userDB” variable holds the result of the concatenated and base64 encoded email and user ID. We will pass this variable to the “hash” function in order encrypt this evidence because we don’t want that the DTS will see what’s inside. We use a SHA-256 hash. We then create a “Verificator” object used to verify the proofs and call “verifiyFacebook” with the hash, the base64 encoded clear text and the id of the user as parameters. This function basically performs these steps:

- Send the hash to the DTS in order to obtain a signed version of our hashInfo
- Verify the hash received by the DTS
- Insert into the database the proof signed by the DTS

This function will return true if the verification doesn’t encounter any problems. You can find the implementation in the “Verificator Class” section at the end. In this case we can show the form for generating the certificate:

```
...
<tr><td><?php if($proofVer) $verificator->showCaptchaForm () ?></td></tr>
...
```

3.3.2 Google+ The authentication system of Google+ works quite in the same way as Facebook.

We begin with the log-in url, generated by the “gplusLogin” method of the “Welcome.php” class:

```
public function gplusLogin() {
  $client = new Google_Client();
  $client->setClientId(Config::GOOGLE_CLIENT_ID);
  $client->setClientSecret(Config::GOOGLE_CLIENT_SECRET);
  $client->setRedirectUri(Config::GOOGLE_REDIRECT_URL);
  $client->setDeveloperKey('http://localhost');
  $client->setScopes(array('https://www.googleapis.com/auth/userinfo.email',
      'https://www.googleapis.com/auth/userinfo.profile'));
  $plus = new Google_PlusService($client);
  $state = md5(rand());
  $_SESSION['gplus_state'] = $state;
  $url = $client->createAuthUrl()."&state=".$state;
  $values = explode("?", $url);
  $fields = explode("&", $values[1]);
  echo "<form action='https://accounts.google.com/o/oauth2/auth' method='post'>";
  foreach ($fields as $value) {
    $expVal = explode("=", $value);
    echo "<input type='hidden' name='$expVal[0]' value='" . urldecode($expVal[1]) .
        "' />";
  }
  echo "<input type='hidden' name='method' value='post' />";
  echo "<input type='submit' value='Register with Google+' /></form>";
}
```

This code does quite the same thing as the one for Facebook. We initialize the Google SDK with the application ID, the secret and the redirect URI. We set the scopes so that we can access the user profile and the email. We generate also a “state” so that we can use it later to identify a possible CSRF. In the case of Facebook this is done automatically but with the SDK of Google we have to do it by hand. Then we call “createAuthUrl” to obtain the desired URL.

The generated link is the following:

```
https://accounts.google.com/o/oauth2/auth?response_type=code
&redirect_uri=https%3A%2F%2Flocalhost%2Fgplus%2F
&client_id=142948380952.apps.googleusercontent.com
&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+
https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.profile
&access_type=offline
&approval_prompt=force
&state=45f6d500228f6349c9fd35c828bcb205
```

The parameters are quite the same as Facebook and they don’t need additional explanation.

When the redirection to our application has completed the following code is executed:

```
...
if (isset($_GET['code']) && $_GET['state'] == $_SESSION['gplus_state']) {
  try {
    $client->authenticate($_GET['code']);
    $verificator = new Verificator();
    $token = json_decode($client->getAccessToken());
    $request = new
        Google_HttpRequest("https://www.googleapis.com/oauth2/v1/userinfo?
    alt=json&access_token=$token->access_token");
    $resp = $client->getIo()->authenticatedRequest($request);
    $userInfoJson = json_decode($resp->getResponseBody());
    $userid = $userInfoJson->{'id'};
    $email = $userInfoJson->{'email'};
    $gplusValue = base64_encode($userid . "." . $email);
    $hashInfo = hash('sha256', $gplusValue);
    $proofVer = $verificator->verifyGooglePlus($hashInfo, $gplusValue,
        $userInfoJson->id);
  } catch (Exception $e) {
    echo $e;
    error_log($e);
  }
} else {
  echo "CSRF token state doesn't match.";
  error_log("CSRF token state doesn't match.");
}
...
```

We first check if we received the "code" parameter used to get the access token and we also check if the state parameter we stored correspond to the one sent by Google. Then with "authenticate" we check if we can obtain the token by using the code received. We make a Google_HttpRequest to retrieve the user profile so we extract the info we need from the JSON we obtain. After we verify the proof and we show the form for generating the certificate as done before with Facebook.

3.3.3 Shibboleth With Shibboleth we have first to configure a service provider (Sp) that will lately communicate with the IdP where the identity of the user resides. In the real case the IdP will be the SWITCHaai system but we will use an IdP made available by the testshib.org project. For the configuration we followed the steps provided on the testshib.org website.

In particular after configuring and installing the Sp we have to configure Apache so that we can have a resource that we can access only after the authentication with the IdP. In our case we add this to the Apache configuration:

```
<Location /shib/>
  AuthType shibboleth
  ShibRequestSetting exportAssertion true
  ShibRequestSetting requireSession 1
  require valid-user
</Location>
```

We will serve the resource https://localhost/shib/ only after an authentication with the IdP. Note the "exportAssertion true" option: this is used to export the SAML assertion exchanged between the IdP and the Sp so that we can use it as a proof (the configuration file of the Sp must be also changed: add the attribute "exportLocation=`"https://localhost/Shibboleth.sso/GetAssertion"`" to the "Session" tag in shibboleth2.xml[8]).

When the user will click on the link it will be redirected to the page of the IdP, more precisely at this URL:

```
https://idp.testshib.org/idp/profile/SAML2/Redirect/SSO?SAMLRequest=
fZFLb8IwEIT%2FSuQ7efEIsghSCociQYkI7aGXynEWYsmxU6%2FTln9fQ6hKVYmb5Z
351jOeIWtkS7PO1moH7x2g9b4aqZBeBinpjKKaoUCqWANILadFtlnT2A9pa7TVXEvi
ZYhgrNBqoRV2DZgCzIfg8Lxbp6S2tkUaBFJzJmuNNihqUZZagq19RB2cgXGQb4s98Z
buBUKxM%2BvXKarWt26AzudrczxfBG77QUi4undQCQPcsYst8VbLlLwdKj6GSVlCEh
4YSzhnbFhGE5bEU3cYjpwMsYOVQsuUTUkcRqNBGA3C6T5K6Dim49Er8fJryAehKqGO
```

```
9xspexHSx%2F0%2BH%2FSJXsDgJY0TkPns3Cu9LDY3Td%2FHsp96yfx%2FmZtTkc%2
BCG2y%2Fo6VPjrNa5loKfvIyKfXnwgCzkJKIBPPe8vfr598%3D
&RelayState=ss%3Amem%3A42a3662cb37594966885ec83aacdb5f9ad944f6e812
a8764df7f3d15c3207c7f
```

After the authentication the page on "https://localhost/shib/":

```
...
if (isset($_SERVER['Shib-Assertion-01']) && isset($_SESSION['shib_state']) &&
    $_SESSION['shib_state'] == $_GET['state']) {
  try {
    $shibAssertion = $_SERVER['Shib-Assertion-01'];
    $verificator = new Verificator();
    $proofVer = $verificator->verifyShibboleth($shibAssertion, $_SERVER['eppn']);
  } catch (Exception $e) {
    echo $e;
    error_log($e);
  }
} else {
  echo "Something wrong happened while accessing this page.";
  error_log("Something wrong happened while accessing this page.");
}
...
```

This code first check if the URL of the SAML assertion exists (session variable $_SERVER['Shib-Assertion-01'] and then check the CSRF token. Then we pass the URL of the assertion to the "verifyShibboleth" function to check the correctness of the assertion with the xmlseclibs. The "$_SERVER['eppn'] contains the user name the authenticated user.

3.3.4 Mail

This kind of authentication works a bit different if compared to the other. In this case we say a user is authenticate with his email if he can submit to the application a code sent to him. If he can read the email he has access to a mail server with a user name and a password.

Instead of a link we have a form where the user can submit the email:

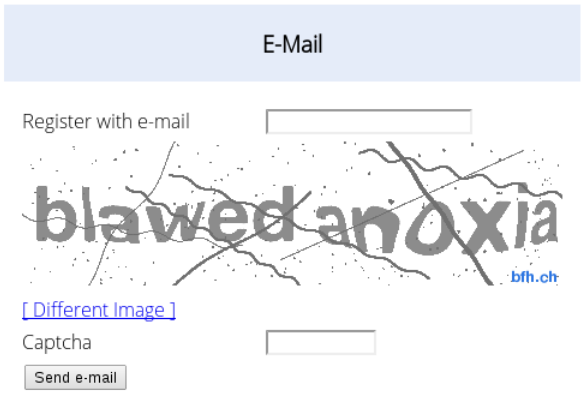


Fig. 5. "The form of the email authentication method."

The CAPTCHA is used to avoid an automation when submitting the form for example by a script that run remotely.

After the submission the following code checks the email and generates the code:

```
if (isset($_POST['captcha_code']) && isset($_POST['email'])) {
  $secureImage = new Securimage();
  $val = $secureImage->check($_POST['captcha_code']);
  $regex =
       '/^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$/';
  $email = strtolower($_POST['email']);

  if (isset($_POST['sendEmail']))
    if ($val && preg_match($regex, $email)) {
      $bin = file_get_contents('/dev/urandom', false, null, 0, 75);
      $random_string = bin2hex($bin);
      $link = "https://localhost/mail/?c=" . $random_string;
      $mailer = new PHPMailer();
      $mailer->IsSMTP();
      $mailer->Port = 587;
      $mailer->AddAddress($email, $email);
      $mailer->Host = "mail.server.com";
      $mailer->From = "register@bfh.ch";
      $mailer->Subject = "Registration";
      $mailer->SMTPAuth = true;
      $mailer->SMTPDebug = 0;
      $mailer->Username = "register@bfh.ch";
      $mailer->Password = "---------";
      $mailer->Body = "Click on the link in order to register yourself: $link";

        if (!$mailer->Send()) {
          return $mailer->ErrorInfo;
        } else {
```

```
        $db = new DB();
        $db->query("INSERT INTO mailAuthRequests (email,code) VALUES
        (:email,:code)", array('email' => $email, 'code' => $random_string));
        return 1;
      }
  } else {
    return -1;
  }
}
```

The first lines are used to check the various parameters like the CAPTCHA (with the secureImage library) or the email with a regexp. After these steps we create the random code we would like to the via email to the user. We use "file_get_contents" to get random data (maximal length 75) from the device "/dev/urandom". This device generates random data and can be useful for these purposes. After that we configure the PHPMailer in order to be able to send the email. If the email has been sent successfully we store the request into the database.

The user will receive a link like this:

```
https://localhost/mail/?c=a9f8970570a751cbc15ab46aa67e7b6bc614de84
f2918f670437b5b25af6f033a7f78c0f6ac6a651cdea9134cdb39bcf55adde8699
cd04b9882c398131c1d7ce38d82bc63627687cb0f68f
```

When he will click the page on "https://localhost/mail" will perform the following:

```
$code = $_GET['c'];
$conn = new PDO('mysql:host=localhost;dbname=project1', 'user', 'pass');
$stmt = $conn->prepare("SELECT email,code,requestTime,verified FROM mailAuthRequests
      WHERE code = :code");
$stmt->execute(array('code' => $code));
...
if ($stmt->rowCount() == 1) {
  $result = $stmt->fetch(PDO::FETCH_ASSOC);
  date_default_timezone_set("Europe/Zurich");
  $storedTime = new DateTime($result['requestTime']);
  $actualTime = new DateTime(date("Y-m-d H:i:s"));
  $difference = $storedTime->diff($actualTime);
  $expired = false;

  if ($difference->format("%i") > 30 || $result['verified']) {
    $expired = true;
  } else {
    $verificator = new Verificator();
    $hashEmail = hash('sha256', $result['email']);
    $proofVer = $verificator->verifyEmail($hashEmail, $result['email'], $code);
    $update = $conn->prepare("UPDATE mailAuthRequests tSET verified = TRUE WHERE
        code = :code");
    $update->execute(array('code' => $code));
  }
}
...
```

This code selects the code from the requests table and if it exists it checks if is it still valid. The duration of each code is 30 minutes. If the code is valid we will generate the proof thanks to the “verifyEmail” function. At the end we update the table of the mail requests.

3.4 Certificates Generation

The step of generating a certificate is one of the goal of this project. An X.509 must be generated when the user has successfully authenticated to an IdP: this certificate will be later used to identify him inside the UniVote system. We have two use cases for generating a certificate:

- **RSA key pair**: one set of certificates must be generated including an RSA key as public key. The relative private key must be kept on the user’s computer. This kind of certificates are used by election administrator in UniVote.
- **ElGamal key pair**: the other set of certificates must include an ElGamal key. Generally is not possible to include an ElGamal key into a X.509, but due to the similarity with DSA this is possible. The key type in the certificate will always be DSA but we include an ElGamal key. These certificates are used for identifying users in UniVote.

The certificate will be client side, this means that our application must only sign the public key that comes from the user. This means that the user has the responsibility to keep the secret key. During the explanation of the certificate generation we will see how these problematics have been solved.

3.4.1 RSA Certificates

To generate these kind of certificates we have used a technology called SPKAC (introduced in the technology section). In particular when the user lands to the redirection page of the chosen IdP he see will a form, this form contains a “keygen” tag:

```
<keygen name='pubkey' challenge='randomchars'>
```

This tag as explained by w3schools.com “specifies a key-pair generator field used for forms. When the form is submitted, the private key is stored locally, and the public key is sent to the server.”[9] So with this method we have the advantage that we don’t need to manage any cryptographical operation inside our web browser. The private key is stored inside the browser while the public key is sent to the server.

When the tag is rendered, the user can chose from a drop down menu the length of the key. If we don’t use the “type” attribute an RSA key will be generated. The browser will then generate a SPKAC that our server will sign inside the certificate.

This is an example of what can be sent to the server[4]:

```
commonname=John+Doe&email=doe@foo.com&org=Foobar+Computing+Corp.&
  orgunit=Bureau+of+Bureaucracy&locality=Anytown&state=California&country=US&
  key=MIHFMHEwXDANBgkqhkiG9w0BAQEFAANLADBIAkEAnX0TILJrOMUue%2BPtwBRE6XfV%0AWt
  KQbsshxk5ZhcUwcwyvcnIq9b82QhJdoACdD34rqfCAIND46fXKQUnb0mvKzQID%0AAQABFhFNb3
  ppbGxhSXNNeUZyaWVuZDANBgkqhkiG9w0BAQQFAANBAAKv2Eex2n%2FS%0Ar%2F7iJNroWlSzSM
  tTiQTEB%2BADWHGj9u1xrUrOilq%2Fo2cuQxIfZcNZkYAkWP4DubqW%0Ai0%2F%2FrgBvmco%3D
```

The private part is stored inside the cryptographical wallet of the browser that has generated the request. We can export the private key for example using PKCS12 so that we can protect it with a password.

In order to generate a valid certificate we can collect the other data needed from Facebook, Google+, Shibboleth or the e-mail. So we will be able to have all the fields like the common name, the e-mail or the organizational unit.

The PHP page “gen.php” will generate the certificate. Here we show the code we use[6]:

```
...
$CAmail = $_SESSION['email'];
$CAorg = "RISIS";
$CAcountry = "CH";
$CAcity = "Bern";
$CAlocality = "Biel";

$key = $_REQUEST['pubkey'];
str_replace(str_split(" \t\n\r\0\x0B"), '', $key);
$keyreq = "SPKAC=" . str_replace(str_split(" \t\n\r\0\x0B"), '', $key);
$keyreq .= "\nemailAddress=" . $CAmail;
$keyreq .= "\n0.OU=" . $CAorg . " client certificate";
$keyreq .= "\norganizationName=" . $CAorg;
$keyreq .= "\ncountryName=" . $CAcountry;
$keyreq .= "\nstateOrProvinceName=" . $CAcity;
$keyreq .= "\nlocalityName=" . $CAlocality;

$opensslconf = "/etc/ssl/openssl.cnf";
$days = '180';
$certfolder = "/tmp/";
$capw = "pass";

file_put_contents($certfolder . $username . ".spkac", $keyreq);

$command = "export HOME=\"/etc/ssl/myCA\" && /usr/bin/openssl ca -config " .
    $opensslconf . " -days " . $days . " -notext -batch -spkac " . $certfolder .
    $username . ".spkac -out " . $certfolder . $username . " -passin pass:'" .
    $capw . "' 2>&1";

error_log(shell_exec($command));
....
```

We now explain the code. With SPKAC we can pass to OpenSSL a request that is stored inside a text file. The format of this file is simple: the first line must be the SPKAC key received followed by a “\n” and by all the other filed needed, always followed by a “\n”. After creating this file (file_put_contents), we invoke the openssl command to generate a certificate using this file. We specify as attributes the configuration of OpenSSL with “-config”, the days of validity with “-days”, “-batch” means to not prompt anything, the location of the SPKAC file with “-spkac”, the location of the generated X.509 certificate with “-out” and at the end the password of the certification authority used to create the certificate with “-passin”. Not that at the beginning we have “export HOME=¨/etc/ssl/myCA¨” because if we omit this export OpenSSL won’t be able to find some file needed during the generation process.

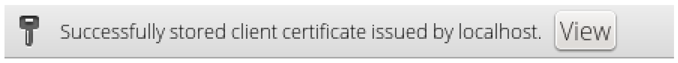
After creating the certificate on the server, we need to send it back to the client:

```
...
```

```
$certName = $serial . ".pem";
$length = filesize($certfolder . $username);
header('Last-Modified: ' . date('r+b'));
header('Accept-Ranges: bytes');
header('Content-Length: ' . $length);
header('Content-Type: application/x-x509-user-cert');
readfile($certfolder . $username);
exit;
```

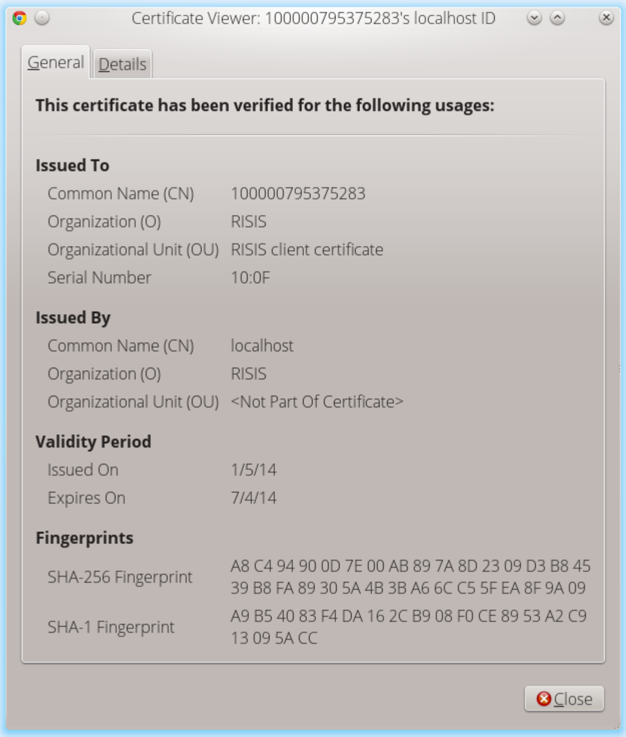
So we take the name of the freshly created certificate we compute the size with “filesize” and we send the headers to the client browser to specify the date we are about to send. Finally with “readfile” we send the certificate“.

If everything works fine the user should see this message inside the browser after clicking submit:



Successfully stored client certificate issued by localhost.

Fig. 6. “The successful created and stored certificate.”



This certificate has been verified for the following usages:

Issued To

Common Name (CN)	100000795375283
Organization (O)	RISIS
Organizational Unit (OU)	RISIS client certificate
Serial Number	10:0F

Issued By

Common Name (CN)	localhost
Organization (O)	RISIS
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	1/5/14
Expires On	7/4/14

Fingerprints

SHA-256 Fingerprint	A8 C4 94 90 0D 7E 00 AB 89 7A 8D 23 09 D3 B8 45 39 B8 FA 89 30 5A 4B 3B A6 6C C5 5F EA 8F 9A 09
SHA-1 Fingerprint	A9 B5 40 83 F4 DA 16 2C B9 08 F0 CE 89 53 A2 C9 13 09 5A CC

Fig. 7. "The freshly created certificate."

3.4.2 ElGamal Certificates The generation of these certificate is a little bit particular because we have to include an ElGamal key inside an X.509 certificate. This use case is at today not defined but due to the similarity with DSA, we can put the ElGamal key at the place of a DSA one. Note that the key in the certificate will be always identified by a DSA key.

We need a library like "Bouncy Castle" for Java that works with PHP. With this library we can create a certificate using the ASN1 notation, so we can insert each element of a X.509 certificate in an independent manner. The idea is to insert an ElGamal key in the "subjectPublicKey" field by leaving the algorithm of the "SubjectPublicKeyInfo" to "dsaEncryption". The ASN1 structure of a X.509 certificate can be found at `http://www.ietf.org/rfc/rfc3280.txt`, point 4.1.

For PHP we found two library that can be useful to do this kind of operation:

- phpseclibs: a security library for PHP that can be used to generate X.509 certificates (`http://phpseclib.sourceforge.net/`).
- PHPASN1: a library that can be used to create arbitrary ASN1 structures (`https://github.com/FGrosse/PHPASN1`).

Unfortunately we were not able to generate this kind of certificates. With the first library the problem is that the DSA algorithm is not supported. At today only RSA is supported but as this page says (`http://phpseclib.sourceforge.net/ideas.htm`) a possible DSA implementation is possible. With the second library, we were able to generate an ASN1 structure that can be compared to the one of a X.509 certificate. This structure can be found in the "pages" directory under the "phpasn1.php" name. The problem with the PHPASN1 library is that it does not support big numbers. So we were not able to put inside some fields big numbers like the ElGamal parameters.

4 Results

The application has been developed and it is functional. A user can authenticate using one of the four IdP proposed and after this step he can generate a digital certificate using only RSA. The proofs are also generated and stored so that we can later verify them. Here we would like to illustrate with some images the final result:

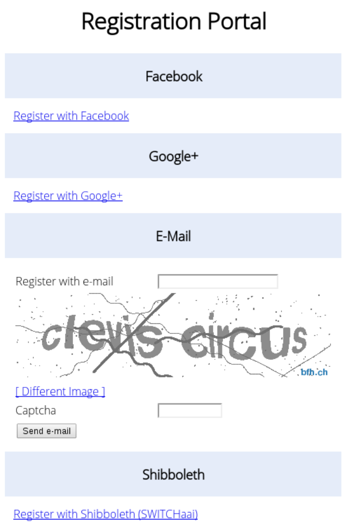
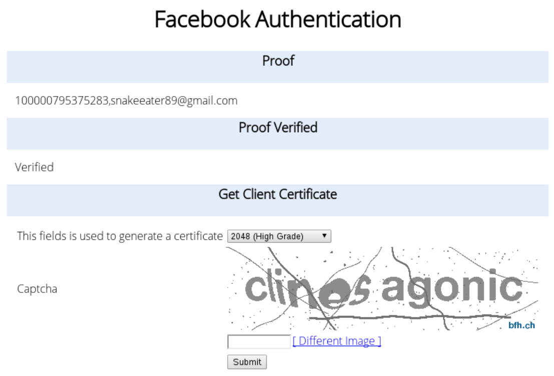


Fig. 8. "The index page of the registration application."

This is the main page with the four authentication mechanisms. The user will see this when typing the address in the browser.

The redirection page after being successfully authenticated with Facebook.


Facebook Authentication

Proof

100000795375283,snakeeater89@gmail.com

Proof Verified

Verified

Get Client Certificate

This fields is used to generate a certificate: 2048 (High Grade)

Captcha

Fig. 9. “The redirection page for Facebook.”

In this page the user should choose the maximal length of the key and then confirm the form submission with a CAPTCHA. The server will then generate a X.509 certificate with an RSA key. This certificate will be installed in the browser.

5 Future Work

For possibles future improvements we can say that the ElGamal certificate generation part must be improved. If the two libraries listed in the section “ElGamal Certificates” will be further developed by including the necessary changes they can be included in the application to provide the functionality. If this functionality will be included we have also to introduce a control in the different redirection pages in order to show the RSA or the ElGamal certificate generation.

6 Conclusion

After the development of this project we learned interesting things about how can we use other already available services to authenticate users. This can have advantages like the fact that we don’t have to keep our local database with emails and passwords. A user can also see this as an advantage so he doesn’t have to register every time with new credentials. We have also see how big enterprises use authentication protocols such as OAuth to provide the authentication for third party applications.

Almost every goal has been reached. The only goal that has not been completely reached is the certificate generation. In fact we are only able to generate digital certificates that make use of RSA as key algorithms. For ElGamal certificates we were not able to find a suitable library like “Bouncy Castle” for Java that we can use to build an X.509 certificate with an ElGamal key instead of a DSA key.

We also stated how security is important. Even if we developed a functional application there can always be threats that can change the behavior of a software so these aspects must be evaluated and taken in account since the beginning of a project.

This integration between security and the various technologies employed has also contributed to increase our personal technical baggage, making this project useful and interesting at the same time.

References

1. UniVote System Specification, `http://e-voting.bfh.ch/projects/univote/`
2. SAML, `http://en.wikipedia.org/wiki/Security_Assertion_Markup_Language`
3. SPKAC, `http://en.wikipedia.org/wiki/SPKAC`
4. The “keygen” tag, `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/keygen`
5. RFC 3280 Public Key Infrastructure, `http://www.ietf.org/rfc/rfc3280.txt`
6. PHP code for SPKAC certificate generation,`https://www.mail-archive.com/whatwg@lists.whatwg.org/msg10411.html`
7. OWASP, `https://www.owasp.org/index.php/Main_Page`
8. Shibboleth Wiki, `https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPAssertionExport`
9. w3schools.com, `http://www.w3schools.com/tags/tag_keygen.asp`

7 Annexes

Here we will list the most important pages and classes used for this project.

7.1 Welcome Class

```
 <?php

/**
 * This class is used to show the different login options of the home
 * page. It's also used to check the e-mail submission
 *
 * @author snake
 */
class Welcome {

        /**
         * Display the Facebook login URL
         */
        public function facebookLogin() {
                require_once("../library/facebook-php-sdk/src/facebook.php");

                $config = array();
                $config['appId'] = Config::FACEBOOK_APP_ID;
                $config['secret'] = Config::FACEBOOK_SECRET;

                $facebook = new Facebook($config);
        $loginUrl = $facebook->getLoginUrl(
                array('scope' => 'email', 'redirect_uri' =>
                    Config::FACEBOOK_REDIRECT_URL));

        //split the values of the url in order to construct a form
        $values = explode("?", $loginUrl);
        $fields = explode("&", $values[1]);

        echo "<form action='https://www.facebook.com/dialog/oauth'
            method='post'>";
        foreach ($fields as $value) {
                $expVal = explode("=", $value);
                echo "<input type='hidden' name='$expVal[0]' value='" .
                    urldecode($expVal[1]) . "' />";
        }
        echo "<input type='hidden' name='method' value='post' />";
        echo "<input type='submit' value='Register with Facebook' />";
        echo "</form>";
}

/**
 * Display the Google+ login URL
 */
public function gplusLogin() {
        require_once
            '../library/google-api-php-client/src/Google_Client.php';
        require_once
            '../library/google-api-php-client/src/contrib/Google_PlusService.php';

        $client = new Google_Client();
        $client->setClientId(Config::GOOGLE_CLIENT_ID);
        $client->setClientSecret(Config::GOOGLE_CLIENT_SECRET);
        $client->setRedirectUri(Config::GOOGLE_REDIRECT_URL);
        $client->setDeveloperKey('http://localhost');
        $client->setScopes(array('https://www.googleapis.com/auth/userinfo.email',
             'https://www.googleapis.com/auth/userinfo.profile'));
        $plus = new Google_PlusService($client);

        //create the state token used for CSRF
        $state = md5(rand());
        $_SESSION['gplus_state'] = $state;
        $url = $client->createAuthUrl() . "&state=" . $state;

        //split the values of the url in order to construct a form
        $values = explode("?", $url);
        $fields = explode("&", $values[1]);

        echo "<form action='https://accounts.google.com/o/oauth2/auth'
            method='post'>";
        foreach ($fields as $value) {
                $expVal = explode("=", $value);
                if ($expVal[0] == 'redirect_uri') {
                        echo "<input type='hidden' name='$expVal[0]'
                            value='https://localhost/gplus/' />";
                } else {
                        echo "<input type='hidden' name='$expVal[0]' value='" .
                             urldecode($expVal[1]) . "' />";
                }
        }
        echo "<input type='hidden' name='method' value='post' />";
        echo "<input type='submit' value='Register with Google+' /></form>";
}

/**
 * Display the e-mail submission form
 */
public function emailLogin() {
        echo "<form name='email-auth' method='POST''>
                <table>
                <tr><td>Register with e-mail</td><td><input type='text'
                    name='email' /></td></tr>
                <tr><td colspan='2'><img id='captcha'
                    src='../library/securimage/securimage_show.php'
                    alt='CAPTCHA Image' /></td></tr>
                <tr><td colspan='2'><a href='#'
                    onclick=\"document.getElementById('captcha').src =
                    '../library/securimage/securimage_show.php?' +
                    Math.random();
                       return false\">[ Different Image ]</a></td></tr>
                <tr><td>Captcha</td><td><input type='text' name='captcha_code'
                     size='10' maxlength='15' /></td></tr>
                <tr><td colspan='2'><input type='submit' value='Send e-mail'
                    name='sendEmail' /></td></tr>
                </table>
        </form>";
}

/**
 * Display the Shibboleth login URL
 */
public function shibbolethLogin() {
        //create the state token used for CSRF
        $state = md5(rand());
        $_SESSION['shib_state'] = $state;

        echo "<form action='" . Config::SHIB_SECURE_URL . "?state=$state'
            method='post'>";
        echo "<input type='submit' value='Register with Shibboleth' />";
        echo "</form>";
}

/**
 * Check the submission of the email
 * @return int return 1 if is all good, -1 if there are problems for example
 * with the e-mail format and a message if there are problems when sendind
     the e-mail.
 */
public function checkEmailSubmission() {
       require_once "../library/securimage/securimage.php";

       if (isset($_POST['captcha_code']) && isset($_POST['email'])) {
              $secureImage = new Securimage();

              $val = $secureImage->check($_POST['captcha_code']);
              $regex =
                  '/^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$/';
              $email = strtolower($_POST['email']);

              if (isset($_POST['sendEmail']))
                     if ($val && preg_match($regex, $email)) {

                            require_once 'DB.php';

                            //generate random code
                            $bin = file_get_contents('/dev/urandom', false,
                                null, 0, 75);

                            $random_string = bin2hex($bin);

                            $link = Config::MAIL_REDIR_URL . "?c=" .
                                $random_string;

                            require_once
                                '../library/PHPMailer/class.phpmailer.php';

                            $mailer = new PHPMailer();
                            $mailer->IsSMTP();
                            $mailer->Port = 587;
                            $mailer->AddAddress($email, $email);
                            $mailer->Host = "mail.anehosting.com";
                            $mailer->From = "facebook@osgate.org";
                            $mailer->Subject = "Authentication Test";
                            $mailer->SMTPAuth = true;
                            $mailer->SMTPDebug = 0;
                            $mailer->Username = "facebook@osgate.org";
                            $mailer->Password = "facebook.2013";
                            $mailer->Body = "Click on the link in order to
                                be able to vote: $link";

                            if (!$mailer->Send()) {
                                   return $mailer->ErrorInfo;
                            } else {
                                   //store the signed request into the db
                                   $db = new DB();
                                   $db->query("INSERT INTO mailAuthRequests
                                        (email,code) VALUES
```

```
                                                            (:email,:code)", array('email' =>
                                                            $email, 'code' => $random_string));
                                                      return 1;
                                             }
                                    } else {
                                             return -1;
                                    }
                  }
         }

}
```

7.2 Verificator Class

```
 <?php

require_once 'DB.php';

/**
 * This class is used to verify the evidence of the various IDP and to
 * show the form used to generate the certificatae
 *
 * @author snake
 */
class Verificator {

        /**
         * Associative array with the result of the DTS
         */
        private $dtsResult;
        /**
         * The public key of the DTS
         */
        private $dtsPubKey;
        /**
         * The database object
         */
        private $db;
        /**
         * Control variable used to check if a user has already a certificate
         */
        private $hasCert;

        public function __construct() {
                $this->db = new DB();
        }

        /**
         * Display the form with the captcha and eventually the certificate fields
         */
        public function showCaptchaForm() {
                echo "
                <form name='confirm' method='post' action='../pages/gen.php'>
                <table>";

        if (!$this->hasCert) {
                echo "
                        <tr><td>This fields is used to generate a
                            certificate</td>
                               <td><keygen name='pubkey'
                                   challenge='randomchars'></td></tr>";
        }

        echo "
                        <tr><td>Captcha</td><td><img id='captcha'
                            src='../library/securimage/securimage_show.php'
                            alt='CAPTCHA Image' /></td></tr>
                        <tr><td></td><td><input type='text' name='captcha_code'
                             size='10' maxlength='15' /><a href='#'
                            onclick=\"document.getElementById('captcha').src =
                            '../library/securimage/securimage_show.php?' +
                            Math.random();
                               return false\">[ Different Image ]</a></td></tr>


                        <tr><td></td><td><input type='submit' name='check'
                            value='Submit' /></td></tr>
                </table>
        </form>";
}

/**
 * Verify an evidence generated by the DTS
 * @param string $data the cleartext data
 * @param string $signature the signautre generated by the DTS
 * @return type
 */
private function verify($data, $signature) {
        if ($this->dtsPubKey == null) {
                $fp = fopen("/etc/httpd/dts.pem", "r");
                $cert = fread($fp, 8192);
                fclose($fp);
                $this->dtsPubKey = openssl_get_publickey($cert);
        }

        $res = openssl_verify($data, $signature, $cert);
        return $res;
}

/**
 * Get the evidence from the DTS
 * @param string $hashedValue the value from where we want the signature
 * @return the signed value
 */
```

```
private function getDTSValue($hashedValue) {
       $curl = curl_init('https://dts?h=' . $hashedValue . '');
       curl_setopt($curl, CURLOPT_FAILONERROR, true);
       curl_setopt($curl, CURLOPT_FOLLOWLOCATION, true);
       curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
       curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, false);
       curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);
       $res = curl_exec($curl);
       $val = explode(".", curl_exec($curl));

       $this->dtsResult = array();
       $this->dtsResult['signature'] = base64_decode($val[0]);
       $this->dtsResult['timestamp'] = $val[1];
       return $res;
}

/**
 * Verify the Facebook evidence
 * @param string $hashedValue the value to be signed from the DTS
 * @param string $evidence the clear text evidence
 * @param string $userID the id of the user
 * @return int
 * @throws Exception if there are problems validating the signature
 */
public function verifiyFacebook($hashedValue, $evidence, $email) {

       $dtsStringResult = $this->getDTSValue($hashedValue);
       $concatHash = $hashedValue . $this->dtsResult['timestamp'];

       $ok = $this->verify($concatHash, $this->dtsResult['signature']);

       if ($ok == 1) {

              // free the key from memory
              openssl_free_key($this->dtsPubKey);
              $this->db->query("SELECT UID FROM fbAuthentications WHERE UID
                  = :id", array('id' => $email));

              //only update the evidence
              if ($this->db->getRowCount() == 1) {
                     $this->db->query("UPDATE fbAuthentications SET
                         proofValue=:proof, fbValue=:fbValue WHERE UID =
                         :id", array('proof' => $dtsStringResult, 'fbValue'
                         => $evidence, 'id' => $email));
              } else { //insert a new user
                     $this->db->query("INSERT INTO fbAuthentications
                         (UID,proofValue,fbValue) VALUES
                         (:id,:proof,:fbValue)", array('id' => $email,
                         'proof' => $dtsStringResult, 'fbValue' =>
                         $evidence));
                     $this->hasCert = false;
              }
```

```
                $_SESSION['authSource'] = "fb";
                $_SESSION['email'] = $email;
        } elseif ($ok == 0) {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);

                throw new Exception("Bad signature verification");
        } else {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);

                throw new Exception("Error checking the signature from the
                    DTS");
        }

        return $ok;
}


/**
 * Verify the Google+ evidence
 * @param string $hashedValue the value to be signed from the DTS
 * @param string $evidence the clear text evidence
 * @param string $userID the id of the user
 * @return int
 * @throws Exception if there are problems validating the signature
 */
public function verifyGooglePlus($hashedValue, $evidence, $email) {

        $dtsStringResult = $this->getDTSValue($hashedValue);
        $concatHash = $hashedValue . $this->dtsResult['timestamp'];

        $ok = $this->verify($concatHash, $this->dtsResult['signature']);

        if ($ok == 1) {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);

                $this->db->query("SELECT UID FROM gplusAuthentications WHERE
                    UID = :id", array('id' => $email));

                //only update the evidence
                if ($this->db->getRowCount() == 1) {
                        $this->db->query("UPDATE gplusAuthentications SET
                            proofValue=:proof, gplusValue=:fbValue WHERE UID =
                            :id", array('proof' => $dtsStringResult, 'fbValue'
                            => $evidence, 'id' => $email));
                } else { //insert a new user
                        $this->db->query("INSERT INTO gplusAuthentications
                            (UID,proofValue,gplusValue) VALUES
                            (:id,:proof,:gplus)", array('id' => $email,
                        'proof' => $dtsStringResult, 'gplus' =>
                        $evidence));
                    $this->hasCert = false;
                }

                $_SESSION['authSource'] = "gplus";
                $_SESSION['email'] = $email;
        } elseif ($ok == 0) {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);
                throw new Exception("Bad signature verification");
        } else {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);
                throw new Exception("Error checking the signature from the
                    DTS");
        }

        return $ok;
}


/**
 * Verify the E-Mail evidence
 * @param string $hashedValue the value to be signed from the DTS
 * @param string $evidence the clear text evidence
 * @param string $authCode the authorization code
 * @return int
 * @throws Exception if there are problems validating the signature
 */
public function verifyEmail($hashedValue, $evidence, $authCode) {
        $dtsStringResult = $this->getDTSValue($hashedValue);
        $concatHash = $hashedValue . $this->dtsResult['timestamp'];

        $ok = $this->verify($concatHash, $this->dtsResult['signature']);

        if ($ok == 1) {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);

                $this->db->query("SELECT UID FROM mailAuthentications WHERE
                    UID = :mail", array('mail' => $evidence));

                //only update the evidence
                if ($this->db->getRowCount() == 1) {
                        $this->db->query("UPDATE mailAuthentications SET
                            proofValue=:proof WHERE UID = :mail",
                            array('proof' => $dtsStringResult, 'mail' =>
                            $evidence));
                } else { //insert a new user
                        //store the evidence into the db
                        $this->db->query("INSERT INTO mailAuthentications
                            (proofValue,UID) VALUES (:proof,:email)",
                            array('proof' => $dtsStringResult, 'email' =>
                            $evidence));
                        $this->db->query("UPDATE mailAuthRequests SET
                            verified=TRUE WHERE code=:code", array('code' =>
                            $authCode));
                        $this->hasCert = false;
                }

                $_SESSION['authSource'] = "email";
                $_SESSION['email'] = $evidence;
        } elseif ($ok == 0) {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);
                throw new Exception("Bad signature verification");
        } else {
                // free the key from memory
                openssl_free_key($this->dtsPubKey);
                throw new Exception("Error checking the signature from the
                    DTS");
        }

        return $ok;
}


/**
 * Verify the Shibboleth evidence
 * @param string $hashedValue the value to be signed from the DTS
 * @param string $evidence the clear text evidence
 * @param string $userID the id of the user
 * @return int
 * @throws Exception if there are problems validationg the signature
 */
public function verifyShibboleth($shibAssertion, $userID) {

        //retrieve the SAML assertion from URL
        $curl = curl_init();
        curl_setopt($curl, CURLOPT_URL, $shibAssertion);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_FAILONERROR, true);
        curl_setopt($curl, CURLOPT_FOLLOWLOCATION, true);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, false);
        curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);

        $result = curl_exec($curl);
        $doc = new DOMDocument();

        $doc->loadXML($result);
        $objXMLSecDSig = new XMLSecurityDSig();
```

```
$objDSig = $objXMLSecDSig->locateSignature($doc);
if (!$objDSig) {
       error_log("Cannot locate signature Node");
       throw new Exception("Cannot locate Signature Node");
}
$objXMLSecDSig->canonicalizeSignedInfo();

//name of the attribute that corresponds to the id of the reference
$objXMLSecDSig->idKeys = array('ID');

$retVal = $objXMLSecDSig->validateReference();

if (!$retVal) {
       error_log("Reference Validation Failed");
       throw new Exception("Reference Validation Failed");
}

$objKey = $objXMLSecDSig->locateKey();
if (!$objKey) {
       error_log("Reference Validation Failed");
       throw new Exception("We have no idea about the key");
}

$objKeyInfo = XMLSecEnc::staticLocateKeyInfo($objKey, $objDSig);

if ($objXMLSecDSig->verify($objKey)) {
       $ok = 1;
       error_log("Signature validated");
} else {
       error_log("Signature validation failed");
       throw new Exception("Signature validation failed");
}

if ($ok == 1) {
       $proofVal = base64_encode($result);

       $this->db->query("SELECT UID FROM shibAuthentications WHERE
           UID = :id", array('id' => $userID));

       //only update the evidence
       if ($this->db->getRowCount() == 1) {
              $this->db->query("UPDATE shibAuthentications SET
                  proofValue=:proof WHERE UID = :id", array('id' =>
                  $userID, 'proof' => $proofVal));
       } else { //insert a new user
              //store the evidence into the db
              $this->db->query("INSERT INTO shibAuthentications
                  (UID,proofValue) VALUES (:id,:proof)", array('id'
                  => $userID, 'proof' => $proofVal));
              $this->hasCert = false;
       }
```

```
                        $_SESSION['authSource'] = "shib";
                        $_SESSION['email'] = $userID;
                }

                return $ok;
        }

}
```

7.3 Index Page

```
<?php
require_once '../library/google-api-php-client/src/Google_Client.php';
require_once '../library/google-api-php-client/src/contrib/Google_PlusService.php';
require_once '../etc/Config.php';
require_once '../class/Welcome.php';

session_start();
$loginPortal = new Welcome();
?>

<html>
        <head>
                <meta charset="UTF-8">
                <title>Registration Portal</title>
        </head>
        <body style="font-family: 'Open Sans Light';">
                <h1 style="width: 100%; text-align: center;">Registration Portal</h1>
                <table align="center" style="padding: 0px;">
                        <tr><td style="background-color: #e5ecf9; text-align:
                            center;"><h3>Facebook</h3></td></tr>
                        <tr><td style="padding: 12px; text-align: center;"><?php
                            $loginPortal->facebookLogin() ?></td></tr>
                        <tr><td style="background-color: #e5ecf9; text-align:
                            center;"><h3>Google+</h3></td></tr>
                        <tr><td style="padding: 12px; text-align: center;"><?php
                            $loginPortal->gplusLogin() ?></td></tr>
                        <tr><td style="background-color: #e5ecf9; text-align:
                            center;"><h3>E-Mail</h3></td></tr>
                        <?php

                        echo "<tr><td>";

                                $res = $loginPortal->checkEmailSubmission();

                                if ($res == 1) {
                                        echo "An e-mail has been sent to " .
                                            mysql_real_escape_string($_POST['email']);
                                } else if ($res == -1) {
                                        echo "The email isn' valid.";
                        } else {
                                echo $res;
                        }
                echo "</td></tr>";
                ?>
                <tr><td style="padding: 12px; text-align: center;"><?php
                    $loginPortal->emailLogin() ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Shibboleth</h3></td></tr>
                <tr><td style="padding: 12px; text-align: center;"><?php
                    $loginPortal->shibbolethLogin() ?></td></tr>
        </table>
    </body>
</html>
```

7.4 Facebook Redirection Page

```
 <?php
session_start();
require_once '../library/facebook-php-sdk/src/facebook.php';
require_once '../class/Verificator.php';
require_once '../etc/Config.php';

$config = array();
$config['appId'] = Config::FACEBOOK_APP_ID;
$config['secret'] = Config::FACEBOOK_SECRET;

$facebook = new Facebook($config);
?>
<html>
        <head>
                <meta charset="UTF-8">
                <title>Facebook Authentication</title>
        </head>
        <body style="font-family: 'Open Sans Light';">


                <?php
                try {
                        //check CSFR token and access token
                        if($facebook->getAccessToken()){
                                $verificator = new Verificator();

                                // Proceed knowing you have a logged in user who's
                                    authenticated.
                                $user = $facebook->api('/me');

                                //encode and serialize the user email and user ID as
                                    evidence
                                $email = $user['email'];
                                $userid = $user['id'];
                                $userDB = base64_encode($userid . "." . $email);
```

```
                    $hashInfo = hash('sha256', $userDB);
                    $proofVer = $verificator->verifiyFacebook($hashInfo,
                        $userDB, $email);
                }
        } catch (Exception $e) {
                echo $e;
                error_log($e);
                exit;
        }
        ?>

        <h1 style="width: 100%; text-align: center;">Facebook
            Authentication</h1>
        <table align="center" style="padding: 0px;">
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Proof</h3></td></tr>
                <tr><td style="padding: 12px;"><?php echo $userid . "," .
                    $email ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Proof Verified</h3></td></tr>
                <tr><td style="padding: 12px;"><?php if ($proofVer) echo
                    "Verified";
        else echo "Not Verified"; ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Get Client Certificate</h3></td></tr>
                <tr><td style="padding: 12px;"><?php if ($proofVer)
                    $verificator->showCaptchaForm() ?></td></tr>
        </table>

        </body>
</html>
```

7.5 Google+ Redirection Page

```
 <?php
require_once '../library/google-api-php-client/src/Google_Client.php';
require_once '../library/google-api-php-client/src/contrib/Google_PlusService.php';

session_start();

require_once '../class/Verificator.php';
require_once '../etc/Config.php';

//initialize the Google SDK
$client = new Google_Client();
$client->setApplicationName('Project1');
$client->setClientId(Config::GOOGLE_CLIENT_ID);
$client->setClientSecret(Config::GOOGLE_CLIENT_SECRET);
$client->setRedirectUri(Config::GOOGLE_REDIRECT_URL);
$client->setDeveloperKey('http://localhost');
```

```
$client->setScopes(array('https://www.googleapis.com/auth/userinfo.email',
    'https://www.googleapis.com/auth/userinfo.profile'));

$plus = new Google_PlusService($client);
?>
<html>
       <head>
              <meta charset = "UTF-8">
              <title>Google+ Authentication</title>
       </head>
       <body style="font-family: 'Open Sans Light';">
              <?php
              //after a succesful authentication with google we are redirected here
                   with a "code" and "state" parameter
              if (isset($_GET['code']) && $_GET['state'] ==
                  $_SESSION['gplus_state']) {
                     try {
                            $client->authenticate($_GET['code']);
                            $verificator = new Verificator();

                            $token = json_decode($client->getAccessToken());
                            $request = new
                                Google_HttpRequest("https://www.googleapis.com/oauth2/v1/userinfo?alt=json
                            $resp =
                                $client->getIo()->authenticatedRequest($request);
                            $userInfoJson = json_decode($resp->getResponseBody());
                            $userid = $userInfoJson->{'id'};
                            $email = $userInfoJson->{'email'};
                            $gplusValue = base64_encode($userid . "." . $email);

                            //get signature from DTS
                            $hashInfo = hash('sha256', $gplusValue);
                            $proofVer = $verificator->verifyGooglePlus($hashInfo,
                                $gplusValue, $userInfoJson->email);
                     } catch (Exception $e) {
                            echo $e;
                            error_log($e);
                     }
              } else {
                     echo "CSRF token state doesn't match.";
                     error_log("CSRF token state doesn't match.");
              }
              ?>

              <h1 style="width: 100%; text-align: center;">Google+
                  Authentication</h1>
              <table align="center" style="padding: 0px;">
                     <tr><td style="background-color: #e5ecf9; text-align:
                         center;"><h3>Proof</h3></td></tr>
                     <tr><td style="padding: 12px;"><?php echo $email . "," .
                         $userid ?></td></tr>
```

```
                        <tr><td style="background-color: #e5ecf9; text-align:
                            center;"><h3>Proof Verified</h3></td></tr>
                        <tr><td style="padding: 12px;"><?php
                                        if ($proofVer)
                                                echo "Verified";
                                        else
                                                echo "Not Verified";
                                        ?></td></tr>
                        <tr><td style="background-color: #e5ecf9; text-align:
                            center;"><h3>Get Client Certificate</h3></td></tr>
                        <tr><td style="padding: 12px;"><?php if ($proofVer)
                            $verificator->showCaptchaForm() ?></td></tr>
                </table>
        </body>
</html>
```

7.6 Shibboleth Redirection Page

```
 <?php
require_once '../library/xmlseclibs/xmlseclibs.php';
require_once '../etc/Config.php';
require_once '../class/Verificator.php';
session_start();
?>
<html>
        <head>
                <meta charset="UTF-8">
                <title>Shibboleth Authentication</title>
        </head>
        <body style="font-family: 'Open Sans Light';">
                <?php
                if (isset($_SERVER['Shib-Assertion-01']) &&
                    isset($_SESSION['shib_state']) && $_SESSION['shib_state'] ==
                    $_GET['state']) {
                        try {
                                //retrieve the assertion UR
                                $shibAssertion = $_SERVER['Shib-Assertion-01'];
                                $verificator = new Verificator();
                                $proofVer =
                                    $verificator->verifyShibboleth($shibAssertion,
                                    $_SERVER['eppn']);
                        } catch (Exception $e) {
                                echo $e;
                                error_log($e);
                        }
                } else {
                        echo "CSRF token state doesn't match.";
                        error_log("CSRF token state doesn't match.");
                }
                ?>
                <h1 style="width: 100%; text-align: center;">Shibboleth
                    Authentication</h1>
            <table align="center" style="padding: 0px;">
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Proof</h3></td></tr>
                <tr><td style="padding: 12px;"><?php echo "SAML Assertion
                    (".$_SERVER['Shib-Assertion-01'].")"; ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Proof Verified</h3></td></tr>
                <tr><td style="padding: 12px;"><?php if($proofVer) echo
                    "Verified"; else echo "Not Verified"; ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Get Client Certificate</h3></td></tr>
                <tr><td style="padding: 12px;"><?php if($proofVer)
                    $verificator->showCaptchaForm () ?></td></tr>
            </table>
        </body>
</html>
```

7.7 Mail Redirection Page

```
 <?php
session_start();
require_once '../class/Verificator.php';
require_once '../class/DB.php';

$db = new DB();
$code = $_GET['c'];

$conn = new PDO('mysql:host=localhost;dbname=project1', 'root', 'adminroot');

//check if the code exists in the database
$stmt = $conn->prepare("SELECT email,code,requestTime,verified FROM mailAuthRequests
     WHERE code = :code");
$stmt->execute(array('code' => $code));
?>
<html>
        <head>
                <meta charset="UTF-8">
                <title>Mail Authentication</title>
        </head>
        <body style="font-family: 'Open Sans Light';">
                <?php
                if ($stmt->rowCount() == 1) {
                        //now check the time
                        $result = $stmt->fetch(PDO::FETCH_ASSOC);

                        date_default_timezone_set("Europe/Zurich");
                        $storedTime = new DateTime($result['requestTime']);
                        $actualTime = new DateTime(date("Y-m-d H:i:s"));
                        $difference = $storedTime->diff($actualTime);
                        $expired = false;
                //is the code is older than 30 minutes or has already been
                    verified is "expired"
                if ($difference->format("%i") > 30 || $result['verified']) {
                        $expired = true;
                } else {
                        $verificator = new Verificator();
                        //get evidence from the DTS
                        $hashEmail = hash('sha256', $result['email']);
                        $proofVer = $verificator->verifyEmail($hashEmail,
                            $result['email'], $code);
                        $update = $conn->prepare("UPDATE mailAuthRequests SET
                            verified = TRUE WHERE code = :code");
                        $update->execute(array('code' => $code));
                }
        }
        ?>

        <h1 style="width: 100%; text-align: center;">Mail Authentication</h1>
        <table align="center" style="padding: 0px;">
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Validity</h3></td></tr>
                <tr><td style="padding: 12px;"><?php if($expired) echo "This
                    authentication token has expired (validity 30 minutes).";
                    else echo "Token valid."; ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Proof</h3></td></tr>
                <tr><td style="padding: 12px;"><?php echo $result['email'];
                    ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Proof Verified</h3></td></tr>
                <tr><td style="padding: 12px;"><?php if($proofVer) echo
                    "Verified"; else echo "Not Verified"; ?></td></tr>
                <tr><td style="background-color: #e5ecf9; text-align:
                    center;"><h3>Get Client Certificate</h3></td></tr>
                <tr><td style="padding: 12px;"><?php if($proofVer)
                    $verificator->showCaptchaForm () ?></td></tr>
        </table>
    </body>
</html>
```

7.8 Certificate Generation Page

```
 <?php

session_start();

require_once "../library/securimage/securimage.php";
require_once '../class/DB.php';

$secureImage = new Securimage();
$db = new DB();
```

```
//First check the captcha
if (isset($_POST['captcha_code'])) {
	$val = $secureImage->check($_POST['captcha_code']);

	if (!$val) {
		echo "The security code entered was incorrect.<br /><br />";
		echo "Please try again.";
		exit;
	}
}

//check if variable needed to the generation of the certificate have been set
if (isset($_REQUEST['pubkey']) && isset($_SESSION['email']) &&
    isset($_SESSION['authSource'])) {

	$authSource = array();
	$authSource['shib'] = "shibAuthentications";
	$authSource['email'] = "mailAuthentications";
	$authSource['fb'] = "fbAuthentications";
	$authSource['gplus'] = "gplusAuthentications";

	$table = $authSource[$_SESSION['authSource']];

	//check if a user already has a certificate
	$resource = $db->query("SELECT certPath FROM :table WHERE UID = :uid",
	    array('table' => $table, 'uid' => $_SESSION['email']));

	$rowCount = $db->getRowCount();

	if ($rowCount == 0) {
		//the user doesn't have a certificate
		$CAmail = $_SESSION['email'];
		$CAorg = "RISIS";
		$CAcountry = "CH";
		$CAcity = "Bern";
		$CAlocality = "Biel";

		$key = $_REQUEST['pubkey'];
		str_replace(str_split(" \t\n\r\0\x0B"), '', $key);
		$keyreq = "SPKAC=" . str_replace(str_split(" \t\n\r\0\x0B"), '',
		    $key);
		$keyreq .= "\nCN=" . $CAmail;
		$keyreq .= "\nemailAddress=" . $CAmail;
		$keyreq .= "\n0.OU=" . $CAorg . " client certificate";
		$keyreq .= "\norganizationName=" . $CAorg;
		$keyreq .= "\ncountryName=" . $CAcountry;
		$keyreq .= "\nstateOrProvinceName=" . $CAcity;
		$keyreq .= "\nlocalityName=" . $CAlocality;

		$opensslconf = "/etc/ssl/openssl.cnf";
		$days = '180';
```

```
            $certfolder = "/tmp/";
            $capw = "ciao";

            //path of the user certificate
            $serial = trim(file_get_contents("/etc/ssl/myCA/serial"));

            file_put_contents($certfolder . $username . ".spkac", $keyreq);

            $command = "export HOME=\"/etc/ssl/myCA\" && /usr/bin/openssl ca
                -config " . $opensslconf . " -days " . $days . " -notext -batch
                -spkac " . $certfolder . $username . ".spkac -out " . $certfolder
                 . $username . " -passin pass:'" . $capw . "' 2>&1";

            error_log(shell_exec($command));

            $certName = $serial . ".pem";

            //apparently we cannot use a token to specify the table in UPDATEso
                we concatenate the name directly in the query string
            $db->query("UPDATE $table SET certPath = :cert WHERE UID = :uid",
                array('cert' => $certName, 'uid' => $_SESSION['UID']));

            //send back the certificate to the user
            $length = filesize($certfolder . $username); // $cerfolder.$uniq =
                /tmp/4k9Bal
            header('Last-Modified: ' . date('r+b'));
            header('Accept-Ranges: bytes');
            header('Content-Length: ' . $length);
            header('Content-Type: application/x-x509-user-cert');
            readfile($certfolder . $username);
            exit;
        }
}
?>
```

7.9 DTS

```
 <?php

//get the data we have to sign
if (isset($_GET['h'])) {

        $fp = fopen("/etc/httpd/privkeydts.pem", "r");
        $priv_key = fread($fp, 8192);
        fclose($fp);
        $pkeyid = openssl_get_privatekey($priv_key);

        $timestamp = time();
        $data = $_GET['h'] . $timestamp;

        //sign the data with our private key
        openssl_sign($data, $signature, $pkeyid);
        openssl_free_key($pkeyid);

        //send back the signature along with a timestamp
        echo base64_encode($signature) . "." . $timestamp;
}

?>
```