THESIS PAPER

IMPLEMENTATION OF A SINGLE SIGN ON BASED AUTHETICATION SYSTEM

by

TAHNIAT ASHRAF (12301054)

SYED SABBIR AHMED (12341004)

SUPERVISOR : AMITABHA CHAKRABARTY

ABSTRACT

The purpose of the thesis study is to understand Single Sign On authentication system, investigate the infrastructure of a Single Sign On based system and to implement it on a local machine. Single Sign On gives user the ability to enter his id and password once and log on to multiple applications within an enterprise. In the thesis study, Shibboleth and CAS were chosen as the primary tools to understand the functionality of Single Sign On. A Single Sign On login system was implemented in the local machine using CAS. The implementation and analysis will serve as the basis for a detailed study and future development of Single Sign On.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

SSO – Single Sign On

IdP – Identity Provider

SP – Service Provider

CAS – Central authentication server

JDK – Java Development Kit

JRE – Java Runtime Environment

SAML – Security Assertion Markup Language

ACKNOWLEDGEMENTS

It gives us great pleasure in acknowledging the support and help of our thesis superviser Mr. Amitabha Chakrabarty , our department chairperson Dr. Mumit Khan and last but not the least, our senior brother Mr. Mohibuzzaman Zico. Without the help of these persons, it wouldn't have been possible to come this far in our thesis project.

I. INTRODUCTION

A. MOTIVATION

From the very first day, our intension was to work in network security. Single Sign On is clearly a thing that matches our interest. This is an authentication process which enhances the security system in protected or secured web based application or resources where a third party relationship is attached.

B. RESEARCH AND GOAL

We are doing a research on Single Sign On. CAS and Shibboleth are the two software we are using to understand the Single Sign On system. Both software implements web based Single Sign On. In our thesis project, we have tried to implement the two softwares in intension to find the bottlenecks of the system.

C. ORGANIZATION OF THESIS

This thesis is organized into five sections. In the second section we have introduced Single Sign On. This section consists of benefits and architecture of Single Sign On. In the third section we have introduced Shibboleth and how it works. After that, description of CAS and its functionality is attached. Implementation of the CAS comes in the fourth section. In the last section we have mentioned our future plans.

II. SINGLE SIGN ON SYSTEM DESCRIPTION

A. BACKGROUND INFORMATION

In the present world, users typically have to log into multiple systems using different sets of credentials. It gets very difficult for a user to remember the different sets of username and password combinations. Users tend to forget these informations easily and make help desk calls in order to request for an account reset. System administrators are having a tough time too. There is a huge amount of workflow involved with creating and managing user accounts. Administrators have to manage these user's accounts so that they are accessed in a coordinated manner that does not affect the integrity of the security policies.

To overcome this challenge - Single Sign On (SSO) system is introduced. In Single Sign On, user enters his credential once and gets access to multiple applications. This takes away the need of remembering multiple username-password combinations and thus makes life easier for both the user and administrator.

B. WHAT IS SINGLE SIGN ON

Single Sign On is an authentication process for a client-server relationship where the user can enter their credentials once and gain the right to access to more than one

application or number of resources within an organization or enterprise. Single Sign on doesn't require the user to re-enter the credentials for authentication for switching the applications under the same organization.

Single Sign On gives the concept of allowing the access control to independent systems. This method allows user to authenticate once and gain access to all systems without needing a new login. The reversed process of Single Sign On system is Single Sign Off. It performs a logout procedure where the user will lose all rights of access to all the systems. The most common Single Sign On configurations involve the use of Smart Cards, OTP tokens, and softwares like CAS, Shibboleth, and Kerberos etc.

In a single sign-on platform, the user performs a single initial (or primary) sign-on to an identity provider trusted by the applications he wants to access. Later on, each time he wants to access an application, it automatically verifies that he is properly authenticated by the identity provider without requiring any direct user interaction. A well designed and implemented single sign-on solution significantly reduces authentication infrastructure and identity management complexity, consequently decreasing costs while increasing security.

Usability as well as security in the domain has increased, after the introduction of SSO. Along with this, the effort to manage the user accounts has also reduced. A system like SSO, which provides an integration and coordination of different accounts, is beneficiary for both the user and administrator. For example:

- Reduction of time exposure for users as they no longer have to sign-on to different domains
- Improved usability by reducing the number of account information that a user has to remember

Single Sign On enables an end-user to access other secondary security domains after signing on to a primary domain. Between these domains there exists a trust relationship. Therefore, the second domain obtains user credentials through the sign-on service from the first domain where the user is already authenticated. In other words, the primary security domain supports the other domains by the user's credentials assumed that the user is logged in correctly.

C. SINGLE SIGN ON BENEFITS

Some of the benefits of Single sign on are given below.

- Reduction in the time taken by users in sign-on operations to individual domains, including reducing the possibility of such sign-on operations failing.
- Improved security through the reduced need for a user to handle and remember multiple sets of authentication information.
- Improved security through the enhanced ability of system administrators to maintain the integrity of user account configuration including the ability to inhibit or remove an individual user's access to all system resources in a co-ordinated and consistent manner.
- Security on all levels of entry or exit or access to systems without the inconvenience of re-prompting users.
- End to end user audit sessions to improve security reporting and auditing.
- Removes application developers from having to understand and implement identity security in their applications.

- Reduces phishing success, because users are not trained to enter password everywhere without thinking.
- Reducing password fatigue from different user name and password combinations.
- Reducing time spent re-entering passwords for the same identity.
- Can support conventional authentication such as Windows credentials (i.e., user name/password).
- Reducing IT costs due to lower number of IT help desk calls about passwords.
- Centralized reporting for compliance adherence.
- Ability to enforce uniform enterprise authentication and/or authorization policies across the enterprise.

Mainly with Single Sign On, users' and administrators' lives become much easier as they will have to deal with a single digital identity for each user. User will have to provide digital identity only once. This will increase user's productivity. The maintenance of authentication data and enforcement of authentication policies become much easier with Single Sign On, since authentications data will be centralized. Moreover, Single Sign On reduces the chance that users will forget or lose their digital identities; therefore it reduces the risk of compromising a security system.

But the sad part of the story is that Single Sign On has a disadvantage; namely the "Key to Kingdom argument"[9]. This argument means that if attackers manage to compromise a Single Sign On authentication data store, they will gain access to all users' digital identities. Moreover, if attackers manage to obtain a single user's digital identity, they will gain access to all services protected by it. Despite the Key to Kingdom argument, Single Sign On can be made extremely secure with careful planning, implementation and

administration. In fact, SSO is considered a valuable and indispensable security product in the overall IT security system nowadays.

D. SINGLE SIGN ON ARCHITECTURE

A simple Single Sign On architecture deals with a single authentication authority. In simple SSO architectures, we can have a single authentication server with a single credentials database as shown in Figure 1.

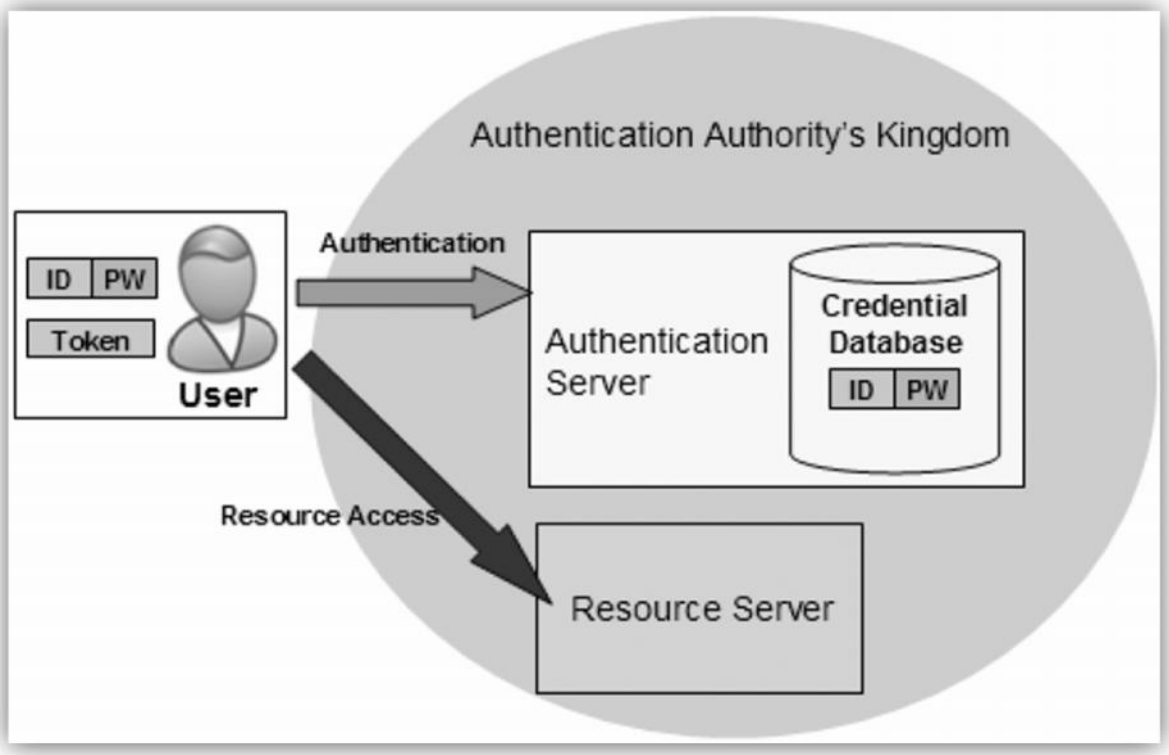


Figure 1 : A simple SSO architecture in an environment with a single authentication[10]

We can also have multiple authentication servers with multiple replicated credentials databases as shown in Figure 2.

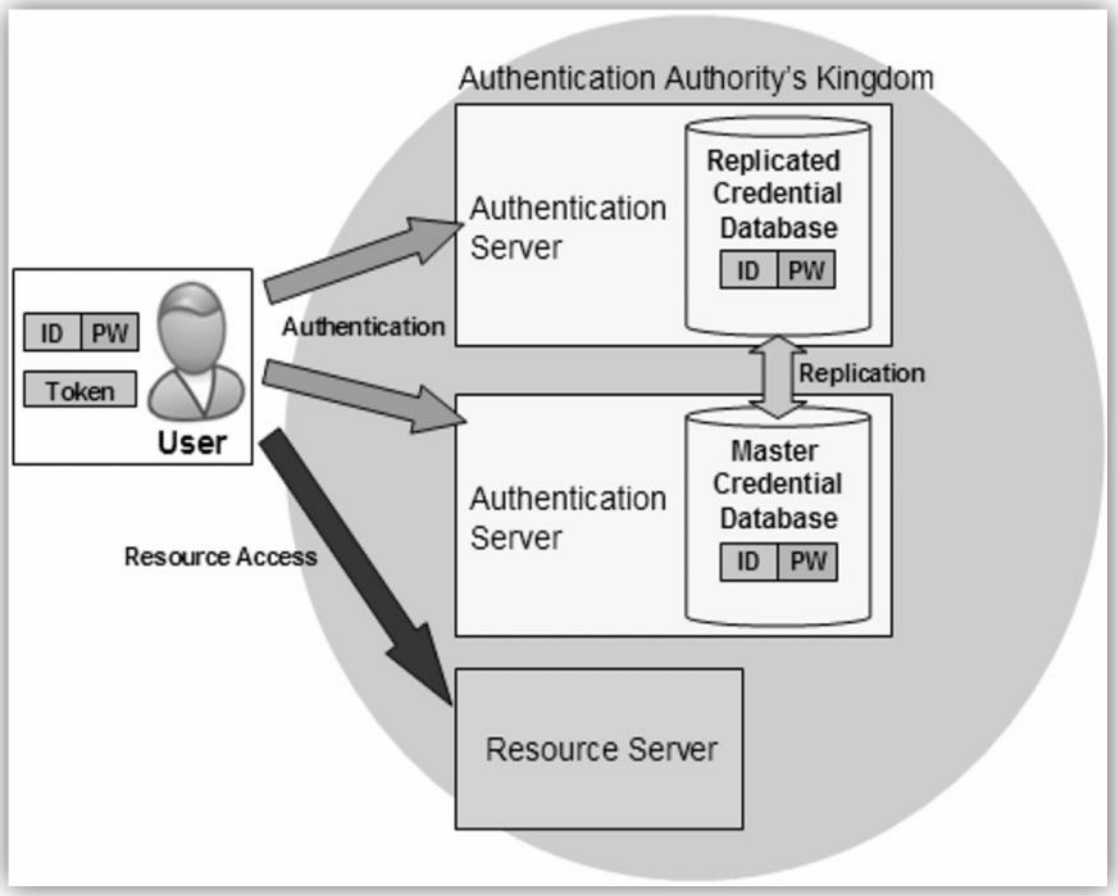


Figure 2. A simple SSO architecture in an environment with a multiple authentication[10]

It is can be seen that the second architecture provides better performance, scalability and availability. It is notable how in both figures a user has a single set of credentials and how he submits them to the authentication authority. The authentication authority validates the credentials using the data stored in the credentials database. If there is a match, the user's identity is considered authentic and he is granted access to the resource server. Later, the authentication authority will issue a token to the user. This token is basically an encrypted string or metadata that proves that the user was authenticated by the authentication authority. Simple Single Sign On architectures can be easily implemented in networks where all computers are running the same operating system.

E. DIFFERENT TYPES OF SINGLE SIGN ON

There are three main types of single sign-on: web SSO, Legacy SSO and Federated SSO. [11]

WEB SINGLE SIGN ON

Wed-based SSO is a widely deployed single sign-on technology, sometimes also called as web access management. It enables a user to provide the credentials, and if authentication succeeds it will establish a relationship of trust that will grant user the access to all web resources for which he/she have permissions.

LEGACY SINGLE SIGN ON

Legacy SSO is also called Enterprise SSO. Like web SSO, legacy SSO is also a technology designed to manage multiple login to target applications after a single authentication event. It has a very similar structure to the web SSO. While web SSO only manages the web-based service, legacy SSO extends the SSO functionality to the traditional legacy applications and network resources (windows GUI based applications, for example), typically within an enterprise's internal network.

FEDERATED SINGLE SIGN ON

Federated SSO is similar to the web SSO but has a much broader concept. It uses Simple Object Access Protocol (SOAP) and Security Assertion Markup Language (SAML) to enables users to sign on once into a member of the affiliated group of organizations, then seamlessly access all the web sites within that trusted federation without requiring re-authentication.

The main advantage of a federated SSO is extending the SSO environment from a user's home domain to other foreign domains. Federated SSO allows the enterprises to maintain the control of its local services and expose these resources to a larger class of users not directly administered by it. Mostly, this solution is used by the businesses to build a complete framework. The most famous federated SSO is the Liberty Alliance Project.

F. THE BENEFITS OF SINGLE SIGN ON FROM MULTIPLE VIEW

- From the user view: In an SSO environment, users only need to authenticate themselves once. This effectively solves the annoying stop-and-go problem which is caused by multiple login requests. Best of all, the SSO solution frees users from remembering a large number of identities and associated passwords.
- From the user view: In an SSO environment, users only need to authenticate themselves once. This effectively solves the annoying stop-and-go problem which is caused by multiple login requests. Best of all, the SSO solution frees users from remembering a large number of identities and associated passwords.
- From the user view: In an SSO environment, users only need to authenticate themselves once. This effectively solves the annoying stop-and-go problem which is caused by multiple login requests. Best of all, the SSO solution frees users from remembering a large number of identities and associated passwords.
- From the user view: In an SSO environment, users only need to authenticate themselves once. This effectively solves the annoying stop-and-go problem which is caused by multiple login requests. Best of all, the SSO solution frees users from remembering a large number of identities and associated passwords.
- Potential Increase in Security: With only one password to remember, it is more reasonable for the user to choose a single complex and more secure password instead of using multiple simple and insecure passwords. This potentially increases the system security.

- Improve Productivity: Employee productivity is dramatically improved, with less time users spend logging into multiple applications and recovering the forgotten passwords.
- Reduction in Costs: “Meta Group estimates 33% reduction in help desk volume when using an enterprise Single Sign-On solution.” By reducing the number of passwords the user must remember, SSO effectively reduces the password-related workload to the help desk and lowers the costs associated with managing passwords across multiple distributed applications.

III.VARIOUS TOOLS USED IN SINGLE SIGN ON IMPLEMENTATION

A. SHIBBOLETH

In present, the whole IT world is scared of identity theft. To prevent this; the idea of Single Sign on has emerged. To implement this, enterprises creates many softwares. Some are commercials, some are open source and some are for only private uses. Shibboleth is an open source software created to implement Single Sign On system most efficiently and widely. This software's popularity gained because of its characteristics, as its IDP section is written in JAVA and SP section is written in C and C++ (the most common languages used in today’s world), the rich library of documentation, enhanced wiki with the whole installation and configuration process, availability of community and

developers resources, it maintain the OASIS standards, faced and handled the newly challenged problem in authentication system in SSO and many things. Actually Shibboleth is widely used in today's world as it has stood tall in front of all the challenges thrown by SSO.

Actually Shibboleth System is a standard based, open source software package for web single sign-on across or within organizational boundaries. It allows sites to make informed authorization decisions for individual access of protected online resources in a privacy-preserving manner (from: http://shibboleth.net/about/index.html). This system is the mechanism which works behind the scene that allow users to access the secure sites of the other institutes or organizations or agencies through using their own credentials that already exists. This systems works upon two steps mainly. Firstly user may be authenticating by their respective institutes and secondly requested institutes authenticate and determine their access, based on the attributes released. In other hand after user has been authenticated to one site within a federation, they can visit other sites for which they are authorized within that federation without having to authenticate again. This privilege lasts until users close their session or the session expires.

So, Shibboleth is a free open source implementation for identity management, providing a web-based single sign-on mechanism across different organizational boundaries. It is a federated system, supporting secure access to resources across security domains. Information about a user is sent from a home identity provider (IdP) to a service provider (SP) which prepares the information for protection of sensitive content and use by applications. Providing a federated single sign-on and attribute exchange framework, Shibboleth also provides extended privacy functionality allowing the browser

user and their home site to control the attributes released to each application. Using Shibboleth-enabled access simplifies management of identity and permissions for organizations supporting users and applications.

The issues that will be addressed as a challenge to overcome to implement Single Sign On by Shibboleth specifically are given below.

- Multiple passwords required for multiple applications.
- Scaling the account management of multiple applications.
- Security issues associated with accessing third-party services
- Privacy.
- Interoperability within and across organizational boundaries.
- Enabling institutions to choose their authentication technology.
- Enabling service providers to control access to their resources.
- Facilitating the rapid and effective integration of disparate third-party services (e.g. cloud computing applications), leveraging campus identity management and trust services.

B. HOW SHIBBOLETH WORKS

Shibboleth; the web based Single Sign On system is a way of exchanging information between an organization and a provider. By using Shibboleth, the information is exchanged in a secure manner, protecting both the security of the data and the privacy of the individual. Shibboleth relies on four main performers.

- User representative Web Browser
- Access restricted contains- Resource
- Identity Providers which authenticate the user
- Single sign On process performers- Service Provider

At first the user wants to access the protected resource. Being for the first time the user does not have the actual proper valid session. By discovering this resource sends the user to service provider to start the Single Sign On process. Then the user comes to the service provider who prepares the authentication request. Then service provider sends both the authentication request along with the user to the identity provider. Now on the arrival at the identity provider, first it checks if the user has a valid existing session. If not the identity provider first authenticate them by prompting the desired credentials and then checks it. After checking for validity has been done or the user has already the valid existing session; the identity provider prepares an authentication response and sends it along with user back to the service provider. When user arrives to the service provider with the authentication response, it validates the response and creates a session for the user while retrieving some information about the user from the response and makes that available to the resource. So now the user has the valid session and he can access the

protected resource as the resource now knows about the user. This is how the whole Shibboleth system works.

As for example, Let's assume, a user wants to access a protected resource hosted in the URL "https://www.resource- x.com". Now, resource-x must have the „Shibboleth SP" software installed in its server. When the user tries to access "https://www.resource-x.com" from his browser, the shibboleth sp software, installed in the server side of resource-x, intercepts the request. As resource-x has no knowledge regarding the user's home organization, the user's browser gets redirected to the „Discovery service". Typically, the user is provided with a list of organizations with which resource-x has trust-relationship". Every organization in this list, must have the „shibboleth IdP" installed in their server. After the user selects his home organization, the browser first gets redirected to resource-x first. Then, resource-x sends an authentication request to the selected organization and submits it through the user browser to the home organization. Now the user sees the familiar interface, where he enters his user name and password and submits it to the home organization's IdP. Idp checks the credentials and upon successful authentication, it creates an assertion (SAML) containing users attributes. This assertion is known as meta data". When resource-x comes into „trust-relationship" with an organization, it notifies the organization regarding the attributes it needs in the meta data. Now, the IdP of users home organization creates the meta data releasing only those attributes, which SP of resource-x needs .After the assertion is being created, users browser gets redirected to the prior requested resource, resource-x. Finally, SP of resource-x checks the assertion and user gets access to the desired content.

C. METADATA, ATTRIBUTES & SAML

Security Assertion Markup Language (SAML) is an XML standard that allows secure web domains to exchange user authentication and authorization data. Using SAML, an online service provider can contact a separate online identity provider to authenticate users who are trying to access secure content.

Attributes are specific bits of information about people. They include such things as a person's name and email address. There are established attributes with specific names that are used for federated identity management. Also, institutions can create their own attributes.

Attributes can be used:

* To determine if someone is authorized to use a particular service. For example, a particular service might be provided only to faculty members.

* To customize services for people after they have logged in. For example, once one is logged in, a service page may greet him by name and show him a customized menu based on what he is authorized to use.

Metadata is a document where the type of the URl for communicating between service provider and identity provider has been written. Not only that, function needed for communication and their various technical aspects also there. Meta data is written in SAML.

The metadata for an identity provider or service provider usually contains the following information:

- a unique identifier, known as an entity id
- a human-readable name and description
- a list of URLs to which messages should be delivered and some information about when to use each
- cryptographic information used when creating and verifying messages

B.CENTRAL AUTHENTICALTION SYSTEM (CAS)

The Central Authentication Service (CAS) is a single sign-on protocol for the web. It allows a user to access multiple applications by providing their credentials only once. It also enables web applications to authenticate users without having to handle a user's security credentials, such as a password. The name CAS also refers to a software package that implements this protocol. CAS was originally created by Yale University.

The CAS protocol involves at least three parties: a client web browser, the web application re-questing authentication, and the CAS server. It may also involve a back-end service, such as a database server, that does not have its own HTTP interface but communicates with a web application.

When the client visits an application desiring to authenticate to it, the application redirects it to CAS. CAS validates the client's authenticity, usually by checking a username and password against a database (such as Kerberos or Active Directory).

If the authentication succeeds, CAS returns the client to the application, passing along a security ticket. The application then validates the ticket by contacting CAS over a secure connection and providing its own service identifier and the ticket. CAS then gives the application trusted information about whether a particular user has successfully authenticated.

The following are the main steps of authentication:

- The user attempts to access an application using its URL. The user is redirected to the CAS login URL over an HTTPS connection, passing the name of the requested service as a parameter. The user is presented with a username/password dialog box.
- The user enters his credential and CAS attempts to authenticate the user. If authentication fails, the target application won't be loaded and it won't have any clue that the user was trying to access it. The user remains at the CAS server.
- If authentication succeeds, then CAS redirects the user back to the target application, appending a parameter called a ticket to the URL. CAS then attempts to create an in-memory cookie called a ticket-granting cookie. This is done to allow for automatic re-authentication later. If the cookie is present, then it indicates that the user has already successfully logged in and the user avoids having to re-enter his username and password.
- The application then validates that this is a correct ticket and represents a valid user by calling the CAS serviceValidate URL by opening an HTTPS connection and passing the ticket and service name as parameters. CAS checks that the supplied ticket is valid and is associated with the requested service. If validation is successful, CAS returns the username to the application. [8]

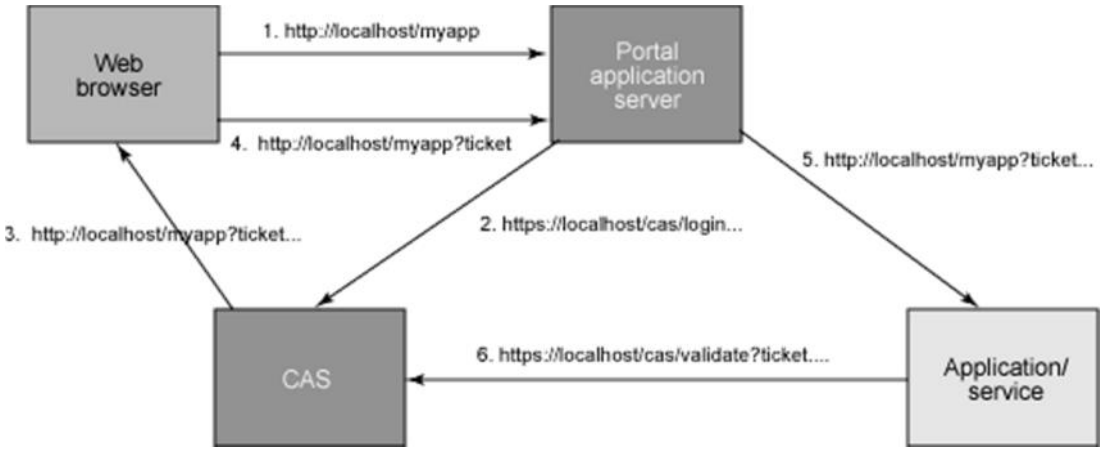


Figure 3. How the CAS protocol performs authentication [8]

IV.OUR IMPLEMENTATION OF A SSO SYSTEM USING CAS

To implement a SSO system, we had to choose between CAS and Shibboleth. We went for CAS because of its simplicity in structure and availability of good learning materials.

For the setup of CAS authentication system in a local machine which is running on windows 7 OS, we followed the guidelines provided by CAS on their official website. It can be found on: https://wiki.jasig.org/display/CASUM/Demo .

The tutorial demonstrates how to CASify web applications, in other words how to protect web applications with CAS. If a web application is configured to be protected by CAS, user has to authenticate himself first to CAS to be able to view the application. Once the user authenticates himself to CAS server, user needs no further sign-ins to visit all the other applications within the same domain. This behavior replicates Single Sign On.

In our setup, we have used Java sdk 1.6 (update 33), tomcat 5.5 and CAS server release version 3.3.5.

A. STEPS OF THE SETUP PROCEDURE

These are the steps of the whole setup procedure:

- First, Java sdk 1.6 (update 33) was downloaded from java.sun.com and then was installed on the machine.

- JAVA_HOME and JRE_HOME environmental variables were pointed to the right directories. In our case, ‘jdk’ was installed on D:\Java\jdk and ‘jre’ was installed on D:\Java\jre directory. Environmental variables were set to their actual path .In our case, for JAVA_HOME it was set to D:\Java\jdk and for JRE_HOME it was set to D:\Java\jre.

- This step shows how to self-author a server certificate using Java Keytool.

To guard an application with CAS it requires a secure connection or SSL.CAS Server necessitates SSL to operate. Certificates are exchanged as part of the SSL (also called TLS) initialization that occurs when any Browser connects to an https: Web site. A certain number of public CA certificates are pre-installed in each Browser by Microsoft, Mozilla, Google or who-ever else makes the Browser. The same set of certificates is installed by Microsoft in every copy of Windows and by Sun in every copy of Java. However, no application, system, or language comes with any certificate that one has created inside his company or personal computer as a Certificate Authority (CA).

When a Browser connects to CAS over an https: URL, the Server identifies itself by sending its own certificate. For CAS to function, the Browser must already have installed a certificate identifying and trusting the CA that issued the CAS Server certificate. If the Browser is not already prepared to trust the CAS server, then an error message pops up saying the server is not trusted. Certificates can be purchased from various commercial certificate authorizers like VeriSign or Thawte. We have used Java Keytool service to generate a self-authenticated certificate.

Java Keytool is a key and certificate management utility. It allows users to manage their own public/private key pairs and certificates. Java Keytool stores the keys and certificates in a keystore. By default the Java keystore is implemented as a file. It protects private keys with a password. A Keytool keystore contains the private key and any

certificates necessary to complete a chain of trust and establish the trustworthiness of the primary certificate. [1]

Below are listed some of the most common Java Keytool commands:

Java Keytool Commands for Creating and Importing –

- **Generate a Java keystore and key pair**

keytool -genkey -alias domain-name -keyalg RSA -keystore keystore.jks -keysize 2048

- **Generate a certificate signing request (CSR) for an existing Java keystore**

keytool -certreq -alias domain-name -keystore keystore.jks -file mydomain.csr

- **Import a root or intermediate CA certificate to an existing Java keystore**

keytool -import -trustcacerts -alias root -file Thawte.crt -keystore keystore.jks

- **Import a signed primary certificate to an existing Java keystore**

keytool -import -trustcacerts -alias mydomain -file domain-name.crt -keystore keystore.jks

- **Generate a keystore and self-signed certificate**

keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass password -validity 360 -keysize 2048 [2]

Some key terms and their meaning:

- RSA is the algorithm used to generate the cryptographic keys, corresponding to the certificate.
- Validity is the number of days the certificate will stay valid. One can enter more than 365.
- Alias is the tool for which keytool and certificate will be generated. [3]

In our case, we typed in the commands listed below from the command prompt to generate the keystore and the self-signed certificate:

```
D:\>cd Java

D:\Java>cd jdk

D:\Java\jdk>cd bin

D:\Java\jdk\bin>keytool -genkey -alias tomcat -keypass changeit -keyalg RSA
Enter keystore password:  changeit
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:  bu
What is the name of your organization?
  [Unknown]:  bu
What is the name of your City or Locality?
  [Unknown]:  dhk
What is the name of your State or Province?
  [Unknown]:  dhk
What is the two-letter country code for this unit?
  [Unknown]:  bd
Is CN=localhost, OU=bu, O=bu, L=dhk, ST=dhk, C=bd correct?
  [no]:  yes

D:\Java\jdk\bin>keytool -export -alias tomcat -keypass changeit -file server.crt
Enter keystore password: changeit
Certificate stored in file <server.crt>

D:\Java\jdk\bin>keytool -import -file server.crt -keypass changeit -keystore ..\jre\lib\security\cacerts
Enter keystore password: changeit
Owner: CN=localhost, OU=bu, O=bu, L=dhk, ST=dhk, C=bd
Issuer: CN=localhost, OU=bu, O=bu, L=dhk, ST=dhk, C=bd
Serial number: 462030d8
Valid from: Fri Jul 13 15:39:36 HST 2012 until: Thu Aug 12 15:39:36 HST 2012
Certificate fingerprints:
MD5: CC:3B:FB:FB:AE:12:AD:FB:3E:D 5:98:CB:2E:3B:0A:AD
SHA1: A1:16:80:68:39:C7:58:EA:2F:48:59:AA:1D:73:5F:56:78:CE:A4:CE
Trust this certificate? [no]: yes
Certificate was added to keystore

D:\Java\jdk\bin>
```

Figure 3. Commands to generate the keystore and the self-signed certificate

- Tomcat was downloaded from http://tomcat.apache.org/download-55.cgi#5.5.2 .It was installed on D:\tomcat5. When prompted for JRE, changed default to %JAVA_HOME%\jre (in our case – D:\Java\jdk\jre), as this should be the home of new 'cacerts' – created in previous step. Environmental variable %CATALINA_HOME% was set to D:\tomcat5
- To configure Tomcat to be SSL enabled, server.xml was opened up and connector element for port 8443 (SSL- by default) was uncommented. Later on , the parameters for keystoreFile, keystorePass, truststoreFile was added as shown below :

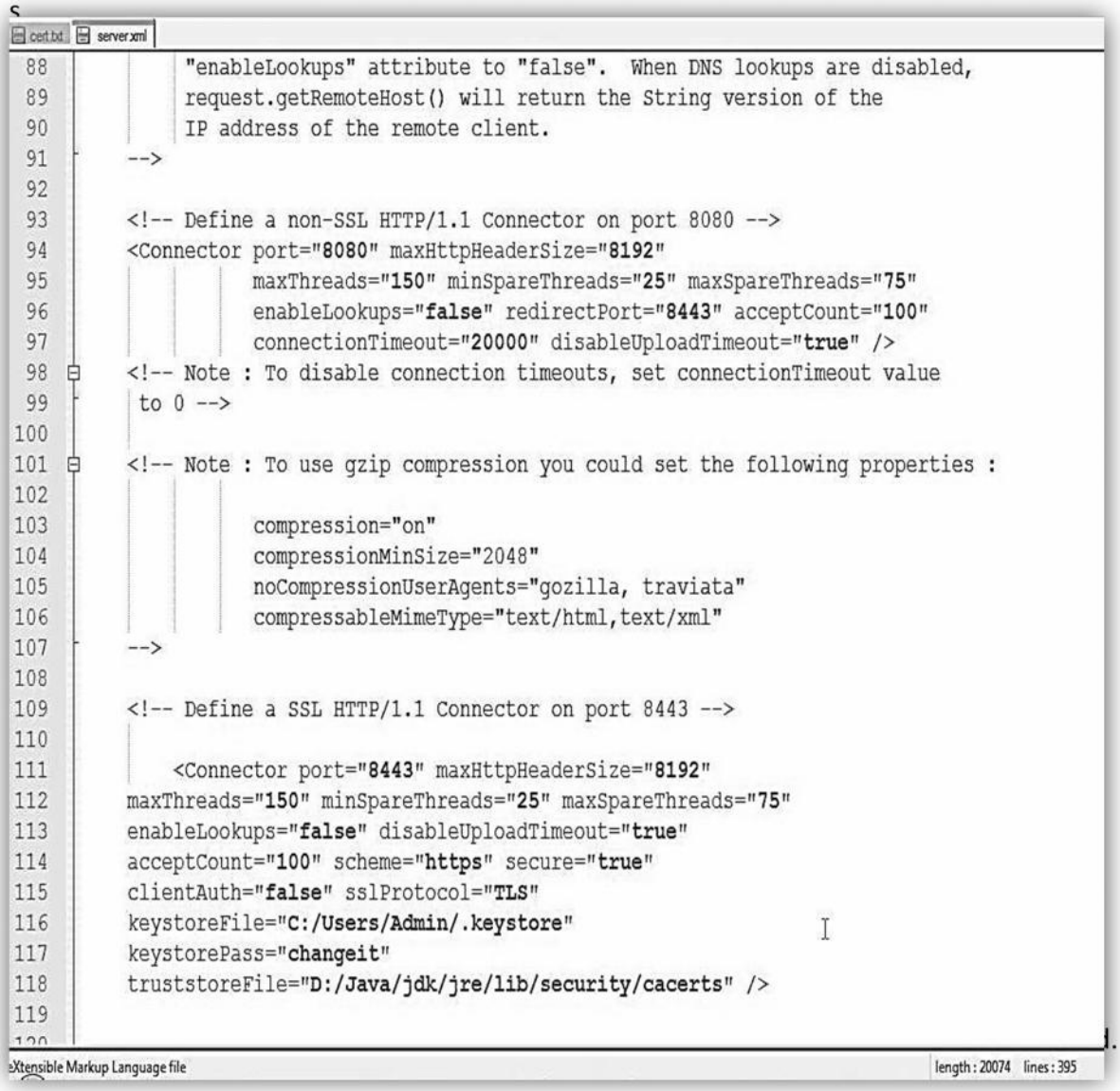
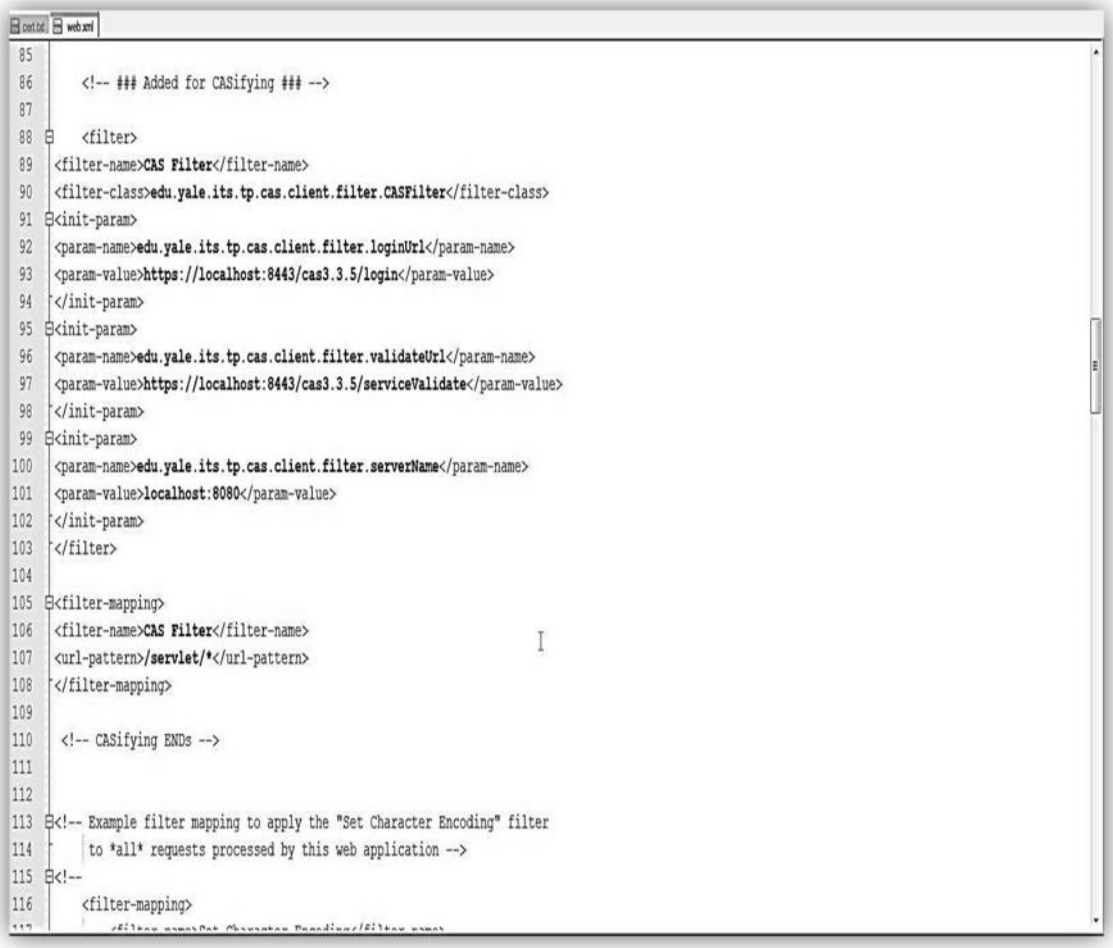


Figure 4. Parameters for keystoreFile, keystorePass, truststoreFiles

- A cas-server-webapp-3.3.5.war file was found inside the modules folder. It was renamed to cas3.3.5.war. cas3.3.5.war was then copied to %CATALINA_HOME%\ webapps (D:\tomcat5\webapps).These deployed CAS in tomcat and a folder

named cas3.3.5 was created inside the %CATALINA_HOME%\ webapps (D:\tomcat5\webapps) directory.

- Casclient-2.1.1.zip was downloaded from http://www.java2s.com/Code/Jar/c/Downloadcasclient2111jar.htm. The zip file was extracted and casclient-2.1.1 jar was copied to %CATALINA_HOME%\ webapps \servlets-examples \ WEB-INF\lib (D:\tomcat5\webapps\servlets-examples\WEB-INF\lib).
- Tomcat comes with some default servlets. These can be found under %CATALINA_HOME%\ webapps\servlets – examples .To Casify the default servlets inside %CATALINA_HOME%\ webapps\servlets – examples , these lines were added to web.xml of the servlets-examples context :



```
85
86      <!-- ### Added for CASifying ### -->
87
88      <filter>
89   <filter-name>CAS Filter</filter-name>
90   <filter-class>edu.yale.its.tp.cas.client.filter.CASFilter</filter-class>
91   <init-param>
92   <param-name>edu.yale.its.tp.cas.client.filter.loginUrl</param-name>
93   <param-value>https://localhost:8443/cas3.3.5/login</param-value>
94   </init-param>
95   <init-param>
96   <param-name>edu.yale.its.tp.cas.client.filter.validateUrl</param-name>
97   <param-value>https://localhost:8443/cas3.3.5/serviceValidate</param-value>
98   </init-param>
99   <init-param>
100  <param-name>edu.yale.its.tp.cas.client.filter.serverName</param-name>
101  <param-value>localhost:8080</param-value>
102  </init-param>
103  </filter>
104
105  <filter-mapping>
106  <filter-name>CAS Filter</filter-name>
107  <url-pattern>/servlet/*</url-pattern>
108  </filter-mapping>
109
110   <!-- CASifying ENDs -->
111
112
113  <!-- Example filter mapping to apply the "Set Character Encoding" filter
114       to *all* requests processed by this web application -->
115  <!--
116      <filter-mapping>
```

Figure 5. Edited web.xml

The table of basic CAS-filter init-param names and their definitions is given below:

init-param name	usage
edu.yale.its.tp.cas.client.filter.loginUrl	The URL whereat CAS offers its Login page. e.g. https://localhost:8443/cas3.3.5/login
edu.yale.its.tp.cas.client.filter.validateUrl	The URL whereat CAS offers its service ticket or proxy ticket validation service. e.g. https://localhost:8443/cas3.3.5/serviceValidate or https://localhost:8443/cas3.3.5/proxyValidate .
edu.yale.its.tp.cas.client.filter.serverName	This parameter specifies the server name and port of the service being filtered (not of the CAS Server itself). E.g. localhost: 8080. Either this parameter or the serviceUrl parameter must be set.
edu.yale.its.tp.cas.client.filter.serviceUrl	This parameter replaces the serverName parameter above. It becomes the URL that CAS redirects to after login. Either this parameter or the serverName parameter must be set.

Table 1. Basic CAS-filter init-param names and their definitions [4]

- Now that Tomcat and CAS was ready, Tomcat was stopped and all the logs were cleared. Tomcat was restarted and all the logs were examined carefully. They appeared normal.
- http://localhost:8080/servlets-examples gives the list of default servlets that tomcat comes with. E.g. – Hello World, Request Info etc.

- http://localhost:8080/servlets-examples/servlet/HelloWorldExample was browsed from the browser. User was redirected to the CAS login page. Similarly http://localhost:8080/servlets-examples/servlet/RequestInfoExample also redirected to the CAS login page.

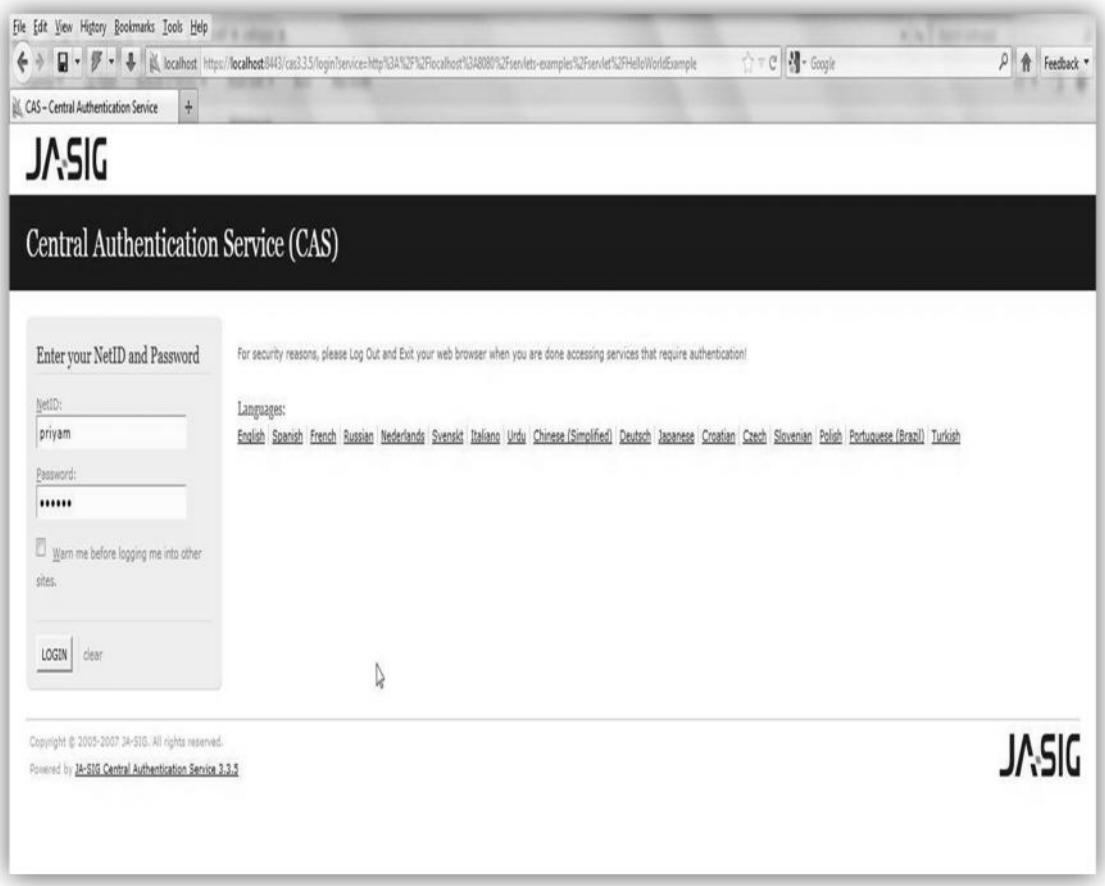


Figure 6. CAS login page

- http://localhost:8080/servlets-examples/servlet/HelloWorldExample was reloaded and redirected to the CAS login page. Any set of same username – password combination (E.g. – Admin / Admin) was entered to login. Login was a success and the page redirected to the original request.

Figure 7. Hello World!

- http://localhost:8080/servlets-examples/servlet/RequestInfoExample was reloaded and it bypassed the login process to CAS, as a valid session already existed, and loaded the original request.

This behavior replicates the one of 'Single Sign On'. Thus CAS was successfully set up in the local machine using Java sdk 1.6, tomcat 5 and CAS server release 3.3.5.

V. LIST OF REFERENCES

[1] X.509 Certificates Authentication Handler, definition of java keytool. Available at : https://wiki.jasig.org/display/CASUM/X.509+Certificates

[2] Java Keytool Commands for Creating and Importing. Available at: http://nl.globalsign.com/en/support/ssl+certificates/java/java+based+webserver/keytool+commands/

[3] How to generate a digital certificate using key tool; the commonly used terms. Available at: http://www.digisigner.com/how_to_generate_digital_certificate.html#how_to_generate_digital_certificate_using_keytool

[4] Using CAS filter; Required CASFilter init-params. Available at : https://wiki.jasig.org/display/CASC/Using+CASFilter

[5] Description of keytool and certificate generation .Available at : http://docs.oracle.com/javase/1.5.0/docs/tooldocs/solaris/keytool.html

[6] Demo CAS setup .Available at: https://wiki.jasig.org/display/CASUM/Demo

[7] Central Authentication Service .Available at: http://en.wikipedia.org/wiki/Central_Authentication_Service

[8] CAS protocol authentication overview. Available at: http://www.ibm.com/developerworks/web/library/wa-singlesign/

[9] Developing a Single Sign On system, A java based authentication platform aimed at the web. Henrik Jervivad.

[10] J. De Clercq and G. Grillenmeier. Microsoft Windows Security Fundamentals. Elsevier, Oxford, UK, 2007.

[11] Web Single Sign-On System -For WRL Company.Si Xiong.June 2005.

A. OTHERS

[1] How Single Sign On works .Available at: http://shibboleth.net/about/basic.html

[2] How Shibboleth works: Advanced concept. Available at: http://shibboleth.net/about/advanced.html

[3] SAML 2.0. Available at: http://en.wikipedia.org/wiki/SAML_2.0

[4] OASIS security services. Available at: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[5] IdP provider metadata. Available at: https://wiki.shibboleth.net/confluence/display/SHIB2/IdPMetadataProvider