Master’s Thesis
submitted in partial fulfilment of the
requirements for the course “Applied Computer Science”

Protocol Independent Federated Authentication and Authorization in Shibboleth

Rashmi Barbate

Gesellschaft für wissenschaftliche
Datenverarbeitung mbH Göttingen (GWDG)

Bachelor’s and Master’s Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

01. June 2015
I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 01. June 2015
Masterarbeit
im Studiengang „Angewandte Informatik“

Protocol Independent Federated Authentication and Authorization in Shibboleth

Rashmi Barbate

01 June 2015

Supervisor
Prof. Dr. Ramin Yahyapour
Department for Practical Informatics
Georg-August-Universität Göttingen

Second supervisor
Dr. Sven Bingert
Gesellschaft für wissenschaftliche Datenverarbeitung mbH
Göttingen
Abstract

Authentication and authorization are an integral part of the federated identity management system. This work aims to authenticate and authorize users from different identity management system with one single Shibboleth system. A service can then be accessed independent of the identity system of the users home organization. In this thesis I developed an extension so that the Shibboleth identity provider acts here as a proxy that redirects the login request to other identity providers independent of the communication protocol.
# Contents

1. **Introduction** .................................................. 1  
   1.1. Motivation ................................................. 2

2. **Foundation** ............................................... 4  
   2.1. Shibboleth .................................................. 4  
      2.1.1. Shibboleth Service Provider ......................... 5  
      2.1.2. Shibboleth Identity Provider ....................... 6  
      2.1.3. WAYF .................................................. 7  
      2.1.4. Flow of data and control in Shibboleth .......... 8  
   2.2. OpenID .................................................... 9  
      2.2.1. Attribute Exchange .................................. 10  
   2.3. OAuth ..................................................... 12  
      2.3.1. OpenID VS OAuth pseudo Authentication .......... 18  
   2.4. Eduroam Basics ........................................... 19  
      2.4.1. Radius protocol ..................................... 19  
      2.4.2. Extensible Authentication Protocol ............... 22  
      2.4.3. Eduroam vs Other web based protocols(OpenID and OAuth) .................................................................................. 23

3. **Identity Management Bridge** ............................ 24  
   3.1. IMB Benefits ................................................. 24  
   3.2. Requirement Analysis ....................................... 26  
   3.3. Possible solutions .......................................... 27  
      3.3.1. Inter Identity Management system communication ........................................... 28  
      3.3.2. Identity Provider discovery ......................... 29  
      3.3.3. User interaction ........................................ 30  
      3.3.4. Security of users data .................................. 32  
      3.3.5. Data transfer protocol ................................. 32  
      3.3.6. Re-authentications and post authentication data .................................................. 32  
      3.3.7. Maintenance of authorization data after authentication .................................................. 33  
   3.4. Use Case ..................................................... 34  
   3.5. Sequence Diagram ........................................... 34

4. **Implementation** ............................................ 36  
   4.1. Other identity provider details ......................... 36  
      4.1.1. OpenID provider ........................................... 36  
      4.1.2. OAuth provider ........................................... 38  
      4.1.3. Eduroam Provider ......................................... 39
Contents

4.2. Implementation Design & Components ........................................... 41
4.3. IMB plug-in development .......................................................... 43
  4.3.1. System Configuration used ..................................................... 43
  4.3.2. Shibboleth username/password login handler extension ............... 44
  4.3.3. Shibboleth Data connector extension ...................................... 61

5. Existing work .............................................................................. 64
  5.1. IMB plug-in vs eduGAIN ............................................................ 64
  5.2. IMB plug-in vs Umbrellaid.org ............................................... 67

6. Test Results .................................................................................. 69

7. Conclusion & Future Work ............................................................. 71

References ....................................................................................... 71

A. Installation Guides ................................................................ ...... 74
  A.1. Shibboleth 2.x Installation ......................................................... 74
    A.1.1. Basic installation ................................................................. 74
    A.1.2. Shibboleth Identity provider Installation & Configuration ........ 74
    A.1.3. Shibboleth Identity provider source code installation ............. 75
    A.1.4. Problem while building in Shibboleth 2.4.3 ......................... 76
    A.1.5. Problem with resulting jar file in Shibboleth 2.4.3 ................. 76
    A.1.6. Analysis for solution of problem: ........................................ 76
    A.1.7. Service provider installation ............................................... 77
  A.2. SimpleID installation ............................................................... 79
    A.2.1. System requirements .......................................................... 79
    A.2.2. Download & Installation ...................................................... 79
List of Figures

1.1. Institution X's user A accesses data from GWDG .......................... 2
1.2. IDEA: users from different IDP login via Shibboleth IDP .............. 3

2.1. Shibboleth Architecture ................................................................. 5
2.2. Shibboleth control flow ............................................................... 8
2.3. OpenID basic flow ................................................................. 9
2.4. OAuth basic flow[1] ............................................................... 17
2.5. OpenID & OAuth protocol[2] .................................................. 18

3.1. Possible communication scenarios .............................................. 29
3.2. User input ................................................................................... 31
3.3. Sequence Diagram ................................................................. 35

4.1. Detailed design[6] ................................................................. 43
4.2. Web Socket Communication ................................................. 58
4.3. HTTP Communication ........................................................... 60

5.1. eduGain structure ................................................................. 64
5.2. eduGain working ................................................................. 65
5.3. This Project idea ................................................................. 66
5.4. Umbrella .................................................................................. 67
5.5. Project Idea .............................................................................. 68

6.1. Test result: Authentication with Shibboleth .............................. 69
6.2. Test result: Shibboleth Authentication failure and OpenID success . 70
6.3. Test result: Shibboleth Auth failure, OpenID failure, Eduroam Success . 70
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Fetch attribute request parameter list</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>OpenID fetch request example</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>OpenID fetch response example</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Authorization grant: Authorization code request parameter</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>Authorization grant: Authorization code response parameter</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Authorization grant: Authorization code error response</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>Authorization grant: Access token request parameters</td>
<td>16</td>
</tr>
<tr>
<td>2.8</td>
<td>Authorization grant: Access token response parameters</td>
<td>16</td>
</tr>
<tr>
<td>2.9</td>
<td>Packet message</td>
<td>22</td>
</tr>
<tr>
<td>2.10</td>
<td>Radius message packet format</td>
<td>22</td>
</tr>
<tr>
<td>2.11</td>
<td>EAP messages</td>
<td>23</td>
</tr>
<tr>
<td>2.12</td>
<td>EAP packet format</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Identity file location information</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>XRDS document snippet</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Test Eapol_test</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Eapol_test configuration</td>
<td>40</td>
</tr>
<tr>
<td>4.5</td>
<td>System Details</td>
<td>44</td>
</tr>
<tr>
<td>4.6</td>
<td>SimpleID server location response</td>
<td>46</td>
</tr>
<tr>
<td>4.7</td>
<td>Query String</td>
<td>47</td>
</tr>
<tr>
<td>4.8</td>
<td>Name value pair: post request</td>
<td>47</td>
</tr>
<tr>
<td>4.9</td>
<td>SimpleID user’s Attributes</td>
<td>48</td>
</tr>
<tr>
<td>4.10</td>
<td>html form for user consent</td>
<td>49</td>
</tr>
<tr>
<td>4.11</td>
<td>Query String</td>
<td>50</td>
</tr>
<tr>
<td>4.12</td>
<td>Post request parameter</td>
<td>50</td>
</tr>
<tr>
<td>4.13</td>
<td>Authorization code Response</td>
<td>50</td>
</tr>
<tr>
<td>4.14</td>
<td>Access token</td>
<td>51</td>
</tr>
<tr>
<td>4.15</td>
<td>EAP authentication response</td>
<td>54</td>
</tr>
<tr>
<td>4.16</td>
<td>List Of Attributes in Access Accept packet</td>
<td>61</td>
</tr>
<tr>
<td>A.1</td>
<td>System Requirements for Shibboleth Installation</td>
<td>74</td>
</tr>
<tr>
<td>A.2</td>
<td>Shibboleth IdP configuration changes</td>
<td>76</td>
</tr>
<tr>
<td>A.3</td>
<td>Shibboleth IdP configuration changes</td>
<td>78</td>
</tr>
<tr>
<td>A.4</td>
<td>Server Requirements</td>
<td>79</td>
</tr>
<tr>
<td>A.5</td>
<td>Browser Requirements</td>
<td>79</td>
</tr>
</tbody>
</table>
List of Code Snippet

4.1. Modified web.xml ................................................. 44
4.2. SimpleID user authentication flow .............................. 45
4.3. Post request for user data ...................................... 46
4.4. parsing post response ........................................... 48
4.5. Google authentication code flow ............................... 51
4.6. configuration file snippet ....................................... 53
4.7. Jradius Authentication .......................................... 54
4.8. BaseDataConnector extension snippet ......................... 61
4.9. Attribute resolver.xml .......................................... 62
4.10. attribute-resolver.xml resolver tag ......................... 62
4.11. attribute-resolver.xml dataconnector ....................... 63
**Nomenclature**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application program interface is used for integrating new features into existing software.</td>
</tr>
<tr>
<td>CHAP</td>
<td>Challenge-Handshake Authentication Protocol</td>
</tr>
<tr>
<td>EAP</td>
<td>Extensible Authentication Protocol</td>
</tr>
<tr>
<td>EAPOL</td>
<td>Extensible Authentication Protocol over LAN</td>
</tr>
<tr>
<td>Eduroam</td>
<td>Education Roaming</td>
</tr>
<tr>
<td>GWDG</td>
<td>Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen</td>
</tr>
<tr>
<td>IDP</td>
<td>Identity Provider</td>
</tr>
<tr>
<td>IMB</td>
<td>Identity Management Bridge</td>
</tr>
<tr>
<td>JAAS</td>
<td>Java Authentication and Authorization Service</td>
</tr>
<tr>
<td>NAS</td>
<td>Network Access Server</td>
</tr>
<tr>
<td>PAP</td>
<td>Password Authentication Protocol</td>
</tr>
<tr>
<td>PPP</td>
<td>Point to Point Protocol</td>
</tr>
<tr>
<td>Radius</td>
<td>Remote Authentication Dial In User Service</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SVN</td>
<td>Sub-Version is a popular version control system used for collaborative project development.</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WPA</td>
<td>Wi-fi Protected Access</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XRDS</td>
<td>eXtensible Resource Descriptor Sequence</td>
</tr>
</tbody>
</table>
## Glossary

<table>
<thead>
<tr>
<th>Terms</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAP</td>
<td>PAP was originally specified for use with PPP in RFC 1334. PAP transmits the user name and password across the network unencrypted</td>
</tr>
<tr>
<td><strong>Authentication</strong></td>
<td>The process of identifying an individual, usually based on a username and password</td>
</tr>
<tr>
<td><strong>Authorization</strong></td>
<td>The process of giving individuals access to system objects based on their defined role</td>
</tr>
<tr>
<td>CHAP</td>
<td>CHAP was originally designed to use with PPP. It is specified in RFC 1994. The authentication server challenges the client, and the client proves that it is in possession of the shared secret by successfully responding to the challenge</td>
</tr>
<tr>
<td>MS-CHAP</td>
<td>MS-CHAP was designed by Microsoft to offer similar functionality to CHAP, but with enhanced functionality for Windows systems. Unlike CHAP, MS-CHAP does not require that the shared secret is stored in cleartext at both ends of the link.</td>
</tr>
<tr>
<td>MS-CHAP-V2</td>
<td>MS-CHAP-V2, which was initially introduced with Windows 2000 and documented in RFC 2759, addressed the shortcomings of MS-CHAP by eliminating the weak encoding of passwords for older clients, providing mutual authentication, and improving keying and key generation.</td>
</tr>
<tr>
<td>EAP-MD5</td>
<td>EAP-MD5 was standardized along with EAP in RFC 2284. Its basic structure is similar to that of CHAP. Like CHAP, it requires that the MD5 password hash on both ends be available</td>
</tr>
<tr>
<td>EAP-GTC</td>
<td>EAP-GTC was standardized along with EAP in RFC 2284. Like PAP, EAP-GTC allows the exchange of cleartext authentication credentials across the network</td>
</tr>
<tr>
<td>EAP-SIM</td>
<td>Used for authentication and session key distribution using the Subscriber Identity Module (SIM) from the Global System for Mobile Communications (GSM). GSM cellular networks uses a subscriber identity module (SIM) card to carry out user authentication.</td>
</tr>
</tbody>
</table>
1. Introduction

Today in the cloud era, users from an organization need to access data and services (e.g.: virtual machine, cloud storage) provided by other organizations. This happens because the user’s home organization relies on several other organizations for the data and services. Consequently, the user is required to create accounts in each organization (on which their home organization depend) to access their data and services confidentially and securely. Hence, there arises an obligatory demand for the user’s to manage plenty of usernames and passwords related to each organization’s account efficiently. The demand is complex for users as they might get confused and mingle the different passwords. Also storage of the password in excel sheets or files instead of storing them in mind can be insecure[8]. To tackle the above problem, users should be able to use data/services from all organizations (on which his home organization depend) using single username and password. This led to the introduction of the federated identity management system[9]. Federated identity management system allows the user to use same identity information (username and password) to access all the resources of the home and other federated organizations. Also, it allows sharing of resources securely, seamlessly and privately without the need to have individual accounts for accessing the respective services[10].

There are numerous federated identity management systems available in the market[11]. Each federated infrastructure may consist of multiple identity providers (system that stores user’s identity information) and service providers (system that provides the service to the user). All these providers communicate with each other using common protocols, e.g. Shibboleth identity management system’s providers communicate with SAML protocol. The most common protocol[1] used by federated identity management system are SAML, OAuth, OpenID, Radius, Diameter etc.

Each organization use their own federated identity management system. Suppose, an organization using SAML based federated identity management system wants to collaborate with an organization using OpenID based federated identity management system. In such a situation multi-protocol[2] federated identity management system needs to be developed. This thesis discovers the various possibilities of such multi-protocol federations.

1 HTTP://en.wikipedia.org/wiki/Federated_identity
2 HTTP://en.wikipedia.org/wiki/Authentication_protocol
1.1. Motivation

Figure 1.1 shows that the user A from institution M wants to login and use the service X provided by GWDG institute. GWDG institute uses Shibboleth system for providing services. Institution M uses the OpenID provider. Therefore, the user A’s identity information is stored in OpenID system. GWDG institute has collaboration with institution M. Due to the collaboration, GWDG institute should be able to provide the service X to user A. In other words, user A’s identity information from institution M should be accepted by GWDG institute to achieve the goal of multi-protocol federated identity management.

![Diagram](image)

Figure 1.1.: Institution X’s user A accesses data from GWDG

The multi-protocol federation can be achieved by the development of identity management bridge (IMB) between institution M and GWDG institute. This IMB converts the SAML information from GWDG to OpenID understandable information for institution M and vice-versa. This conversion leads to successful communication between the two federated systems. With this federation, GWDG system can retrieve user A’s identity information from institution X to authenticate and authorize user A. Hence, the aim of this thesis is to build the IMB plug-in to establish successful communication between GWDG system and institutes using other protocol based federated identity management.

3http://www.gwdg.de
management solution. This will lead to the exchange of user data and attribute information between various federated identity management systems without the need to have separate identity management system supporting individual protocols.

Figure 1.2 describes an infrastructure for multi-protocol based identity management system to be built as an end result for this thesis work. GWDG portal should successfully authenticate and authorize the authentic users from other federated identity management system. The user’s data and identity information is stored in their local institution’s provider database.

This thesis will consider Shibboleth, OpenID, OAuth and Radius providers for multi-protocol federation with Shibboleth identity management system. The users can store their identity information in any of the above mentioned providers supported by their home institution or whom the user can trust on. The user A, user B, user C and user D can login and use the Shibboleth service A, service B and service C depending on their access rights. Hence, this thesis will deal with multi-protocol based user authentication (login) and authorization (provision of services on basis of access rights).
2. Foundation

This thesis considers Shibboleth, OpenID, OAuth and Radius providers for multi-protocol federation with the Shibboleth system. This section describes the basic working and characteristics of each system and protocols mentioned above. This section also emphasizes on authentication flow and attribute exchange information flow after authorization for each protocol. The attributes are the user’s other details (such as middle name, sex, date of birth etc.) that are stored with the identity information in the identity provider. Authentication takes place with the user’s basic information such as username and password. After authentication, the service may require user’s other details to display or store any user related data. For e.g. an email service may require user’s contact list from home organization. This process of fetching the user’s contact list from their home organization after authentication is called attribute exchange. The home organization will decide whether to make the user’s contact list accessible to Shibboleth system. This decision process is called authorization.

2.1. Shibboleth

Shibboleth is an open-source implementation developed by internet2 which provides authentication and authorization services to federated organizations using Security Assertion Markup Language (SAML) protocol. In this section we will discuss the architecture of Shibboleth. The two main components of Shibboleth involved in web based single sign-on and attribute exchange are the identity provider() and the service provider(). Both components comprise of many different core sub-components as described in Fig. 2.1 and further explained in following sub sections. The architecture explained in following sub-sections has been derived from the official Shibboleth architecture documentation.

---

1 HTTP://www.internet2.edu/
2.1.1. Shibboleth Service Provider

The Shibboleth service provider (SP) resides at the organization hosting the service. Service provider’s primary task is to protect unauthorized access to a service, specifically an URL and its contents in the web server. Service provider resides in the same server as the HTTP server (preferably Apache) hosting the service. All provided technical details, discussions and development work has been performed using Shibboleth SP version 2.5.4.

Service Provider consists of the following main components:

1. Assertion Consumer Service: It is an URL/URLs (HTTP resource) hosted by the Apache server which acts as receptor of HTML form submissions and eventually redirects the user to a requested resource. The secure context is established through new user session which is further handled by Shibboleth service provider software.

2. Session Initiator: Used for generating authentication request between a user’s browser connecting to Shibboleth service provider and extending the request to Shibboleth identity provider.

https://webauth.service.ohio-state.edu/~shibboleth/shire.html
3. DS (Discovery Service): Primary task of discovery service is to facilitate automatic/manual selection of shibboleth identity provider which is serving a single shibboleth service provider.

4. SAML POST: Part of assertion consumer service that handles the received SAML assertion from shibboleth identity provider.

5. Shib Module: It is a module build to facilitate shibboleth service provider integration with Apache or any other HTTP server.

6. Shib Daemon: Dedicated service running in the background in an operating system to invoke shibboleth service provider related calls and services.

7. Web Resource: The web resource which resides in HTTP server and protected by shibboleth service provider.

2.1.2. Shibboleth Identity Provider

The shibboleth identity provider is the entity responsible for authenticating users and release of attributes. The authentication information and attributes are together bundled in a SAML assertion which is sent to the service provider. All provided technical details, discussions and development work has been performed using shibboleth IDP version 2.4.3. The components involved in the authentication and authorization process by shibboleth identity provider are:

1. Authentication Engine/Authority: It is a SAML based service responsible for authenticating the user’s credentials and issuing an authentication assertion for users to be able to access the respective shibboleth service provider. Authentication engine does not directly authenticate the user but uses the defined authentication handler for the given user. The authentication handler information is set by a identity provider administrator during installation. For more information please refer to installation guide in appendix A.

2. Attribute Authority: SAML based service used for processing attribute query requests and issuing attribute assertions to service providers with mutual authentication. Further task include filtering of attributes before release based on service provider specific filtering policy within the federation.

3. Attribute DB: It is the database or application used in the backend for storing attributes belonging to users. Multiple backend storage systems are already supported by shibboleth identity provider. The most popular are LDAP and Windows Active Directory. Other extension based backend support could be provided for MySQL, MongoDB[14], etc. Generally speaking, any attribute providing system can be ported into shibboleth identity provider provided that a java connector API exists for the specific application.
4. Authentication DB: It is the database used for storage of authentication information of users. Shibboleth identity provider provides active support for authentication via LDAP server for username/password based authentication. Third party extensions are required to provide authentication support for MySQL. Some other third party authentication extension could be obtained from contribution page of shibboleth.

5. Single Sign-On (SSO) service/profile: It is an HTTP resource contained within shibboleth identity provider that receives and processes authentication requests sent from user’s browser via shibboleth service provider. Authentication process is initiated by SSO service, further it helps in issuing proper HTTP responses to user’s browser after interacting with authentication authority.

6. Username Password Authentication: It is one of the many authentication handler types available in shibboleth which displays the authentication form after redirecting to identity provider from service provider, takes the user’s credentials for authentication and sends the data to the authentication authority via HTTP POST. Other available authentication handlers are IP address based authentication, Remote authentication, etc. Any other authentication handler other then Username Password authentication is out of scope of this research paper. Interested people can refer more at types of authentication in shibboleth identity provider.

2.1.3. WAYF

“Where Are You From” is an optional service in shibboleth which sits in between shibboleth service provider and identity provider. Its task is to discover correct identity provider for incoming requests from service provider. Mostly, the shibboleth service provider contains an in-built WAYF service. In our case WAYF is already implemented in shibboleth service provider and a more extended version of it is available as shibboleth discovery service. Going deeper into this topic is not required as it is not relevant for this thesis work. Interested people can read more about it at Shibboleth Discovery Service wiki at official page for shibboleth documentation.
2.1.4. Flow of data and control in Shibboleth

Figure 2.2. Shibboleth control flow

Figure 2.2 shows the complete flow of data and control between the various parts of Shibboleth as discussed in previous sections. Following is the explanation of Shibboleth control and flow where the numbers correspond to the process number in figure 2.2.

1. User accesses the service through a web browser, the service which is protected by Shibboleth service provider checks if the user is authenticated to access the service.

2. Session initiator creates authentication request and sends it to the browser with information about how to reach Shibboleth identity provider (URL).

3. Browser sends the request to Shibboleth identity provider using HTTP POST. Identity provider checks if user is authenticated or not, SSO service will take appropriate steps for selection of proper authentication handler.

4. Web browser gets redirected to username/password login handler. The selected authentication handler is username/password authentication handler. Appropriate web form is displayed to the user for authentication.
5. User enters authentication information and presses submit. The browser sends HTTP POST with the authentication data to the authentication engine. After successful authentication, attribute authority proceeds with attribute resolution and filtering. A SAML assertion is prepared containing an authentication statement and an attribute statement containing user’s attributes.

6. The SAML assertion is sent as HTTP POST from Shibboleth identity provider to Shibboleth service provider. Service provider processes the SAML assertion, releases attributes after service provider side filtering and redirects the control towards the requested resource at the beginning by the user.

7. After redirect, Shibboleth service provider checks again whether the user is authenticated, and finds the user as authenticated this time. Finally, user is provided access to the resource. The web page related to the service requested is displayed in user’s browser.

2.2. OpenID

OpenID is a standard protocol for user authentication[15]. In OpenID, the communication between the relying party and the identity provider takes place with the help of URI, HTTP, SSL.

The user enters the OpenID URL in the web form given by OpenID consumer. OpenID consumer fetches the identity URL entered by the user. It then retrieves the location of OpenID provider with the help of the fetched identity URL. The consumer sends the
associate request to the OpenID provider. OpenID provider sends the associate handle response to the consumer. Both, the consumer and provider have the shared secret. The consumer then redirects the end user’s query string by attaching nonce and associate handle to the provider. The end user then enters the password in provider. Followed by this, the provider redirects end user to the consumer URL with a query string such as a session. With the session in place the end user will be successfully logged in.

Figure 2.3 shows the detailed working of OpenID protocol. User visits a website, the webserver provides a login form to the user. Through this form, web server asks the users to enter their OpenID URL. User then provides his OpenID URL and submits the web form. After getting the OpenID URL the web server performs OpenID server discovery based on the user’s OpenID URL provided. The OpenID server gives back the XRDS document specifying its location and URL to the web server. Web server then sends a login request to the OpenID server URL for the user. User enters his credentials (username, password) in OpenID server URL web page. OpenID server authenticates the user and the user is successfully logged in. OpenID server sends user identity to the web server to confirm about authenticity of user. User can now access the services from the web server.

OpenID provider is secure to store private identity information of the user. The general security concern may arise when the user enters the OpenID URL; the malicious attacker can visit the entered URL and can get the user’s username and password. This is called phishing attack. The OpenID provider provides the security from phishing attacks by providing a nonce, SSL and certificates. Also provider keeps track of the site using the identity provider for authentication. Hence, there is strong trust factor between user and OpenID provider.

2.2.1. Attribute Exchange

An OpenID attribute[16] is an unit of personal identity information which is identified in the form of a unique URI. It refers to the information user wants to store. There are two message type, “store” and “fetch” to transfer attributes. The “store” saves/updates the attribute information in the OpenID provider whereas “fetch” retrieves the information from the provider. Both of these messages originate from relying party. OpenID attribute exchange service provides an information model for moving the identity information between different sites. Attribute has type identifier URI and its value could be any information. Attribute is associated with a subject identifier.

Subject identifier is an URI that corresponds to the end user identifier in the authentication part of message. Since, the subject identifier for identity attribute is same as the identifier of end user; subject identifier is not included in attribute exchange.

An attribute type identifier is an URI for referring to the attribute value. Attribute type can be resolved and then dereferenced to get a description of attribute property.
OpenID providers can use metadata obtained through dereferencing new or unknown attribute types to dynamically assist the user in providing the attribute. Attribute value is a string in UTF-8 format and it should not contain new lines.

Fetch message is a keyword used to fetch the message from OpenID provider. The fetch request requires OpenID.ax.mode field to be compulsory set. The various other fields listed in Tab 2.1 can also be mentioned in fetch request.

<table>
<thead>
<tr>
<th>Fetch message request fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenID.ax.mode</td>
<td>Value of this field should be “fetch_request” for sending fetch attribute request to OpenID provider and “fetch_response” for retrieving the response.</td>
</tr>
<tr>
<td>OpenID.ax.type.&lt;alias&gt;</td>
<td>It specifies the type identifier URI of a requested attribute</td>
</tr>
<tr>
<td>OpenID.ax.required</td>
<td>This field is used to specify the values of attribute alias.</td>
</tr>
<tr>
<td>OpenID.ax.if_available</td>
<td>This field allows the relying party to complete interaction with end user even when attribute fields are not available</td>
</tr>
<tr>
<td>OpenID.ax.count.&lt;alias&gt;</td>
<td>This field states the number of value requested by an relying party to the OpenID provider. By default the value is 1.</td>
</tr>
<tr>
<td>OpenID.ax.update_URL</td>
<td>If this field contains a value then the OpenID provider will redirect the fetch response to the specified URL.</td>
</tr>
</tbody>
</table>

Table 2.1.: Fetch attribute request parameter list

Example for a fetch request is in Tab 2.2. Fetch request message specifies that the requested attributes in the fetch request are fullname (http://example.com/schema/fullname), gender (http://example.com/schema/gender), favourite_dog (http://example.com/schema/favourite_dog) and favourite_movie (http://example.com/schema/favourite_movie). The favourite_movie can contain a maximum of 3 values. Mandatory required attributes are fullname and gender. Other two attributes (favourite_dog and favourite_movie) should be listed if there values are available. Fetch attribute response should be redirected to the OpenID consumer “http://idconsumer.com/update?transaction_id=a6b5c41” URL.

<table>
<thead>
<tr>
<th>OpenID.ns.ax=<a href="HTTP://OpenID.net/srv/ax/1.0">HTTP://OpenID.net/srv/ax/1.0</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenID.ax.mode=fetch_request</td>
</tr>
<tr>
<td>OpenID.ax.type.fname=<a href="HTTP://example.com/schema/fullname">HTTP://example.com/schema/fullname</a></td>
</tr>
<tr>
<td>OpenID.ax.type.gender=<a href="HTTP://example.com/schema/gender">HTTP://example.com/schema/gender</a></td>
</tr>
<tr>
<td>OpenID.ax.type.fav_dog=<a href="HTTP://example.com/schema/favourite_dog">HTTP://example.com/schema/favourite_dog</a></td>
</tr>
<tr>
<td>OpenID.ax.type.fav_movie=<a href="HTTP://example.com/schema/favourite_movie">HTTP://example.com/schema/favourite_movie</a></td>
</tr>
<tr>
<td>OpenID.ax.count.fav_movie=3</td>
</tr>
<tr>
<td>OpenID.ax.required=fname,gender</td>
</tr>
<tr>
<td>OpenID.ax.if_available=fav_dog,fav_movie</td>
</tr>
<tr>
<td>OpenID.ax.update_URL=<a href="HTTP://idconsumer.com/update?transaction_id=a6b5c41">HTTP://idconsumer.com/update?transaction_id=a6b5c41</a></td>
</tr>
</tbody>
</table>

Table 2.2.: OpenID fetch request example

[^3]: HTTPS://OpenID.net/specs/OpenID-attribute-exchange-1_0.html
Fetch response message contains the information requested in fetch request. All attributes requested contains the assigned alias value prefixed by OpenID.ax.value. Attribute types are returned in “OpenID.ax.type.<alias>” field. Example of response of fetch request sent above in Tab 2.2 is shown below in Tab 2.3. Response depicts that attribute value for fullname, gender, favourite_dog and favourite_movie are fetched. Value for fullname is “John Smith”, the gender value is null as count is “0”, favourite_dog is “Spot” and movie count is 2. Hence, the two values for favourite movies “Movie1” and “Movie2” are returned.

Table 2.3.: OpenID fetch response example

| OpenID.ns.ax=HTTP://OpenID.net/srv/ax/1.0 |
| OpenID.ax.mode=fetch_response |
| OpenID.ax.type.fname=HTTP://example.com/schema/fullname |
| OpenID.ax.type.gender=HTTP://example.com/schema/gender |
| OpenID.ax.type.fav_dog=HTTP://example.com/schema/favourite_dog |
| OpenID.ax.type.fav_movie=HTTP://example.com/schema/favourite_movie |
| OpenID.ax.value.fname=John Smith |
| OpenID.ax.value.gender=0 |
| OpenID.ax.value.fav_dog=Spot |
| OpenID.ax.value.fav_movie=2 |
| OpenID.ax.value.fav_movie.1=Movie1 |
| OpenID.ax.value.fav_movie.2=Movie2 |
| OpenID.ax.update_URL=HTTP://idconsumer.com/update?transaction_id=a6b5c41 |

2.3. OAuth

OAuth is an open standard for authorization[17]. It is used in conjunction with authentication protocol to provide secure access to the server resources on behalf of end user/owner. OAuth enables applications to access each other’s user data. e.g.: location based application can access the Foursquare data, the online job application can access the user’s LinkedIn and Xing data.

Participants in OAuth process

There are four participants in OAuth process[7]:

1. Resource owner is the person who is owner of data stored in the server that needs to be shared with other web applications.

4https://foursquare.com/
5https://www.linkedin.com/
6https://www.xing.com/
7HTTP://tutorials.jenkov.com/OAuth2
2. Resource server is the server that hosts services that can be used by the user.

3. Authorization server authorizes user by providing access tokens to access the resource from resource server. Authorization server and resource server might be same or different.

4. Client application is the application that wants to access user’s (resource owner) information from resource server. Client application can be of two types, public client application and confidential client application. Client application that requires data access of the resource owner could be any desktop application, web application, user agent application or mixture of applications.

**Client Id and Client secret distribution and retrieval**

OAuth[18] authorization server is responsible to provide client Id and client secret either confidentially or openly depending on client application. The process of providing client Id and secret confidentially to the client application takes place in case of confidential client application. This process can be used to overcome the security constraints. The process of providing client Id and password openly to client application takes place in the case of public client application. In this client password can easily get logged and traced resulting in security threat.

To get client Id and client secret from OAuth authorization server, prior to access of resources, client application registers itself to the authorization server. Once, the registration is done the client will be valid till the client application registration is revoked. During the registration process, the client application is given client Id and client secret by the authorization server. Also, while doing registration, the client has to provide a URI where the client application wants to redirect the client after successful authorization.

**Types of Authorization grant**

When client application gets an authorized access to the resources from resource server, it is known as authorization grant. Authorization grant can be given by resource server to client application in four possible ways - authorization code, implicit grant, resource owner password credentials and client credentials.

1. In authorization code resource owner access the client application. Resource owner login into client application with the authorization server credentials. Client application also sends the client Id so as to let the authorization server know its identity. After this, user’s consent is taken to grant the resource access to the client application. Once, the user agrees, authorization server releases authorization code by attaching it to the redirected URI. This redirected URI was given by the client during the registration process. Client application sends the redirected
URIs, authorization code, client Id and client secret to the authorization server. After, validation of the data sent, authorization server releases the access token to the client application. Client application then uses the access token to access the resources from the resource server.

2. In implicit grant the authorization server releases the access token instead of authorization token after resource owner’s authorization and agreement by the resource owner to the consent specifications. This makes the system more vulnerable to security attacks as access token is accessible quite easily.

3. In resource owner password credential method, resource server’s username and password is given to the client application. Client application can then directly access the resource owner’s resource by entering the resource server’s credentials. This type of grant is based on high trust relationship between the client application and the resource server.

4. In client credential authorization grant the client application demands the access of resources that are not related to resource owner. This may be some commonly stored data such as map data from Google map API. This data does not belong to the resource owner.

In this thesis work I have used authorization code type grant because it is used by majority of OAuth enabled sites. As discussed before, to get the authorization code from the authorization server, client application has to send the authorization code request. The request parameters to be sent in authorization code request are response_type, client_id, scope, redirect_URI and state. Below in Tab 2.4 listed is the detailed description of the request parameters that are required to be sent in the authorization code request.

<table>
<thead>
<tr>
<th>Request parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>response_type</td>
<td>Its a mandatory parameter. Its set to code, it specifies the type of response expected.</td>
</tr>
<tr>
<td>client_id</td>
<td>Its also a mandatory field. The client Id is the id which client recieves after registration with the authorization server</td>
</tr>
<tr>
<td>scope</td>
<td>This is an optional field. This field is given to define the scope of the request.</td>
</tr>
<tr>
<td>redirect_URI</td>
<td>This is also the optional field. It specifies the field specified by the client during registration.</td>
</tr>
<tr>
<td>state</td>
<td>This is also an optional field. This field takes the client state as the input.</td>
</tr>
</tbody>
</table>

Table 2.4.: Authorization grant : Authorization code request parameter

The response returned from authorization server after sending above request parameters may result in either successful response or error response as listed below in Tab
Chapter 2. Foundation

2.5 and 2.6 respectively. After successful authentication the authorization server sends the response parameter “code and state” with the redirect URI. The description of the parameters is given in Tab 2.5.

<table>
<thead>
<tr>
<th>Response parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>code is the authorization code requested by client application</td>
</tr>
<tr>
<td>state</td>
<td>If the client application sends the client state then the same value is returned by the authorization server</td>
</tr>
</tbody>
</table>

Table 2.5.: Authorization grant : Authorization code response parameter

If the authorization server receives some false request parameter then it will notify the resource owner error. If the authorization server successfully authenticates the client application or error is due to network failure or connection fault then the error response description with error response parameter is as listed in the Tab 2.6.

<table>
<thead>
<tr>
<th>Error response parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>Contains the error code.</td>
</tr>
<tr>
<td>error_description</td>
<td>Contains a text describing about the error</td>
</tr>
<tr>
<td>error_URI</td>
<td>Contains the URI of the webpage that describes the more details about the error page</td>
</tr>
<tr>
<td>state</td>
<td>Contains the same value of client state sent with the request to the authorization server.</td>
</tr>
</tbody>
</table>

Table 2.6.: Authorization grant : Authorization code error response

After retrieving the authorization code from the redirect URI, the client application can request for the access token from the resource owner to access the resources. The request parameters sent to resource server to retrieve the access token are listed in Tab 2.7 with their description.
Chapter 2. Foundation

Request parameter | Description
--- | ---
client_id | This is mandatory field. It contains the id which the client application gets after registering with the authorization server.
client_secret | This is mandatory field. It contains the password which the client application gets after registering with the authorization server.
grant_type | This is mandatory field. It contains the grant type. Here it is authorization code type grant.
code | This is mandatory field. It contains the authorization code fetched from the request.
redirect_URI | This is mandatory field. It contains the request URI given previously during the request.

| Response parameter | Description |
--- | ---
access token | Contains the access token returned by authorization server. |
expires in | Contains time period (in sec) after which access token expires and becomes invalid.
token type | Contains the token type assigned by authorization server. |
refresh token | Contains the token that can be used to obtain new access token after the expiration of old access token. |

Table 2.7.: Authorization grant : Access token request parameters

Resource server returns the response to the token access request in JSON string format with key value pair. Response parameters returned are listed below in Tab 2.8. After getting the access token, the client application can access the resource owner’s personal data.

Table 2.8.: Authorization grant : Access token response parameters
Figure 2.4.: OAuth basic flow

Figure 2.4 describes the basic working of OAuth protocol. The user accesses the client application. The client application provides the list of authentication application/provider and asks the user to login with possible authentication provider. For e.g. in above diagram the list contains Facebook, Google, Yahoo, Twitter etc. The user choose to login via the identity provider in which he has the account which might be Facebook. As soon as user chooses to login via its identity provider, the user is redirected to the identity provider application login form. The user enters his credentials in the form and submits it. After submission of form, the identity provider authenticates the user. The user is then asked whether he/she agrees to grant the access of his/her data to the client application. As soon as the user accepts the agreement, the user is redirected to the client application URL that client application has provided to the identity provider application. After getting authorization code, the client application sends the authorization code, client Id, client secret to the resource server. In above figure authorization server and resource server are same. Resource server then provides access token to the client application to access user’s personal data.
2.3.1. OpenID VS OAuth pseudo Authentication

The comparison between OAuth and OpenID is done because it is said that OAuth works on the top of authentication. To make more clear line of distinction between the two protocol a comparison have been performed in this section. Figure 2.5 illustrates both the protocols in detail. The end user is asked to enter the identity URL for OpenID authentication whereas the end user application is asked to enter the access token in case of OAuth authentication. In OpenID the API provider allows access to the users because it trusts the OpenID provider. In OAuth the API provider allows application access as it trusts its own validation key (access token).

Figure 2.5.: OpenID & OAuth protocol[2]

2.4. Eduroam Basics

Eduroam (education roaming) provides safe and secure network access for the user of an institution to access the network internationally in another institution\[19\]. The user can access the secure network from all over the world where Eduroam service is available\[20\]. The authentication of user takes place at their home institution with the help of Radius server whereas authorization of users is performed at the other institution where the Eduroam service is available\[21\]. Figure 2.6 shows basic Eduroam device that participate to perform Eduroam authentication. The supplicant connects to the authenticator with EAPOL protocol. The authenticator than connects to authentication server with the help of EAP over Radius protocol. The authentication server is further connected with a database.

![Diagram of Eduroam participants](image)

Figure 2.6.: Eduroam participants\[3\]

2.4.1. Radius protocol

Remote Authentication Dial In User Service (Radius) is the protocol that provides authentication and authorization service\[22\]. The Radius server and NAS device communicate using the User Datagram Protocol (UDP). The Radius protocol is a connectionless service. Retransmission of packet, timeouts, and server availability is handled by Radius enabled device and not by transmission protocol.
Radius protocol is a client/server protocol. Figure 2.7 shows the basic communication setup of the Radius Protocol. The NAS device/captive portal acts like Radius client and the daemon process running in the Linux or Windows system acts like a Radius server. The NAS device passes the user information to the Radius server with the help of proxy Radius server. This proxy Radius server act as an intermediate client and they further pass the user information to actual Radius server. The proxy is used only if the user is not in their institute premises. The Radius server on receiving users data authenticates it and sends the response with the configuration information to the Radius client. The Radius client uses this configuration information to deliver the service to the user.

Figure 2.8 shows the Radius authentication. The supplicant initiates the PPP request to the Radius client. The NAS device asks the supplicant for its EAP identity. The supplicant provides the EAP identity and the NAS device initiates the Radius access request to the authentication server. An SSL/TLS tunnel is created between the supplicant and the authentication server based on EAP protocol for added security. The most common EAP protocols that uses SSL tunnel are EAP-TLS, EAP-PEAP and EAP-TTLS. Inside this SSL tunnel actual username and password is shared between the authentication server and the supplicant. The authentication server sends the Radius access challenge to the NAS device. The NAS device forwards the access challenge packet to the supplicant. The access requests may seek for username and password or certificate depending on the protocol [PAP or CHAP]. The supplicant sends the credentials and certificates in EAP packet to the NAS device. The NAS device embed the EAP packet in the Radius packet and forwards the Radius packet to Radius server. Radius server sends accept or reject as a response to the client. The Radius client performs the action on basis of the response send by the Radius server.
The Radius server supports variety of protocols such as PAP, CHAP, PPP, unix login, EAP etc. The user login process contains the Access-request query that moves from NAS device to the Radius server and the response to the request in form of access accept or access reject from the Radius server to the device. The Access request consists of the username, realm, encrypted password, NAS IP address and port. 1812 is the port assigned by RFC 2865 for the Radius server. Upon receipt of the access request from the NAS client the Radius server searches for the credentials in the database. If the credentials are fetched access accept is sent or else default profile is loaded or access reject message is sent with the code and text message for failure.

In the Radius authentication and authorization are done together. If the username and password are found then the list of attributes and values to be used for the session are returned. Some of the attributes are service type, protocol type, user’s ip address, access list, static route for NAS routing table. All this information gets installed in NAS device.

Communication between Radius client and server consists of eight messages. The message and its description is listed in Tab 2.9.

Figure 2.8.: Radius Authentication [5]
Chapter 2. Foundation

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Request</td>
<td>This message flows from NAS device to the Radius server to authenticate the user</td>
</tr>
<tr>
<td>Access Challenge</td>
<td>This message flows from Radius server to NAS device used to ask something from the user</td>
</tr>
<tr>
<td>Access Accept</td>
<td>This message is sent from the Radius server to NAS device to show successful authentication of the user.</td>
</tr>
<tr>
<td>Access Reject</td>
<td>This message is sent from the Radius server to NAS device to show failure of authentication of the user.</td>
</tr>
<tr>
<td>Accounting Request</td>
<td>This message flows from client to accounting server to convey accounting information about the services provided to the user by the server.</td>
</tr>
<tr>
<td>Accounting Response</td>
<td>This message flows from server to client to convey receipt of accounting information by the server.</td>
</tr>
<tr>
<td>Status Server</td>
<td>This message is experimental</td>
</tr>
<tr>
<td>Status Client</td>
<td>This message is experimental</td>
</tr>
</tbody>
</table>

Table 2.9.: Packet message

As described in Tab 2.10, the Radius message packet contains the header, id, code, length and authenticator field followed by payload. The payload contains following list of attributes.

1. attribute type: describes the type of attribute
2. attribute length : describes the length of attribute in octets
3. attribute value: contains the value that attribute will carry.

<table>
<thead>
<tr>
<th>code</th>
<th>identifier</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>authenticator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>attributes</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.10.: Radius message packet format

The Radius request carries messages from NAS devices to server and vice-versa in the form of Radius packets. This Radius packet contains EAP packets inside the attributes field especially for the Access request and Access challenge message.

2.4.2. Extensible Authentication Protocol

Extensible Authentication Protocol (EAP) is the generic authentication framework defined in RFC 3748 that allows NAS devices to understand the multiple authentication
protocol without upgradation\cite{23}. EAP do not perform authentication, it merely acts as a framework between the user and the Radius server. EAP helps the supplicant to communicate with the Radius server. The EAP hears the conversation and then comes again at the end of the message. The four type of EAP messages are described in Tab 2.11.

<table>
<thead>
<tr>
<th>EAP message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAP request message</td>
<td>EAP request information is sent by Radius server to the user to present its information. eg: identity information</td>
</tr>
<tr>
<td>EAP response message</td>
<td>EAP response message is sent from user to the Radius server to send the information requested.</td>
</tr>
<tr>
<td>EAP success</td>
<td>Returned by Radius server if the authentication is successful.</td>
</tr>
<tr>
<td>EAP failure</td>
<td>Returned by Radius server if the authentication is failed.</td>
</tr>
</tbody>
</table>

Table 2.11.: EAP messages

EAP can run over PPP, wired or wireless LAN. The EAP packet format is shown in Tab 2.12 below. The code field contains the integer number (1 for EAP request, 2 for EAP response, 3 for EAP success, 4 for EAP failure). The identifier field contains the transaction id for matching the request and response messages. The length field contains the length of message data and the data field contains the information to be carried out between user and the server.

<table>
<thead>
<tr>
<th>code</th>
<th>identifier</th>
<th>length</th>
<th>data</th>
</tr>
</thead>
</table>

Table 2.12.: EAP packet format\cite{7}

2.4.3. Eduroam vs Other web based protocols (OpenID and OAuth)

Eduroam is non browser based authentication protocol. Eduroam uses the EAP protocol to communicate with the Radius server. Eduraoam exists only for the educational institutes. Eduroam does not allows the users credential to be provided to the third party web services.

The other web based protocol can be browser/non browser based authentication protocol. The OpenID and OAuth uses HTTPS protocol to communicate with the authentication server. OpenID and OAuth can be used in any web based organization. User credentials can be given in trustworthy third party web services.
3. Identity Management Bridge

The *identity management bridge* (IMB) is a platform to convert Shibboleth SAML data from GWDG to other protocols (such as Radius, OpenID, OAuth, SAML) data and vice-versa to facilitate communication. This inter protocol conversion is required for successful communication between two identity management systems. IMB is also responsible for secure data exchange between two identity systems. IMB is the key element for multi-protocol identity management system federation. The goal of this thesis is to develop an IMB plug-in for Shibboleth identity provider. This chapter enumerates the benefits of the IMB plug-in as well as analyzes the requirements and derive the efficient solutions from various possible solutions for the implementation of IMB.

3.1. IMB Benefits

This section describes the benefits of using IMB plug-in. IMB joins two identity management systems based on different protocols together. The benefits have been described from various perspectives.

**Multi-protocol federation perspective**

- The federation of GWDG and other identity management system results in availability of more services for the users especially from other identity management systems. For e.g. an OpenID user can use the services provided by GWDG as well as services provided by OpenID.

- The federation results in a secure collaboration and information exchange. IMB provides a common and secure information exchange platform for all the same protocol based identity management system. If this process would not be federated than bilateral trust agreement would have to be established between each organization and GWDG. For e.g. if organization A and organization B both use OpenID based identity management system than, the bilateral trust agreement would have been signed between GWDG and organization A, GWDG and organization B. This may arise standardization issues as each institution would have individual attribute requirement and authentication policy.
GWDG service provider perspective

- The service provider is benefited as the number of users accessing the service provided by the service provider increases. This could increase the business profit.

- The cost per user is lowered as the increasing size of user do not demand for password storage and user support.

GWDG identity provider perspective

- With IMB, users from other protocol based identity provider are also the part of GWDG identity provider. Hence, the GWDG identity provider can offer more to their user. The user can access more services compared to the services that are available in their local identity management system.

- There is no extra administration work required to provide access rights to the user.

- There is no need to maintain distinct user credentials for separate applications.

Identity holders perspective

- The identity holders do not have to remember and manage the credentials for separate applications. Using one identity, they can access all the services.

- The identity holder can access wide range of services.

Overall Benefits

- The IT cost is reduced because identity provider system will automatically access or revoke rights of the user. Hence, there is no need to perform the task with the help of an application or manually.

- The help desk cost is reduced because there are minimal call for forgotten passwords from the user. Hence, there is no need to reset the passwords. Therefore, fewer numbers of employees are needed in help desk department.

- Time is reduced because users do not have to create user account for all the services. Also, the organizations do not need to invest time in understanding the agreement and attribute exchange policy of other institutes due to the use of a standardized federated identity management application.

2HTTP://www.dummies.com/how-to/content/how-to-benefit-from-identity-management-in-cloud-c.html
Chapter 3. Identity Management Bridge

- Multiple installations of service provider and identity provider is not required. Due to federation and trust the institute does not need to deploy their own service provider and identity provider server for each and every protocol. They can use other institutes provider without any issues.

- Data management becomes easier due to distributed identity provider. As data is distributed among various providers, the deletion, update and insertion of data becomes easier. If it would be centralized then it would be difficult to handle bulky databases. Redundant identities are minimized.

3.2. Requirement Analysis

GWDG institute uses Shibboleth identity management system hence, this thesis work will use Shibboleth service provider and identity provider as the base. Shibboleth identity management system will be extended to communicate with other identity management systems for the purpose of user’s authentication and authorization. This plug-in is called identity management bridge (IMB). The plug-in will support Shibboleth identity provider 2.4.x and higher versions. This section analyzes various requirements that should be supported by IMB.

- Collated IDP Installation: GWDG should not require the installation of multiple identity management providers in its system. GWDG system currently uses Shibboleth. To avail the communication between Shibboleth system and other providers there should not be any installation requirement of application belonging to other identity management providers. The installation of providers in GWDG system will create unnecessary burden on each services causing lots of inconvenience and process slow down. Also, it’s not easy to maintain all the identity provider installation for one system. Hence, the IMB should be designed in such a way that it does not require other provider’s setup to be installed at GWDG.

- Inter identity management system communication: This project aims to avail the services provided by GWDG to the users of other organizations. GWDG uses Shibboleth services whereas users from other institutions may use different providers. To avail the services successfully GWDG services should be able to communicate with the different institutions provider to check the user’s authenticity. Hence, the IMB should be designed to avail communication between different providers and Shibboleth service.

- IDP discovery: The aim of this thesis is to avail GWDG service provider to the users of other institutes. The other institutes may have several identity providers setup. It is necessary to recognize the user’s identity provider from several other
distinct providers to establish successful communication between GWDG service and identity provider. Hence, there arise the necessity for a solution that defines some identity provider discovery mechanism.

• **UI design and user input**: To use GWDG services, the user need to authenticate themselves. For the process of authentication user needs to give their credentials as input. This user input varies among different identity provider. For e.g. OpenID identity provider requires the URL & credentials for the authentication. Hence, some common user input parameter list should be defined.

• **Identity data security**: User provides the input data (such as credentials) to the service. The service then communicates with other providers. During communication there is inter-change of user’s information. User’s data can be hacked during the communication between the providers and GWDG service. Hence, the mechanism should be designed to protect user data.

• **Attribute exchange**: After checking the authenticity of the users and availing successful communication between GWDG services and other identity provider, next step is to fetch user’s data from other identity providers after authentication. This is required because GWDG services may requires user’s information such as his email address, contact list or personal identification data [e.g. fullname, gender, nationality etc.]. Hence, the solution should be able to retrieve user’s data (attributes) from other identity providers for the GWDG services.

• **Post authentication user data storage**: GWDG services requires user’s data such as his email contact or his personal information [sex, age, birthday etc.] from the identity provider after authentication. This data needs to be stored either temporarily for a particular session or permanently in the service provider’s system. Also, the user’s data should be deleted or disposed from time to time so as to maintain the size of database and to make the system more efficient, up to date and effective. Hence, this requires the design of a solution for data storage & disposal.

• **Re-authentication of same user**: After the current session expires, the same user might again want to use GWDG services. For this the user will enter the same credentials again. Also, it might be possible that same set of user attributes are required by the service. To achieve this, the service has to communicate again with the identity provider or there might be some cache system. Hence, the solution should be designed to handle the re-authentication step more efficiently.

### 3.3. Possible solutions

In this section possible solutions for the requirements discussed above have been mentioned. These solutions are carefully analyzed and the most suitable solution that IMB
should consist has been selected.

### 3.3.1. Inter Identity Management System Communication

The goal of this thesis is to allow users from various institutions/providers to use GWDG services. To achieve this either GWDG service can install all the providers in GWDG local system or GWDG service provider needs to be able to communicate with other identity providers to authenticate the user. Installation of all the identity providers in GWDG system will create heavy burden and data redundancy. Hence, GWDG service should be able to communicate with other identity providers. The communication can be accomplished in two ways as listed below and shown diagrammatically in Fig 3.1.

**Shibboleth Service Provider talks directly to all providers**

The idea behind this is Shibboleth service provider can communicate directly to other identity provider in the same way as it communicates with Shibboleth identity provider as shown in part 1 of Fig 3.1. This would require creation of an IMB plug-in in Shibboleth service provider. This would result in installation of IMB plug-in in each service provider that needs to communicate with other identity provider. A single system contains several services depending on the identity provider. Hence, all the Shibboleth service providers are required to be changed. This would be quite tedious as services would not work during the installation. Also, an update in the IMB plug-in will also be required from time to time resulting in an outage.

**Shibboleth identity provider as an intermediate**

The idea behind this solution is, Shibboleth service provider talks to Shibboleth identity provider and Shibboleth identity provider talks to other identity provider as shown in part 2 of Fig 3.1. IMB plug-in would be required to be installed only in Shibboleth identity provider. Hence, this would not require any service provider changes. Since, there are more services for a single identity provider; this method is more beneficial and efficient as its lot more convenient, time effective and cost effective to change an identity provider instead of changing multiple service providers.
3.3.2. Identity Provider discovery

As soon as the user enters the credentials, Shibboleth identity provider will check the credentials against its database. If Shibboleth identity provider does not have the user information mentioned in credentials, the Shibboleth identity provider should redirect the authentication request to other identity providers with the help of IMB plug-in. In this scenario the Shibboleth identifier has to discover all the trustworthy identity providers and re-direct the user’s credentials to them. The credentials could be redirected either to all identity providers or some of them. All the possible ways of redirection has been discussed below.

User credential broadcasting

In this case, the Shibboleth identity provider will broadcast user credentials to all the available identity provider in the network. Due to this, there will be heavy traffic and
congestion in the network. This can slow down the system performance. Also, broad-
casting of password to all other identity provider may lead to security issues because
the server in external network may sniff the data. Also, a fake identity provider can
connect itself to the network to receive the user credentials. Also, the broadcasting
network might not be reliable.

**User inputs his Identity Provider information**

In this case, the user is asked to feed username, password and their identity provider
name or email id in Shibboleth identity provider. On basis of fed details, the request
is redirected to desired identity provider. This will create an additional task for the
user as they have to enter additional information. This will also create additional step
for Shibboleth identity provider as it needs to process one more field and extract the
information about other identity provider.

**User’s email as username**

In this scenario, the user is asked to enter his email as the username. Based on do-
main name specified in the email address the Shibboleth identity provider redirects
the request to specific identity provider. The only constraint in this solution is that the
user has to feed his email id or data in a specific format in the username field. Also,
this method requires additional software mechanism to extract the domain name and
transferring the data to correct identity provider.

**The fallback method**

In this method Shibboleth identity provider only takes the basic credential from the
user. The IMB plug-in then checks the user credential against the Shibboleth JAAS
authentication. If the identity information doesn’t matches, it checks with all other
protocol identity provider one by one. In this thesis, this method has been chosen
because it requires minimal user intervention. Also broadcasting of data might not
be safe as other server might sniff the data and other methods require either the user
intervention or restrict the user to provide data in particular format. In this method
user can enter data in any format in the username field based on their identity provider.
Hence, this method overcomes all the security loopholes. The only disadvantage of this
method is it is time consuming to check each identity provider one by one.

**3.3.3. User interaction**

The user gives their credentials as input for the purpose of authentication. This user’s
input can vary based on protocol and authentication provider. For e.g. OpenID provider
takes the user’s URL and credentials as input. Each identity provider requires their own
input parameter. As a solution to this problem we can either direct the user to their respective identity provider to give input and check authenticity or Shibboleth identity provider can take the input. Below there is more detailed discussion on each method.

![Diagram of Identity Management Bridge](image)

**Figure 3.2.: User input**

**Users interact with their Identity Provider**

The user at first may enter their email address, the Shibboleth service provider will extract the user’s identity provider information based on domain name present in the
Chapter 3. Identity Management Bridge

email. The other option is user directly choose the his identity provider as shown in part 1 of Fig 3.2. User will be redirected to the identity provider URL where he will enter his credentials and get authenticated. The advantage of this process is, user do not have to expose his credentials to Shibboleth system (although user has trust relationship with Shibboleth service provider and Shibboleth service provider has trust relationship with Shibboleth identity provider). On other hand, user has to trust on the other identity provider URL redirected by Shibboleth service provider. The user cannot authenticate whether he is redirected to trust-worthy URL or any malicious site.

Users interact with Shibboleth

The user enters his credential through Shibboleth identity provider as shown in part 2 of Fig 3.2. Shibboleth service provider redirects the authentication request to Shibboleth identity provider. If the user is not authenticated by Shibboleth identity provider, it redirects the request to other identity providers. In this case, user has to trust on Shibboleth system and expose their credentials to them. This method is more suitable as user has trust relationship with Shibboleth identity provider. The user also does not have to worry about the malicious URL issue. Also, IMB plug-in should use secure mechanism to transfer the user’s identity data from Shibboleth IDP to user’s IDP.

3.3.4. Security of users data

The user’s credential data is send from Shibboleth identity provider to other identity provider. If proper security measures are not taken in account, the user credential data could be leaked while sending. To overcome security loopholes, this thesis uses secured HTTPS connection to send the data. Also, other features such as nonce, etc. are used as per industrial standards.

3.3.5. Data transfer protocol

In this research paper, HTTPS authentication over HTTP is used for sending user’s identity data through post command to other IDP. Same method is used for receiving the response and attributes. HTTPS based authentication information transfer is the most basic form of URL authentication. In this the users data is passed in form of post or get request. For, non-browser based identity provider, java library or command line utility could be used to communicate between two identity providers.

3.3.6. Re-authentications and post authentication data

After the current session expires, the same user might again want to use the GWDG service. For this the user will enter same credentials again. Also, it might be possible
same set of user attributes are required again by the service. To achieve this, the service has to communicate again with the identity provider or there might be some cache system. There are two possible ways to handle the re-authentication as follows:

**User data storage in Shibboleth Identity Provider database**

In this case, the user’s credential data is stored in Shibboleth database. The storage of username and password may result in faster and easy authentication as the user will be authenticated with Shibboleth identity provider itself. On other hand, the data storage may lead to large database, redundancy of information and synchronization problem. If the data from other identity provider is stored in Shibboleth database then Shibboleth identity provider will know the information and data of the user permanently. If the user does not want to have any interaction or data storage in Shibboleth system in future then it will not be possible as Shibboleth will have user’s past data record which may lead to trust and security issues in future.

Also, if the user deletes the account in another identity provider or updates the data then there is requirement of a synchronization plug-in that could check the data in specific interval of time as data in Shibboleth database may become obsolete after updation of data in other identity provider.

There is copy of information at two places one in Shibboleth database and other in another identity provider. This results in twice the consumption of memory space which is not recommendable. Also, cost of storage at two places will also be high.

Shibboleth uses the file or structured database in most of the systems. User data storage from other identity provider may increase the size of database. This may affect the overall performance of the Shibboleth system as bulky files or database works slower and will also lead to longer backup times.

**No database storage**

Another option is to create a session variable and pass the user information in that variable during each authentication. The life of user data will depend on life of the session variable. This will raise the need for re-authentication of user after expiration of session variable. This process may be time taking as each time all the identity providers will be checked for user authentication until perfect match is found but this will save data redundancy, database size growth, synchronization problem. Hence, this method will be used in this work for re-authentication process.

**3.3.7. Maintenance of authorization data after authentication**

Authorization data will be fetched from other identity provider once after user logins. Since, Shibboleth fetches all the required attributes after login, the authorization data
from other identity provider will be mapped to the attribute data required by Shibboleth service provider. This data will then subsequently pass from one layer to another via session variable.

The user attributes will not be saved in the database and will be dynamically fetched during each login. Hence, there will be no storage setup requirement resulting in relief from cost factor, redundancy and wrong data problem.

3.4. Use Case

The users can login with different identity provider credentials in Shibboleth system to use the Shibboleth service. The user can also logout from Shibboleth service to end his session.

3.5. Sequence Diagram

Figure 3.4 describes the sequence of process that will be performed in this thesis work. This process will contribute in end user’s authentication and authorization. There are four components in this diagram. The browser is a web-browser through which user can interact with the identity provider. The Shibboleth service provider is an entity that provides services. The Shibboleth identity provider is the entity that stores user’s identity information [e.g. username, password]. The other identity provider can be OpenID, OAuth, Eduroam provider. This also stores user’s identity information.

Suppose, the user desires to use the service of Shibboleth service provider. The user opens the service through the web browser. As soon as, service is open, the web browser will send the request for the service access to the Shibboleth service provider. Shibboleth service provider will redirect the browser to Shibboleth identity provider. Browser will send the authentication request in SAML protocol to Shibboleth identity provider. Shibboleth identity provider sends a HTML form to the web browser. User inputs his credentials in the web form to get authenticated. Shibboleth identity provider authenticates the user against its database. If the user entry is not found in the database of Shibboleth identity provider, the system sends the user information to other identity provider in the form of OpenID request or OAuth request or Radius-EAP request with the help of IMB plug-in. Other identity providers searches the user entry in their database and sends the authentication response to Shibboleth identity provider via IMB plug-in. Shibboleth identity provider will convert the response in SAML format and send the response to the web browser. Web browser sends the SAML response to Shibboleth service provider. Shibboleth service provider provides access to the service or resources based on the authentication response provided by the web browser.
Figure 3.3.: Sequence Diagram
4. Implementation

4.1. Other identity provider details

4.1.1. OpenID provider

There are numerous OpenID providers available in the market. SimpleID is a simple and personal OpenID provider written in PHP. This project uses a SimpleID provider as it contains all the features such as SSL, certificates, nonce etc. The SimpleID supports both OpenID 1.0 and OpenID 2.0 protocol. It supports attribute exchange too. It authenticates against flat files and no database is required. Hence, it is suitable for our testing purpose. For beginners, it is easy to understand and use SimpleID for development purpose compared to some external identity provider. We setup our own OpenID provider because OpenID providers from organizations (such as Yahoo, Google) have complex configurations, hence it would have been difficult to understand the working of those providers. The SimpleID provider also provides finer control over the identity configuration of the user.

SimpleID configurations

SimpleID is installed and setup according to the steps given in the installation guide in appendix A.2 and an identity (user) is created in the file storage. SimpleID stores the identity information of the user in a text file called identity file. Each user has its own identity file. The identity file is named using the username as a filename. For this project user_simple is a SimpleID username. Hence, the file inside which the user_simple’s identity information will be stored will have name “user_simple.identity”.

The identity files stores multiple user identity information. The passwords are stored in MD5 (or other) hash format in the identity files. The identifier parameter is a URL/URI.

\[1\text{HTTP://OpenID.net/get-an-OpenID/}\]
\[2\text{HTTP://wiki.OpenID.net/w/page/12995226/Run%20your%20own%20identity%20server}\]
\[3\text{http://simpleid.koinic.net/}\]
\[4\text{http://simpleid.koinic.net/docs/1/pear/}\]
Chapter 4. Implementation

The identifier must be unique and accessible from the internet. The administrator parameter (specifies whether the user will have admin rights) is set according to user’s role.

The user enters the identity URL in the relying party to claim his identifier i.e. the user tells the relying party that his identity information is stored in that identifier. This process helps websites/relying party to find the identity provider of the user when he logs in. This can be done using a `<link>` tag or with YADIS protocol.

If the user uses a `<link>` tag then, the webpage returned by identity URL is edited by the relying party and the identity file installation location is given under `<head>` tag as shown in Tab 4.1. The `identifier location` is present in `href`.

```xml
<link rel="OpenID.server" href="http://localhost/SimpleID/" />
<link rel="OpenID2.provider" href="http://localhost/SimpleID/" />
```

**Table 4.1.: Identity file location information**

If the user uses YADIS protocol then the get page request is sent to the identity URL. For e.g. get request for user `test1` can be “http://localhost/simpleid-idp/index.php?q=xrds/test1” identity URL. This will return an XML document (XRDS document) specifying the location of the identity provider as shown in Tab 4.2.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns="xri://$xrd*($v*2.0)" xmlns:xrds="xri://$xrds"
xmlns:simple="HTTP://xrds-simple.net/core/1.0">  
<XRD version="2.0">
  <Service priority="10">  
    <Type>HTTP://specs.OpenID.net/auth/2.0/signon</Type>  
    <URI>HTTP://localhost/simpleid-idp/</URI>  
    <LocalID>HTTP://localhost/simpleid-idp/</LocalID>  
  </Service>  
  <Service priority="20">  
    <Type>HTTP://OpenID.net/xmlns/1.0</Type>  
    <URL>HTTP://localhost/simpleid-idp/</URL>  
    <LocalID>HTTP://localhost/simpleid-idp/</LocalID>  
    <IDP>HTTP://localhost/simpleid-idp/</IDP>  
  </Service>  
</XRD>  
</xrds:XRDS>
```

**Table 4.2.: XRDS document snippet**

The webpage returned by the identity URL can be edited to point to the XRDS document in following ways:

- If the webpage is generated programmatically and contains the MIME type `application/xrds+xml` then the document can be directly returned in response.
Chapter 4. Implementation

- If the webpage is generated programmatically and MIME type is other than above mentioned type, then the HTTP header is given as
  X-XRDS-Location: http://localhost/SimpleID/index.PHP ? q=xrds/test1

- The last alternative is to include XRDS document in the <head> tags in the form of a <meta/> tag as shown below:

```
<meta HTTP-equiv="X-XRDS-Location" content="HTTP://localhost/SimpleID/index.PHP?q=xrds/test1" />
```

4.1.2. OAuth provider

I tried various OAuth2.0 Protocol provider such as Apache OLTU\(^5\), Spring security\(^6\) and Google\(^7\). Apache OLTU provides authorization server, resource server and client. Documentation\(^8\) of Apache OLTU has provided good analysis of the system. The documentation stating the internal working of server is not sufficient. Also, there is a manual OAuth2 server default endpoint implementation requirement. Therefore, I did not chose Apache OLTU for this thesis.

Next analysis is on Spring security. It supports OAuth 1.0 and 2.0. Spring security samples\(^9\) Sparklr2 and Tonr2 were analyzed. Sparklr2 is a photo sharing application (OAuth provider), Tonr2 is photo printing application (OAuth consumer). Spring security has huge community and lot of help available online. It is easy to configure. Only shortcoming is that there is no GUI for the client application and resource server management.

Shibboleth extensions were also read. Shibboleth has facebook OAuth extension named as Facebook Login Servlet\(^9\). This extension extends shibboleth external authentication. We cannot use external authentication extension for this project as with the external authentication we cannot authenticate Shibboleth internal users. Hence, for this project work we need to extend UsernamePassword authentication servlet.

For this project the Google API is used for testing purpose. There is proper documentation available online in Google developer website about Google API\(^10\). This method has been used in this thesis because making our own inbuilt OAuth2.0 provider contains only simple and basic functionality. Hence, a real world OAuth provider has been used in order to work with more complex functionality and mechanism. Also, it will give some real working experience.

\(^5\)https://oltu.Apache.org/
\(^6\)http://projects.spring.io/spring-security-oauth/
\(^7\)http://OAuth.net/2/
\(^8\)https://cwiki.apache.org/confluence/display/OLTU/Documentation
\(^9\)http://apps.man.poznan.pl/trac/idp-facebook-integration/wiki
\(^10\)HTTP://developers.Google.com/identity/protocols/OAuth2

38
Google OAuth Configuration and Working

Google API uses OAuth 2.0 for user authentication and authorization. Shibboleth identity provider accesses the user data with the help of Google API. Google server and API share the secret, such as client id and client secret. To access the Google server with Shibboleth identity provider, Shibboleth IDP should know the client id and secret. To get the client id and secret, the API needs to be registered in the Google server. Google server sends the client id and secret during the registration process to the API.

Shibboleth identity provider should obtain access token from the Google authorization server for the API in order to access the user data. User consent is taken by Google authorization server to send the access token to Shibboleth IDP.

Shibboleth IDP then sends the access token to the Google API via HTTP authorization header. The access token are valid to access limited set of user information and resources as described in the scope of token request. The API sends the resource token to Shibboleth IDP. With the help of resource token, Shibboleth IDP can access the resources and information from the Google server.

4.1.3. Eduroam Provider

This project will use the GWDG Radius server as Eduroam provider. GWDG Radius server stores the authentication information of the students and other users (such as post-docs, research assistants, professors etc.) of university of Göttingen. In systems, a WPA supplicant tool is used to connect to the WPA2 enterprise network such as Eduroam. In this project we have used WPA supplicant and its eapol_test program to connect to the Eduroam provider.

WPA_Supplicant (program eapol_test configuration and working)

WPA_supplicant is a software from IEEE 802.11 supplicant to support WEP, WPA, WPA2 protocols[25]. The eapol_test is a program that links together EAP and Radius peer implementation that WPA_supplicant is using. This program integrates the access point i.e. authenticator with the supplicant i.e. wireless client. The eapol_test program provided by WPA_supplicant is one of the EAP testing tool that tests the interoperability of EAP methods with the Radius server.

After compilation, the eapol_test test command is executed to test the working as depicted in Tab 4.3. The test.conf file (shown in box below) contains the network configuration. The network configuration varies with the protocol. The box depicts configuration for MSCHAPV2 protocol. The authentication is performed against the Radius server running in the localhost.
network={
    ssid="example"
    key_mgmt=WPA-EAP
    eap=PEAP
    identity="bob"
    anonymous_identity="anonymous"
    password="hello"
    phase2="authap=MSCHAPV2"
    # ca_cert="/etc/raddb/certs/ca.der"
}

```
#  test.conf file
```

```
eapol_test -ctest.conf -a127.0.0.1 -p1812 -ssecret -r1
```

Table 4.3.: Test Eapol_test

Some of the commands configured to execute eapol tests with the description are listed below in Tab 4.4.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Configuration to be used. It has same format as wpa_supplicant.conf file.</td>
</tr>
<tr>
<td>-a</td>
<td>IP address of the authentication server.</td>
</tr>
<tr>
<td>-A</td>
<td>IP address of client</td>
</tr>
<tr>
<td>-p</td>
<td>UDP port of the authentication server.</td>
</tr>
<tr>
<td>-s</td>
<td>Shared secret with the authentication server.</td>
</tr>
<tr>
<td>-r</td>
<td>Number of re-authentication</td>
</tr>
<tr>
<td>-t</td>
<td>Time out in seconds</td>
</tr>
<tr>
<td>-C</td>
<td>Radius connect information</td>
</tr>
<tr>
<td>-M</td>
<td>The client’s MAC address</td>
</tr>
<tr>
<td>-o</td>
<td>Location to write out the server certificate</td>
</tr>
<tr>
<td>-N</td>
<td>This sends arbitrary attribute value pairs</td>
</tr>
<tr>
<td>-n</td>
<td>Indicates that no MPPE keys are expected</td>
</tr>
<tr>
<td>-W</td>
<td>Wait for the control interface monitor before starting</td>
</tr>
<tr>
<td>-S</td>
<td>To save the configuration after authentication</td>
</tr>
</tbody>
</table>

Table 4.4.: Eapol_test configuration

The eapol_test command is used in this thesis work as the NAS device and supplicant.
4.2. Implementation Design & Components

The chart above shows the basic Shibboleth authentication process. The user enters the URL of the service provider to access the service. The service provider sends the user authentication request in the form of SAML through the web browser to the profile handler. The profile handler forwards the request to the Authentication engine. The authentication engine forwards the request to the appropriate login handler based on configuration in "handler.xml" file present under Shibboleth IDP configurations. Shibboleth uses five types of login handlers: "UserPasswordLoginHandler", "RemoteUserLoginHandler", "IPAddressLoginHandler", "PreviousSessionLoginHandler", "ExternalAuthnSystemLoginHandler". The UserPasswordLoginHandler is used for user authentication against LDAP system. The RemoteUserLoginHandler is used for remote user authentication. The IPAddressLoginHandler is used to authenticate the user on basis of his IP address. The PreviousSessionLoginHandler maintains the same session for each authentication request. This handler has to be used with all other handlers to maintain the session variable. The ExternalAuthnSystemLoginHandler is used to authenticate the user against external application. In this thesis work we will use UserPasswordLoginHandler.

11https://wiki.shibboleth.net/confluence/display/SHIB2/IdPDevExtLoginHandler
Chapter 4. Implementation

The LoginHandler authenticates the user and send the response to the AuthenticationEngine. The AuthenticationEngine creates the IDP login session and returns the control to ProfileHandler. The ProfileHandler maps the users attribute data in the SAML response and sends the response to the browser. The browser sends the SAML response to the service provider.

In this thesis work we will extend ProfileHandler and LoginHandler. The LoginHandler will be extended to support the multi-protocol authentication. The ProfileHandler will be extended to make a new data connector to add the new users data released from multi-protocol authentication.

Figure 4.1 describes the detailed design and components of my implementation work for this thesis. The solid line depicts the existing work and the dashed line denotes the extensions implemented in this thesis work. Shibboleth service provider has four components assertion consumer service, attribute requester, resource manager and resources. Shibboleth identity provider has two components, handler service and attribute authority. IMB plug-in will be developed for SimpleID, Google and GWDG Radius server's user authentication.

User requests the service from the assertion consumer service (ACS). The ACS sends the user request to WAYF component. WAYF discovers the correct identity provider for the user and redirects the user to the Shibboleth identity provider's handler service component. The handler service component asks the user to input the credentials. The handler service checks the user data in its database. If the user data is not found the handler service will check with the help of IMB plug-in for user data in SimpleID provider. If data is not found in it then, IMB plug-in will check in Google provider. If the data is not found, the IMB plug-in will check in GWDG Radius server server. If the data is found in any of the above mentioned identity providers, it will be send to the IMB plug-in and it will convert the attributes and response in SAML format and return the response to the handler service. Also, IMB plug-in will make the data connector to save the attributes from this identity provider in Shibboleth identity provider for particular session. The handler service sends the response to ACS component. The ACS component forwards the response to attribute requestor. The attribute requestor component requests the attributes from the attribute authority in Shibboleth identity provider. The attribute authority sends the attribute to the attribute requestor. The attribute requestor sends the attribute to resource manager and the resource manager gives the access of resources to the user.

In this thesis work, IMB plug-in will be developed to handle protocol conversion, check users identity and attribute data exchange. This plug-in will connect Shibboleth identity provider with other identity providers. Under this, the handler service of the Shibboleth will be extended. Also, new data connectors will be developed to connect appropriate identity providers.
4.3. **IMB plug-in development**

The IMB plug-in will establish the successful communication between Shibboleth identity provider and SimpleID, Google and Eduroam providers. To implement this plug-in Shibboleth identity provider has to be extended. First, the username/password login handler is extended to authenticate users from other identity provider’s successfully. The next step includes creation of new data connector to handle new user attributes coming from other identity providers.

### 4.3.1. System Configuration used

Table 4.5 depicts the identity provider details and version used for the development of the IMB plug-in. IMB plug-in might support the higher or advanced versions.
Chapter 4. Implementation

<table>
<thead>
<tr>
<th>Protocol/tool</th>
<th>Version/type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shibboleth</td>
<td>2.5 or above</td>
<td>GWDG service &amp; identity provider</td>
</tr>
<tr>
<td>OpenID</td>
<td>SimpleID 1.0</td>
<td>other identity protocol</td>
</tr>
<tr>
<td>OAuth</td>
<td>Google API</td>
<td>other identity protocol</td>
</tr>
<tr>
<td>data connector</td>
<td>LDAP 2.0</td>
<td>data connection &amp; storage</td>
</tr>
<tr>
<td>Eduroam</td>
<td>GWDG Radius server</td>
<td>other identity protocol</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>8.018</td>
<td>run Shibboleth identity provider</td>
</tr>
<tr>
<td>Apache web server</td>
<td>2.4</td>
<td>HTTP server for hosting service</td>
</tr>
</tbody>
</table>

Table 4.5.: System Details

4.3.2. Shibboleth username/password login handler extension

**IMB plug-in** will extend the *username/password login handler*. The idea behind this extension is to transfer user’s information [username, password etc.] to other identity providers [SimpleID, Google, GWDG Radius server]. Also, **IMB plug-in** will retrieve the response from the identity providers and convert it into SAML. The section below covers the implementation in more details.

**Shibboleth side implementation to enable the IMB plug-in**

The *web.xml* file present inside the Shibboleth IDP source code is modified to enable the plug-in. This was done, to make the **IMB plug-in** work with the *Shibboleth identity provider*. Below source code snippet 4.1 shows the modified *web.xml* file.

This is done to specify the servlet configuration. Through this configuration file, Shibboleth identity provider gets the details about the authentication servlet that it should use in the login handler. In servlet-name the random name to be assigned to the servlet is specified. The Servlet-class specifies the path where the class is located. The load-on-startup value signifies the time of loading of the servlet. A lower number in load-on-startup will load the servlet before the higher numbered load-on-startup of a servlet.

```
1 <servlet>
2  <servlet-name>UsernamePasswordAuthHandler</servlet-name>
3  <servlet-class>uk.org.ukfederation.uaattribute.authn.UserAgentUsernamePasswordLoginServlet</servlet-class>
4  <load-on-startup>3</load-on-startup>
5 </servlet>
```

**Code Snippet 4.1: Modified web.xml**

After receiving the user attributes in the response the **IMB plug-in** directs the attributes to the *Shibboleth service provider*. The plug-in performs this operation with the help of data connector.
Shibboleth OpenID Bridge

This section will describe the implementation details on the process of the data exchange between Shibboleth identity provider and OpenID provider. To exchange data, **IMB plug-in** converts SAML to OpenID and vice-versa.

The Shibboleth identity provider connects with SimpleID provider using the **IMB plug-in**. To achieve successful communication between Shibboleth and SimpleID, the Shibboleth identity provider’s **Authentication module** was extended. Authentication module contains **login handler service module** inside **handler service**. The **login handler module** contains the class to handle the **username/password** login. This class is extended with the help of the **IMB plug-in**. The class is present inside “**UsernamePasswordAuthHandler**” servlet. The **class path** stated in servlet source code snippet 4.1 inside <servlet-class> is the path to new extended class.

Existing Shibboleth servlet class inside “**UsernamePasswordAuthHandler**” servlet checks the username and password against **JAAS** authentication configuration. If the user will input his SimpleID credentials [username and password], the Shibboleth JAAS authentication will fail and the new extended class module for SimpleID authentication will be checked. In new extended class, the **authenticate User** function is modified and new flow for authentication check for OpenID users is added as shown in source code snippet 4.2.

```java
String openiurl = "HTTP://localhost/simpleid-idp/index.PHP?q=my/profile";
UserAgentUsernamePasswordLoginServlet HTTP = new UserAgentUsernamePasswordLoginServlet();
CookieHandler.setDefault(new CookieManager());
String page = HTTP.getPageContent(URL);
List<NameValuePair> postParams = HTTP.getFormParams(page, username, password);
Boolean a = HTTP.sendPost(URL, postParams);
if (a)
    {String result = HTTP.getPageContent1(openiurl);
    listA = HTTP.getFormParams1(result, username, password);
    request.setAttribute(LoginHandler.PRINCIPAL_KEY, new UsernamePrincipal(principalName));
    request.setAttribute(LoginHandler.AUTHENTICATION_METHOD_KEY, authenticationMethod);
    }
```

**Code Snippet 4.2:** SimpleID user authentication flow

In SimpleID authentication flow, the user gives SimpleID URL and password of himself. The IMB plug-in talks to SimpleID server using HTTP POST. The IMB plug-in sends HTTP POST to the SimpleID identity URL. The SimpleID server returns the details of server location in the response inside <URL> tag as shown in Tab 4.6.
Chapter 4. Implementation

The IMB plug-in then extracts the username from identity URL input by user. Then, the plug-in sends HTTP POST with the username and password to the SimpleID server URI as shown in source code snippet 4.3. The SimpleID server returns the response of HTTP POST request in the form of attributes and response code.

```
1 HTTPPost post = new HTTPPost(URL);
2 post.setHeader("User-Agent", USER_AGENT);
3 post.setHeader("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
4 post.setHeader("Accept-Language", "en-US,en;q=0.5");
5 post.setHeader("Cookie", getCookies());
6 post.setHeader("Connection", "keep-alive");
7 post.setHeader("Content-Type", "application/x-www-form-urlencoded");
8 post.setEntity(new urlEncodedFormEntity(postParams));
9 HTTPResponse response = client.execute(post);
```

Code Snippet 4.3: Post request for user data

The user parameter sends HTTP POST to SimpleID server using java based HTTP Post. This method comes with java.net “HTTPurlConnection” library. To send the HTTP POST data, the request header parameters are set. The User-Agent parameter contains the information about the user agent originating the request. This field is for statistical purposes, the tracing of protocol violation and automatic recognition of the user agent to avoid user agent limitation. The Accept field denotes the content type acceptable for the responses. In above source code snippet, the accept field is a text or html format. The Accept-language field specifies the language acceptable for the response. In above snippet it is English. The Cookie field has the cookie value sent by server during previous conversation. The Connection parameter specifies the details for the server to send the message packets. The details like connection length, hop (mid nodes) receiving the packet are specified. In above snippet a permanent connection is set. This will keep the connection open till certain criterion are met. Normally connection is closed after each response. Content Type denotes the MIME (media) type of the body of the request. In above snippet the media type is application/x-www-form-urlencoded. For this media type
the post parameter (name value pairs) are sent in the form of giant query string with each name, value pair separated by ampersand symbol. The name is separated from value using ‘=’ symbol. The query string and the name value pair is sent as enlisted in Tab 4.7 and 4.8 respectively. The set Entity sets the query string denoted in the Tab 4.7 in the HTT POST request body. The response is then received by sending HTTP POST.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>login</td>
</tr>
<tr>
<td>destination</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>credentials</td>
</tr>
<tr>
<td>nonce</td>
<td>2015-04-25T09%3A46%3A35Zcaec272d</td>
</tr>
<tr>
<td>digest</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>test1</td>
</tr>
<tr>
<td>pass</td>
<td>abc123</td>
</tr>
<tr>
<td>autologin</td>
<td>1</td>
</tr>
<tr>
<td>op</td>
<td>Log+in</td>
</tr>
</tbody>
</table>

Table 4.7.: Query String

The query string is inserted in HTTP POST request to send the login request to the provider. The q name specifies the action to be done. Currently the value is login. This means the user wants to perform login action. Other value can also be Logout. Destination name specifies the page URL in which the user wants to land-in after login. By default null value specifies the user wants to land on home page. Mode specifies the type of input given by user for login. The value credentials specify the user want to give the username and password for authentication. Other modes could be authentication via certificate. The nonce field has a self generated unique value for each session. This value is generated in the start of conversation in SimpleID server. This value assures the security measure by ensuring that user cannot login with the duplicate nonce value again. Hence, if sniffer catches the username and password he/she cannot login in SimpleID server with same nonce value. For security reason, it is possible to send username and password in encrypted format. In such a case, the digest field encryption is used. In this packet md5(nonce + ‘:’ + md5 (username + ‘:’ + md5(password))) is sent. Since no encryption is used, the name field contains simple username in clear text, password field contains the password in clear text. The autologin field specifies the possibility of autologin in SimpleID server. Currently, the value is 1. The op name denotes the operation to be performed. Currently, the value of operation to be performed is login.
Chapter 4. Implementation

```java
Document doc = Jsoup.parse(html);
Elements inputElements = doc.getElementsByTagName("tbody");
List<NameValuePair> paramList = new ArrayList<NameValuePair>();
Elements rows = inputElements.select("tr");
Elements td = rows.get(i).children();
String company_value = td.get(1).text();
Map listA = new HashMap();
listA.put("company", company_value);
```

Code Snippet 4.4: parsing post response

<table>
<thead>
<tr>
<th>tr</th>
<th><a href="http://OpenID.net/schema/company/name">http://OpenID.net/schema/company/name</a></th>
<th>Example Company Limited</th>
</tr>
</thead>
</table>

Table 4.9.: SimpleID user’s Attributes

The response of HTTP POST comes in html/text form. The html parsing is done using Jsoup library to fetch the attributes. The source code snippet 4.4 is trying to fetch the company name attribute. In SimpleID the attribute name and value comes in Table row. The attribute name is present in the URI form whereas value is normal text value as shown in Tab 4.9.

Shibboleth OAuth bridge

The idea behind this implementation is to establish the successful communication between Shibboleth Identity provider and OAuth identity provider so that a OAuth user can be successfully authenticated against Shibboleth system (GWDG system). To achieve this, IMB plug-in is created to extend Shibboleth identity provider. The work of this plug-in is to transfer the user’s information such as username and password to the OAuth provider and fetch the response and pass the released attributes to the service provider with the help of data connector.

The Shibboleth identity provider connects with Google provider using an IMB plug-in. To achieve successful communication between plug-in and Google provider the Shibboleth’s Authentication module has been extended. The login handler module’s class present in “UsernamePasswordAuthHandler” servlet is modified.

The existing class checks the username and password against JAAS authentication configuration. If the user will give OAuth username and password, the authentication will fail and the OpenID module authentication check will take place, if the OpenID authentication fails then the control will go to OAuth authentication. For this the authenticateUser function is modified and new OAuth flow is added.
In Google authentication the user consent form is mandatory to be agreed by the user. Hence, there are two solutions designed to perform Google authentication.

**Solution 1:**

The plug-in takes the *client id, redirect URI, response type, scope, state* as input parameters and makes the URL of it. After the URL is successfully made, the plug-in tries to establish connection with the URL with the help of `HTTPURLConnection` library. After the connection is established, the plug-in sends the users parameter such as username, password etc. in the HTTP POST method. The HTTP POST method is used to send the user parameter to the Google server in form of query string. After getting the HTTP POST parameters Google server authenticates the user and sends the response. The response sent by the server returns the page for user consent as shown in Tab 4.10. It notifies the user that whether they agree to provide access to their data. The plug-in automates the accept request on the behalf of user using java selenium.

```html
<input type="hidden" id="submit_access" name="submit_access" value="">
<button id="submit_approve_access" type="submit" disabled tabindex="1" class="goog-buttonset-action">Akzeptieren</button>
<button id="submit_deny_access" type="submit" disabled tabindex="2">Abbrechen</button></form>
<br/>Wenn Sie auf "Akzeptieren" klicken, werden Sie weitergeleitet zu: HTTP://localhost:8080/OAuth2callback
</div>
</div><div id="tooltip_bubble"></div><script type="text/javascript">window.onload = lso.dynamicAdjustHeight;</script></body></html>

Table 4.10.: html form for user consent

Table 4.11 shows the post parameter passed by the plug-in in the query string to Google server. The HTTP POST requests the Google server to send the authentication code. As shown in Tab 4.11, the request parameter contains the URL with *client id*, *client secret*, *redirect URI*, *response type* and *scope*. The *client id* is received by the application while registering with the Google OAuth server. The *redirect URI* is the URI given to OAuth server by client application while registration. This URI describes the URI where the client should be redirected with the authentication code after authentication. The *scope* determines the scope or level of access client should be provided. The *response type* describes the representation format of the response expected by the OAuth server. Here in the table, *scope* is *email* as the client wanted to limit its access to the e-mail of the users, the *response type* is *code* as the client wants to get authentication code. With the URL, other parameters such as user *email id, password, signin* that are listed below are also send. The *Email* contains the email id of Google user, *Passwd* contains the password of Google user. *SignIn* describes the action to be performed for the Google account. The HTTP POST parameter and there description is listed in Tab 4.12.
Chapter 4. Implementation

Table 4.11.: Query String

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GALX</td>
<td>Contains the URL to which post request has to be sent</td>
</tr>
<tr>
<td>_utf8</td>
<td>Describes the utf8 format</td>
</tr>
<tr>
<td>bgresponse</td>
<td>Contains the</td>
</tr>
<tr>
<td>Email</td>
<td>Contains email id of the user</td>
</tr>
<tr>
<td>Passwd</td>
<td>Contains the password of the user</td>
</tr>
<tr>
<td>signIn</td>
<td>Contains the action to be performed</td>
</tr>
<tr>
<td>PersistentCookie</td>
<td>Contains the description about maintainance of cookie</td>
</tr>
</tbody>
</table>

Table 4.12.: Post request parameter

The Tab 4.13 shows the response returned by the Google server. The response contains the redirected URI given by the client during registration with Google server. The other parameters such as session state, authentication code, authuser, prompt are attached to the redirect URI. The authorization code contains the code which is required by client application to further request the resource server for the access token.

Table 4.13.: Authorization code Response

Once the client application receives the authentication code, it requests the Google OAuth server for the access token. The source code snippet 4.5 describes the flow for getting resource access token. The request parameter contains the client id, client secret, redirect URI, code, grant_type. The client id and secret are received by client application while registering with Google server. The redirect URI is the URL where client wants to land in after request fulfillment, it is given by the client application to Google server. The code is the authentication code and grant_type defines the grant type the user is requesting i.e authorization type in current case. It can be client credentials, authorization,
Chapter 4. Implementation

*implicit etc.*

```
0AuthClient OAuthClient = new OAuthClient(new urlConnectionClient());
0AuthAccessTokenResponse OAuthResponse = OAuthClient.accessToken(request);
0AuthBearerClientRequest bearerClientRequest =
    new OAuthBearerClientRequest("HTTPs://www.googleapis.com/0Auth2/v1/userinfo?access_token=")
    .setAccessToken(OAuthResponse.getAccessToken()).buildQueryMessage();
0AuthResourceResponse resrcResponse = OAuthClient.resource(bearerClientRequest,
    OAuth.HTTPMethod.GET, OAuthResourceResponse.class);
```

Code Snippet 4.5: Google authentication code flow

Table 4.14 is the Google Server response with the access token. The response returned contains the email id, name, gender, family name, given name, picture link, id of the Google user. Also response contains the the details about access token such as expiration time in seconds etc.

<table>
<thead>
<tr>
<th>Access Token:</th>
<th>ya29.YwGMywTASxcARbG4IKBiKFrpoqT4acWc84og8H9_D0Qq-pimx-JWm29omAutCnrkI0F_AIP91A2yYg, Expires in: 3600 resrc response Token:</th>
</tr>
</thead>
<tbody>
<tr>
<td>email: <a href="mailto:abc.def@gmail.com">abc.def@gmail.com</a>, verified_email: true, name: abc def, given_name: abc, family_name: def, link: <a href="HTTPS://plus.google.com/1075479713605497893">HTTPS://plus.google.com/1075479713605497893</a>, picture: <a href="HTTPS://lh5.googleusercontent.com/-F38NywNQL38/AAABAAAAAIA/AAAAAACAia/9N2OLJb67m8/photo.jpg">HTTPS://lh5.googleusercontent.com/-F38NywNQL38/AAABAAAAAIA/AAAAAACAia/9N2OLJb67m8/photo.jpg</a>, gender: female</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.14: Access token

After receiving the user attributes in the response the plug-in directs the attributes to the Shibboleth service provider. The plug-in performs this operation with the help of data connector.

**Solution2:**

In this various client parameters such as (ClientID, redirectURL) are taken by the IMB plugin. Based on this parameter an URL is formed. The user is redirected to Google authentication form. After user successfully accepts the agreement, the IMB plugin redirects the user back to the IMB plugin servlet with authorization code. The IMB plugin extracts the authorization code and retrieve the users attributes from the resource server. Then, the user data is passed to the authentication engine.The plugin than maps the fetched attributes to the Shibboleth attributes with the help of the data connector.

**Shibboleth Eduroam Bridge**

This plug-in establish successful communication between Shibboleth identity provider and Eduroam server. To implement this, the plug-in takes the user credentials as input and acts like a NAS device and laptop to connect to Eduroam server. The other method
is to implement a web browser plug-in that takes users credential and transfer it to the Shibboleth identity provider. The Shibboleth identity provider further connects to the Eduroam server. The detailed implementation for both the methods is described below.

**Authentication with web browser** In this the Shibboleth identity provider will take the username and password information from the user and then authenticates against the Radius server. During this, user credential will be exposed to the third party website i.e Shibboleth identity provider. This contradicts with the goal of actual designing and concept of Eduroam. Eduroam does not allows its user credential to be exposed to the third party website. Also, during Eduroam authentication, the NAS device do not know the user credential and packet information are also encapsulated during transmission via NAS device, but in this case shibboleth identity provider will know the credentials as well as packet exchange data. Since, GWDG is trustworthy organization, hence for the primary testing purpose the user credentials were given in GWDG Shibboleth identity provider browser.

For authentication purpose, libraries like Jradius\(^{12}\) tinyradius\(^{13}\) were analyzed. Several command line networking commands such as netsh for window, iwconfig, ifconfig, ifup were studied to setup WPA_Supplicant manually in the system. Also WPA_supplicant was installed and analyzed. I have implemented two solutions to authenticate the user against Radius server. One solution uses the Jradius library and other uses command line commands.

Shibboleth identity provider connects with Radius server using the IMB plug-in. To establish successful communication Shibboleth identity provider’s Authentication module was extended. The login handler module’s class present in “UsernamePasswordAuth-Handler” servlet is modified in Shibboleth authentication module.

The existing class in login handler module checks the username and password against JAAS authentication configuration. If the user will input Eduroam username and password, the normal JAAS authentication will be failed and the OpenID module authentication check will take place, if the OpenID authentication fails then the control will go to OAuth authentication. The OAuth authentication will also fail if username and password are not found, and control will go to the Eduroam module. For this the authenticateUser function is modified and new Eduroam flow is added.

The Eduroam authentication first takes the username and password from the user and writes it in network configuration file (edu.conf) using java buffer writer. The code snippet 4.6 shows the snippet of configuration file. The key_mgmt takes the acceptable key management protocol such as IEEE8021X, eap takes the acceptable EAP method such as MSCHAP, PEAP, PAP etc., the identity describes the user’s identity information.

---

\(^{12}\)http://jradius-client.sourceforge.net/

\(^{13}\)http://tinyradius.sourceforge.net/
Chapter 4. Implementation

The password field specifies the password of the user against which EAP authentication has to take place. ca_cert contains the path to the certificate. If the certificate is not given then Radius server will not be verified. This may lead to security issues. phase2 is given for inner authentication with TLS tunnel. The value ‘auth=MSCHAPV2’ is used for PEAP authentication. The anonymous_identity is the identity with a realm to hide username. In the Listing 4.6 user is a realm to hide actual username. The second part “example.com” will not be hided as it is used to track the actual Radius server by the proxy server for the authentication.

```plaintext
1 network={
2     key_mgmt=IEEE8021X
3     eap=PEAP
4     eapol_flags=0
5     identity="user@example.com"
6     password="foobar"
7     ca_cert="/etc/cert/ca.pem"
8     phase2="auth=MSCHAPV2"
9     anonymous_identity=user@example.com
10 }
```

Code Snippet 4.6: configuration file snippet

After updating the configuration data in edu.conf file, the eapol_test binary is called to check the authentication of the user with the help of process builder command. The binary is given with the Radius server ip address, number of re-authentication, expiration time, shared key, Radius connect information and network configuration file as shown in snippet below. In this the shared key is the key generated by radius server for the NAS IP address.

```plaintext
```

The response returned by process builder may be successful or failed authentication. Also the response contains some configuration attributes for NAS device. The snippet of response is shown in Tab 4.15. The snippet shows the authentication is completed successfully. Also, the response contains the attributes like NAS port, ip address, calling station id etc. This attributes are used for the NAS configuration and authorization purpose.
Chapter 4. Implementation

Radius packet matching with station
MS-MPPE-Send-Key (sign) - hexdump(len=32): 18 9e 13 80 4f 9f 53 46 aa a9 d5 83 9d 4a e0 af fa bb 54 08 f4 a9 03 13 13 8f e8 cf 9a 44
MS-MPPE-Recv-Key (crypt) - hexdump(len=32): 61 cd 2a fe 42 82 e5 a5 bd b0 eb 47 64 23 e5 f0 e6 35 58 3d d3 ed 58 03 11 73 81 72 19 3f 87

decapsulated EAP packet (code=3 id=4 len=4) from Radius server: EAP Success
EAPOL: Received EAP-Packet frame
EAPOL: SUPP_BE entering state REQUEST
EAPOL: getSuppRep EAP. EAP entering state RECEIVED
EAP: Received EAP-Success
EAP: EAP entering state SUCCESS
CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
EAPOL: SUPP_PAEBE entering state AUTHENTICATED
EAPOL: SUPP_BE entering state RECEIVE
EAPOL: SUPP_BE entering state SUCCESS
EAPOL: SUPP_BE entering state IDLE
eapol_sm_cb: success=1
EAPOL: Successfully fetched key (len=32)

PMK from EAPOL - hexdump(len=32): 61 cd 2a fe 42 82 e5 a5 bd b0 eb 47 64 23 e5 f0 e6 35 58 3d d3 ed 58 03 11 73 81 72 19 3f 87
cmplx
EAP: deinitialize previously used EAP method (25, PEAP) at EAP deinit
ENGINE: engine deinit
MPPE keys OK: 2 mismatch: 0 SUCCESS

Table 4.15.: EAP authentication response

The above authentication process describes the authentication using command line. The code snippet 4.7 describes the client authentication using Jradius library. The code takes username, password, Radius server IP address, shared secret, outer authentication protocol and inner authentication protocol (protocol to be used after tunnel formation) as input and gives the authentication information and attributes as output.

```java
  AttributeDictionaryImpl");
InetAddress host = InetAddress.
  getByName("Radius/uni2423ip/uni2423adress");
RadiusClient rc = new
  RadiusClient(host, "shared/uni2423secret", 1812, 1813, 1000);
AttributeList attrs = new
  AttributeList();
attrs.add(new
  Attr_UserName("username"));
attrs.add(new
  Attr_NASPortType(Attr_NASPortType.Wireless80211));
attrs.add(new
  Attr_NASPort(new Long(1)));
RadiusRequest request = new
  AccessRequest(rc, attrs);
request.addAttribute(
  new
  Attr_UserPassword("password"));
System.out.println("Sending:
" + request.toString());
RadiusResponse reply = rc.authenticate((AccessRequest)request,
  new
  EAPMSCHAPv2Authenticator(), 5);
System.out.println("Received:
" + reply.toString());
boolean isAuthenticated = (reply instanceof AccessAccept);
String replyMessage = (String)reply.getAttributeValue(Attr_ReplyMessage.TYPE);
if (replyMessage != null)
{
  System.out.println("ReplyMessage:
" + replyMessage);
}
```

Code Snippet 4.7: Jradius Authentication

54
After the user is authenticated the user successfully logins to Shibboleth identity provider creating a session id.

**Authentication with web browser plug-in** In this method, the idea is to design a web browser add-on plug-in. This plug-in will take the user’s Eduroam credentials and pass them to Shibboleth identity provider. The Shibboleth identity provider will receive the conversation either by use of websocket connection or HTTP connection. The Shibboleth identity provider will further send the user information to the Radius authentication server. As soon as identity provider hears access accept in response, the user will be authenticated and the user will be able to login successfully.

By the development of such system the Eduroam credentials will be accessible by the plugin. The Shibboleth identity provider will only receive the EAP packets and the Radius packets. Hence, the user data will not be exposed to the Shibboleth identity provider resulting in greater security.

**Client server protocol**

The aim of this project is to establish the communication between the browser and the Radius server with the web server as an intermediate. To establish a secure communication between the web server and the browser we need to develop a client side extension. This extension should be able to communicate with the web server using client server protocol. This can be achieved with HTTP connection or Web Socket programming.

HTTP is an uni-directional protocol. In this, the client sends the request to the server. The server processes the request and sends the response to the client. The websocket is a bi-directional protocol. In this client and server both can send message to each other.

The HTTP connection is half duplex. In this either client sends the request or server sends the response at a time. Hence, there is only one communication taking place at a time. The Web Socket connection is full duplex. The client and server can send the message to each other at the same time.

During each request/response procedure, the HTTP opens a new TCP connection whereas the entire communication for a session in Web Socket takes place through single TCP connection.

**Browser Extension Vs Server embedded Java Script**

To establish a secure communication between the browser and the web server, one need to define a client side interface to interact with the user. This client side interface can either be developed as an extension in web browser or there can be embedded client side code in the web server to interact with the user.
With the extension, one can implement the same functionality in various web applications i.e. the same Shibboleth system extension can be used for other identity management system with slight configuration changes. The embedded JavaScript works for a particular web application.

The user can download the extension, test it and then can use it with the browser to interact with the web server. On the other hand, the server embedded JavaScript may get updated from time to time without acknowledgment to the user. The user cannot test for the authenticity of the JavaScript code embedded in the web server. Hence, extension based method is more secure and reliable to use from the user’s perspective.

The extension needs to be downloaded and setup during frequent updates/code additions. This may lead to wastage of time and resources. On the contrary, the embedded JavaScript will be updated in the server. Hence, whenever browser will use it, fresh JavaScript code will be extracted automatically.

The extension has to be coded in different ways based on the browser compatibility. Also, whenever browser will be updated and their is change in UI functionality of the browser then, the extension needs to update itself.

The Extension’s UI design

This module describes the basic user interface design for the extension plugin. The input controls that need to be present in UI are as follows:

- The “Security Type” will always be WPA2 enterprise for Eduroam authentication.
- The “Inner Authentication/Authentication Method” contains various protocols such as PAP (Password Authentication Protocol), CHAP (Challenge Handshake Authentication Protocol), Microsoft CHAP (MS-CHAP), Microsoft CHAP version 2 (MS-CHAP-V2), EAP-MD5 Challenge (EAP-MD5), EAP-Generic Token Card (EAP-GTC). The inner protocol used depends on the network authentication protocol selected. On selecting TTLS network authentication method the PAP, CHAP, MS-CHAP, MS-CHAP-V2, EAP-MD5, EAP-GTC & EAP-TLS protocol becomes available. On selecting PEAP authentication method EAP-MS-CHAP-V2, EAP-MD5, EAP-TLS, EAP-GTC, EAP-SIM protocol becomes available.
- The “Trusted Root Certificate/CA certificate” requires the digital certificate of the home institution.
• The “Anonymous Identity” is the identity provided by the user to mask his actual username from the web server/NAS device and proxy servers. The proxy server redirects the user data based on the domain name provided in the second half of the identity.

• The “Username” is the user’s identity name for the Eduroam.

• The “Password” is the password for the Eduroam identity.

• The “Certificate” may be given by user instead of username and password in some authentication protocols.

**Communication between Extension, Web Server and Radius Server**

Figure 4.2 shows the working of web socket communication between the extension, web server and Radius server. The extension acts as a web socket client and the web server acts as a web socket server. To initiate the communication the web server establishes the web socket communication with the client. After successful communication establishment [Handshake], the web server sends the EAP request to the client. This EAP request packet will be made with the help of Eapol.c program present in WPA_Supplicant. The EAP packet is sent in String format [convert byte to string] or in JSON format or as Array Buffer or as BLOB inside the web socket. The extension sends the EAP response to the web server. This response contains the user information in encrypted format. The web server sends the Radius Access Request to the Radius server in the form of Radius packets. The Radius packets contains the EAP response packets received from the extension. The Radius packets can be made using JRadius client library or TinyRadius library from java. JRadiusClient is an implementation of a Radius Client compliant with IETF RFC 2865 and 2866. It is intended to be used as a library to build your own Network Access Server. The tiny Radius sends and receives the Radius packets. This uses CHAP and PAP authentication for Access Request messages. The Radius server sends the Radius Access challenge based on the authentication type set by the user. The web server sends the EAP Request packet to the extension through the open web socket connection to request for the credentials/certificate based on the authentication set by the user. The extension sends the EAP response to the web server through the open web socket connection. The web server then sends the Radius Access Request to the Radius server through JRadius Client or Tiny Radius. The Radius server returns the Radius Access Accept packet to the web server. The web server sends the EAP Success packet through the Web Socket connection. The web server closes the Web Socket connection.


Figure 4.3 shows the working of HTTP communication between the extension, web server and Radius server. The extension acts as a HTTP client and the web server acts as a HTTP server. To initiate the communication the web server establishes the HTTP communication with the client. After successful communication establishment [Handshake], the extension sends the EAP request to the web server. This EAP request packet will be made with the help of Eapol.c program present in WPA_Supplicant. The EAP packet is sent in encoded BASE 64 format or in JSON format or as MIME application/octet-stream inside the HTTP POST request in the entity field of the HTTP connection. The web server sends the Radius Access Request to the Radius server in form of Radius packets. The Radius packets contain the EAP request packets received from the extension. The Radius packets can be made using JRadius client library or TinyRadius library from java. JRadiusClient is an implementation of a Radius Client.
compliant with IETF RFC 2865 and 2866. It is intended to be used as a library to build your own Network Access Server.\(^{16}\) The tiny Radius sends and receives the Radius packets. This mechanism uses CHAP and PAP authentication for Access Request messages.\(^{17}\) The Radius server sends the Radius Access challenge based on the authentication type set by the user. The web server sends the EAP Request packet to the extension in the form of HTTP response to request for the credentials/certificate based on the authentication set by the user. The extension sends the EAP response to the web server through the HTTP post request method. The web server then sends the Radius Access Request to the Radius server through JRadius Client or Tiny Radius. The Radius server returns the Radius Access Accept packet to the web server. The web server sends the EAP Success packet in the HTTP response. The web Server closes the HTTP connection.

\(^{16}\)HTTP://jradius-client.sourceforge.net/
\(^{17}\)HTTP://tinyradius.sourceforge.net/
General Attributes Released

Table 4.16 shows the most common attributes released from Radius server after the user authentication. This may not be the complete list. The attributes released depends on the Radius server.
User-Name  
Service-Type  
Framed-Protocol  
Framed-IP-Address  
Framed-IP-Netmask  
Framed-Routing  
Framed-MTU  
Login-Service  
Login-TCP-Port  
Callback-Number  
Callback-Id  
Framed-IPX-Network  
State  
Session-Timeout  
Idle-Timeout  
Termination-Action  
Login-LAT-Service  
Login-LAT-Node

Table 4.16.: List Of Attributes in Access Accept packet

4.3.3. Shibboleth Data connector extension

After getting the attribute value from all other identity providers the value is passed to Shibboleth service provider using a data connector. Currently only Shibboleth identity provider attributes are passed to Shibboleth service provider. To pass attribute values from all other identity providers, Shibboleth attribute passing functionality is extended. This is achieved by extending base data connector class. This is done with the help of resolve function. In this function if the user is verified using JAAS authentication then the attribute value comes with the principal else the attribute value comes in the list. The attribute value are fetched from the list and mapped to the attribute name of Shibboleth attribute. The Code snippet 4.8 shows the basic snippet for adding custom attributes.

```java
public class MyDataConnector extends BaseDataConnector {
    private String lookupurl;
    public MyDataConnector(String URL) {
        this.lookupurl = URL;
    }
    public Map<String, BaseAttribute> resolve(ShibbolethResolutionContext resolutionContext) throws AttributeResolutionException {
        Map<String, BaseAttribute> result = new HashMap<String, BaseAttribute>();
        String username = resolutionContext.getAttributeRequestContext().getPrincipalName();
        // The attributes from other idp's are added here the result here.
        return result;
    }
}
```

[HTTPs://wiki.Shibboleth.net/confluence/display/SHIB2/IdPDevExtDataCtr](HTTPs://wiki.Shibboleth.net/confluence/display/SHIB2/IdPDevExtDataCtr)
Chapter 4. Implementation

Code Snippet 4.8: BaseDataConnector extension snippet

The schema definitions are added in meta-inf inside schema files “spring.handlers” and “spring.schema”. The attribute.xsd file located in schema contains all the required details about attributes such as attribute name, value etc. to be fetched from attribute-resolver.xml. The snippet 4.9 shows the attribute-resolver.xml file. The import tag in snippet below specifies the schema location. The attribute details are given in attribute tag inside the complex type tags.

```
<schema

targetNamespace="HTTP://ukfederation.org.uk/schemas/uaattribute/resolver"
xmlns="HTTP://www.w3.org/2001/XMLSchema"
xmlns:resolver="urn:mace:Shibboleth:2.0:resolver" elementFormDefault="qualified">
  <import namespace="urn:mace:Shibboleth:2.0:resolver" schemaLocation="classpath:/
  schema/Shibboleth-2.0-attribute-resolver.xsd" />
  <complexType name="UserAgentMappedAttributes">
    <complexContent>
      <extension base="resolver:BaseDataConnectorType">
        <sequence>
          <element name="Mapping" minOccurs="0" maxOccurs="unbounded">
            <complexType>
              <attribute name="cidrBlock" type="string" use="required" />
              <attribute name="attributeId" type="string" use="required" />
              <attribute name="attributeValue" type="string" use="required" />
            </complexType>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

Code Snippet 4.9: Attribute resolver.xml

To add the data connector, the resolver tag also needs to be added in attribute-resolver.xml file. The namespace declaration is done inside this tag. Also the xml schema file location is added. In the snippet 4.10 xmlns:uadc and xsi:schemaLocation value specifies the namespace and location of schema file.

```
<resolver:AttributeResolver
  xmlns:resolver="urn:mace:Shibboleth:2.0:resolver"
  xmlns:xsi="HTTP://www.w3.org/2001/XMLSchema-instance"
  xmlns:pc="urn:mace:Shibboleth:2.0:resolver:pc"
  xmlns:ad="urn:mace:Shibboleth:2.0:resolver:ad"
  xmlns:dc="urn:mace:Shibboleth:2.0:resolver:dc"
  xmlns:enc="urn:mace:Shibboleth:2.0:attribute:encoder"
  xmlns:sec="urn:mace:Shibboleth:2.0:security"
  xmlns:uadc="HTTP://ukfederation.org.uk/schemas/uaattribute/resolver"
  xsi:schemaLocation="urn:mace:Shibboleth:2.0:resolver:classpath:/schema/Shibboleth
  -2.0-attribute-resolver.xsd" urn:mace:Shibboleth:2.0:resolver:pc:classpath:/schema/Shibboleth
  -2.0-attribute-resolver-pc.xsd" urn:mace:Shibboleth:2.0:resolver:ad:classpath:/schema/Shibboleth
  -2.0-attribute-resolver-ad.xsd" urn:mace:Shibboleth:2.0:resolver:dc:classpath:/schema/Shibboleth
  -2.0-attribute-resolver-dc.xsd"
Chapter 4. Implementation

Additionally, the data connector is added as shown in the snippet 4.11. The data connector contains an **unique ID**, type defined in **Complex type** tag of xsd file. The connector also contains the attribute details such as attribute value, attribute id etc.

```
1 <resolver:DataConnector id="myIP" xsi:type="uadc:UserAgentMappedAttributes">
2 <uadc:Mapping cidrBlock="127.0.0.1/32" attributeId="cn" attributeValue="" />
</resolver:DataConnector>
```

**Code Snippet 4.11: attribute-resolver.xml dataconnector**
5. Existing work

This chapter compares the IMB plug-in with existing works such as eduGAIN and Umbrellaid.org.

5.1. IMB plug-in vs eduGAIN

eduGAIN is an international inter-federation service which helps in interconnecting research and education identity federations. It enables secure exchange of information related to identity, authentication and authorization between participating federations.

![eduGain structure](image)

Figure 5.1.: eduGain structure

Figure 5.1 shows eduGain as a service that interconnects participating federation. This federations agree on a set of common standards and policies to ensure interoperability. The edugain service enables the trustworthy exchange of information related to users as well as authentication and authorization between members. All user who have their accounts in federation under edugain can login with their organizational credentials.

and use other organization’s service. All organizations trust each others authentication information.

Figure 5.2. shows the eduGain working. The eduGain is one platform under which all federations work. Suppose, User A from institution M federation wants to login in GWDG federation to use service X. As soon as user A chooses that he is from institution M, GWDG will redirect him to institution M’s login page. After authenticating user A, institution M will redirect user A to use the GWDG service in GWDG portal.

Figure 5.2.: eduGain working
Figure 5.3 shows the idea of this project. User A from institution M wants to use service X from GWDG. User A logins with institution M credentials in GWDG portal. User A data will be sent to institution M. After checking users data institution M will return the response to GWDG. Based on response, GWDG will provide the user with service access or denial.

The difference between my work and eduGAIN is as follows:

- In eduGain, user is authenticated by their home identity provider and authentication acknowledgment is sent to the service provider whose service is desired to be used by the user. But, in my work the user is authenticated by their home identity provider with Shibboleth identity provider in the middle. Hence, no authentication acknowledgment needs to be sent.

- In eduGain the users credential are only exposed to the home identity provider whereas in my work users credential are exposed to Shibboleth identity provider.

- In eduGain the user has to trust on redirected URL and has to feed in credentials into it. Whereas, in my work there is no redirection hence, user has to trust only Shibboleth identity provider.
5.2. IMB plug-in vs Umbrellaid.org

Umbrella is the pan-European federated identity system for the users of the European large photon/neutron facilities. In this system there is only one identity provider. The user is given a single identity and with this identity user can access different services of the participating facility.

![Umbrella Diagram]

Figure 5.4.: Umbrella

Figure 5.4 above shows the basic idea of umbrella. User abc creates an umbrella account. User abc then links his account in photon/neutron research facilities with the umbrella account by login into the application. The user now just needs to remember his umbrella account credentials to login into his various accounts. The existing four other accounts are now permanently linked to each other.

\[\text{HTTPs://www.umbrellaid.org/euu/}\]
Figure 5.5 shows the idea of this thesis work. In this work the users can login into GWDG portal using credentials from any organizations trusted by GWDG. In this thesis work there is no central umbrella. The aim of this project is to allow user \(abc\) to login into the system with the local account. If user \(abc\) have account in any organization trusted by GWDG then he can login with that organization credential. He is not required to create a new account in GWDG.

The difference between Umbrellaid.org and my work is that, Umbrellaid.org is a centralized approach. The user has to create a central account and link his existing account to it. My work is not based on a centralized approach. The user is not required to create an account in GWDG if he has a account in other associated identity management system. Also, there is no inter linking of account in my work.
6. Test Results

Under this, I have performed the tests using Apache Jmeter to measure the time taken by each protocol for the successful authentication and authorization. The test was performed for 10 parallel user threads. Each user login with the same protocol. The response time is measured for each login request. There are two response time. One is for Unsolicited URI SSO and other is for Post using login credentials. For Unsolicited URI specific request do not originate from service provider. Instead the profile handler specific IDP url is given to initiate the authentication request. In Post using login credentials the login request originates from service provider.

The Figure 6.1 shows the results for the user authentication with Shibboleth.

![Response Time Graph]

Figure 6.1.: Test result : Authentication with Shibboleth

Figure 6.2. shows the test result for user authentication failure with Shibboleth and successful authentication with OpenID. The response time is almost same as Shibboleth authentication.
Chapter 6. Test Results

Figure 6.2.: Test result: Shibboleth Authentication failure and OpenID success

Figure 6.3. shows the test results for the Shibboleth Authentication failure, OpenID authentication failure and Eduroam authentication success. The response time here is almost same as Shibboleth authentication.

Figure 6.3.: Test result: Shibboleth Auth failure, OpenID failure, Eduroam Success

The test for OAuth protocol was not performed as Google needs manual clicking on the user consent form. Hence, JMeter is not able to automate the login process.
7. Conclusion & Future Work

The aim of this thesis was to develop a solution to accomplish the communication between Shibboleth identity management system and other protocol based identity providers. For this, three different protocols based identity providers were considered. At the end Shibboleth identity management system was able to successfully authenticate the users who had their identity information stored in other protocol based identity provider.

In this thesis, I have implemented authentication and authorization of OpenID and OAuth identity provider based users via Shibboleth identity management system. The solution for Eduroam based user authentication is fully implemented for web browser based authentication but partially implemented for web browser plug-in based authentication. I have designed the solution for the web browser plug-in based authentication. In the future, this thesis work can be extended to implement the designed solution in a production system of an organization.

In the future, the realization of this thesis can be done. I have only considered the communication initiating from Shibboleth identity management system. The solutions for initiating the communication from other protocol based identity management system with other protocol based identity providers can be designed.
References


A. Installation Guides

A.1. Shibboleth 2.x Installation

A.1.1. Basic installation

The complete Shibboleth installation requires the installation of Shibboleth identity provider and Shibboleth service provider. Shibboleth installation encapsulates the installation process followed by configuration. The installation process generates the IDP’s entity id, IDP’s initial metadata, the key pair with self-signed certificate and IDP configuration files. Prior to installation the SSL certificate (used to secure IDP’s browser) and source of SAML metadata for service provider is required.

System requirements

<table>
<thead>
<tr>
<th>Product</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.5 &amp; Higher</td>
</tr>
<tr>
<td>Apache tomcat server</td>
<td>6.0 &amp; Higher</td>
</tr>
<tr>
<td>Apache web server</td>
<td>2.2 &amp; Higher</td>
</tr>
</tbody>
</table>

Table A.1.: System Requirements for Shibboleth Installation

The basic requirements of system for IDP installation are:

A.1.2. Shibboleth Identity provider Installation & Configuration

Installation begins by setting up Apache tomcat 8.0 server available in Apache tomcat’s official website. I am using the Apache web server installed in WAMP server (2.5). Shibboleth IDP is downloaded from Shibboleth website. Once the IDP zip file is

1https://wiki.shibboleth.net/confluence/display/SHIB2/IdPInstall
2http://shibboleth.net/downloads/identity-provider/2.4.3/shibboleth-identityprovider-2.4.3-bin.zip
downloaded the file is unzipped and install.bat file is executed for windows. The installation path is set to “path/to/opt/Shibboleth-idp” and domain name is set to “127.0.0.1” i.e. localhost.

The configuration of Apache webserver requires the addition of “ProxyPass /IDP/ ajp://localhost:8009/IDP/” line in “HTTPd.conf” file. This is done to pass the IDP requests from HTTP to tomcat. We are doing username/password based authentication so, the user’s credential information on LDAP server. The Apache tomcat server configuration requires to add “<Connector port='8009' enableLookups='false' redirectPort='8443' protocol='AJP/1.3' request.tomcatAuthentication='false' address='127.0.0.1' />” to ’/usr/local/tomcat/conf/server.xml’ file. This is done to send usernames to IDP and the stop tomcat server itself from authenticating the user.

The Shibboleth IDP installation generates the war file which is deployed in Apache tomcat server once above configuration is done. To deploy war file one can either paste it in “apache-tomcat-8.0.17\webapps” and delete the pre-existing IDP folder and war file. The other method could be to give the path of the war file to be deployed in “apache-tomcat-8.0.17\conf\Catalina\localhost\IDP.xml” file.

A.1.3. Shibboleth Identity provider source code installation:

The above installation was done to understand the basic working and ingredients of Shibboleth. For development of the plug-in, the Shibboleth identity provider source code installation is required. The Shibboleth source code is downloaded with SVN3. The source code is the maven project4. To run the IDP in eclipse the installation of check-style plug-in and TestNG framework is required. The latest Eclipse luna contains maven m2E. The jdk version 1.7 is used in our project. To import the project the file menu is opened in eclipse and the following sequence of procedure is done “import-existing maven project->select the downloaded project->finish”. Once the project is imported it can be build using maven clean followed by maven install. The maven install for Shibboleth 2.4.3 results in jar file in target folder.

Post installation the following configuration files needs to be changed present at (path/to/IdP/installation)/conf:

3https://svn.Shibboleth.net/java-shib-idp2/tags/2.4.3/
4https://svn.shibboleth.net/java-shib-idp2/tags/2.4.3/
Appendix A. Installation Guides

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>relying-party.xml</td>
<td>Add <code>&lt;metadata:MetadataProvider&gt;</code> with IdP’s and SP’s metadata.xml files path</td>
</tr>
<tr>
<td>login.config</td>
<td>Add LDAP connection parameters</td>
</tr>
<tr>
<td>logging.xml</td>
<td>Change logging level as per requirement</td>
</tr>
<tr>
<td>handler.xml</td>
<td>Allow only <code>&lt;ph:LoginHandler xsi:type=&quot;ph:UsernamePassword&quot;&gt;</code> and <code>&lt;ph:LoginHandler xsi:type=&quot;ph:PreviousSession&quot;&gt;</code> to provide proper login page to user</td>
</tr>
<tr>
<td>attribute-resolver.xml</td>
<td>Add <code>&lt;resolver:DataConnector&gt;</code> and <code>&lt;resolver:AttributeDefinition&gt;</code> to query attributes and set them to release</td>
</tr>
<tr>
<td>attribute-filter.xml</td>
<td>Add <code>&lt;afp:AttributeRule&gt;</code> to filter attributes before release to SP</td>
</tr>
</tbody>
</table>

Table A.2.: Shibboleth IdP configuration changes

A.1.4. Problem while building in Shibboleth 2.4.3

The missing “tools.jar” file error occurred as the eclipse was pointing to JRE instead of JDK. To overcome this JDK path was set in “window->preferences->java->Installed Jre’s”.

A.1.5. Problem with resulting jar file in Shibboleth 2.4.3

For the deployment of IDP in tomcat server war file is required. Jar file does not contain all the dependencies and required files.

A.1.6. Analysis for solution of problem:

The jar file renaming to war file was done. The war file was not able to give the same results as expected.

The new dynamic web project was made (file->new->other->dynamic web project->project name->finish). The web project was converted to maven project (right click->configure->convert to maven project->packaging (war)->name->finish). Tried to add the entire source file in newly created project but it was complicated due to the structural differences.

The creation of war file requires specific structure of maven project. We tried to recreate the structure by swapping the files. When project were rebuilt there were numerous built errors.
We also tried to discover the possibility of conversion of maven project into dynamic web project and then again reconverting to maven project. This was thought because conversion of maven project to dynamic web project will restructure the files required for war build. After reading various documents we found that no proper procedure was described for this technique.

After reading various documents it was concluded that pom.xml is the file that is used for jar or build. Hence, we tried restructuring ‘pom.xml’ file. Pom.xml details were read and structure was understood.

We tried modifying packaging tag to pom.xml file. The jar was replaced by war.

`<packaging>war</packaging>`

Also plug-in structure was also changed from jar to war

`<plug-in> <groupId>org.Apache.maven.plug-ins</groupId> <artifactId>maven-war-plug-in</artifactId> <configuration>`

The maven build was done and war file was generated. When this war file was deployed in Apache tomcat server the IDP didn’t got start. When the tomcat logs were checked “java.io.FileNotFoundException: Could not open ServletContext resource [/$IDP_HOME$/conf/internal.xml]” error occurred. The digging of error led to information that there is some issue with web.xml file present in web-INF folder.

When the difference between pre-installed web.xml and presently generated web.xml was analysed using win merge it was

Pre-installed `<param-value> file:/f:/opt/Shibboleth-idp//conf/internal.xml;file:/f:/opt/Shibboleth-idp//conf/service.xml;</param-value>`

present `<param-value> $IDP_HOME$/conf/internal.xml; $IDP_HOME$/conf/service.xml; </param-value>`

The value was hardcoded and the war file was again deployed and it runs successfully in outside the browser but inside eclipse it still give 404 error.

### A.1.7. Service provider installation

SP comes in binary format for Windows operating system which could be installed using windows installer. In Linux systems, SP could be installed using `apt-get install libapache2-mod-shib2` or it could be built from source. Following installation steps has been copied from shibboleth official wiki available at [https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPLinuxSourceBuild](https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPLinuxSourceBuild).
Appendix A. Installation Guides

The following packages must be built in the following order using the `./configure` commands listed. Unless specific version notes are given, any new release is sufficient. Be sure to `make` and `make install` as appropriate for each package, including Shibboleth itself. You will also need the Boost\(^5\) headers available (but don't need to actually build or install the full Boost library set). Due to a bug not yet fixed in a release, only versions up to 1.52 will work\(^2\).

- **log4shib**: `./configure --disable-static --disable-doxygen --prefix=/opt/shibboleth-sp`
- **Xerces-C**: `./configure --prefix=/opt/shibboleth-sp --disable-netaccessor-libcurl`
- **XML-Security-C**: `./configure --without-xalan --disable-static --prefix=/opt/shibboleth-sp`
- **XMLTooling-C**: `./configure --with-log4shib=/opt/shibboleth-sp --prefix=/opt/shibboleth-sp`
- **OpenSAML-C**: `./configure --with-log4shib=/opt/shibboleth-sp --prefix=/opt/shibboleth-sp`

When building Shibboleth, you can usually rely on the configure script to detect your Apache version and do the right thing, but in special cases, you can pass various options to control the Apache module version and locate the apxs script to use.

- **Shibboleth**: `./configure --with-log4shib=/opt/shibboleth-sp --enable-apache-24 --with-apxs2=/usr/local/apache2/bin/apxs --prefix=/opt/shibboleth-sp`

Here `/opt/shibboleth-sp` could be any path where shibboleth SP should be installed. Post installation the following configuration files needs to be changed present at `{path/to/SP/installation}/etc/shibboleth`:

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>shibboleth2.xml</td>
<td>Add <code>&lt;Host name=&quot;&quot;</code> of server, <code>&lt;ApplicationDefaults entityId=&quot;&quot;</code> of SP’s URL, <code>&lt;SSO entityId=&quot;&quot;</code> of IdP’s URL and <code>&lt;MetadataProvider&gt;</code> for IdP</td>
</tr>
<tr>
<td>attribute-map.xml</td>
<td>Add <code>&lt;Attribute&gt;</code> coming from IdP before passing on to service</td>
</tr>
</tbody>
</table>

Table A.3.: Shibboleth IdP configuration changes

Finally generate the SP’s metadata by accessing the URL `{SP’s URL}/Shibboleth.SSO/Metadata`. Share this metadata file with shibboleth IdP.

The trouble shooting guide is very handy as it lists the most frequent issues faced by the users. The guide is available at [https://wiki.shibboleth.net/confluence/display/SHIB2/Troubleshooting](https://wiki.shibboleth.net/confluence/display/SHIB2/Troubleshooting).

\(^5\)http://www.boost.org/
 Appendix A. Installation Guides

A.2. SimpleID installation

A.2.1. System requirements

The basic requirements of system for SimpleID installation are:

**Server Requirements**

<table>
<thead>
<tr>
<th>Product</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>web server</td>
<td>HTTPS support, able to write files to filesystem</td>
</tr>
<tr>
<td>PHP</td>
<td>5.3.0 or greater version, enable extensions [bcmath;pcre;session;xml;hash], register_global switched off</td>
</tr>
</tbody>
</table>

Table A.4.: Server Requirements

**Browser Requirements**

Browser should support jquery and javascript.

<table>
<thead>
<tr>
<th>Browser</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet explorer</td>
<td>6.0 or later</td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td>2.0 or later</td>
</tr>
<tr>
<td>Safari</td>
<td>2.0 or later</td>
</tr>
<tr>
<td>Opera</td>
<td>9.0 or later</td>
</tr>
</tbody>
</table>

Table A.5.: Browser Requirements

**HTTPs Support**

For SimpleID HTTPS support is required. This is a security measure so as to assure that the login forms from the connections are encrypted.

A.2.2. Download & Installation

Below are the steps followed for installation and setup of SimpleID installation:

- The latest SimpleID version is downloaded from [http://simpleid.koinic.net/]
The SimpleID file was moved under web server folder. Since, for this project Wamp server was used. Hence, project was moved in Wamp’s folder. There are four folders to be moved. The cache folder stores temporary data. This directory should be moved at a place which is writable by webserver not accessible publically. The other folder is the identities directory which stores user’s identity information, should be moved in the place readable by webserver. The next is the store directory which stores user settings. This should be moved in the place readable and writable by web server. The other is www directory. The actual code is written in this folder. This should be moved in the place which is accessible by the users.

Next step is to configure the config file. For this purpose config.PHP.dist was copied in web directory and renamed to config.PHP. The configurations were set based on requirement given in official documentation at http://simpleid.koinic.net/docs/1/#getting-started.