

THE OPENID BOOK

A COMPREHENSIVE GUIDE TO OPENID PROTOCOL AND
RUNNING OPENID ENABLED WEB SITES

OPENID

Getting started with OpenID
Creating OpenID-enabled web sites
Running OpenID server
OpenID in the Enterprise

RAFEEQ UR REHMAN, CISSP

Copyright © Notice

This book is copyright © 2007 of Rafeeq Ur Rehman. No part of this book can be distributed or reproduced in any form or shape without written permission of the Author and the Publisher.

Disclaimer

The book is made available without any direct, indirect, or implied warranty of any kind, including the correctness of material presented here. The author and the publisher of this book are not responsible for any direct or indirect loss as a result of use of this book.

Trademarks and Service Marks

All references to trademarks and service marks, used in this book are the property of respective owners.

Published By

Conformix Books, a division of Conformix Technologies Inc.

ISBN13: 978-0-9724031-2-2

ISBN: 0-9724031-2-4

Web: <http://www.openidbook.com>

<http://www.conformix.com>

Email: info@conformix.com

Table of Contents

1	INTRODUCTION TO OPENID	17
1.1	A FIVE MINUTE TOUR OF OPENID	20
1.1.1	<i>Creating an Account with Verisign PIP</i>	<i>21</i>
1.1.2	<i>Using PIP ID with LiveJournal Web Site</i>	<i>26</i>
1.1.3	<i>Using the OpenID to Login to Multiple Sites</i>	<i>29</i>
1.2	SO WHAT HAPPENED BEHIND THE SCENES?	30
1.3	ENTERING USER SETTING	31
1.4	OTHER IDENTITY PROVIDERS.....	33
1.5	OPENID IDENTITY URL COMPOSITION	33
1.6	CHAPTER SUMMARY.....	34
1.7	REFERENCES.....	35
2	AUTHENTICATION AND AUTHORIZATION	36
2.1	WHAT IS AN IDENTITY?.....	37
2.2	AUTHENTICATION AND AUTHORIZATION.....	38
2.3	USERNAME AND PASSWORD	39
2.4	AUTHENTICATION METHODS	40
2.4.1	<i>Password Authentication</i>	<i>40</i>
2.4.2	<i>PIN Authentication</i>	<i>41</i>
2.4.3	<i>One Time Password (OTP) Authentication.....</i>	<i>41</i>
2.4.4	<i>Smart Card Authentication.....</i>	<i>42</i>
2.4.5	<i>Biometric Authentication</i>	<i>42</i>
2.4.6	<i>Certificate Based Authentication.....</i>	<i>43</i>
2.4.7	<i>USB Devices</i>	<i>44</i>
2.5	WEAK AND STRONG AUTHENTICATION.....	44
2.5.1	<i>Two –Factor Authentication</i>	<i>45</i>
2.6	SINGLE SIGN-ON (SSO) AND FEDERATED IDENTITIES	45
2.7	IDENTITY MANAGEMENT DILEMMA	46
2.8	DIRECTORY SERVICES	47

2.9	RISK BASED AUTHENTICATION AND AUTHORIZATION	48
2.10	NEW AUTHENTICATION MECHANISMS	48
2.10.1	<i>OpenID</i>	49
2.10.2	<i>Microsoft CardSpace</i>	49
2.10.3	<i>Bandit</i>	49
2.10.4	<i>Higgins</i>	49
2.10.5	<i>LID</i>	50
2.10.6	<i>Yadis</i>	50
2.11	CHAPTER SUMMARY.....	52
2.12	REFERENCES.....	52
3	OPENID PROTOCOL AND MESSAGES	54
3.1	OPENID CONCEPTS AND TERMINOLOGY.....	55
3.1.1	<i>OpenID Definitions</i>	55
3.1.2	<i>Communication among OpenID System Components</i>	56
3.2	DIRECT AND INDIRECT COMMUNICATION.....	60
3.3	OPENID MODES OF OPERATION.....	60
3.3.1	<i>Dumb Mode Communications Flow</i>	60
3.3.2	<i>Smart Mode</i>	63
3.3.3	<i>Using Ajax with Dumb and Smart Modes</i>	68
3.4	OPENID IDENTITY URL PAGE.....	68
3.5	OPENID SPECIFICATION VERSIONS	70
3.6	OPENID MESSAGES	71
3.6.1	<i>The associate Request Message</i>	72
3.6.2	<i>The associate Response Message</i>	75
3.6.3	<i>The checkid_setup and checkid_immediate Request Messages</i>	77
3.6.4	<i>The checkid_setup and checkid_immediate Response Messages</i>	79
3.6.1	<i>The check_authentication Request Message</i>	82
3.6.2	<i>The check_authentication Response Message</i>	84
3.7	HOW OPENID WORKS: SOME SCENARIOS.....	84
3.7.1	<i>Scenario One: First Time Login to a Web Site Using OpenID in Dumb Mode</i>	85
3.7.2	<i>Scenario Two: Login to a Trusted Web Site Using OpenID in Smart Mode</i>	86

3.8	PROBLEMS SOLVED BY OPENID.....	86
3.9	OPENID SUPPORT IN DIFFERENT LANGUAGES	87
3.10	MAJOR COMPANIES SUPPORTING OPENID.....	88
3.11	CHAPTER SUMMARY.....	88
3.12	REFERENCES.....	89
4	CREATING OPENID CONSUMER WEB SITES	90
4.1	OPENID CONSUMER: STEP-BY-STEP PROCESSING.....	92
4.2	RUNNING A SIMPLE CONSUMER USING JANRAIN LIBRARY	93
4.2.1	<i>The Consumer System Configuration</i>	<i>94</i>
4.2.2	<i>PHP Configuration</i>	<i>94</i>
4.2.3	<i>Apache Configuration</i>	<i>95</i>
4.2.4	<i>Running the Consumer Example</i>	<i>96</i>
4.2.5	<i>URL Sent to the OpenID Server</i>	<i>98</i>
4.2.6	<i>Storage of Association Information</i>	<i>100</i>
4.3	DISCUSSION ON SAMPLE CONSUMER.....	102
4.3.1	<i>Sample index.php File.....</i>	<i>104</i>
4.3.2	<i>Sample try_auth.php File</i>	<i>105</i>
4.3.3	<i>Sample finish_auth.php File</i>	<i>107</i>
4.4	REQUESTING ADDITIONAL PARAMETERS USING SIMPLE REGISTRATION EXTENSION 108	
4.5	RISK BASED ACCESS CONTROL AND GRADED AUTHORIZATION.....	114
4.5.1	<i>Sample Web Site Using OpenID Consumer</i>	<i>115</i>
4.5.1.1	<i>Backend Database</i>	<i>117</i>
4.5.1.2	<i>Source Files</i>	<i>118</i>
4.5.1.3	<i>Application Logic</i>	<i>123</i>
4.5.1.4	<i>Using Sample Application</i>	<i>124</i>
4.5.2	<i>One-Time Authorization.....</i>	<i>128</i>
4.6	STORING CREDENTIALS BY OPENID CONSUMERS.....	128
4.7	WEB BROWSER SUPPORT AND BROWSER PLUGINS.....	128
4.8	OPENID LIBRARIES.....	129
4.9	CHAPTER SUMMARY.....	130
4.10	REFERENCES.....	130
5	RUNNING OPENID SERVER	132

5.1	PHP OPENID SERVER INSTALLATION	135
5.1.1	Downloading and Extracting Files.....	135
5.1.2	Configuring Apache	136
5.1.3	Installing Smarty	137
5.1.4	Install and Configure JanRain PHP OpenID Library	138
5.1.5	Configuring MySQL Database.....	138
5.1.6	Updating Configuration Files	139
5.1.7	Testing Server	142
5.1.8	Testing Consumer with the Server	148
5.1.9	Database Changes during Server Configuration.....	152
5.1.10	Database Table Description.....	157
5.2	DEEP DIVE INTO OPENID PROTOCOL: DUMB MODE	160
5.2.1	Yadis and XRD Document.....	160
5.2.2	Indirect communication between Consumer and Identity Provider	162
5.2.3	Identity Provider Asks User for Authentication to IdP.....	166
5.2.4	Identity Provider's Positive Assertion to Consumer.....	167
5.2.5	Verification between Consumer and Identity Provider using check_authentication Message	170
5.2.6	Authentication Completion	171
5.3	OPENID ASSOCIATION MESSAGES.....	173
5.4	DIFFIE-HELLMAN (DH) KEY EXCHANGE MECHANISM	176
5.4.1	Basic Process for Generating DH Keys	176
5.4.2	Diffie-Hellman Variants.....	178
5.5	CHAPTER SUMMARY.....	179
6	OPENID EXTENSIONS	180
6.1	SIMPLE REGISTRATION OR PROFILE EXCHANGE.....	182
6.1.1	How It Works	183
6.1.2	Typical Use Cases.....	183
6.1.3	Message Description.....	184
6.1.4	Simple Registration and Yadis	187
6.2	OPENID SERVICE KEY DISCOVERY	187
6.2.1	How it Works.....	188
6.2.2	Typical Use Cases	188

6.2.3	<i>Message Description</i>	189
6.3	HOW TO SUBMIT NEW SPECIFICATION	189
6.4	CHAPTER SUMMARY.....	189
6.5	REFERENCES.....	190
7	OPENID AS ENTERPRISE SOLUTION	191
7.1	CROSS COMPANY AUTHENTICATION SOLUTIONS AND OPENID	192
7.1.1	<i>General Architecture for OpenID Cross Company Authentication</i>	193
7.1.2	<i>User Interface for OpenID Server</i>	195
7.1.3	<i>Partner/Hosted Web Sites</i>	196
7.1.4	<i>Security Controls</i>	196
7.2	SECURE OPENID AND DIGITAL CERTIFICATES.....	196
7.2.1	<i>Certifi.ca</i>	197
7.2.2	<i>Prooveme</i>	198
7.2.3	<i>Getting Free Certificates</i>	203
7.3	OPENID AND OPENSso.....	203
7.4	CHAPTER SUMMARY.....	203
7.5	REFERENCES.....	204
8	OPENID PROTOCOL: MISCELLANEOUS TOPICS	205
8.1	OPENID SECURITY ISSUES	206
8.1.1	<i>Relay Attacks</i>	206
8.1.2	<i>Phishing Attack</i>	206
8.1.3	<i>OpenID and Use of SSL</i>	207
8.1.4	<i>OpenID and Browser History</i>	207
8.2	OPENID AND PRIVACY	208
8.2.1	<i>Saving OpenID Credentials on Identity Provider Web Sites</i>	208
8.2.2	<i>Identity Providers Logging</i>	208
8.3	OPENID FOR DESKTOP CLIENTS AND MISCELLANEOUS USES OF OPENID	209
8.3.1	<i>Send-a-Message Protocol</i>	209
8.4	CHAPTER SUMMARY.....	209
9	GLOSSARY.....	210
10	REFERENCES AND USEFUL LINKS.....	212

10.1	REFERENCES.....	212
10.2	RFCs	213
10.3	OPENID LIBRARIES.....	214
10.4	OPENID PROVIDERS	214
10.5	MISCELLANEOUS.....	215
11	INDEX	216
12	ADVERTISEMENT IN THE BOOK	217

Table of Figures

Figure 1-1: Verisign PIP web site home page	22
Figure 1-2: Creating a new account using Verisign PIP web site.....	23
Figure 1-3: PIP account settings.....	25
Figure 1-4: LiveJournal login page.....	27
Figure 1-5: Authorization and trust request.....	28
Figure 1-6: LiveJournal home page after login using OpenID.....	28
Figure 1-7: PIP Profile Categories	32
Figure 1-8: PIP activity log.....	32
Figure 2-1: Yadis input and output	51
Figure 3-1: Communication among different components of the OpenID System with the URI identifier and the Identity Provider on the same machine.	58
Figure 3-2: Communication among different components of the OpenID System with Identifier URI and Identity Provider are on different machines.....	59
Figure 3-3: Dumb mode communications	63
Figure 3-4: Smart mode communication flow, during the first time login.....	65
Figure 3-5: Smart mode communication flow and the subsequent login where the Consumer and Identity Provider have already established a shared secret.	67
Figure 3-6: The flow for the checkid_setup request message.	79
Figure 3-7: The flow for the checkid_setup response message.	82

Figure 4-1: Running sample Consumer application included in the JanRain library.....	97
Figure 4-2: Sample client authentication showing <i>success</i> with the Identity Provider	99
Figure 4-3: Use of sample client source files during authentication request processing.....	104
Figure 4-4: Requesting multiple optional parameters using simple extensions.	110
Figure 4-5: Requesting multiple <i>optional</i> and <i>required</i> parameters using simple extensions.....	112
Figure 4-6: Response for multiple parameters.	114
Figure 4-7: Main page for sample Knowledgebase application	124
Figure 4-8: Login page for sample Knowledgebase application	125
Figure 4-9: Read-only access to knowledgebase.....	126
Figure 4-10: Update access to knowledgebase.....	126
Figure 4-11: Updating articles in the database.....	127
Figure 4-12: Full access to knowledgebase	127
Figure 5-1: Set up used for OpenID server, Consumer, and User Agent.	134
Figure 5-2: The welcome screen for the OpenID server.	143
Figure 5-3: New user registration with the OpenID server.	144
Figure 5-4: Completion of your OpenID user registration process.....	146
Figure 5-5: Creating OpenID profiles	148
Figure 5-6: Using the sample Consumer application to test new OpenID URL.	149

Figure 5-7: List of optional parameters requested by the Consumer	149
Figure 5-8: Confirmation of OpenID authentication.....	151
Figure 5-9: Trusted web site.....	151
Figure 5-9: Request and response messages between IdP and Consumer	162
Figure 7-1: A typical enterprise OpenID environment enabling CCA.....	194
Figure 7.2: List of certificates in Internet Explorer.	200
Figure 7-3: List of certificates in Firefox browser.	201
Figure 7-4: Detail of Prooveme certificate in Firefox.....	202

Preface, Acknowledgements and Introduction

This book provides discussion around OpenID, authentication, and authorization.

Acknowledgements

I am thankful to all of the following who helped in the preparation of manuscript for this book.

- Steve Kerns helped in proof reading as well as gave many suggestions.
- Todd Sharer gave many useful suggestions and I am thankful to Todd for encouragement.
- Ryan Fitzpatrick from LiveJournal gave permission to add screenshots from LiveJournal.
- Gary Krall from Verisign Labs gave permission to add screenshots from Verisign PIP web site.

I am thankful to all other friends and well-wishers for their encouragement and useful suggestions. Above all, I am thankful to all who contributed to OpenID to make it useful for community.

Questions, Comments, Criticism, Appreciations

Please contact the Author, Rafeeq Ur Rehman, at rafeeq.rehman@gmail.com for any questions or comments or provide any feedback that can be helpful in the next version of this book. All and any critique is welcomed.

You can also put your comments on Author's blog web site for OpenID at URL
<http://openid.blogspot.com>



Conducted by

Rafeeq U Rehman

CISSP

On-Site Training
for a group of 5 or more

Contact Conformix Technologies Inc
at
info@conformix.com

Chapter One

Introduction to OpenID

Identity management has emerged as one of the important fields in information technology, especially information security. There are many ways to create and manage digital identities. Three most commonly used environments are: Unix/Linux, Microsoft Windows, and Mainframe. In UNIX/Linux, identity management is done using LDAP, NIS, RADIUS, Kerberos, and a number of other mechanisms. In the Microsoft world, Active Directory (AD) is the most commonly used mechanism. There are multiple mechanisms for identity management in mainframe systems as well. In addition to operating systems based identity management solutions, a number of commercial and open source

products are also available for this purpose. The products provide facilities to manage user identities across multiple platforms and services such as single sign on (SSO), cross company authentication (CCA), etc.

Most of the above mentioned identity management methods are used in the in different ways, both for front-end user authentication and authorization as well as for back-end systems. Typically these systems work very well in a closed environment where all applications are managed by a single company. With the popularity of web-based systems for common users, common users have their accounts with several web sites and this is where the problem starts. Now a user has to create identities for a all of the web sites and remember username and passwords. Obviously, this has created a number of issues not only for users but from security perspective as well.

For a user having multiple accounts at different web sites, a common way of identity management from the “user perspective” is needed to overcome different problems. The most common problem faced by a user of the Internet is: how to effectively manage multiple identities at many web sites that a user has to interact with.

OpenID is an *open* protocol that enables a person to use a URL as an *identity* and use the same identity at multiple web sites that support OpenID. Web-enabled applications can use the *identity URL* for authentication, authorization and other purposes. It is a relatively new concept which puts the control of the identity into the hands of its owner, *the end user*. The owner of the identity can decide, and has control over, which information should be presented to an application or web site for authentication purpose. Among other things, OpenID enables owners to:

- Login to web-enabled applications and web sites without ever entering any username and password information.

- Enable web sites to request information from a user; and empowering the user to choose which information is to be sent to a web site during authentication/authorization process.
- Chose which sets of information (also known as profiles or cards) to be sent to different web sites, based upon need and risk level.
- Allows implementation of graded and risk-based authorization.
- Implement an alternate to single sign on (SSO) for multiple applications within an organization.
- Implement cross-company authentication (CCA) for affiliates and business partners.
- Integrate applications into the OpenID system using a simple and elegant mechanism.
- Lower cost of implementation and maintenance.

In this book, we shall look into many of these features in detail. The objective is to give the reader enough information by the end of this book to:

- Understand OpenID concepts and the other systems/protocols that work with OpenID.
- Understand the OpenID protocol in detail and the different types of messages that are used to convey information from one component to another component in the OpenID system.
- Implement OpenID enabled web applications.
- Run an OpenID server.
- Implement Cross Company Authentication (CCA) using OpenID

- OpenID interoperability with other systems like Microsoft CardSpace.

This chapter will help new users of OpenID understand how the system works at a very high level. Instead of starting with all of the theory behind the OpenID protocols and technical descriptions, I have chosen to provide a practical example. This example creates a new OpenID and then uses it on multiple web sites to help a new user understand how the process works. The next chapters will explain the technical details of the OpenID protocol and how to use it in real-life scenarios.

1.1 A Five Minute Tour of OpenID

OpenID is a system that enables you to use a URL as your identification and login to any OpenID-enabled web site. You don't need to create user IDs and passwords on individual web sites. The benefit: As a user of the OpenID system, you don't have to remember the usernames and passwords for individual web sites.

Before going into detail of OpenID protocol, I would like to give you short demo of how OpenID system works. In this demo, you will create an OpenID URL for yourself and then login to OpenID enabled web sites (also called *Consumer* or *Relying Party*) using this URL as your identity. A detailed discussion will follow in next chapters but for right now, just try to get a feel of how the system works on a user level.

For the purpose of this demonstration, we have selected Verisign *Personal Identity Provider* or PIP as the *Identity Provider* (The place where you create the Identity URL) and LiveJournal as the *Consumer* (the web site where you use your OpenID to login). Thanks to Verisign and LiveJournal for the permission to use screenshots from their web sites. It should be noted that the objective is not to promote a particular vendor. There are many open source implementations of OpenID protocol and you can choose any of those

implementations. Some examples in this book use these open source implementations and you will get references to these implementations at multiple places in this book. You can also see a list of these implementations on the <http://openid.net> web site.

The following sections provide a step-by-step approach for creating an ID and then using it. For some readers, this may seem to be very simplistic. However, this section will give a head-start to new comers in the OpenID world.

1.1.1 Creating an Account with Verisign PIP

We will use Verisign *Personal Identity Provider* or PIP and create an ID. PIP is a service that allows you to create an OpenID identity URL for free (at least for now). You will be pleasantly surprised how easy it is to create a new Identity URL¹ using PIP. To do so, go to the PIP web site <http://pip.verisignlabs.com> and you will see a web page similar to the one shown in Figure 1-1. Note that at the time of this writing, the web site is still in beta phase of development. So the web pages may have a different look and feel but the process of creating a new ID and managing it should not change drastically. Also, PIP is very intuitive web site and you will not find it difficult to explore different areas of the web site.

¹ Other web sites and identity providers may have slightly different processes for creating new IDs.

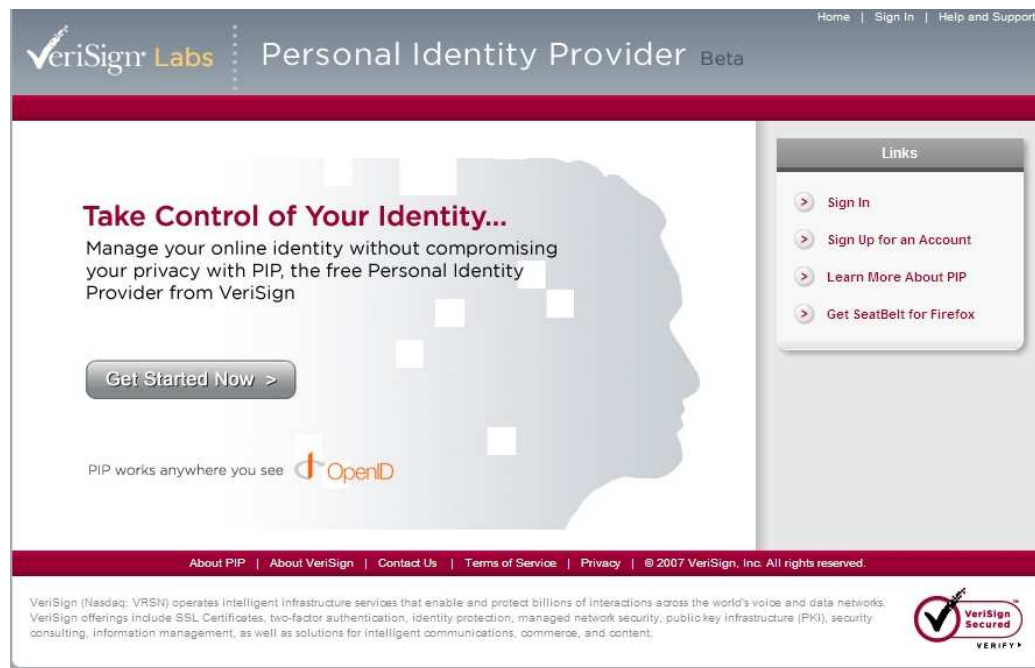



Figure 1-1: Verisign PIP web site home page

The process of creating a new identity is very simple and takes only few minutes. In Figure 1-1, click on the “Get Started Now” link and you will see the following screen in your browser where you will create your account.

Account Information	
* Full Name	<input type="text"/>
* Email	<input type="text"/>
* Username	<input type="text"/> The username that you enter will be used to create your OpenID.
* Password	<input type="password"/> Minimum of 6 characters and must contain a number and a letter.
* Confirm Password	<input type="password"/>

Personalize Your Account	
Personal Icon	<input type="text"/> <input type="button" value="Browse..."/> ? Large images may be slower to upload.

Verification Code	
* Verification Code	<input type="text"/> For security purposes, type the code seen in the image below. 

By creating my account I have read and agree to the [Terms of Service](#) and [Privacy Policy](#)

Figure 1-2: Creating a new account using Verisign PIP web site

In Figure 1-2, you need to enter required fields marked with asterisk. After entering these items, you enter the CAPTCHA² code in the image which is used to distinguish a real person from a machine. This simple process creates your

² CAPTCHA (jumbled characters in an image) is used to stop scripting attacks on a web site. CAPTCHA stands for “Completely Automated Public Turing test to tell Computers and Humans Apart”.

account. Your account provides you the URL that acts as your OpenID identity and you will use this URL to login to different web sites as explained next.

You will receive an email with a link to activate your account. This is necessary to verify your email address. Using the link in this email, you will activate your account. Once activated, you can login to the PIP web site and create your *profile*. Creating a profile is not necessary, although recommended. In some cases, the profile may be required but we will talk about that later. Note that this account is the only place where you will have a username and password to manage your OpenID identity. On other web sites, you will just use your OpenID URL without any username and password and without creating a unique user ID for each web site.

For the purpose of this demo, we have created a login name *rrpip* which you can use to login and set up profile. Once you login to the web site, you will see your Identity URL which is *rrpip.pip.verisignlabs.com*. Now you can make different settings to your account by clicking on “My Account” link. When you click on this link, you will see something like shown in Figure 1-3.

Once again and just to clarify, the login name for the PIP web site (identity provider) is *rrpip* whereas your OpenID identity URL is *rrpip.pip.verisignlabs.com*. When you login to PIP to manage your profile, you will use username *rrpip*. But when you go to other web sites that support OpenID, you will use *rrpip.pip.verisignlabs.com* to login.

After login to PIP, not only you can use your OpenID identity, but also create credentials using Verisign Identity Protection (VIP) as well as create Microsoft CardSpace card to associate with the OpenID URL. We are leaving discussion on these two features (CardSpace Cards and Verisign VIP) for a later stage and concentrate only on OpenID URL for the time being. The main thing to remember is that Verisign PIP now provided integration with Microsoft CardSpace and VIP.

Account Information	
Username	rrpip
* Email Address	<input type="text" value="rafeeq.rehman@gmail.com"/>
Mobile Phone	<input type="text"/> Example: 555-555-5555 ?
Personal Icon	<input type="text"/> <input type="button" value="Browse..."/>

Change Password	
* Old Password	<input type="text"/>
* New Password	<input type="text"/> Minimum of 6 characters and must contain a number and a letter.
* Re-enter Password	<input type="text"/>



Strong Authentication	
<div><p>VIP Credential ?</p><p>Protect access to your account by adding an extra layer of security using your VeriSign Identity Protection (VIP) credential.</p><p>Learn More > Get a credential ></p><input type="button" value="Add Credential"/></div>	<div><p>Information Card ?</p><p>Information Cards let you use virtual cards to identify yourself. Create an Information Card for accessing your VeriSign Personal Identity Provider account.</p><p><input type="button" value="Create Your Card"/></p></div>

Figure 1-3: PIP account settings

You will use the identity URL `rrpip.pip.verisignlabs.com` to login to other web sites. This URL is created by pre-pending your user ID (`rrpip`) to `pip.verisignlabs.com`. Other Identity Providers may have a different naming convention and OpenID does not put any restriction on the composition of URL. Note that your user ID (`rrpip`) and password is used to login to PIP web site only (called the *Identity Provider* or *IdP*). You will not need these to login

to other OpenID-enabled web sites (called *Consumers* or *Relying Parties*), where you will use only the URL for login purpose.

Now is the time to test the newly created identity URL. In the next section, we shall login to LiveJournal web site using OpenID identity URL.

1.1.2 Using PIP ID with LiveJournal Web Site

Now let us test the newly-created OpenID with LiveJournal.com. LiveJournal is one of many OpenID-enabled web sites that accept OpenID from other OpenID Identity Providers. First of all, go to www.livejournal.com and instead of creating a new account, find a link that shows “Login with OpenID” and click on it. You can also go directly to <http://www.livejournal.com/openid/> link to test your PIP OpenID. You will see something like Figure 1-4 when you go to this web site³.

LiveJOURNAL™

Create an Account Post to Journal Explore Gift Shop

Username: Password: [Create an Account](#) [Forgot your login?](#) [Login w/ OpenID](#)

☐ Remember Me [English • Español • Deutsch • Русский...](#)

OpenID

What is OpenID?

LiveJournal.com supports the [OpenID](#) distributed identity system, letting you bring your LiveJournal.com identity to other sites, and letting non-LiveJournal.com users bring their identity here. After all, not everybody uses the same websites, but you should still be able to play together.

Using your OpenID here.

If you're not a member of LiveJournal.com but want to leave authenticated comments and let people add you as their friend, trust your comments, etc., then you can login either in the form below, or from any comment entry form. Once you're logged in, you'll also be able to read friends-only posts that LiveJournal.com users have indicated you're allowed to read.

BETA:
Our OpenID consumer support is very new. That is, external users logging in with their identity here will find some rough edges while we work on smoothing it all out.
Our server support is relatively complete, though.

Your OpenID URL:
For example: [melody.someblog.com](#) (if your host supports OpenID)

³ Other OpenID enabled web sites will have similar text boxes for using with OpenID.

Figure 1-4: LiveJournal login page

Note that instead of entering a username and password, you will enter your OpenID URL as shown in this figure and then click on the Login button. The LiveJournal web site will figure out (using some background processing) that it needs your credentials from `pip.verisignlabs.com` and will redirect your browser to the PIP web site. That web site (`pip.verisignlabs.com`) will ask you to enter your username and password to ensure only you can access and authorize the use of your OpenID⁴. Once you do that, a new PIP screen will appear that allows you to select one of the following options:

- Opt for one time authentication to LiveJournal
- Allow authentication forever
- Specify an end- time when you will be asked to re-authenticate

These options, in addition to some other options are shown in Figure 1-5. You will select one of these options and then click on “Allow” button.



⁴ If you are already logged in to `pip.verisignlabs.com` in another browser window or a browser tab, you may not be asked to login and you will directly go to the next step.

Figure 1-5: Authorization and trust request

In Figure 1-5, just keep the default values and allow authorization for using your ID with LiveJournal web site only once (the first option under Authorization Request). For this purpose, just click on the “Allow” button towards the bottom of this screen. After you click the “Allow” button, some other background processing will occur behind the scene and you will be redirected back to LiveJournal web site where you will be logged in using your OpenID `rrpip.pip.verisignlabs.com` which is shown in the next figure⁵.



Figure 1-6: LiveJournal home page after login using OpenID

Congratulations; you are now logged into the LiveJournal web site. Note that the LiveJournal web page shows that you are logged in as `rrpip.pip.verisignlabs.com` (on top left corner of the Figure 1-6).

⁵ Note that if you select second or third option in Figure 1-5, you will not see this step altogether for future logins to LiveJournal.com web site and everything will happen behind the scenes for you.

Depending upon how the Consumer web site is configured and what parameters your Identity Provider (PIP in this case) supplies to the Consumer web site, the Consumer web site may ask you some additional parameters. However, you would have the control of which parameters you want to send to the Consumer web site.

1.1.3 Using the OpenID to Login to Multiple Sites

Now that you have your PIP OpenID identity URL created, you can use it to login to any other web site that supports OpenID (in addition to LiveJournal). Following is a very short list of some other web sites that support OpenID. Try your newly created ID to logon to these sites.

- <http://openid.net/wiki/>
- <http://www.lifewiki.net>
- <http://www.zoomr.com>

So now you know that with OpenID you can go to any web site that supports OpenID and that you don't need to create (and remember!) usernames and passwords on each and every web site. Among others, this is one of the major advantages of using OpenID.

There are other systems in the market that provide similar functionality and we will consider some of these systems at a later stage in this book. OpenID is a true open system which makes it more attractive for use. Additionally some of these other systems can inter-operate with OpenID, and we will look into interoperability issues later on.

1.2 So What Happened Behind the Scenes?

When you login to a web site using OpenID identity URL, many things happen behind the scenes. While we will get back to this in detail in later chapters, here is a short description. First of all, the Consumer web site (in our example the LiveJournal) communicates to the Identity Provider (in our example pip.verisignlabs.com) using various methods and exchanges messages in a defined format. These methods depend upon how intelligent the Consumer web site is. Once the Consumer web site has received a confirmation from the Identity Provider about the validity of the OpenID URL, it allows the user to login. If this is the first time a person is logging in to the Consumer web site, the web site may ask for some additional information to create a user profile and authorization parameters.

There are two basic methods or modes of communication between the Consumer and the Identity Provider depending upon how consumer is configured:

- **Dumb mode**, in which the Consumer does not maintain the *state* of the connection between the Consumer and the Identity Provider and has to go through more steps to authenticate a user. In this scenario, there are more HTTP request and response messages among the Consumer, the Identity Provider, and the User Agent (browser).
- **Smart mode**, in which the consumer maintains the state of connection which helps in reducing HTTP traffic for the authentication purposes. Sometimes this is also referred to as *store mode*. In this mode, the Consumer and Identity Provider maintain/store a shared key for encryption.

In both modes, the Identity Provider and the Consumer communicate using a shared secret to ensure confidentiality and integrity of the data being exchanged.

When you enter your OpenID URL to login to a web site, depending upon which mode is used, the web site contacts the Identity Provider via browser redirect, as well as direct communication between the Consumer web site and the Identity Provider. The Identity Provider then ensures that it is presenting the correct credential and it may ask you to authenticate to the Identity Provider. The OpenID does not care how the Identity Provider ensures that you are the same person who you claim to be. This means that different Identity Providers can use different mechanisms for this purpose. In the next chapters, we will discuss some of these mechanisms.

In the following section, you will see some additional features provided by PIP. Other Identity Providers may provide these or similar features in some other ways.

1.3 Entering User Setting

When you use PIP, you can create your profile settings using “My Information” link on the web site. Figure 1-7 shows the list of fields under your profile. You can click on the “Edit” link in front of each category to make changes in each data field.

Conformix **My Information**

The table below lists the information you entered when an OpenID Web site requested additional data from you. Use the action links to manage the information associated with your account.

[Add a Field](#) Show Tag: All

Field Name ▼	Value	Tags ?	Action
Full Name	Rafeeq Rehman		Edit
Nick Name	rr		Edit
Email Address	rr@conformix.com		Edit
Language			Edit
Gender	M		Edit
Date of Birth	1930-01-25		Edit
Country			Edit
Postal Code			Edit
Time Zone			Edit

[Return Home](#)

Links

- [My Account](#)
- [My Information](#)
- [My Trusted Sites](#)
- [My Activities](#)
- [Return Home](#)

Figure 1-7: PIP Profile Categories

The “My Activities” links shows log entries and tells you about the usage of your account and the web sites that you have logged in to while using your OpenID. This is shown in Figure 1-8.

Conformix **My Activities**

The table below lists the activity for your account. This includes when you have accessed your account, and sites where you have used your OpenID and Information Cards.

Displaying 0 - 25 of 74 [1](#) [2](#) [3](#) [Next](#) [Last](#)

Date ▲	Source ▼	Event ▼ ?	Result ▼	Action
2007-07-25	https://pip.verisignlabs.com	Login - Password Only	Success	Delete
2007-07-25	http://www.livejournal.com/	OpenID - Authentication Only	Authorized	Delete
2007-06-12	https://pip.verisignlabs.com	Login - Password Only	Success	Delete
2007-06-12	http://openidbook.com:80/knowledgebase	OpenID - Unknown	Authorized	Delete
2007-06-12	http://openidbook.com:80/knowledgebase	OpenID - Unknown	Authorized	Delete
2007-06-12	http://openidbook.com:80/knowledgebase	OpenID - Unknown	Authorized	Delete

Links

- [My Account](#)
- [My Information](#)
- [My Trusted Sites](#)
- [My Activities](#)
- [Return Home](#)

Figure 1-8: PIP activity log

This activity log is helpful in diagnosing some problems if you are not able to login to a particular web site for some reason. It also enables you to audit the user of your identity URL.

1.4 Other Identity Providers

Verisign PIP is one of many OpenID providers to choose from. You can also run your own Identity Provider web site and there are a number of libraries available for free. Please go to <http://openid.net/wiki/index.php/OpenIDServers> web site where you can find a list of OpenID providers and many of them provide their services for free.

Later in this book, you would also learn how to install and run your own Identity Provider in detail.

1.5 OpenID Identity URL Composition

There is no restriction on the type of URL that you can use as OpenID Identifier. Typically, it is a combination of a base URL and the username assigned to you by the Identity Provider. For example, if the identity provider is “idp.conformix.com” and you are assigned an ID “boota”, the Identity URL may be any one of the following (depending upon the choice of Identity Provider)

- <http://idp.conformix.com/boota>
- <http://boota.idp.conformix.com>
- <http://idp.conformix.com/?user=boota>

These are just few examples of how a URL can be formed. However, as mentioned earlier, there is no restriction on the composition of the Identity URL.

If you are running your own identity provider, the only thing to remember about the OpenID URL is that it should be consistent, user friendly, and easy to remember. Depending upon the composition of the identity URL, there may be some implications related to domain name server (DNS). If you are not managing your own DNS, then it is better to consult the DNS administrator before selecting the URL composition.

1.6 Chapter Summary

The purpose of the first chapter was to get a new user started with OpenID. The focus was to give the reader an introduction about how OpenID works at a very high level and without going into technical detail. In this chapter you created your own OpenID identifier and then used it with an OpenID enabled web site.

You have also learned some basic terminology about the OpenID system including the following:

- The OpenID identity URL is also called an *Identifier*.
- The OpenID services provider is also called *Identity Provider*.
- The web site that allows you to login using OpenID URL is also called a *Consumer* or *Relying Party*.
- Communication between Identity Provider and the Consumer occurs in one of the two modes: either *smart mode* (also known as *store mode*) or *dumb mode*.
- OpenID puts no restriction on the composition of identity URL.

In next chapters, many exciting things about OpenID will be presented.

1.7 References

For more information, you can refer to the following:

- Main OpenID web site at <http://openid.net>
- Verisign PIP <http://pip.verisignlabs.com>
- LiveJournal <http://livejournal.com>
- Web site for this Book at <http://www.openidbook.com>
- OpenID presentation at <http://openidbook.com/presentations/COLUG-OpenID.pdf>
- OpenID Blog at <http://openid.blogspot.com>

Chapter Two

Authentication and Authorization

Authentication is used to establish someone's identity and authorization is used to grant or deny access to someone after authentication. Traditionally, authentication and authorization are very system-centric. However, authentication and authorization are moving towards more user-centric and risk-based methods. The objectives of these methods are:

- To give the user control over which security tokens are sent to a web site for authentication
- Allow web sites to request tokens based upon risk level of information being accessed

- Allow identity providers to issue digital identities instead of username and password. Digital identities may contain different sets of tokens depending upon needs and circumstances
- Simplify the authentication process for end users

To meet these objectives, different types of systems are being developed. OpenID is one of the leading efforts in the open source arena.

This is a short chapter to provide a very basic discussion around authentication and authorization. It will help to clarify basic concepts about different methods used for authentication and authorization. It will also show issues facing the industry as well as problems with these methods. You will notice that none of the available methods provides sufficient security and a combination of multiple methods is needed for adequate security and reliability.

Once you have read this chapter, it will make more sense as to why OpenID exists in the first place and what problems it is going to solve.

2.1 What is An Identity?

This is not a new question: it has been there for centuries, way before computers were invented. Identity is something that is used to distinguish something or someone from others. If you think about the identity of human beings, is it a person's name? Probably not, because somewhere in this world, there may be many other people with the same name as yours. Is it someone's social security number, color of eyes, the language a person speak, the place you live, or something else?

In most of the cases, identity is a combination of factors mentioned above. In this chapter you will take a look into how identity can be established and

managed in a better way. Some additional discussions about identity will be presented later in this book.

2.2 Authentication and Authorization

The origin of authentication is from the Greek which has the notion of authentic or genuine. Authentication is the act or process of establishing authenticity of something or someone⁶. The dictionary meaning for authentication is to establish something (or someone) as a valid entity.

In terms of computer security, authentication is a process whereby an entity (a user, an application, a device, etc) establishes that it is what it claims to be. The most commonly used authentication method is the username and a password. There are many other authentication methods which will be introduced shortly.

Authorization is the process that typically comes after authentication and is used to grant or deny access to a computing resource⁷. So once a person or device has been authenticated, authorization enables access control to a resource for only those who have a legitimate need to gain access.

For example, an employee badge may be needed to establish identity and get inside a company building (authentication). However, only few employees may be allowed to go inside the check printing room (authorization).

⁶ <http://en.wikipedia.org/wiki/Authentication>

⁷ <http://en.wikipedia.org/wiki/Authorization>

2.3 Username and Password

Username and password are typically two strings composed of numbers, alphabets, or special characters. You should remember the following about usernames and passwords.

- Sometimes a username is also known as *User ID*, *Login ID*, or *Login Name* or simply an *ID*.
- A username is typically six to eight characters long but different companies will have their own standards for the length of username.
- A username is not usually someone's real name and it can be any string of character and/or numbers.
- A password should be a complex set of uppercase and lowercase characters, numbers, and special characters (like question mark).
- Company policies may be enforced to require complex passwords. For example, most of the computer systems allow a system administrator enforce a policy thereby not allowing dictionary words as passwords.

While the username stays the same, it is recommended to change the password on periodic basis. Some systems will force users to change their password after a certain period of time or after a certain number of uses.

Typically, you will have unique/distinct username and password for all systems and web sites you login to. OpenID solves this problem by creating a unique URL for you that you can use to login to different systems and web sites. So in a way, this is a replacement of traditional username and password and simplifies end-user's life to a large extent. However, OpenID does many other things as well which will be covered later in this book.

2.4 Authentication Methods

Authentication to computer systems is performed using many different methods. Depending upon a particular situation, sensitivity of the data or system, one method or a combination of methods may be used for authentication. This section introduces some of the commonly used authentication methods. Please note that there may be other authentication methods that are not covered here.

2.4.1 Password Authentication

Password based authentication is the most commonly used method in all types of systems and applications. Password authentication has been used to log on to operating systems, client-server applications, desktop applications, as well as web-based applications. For most people, when you mention authentication, username and password is the first thing that comes to mind.

Password-based authentication has served its purpose very well over time. However, a number of security issues are well established with password-based authentication. The most common issue is the sheer number of username/password combinations that a person has to remember. Most of the computer users have a number of accounts at their workplace, their home, their financial institutions, insurance providers, email accounts, etc. It has become almost impossible to remember that many passwords. As a result, typical users will:

- Write these passwords somewhere to avoid forgetting these passwords
- Use the same password for all accounts

Both situations are not good from security perspective. If you write down your passwords on a paper, it is inevitable that someone will see that paper you are hiding in your drawer or the sticky note on the back of your display monitor. On

the other hand, if you use the same password for all of your accounts, all of your accounts will be compromised if that password is disclosed. Since the same password is stored at many different places, the probability of its disclosure increases with the number of places it is stored at.

Also, over time, malicious attackers have found many ways to get others' passwords and password cracking tools have become very sophisticated. Phishing⁸ attacks have revealed weaknesses in using passwords as a single means of authentication.

2.4.2 PIN Authentication

PIN or Personal Identification Number is a string of numbers or a combination of numbers and letters. Typically a PIN is smaller than a password. PINs are used in many scenarios like ATM cards, authentication in telephone based systems, also known as Interactive Voice Response (IVR) and so on. PINs are also used in handheld devices where it is impractical to use long passwords.

PINs are considered to be a weak authentication mechanism and can be easily cracked by exhaustive search. Use of PIN is a reasonable authentication mechanism as long as it is used in combination with something that you have or something that you know, also known as two factor authentication.

2.4.3 One Time Password (OTP) Authentication

OTP is used only once to avoid problems with compromised passwords. There are a number of ways to generate one time passwords. One of these methods is the use of electronic tokens that you can carry with your key chain.

One time password generation tokens have been in use for quite some time. These are usually small devices that can be carried as part of your key ring.

⁸ See Wikipedia entry for phishing at <http://en.wikipedia.org/wiki/Phishing>

These devices generate random number strings, either on specified time intervals or when a user presses a button on the device itself. The user would then use the random number to authenticate to a system. Typically this random number is used in combination with a username and password.

There are some problems with these tokens as well. Different vendors have their proprietary systems that seldom work with each other. Other than interoperability issues, a person has to carry multiple tokens if this method is used for different systems. This makes the OTP mechanism difficult to use in current environment due to the fact that the typical user would need to carry multiple tokens (how many tokens are you willing to carry?). The OTP system has worked very well where a single token was used, usually to access company information or sensitive data. Also, the use of OTP does not eliminate the use of passwords which should be used in conjunction with OTP.

2.4.4 Smart Card Authentication

Smart cards are usually scanned or plugged into a system for authentication purposes. Smart cards are being used for physical security as well as IT security. Typically these cards are small enough (size of a credit card) so that people can carry them conveniently.

Many times smart cards are used in combination with a PIN. Smart cards have magnetic strips and/or embedded chips that are able to do a number of things. There are issues of interoperability with smart cards as well.

2.4.5 Biometric Authentication

In biometric authentication systems, finger prints, eye retina scans, or some other body identification mechanism is used. Biometric systems are not very scalable and also have issues with reliability in addition to being expensive to install and maintain.

Biometric methods, if not used properly, may also cause additional risks to personal privacy.

In a typical biometric system, a person will be asked to enter a pin in addition to biometric authentication. Some laptops also use biometric authentication without entering a pin.

2.4.6 Certificate Based Authentication

Digital certificates⁹ (also known as X.509 certificates) are also used as authentication mechanisms in many scenarios. SSH (Secure Shell) is one example where certificates are used frequently for the authentication purpose. Many companies also use certificates as second factor for remote access, such as VPN. Certificate-based authentication is considered very secure if the proper infrastructure for certificate management is available. However, the infrastructure may be cost prohibitive for many companies. Fortunately there are certificate provider companies that provide services to manage digital certificates.

Digital certificates can be revoked when an employee leaves a company or if a certificate is lost or stolen. Digital certificates also have an expiration mechanism. Any system that relies on digital certificates for authentication purposes is usually able to check certificate validity, expiration, or revocation. The certificate providers usually maintain a list, called *Certification Revocation List (CRL)*, that keeps track of all revoked certificates.

⁹ Digital certificates are used for many purposed in addition to authentication. Some of these include SSL (Secure Socket Layer) for secure web sites, sending and receiving secure email, digital signatures, etc. Please see additional information in Glossary section of this book.

2.4.7 USB Devices

Some USB devices carry authentication information, like an X.509 certificate. USB devices are easy to carry compared to smart cards and can also be used for carrying files with you. Another advantage is that the same USB device may be used to carry multiple credentials.

2.5 Weak and Strong Authentication

Different people define weak and strong passwords differently. Most commonly understood definition is that if only username and password combinations are used to authenticate to a system, it is called *weak authentication*. However, if you use a combination of different methods to authenticate to a system, it is called a *strong authentication* method. Use of just a username and password is weak because it can be compromised relatively easily compared to any strong authentication method.

For strong authentication, you use multiple methods or “*factors*”. For example, OTP token is a *factor* and password is another *factor*. Factors are divided into three broad categories.

- ***Something you know***; the example is a username and password. In other cases, many web sites may ask you security questions, like the name of your childhood pet.
- ***Something you have***; the example is a USB device, OTP token or a smart card.
- ***Something you are***; the example is finger print, eye retina or the way you walk (yes!).

Strong authentication uses a combination of factors from these categories.

2.5.1 Two –Factor Authentication

When you use two *factors* in combination to authenticate to a system, it is called a *two-factor authentication*. Each of the two factors should result in a success to authenticate a person. For example, if you are using username/password and OTP token, both the username/password and OTP token string should match for successful authentication. In this case even if someone steals your password, that person will not be able to login unless he/she gets hold of your token device as well.

Two-factor authentication is highly recommended for granting access to financial, medical, and customer data.

Also note that using the same factor twice does not make it two-factor authentication. As an example, using two passwords does not make it two factor authentication. For two-factor authentication, you have to use two of the three factors described earlier.

2.6 Single Sign-On (SSO) and Federated Identities

Sometimes people don't agree on one definition for single sign-on. One definition is a mechanism whereby you have to login once to get access to multiple resources. So for example, there may be many applications in a company's network. However, when you login to one application and then move to the next one, you don't have to authenticate again. Your credentials are handed over to all applications that take part in an SSO system and all of this happens in the background.

SSO solves a number of problems with usernames and passwords. It eliminates the need for multiple username and passwords which are difficult to remember. With SSO, you can implement better controls for password strength so that users must use difficult-to-guess passwords. However, SSO is like the *keys to*

kingdom, and if you lose your username and password, someone can get access to all applications and systems that you have access to.

It is more prudent to use SSO with multi-factor authentication so that the *keys to the kingdom* scenario becomes less likely and risk associated with SSO is reduced.

Typically SSO is used within a company and becomes useless when authentication is needed across different companies. This is because a user may have one username and password for accounts in one company and another set of username and password for another company. *Federated Identity* is another mechanism that allows authentication across companies but requires a heavy infrastructure as well as a trust factor. For example, unless your company trusts the ID administration of the other companies, it would be risky to rely on credentials provided by those companies.

2.7 Identity Management Dilemma

Over the last two decades, computer use has become ubiquitous in daily life. Things that were done on paper, or by postal service, are now done electronically with the help of computer. People manage their bank accounts, pay bills, do shopping, and many other things on daily basis with the help of computers. However, authentication is needed for all such activities. As a result, there is a real crisis and challenge for the computer industry: How to manage IDs?

The problem is how to minimize the number of username and passwords. Ideally only one username and password per person would provide ease of use. Single Sign On (SSO) mechanisms have been used successfully inside companies to reduce number of username and passwords. However, there are practical problems with using SSO across different companies. Federated

Identity is considered another solution for inter-company authentication but it needs a trust mechanism which is difficult to agree-upon as mentioned earlier.

So the question is what other mechanisms can be used to perform this necessary function better? Fortunately a number of solutions have been proposed to overcome the many problems that we are facing today. We will look at some of those in next sections.

2.8 Directory Services

Directory services are used to store user credentials. User credentials may be username/password, as well as many other types of information that may be helpful in authentication and authorization process. Directory services consist of two very basic components:

- A storage mechanism where all credentials are stored in a secure and structured manner
- A mechanism to enable access to the storage, usually in the form of a well-defined protocol

There are a number of directory services available today. The most widely used directory services are:

- The *Light Weight Directory Access Protocol* (LDAP) which has been standardized by the *Internet Engineering Task Force* (IETF). Both open source as well as commercial products are available that implement LDAP.
- Microsoft *Active Directory* (AD) is the primary directory on Microsoft Windows systems. AD can interoperate with LDAP, which is good news!

- The *Oracle Internet Directory* (OID) also implements LDAP and works with other LDAP systems.
- CICS is extensively used in the mainframe world.

The computer industry has become more mature in the directory services area and interoperability. Most of the directory services used today use LDAP in some fashion.

2.9 Risk Based Authentication and Authorization

This is a relatively new concept in the authentication and authorization arena. Basically, the concept is to have different levels of authentication/authorization which depend upon the risk level associated with the resources that are being accessed. For example, a weak authentication may be fine for checking the amount of payment due on your monthly cable television bill. However, a stronger authentication is needed if you want to make changes to your mailing address. So there may be multiple level of authorization needed for different types of services. Sometimes it is also being referred to as *graded authentication* and *graded authorization*.

2.10 New Authentication Mechanisms

As you have seen earlier in this chapter, although there are many authentication mechanisms, all of them have some drawbacks or problems. Many companies, as well as the open source community have been working very hard to come up with systems that are sophisticated, solve the current and future problems with identity management and make computer use easier and less complicated.

One new concept is to have a user-centric authentication and authorization framework whereby a user can control which information be sent to an entity (e.g. a web site) for authentication and authorization. At the same time,

depending upon risk level (see Risk Based Authentication above), a new idea is to allow the relying parties (e.g. web sites) to request information that is necessary to grant a specific level of access to resources. The following systems are being introduced to implement frameworks around these ideas.

2.10.1 OpenID

OpenID is a system that allows you to use a URL for authentication purposes. You can create your profile with an identity provider. This profile can be used with any OpenID enabled web site without creating a username and password for each web site. You have seen a demo of this mechanism in Chapter One demonstrating how OpenID works. The next chapters will provide detailed information about the OpenID protocol.

2.10.2 Microsoft CardSpace

Microsoft CardSpace was introduced with Windows Vista and in some literature it is also called Microsoft InfoCard. It is an effort to allow Windows applications to use digital identities provided by any identity provider in a consistent manner. Many concepts in OpenID and CardSpace are common and we shall explore the interoperability issues later on.

2.10.3 Bandit

The Bandit¹⁰ project provides many software components which enable authentication and authorization, role based access, compliance, and auditing.

2.10.4 Higgins

Higgins¹¹ is another effort to provide users control over their identity information that is shared with relying parties.

¹⁰ Bandit project home page: <http://www.bandit-project.org/>

2.10.5 LID

Light-Weight Identity or LID¹² is another system that uses URL based identities. These systems, like OpenID, implement user-centric identification.

2.10.6 Yadis

Yadis provides a mechanism to discover services available at a particular URL. Yadis is an XML-based simple protocol and enables a Consumer to discover authentication as well as other services provided by an identity URL. Since a URL can be used with multiple systems (e.g. LID and OpenID), Yadis is useful to identify the type of service and identification mechanism.

Typically an OpenID Consumer will use Yadis in the first step to get information about the identity URL. Yadis works over the HTTP protocol only according to its current specifications. When Yadis client sends a request, usually an XML document is returned in a format known as eXtensible Resource Descriptor or XRD. Following is an example of XRD document which will be used in Chapter 5 where protocols are described in more detail.

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS
  xmlns:xrds="xri://$xrds"
  xmlns:openid="http://openid.net/xmlns/1.0"
  xmlns="xri://$xrd*( $v*2.0) ">
  <XRD>
    <Service>
      <Type>http://openid.net/signon/1.1</Type>
      <Type>http://openid.net/sreg/1.0</Type>
      <URI>http://idp.conformix.com/index.php/serve</URI>
```

¹¹ Higgins project home page: <http://www.eclipse.org/higgins/>

¹² LID is available at <http://lid.netmesh.org>

```
<openid:Delegate>http://idp.conformix.com/?user=openidbook</openid:Dele  
gate>  
  </Service>  
</XRD>  
</xrd:XRDS>
```

Detail about different parts of the Yadis document (like the one shown above) will be discussed in Chapter 5. For the time being, it is enough to understand that the *<Service>* elements define a service. The *<Type>* element of the XML document shows a particular version of a service. There may be multiple *<Type>* elements in an XRD document and priorities can be set to use a particular version. The *<URI>* element shows the actual URL where the client should contact to get a service.

Figure 2-1 shows a Yadis black-box service which takes a URL as its input and gives an XML document as output. It is up to the Consumer to interpret the XML document and use the available services.

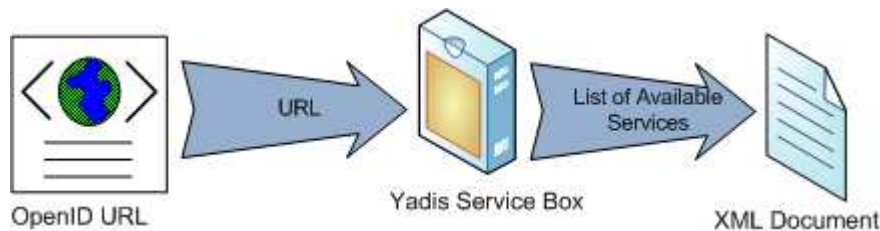


Figure 2-1: Yadis input and output

Yadis is very helpful in providing interoperability among different types of Identity services as well as setting up high availability environments using priority numbers.

More information about Yadis is available at <http://yadis.org>. OpenID also uses Yadis and this will become clear in next chapters. A typical OpenID client will make a Yadis call to Identity Provider to determine list of services and then make a second call for actual authentication request.

2.11 Chapter Summary

This chapter provided basic information about authentication and authorization and different methods currently in use for this purpose. It also introduced issues facing these methods.

The IT industry is going through a maturity and evolution process. The future of the authentication and authorization will be more *user-centric* and *risk-based* allowing users to use identities provided by different identity providers in a more consistent way. The users will also get control over what pieces of information are shared with the consumers of identities. In the next chapter, we shall focus on OpenID and how it solves many of the problems discussed above.

2.12 References

For more information, you can refer to the following:

- Main OpenID web site <http://openid.net>
- Verisign PIP <http://pip.verisignlabs.com>
- OpenID Book <http://www.openidbook.com>
- OpenID presentation at <http://openidbook.com/presentations/COLUG-OpenID.pdf>
- OpenID Blog at <http://openid.blogspot.com>

- LDAP information at <http://www.openldap.org>
- Bandit project at <http://www.bandit-project.org>
- Higgins project at <http://www.eclipse.org/higgins/>
- OpenSSO at <http://opensso.dev.java.net>
- Yadis at <http://yadis.org>
- LID at <http://lid.netmesh.org>

Chapter Three

OpenID Protocol and Messages

In Chapter One, you explored the use of OpenID from a very high level perspective without going into detail of how it works. In the Chapter Two, you looked at some of the authentication and authorization methods as well as the advantages and drawbacks of each of these. In this chapter you are going to look into some of the interworking of the OpenID protocol. Specifically, you shall start with OpenID concepts and terminology and then explain the message types and message flow to show what is happening in the background.

After going through this chapter, you will be able to understand the OpenID system in more detail and be ready for the next chapters where you are going to create your own OpenID server and Consumer applications.

3.1 OpenID Concepts and Terminology

OpenID is a system that allows a user to use a Uniform Resource Identifier or URI (like web site URLs) for authentication purposes. This URI is used as the username or identity for a person¹³ and is also called as the *Identifier*. In this book, we shall be using URI and URL interchangeably in different places.

This section provides introductory information about the OpenID systems and builds the foundation on which you will be able to understand the information provided in this and next chapters.

3.1.1 OpenID Definitions

To better understand the discussion in this chapter as well as in the following chapters, it is important to understand OpenID terminology. We would like to point out that in some cases, slightly different terms are used for the same thing in OpenID documentation. While defining terms, I shall point out these differences and commonalities. Also, OpenID specifications 1.1 and 2.0 have some slight changes in terminology and this will also be pointed out wherever applicable.

End User

End User is the real user or a real person who is using the OpenID system to login to different web sites using his/her credentials stored at the Identity Provider.

Consumer or Relying Party (RP)

Consumer is the actual web site where you login using OpenID. It is called Consumer because it *consumes* the OpenID credentials provided by the Identity

¹³ <http://openid.net/>

Provider. All web sites that support login using OpenID are Consumers. In OpenID specification 2.0 as well as in some other literature related to ID administration, a Consumer is also called a *Relying Party*.

Identifier

Identifier is the URL that identifies digital identity of End User.

Identity Provider or IdP (OP)

Identity Provider is the host where a user's credentials are stored. The OpenID URI points to the identity provider. During the authentication process, the Consumer will validate an ID by exchanging some messages with the Identity Provider. Sometimes it is also called as *OpenID Server*, *OpenID Provider* or simply *OP*.

User Agent

In simple words, *User Agent* is your browser. A user interacts with the User Agent directly.

There are some other terms as well but we shall introduce those ones as needed. The definitions mentioned here are sufficient to start the discussion about the OpenID protocol and communication flow.

3.1.2 Communication among OpenID System Components

There are three major components in any OpenID system: Consumer, Identity Provider, and User Agent. These components interact with each other during the authentication process. Roles for these components are shown next.

- The Consumer, which is the web site where you are trying to login, interacts with the Identity Provider and the User Agent (the web browser). An End User will try to login to the Consumer web sites using OpenID. During the authentication process, the Consumer will send

some messages to the Identity Provider directly as well as via User Agent with the help of HTTP redirect messages.

- The Identity Provider is the OpenID server that holds an End User's credentials. The Identity Provider will validate the ownership of an identity URL to the Consumer using two basic mechanisms which we shall discuss later in this chapter.
- An End User will interact with the Consumer and Identity Provider using the User Agent. The User Agent is your web browser.

During the authentication process, the browser acts as the middle man between the Identity Provider and the Consumer web site for some messages. Typically, a Consumer will interact with the web browser as well as the Identity Provider during authentication process. However in some cases, the Consumer may use cached keys to authenticate a user without any direct communication with the Identity Provider.

Figure 3-1 shows the communication path among these three entities. Here the assumption is that the Identifier URI also resides on the Identity Provider server.

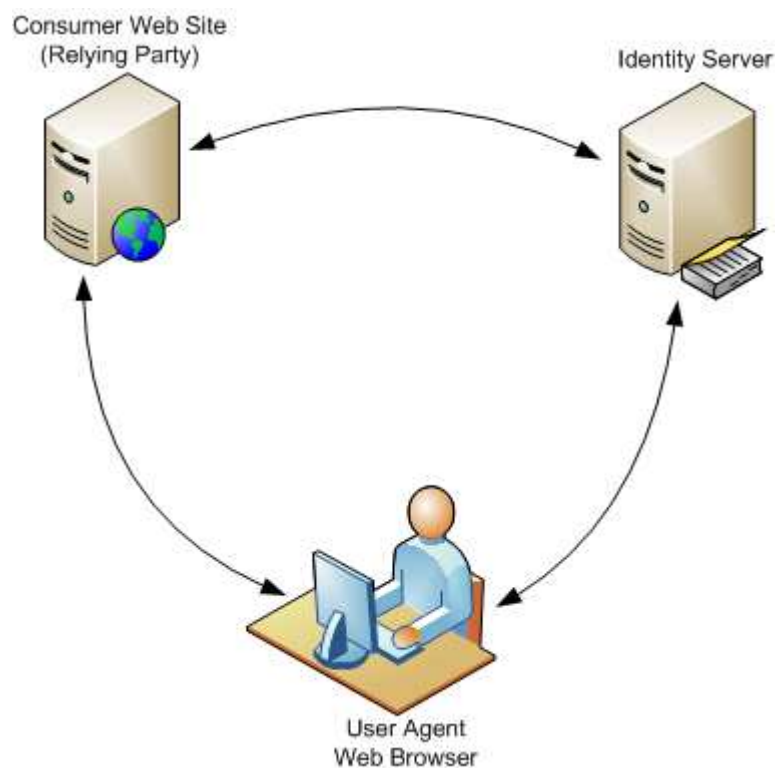


Figure 3-1: Communication among different components of the OpenID System with the URI identifier and the Identity Provider on the same machine.

Note that you can have your Identifier URI pointing to the Identity Provider machine or a different place. In fact, you can put your Identifier on any machine you like. Since the URI is your identity, it is important that as an End User you have control over that URI. If you own your URI, you can use a different server and your identity will remain the same.

The Consumer has a mechanism to identify the Identity Provider from the URI, which we shall discuss shortly. Figure 3-2 shows the communications path when your URI is located at a different place other than the Identity Provider.

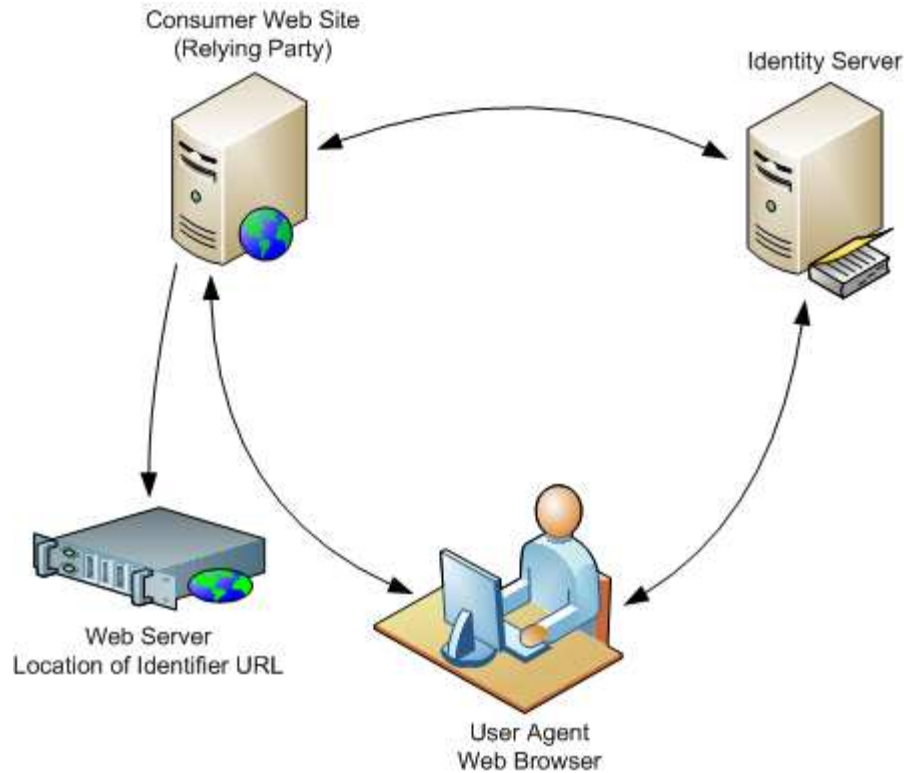


Figure 3-2: Communication among different components of the OpenID System with Identifier URI and Identity Provider are on different machines.

As shown in Figure 3-2, only the Consumer will fetch the URI in the OpenID system and no other machine needs to contact the URI.

Note that it is possible to have your browser machine also acting as the OpenID Identity Provider. In that scenario, there are only two physical machines taking part in the communication.

3.2 Direct and Indirect Communication

There are two basic communications methods among different entities in an OpenID system: Direct Communication and Indirect Communication.

In the Direct Communication mechanism, two entities directly talk to each other using the HTTP protocol. The HTTP POST method is used for direct communication.

With the Indirect Communications, two entities talk to each other via a third entity. This third entity is typically the web browser. Indirect communication may happen via HTTP Redirect or via HTML Form redirection.

3.3 OpenID Modes of Operation

OpenID has two major modes of operation: the Dumb mode and the Smart mode. These modes are based upon how intelligent the Consumer is. In Dumb mode, as the name implies, the consumer is not *that* smart and has to perform few additional steps every time a user logs in. In Smart mode, the Consumer keeps state information and caches shared keys for future use. In this section, we shall share more detailed information about how these two modes work.

3.3.1 Dumb Mode Communications Flow

In Dumb mode, the Consumer (relying party) does not maintain the state of the connection, so any information that was used in a previous login, can't be used again. Every time an End User logs in to a Dumb Consumer web site, the same process is repeated. In a nutshell, the process goes through the following steps. We shall be referring to our example in Chapter One so that you can correlate your experience with each process step-by-step. Note that in Chapter One, we had:

- The Identity Provider as `pip.verisignlabs.com`

- The Consumer as livejournal.com

Following is the step-by-step authentication and login process where communication between the Consumer, the User Agent (web browser), and the Identity Provider takes place. These steps are mapped graphically using Figure 3-3.

1. You visit the Consumer web site where you want to login. In Chapter One, the web site was livejournal.com.
2. The web site presents a web page where you enter your identity URL. Typically, you will enter your Identity URL and click on Login button. Different web sites may have different types of web pages but the process will remain the same. In Chapter One, this is the place where you entered your URL “rrpip.pip.verisignlabs.com”. Note that you don’t need to add “HTTP://” in the beginning or any slash character at the end of the URL. OpenID specification requires that all Consumers should be intelligent enough to understand a URL in different formats.
3. The Consumer web site (livejournal.com) will clean up the Identifier URL and fetch it from its current location. This location may be the same as the Identity Provider or may be a different host. In Chapter One, the URL location and the Identity Provider are at the same place (pip.verisignlabs.com). Note that in OpenID specification version 2, another XML protocol, known as Yadis, can also be used at this stage for service discovery. Yadis is discussed in more detail later in this chapter and the next one.
4. After fetching the page, the Consumer (livejournal.com) will then parse it and determine the location of Identity Provider (OpenID Server). This information is embedded inside the HTML web page and we shall discuss the page itself shortly. This parsing process is also called *discovery*. After parsing, the Consumer (livejournal.com) will then

redirect the web browser to the Identity Provider to obtain the assertion information. This happens using the HTTP GET method. Optionally, the consumer may establish a connection with the identity provider at this point and exchange a shared secret for further communication. This is shown with a dotted line in Figure 3-4 and marked as step 4a.

5. If the end user is not already logged into to Identity Provider, the Identity Provider may ask the End User to login. This is what happened in our example in Chapter One. However, it should be noted that this part is outside the OpenID specifications and it is left to the Identity Provider to decide how to authenticate the End User. In some circumstances, if you have already logged into the Identity Provider web site, this part may be skipped altogether.
6. The Identity Provider (pip.verisignlabs.com) will return the assertion information with its signature to the Consumer (livejournal.com) via browser redirect. This assertion will represent either an authentication success or failure. The HTTP GET method is used in this step as well. Note that this is the Indirect Communication between Identity Provider and the Consumer.
7. During successful assertion, the Consumer (livejournal.com) will establish a direct connection with the Identity Provider (pip.verisignlabs.com), preferably over a secure SSL session. It will request the authentication information directly from the Identity Provider and compare it with the assertion information it received via User Agent (web browser). This is to double check the validity of the assertion in case a User Agent (or a malicious attacker) is trying to cheat.
8. If there is a match in the previous step, the End User will login to the web site. Otherwise the login will fail.

A detailed step-by-step process is shown in Figure 3-3 where you see all the communication steps. Detailed protocol level information about OpenID messages is presented in Chapter 5 where you will see actual HTTP messages. By then, you will have installed your own Identity Provider and Consumer and then you may want to use a sniffer to look at the HTTP traffic and analyze the protocol further.

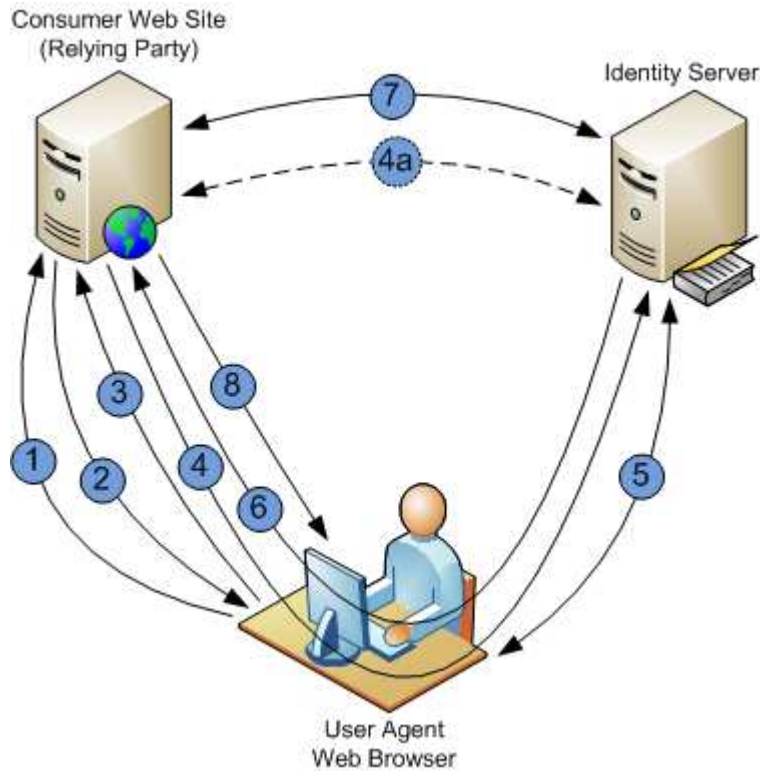


Figure 3-3: Dumb mode communications

3.3.2 Smart Mode

The smart mode authentication is similar to the dumb mode authentication with the exception of step number 7 in Figure 3-3. Now the Consumer already

has the shared secret (from step number 4a) and it can decrypt and verify the assertion from step 6 and determine if the Identity Provider really signed it. Note that step 4a will happen only once in a while when the Consumer needs to refresh the cached secret or get it the first time. This is shown in Figure 3-4. The step-by-step process is as follows:

1. The End User visits the Consumer web site.
2. The web site presents a web page where you enter your identity URL and click on the Submit or Login button.
3. The Consumer web site will clean up the Identifier URL and fetch it from its current location.
4. After fetching the page, the Consumer parses it and determines the location of Identity Provider (OpenID Server). After parsing, the Consumer will redirect the web browser to the Identity Provider to get the assertion information. Optionally, the Consumer may send an association request with the Identity Provider and exchange a shared key as shown in step 4a in Figure 3-4.
5. If the End User is not already logged into to Identity Provider, the Identity Provider may ask the End User to login.
6. The Identity Provider will return the assertion information with its signature to the Consumer via browser redirect. This assertion will represent either as authentication success or failure.
7. After a successful assertion, the Consumer verifies the assertion using the cached shared key. If there is a match in the previous step, the End User will login to the web site. Otherwise login will fail.

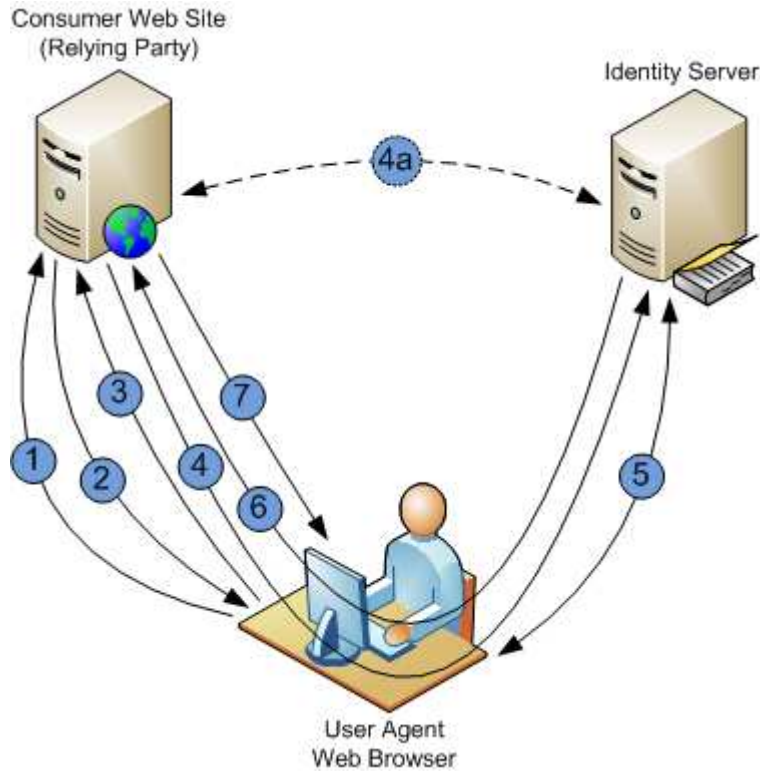


Figure 3.4: Smart mode communication flow, during the first time login.

In many cases when you have already established your credentials with the Identity Provider, it will not prompt you again for login. There are many techniques that can be used by the Identity Provider to achieve this goal, including active browser sessions and cookies. In such a scenario, and assuming that the Consumer has already cached the Identity Provider shared secret, the communication will become very simple and the End User will login to the Consumer web site without any interactive session with the Identity Provider. This is shown in Figure 3-5. In this case, the steps will be as follows. Note that the End User will get immediate access to the web site after entering the Identifier URL.

1. You visit the Consumer web site.
2. The web site presents a web page where you enter your identity URL.
3. The Consumer web site will clean up the Identifier URL and fetch it from its current location.
4. After fetching the page, the Consumer parses it and determines the location of the Identity Provider (OpenID Server). After parsing, the Consumer will then redirect the web browser to the Identity Provider to obtain the assertion information.
5. The Identity Provider will return the assertion information with its signature to the Consumer via a browser redirect method. This assertion will be represented as either an authentication success or failure.
6. After a successful assertion, the Consumer verifies it using the cached shared key. If there is a match in the previous step, the End User will login to the web site. Otherwise login will fail.

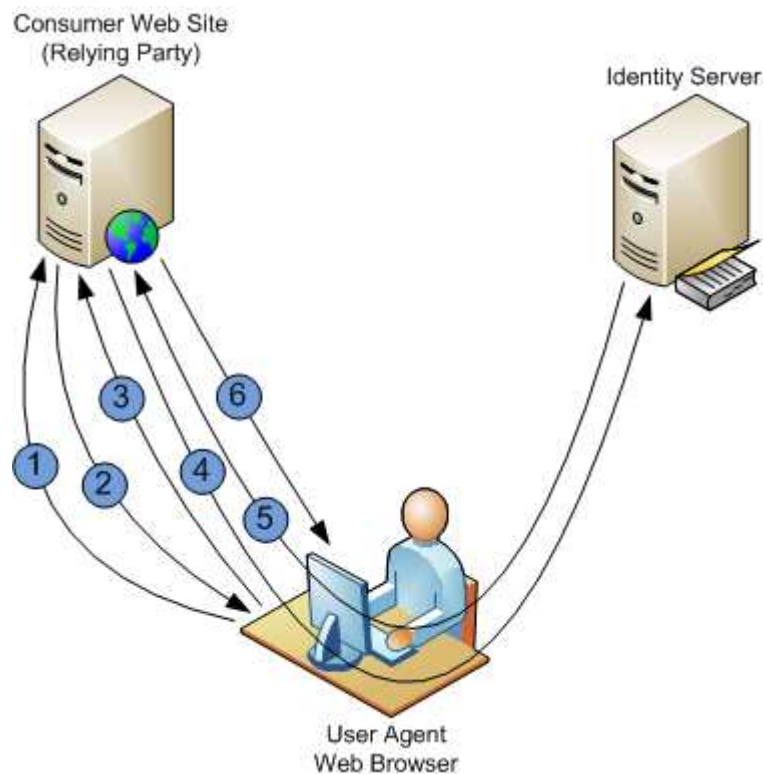


Figure 3.5: Smart mode communication flow and the subsequent login where the Consumer and Identity Provider have already established a shared secret.

Note that in Figure 3-5, the communication with the Identity Provider is transparent to the user and operations happen behind the scenes via browser redirects.

3.3.3 Using Ajax with Dumb and Smart Modes

It is appropriate for a Consumer to use Ajax¹⁴ on the login page to make the page redirection mechanism more transparent to the user. The Ajax page will give an impression to the End User that authentication is complete without leaving the web page. Although this is not mandatory, it would definitely improve user experience.

3.4 OpenID Identity URL Page

We have talked a lot about OpenID Identity URL. Now let us have a short discussion about what information is present in an OpenID URL page.

First of all, you should have a clear understanding that you can put your OpenID URL on *any* web server. You are not *required* to put it on the Identity Provider server. You just need to create an HTML document with the specific information in it and then put it on a web server. The URL to this document will be your Identifier URL. You can also use an XRI¹⁵ (eXtensible Resource Identifier) based URI but we will explain this at a later stage to keep things simple for the time being.

The HTML document will have information about the OpenID server in its HEAD section. As an example, if you are using pip.verisignlabs.com as your Identity Provider, you will have something like the following in the HEAD section of your document.

```
<link rel="openid.server" href="https://pip.verisignlabs.com"/>
```

¹⁴ For more information on Ajax, please visit <http://en.wikipedia.org/wiki/AJAX>

¹⁵ For more information about XRI, refer to <http://www.oasis-open.org/committees/xri/faq.php>

Note that in the above line, “openid.server” is a keyword and it should appear exactly as shown above. The “href” part may vary depending upon which Identity Provider you are using. Also note the URL to the Identity Provider may start with HTTP or HTTPS depending upon Identity Provider configuration.

As you will see in Chapter 5 (where you run your own OpenID Server), a typical HTML document at a URL may work like the following:

```
<html>
  <head>
    <link rel="openid.server"
href="http://idp.conformix.com/index.php/serve">
    <link rel="openid.delegate"
href="http://idp.conformix.com/?user=openidbook">
  </head>
  <body>
    <h3>OpenID Identity Page</h3>

    <p>
      This is the identity page for the user <strong>openidbook</strong>.
    </p>
  </body>
</html>
```

In this case, the following line shows the location of OpenID server that the Consumer will contact for authentication purpose.

```
<link rel="openid.server"
href="http://idp.conformix.com/index.php/serve">
```

The following line shows the OpenID URI. Note that the *delegate* keyword is used when the URI may be on a different machine than the server itself.

```
<link rel="openid.delegate"
href="http://idp.conformix.com/?user=openidbook">
```

In this case the OpenID URI is “http://idp.conformix.com/?user=openidbook”

If the Consumer and Server support OpenID specifications version 2, an XRD document can also be used instead of HTML document. A typical XRD document may be as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS
  xmlns:xrds="xri://$xrds"
  xmlns:openid="http://openid.net/xmlns/1.0"
  xmlns="xri://$xrd*($v*2.0)">
  <XRD>
    <Service>
      <Type>http://openid.net/signon/1.1</Type>
      <Type>http://openid.net/sreg/1.0</Type>
      <URI>http://idp.conformix.com/index.php/serve</URI>

    <openid:Delegate>http://idp.conformix.com/?user=openidbook</openid:Dele
gate>
    </Service>
  </XRD>
</xrds:XRDS>
```

Detailed information about XRD documents will be presented in Chapter 5.

3.5 OpenID Specification Versions

OpenID specifications have two commonly used versions at the time of writing this book. These are specification version 1.1 and version 2.0. There are some differences between these two versions. Version 2.0 can be used in backward compatible fashion, also referred to as “1.1 Compatibility Mode”. In this book, we shall elaborate on the differences where appropriate. The original version of OpenID specification was 1.0 which may still be in use in some implementations.

The main differences between OpenID specifications 1.1 and 2.0 are:

- Use of Yadis for service discovery
- Support for XRD documents
- Security improvements by adding nonce which protects against relay attacks
- Support of DH-SHA256 key exchange and HMAC-SHA256 for stronger encryption

There are some other minor changes and they will be noted on as-needed basis. The version 2 can be backward compatible with version 1.0 and 1.1.

3.6 OpenID Messages

OpenID components exchange different messages during the authentication process. These messages have very well defined formats and the Consumers and Identity Providers have to adhere to these formats for successful communication to occur.

Each message has a request and response combination and the request and response may have different sets of parameters. When we explore these messages, we shall explore the request and response parameters separately.

Depending upon the type of message, the Consumer and Identity Provider can use direct or indirect communication methods as explained earlier in this chapter. Note that the HTTP POST method is used for direct communication whereas HTTP GET method is used for indirect communication.

There are four basic messages used in the OpenID system. These are as follows:

1. The *associate* message
2. The *check_immediate* message

3. The *check_setup* message
4. The *check_authentication* message

In the following sections, you will find detailed information about these messages. This information is more descriptive to keep things simple. Detailed protocol level sniffer output will be discussed in Chapter 5.

3.6.1 The *associate* Request Message

The *associate* message is sent by the Consumer (Relying Party) to the Identity Provider. The primary purpose of the association message is to establish a shared secret between the Consumer and the Identity Provider. Note that the Consumer web site can send this message to the Identity Provider at any time it is required.

Since this is a direct communication between the Consumer and the Identity Provider, the HTTP POST method is used for the *associate* message. While initiating the *associate* request, the Consumer web site will send a number of parameters with the request. The following is a list of parameters that can be sent with the *associate* request.

- openid.ns
- openid.mode
- openid.assoc_type
- openid.session_type
- openid.dh_modulus
- openid.dh_gen
- openid.dh_consumer_public

Let us have a short discussion about all of these parameters and get an understanding of where they are used.

The openid.ns Parameter

This is an optional parameter. The primary reason of this parameter is to define the OpenID specification version number being used for a particular message. The value of openid.ns will be “http://specs.openid.net/auth/2.0” if you are using OpenID specification 2.0. If this parameter is not present or if the value of this parameter is set to “http://openid.net/signon/1.1” or “http://openid.net/signon/1.0”, the Identity Provider will fall back to older versions of OpenID (OpenID Authentication Compatibility Mode).

Note that the openid.ns parameter was not present in OpenID specifications version 1.1 and it is included in version 2.0 and above.

In the future when new versions of OpenID specifications become available, the value of this parameter may change accordingly.

The openid.mode Parameter

This parameter shows the type of message and it is used to distinguish which message is being sent or received. This parameter is present in all OpenID messages.

The value of this parameter is “associate” for the *associate* request message.

The openid.assoc_type Parameter

This parameter is used to convey the algorithm used for signing the message. OpenID uses the following algorithms for signing:

- If the value of this parameter is “HMAC-SHA1”, the signing mechanism is the HMAC-SHA1 as defined in RFC-2104.

- If the value of this parameter is “HMAC-SHA256”, the signing is performed using algorithm HMAC-SHA256.

Currently these are the only two values supported for OpenID.

The openid.session_type Parameter

The openid.session_type parameter is used to show the type of encryption used in the message to encrypt MAC (Message Authentication Code) key. A MAC is short code that is calculated using a secret key and input message. A MAC algorithm takes the private key and the message as input and provides the MAC code as output. Hashed Message Authentication Code or HMAC is one type of MAC.

In OpenID messages, different types of encryption are used to encrypt MAC as listed below:

- A value of this parameter is “DH-SHA1” for Diffie-Hellman SHA1
- A value of this parameter is “DH-SHA256” for Diffie-Hellman SHA256
- A value of “no-encryption” for sending MAC in the clear, without any encryption. It is recommended to always use encryption. However, if an SSL connection is used between the Identity Provider (IdP or OP) and the Consumer web site, you can elect not to encrypt the MAC because the transport layer is providing the encryption.

Note that if the Identity Provider can’t support the openid.assoc_type or openid.session_type parameters present in the request, it will reply with an unsuccessful association response.

The openid.dh_modulus Parameter

The openid.dh_gen Parameter

The openid.dh_consumer_public Parameter

If you chose DH-SHA1 or DH-SHA256 for openid.session_type, the above three parameters are part of the association message. OpenID specifications describe how to generate these parameters. Detailed information about the Diffie-Hellman algorithm is present in RFC-2631 (Diffie-Hellman Key Agreement Method) which is available at <http://www.ietf.org/rfc/rfc2631.txt>.

A typical association request is as follows which will be discussed in more detail in Chapter 5.

```
openid.mode=associate&openid.assoc_type=HMAC-  
SHA1&openid.session_type=DH-  
SHA1&openid.dh_consumer_public=KC6IpA00A6SlCikafFSlrTGq19H8+de6GFi5YLKz  
4pyDxUMS5Z8pMOm/PtrlgFmCcgAXjFbuxS73ZutDTFJYpADoIntFVrah9eaezMcw6SDR24c  
nFjNc14xq0zGt3QcRLXaNTRVKfMW8evDAmLCrvEhU5c7B3eqmk+bMMrbQpcE=&openid.dh  
_modulus=ANz5OguIOXLsDhmYmsWizjEOHTdxfo2Vcbt2I3MYZuYe9louJ4mLBX+YkcLiem  
OcPym2CBRYHNOyyjmG0mg3BVd9RcLn5S3IHHoXGHblzqdLFEi/368Ygo79JRnxTkXjgmY0r  
xlJ5bU1zIKaSDuKdiI+XUkKJX8Fvf8W8vsixYOr&openid.dh_gen=Ag==
```

3.6.2 The associate Response Message

The *associate* response message is sent from the Identity Provider to the relying party. This is an HTTP 200 message as defined in RFC 2616 (<http://www.ietf.org/rfc/rfc2616.txt>). The message may show a successful or failed association. In case of a successful association, the message will contain a message handle and life of that handle in number of seconds. In case of a failed association, an error is returned. Below is a list of parameters that are part of the association response message. Note that the parameters that are already explained will only be listed without any further explanation.

For successful association, the following parameters will be part of the response message.

- **openid.ns**, as discussed earlier.
- **openid.assoc_handle** which is a printable ASCII string with maximum length of 255 characters. Note that printable ASCII characters have codes from 33 to 126. The association handle can be used in subsequent messages. Typically an association will last for some time. The association handle is used to determine which key should be re-used for encryption/decryption.
- **openid.session_type** parameter is the same as in the request if the association is successful. If the association fails for any reason, the value of this parameter will be “unsuccessful response”. There may be different reasons for an unsuccessful association. For example, if the Consumer is requesting to use SHA256 and the Identity Provider can only support SHA1, the association will fail.
- **openid.assoc_type** parameter is the same as in the request if the association is successful. If the association fails for any reason, the value of this parameter will be “unsuccessful response”.
- **openid.expires_in** is a time in seconds after which the association expires and the relying party should request a new association.
- **openid.mac_key** parameter is used only if the value of “openid.session_type” was “no-encryption”. The value of this parameter is base-64 encoded MAC key.
- **openid.server_public** is the Identity Provider’s public key. This parameter is used if the Diffie-Hellman algorithm was used in the request.
- **openid.enc_mac_key** is the encrypted MAC key. This parameter is used if the Diffie-Hellman algorithm was used in the request.

A typical response for association request is as follows and will be discussed in more detail in Chapter 5.

```
assoc_handle:{HMAC-SHA1}{4607344a}{oDFF0g==}  
assoc_type:HMAC-SHA1  
dh_server_public:AIPkx6xJ3b1Wnr1o1WL7suoZnABDc+lJRR9DeNIBolGXQX3W2e+4ud  
Y2p+dUcF5jKE6uoZuXLVPbimHbndBOYhUDUfkKaAjQtVvONerAjd5RHyt2i2AoYrkjD26tr  
aC4jzg7NukZlmrRjfPRg4q3gwW+EZEXvz+ba9JnQfsXx+iH  
enc_mac_key:UtQHBswQimAZAp4s/9sfSQSpuq0=  
expires_in:1209600  
session_type:DH-SHA1
```

In case of unsuccessful association attempt, an “error” and “error_code” parameters are also returned. The unsuccessful response may have other optional parameters as well, but the error and error_code parameters are important.

3.6.3 The checkid_setup and checkid_immediate Request Messages

The checkid_immediate and checkid_setup messages are used to get assertion information from the OpenID server. These messages are initiated by the Consumer web site. Indirect communication is used for these messages, which means the Consumer will use HTTP GET method (instead of HTTP POST) for sending and receiving these messages. It also means that these messages will pass through the user agent (the web browser).

The checkid_immediate message is typically used by the Consumers that support Ajax whereas checkid_setup is typically used by non-Ajax Consumers. Otherwise these messages are similar with some minor difference and use cases.

The following are the parameters used with `checkid_setup` request message. The parameters which are already discussed in the previous discussion are not explained again.

- **openid.ns**
- **openid.mode** which will have a value “`checkid_setup`”.
- **openid.claimed_id**, is an optional parameter showing claimed identifier. *Claimed Identifier* is a URL that the End User is claiming to own but it is not yet verified.
- **openid.identity** is another optional parameter.
- **openid.assoc_handle** parameter is also optional and if present, it shows an *association* that has already been established between OpenID server and the Consumer. You have seen this in the *associate* request already.
- **openid.return_to** parameter is used to inform the OpenID server the location of the URL where it should redirect the browser after processing the request. The OpenID server will use this URL to send the response back to the Consumer.
- **openid.realm** parameter is another optional parameter. Realm is a URL that OpenID server uses to identify a Consumer in a unique way. Realm may contain wildcards like “*”. An example of realm would be `http://*.conformix.com`.

Note that in OpenID specifications 1.1, `openid.claimed_id` and `openid.realm` parameters are not present. Instead, another parameter `openid.trust_root` is present which is optional and shows the actual URL for the Consumer web site.

You should also note that the request message reaches the OpenID server in two steps. In the first step, HTTP 302 redirect method is used from the Consumer web site to the web browser. Then in the next step, the browser sends HTTP GET request to the OpenID server. This is shown in Figure 3-6.



Figure 3-6: The flow for the checkid_setup request message.

The following is a sample of checkid_setup message captured from a real communication between OpenID server and a Consumer.

```
GET /index.php/serve?openid.assoc_handle={HMAC-
SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=checkid_setup&openid.return_to=http://consumer
.conformix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sreg.optional=e
mail&openid.trust_root=http://consumer.conformix.com:80/ HTTP/1.1
```

More information about this message will be in Chapter 5.

3.6.4 The checkid_setup and checkid_immediate Response Messages

Once the OpenID server receives the checkid_setup message, it does some processing and sends a response back to the Consumer via web browser.

Optionally, the OpenID server may ask the End User to authenticate to the OpenID server to ensure only the owner of the Identifier URL can authorize access to the Identifier. This may include presenting a login page to the End User. As mentioned earlier, implementation of that part is left to the OpenID server and is not included in the OpenID specifications.

In the response, the OpenID server will send multiple parameters back to the Consumer web site. Some of these parameters are optional and other are required. Discussion on those parameters which are already discussed in previous section is omitted.

- **openid.ns**
- **openid.mode** which will have a value “id_res”. In case an association fails, the value of this parameter will be:
 - “setup_needed” in case the request was checkid_immediate.
 - “cancel” if the request was checkid_setup.
- **openid.op_endpoint** shows the OpenID Server URL
- **openid.claimed_id**
- **openid.identity**
- **openid.assoc_handle** is the handle that was used to sign this message. The Consumer will use this handle for verification purpose. This handle may be the same as what was sent by the Consumer with the request message (in case an association already exists). It may be different than the original handle that the consumer sent with the request if the OpenID server does not recognize the original handle. In that case, the server will keep a record of the new handle so that the

Consumer can use it for verification purpose (using `check_authentication` message discussed shortly).

- **openid.return_to** parameter is the same copy of the URL that the Consumer sent with the request message.
- **openid.response_nonce** parameter is used to avoid relay attacks and is unique for each message. The maximum length of nonce is 255 characters and it consists of server time stamp and additional ASCII characters to make it unique. Time is taken in UTC¹⁶ format. Note that if the Consumer can reject as association if the timestamp is too far from the current time.
- **openid.invalidate_handle** is used to show if the handle attached with the request was valid or not. If the handle was valid, this parameter is optional. Otherwise, the invalid handle should be attached with this parameter. This will help the Consumer remove invalid handles from its records.
- **openid.signed** contains a list of parameters that are signed. The list is comma separated.
- **openid.sig** contains the signature which is base-64 encoded.

As with the request message, the response message reaches the Consumer web site in two steps. In the first step, the OpenID Server uses the HTTP 302 redirect method from the OpenID Server to the web browser. Then in the next step, the browser sends the HTTP GET request to the Consumer web site. This is shown in Figure 3-7.

¹⁶ UTC time format information may be found in RFC 3339 at <http://www.ietf.org/rfc/rfc3339.txt>



Figure 3-7: The flow for the checkid_setup response message.

Following is a sample of checkid_response message from a real communication between an OpenID server and a Consumer web site.

```
GET /finish_auth.php?nonce=nC5sKquX&openid.assoc_handle={HMAC-
SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=id_res&openid.return_to=http://consumer.confor
mix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sig=nXWc+07GLaSf+RghmG
ubGPPglZc=&openid.signed=mode,identity,return_to,sreg.email&openid.sreg
.email=rr@conformix.com HTTP/1.1
```

Note that in OpenID specifications 1.1, if an assertion fails, a different parameter is sent back in the response message. This parameter is “openid.user_setup_url” which has a URL as its value. This URL can be used to redirect the web browser for further steps.

3.6.1 The check_authentication Request Message

The check_authentication request and response messages are a necessary tool to verify the assertion received from the User Agent (web browser). This is to ensure that an attacker (malicious person) is not sending crafted assertion messages on behalf of the OpenID Server. You should note the following about the check_authentication messages.

1. This message is not sent if an association already exists between the Consumer web site and OpenID Server. The association is initially established using the *associate* message as already explained.
2. If an existing association is used, the Consumer will send the “openid.assoc_handle” parameter in the request and the OpenID Server will send back the same handle in the response. If that happens, the Consumer web site would know that the OpenID Server has agreed to use the existing association handle.
3. If the OpenID Server does not agree to use the association handle provided by the Consumer web site, the response will include the “openid.invalidate_handle” parameter in the response message and a different “openid.assoc_handle”. Also the Consumer will use the *check_authentication* message to validate the assertion.
4. When dumb mode is used, this response message will always be used because the Consumer is stateless and has no record of any previous association handle.

Note that this message is a direct communication between the Consumer and the OpenID Server and the HTTP POST method is used for this communication. The request has “openid.mode” parameter with a value “check_authentication” and all other parameters that were part of the assertion message. A typical message looks like the following:

```
openid.assoc_handle={HMAC-SHA1}{460730e1}{zrlgKg==}&openid.identity=http://idp.conformix.com/?user=openidbook&openid.invalidate_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}&openid.mode=check_authentication&openid.return_to=http://consumer.conformix.com:80/finish_auth.php?nonce=mAotRbGM&openid.sig=4hwwyWbPtSAmP2dYxEC+dq6050s=&openid.signed=mode,identity,return_to,sreg.email&openid.sreg.email=rr@conformix.com
```

In Chapter 5, you will also see more detail about this message.

3.6.2 The `check_authentication` Response Message

In response to `check_authentication` message, the OpenID Server will send a short message with following parameters:

- **`openid.ns`**
- **`is_valid`** parameter which has a value of “true” or “false”
- **`invalidate_handle`** parameter which is optional and in case of `is_valid` parameter true, the Consumer will remove the handle from its stored list of handles.

A typical message is as follows:

```
invalidate_handle:{HMAC-SHA1}{46071e25}{Tt8MwQ==}  
is_valid:true
```

Note that this is a very short message and the reply includes only success or failure of the authentication check.

3.7 How OpenID Works: Some Scenarios

As you know by now, OpenID is used to authenticate you on different web sites using your credentials stored at your identity server of choice. There are different scenarios of how OpenID authentication will take place.

3.7.1 Scenario One: First Time Login to a Web Site Using OpenID in Dumb Mode

When you login to an OpenID-enabled web site for the first time using OpenID, a couple of things will happen. The following is a list of steps at a very high level:

1. You enter your OpenID URL at the login page of the Consumer web site and click on the Login button.
2. The Consumer web site locates your OpenID server and may use the Yadis protocol to discover the services provided at the URL. It will then redirect your browser to that OpenID server to get your credentials.
3. Since this is the first time you went to this web site, your OpenID server does not know if you trust this web site or not. So your OpenID server will display a login screen to you where you will login to your OpenID server.
4. You mark this web site as a trusted web site at your OpenID server.
5. You are redirected back to this Consumer web site.
6. The web site checks your authentication with the OpenID server and the authentication is complete.

Note that in Smart mode, the Consumer web site as well as the OpenID server will establish an association and will keep a record of the association handle for future use. But in Dumb mode, no association is established and Consumer will not record anything for future use.

3.7.2 Scenario Two: Login to a Trusted Web Site Using OpenID in Smart Mode

You may visit many web sites on regular basis. If these web sites support OpenID, you can mark these web sites as “trusted” at your OpenID server where you store your credentials. Once these web sites are trusted, the login process happens automatically through the following steps.

1. You enter the URL on the Consumer web site login page and click on the Login button.
2. The Consumer web site will establish an association with the OpenID server, if a valid association does not exist.
3. The web site will then locate your OpenID server and redirect your browser to that OpenID server to get your credential. It may also use Yadis to discover the services as well.
4. The OpenID server knows that this web site is trusted, and knows which parameters to pass to this web site. If you have already authenticated to the OpenID server, it will provide the needed credentials to the web site which will be used for login purpose. You will be logged into the web site and all of the steps will happen automatically without any further intervention from your side.

3.8 Problems Solved by OpenID

OpenID solves a number of issues with Identity Management. Some of these are as follows:

- Users get control of what data should be shared with the Consumer web site.

- OpenID allows using stored credentials across all OpenID-enabled web sites. So you don't need to create username and password on each web site individually.
- Stops replay attacks by using one time use nonce variable. A Consumer can ignore a positive assertion by looking at the timestamp in the nonce variable. If the time stamp is too far off from the current time, the Consumer can reject it.

3.9 OpenID Support in Different Languages

OpenID is supported in many programming languages and API's are available. You can find OpenID libraries in the following languages:

- Java
- PHP
- Perl
- C/C++
- C#
- Python
- Ruby
- Cold Fusion

Other companies are working on support in additional languages as well.

3.10 Major Companies Supporting OpenID

Many companies have started supporting OpenID (or a variant of OpenID) on their web sites. Some of the companies and their web sites are as follows:

- AOL at <http://dev.aol.com/openauth> that uses OpenID with some additional features.
- Drupal
- LiveJournal

You can find a list of many other companies at <http://openiddirectory.com>

3.11 Chapter Summary

This chapter covered OpenID protocol flow and OpenID messages. The concepts presented here were:

- OpenID definitions
- Smart and Dumb modes of operation. The smart mode is also called as the “store” mode.
- How OpenID protocol flow works in a step-by-step approach.
- OpenID message types.
- Format of different OpenID messages.
- OpenID URL page and XRD document structure
- HTTP protocol packets captured using Wireshark packet sniffer.

- Difference between OpenID direct and indirect messages. Direct messages use HTTP POST method whereas indirect messages use HTTP GET method.
- OpenID use case scenarios

In the next chapter, you are going to learn how to use OpenID libraries and how to build OpenID enabled web sites.

3.12 References

For more information, you can refer to the following:

- OpenID web site at <http://openid.net>
- Web site for this book at <http://www.openidbook.com>
- Conformix Technologies Inc. <http://www.conformix.com>
- OpenID presentation at <http://openidbook.com/presentations/COLUG-OpenID.pdf>
- OpenID Blog at <http://openid.blogspot.com>
- OpenID information at <http://www.openidenabled.com>
- OpenID Directory at <http://www.openiddirectory.com>

Chapter 4

Creating OpenID Consumer Web Sites

In the previous chapters, there has been plenty of discussion about OpenID concepts to build a basic understanding of OpenID system. At this point you should have fairly good knowledge of the OpenID protocol and how it works. You should also have understanding of different components of an OpenID system and roles of these components. This is the first chapter where you will get into hands-on information about OpenID Consumer (Relying Party or RP) and Identity Provider (OpenID Provider or OP) and how they work together. Specifically, the focus is on building an understanding of the OpenID Consumer

functionality and building OpenID-enabled web sites. You will use JanRain PHP OpenID library¹⁷ and a sample Consumer web site included in this library. However, note that there is no specific reason to use JanRain PHP library examples in this book. Many other OpenID library implementations exist and, at a high level, they work in similar ways although every library has different API functions exposed to end users.

In this Chapter, first of all you will use the sample Consumer program that comes with the library. An existing Identity Provider will be used to test this program. Then you will build your own Consumer web site using the OpenID library.

There are some other topics as well in this Chapter, including graded authorization. You will also explore sample source code of the Consumer applications.

After reading this chapter, you will be able to:

1. Run sample Consumer application
2. View information contained in the OpenID authentication request and response packets
3. Understand how an OpenID library can be integrated into a web application
4. Build your own OpenID-enabled applications
5. Create applications that enable graded authorization

¹⁷ JanRain library can be downloaded from <http://www.openidenabled.com/openid/libraries/>

Sample code presented in this Chapter will be made available for download from OpenID book web site at URL <http://www.openidbook.com>. Sample web application can be tested at <http://www.openidbook.com/knowledgebase> where you can use any OpenID identifier to test graded authorization.

4.1 OpenID Consumer: Step-by-Step Processing

Typically a Consumer will go through multiple steps during the authentication process. You should have a high-level idea of protocol flow based upon your knowledge from previous chapters. In this Chapter, the discussion will be more technical to show how the protocol implementation will work at the source code and packet level.

Most probably, you will use OpenID library provided by some other vendor¹⁸ to build the Consumer application. There are a number of OpenID libraries available in the open source domain. As mentioned earlier, discussion in this book uses JanRain library unless otherwise specified.

A typical Consumer application will go through authentication process as follows:

- Display a web page where an End User can enter identity URL.
- Get the OpenID URL from the End User and create OpenID authentication request.
- Initialize a storage place where the Consumer will store information, if it is working in smart mode. Just to remind you that in Smart Mode, the

¹⁸ You can create your own library if you want to re-invent the wheel!

Consumer has to keep the session information in a persistent storage so that it can be re-utilized.

- Add any additional parameters to the request, like simple registration parameters (e.g. first name, email, and so on)
- Send the request to OpenID server for authentication. The request will also include the redirect path to the Consumer.
- Receive response from the server. This response will be received by the URL included in the request. The web page at receiving URL will process the response and depending upon success or failure, redirect the browser to the next page. In case of authentication failure, this next page is usually an error page displaying a message that authentication failed. Otherwise the user will be logged in and the next page will contain some information for the user showing that the user has logged in.

Different applications and libraries can handle these steps in different ways. In this chapter, the examples will show how sample Consumer applications work.

4.2 Running a Simple Consumer Using JanRain Library

The JanRain library (and other libraries) is available at <http://www.openidenabled.com/openid/libraries/>. The library contains OpenID API functions and sample programs.

After downloading, you will untar the library into a directory. For the demonstration purposes, we have unpacked it under /backup/consumer directory. The source code contains multiple folders under this directory. The main folders of interest are as follows:

- The “Services” folder contains the Yadis protocol f
- The “Auth” folder contains main OpenID library files. This is the main folder for OpenID library and should be included in PHP search path.
- The examples folder contains sample OpenID client and server files. These samples use library files in the above two folders (Services and Auth).

Once you have these files in place, you have to configure the web server so that these library files become accessible when a page request is received.

4.2.1 The Consumer System Configuration

Following is the configuration for the machine on which the sample Consumer application is created.

- Fedora 6
- Apache version 2.2.3
- PHP version 5.1.6

You can also test it with other versions of Linux or Windows machines. I have tested it with Fedora 4 as well and it works fine.

4.2.2 PHP Configuration

PHP must know where OpenID library is installed so that it can access it when needed. There are three basic ways of making PHP aware of OpenID library as listed below:

1. Copy the library files to a location which is already included in the PHP *include path*

2. Modify PHP include path such to include location where you have installed the library files
3. Add library files to the directory tree of your own application

If you are running your application in a third party hosting company environment, the third option listed above may be the best scenario as hosting company may not allow you to change PHP configuration file.

For the sake of example in this book, I have chosen to edit the PHP configuration file. To do so, edit `/etc/php.ini`¹⁹ file to include the following line in it.

```
include_path = "/backup/consumer"
```

Note that the `/backup/consumer` is the directory under which “Auth” and “Services” directories are located. Main OpenID library files are present under the “Auth” directory and Yadis files are under “Services” directory.

After making this change, I would recommend restarting Apache web server to ensure new configuration changes take effect.

4.2.3 Apache Configuration

To test the sample OpenID Consumer application, you have to configure the Apache server so that you can access the application web page. As mentioned earlier, we have installed source code of the sample application in `/backup/consumer` folder. The following lines in Apache configuration are used to create a web site <http://consumer.conformix.com> which will be used for test purposes.

```
<VirtualHost *:80>
```

¹⁹ Depending upon Linux distribution, location of `php.ini` file may be different.

```
ServerAdmin webmaster@consumer.conformix.com
DocumentRoot /backup/consumer
ServerName consumer.conformix.com
ErrorLog logs/consumer.conformix.com-error_log
CustomLog logs/consumer.conformix.com-access_log common
</VirtualHost>
```

Once you have this configuration done, you have to restart Apache for the settings to take effect. On Fedora Linux machine, you can do it by executing the following command. The process/command may be different for other operating systems.

```
/etc/init.d/httpd restart
```

Make sure that you have created a record for `consumer.conformix.com` in the DNS or created an entry in `/etc/hosts` file so that the web address resolves properly while Apache restarts. Otherwise, Apache may complain about it when you restart Apache.

After restarting Apache, now you are ready to launch a web browser and test the application. The sample Consumer files are present under `/backup/consumer/examples/consumer` folder which is the default location for these files. Note that if you like, you can copy files in this folder to any other place as well.

4.2.4 Running the Consumer Example

The sample Consumer application included in the library is very simple. It does not do much other than:

1. Displaying a web page where you enter your OpenID URL
2. Displaying a message showing whether or not the authentication was successful

A more meaningful example will be presented later in this chapter.

To test the application, go to URL “<http://consumer.conformix.com/examples/consumer>” using web browser and you will see the following screen where you will enter your OpenID URL. Note that the DNS must be configured properly to resolve address for consumer.conformix.com.

We created an OpenID Identity URL rrpip.pip.verisignlabs.com in Chapter One. Let us use the same Identity URL with the sample Consumer application as shown in the next Figure.

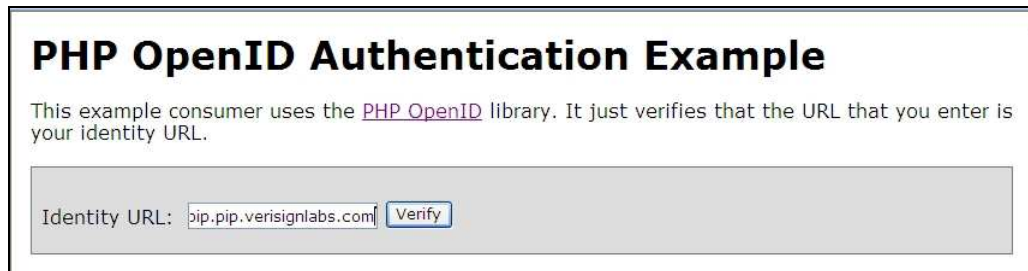
The screenshot shows a web browser window with the title "PHP OpenID Authentication Example". Below the title, there is a paragraph of text: "This example consumer uses the [PHP OpenID](#) library. It just verifies that the URL that you enter is your identity URL." Below this text is a form with a label "Identity URL:" followed by a text input field containing the value "rrpip.pip.verisignlabs.com" and a "Verify" button.

Figure 4-1: Running sample Consumer application included in the JanRain library.

In Figure 4-1, a simple form is displayed where you enter OpenID Identity URL. The form submits the URL to the Consumer. Relevant source code segment for this form is as follows.

```
<form method="get" action="try_auth.php">
  Identity URL:
  <input type="hidden" name="action" value="verify" />
  <input type="text" name="openid_url" value="" />
  <input type="submit" value="Verify" />
</form>
```

Note that the form is submitted back to “try_auth.php” file using the GET method. In your own implementation, you can use POST method as well and OpenID puts no restriction here on the use of GET or POST. Also, in your own

application, you may have a more sophisticated form. For example, you may do client side validation using JavaScript.

4.2.5 URL Sent to the OpenID Server

Once you click the “Verify” button in Figure 4-1, the form will be submitted to “try_auth.php” page which will send an authentication request and you will be redirected to the Identity Provider “pip.verisignlabs.com” where you will be asked to authenticate to allow your Identity URL to be used with the sample client. You can use a packet sniffer like Ethereal²⁰ or Wireshark²¹ to capture the OpenID authentication request/response packets. From the captured packets, you may find something like the following when request is sent to the Identity Provider.

```
https://pip.verisignlabs.com/server?openid.assoc_handle=%7BHMACHMAC-SHA1%7D%7B46033855%7D%7BhpDFZg%3D%3D%7D&openid.identity=http%3A%2F%2Frrpip.pip.verisignlabs.com%2F&openid.mode=checkid_setup&openid.return_to=http%3A%2F%2Fconsumer.conformix.com%3A80%2Fexamples%2Fconsumer%2Ffinish_auth.php%3Fnonce%3DvvVS97if&openid.sreg.optional=email&openid.trust_ro
ot=http%3A%2F%2Fconsumer.conformix.com%3A80%2Fexamples%2Fconsumer
```

Note that there are many characters which are encoded in the above snapshot. It makes it difficult to view the packet in readable format. Following is the same captured packet but with decoded characters so that you can see HTTP parameters in a more clear way.

```
https://pip.verisignlabs.com/server?openid.assoc_handle={HMAC-SHA1}{46033855}{hpDFZg==}&openid.identity=http://rrpip.pip.verisignlabs.com/&openid.mode=checkid_setup&openid.return_to=http://consumer.conformix.com:80/examples/consumer/finish_auth.php?nonce=vvVS97if&openid.sreg
```

²⁰ Available for download at <http://www.ethereal.com>

²¹ Available for download at <http://www.wireshark.org>

```
.optional=email&openid.trust_root=http://consumer.conformix.com:80/exam  
ples/consumer
```

Once you have authorized your ID to be used with the Consumer by entering your username and password on the PIP web site, the authentication will take place and you will see the following screen as shown in Figure 4-2. However, note that many operations take place behind the scenes before you see the screen shown in Figure 4-2. We shall discuss these operations later in this chapter. Basically, some operations take place on the Consumer side and some on the OpenID server side. This chapter is dedicated to the Consumer side processing while the next chapter will show the server side processing in detail.



Figure 4-2: Sample client authentication showing *success* with the Identity Provider

Note that the screenshot shown in Figure 4-2 shows the successful authentication. It also shows the Email address that the OpenID Server returned with the authentication. The sample application does nothing more than that, however it is useful to demonstrate the OpenID protocol.

To give you a little more idea about the response from the OpenID Server, the following is a listing of the response from the server (captured using Wireshark). First the response is shown in encoded form and then after in decoded form to make it more readable. Note that this response shows that the

authentication was successful and also provides the email address that was returned with the authentication.

```
http://consumer.conformix.com/examples/consumer/finish_auth.php?nonce=vvVS97if&openid.sig=AXU8CPMEvGBOqVVzDh%2B%2F72QDins%3D&openid.mode=id_res&openid.return_to=http%3A%2F%2Fconsumer.conformix.com%3A80%2Fexamples%2Fconsumer%2Ffinish_auth.php%3Fnonce%3DvvVS97if&openid.sreg.email=rr%40conformix.com&openid.identity=http%3A%2F%2Frrpip.pip.verisignlabs.com%2F&openid.signed=identity%2Creturn_to%2Cmode%2Csreg.email&openid.assoc_handle=%7BHMACHMAC-SHA1%7D%7B46033855%7D%7BhpDFZg%3D%3D%7D
```

The decoded²² URL is as follows:

```
http://consumer.conformix.com/examples/consumer/finish_auth.php?nonce=vvVS97if&openid.sig=AXU8CPMEvGBOqVVzDh+/72QDins=&openid.mode=id_res&openid.return_to=http://consumer.conformix.com:80/examples/consumer/finish_auth.php?nonce=vvVS97if&openid.sreg.email=rr@conformix.com&openid.identity=http://rrpip.pip.verisignlabs.com/&openid.signed=identity,return_to,mode,sreg.email&openid.assoc_handle={HMAC-SHA1}{46033855}{hpDFZg==}
```

In this and next chapters, there will be a detailed discussion about the contents of the authentication request and response messages. At this stage, we just wanted to show you how request and response packets look like.

Also note that the listings shown above are not complete HTTP packets. We have taken out HTTP header and other parts and have shown only the relevant portions here. When you use Ethereal or Wireshark, you will be able to see the complete HTTP header along with this information.

4.2.6 Storage of Association Information

As you know, during the authentication process, the Consumer will create an *association* with the OpenID server. The association information is used for

²² We have used <http://meyerweb.com/eric/tools/dencoder/> to encode and decode URLs. There are a number of other online tools for encoding and decoding URLs.

subsequent authentication requests in the smart mode as discussed earlier in this book. The sample Consumer application uses files on the disk to store association information. The association information is stored in *store* path which is “/tmp/_php_consumer_test” directory.

You can choose other types of storage with the library as well. For example, you can use MySQL database to store association information.

If you look at the “/tmp/_php_consumer_test” folder, you will see some other folders inside it, which are used to save different files. The following is a typical list of files in this directory.

```
[root@conformix consumer]# ll /tmp/_php_consumer_test/
total 12
drwxr-xr-x  2 apache apache 4096 May 13 21:53 associations
drwxr-xr-x  2 apache apache 4096 May 13 21:53 nonces
drwxr-xr-x  2 apache apache 4096 May 13 21:53 temp
[root@conformix consumer]#
```

The listing below shows contents of a file in which association data is stored. Note that there may be multiple files for storing the association data. Typically there will be one association record for each OpenID server that the application has established association.

```
[root@conformix consumer]# cat
/tmp/_php_consumer_test/associations/https-pip.verisignlabs.com-
QjQghlAPpZnm2u07UII5ffmbXKY-oprnnko0Qnqv3evXuQhrAe.C.yE
assoc_type:HMAC-SHA1
handle:{HMAC-SHA1}{46033855}{hpDFZg==}
issued:1174616028
lifetime:120960
secret:EQvSUvYDv0j4HYQvCluxzbUiWi4=
version:2
[root@conformix consumer]
```

Note that this file has a lot of information in it including the *shared secret* and the *lifetime* for the shared secret.

4.3 Discussion on Sample Consumer

The sample consumer program that comes with the JanRain library is very simple and consists of only few files. The processing starts with “index.php” file which displays the login screen shown in Figure 4-1. This file gets the OpenID URL from the End User and then sends it to “try_auth.php” file. Once the “try_auth.php” file has received the OpenID URL, it does the following:

1. Use “common.php” file that would initialize a Consumer object in the memory. This object is used for authentication purposes and handles all communication. The “common.php” file also initializes a “store” where data will be stored by the Consumer. A *store* is nothing more than a storage place and is used to store *association* information between Consumer and the Identity Provider. As mentioned earlier, the default store for this sample application is the “/tmp/_php_consumer_test” directory.
2. Construct a “process_url” variable which contains the URL where the OpenID server will redirect the browser after authentication process is complete.
3. Construct “trust_root” variable that contains base URL where the OpenID Consumer applications resides.
4. Create authentication request object that will contain all information to be sent to the OpenID server. At this point, the application also adds “email” to the authentication request which is an optional parameter in the request.

5. Send the authentication request to the OpenID server and get the redirect URL from the server upon successful authentication.

Once the request has been sent to the OpenID server, the “try_auth.php” file has completed its job. Now the server will perform authentication and will redirect the web browser using the URL included in the authentication request (the URL contained in “process_url” variable). This URL points to “finish_auth.php” file which will display appropriate message to the End User after authentication is complete.

This whole process is shown in Figure 4-3. When you look at this picture, you can see the “index.php” page gets the OpenID URL from an End User and sends it to “try_auth.php” file. This file then constructs an authentication request and sends it to the OpenID server. The OpenID server processes this request and then sends the result back to “finish_auth.php” file using web browser redirection method.

Please also note that in Figure 4-3, the association mechanism is not shown. This diagram is only to show how the sample Consumer application works and which files are used in the authentication sequence.

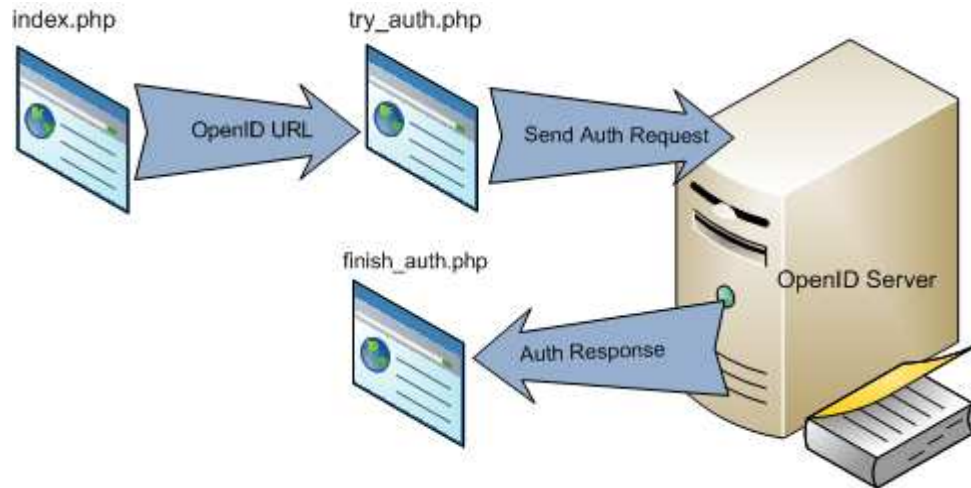


Figure 4-3: Use of sample client source files during authentication request processing.

If you are building your own Consumer application, obviously you will use your own files and file names for this purpose. You can have one complex page to do all of the work that is done by three simple pages shown in Figure 4-3. The point is that as a developer, you have choice of using the OpenID library the way you want.

Next, you will see relevant parts of the three files in the sample Consumer application.

4.3.1 Sample index.php File

This file is responsible to display a web page with an HTML form. You enter your OpenID URL on this web page. The source code for this form is as shown below.

```
<form method="get" action="try_auth.php">
  Identity&nbsp;URL:
  <input type="hidden" name="action" value="verify" />
  <input type="text" name="openid_url" value="" />
  <input type="submit" value="Verify" />
```



```
</form>
```

As you can see, this is very simple page and there is nothing special about this. Once a user clicks on the “Verify” button, the form is submitted to “try_auth.php” script as shown next.

4.3.2 Sample try_auth.php File

The “try_auth.php” is responsible for creating OpenID request and sending it to the OpenID server. The following is a listing of this file.

```
<?php
require_once "common.php";
session_start();

// Render a default page if we got a submission without an openid
// value.
if (empty($_GET['openid_url'])) {
    $error = "Expected an OpenID URL.";
    include 'index.php';
    exit(0);
}

$scheme = 'http';
if (isset($_SERVER['HTTPS']) and $_SERVER['HTTPS'] == 'on') {
    $scheme .= 's';
}

$openid = $_GET['openid_url'];
$process_url = sprintf("%s://%s:%s/finish_auth.php",
    $_SERVER['SERVER_NAME'],
    $_SERVER['SERVER_PORT'],
    dirname($_SERVER['PHP_SELF']));

$trust_root = sprintf("%s://%s:%s",
    $_SERVER['SERVER_NAME'], $_SERVER['SERVER_PORT'],
    dirname($_SERVER['PHP_SELF']));
```

```
// Begin the OpenID authentication process.
$auth_request = $consumer->begin($openid);

// Handle failure status return values.
if (!$auth_request) {
    $error = "Authentication error.";
    include 'index.php';
    exit(0);
}

$auth_request->addExtensionArg('sreg', 'optional', 'email');

// Redirect the user to the OpenID server for authentication. Store
// the token for this authentication so we can verify the response.

$redirect_url = $auth_request->redirectURL($trust_root,
                                           $process_url);

header("Location: ".$redirect_url);

?>
```

Note that the optional parameters are inserted in the request using the following line. In this case you are requesting only email address from the OpenID server.

```
$auth_request->addExtensionArg('sreg', 'optional', 'email');
```

You can create a comma separated list of multiple parameters if want to request more information. For example, the following line will request email and date of birth as optional parameters.

```
$auth_request->addExtensionArg('sreg', 'optional', 'email,dob');
```

Use of multiple parameters is discussed in more detail later in this chapter.

After receiving the request sent by the “try_auth.php” file, the OpenID server processes the request and will send it back to the “finish_auth.php” file which is

shown next. If you look carefully at the listing shown above, you will see that URL pointing to “finish_auth.php” is included in the request.

4.3.3 Sample finish_auth.php File

Following is listing of the finish_auth.php file. Note that this is invoked by a redirect request from the OpenID server to the web browser. All authentication parameters are part of the query string attached to the URL. So this PHP script will look into the query string and determine if the response is a success or failure. It will then display appropriate message.

Note that this script also checks XRI but for the time being we are just going to ignore it and concentrate only on other parameters.

```
<?php

require_once "common.php";
session_start();

// Complete the authentication process using the server's response.
$response = $consumer->complete($_GET);

if ($response->status == Auth_OpenID_CANCEL) {
    // This means the authentication was cancelled.
    $msg = 'Verification cancelled.';
} else if ($response->status == Auth_OpenID_FAILURE) {
    $msg = "OpenID authentication failed: " . $response->message;
} else if ($response->status == Auth_OpenID_SUCCESS) {
    // This means the authentication succeeded.
    $openid = $response->identity_url;
    $esc_identity = htmlspecialchars($openid, ENT_QUOTES);
    $success = sprintf('You have successfully verified '
        . '<a href="%s">%s</a> as your identity.',
        $esc_identity, $esc_identity);

    if ($response->endpoint->canonicalID) {
```

```
        $success .= ' (XRI CanonicalID: '.$response->endpoint->canonicalID.') ';\n    }\n\n    $sreg = $response->extensionResponse('sreg');\n\n    if (@$sreg['email']) {\n        $success .= " You also returned '".$sreg['email']."' as your\nemail.";\n    }\n}\n\ninclude 'index.php';\n\n?>
```

The “index.php”, included at the end of this script, will display appropriate message contains in the “success” variable and then display the form where you can check a different OpenID URL. Please refer to Figure 4-2 to see the output of this process.

4.4 Requesting Additional Parameters using Simple Registration Extension

OpenID Simple Registration extension is used to get most commonly used parameters associated with a user. For example, you can request date of birth, email address, full name, and so on while sending an authentication request. Although, this extension will be discussed in detail in Chapter 6, here we just want to give you a short overview about how it is used with JanRain library.

In the previous section, when you sent authentication request and also requested the email address with it, you actually used simple registration extension. The line of code that you used to add email address to the request was as follows:

```
$auth_request->addExtensionArg('sreg', 'optional', 'email');
```

In the above line, there are three arguments to the function call `addExtensionArg`. These are explained below.

1. The “sreg” argument shows that you are adding simple registration extension.
2. The “optional” argument shows that you are informing the Identity Provider (OpenID server) that this parameter is optional. This means that Identity Provider may ignore this parameter if it needs to. It also means that the End User who is the owner of the Identity URL may also chose not to send this parameter back with the response message, if the Identity Provider gives the End User an option to do so. If the Consumer must have this parameter from the Identity Provider, you can replace “optional” with “required” to inform the Identity Provider that this argument is not optional and authentication will fail if it is not available.
3. The third argument, which is “email” shows the actual information that the Consumer is requesting. In this case it is email address of the End User.

You can also request multiple parameters with an authentication request. The following line in the program will send a request for four parameters: email address, fullname, nickname, and date of birth.

```
$auth_request->addExtensionArg('sreg', 'optional',  
'email,fullname,nickname,dob');
```

Note that all of the “optional” parameters are bundled together in a single function call. The parameters in the list are separated by commas.

When you send authentication request with multiple parameters as mentioned above, the OpenID server will give you a chance to decide which parameters you want to return back the Consumer. This is shown in Figure 4-4 where the

OpenID server is asking the End User which parameters should be sent to the Consumer.

In Figure 4-4, and the remaining part of this book, you will use the following settings:

- Identity Provider or OpenID server will be “idp.conformix.com” which is installed for the purpose of examples in this book. You will learn how to install your own OpenID server in the next chapter.
- Consumer application will be at web location “consumer.conformix.com”.

Some other sample applications will also be built and we shall show how to do so as we go through this book.

OpenID Book Test Server

[Home](#)
[My Profile](#)
[Sites](#)
[Log out](#)
Logged in as *openidbook*

Do you wish to confirm your identity URL (<http://idp.conformix.com/?user=openidbook>) with <http://consumer.conformix.com:80/?>

The server has also requested the transfer of profile information. The required and optional fields are listed below. Uncheck the "Send?" checkbox for fields that you don't want to send.

Send?	Name	Value	Status
<input checked="" type="checkbox"/>	Nickname	rr	Optional
<input checked="" type="checkbox"/>	E-mail address	rr@conformix.com	Optional
<input checked="" type="checkbox"/>	Full name	Rafeeq Rehman	Optional
<input checked="" type="checkbox"/>	Birth date	1904-01-25	Optional

The server supplied no data policy URL.

OpenID Book Test Server | Contact rr@conformix.com

Figure 4-4: Requesting multiple optional parameters using simple extensions.

The web page shown in Figure 4-4 is displayed *after* you have sent authentication request and the *before* OpenID has responded back. Here the server is showing that a Consumer “http://consumer.conformix.com” running on port number 80 is requesting authentication for URL “http://idp.conformix.com/?user=openidbook”. The Consumer is also requesting four optional parameters as shown in this Figure under the “Name” column. The server also displays the current values of these parameters in the “Value” column. These values are taken from the stored profile for the Identity URL. As shown in the Figure, the stored value of “Nickname” inside the server database is “rr”, email address is “rr@conformix.com”, and so on²³. If you keep all checkboxes as checked, all of these values will be sent back to the Consumer. However, if you don’t want to send some of these values back to the Consumer, you can uncheck the checkboxes. The last column with the “Status” heading shows whether a parameter is “optional” or “required”. In this Figure, all parameters are shown as “optional”. Remember that you may chose not to send back value of any “optional” parameter during the authentication process.

You can also request a combination of “optional” and “required” parameters with the authentication request. The following two lines of code add three “optional” parameters which are email, fullname, and nickname and one “required” parameter which is date of birth (dob). Note that all “optional” parameters are bundled together and all “required” parameters are bundled together in the next API function call.

```
$auth_request->addExtensionArg('sreg', 'optional',  
    'email,fullname,nickname');  
$auth_request->addExtensionArg('sreg', 'required', 'dob');
```

When you send this request to the OpenID server (idp.conformix.com), it will show a web page to you similar to the one in Figure 4-5.

²³ Note that the January 25, 1904 is not my real date of birth. Just wanted to clarify!

OpenID Book Test Server

[Home](#)
[My Profile](#)
[Sites](#)
[Log out](#)

Logged in as *openidbook*

Do you wish to confirm your identity URL (<http://idp.conformix.com/?user=openidbook>) with <http://consumer.conformix.com:80/>

The server has also requested the transfer of profile information. The required and optional fields are listed below. Uncheck the "Send?" checkbox for fields that you don't want to send.

Send?	Name	Value	Status
<input checked="" type="checkbox"/>	Nickname	rr	Optional
<input checked="" type="checkbox"/>	E-mail address	rr@conformix.com	Optional
<input checked="" type="checkbox"/>	Full name	Rafeeq Rehman	Optional
<input checked="" type="checkbox"/>	Birth date	1904-01-25	Required

The server supplied no data policy URL.

OpenID Book Test Server | Contact rr@conformix.com

Figure 4-5: Requesting multiple *optional* and *required* parameters using simple extensions.

In this Figure, the “Status” column shows that the date of birth is a “Required” field and if you uncheck and don’t send it back to the Consumer, the authentication will fail.

In addition to sending request for different parameters with the authentication request, the Consumer should also be able to handle the returned parameters when the reply is received. In the sample Consumer application, the “finish_auth.php” script handled the returned parameter “email” using the following lines of code.

```
$sreg = $response->extensionResponse('sreg');

if (@$sreg['email']) {
    $success .= " You also returned '". $sreg['email']. "' as your email.";
}
```


When you are requesting multiple parameters, you have to add code in the “finish_auth.php” file to handle additional parameters. For example, you can have following lines in this file to check all of the four parameters returned by the OpenID server.

```
$sreg = $response->extensionResponse('sreg');

if (@$sreg['email']) {
    $success .= " You also returned '". $sreg['email']. "' as your email.";
}
if (@$sreg['fullname']) {
    $success .= " You also returned '". $sreg['fullname']. "' as your
fullname.";
}
if (@$sreg['nickname']) {
    $success .= " Your postal code is '". $sreg['nickname']. "' as your
nickname";
}
if (@$sreg['dob']) {
    $success .= " Your postal code is '". $sreg['dob']. "' as your Date of
Birth";
}
```

Figure 4-6 shows the response web page after making the above changes. In your application login process, if you are expecting a parameter which is not included in the response from the OpenID server, you can display an appropriate message to indicate that a required parameter is missing.

PHP OpenID Authentication Example

This example consumer uses the [PHP OpenID](#) library. It just verifies that the URL that you enter is your identity URL.

You have successfully verified <http://idp.conformix.com/?user=openidbook> as your identity. You also returned 'rr@conformix.com' as your email. You also returned 'Rafeeq Rehman' as your fullname. You also returned 'rr' as your nickname. You also returned '1904-01-25' as your Date of Birth.

Identity URL:

Figure 4-6: Response for multiple parameters.

Declaring parameters as “optional” or “required” is a powerful mechanism that can be used for different purposes. Graded authorization is one of these mechanisms which is discussed next. It is also used for a new user registration by many Consumer web sites when a person logs into a Consumer web site for the first time.

4.5 Risk Based Access Control and Graded Authorization

Using OpenID, A web site can provide different levels of access based upon information received from Identity Provider in a user profile. For example, a company providing blog space can provide multiple levels of access to web site visitors. Following is an example of some of the features that can be implemented.

- The web site can accept any credentials to provide read-only access to the web site visitors.

- If a visitor wants to post comments, the person must provide an email address, first name, and last name.
- For anyone who wants to get access to post new articles, needs to provide zip code in addition to all other information.

Similarly, a financial services company may allow a person to look at the current statement by providing a basic set of information and after matching the provided information with the one which is already on customer record file. On the other hand, to make a financial transaction, the person has to provide a higher level of information which may include date of birth, first/last name, mailing address, etc, depending upon the risk level. OpenID enabled web sites can be built to detect which information is needed for a particular type of service provided by web sites.

In a typical web application with graded authorization, you will assign an access level to different types of privileges. When someone wants to get those privileges, you would check if sufficient credentials are presented to the application. If not, you would allow only that level of access for which the credentials are sufficient.

In the next section, you are going to build an application that provides three levels of access depending upon which parameters you provide with the authentication.

4.5.1 Sample Web Site Using OpenID Consumer

Conformix Knowledge Base is a sample implementation of graded authorization. The application implements a knowledge system that is used for technical support purposes. Customers can view, update, or add knowledgebase articles depending upon level of access granted. The level of access is determined based upon different parameters that are requested during the authentication process.

Clarity is the main purpose for this application and there are much better ways to implement it in a real environment. The purpose is to clarify the concept of graded authorization. The application is available at <http://www.openidbook.com/knowledgebase> where you can test using your OpenID URL.

Articles are stored in a MySQL backend database. Each knowledgebase article in this application has three simple parts as listed below:

1. Article name
2. Summary
3. Detail

Users are authenticated at different levels to get access these articles. The more information a user provides, the higher level of access the user gets. The following rules apply to this application.

1. A user can login using OpenID identity URL. However, the user has to provide email address which is mandatory. If an email address is not provided, the user is sent back to login page.
2. Once a user logs in using OpenID identity URL and provides “email” parameter, the user gets read-only access to the list of knowledgebase articles. However, the user can only see the article “Name” and “Summary” parts and is not able to view the “Detail” part of the article.
3. If a user also provides “fullname”, in addition to “email”, the user also gets access to the “Detail” part of knowledgebase articles. The user can also edit the article but can’t add a new one.
4. If a user provides “email”, “fullname”, and “dob” (date of birth), the user also gets the privilege to edit knowledgebase articles.

These rules are implemented in this application. Next, you will find detailed information about how this application is built and how to test it.

4.5.1.1 Backend Database

A backend database is used to store knowledgebase articles and capture user information. There are two main tables in the database: the “user” table saves information about the web site users and the “article” table stores information about knowledgebase articles.

The “user” table has the following fields:

- userId
- OpenIDURL
- fullName
- email
- dob

The “article” table has following fields

- articleId
- name
- summary
- detail

The following script creates the database “knowledgebase”, a user “kbuser” with a password “kbpassword”. The script then creates these two tables in the database. This is a MySQL script and can be used with other databases as well with little or no changes.

```
CREATE DATABASE knowledgebase;
GRANT ALL ON knowledgebase.* TO 'kbuser'@'localhost' IDENTIFIED BY
'kbpasword';
USE knowledgebase;
DROP TABLE IF EXISTS `article`;
CREATE TABLE `article` (
  `articleId` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `summary` text,
  `detail` text,
  PRIMARY KEY (`articleId`)
) ENGINE=InnoDB;
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `userId` int(10) unsigned NOT NULL auto_increment,
  `OpenIDURL` varchar(255) default NULL,
  `fullName` varchar(255) default NULL,
  `email` varchar(255) default NULL,
  `dob` datetime default NULL,
  PRIMARY KEY (`userId`)
) ENGINE=InnoDB ;
```

Once you have created the database, you can download the application and install it on your own computer. You need to have PHP, MySQL, and Apache running on the computer where you install it.

4.5.1.2 Source Files

Following is a list of source files used in this sample application. You can also download the source code files in tar format from <http://www.openidbook.com>. JanRain library is used for OpenID authentication and you can configure it as discussed earlier in this chapter. Other source files are as follows:

- The “index.php” file shows basic information about this application.
- The “login.php” file is used to login to the application.

- The “article.php” file is used to display knowledgebase articles. It used PHPMyEdit library to generate grid layout.
- The “finish_auth.php” controls authorization level based upon information provided by the Identity Provider. This file is included in the “article.php” and is not directly called.
- The “include/menu.php” creates menus for the application.
- The “include/header.php” and “include/footer.php” display header and footer information respectively.
- The “config.php” file present in the “include” directory includes database configuration and access information.

There are other files in this application which are part of PHPMyEdit. These files are used to display records in the database in the grid format. For understanding this application, you can skip those.

The index.php source code is as shown below. This file displays the main page of the application where you can click on the “Login” link to go to the login page.

```
<?php
/*****
***
Written by: Rafeeq Ur Rehman
    Copyright (c) 2001-2007 Conformix Technologies Inc. All rights
reserved.
*****/
ini_set('output_buffering', '9000');
ini_set('display_errors', '0');
?>
<link rel="stylesheet" type="text/css" media="screen"
href="include/new.css">
<?php
```

```
header("Cache-control: private");

include "include/header.php";
include "include/menu.php";
show_menu_submit("home");
show_menu_blank();
?>

<p><table><tr><td bgcolor=#02a043><center><H1><font color=white>
    Knowledgebase Sample Application</td></tr></table>
<p><table><tr><td bgcolor=#F8F8F8><center><H1>Welcome to the
    Knowledgebase Sample Application Web Site</H1></center><p>
    <center>This web application is used to demonstrate risk-based
    and multi-level access control using OpenID protocol. Depending
    upon the parameters received from OpenID Provider, different
    level of access will be granted to a user.
    <p>For more information, contact <a
href=mailto:rafeeq.rehman@gmail.com>
    rafeeq.rehman@gmail.com </a></td><td></tr></table>

<?php include "include/footer.php";?>
```

The login.php source code is shown below. This page displays a text box where you can enter your OpenID URL.

```
<?php
/*****
***
Written by: Rafeeq Ur Rehman
    Copyright (c) 2001-2007 Conformix Technologies Inc. All rights
reserved.
*****/
ini_set('output_buffering', '9000');
ini_set('display_errors', '0');
?>
<link rel="stylesheet" type="text/css" media="screen"
href="include/new.css">
```



```
<?php
header("Cache-control: private");

include "include/header.php";
include "include/menu.php";
show_menu_submit("login");
show_menu_blank();

echo "<p><table><tr><td bgcolor=\"\#02a043\"><center><H1><font
color=white>" .
        "Knowledgebase Sample Application" .
        "</td></tr></table>";

?>

<form method="get" action="try_auth.php">
    Enter your OpenID URL:
    <input type="text" name="openid_url" value="" />
    <input type="submit" value="Login" />
</form>

<?php include "include/footer.php"; ?>
```

The following part in the “finish_auth.php” file sets the permission level for the user who logged in. This is done by setting a session variable “level”. Note that all pages in the application can check value of this variable and then grant certain level of permission to the logged in user.

```
$email=$_SESSION['email'];
$fullname=$_SESSION['fullname'];
$dob=$_SESSION['dob'];
$nickname=$_SESSION['nickname'];
$openid=$_SESSION['openid'];

if (strlen($email)>0 AND strlen($dob)>0 AND strlen($fullname)>0) {
    $_SESSION['level'] = ACCESS_FULL;
}
else if (strlen($email)>0 AND strlen($fullname)>0) {
    $_SESSION['level'] = ACCESS_UPDATE;
```

```
}  
else if (strlen($email)>0) {  
    $_SESSION['level'] = ACCESS_READ_ONLY;  
}  
else {  
    session_unset();  
    session_destroy();  
    header("location: login.php");  
}  
header("location: article.php");
```

Also note that if “email” is not provided with the authentication response from the OpenID server, the user is sent back to the login page (because email is a “required” parameter).

The following section in the “article.php” file checks the value of the “level” variable and then sets appropriate level of permission.

```
session_start();  
$email=$_SESSION['email'];  
$fullname=$_SESSION['fullname'];  
$dob=$_SESSION['dob'];  
$nickname=$_SESSION['nickname'];  
$openid=$_SESSION['openid'];  
$level=$_SESSION['level'];  
  
if ($level == ACCESS_FULL) {  
    $opts['options'] = 'LACPVDf';  
}  
else if ($level == ACCESS_UPDATE) {  
    $opts['options'] = 'CVL';  
}  
else if ($level == ACCESS_READ_ONLY) {  
    $opts['options'] = 'L';  
}  
else {  
    session_unset();  
    session_destroy();
```

```
header("location: login.php");  
}
```

You can check documentation for PHPMyEdit for more information about how values of variables like 'LACPVDF' control access to data.

4.5.1.3 Application Logic

The web application is very simple. When a user enters the OpenID URL, the web application acts as relying party (Consumer) and authenticates the user against OpenID server. It sends additional parameters in the authentication request as listed earlier. Following are important points about the application logic.

- Request for the “email” parameter is sent as “required” parameter.
- Request for “dob”, “nickname”, and “fullname” is sent as optional parameters.
- If the received response does not contain “email” address, or the email address is not valid, the authentication fails and user is asked to authenticate again.
- If the “email” parameter is returned back but “dob” and “fullname” are missing, read-only access to article “name” and “summary” part is granted.
- If “email” and “dob” parameters are returned but “fullname” is missing, read-only access to “name” and “summary” parts of articles is provided.
- If “email” and “fullname” parameters are returned but “dob” is missing, read-write access to all parts of articles is provided. However, you can’t add a new article to the database.

- If all “email”, “fullname” and “dob” parameters are returned, full read-write access is granted to the user.

Using these principles, you can understand the operation of this application. In the next part, you will use this application to verify these rules.

4.5.1.4 Using Sample Application

After installing the application, you will point your browser to the application URL. You can also use the pre-configured application at <http://www.openidbook.com/knowledgebase> for simplicity. You will see something like Figure 4-7 after launching this application.



Figure 4-7: Main page for sample Knowledgebase application

To login to the application, click on the Login link on the top-left side of this page and you will see the text box to enter your OpenID URL as shown in Figure 4-8.

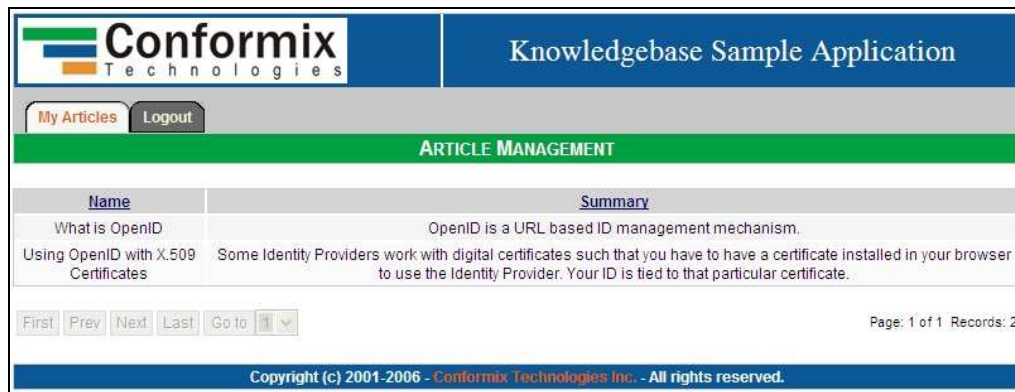
In Figure 4-8, you will enter the OpenID URL and then click “Login” button.



Figure 4-8: Login page for sample Knowledgebase application

Once you have clicked on the “Login” button in Figure 4-8, depending upon your OpenID URL, the browser will be redirected to the OpenID server. The OpenID server may present web page where you may have to authenticate yourself and then ask you which parameters are requested and which values to send back. This may something similar to as shown in Figure 4-5.

Once you have authenticated to the OpenID server, the server will send back the selected parameters. Depending upon returned values, the application will give you appropriate permissions to certain features. Figure 4-9 shows lowest level of access where you can see only “Name” and “Summary” for each article and nothing more.



<u>Name</u>	<u>Summary</u>
What is OpenID	OpenID is a URL based ID management mechanism.
Using OpenID with X.509 Certificates	Some Identity Providers work with digital certificates such that you have to have a certificate installed in your browser to use the Identity Provider. Your ID is tied to that particular certificate.

Figure 4-9: Read-only access to knowledgebase

Figure 4-10 shows the screenshot where you get the privilege to view and update existing articles. For this purpose, you can use button at the bottom of the list or icons in the left-most column.



Figure 4-10: Update access to knowledgebase.

When you edit an article, you will see a screenshot shown in Figure 4-11 where you can make changes and then update the article.

The screenshot shows the 'ARTICLE MANAGEMENT' form in the Conformix Knowledgebase Sample Application. The form has three main sections: 'Name', 'Summary', and 'Detail'. The 'Name' field contains 'What is OpenID'. The 'Summary' field contains 'OpenID is a URL based ID management mechanism.' The 'Detail' field contains 'Detail about OpenID can be found at <http://openid.net>. A book is also available on OpenID at URL <http://www.openidbook.com>'. At the bottom of the form are 'Save', 'Apply', and 'Cancel' buttons. The footer of the application states 'Copyright (c) 2001-2006 - Conformix Technologies Inc. - All rights reserved.'

Figure 4-11: Updating articles in the database.

Figure 4-12 shows a screenshot where you get full rights for the application and you are able to add/delete/modify/view articles.

The screenshot shows the 'ARTICLE MANAGEMENT' table in the Conformix Knowledgebase Sample Application. The table has two columns: 'Name' and 'Summary'. The first article is 'What is OpenID' with the summary 'OpenID is a URL based ID management mechanism.' The second article is 'Using OpenID with X.509 Certificates' with the summary 'Some Identity Providers work with digital certificates such that you have to have a certificate installed in your browser to use the Identity Provider. Your ID is tied to that particular certificate.' Below the table are navigation buttons: 'First', 'Prev', 'Add', 'View', 'Change', 'Copy', 'Delete', 'Next', 'Last', 'Go to', and a dropdown menu. The status bar indicates '1 record changed' and 'Page: 1 of 1 Records: 2'. The footer of the application states 'Copyright (c) 2001-2006 - Conformix Technologies Inc. - All rights reserved.'

Figure 4-12: Full access to knowledgebase

This example shows how OpenID can be used to grant different level of access to users of an application. In some literature this is also known as *graded authorization*.

4.5.2 One-Time Authorization

In some cases, an application may not be interested in authentication itself. The application may want to give access to a user depending upon the information provided. In that scenario, the application will ask some parameters to identify a user by comparing response from the Identity Provider by some data that the application already knows about a user.

For example, an insurance company may ask data like first name, last name, date of birth, and address to allow a user to pay bills, without showing any policy information. One Time Authorization may be useful in such a scenario.

4.6 Storing Credentials by OpenID Consumers

A Consumer will store OpenID association credentials when working in the smart mode. These credentials can be stored in different ways, including caching in database, using flat files on disk, and so on.

In the sample application, we have used flat files “store” to cache credentials. By changing the type of “store”, you can cache this data into database as well. The OpenID libraries provide different mechanisms for selecting “store”.

4.7 Web Browser Support and Browser Plugins

OpenID specifications provide recommendation about creating the Login page for Consumer web sites so that it is east for web browsers or other user agents to identify if the web page is requesting an OpenID URL. Following is the extract from section 7.1 of the OpenID specification version 2, draft 11.

“The form field's "name" attribute SHOULD have the value "openid_identifier", so that User-Agents can automatically determine that this is an OpenID form. Browser extensions or other software that support OpenID Authentication may not detect a Relying Party's support if this attribute is not set appropriately.”

OpenID SeatBelt is one such plugin for Firefox browser from Verisign Labs. It detects if a web page has certain identifiers as listed about on the web page. If it finds a web page as OpenID login page, it can automatically fill in your OpenID URL in the appropriate text box. The plugin can also help in preventing phishing URLs.

Detailed information about the SeatBelt plugin is available at <http://beta.abtain.com/account/resources.jsp> or <https://jpip.verisignlabs.com/seatbelt.do>

The advantage of using a plugin like this is that you authenticate to your Identity Provider only once and then the plugin keeps you logged in. Now when you go from one OpenID enabled web site to another, you the plugin does the login work for you.

4.8 OpenID Libraries

There are many OpenID libraries that are available in different languages. Some of the libraries are as listed below:

- PHP
- Java
- Perl
- Ruby

- Python
- C

To get a complete and latest list of all libraries, please visit <http://openid.net/wiki/index.php/Libraries>

New libraries are being created continuously so I would advise to check the above URL if you are looking for a particular library.

4.9 Chapter Summary

In this chapter, you have learned some very important concepts about OpenID. You also learned how to create OpenID enabled web sites and perform graded authorization. You also saw some of the OpenID messages. A sample OpenID application was also presented.

OpenID simple registration extensions are used to pass additional parameters to the Consumer, if requested. The owner of the OpenID gets a chance to review request for these additional parameters and can make a decision about which parameters to send back and which ones should be refused. The Consumer application then decides what level of access to grant depending upon which parameters are sent back to Consumer.

Sample Consumer application which implements one way of graded authorization is available at <http://www.openidbook.com/knowledgebase>.

4.10 References

For more information, you can refer to the following:

- OpenID web site at <http://openid.net>

- Web site for this book at <http://www.openidbook.com>
- Conformix Technologies Inc. <http://www.conformix.com>
- OpenID presentation at <http://openidbook.com/presentations/COLUG-OpenID.pdf>
- OpenID Blog at <http://openid.blogspot.com>
- OpenID information at <http://www.openidenabled.com>
- OpenID Libraries at <http://openid.net/wiki/index.php/Libraries>
- JanRain PHP Libraries at <http://www.openidenabled.com/openid/libraries/php>
- SeatBelt Plugin for Firefox browser at <http://beta.abtain.com/account/resources.jsp>
- OpenID Graded Authorization sample application at <http://www.openidbook.com/knowledgebase>

Chapter Five

Running OpenID Server

Now that you have already created your web site as Consumer, it is the time to run your own Identity Server. If you are an individual user of OpenID, most probably you don't need to run your own OpenID server. There are so many free OpenID servers already available on the Internet. However, if you are thinking about using OpenID for your own service on the Internet or inside your organization, running your own server may be a good idea.

Like OpenID libraries, there are multiple implementations of OpenID server in different environments. In fact, JanRain OpenID library comes with sample implementation of a server which can be used for understanding and testing. However, for a real server, this example implementation is not enough because

you have to have a back-end database as well as a user interface where your users can go and create new ID URLs.

The OpenID server should also implement the Yadis protocol used for service discovery. Yadis helps OpenID clients to identify the available services and we shall look into Yadis in a little more detail in this chapter.

The main objective of this chapter is to set up a working server, show how clients will work with this server, and show a database that works behind the scenes to store information. You will also see detailed view of OpenID messages. For the purpose of in-depth demonstration, we have used Wireshark (<http://www.wireshark.org>) packet sniffer to capture HTTP packets flowing on the network. These packets will show you how the OpenID protocol messages are embedded in the HTTP packets.

In this chapter, we have used JanRain OpenID server version 1.1. We have used sample consumer application that comes with JanRain OpenID API implementation. The set up used in this chapter is as shown in Figure 5-1.

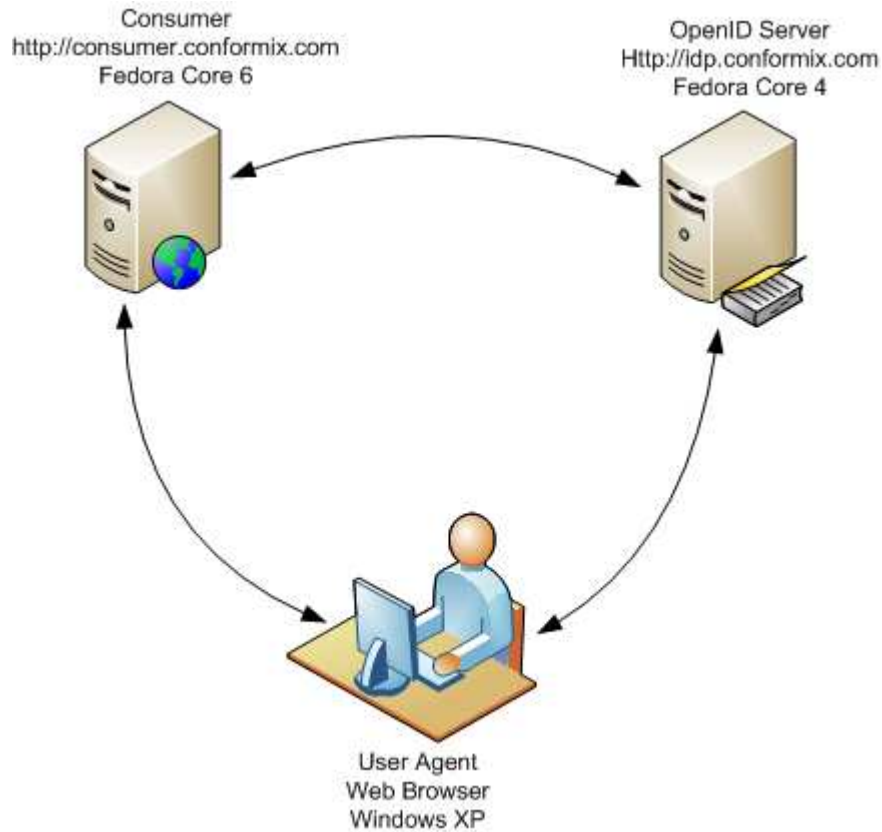


Figure 5-1: Set up used for OpenID server, Consumer, and User Agent.

The platform used for OpenID server (idp.conformix.com) is:

- Fedora Core 4
- MySQL version 4.1.20
- PHP version 5.0.4
- Apache version 2.0.54

The platform used for the consumer application (consumer.conformix.com) is:

- Fedora Core 6
- MySQL version 5.0.27
- PHP version 5.1.6
- Apache version 2.2.3

The End User is using a Microsoft Windows XP client with IE7 browser to login to the Consumer application.

Other helping software used to set up the server correctly is Smarty (<http://smarty.php.net/>) and we used version 2.6.18 for this. Also note that you must have XML support on OpenID server and Consumer machines to handle XML requests that are part of Yadis protocol.

5.1 PHP OpenID Server Installation

To install OpenID server, you have to have Apache (or some other web server) installed. You would also need a database that the server can use to store data. For this chapter, we are going to use Apache web server and MySQL database. During installation, you will uncompress the server files and put them inside a folder accessible by the web server.

We have configured a virtual web server to serve a sub-domain `idp.conformix.com`.

5.1.1 Downloading and Extracting Files

Download the server tar file from <http://www.openidenabled.com/resources/downloads/php-server/PHP-server-1.1.tar.gz>. There may be a newer version available by the time you read

this book. Please note that all links are available at the book's web site <http://www.openidbook.com> where latest links are updated.

After downloading, extract files under the folder used by the virtual server, using the following command.

```
tar zxvf PHP-server-1.1.tar.gz
```

This will extract all files. We have placed the extracted files under /backup/openidbook folder. A number of sub-directories will be created when you untar the source code. You can rename the folder as you wish, but you have to enable access to the template_c folder to the account running the web server. Use the following command for this purpose. Note that there are more secure ways to achieve the same objective and this example is to use the simplest method to get the server up and running.

```
chown apache.apache templates_c
```

On other types of servers (other than Apache on Linux), the method to set permissions may be different. Once you have set the permissions correctly, it is the time to move to the next step.

5.1.2 Configuring Apache

Apache configuration is simple. We are creating a virtual host idp.conformix.com that will be used as OpenID server. You can create a virtual host with a name of your choice.

For creating a virtual server, you will add the following lines at the end of /etc/httpd/conf/httpd.conf file.

```
<VirtualHost *:80>
```



```
ServerAdmin webmaster@ipd.conformix.com
DocumentRoot /backup/openidbook/srv
ServerName idp.conformix.com
ErrorLog logs/idp.conformix.com-error_log
CustomLog logs/idp.conformix.com-access_log common
</VirtualHost>
```

Note that DocumentRoot points to /backup/openidbook/srv where OpenID server source code is placed in the previous step.

Once you have saved the httpd.conf file, you have to restart the Apache server (or send HUP signal) so that it reloads the configuration file. On Fedora Core 4, you will use the following command to restart Apache.

```
/etc/init.d/httpd restart
```

After executing the above command, the Apache server is ready to accept requests for <http://idp.conformix.com>.

5.1.3 Installing Smarty

Smarty (<http://smarty.php.net>) is a PHP package and is needed for PHP templates for the OpenID server. Grab the latest version of Smarty, untar it and then put the library files under /usr/local/lib/php/Smarty directory. Note that you can use any other directory as long as it is under PHP path. To accomplish this, you will use the following three commands.

```
tar zxvf Smarty-2.6.18.tar.gz
mkdir /usr/local/lib/php/Smarty
cp -r Smarty-2.6.18/libs/* /usr/local/lib/php/Smarty
```

Installing Smarty is done once you have copied the library files as listed above.

5.1.4 Install and Configure JanRain PHP OpenID Library

To run the JanRain OpenID server, you have to have the JanRain PHP library for OpenID installed and configured properly. You can install the library anywhere as long as it is accessible to PHP. We have installed it under `/share/openidbook` directory.

First of all, download the library from <http://www.openidenabled.com/resources/downloads/php-openid/PHP-openid-1.2.2.tar.gz> which is the latest version at the time of writing this book. Then use the tar command to extract files as follows:

```
tar zxvf PHP-openid-1.2.2.tar.gz
```

Once you have extracted file, you will put them in a directory and include the part to the directory in `/etc/php.ini` file. Since we have placed it under `/backup/openidbook`, we need to have the following line in `/etc/php.ini` file.

```
include_path = ".: /backup/openidbook: /usr/share/pear"
```

We would recommend restarting the Apache server after making these changes to the `php.ini` file.

5.1.5 Configuring MySQL Database

As mentioned previously, the OpenID server needs a database where it will store its permanent as well as transient information. You just need to create the database and a user that has appropriate privileges to the database. The server will create tables in the database by itself.

To create the database, login as root and then you will use the “mysql -p” command which will ask you to enter password for user root. Once you see the “mysql>” prompt, then you can use the following three commands to create a new database and a new user with sufficient privileges.

```
create database idpDB;
```

```
grant all privileges on idpDB.* to 'idpUser'@'localhost' identified by  
'idpPassword';  
flush privileges;
```

The first command creates a database with name “idpDB”. The second command creates a user “idpUser” with a password “idpPassword” and assigns privileges to this user to manage the database. The third command forces the MySQL server to re-reads privileges set by the second command.

Once you have completed these tasks, the database is ready for use. The OpenID server will create different in the database to:

- Store user information and their password so that all OpenID URL holders can manage their accounts using the web interface.
- Information about trusted sites.
- Information about associations for OpenID smart mode. This information will include shared keys associated with different Consumers.
- Session log information

There are other pieces of information that will be stored in the database and we are going to provide details of different tables in the following pages in this chapter.

5.1.6 Updating Configuration Files

The src/config.php file is the main configuration file for the OpenID server. Major changes needed in this file are as follows:

1. Configuring database parameters including database host, database name, username, and password.

2. Configuring Smarty directory where Smarty library files and profiles are present.
3. Configure contact Email address which is displayed on user interface.
4. Configuring site title
5. Configuring minimum username and password lengths.
6. Whether to allow public user registration or not.
7. Administrator username and password
8. Storage and authentication backend database.

The sample configuration config.php file is as follows:

```
<?php
/**
 * The location of the Smarty templating system; set this to the
 * directory that contains Smarty.class.php. Must end in a trailing
 * slash.
 */
define('SMARTY_DIR', '/usr/share/php/Smarty/');

/**
 * The site title; this will appear at the top and bottom of each
 * page, as well as in the browser title bar.
 */
define('SITE_TITLE', "OpenID Book Test Server");

/**
 * The administrator's email address. You may leave this empty if you
 * wish. If empty, the "Contact (email address)" message will not
 * appear on every page footer.
 */
define('SITE_ADMIN_EMAIL', "rr@conformix.com");
```

```
/**
 * Minimum username length for account registration.
 */
define('MIN_USERNAME_LENGTH', 2);

/**
 * Minimum password length for account registration.
 */
define('MIN_PASSWORD_LENGTH', 6);

/**
 * Set this to true if you want to allow public OpenID registration.
 * In either case, the ADMIN_USERNAME account specified below will be
 * able to log in to create and remove accounts.
 */
define('ALLOW_PUBLIC_REGISTRATION', true);

/**
 * Set these values for administrative access. This account will be
 * able to create and remove accounts from the auth backend. This
 * username will not be permitted to use an OpenID. The password MUST
 * be an MD5 hexadecimal hash of the password you want to use.
 * Example:
 *
 * define('ADMIN_PASSWORD_MD5', '21232f297a57a5a743894a0e4a801fc3');
 *
 */
define('ADMIN_USERNAME', 'admin');
define('ADMIN_PASSWORD_MD5', '');

/**
 * Storage backend to use. Currently the only choice is "MYSQL". See
 * storage.php for storage backend implementations. Parameters for
 * connecting to the storage backend. See storage.php if you want to
 * create your own backend.
 */
define('STORAGE_BACKEND', 'MYSQL');
global $storage_parameters;
```

```
$storage_parameters = array('username' => 'idpUser',
                             'password' => 'idpPassword',
                             'database' => 'idpDB',
                             'hostspec' => 'localhost');

/**
 * Authentication backend for authentication queries. Default (and
 * only) choice is "MYSQL". See auth.php for backend implementations
 * if you want to create your own. This default setting just puts the
 * authentication data in the same database with the storage data
 * (above), so you probably don't need to adjust this.
 */
define('AUTH_BACKEND', 'MYSQL');
global $storage_parameters;
$storage_parameters = array('username' => 'idpUser',
                             'password' => 'idpPassword',
                             'database' => 'idpDB',
                             'hostspec' => 'localhost');

/**
 * Authentication backend for authentication queries. Default (and
 * only) choice is "MYSQL". See auth.php for backend implementations
 * if you want to create your own. This default setting just puts the
 * authentication data in the same database with the storage data
 * (above), so you probably don't need to adjust this.
 */
define('AUTH_BACKEND', 'MYSQL');
global $auth_parameters;
$auth_parameters = $storage_parameters;

?>
```

5.1.7 Testing Server

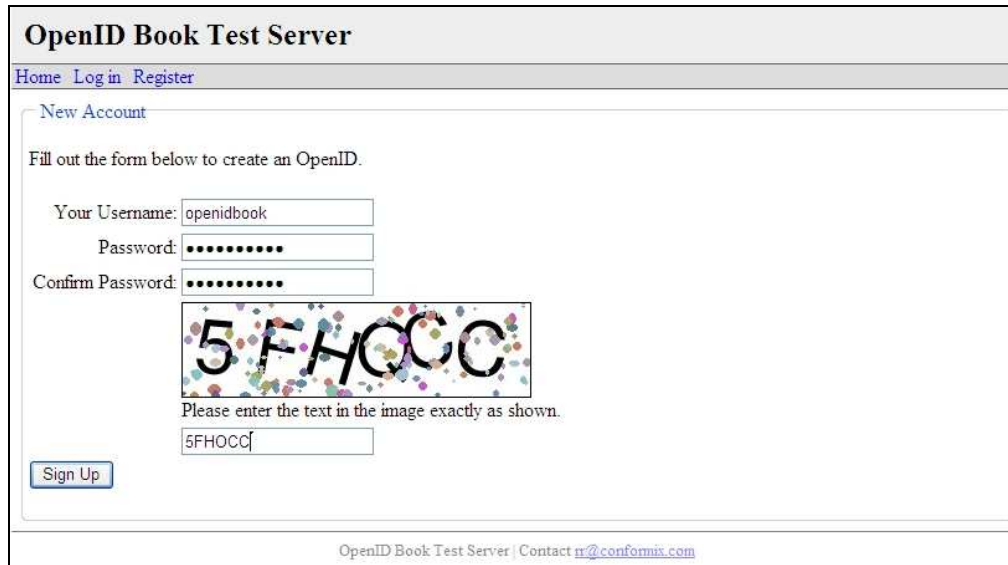
Once you have configured the OpenID server, you will test it using your browser. When you visit the server URL for the first time, you will see

something as shown in Figure 5.2. Here you get the basic information about how to edit the main page using template. Also note that contact email address and the server name are present in the footer of the page. These items were configured in config.php file.



Figure 5-2: The welcome screen for the OpenID server.

In Figure 5.2, you can see some links at the top of the web page. The main links are “Login” and “Register”. The first step would be to register a new user with the server. When the account is created, the user will be assigned an OpenID URL that can be used with any OpenID Consumer. When you click on “Register”, you will see the screen shown in Figure 5.3 next.



The screenshot shows a web browser window with the title "OpenID Book Test Server". Below the title is a navigation bar with links "Home", "Log in", and "Register". The main content area is titled "New Account" and contains the following elements:

- A heading "New Account" with a small blue arrow icon.
- A paragraph: "Fill out the form below to create an OpenID."
- A form with three input fields:
 - "Your Username:" with the value "openidbook" entered.
 - "Password:" with masked characters "*****".
 - "Confirm Password:" with masked characters "*****".
- A CAPTCHA image showing the text "5FHQCC" overlaid on a background of colorful dots.
- A text prompt: "Please enter the text in the image exactly as shown."
- A text input field containing the CAPTCHA text "5FHQCC".
- A "Sign Up" button.

At the bottom of the page, there is a footer that reads: "OpenID Book Test Server | Contact rr@conformix.com".

Figure 5-3: New user registration with the OpenID server.

Note that in Figure 5-3, the registration process is very simple, you just enter your user name, password and the CAPTCHA character string shown in the image. Note that CAPTCHA is used to distinguish a real person from automated scripts to stop attackers from creating large number of users using scripting attacks.

Here we have registered as a user “openidbook”. This user account will be used to manage our profile on this server. Note that this username or password will never be sent to a Consumer web site.

Once the user registration process is complete, you will see a web page something like shown in Figure 5-xxx. Everything is done and your OpenID is ready for use. Your OpenID URL is:

`http://idp.conformix.com/?user=openidbook`

You can immediately start using this URL. However, if you chose to create your own URL and use this server only for authentication purposes, you will create an HTML page on your own URL with the following two lines in the HEAD part of that HTML document.

```
<link rel="openid.server" href=
"http://idp.conformix.com/index.php?serve">
<link rel="openid.delegate" href=
"http://idp.conformix.com/?user=openidbook">
```

The first line will tell the Consumer the location of the OpenID server and the second line will tell Consumer the OpenID that it needs to verify using that server. This is the delegation process as you already know. The delegation mechanism helps you keep the same ID when you change your OpenID provider. For example, if you start using a new OpenID server for some reason (Your OpenID provider goes out of business or their server is down for some reason), you can keep your OpenID URL. If you don't use delegation, your new OpenID URL will be considered a different person by Consumers and you may lose access to your old accounts.

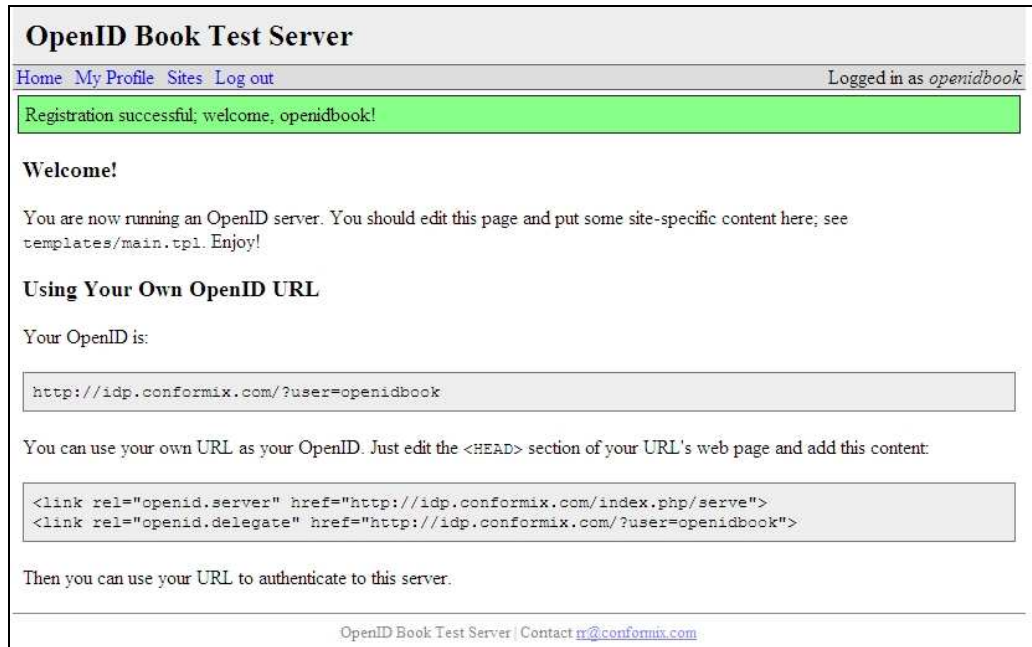


Figure 5.4: Completion of your OpenID user registration process.

If you try to load the newly created OpenID URL (<http://idp.conformix.com/?user=openidbook>) in your browser, you will see a short message showing that this URL is identity URL. However, if you view the “source” of the page displayed, it will be something as follows:

```
<html>
  <head>
    <link rel="openid.server"
href="http://idp.conformix.com/index.php/serve">
    <link rel="openid.delegate"
href="http://idp.conformix.com/?user=openidbook">
  </head>
  <body>
    <h3>OpenID Identity Page</h3>

    <p>
```

```
    This is the identity page for the user <strong>openidbook</strong>.  
  </p>  
</body>  
</html>
```

Note the two lines starting with “<link” (and shown in boldface) in the HEAD section of the HTML page. These two lines show the *server* and the *delegate* parts.

Now that you have created your OpenID on this server, it is the time to set some parameters on this server. This is your profile, as shown in Figure 5-5. Note that you can choose not to put any information in your profile or populate all fields. When a Consumer requests your profile from the OpenID server, this information may be delivered. It is up to the Consumer how it wants to use this profile. For example, a Consumer may not grant you access if your nickname and full name are not present in your profile. Other consumers may be interested only in the authentication and may not care at all about optional information in your profile.

Some Consumers may use information in your profile to grant you a certain level of access (graded and risk based authentication). For example, a Consumer may allow you to make a payment on your insurance policy if postal code, birth date and email address are present and match your record. If they don't, it may allow you to only view (read-only) very basic information about your insurance policy.

Please also note that servers can implement different profiles. Some servers will allow you creating multiple profiles (e.g. pip.verisignlabs.com). Some servers will even allow you to create custom profile such that you can create your own fields in the profile. OpenID protocol is very open and flexible about how profiles are handled. In Chapter One, an example was provided about how to create OpenID profiles.

OpenID Book Test Server

Home My Profile Sites Log out Logged in as *openidbook*

My Profile

This is your account profile. Values you enter here can be sent to sites that support OpenID Simple Registration. When you authenticate to such a site, you'll be asked for permission to transmit this information. All fields are optional!

Nickname:

Full Name:

E-mail address:

Birth date:

Postal code:

Gender:

Country:

Time zone:

Preferred language:

OpenID Book Test Server | Contact rr@conformix.com

Figure 5-5: Creating OpenID profiles

The OpenID Consumer can request these additional parameters during the authentication process as you will see later in this chapter. A Consumer can request as many parameters it wants but it is up to the server about which parameters it wants to return. This gives you, the real person, control over which information you want to send to a Consumer.

5.1.8 Testing Consumer with the Server

Now that you have created your OpenID URL, it can be tested using the sample Consumer that you used in the previous chapter. The sample Consumer is present at <http://consumer.conformix.com> and when you go to this Consumer web site, you will see the familiar screen as shown in Figure 5-6. Here you will use your newly created Identity URL to verify that the server is working properly. Note that your identity URL is

<http://idp.conformix.com/?user=openidbook>. After entering this URL in the box, you will click on the “Verify” button.

PHP OpenID Authentication Example

This example consumer uses the [PHP OpenID](#) library. It just verifies that the URL that you enter is your identity URL.

Identity URL:

Figure 5-6: Using the sample Consumer application to test new OpenID URL.

Once you click on the “Verify” button, the browser will show you the screen shown in Figure 5-7 where you will chose if you want to allow this Consumer once or forever. For this sake of this discussion, click on the “Allow Once” button. Note that if you are not already logged into the OpenID server, you may also be asked to login first (before you see this in browser).

OpenID Book Test Server

[Home](#) [My Profile](#) [Sites](#) [Log out](#) Logged in as *openidbook*

Do you wish to confirm your identity URL (<http://idp.conformix.com/?user=openidbook>) with <http://consumer.conformix.com:80/?>

The server has also requested the transfer of profile information. The required and optional fields are listed below. Uncheck the “Send?” checkbox for fields that you don't want to send.

Send?	Name	Value	Status
<input checked="" type="checkbox"/>	E-mail address	rr@conformix.com	Optional

The server supplied no data policy URL.

OpenID Book Test Server | Contact rr@conformix.com

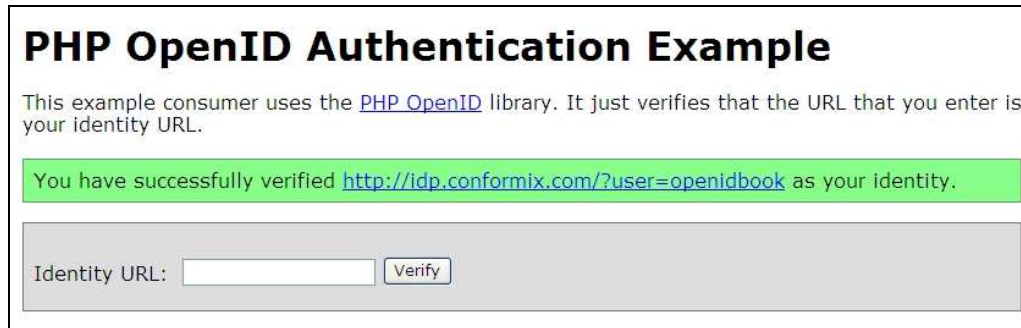
Figure 5-7: List of optional parameters requested by the Consumer

The important thing to note about Figure 5-7 is the list of parameters in the profile. You should observe the following things related to this screen:

- The server is requesting certain parameters in your profile. List of these parameters will be shown here. The OpenID server is telling you which parameters are being requested by the Consumer and giving you a choice to decide which parameters should be sent to the Consumer.
- In the first column, if you uncheck the checkbox, that parameter will not be sent back to Consumer.
- The second column shows the name of the parameter. In this figure, you can see that the Consumer is asking for Email address.
- The third column shows the value of the parameter. This value is taken from the profile that you have already set on OpenID server. The value is shown here so that you, as OpenID owner, can see what is being sent back.
- The fourth column shows the status of the parameter request. The “Optional” here means that the Consumer is requesting this parameter but leaving it up to you to send it or not. If a parameter is not optional, the Consumer may decide not to authenticate you because you did not send the parameters requested. A consumer may also give you a lower level of access. However, it is up to the Consumer to take whatever action it wants to take, OpenID does not dictate specific actions.

These optional parameters are part of the OpenID protocol extensions call “Simple Registration” and you will find detailed information about this and other extensions later in this book.

Once you have clicked on “Allow Once” button, the authentication will be complete and you will see success result as shown in Figure 5-8.



PHP OpenID Authentication Example

This example consumer uses the [PHP OpenID](#) library. It just verifies that the URL that you enter is your identity URL.

You have successfully verified <http://idp.conformix.com/?user=openidbook> as your identity.

Identity URL:

Figure 5-8: Confirmation of OpenID authentication.

If authentication fails, you may see a failure message as well in Figure 5-8. Note that the sample Consumer shows only success or failure. A real Consumer will do some other tasks based upon results of authentication success or failure.

If you select a web site to be “Trusted Forever” such that you don’t have to authorize the use of OpenID every time the Consumer requests an authentication, the server and consumer will keep a shared secret. In that case, the server will show this site as trusted (elaborate this wording) as shown in Figure 5-9. This is shown when you click on “Sites” link on the top bar of the server.



OpenID Book Test Server

[Home](#) [My Profile](#) [Sites](#) [Log out](#) Logged in as *openidbook*

[Sites](#)
This page lists past trust decisions you have made when logging into sites using your OpenID.

Site	Status
<input type="checkbox"/> http://consumer.conformix.com:80/	Trusted

OpenID Book Test Server | Contact tr@conformix.com

Figure 5-9: Trusted web site.

Note that once a web site becomes trusted, the OpenID server will authenticate you with the web site without prompting you for optional parameters.

Note that you can also remove a previously trusted web site from the list. Once you remove it, it becomes un-trusted once again.

You can use the “Deny” button as shown in Figure 5-9 to remove a Consumer web site from trusted list.

5.1.9 Database Changes during Server Configuration

During the server installation, you created a database where the server could store its information. This section provides information about this database in a little more detail. I would like to emphasize that a reader should take this database as an example. Different server implementations may have different database schemas. They also may chose to store a different set of information than listed here. The database should still provide a good foundation about the interworking of an OpenID server and how information is stored and used.

First of all, let us have a look at the database tables. All of these tables are created by the server itself using the information provided in the config.php file. The database specific information in the config.php file includes:

- The database name
- The hostname where database is stored.
- The username and password to access the database

Using this information, the following tables are automatically created by the OpenID server. We are using “show tables” command to display list of all tables. Note that to fully understand this section, you should have some knowledge of MySQL database commands.

```
mysql> show tables;
```



```
+-----+
| Tables_in_idpDB |
+-----+
| accounts          |
| identities         |
| oid_associations  |
| oid_nonces        |
| oid_settings      |
| personas          |
| personas_id_seq    |
| sites             |
+-----+
8 rows in set (0.00 sec)
```

```
mysql>
```

Different pieces of information are stored in different tables. Let us look at all of these tables one by one. First of all, the *accounts* table stores all user accounts. The *id* field is just an identifier number. The *username* field shows the login name for a user. Passwords are stored in hashed form. The following command shows two users and their passwords in hashed form.

This table is used to login to the OpenID server. Once a user is logged in, he/she can carry out other tasks related to ID management.

```
mysql> select * from accounts;
+-----+-----+-----+
| id | username    | password                                     |
+-----+-----+-----+
| 1  | rurehman    | 1c0ada6386ca9d746a486bb27af10228 |
| 2  | openidbook  | 4701125c06b2ba8ea0cfe58909ccff01 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

The following command shows all identities stored in the database. Each identity corresponds to a username in the *accounts* table.

When a Consumer requests for authentication and sends a URL, this is the table where the server will look into to find which user owns that URL. Once the server know the owner of the URL, it will check if a site is trusted or not and take different actions accordingly.

```
mysql> select * from identities;
+-----+-----+-----+
-----+
| id | account      | url
|
+-----+-----+-----+
-----+
| 1 | rurehman     | http://idp.conformix.com/?user=rurehman
|
| 2 | openidbook   | http://idp.conformix.com/?user=openidbook
|
+-----+-----+-----+
-----+
2 rows in set (0.00 sec)

mysql>
```

The *oid_associations* table holds shared secrets with different Consumers. In addition to shared key, it also stores the association type, association handle, timestamp when the shared secret was created and a lifetime for the shared secret. Once the lifetime of the shared secret is over, a new association will be established between a Consumer and OpenID Server. The following tables shows two entries in this table.

```
mysql> select * from oid_associations;
+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
```

```
| server_url          | handle          | secret
| issued            | lifetime | assoc_type |
+-----+-----+-----+-----+
-----+-----+-----+-----+
| http://localhost/ | normal | {HMAC-SHA1}{460709e5}{/Qw2uA==} | e @ |
1174866405 | 1209600 | HMAC-SHA1 |
| http://localhost/ | normal | {HMAC-SHA1}{46071e25}{Tt8MwQ==} | |
1174871589 | 1209600 | HMAC-SHA1 |
+-----+-----+-----+-----+
-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

```
mysql>
mysql> select * from oid_nonces;
Empty set (0.00 sec)

mysql>
```

```
mysql> select * from oid_settings;
Empty set (0.00 sec)

mysql>
```

The *personas* table stores profile for users. The following listing shows two user profiles. Note that optionally you can set you profiles and each element in a profile can be used to fulfill requests from a Consumer web site during the authentication process.

```
mysql> select * from personas;
+---+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
```

```

| id | account      | nickname | email              | fullname      | dob
| gender | postcode | country | language | timezone |
+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+
| 1 | rurehman    | rr       | rr@conformix.com | Rafeeq Rehman | 0000-
00-00 |          |          |          |          |
| 2 | openidbook | rr       | rr@conformix.com | Rafeeq Rehman | 0000-
00-00 |          |          |          | EN          |
+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

The *sites* table keeps record of trusted or non trusted Consumer web sites. The following list shows two Consumer web sites and both web sites are not trusted.

Note that a Consumer web site becomes “trusted” when you click on “Allow Forever” button in Figure 5-7.

```

mysql> select * from sites;
+-----+-----+-----+-----+-----+-----+
-----+
| account      | trust_root |
trusted |
+-----+-----+-----+-----+-----+-----+
-----+
| openidbook | http://idp.conformix.com:80/examples/consumer |
0 |
| openidbook | http://consumer.conformix.com:80/consumer |
0 |
+-----+-----+-----+-----+-----+-----+
-----+
2 rows in set (0.01 sec)

mysql>

```

5.1.10 Database Table Description

In this section, the *describe* MySQL command output is shown so that you can get a quick overview of the type of information that goes into different tables. All user commands are shown in boldface.

If you are implementing your own OpenID server, you can create your own database schema and decide which information to store.

```
mysql> describe accounts;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type           | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)        |      | PRI | NULL    | auto_increment |
| username   | varchar(255)   | YES  | MUL | NULL    |                 |
| password   | varchar(32)    | YES  |      | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql>
```

```
mysql> describe identities;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type           | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)        |      | PRI | NULL    | auto_increment |
| account    | varchar(255)   |      | MUL |          |                 |
| url        | text           |      |      |          |                 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql>
```

```
mysql> describe oid_associations;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type           | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
```

```
| server_url | blob          |          | PRI |          |          |
| handle    | varchar(255) |          | PRI |          |          |
| secret    | blob         | YES      |     | NULL      |          |
| issued    | int(11)      | YES      |     | NULL      |          |
| lifetime  | int(11)      | YES      |     | NULL      |          |
| assoc_type | varchar(64)  | YES      |     | NULL      |          |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

```
mysql>
mysql> describe oid_nonces;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nonce | char(8) |      | PRI |          |       |
| expires | int(11) | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql>
mysql> describe oid_settings;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| setting | varchar(128) |      | PRI |          |       |
| value   | blob         | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql>
mysql> describe personas;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)       |      | PRI | NULL     | auto_increment |
| account | varchar(255) |      |     |          |               |
| nickname | varchar(255) | YES  |     | NULL     |               |
| email  | varchar(255) | YES  |     | NULL     |               |
+-----+-----+-----+-----+-----+-----+
```

```
| fullname | varchar(255) | YES | | NULL | |
| dob      | date         | YES | | NULL | |
| gender   | char(1)      | YES | | NULL | |
| postcode | varchar(255) | YES | | NULL | |
| country  | varchar(32)  | YES | | NULL | |
| language | varchar(32)  | YES | | NULL | |
| timezone | varchar(255) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

```
mysql>
mysql> describe personas_id_seq;
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | int(10) unsigned    |      | PRI | NULL     | auto_increment |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql>
mysql> describe sites;
+-----+-----+-----+-----+-----+-----+
| Field      | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| account    | varchar(255)        |      | MUL |          |                |
| trust_root | text                | YES  |     | NULL     |                |
| trusted    | tinyint(1)          | YES  |     | NULL     |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql>
```

From these tables, you have an idea about data types used for different fields and length of these fields.

5.2 Deep Dive Into OpenID Protocol: Dumb Mode

Having a good understanding of the OpenID server and the database used behind it, the next step is to have an in-depth understanding of different OpenID protocol messages. For this purpose, I have used Wireshark (<http://www.wireshark.org>) as packet sniffer to capture real data on the network. After filtering this data for relevant messages, this section will go into detail of these messages and how these are exchanged. This section will also provide good information for troubleshooting of the protocol problem when you run your own server. Note that only HTTP part of the packet is shown and all other information is stripped off. I have also “decoded” HTTP characters to show the protocol information more clearly.

Note that I have used HTTP protocol on port 80. This was to capture packets without any encryption. It is strongly recommended to use HTTPS (port 443) for all OpenID transactions to encrypt the transport layer.

5.2.1 Yadis and XRD Document

As mentioned in earlier chapters, Yadis is used as a first step by Consumers to discover services. The following packet shows a request going from the Consumer to the OpenID Server. The request was initiated with OpenID URL <http://idp.conformix.com/?user=openidbook>

```
GET /?user=openidbook HTTP/1.1Host: idp.conformix.com Accept:
application/xrds+xml
```

When the OpenID Server received this request, it returned an XRD document. XRD is an XML document that shows services provided by the server. The response to the GET request is as follows:

```
HTTP/1.1 200 OK
Date: Mon, 26 Mar 2007 01:14:38 GMT
Server: Apache/2.0.54 (Fedora)
X-Powered-By: PHP/5.0.4
```


Set-Cookie: PHPSESSID=70rdu7mmk9nljmh7acoa7pud84
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 482
Connection: close
Content-Type: application/xrds+xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS
  xmlns:xrds="xri://$xrds"
  xmlns:openid="http://openid.net/xmlns/1.0"
  xmlns="xri://$xrd*($v*2.0)">
  <XRD>
    <Service>
      <Type>http://openid.net/signon/1.1</Type>
      <Type>http://openid.net/sreg/1.0</Type>
      <URI>http://idp.conformix.com/index.php/serve</URI>

    <openid:Delegate>http://idp.conformix.com/?user=openidbook</openid:Dele
gate>
    </Service>
  </XRD>
</xrds:XRDS>
```

You should note the following about this XML document.

- The document starts with XRDS node.
- There may be multiple “Service” elements in the document inside the XRD node.
- There may be multiple “Type” elements in this document inside each “Service” element. Note that there are two services listed in this document. The first line shows the Authentication (signon) service, while the second line shows Simple Registration (sreg) service. You will

get more information about the Simple Registration service in Chapter 6.

-

Note that if XRD discovery fails, then the Consumer will use the HTML based discovery using HEAD part of the retrieved document.

5.2.2 Indirect communication between Consumer and Identity Provider

Once this information is found from `idp.conformix.com` by the Consumer, the Consumer will analyze it using Yadis to determine if the Server can provide OpenID service. After the Consumer has analyzed the XRD document and knows that the server can serve the OpenID requests, it sends the following request to the Server using Browser redirection. Note that HTTP 302 message is used for redirection as shown in the sniffer output next. Figure 5-2 shows the direction of request and response messages and message types.

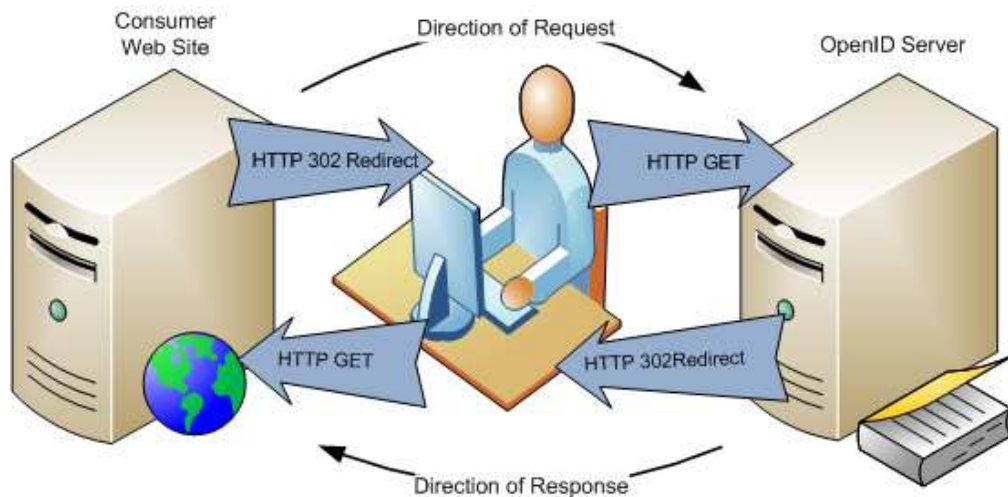


Figure 5-9: Request and response messages between IdP and Consumer

This request has a number of parameters in it. Have a look on the actual request first and then we shall discuss these parameters. Note that after capturing it using Wireshark, it has been decoded for clarity purposes (encoded HTTP requests have a number of HEX encoded character with % sign in front of them).

```
HTTP/1.1 302 Found
Date: Mon, 26 Mar 2007 01:16:42 GMT
Server: Apache/2.2.3 (Fedora)
X-Powered-By: PHP/5.1.6
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Location:
http://idp.conformix.com/index.php/serve?openid.assoc_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?user=openidbook&openid.mode=checkid_setup&openid.return_to=http://consumer.conformix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sreg.optional=email&openid.trust_root=http://consumer.conformix.com:80/
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

You should note the following things about the above request. The first thing to note is that it is a redirect message to the web browser.

- It has a redirection URL that enables the web browser to forward the request to the OpenID server. This URL in the above request is shown after keyword “Location:” and is `http://idp.conformix.com/index.php/serve`. Note that I have removed all of the query string from this URL. The query string actually contains
- The parameters are passed as GET request (using query string), so it is important to have the OpenID URL with HTTPS protocol, instead of plain HTTP.

- Names of all parameters are preceded by keyword `openid` which is part of OpenID specifications version 2.0. This helps in identifying an OpenID protocol parameter from any other parameter.
- The first parameter, `openid.assoc_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}`, shows the association handle and encryption method used.
- The second parameter, `openid.identity=http://idp.conformix.com/?user=openidbook`, shows the identity that the Consumer is trying to validate.
- The third parameter, `openid.mode=checkid_setup`, shows the message type.
- The fourth parameter, `openid.return_to=http://consumer.conformix.com:80/finish_auth.php`, shows the URL where the OpenID server should redirect the browser back after validating the ID. Note when the server replies back, it will use this URL with some additional query string parameters.
- The fifth parameter, `nonce=nC5sKquX` is used to prevent relay attacks. It has a timestamp encoded in it.
- The sixth parameter, `openid.sreg.optional=email`, is used to request an optional item from the server: the email address. Note that a Consumer can request any additional items. The OpenID server may decide which of these to send back or not.
- The seventh parameter, `openid.trust_root=http://consumer.conformix.com:80/`, shows the URL for the Consumer web site. In case the ID owner wants to mark this

web site as trusted one for future authentication, this URL can be saved by the OpenID server.

Once browser has received the redirect message from the Consumer, it sends HTTP GET message to idp.conformix.com (IdP). This message is as shown below. Note that the “GET” part of this message is the same as the “Location” part in the previous listing.

```
GET /index.php/serve?openid.assoc_handle={HMAC-
SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=checkid_setup&openid.return_to=http://consumer
.conformix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sreg.optional=e
mail&openid.trust_root=http://consumer.conformix.com:80/ HTTP/1.1
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*
```

Referer:

```
http://consumer.conformix.com/finish_auth.php?nonce=ZwgGkLwy&openid.ass
oc_handle={HMAC-
SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=id_res&openid.return_to=http://consumer.confor
mix.com:80/finish_auth.php?nonce=ZwgGkLwy&openid.sig=bLXPr2BJtj4M/vVGyh
Y0uZjvoHQ=&openid.signed=mode,identity,return_to,sreg.email&openid.sreg
.email=rr@conformix.com
```

Accept-Language: en-us

UA-CPU: x86

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727)

Cookie: PHPSESSID=94tee0m8oflcsvf49rkv0l0tu1;

PHPSESSID=94tee0m8oflcsvf49rkv0l0tu1

Connection: Keep-Alive

Host: idp.conformix.com

5.2.3 Identity Provider Asks User for Authentication to IdP

Note that the Identity Provider may ask the End User to authenticate before it replies back to the Consumer with an assertion. Depending upon implementation, there may be multiple mechanisms to accomplish this. In the following packet capture, the Identity Provider simply redirects the browser to verify the trust relationship. There may be a login page or some other mechanism as well.

The following is redirect message from the Identity Provider to the web browser. Note that Identity Provider is redirecting the browser to one of its own pages.

```
HTTP/1.1 302 Found
Date: Mon, 26 Mar 2007 01:14:38 GMT
Server: Apache/2.0.54 (Fedora)
X-Powered-By: PHP/5.0.4
Set-Cookie: PHPSESSID=94tee0m8oflcsvf49rkv0l0tu1
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Location: http://idp.conformix.com/?action=trust
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

The following packet capture shows the web Browser redirect back to Identity Provider. Basically it is sending back a Cookie. (Verify it)

```
GET /?action=trust HTTP/1.1
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*
Referer:
http://consumer.conformix.com/finish_auth.php?nonce=ZwgGkLwy&openid.assoc_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=id_res&openid.return_to=http://consumer.confor
mix.com:80/finish_auth.php?nonce=ZwgGkLwy&openid.sig=bLXPr2BJtj4M/vVGyh
Y0uZjvoHQ=&openid.signed=mode,identity,return_to,sreg.email&openid.sreg
.email=rr@conformix.com
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727)
Cookie: PHPSESSID=94tee0m8oflcsvf49rkv0l0tul
Connection: Keep-Alive
Host: idp.conformix.com
```

5.2.4 Identity Provider's Positive Assertion to Consumer

After the End User's authentication to the Identity Provider is established, the IdP sends the following packet back to Browser to redirect is back to the Consumer web site. This packet contains a successful assertion for the Consumer.

```
HTTP/1.1 302 Found
Date: Mon, 26 Mar 2007 01:14:40 GMT
Server: Apache/2.0.54 (Fedora)
X-Powered-By: PHP/5.0.4
Set-Cookie: PHPSESSID=94tee0m8oflcsvf49rkv0l0tul
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-
check=0
Pragma: no-cache
```

```
location:
http://consumer.conformix.com:80/finish_auth.php?nonce=nC5sKquX&openid.
assoc_handle={HMAC-
SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=id_res&openid.return_to=http://consumer.confor
mix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sig=nXWc+07GLaSf+RghmG
ubGPPglZc=&openid.signed=mode,identity,return_to,sreg.email&openid.sreg
.email=rr@conformix.com
Connection: close
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

You should note the following about this packet.

- The “*location:http://consumer.conformix.com:80/finish_auth.php*” part directs the browser where it should send the redirection. There are a number of query string parameters that we shall discuss next.
- The “*nonce=nC5sKquX*” parameter shows the nonce value discussed earlier and it is used to prevent relay attacks.
- The “*openid.assoc_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}*” shows the association handle. If the Consumer already has an association established, it can use this handle to verify the authenticity of this message.
- The “*openid.identity=http://idp.conformix.com/?user=openidbook*” parameter shows the identity that is being established using this protocol.
- The “*openid.mode=id_res*” shows the message type as discussed in Chapter 3.

- The “`openid.return_to=http://consumer.conformix.com:80/finish_auth.php`” is a copy of the original parameter that was present in the request.
- The “`openid.sig=nXWc+o7GLaSf+RghmGubGPPglZc=`” shows the digital signature.
- The “`openid.signed=mode,identity,return_to`” shows which parameters are signed when calculating value of the `openid.sig` parameter.
- The “`openid.sreg.email=rr@conformix.com`” shows the optional parameter email parameter that was requested by the Consumer.

After receiving this redirect message from the Identity Provider, the web browser sends the following message to the Consumer which completes indirect communication.

```
GET /finish_auth.php?nonce=nC5sKquX&openid.assoc_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=id_res&openid.return_to=http://consumer.confor
mix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sig=nXWc+o7GLaSf+RghmG
ubGPPglZc=&openid.signed=mode,identity,return_to,sreg.email&openid.sreg
.email=rr@conformix.com HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*
Referer: http://idp.conformix.com/?action=trust
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727)
Host: consumer.conformix.com
Cookie: PHPSESSID=og3ln8j9bcr157qgn7a97nm0i0
Connection: Keep-Alive
Cache-Control: no-cache
```

5.2.5 Verification between Consumer and Identity Provider using check_authentication Message

The Consumer has to verify this assertion from the Identity Provider. The mechanism to verify the assertion is the check_authentication message. The Consumer sends all parameters that it received from the Identity Provider via checkid_setup or checkid_immediate messages. This is a direct, out of band, communication between the Consumer and the Identity Provider. For all direct communication, HTTP POST method is used. Following is a typical check_authentication message.

```
POST /index.php/serve HTTP/1.1
Host: idp.conformix.com
Accept: */*
Content-Length: 519
Content-Type: application/x-www-form-urlencoded

openid.assoc_handle={HMAC-SHA1}{460730e1}{zrlgKg==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.invalidate_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}&openid.mode=check_authentication&openid.retur
n_to=http://consumer.conformix.com:80/finish_auth.php?nonce=mAotRbGM&op
eid.sig=4hwwyWbPtSAmP2dYxEC+dq6050s=&openid.signed=mode,identity,retur
n_to,sreg.email&openid.sreg.email=rr@conformix.com
```

Once the Identity provider has received this message, it will determine the validity of the assertion and reply back with a “yes” or “no” answer as shown below. Note that the “is_valid” parameter at the end of the following output shows the success or failure of an assertion.

```
HTTP/1.1 200 OK
Date: Mon, 26 Mar 2007 02:33:05 GMT
Server: Apache/2.0.54 (Fedora)
X-Powered-By: PHP/5.0.4
```

```
Set-Cookie: PHPSESSID=9mpri62iipjojam625rh0kalt2
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-
check=0
Pragma: no-cache
Connection: close
Content-Length: 64
Content-Type: text/html; charset=UTF-8

invalidate_handle:{HMAC-SHA1}{46071e25}{Tt8MwQ==}
is_valid:true
```

Once the Consumer has received this success message, the authentication process is complete. The consumer can then allow the End User to login to the Consumer web site. The Consumer can also take other actions. For example, if the Identity Provider did not return all of the parameters that the Consumer had requested, it may grant lesser level of access to data. If a user has logged into the Consumer web site the first time, the Consumer may also go through a registration process. The bottom line is that the Consumer actions after the authentication process is complete are out of the scope of the OpenID protocol. Depending upon how the Consumer web site is programmed, you may see different behavior.

5.2.6 Authentication Completion

In the sample Consumer web site that we have used for the purpose of testing the OpenID server, the Consumer does not do anything and it just displays a success message in the Web Browser. Following is the HTML code of this page which is included here only for the sake of completeness; otherwise there is nothing special about this.

```
HTTP/1.1 200 OK Date: Mon, 26 Mar 2007 01:16:45 GMT Server:
Apache/2.2.3 (Fedora)X-Powered-By: PHP/5.1.6Expires: Thu, 19 Nov 1981
08:52:00 GMT Cache-Control: no-store, no-cache, must-revalidate, post-
check=0, pre-check=0Pragma: no-cache Content-Length: 1610Connection:
close Content-Type: text/html; charset=UTF-8
```

```
<html>
  <head><title>PHP OpenID Authentication Example</title></head>
  <style type="text/css">
    * {
      font-family: verdana,sans-serif;
    }
    body {
      width: 50em;
      margin: 1em;
    }
    div {
      padding: .5em;
    }
    table {
      margin: none;
      padding: none;
    }
    .alert {
      border: 1px solid #e7dc2b;
      background: #fff888;
    }
    .success {
      border: 1px solid #669966;
      background: #88ff88;
    }
    .error {
      border: 1px solid #ff0000;
      background: #ffaana;
    }
    #verify-form {
      border: 1px solid #777777;
      background: #dddddd;
      margin-top: 1em;
      padding-bottom: 0em;
    }
  </style>
  <body>
    <h1>PHP OpenID Authentication Example</h1>
```

```
<p>
  This example consumer uses the <a
  href="http://www.openidenabled.com/openid/libraries/php/">PHP
  OpenID</a> library. It just verifies that the URL that you enter
  is your identity URL.
</p>

  <div class="success">You have successfully verified <a
  href="http://idp.conformix.com/?user=openidbook">http://idp.conformix.c
  om/?user=openidbook</a> as your identity. You also returned
  'rr@conformix.com' as your email.</div>
  <div id="verify-form">
    <form method="get" action="try_auth.php">
      Identity&nbsp;URL:
      <input type="hidden" name="action" value="verify" />
      <input type="text" name="openid_url" value="" />
      <input type="submit" value="Verify" />
    </form>
  </div>
</body>
</html>
```

A more realistic web site will show something different than this page.

5.3 OpenID Association Messages

All messages discussed in the previous section used Dumb mode of communication. In Smart mode of communication, association is established between the Consumer web site and the Identity Provider. This section shows the process of establishing this association.

The association process is initiated by the Consumer using a direct communication message called “associate”. This is HTTP POST method with a number of parameters. These parameters were discussed in Chapter 3 and here you will see a real example about how these parameters are used for establishing association.

Following example shows an “association request” request message going from the Consumer web site to the Identity Provider.

```
POST /index.php/serve HTTP/1.1
Host: idp.conformix.com
Accept: */*
Content-Length: 504
Content-Type: application/x-www-form-urlencoded

openid.mode=associate&openid.assoc_type=HMAC-
SHA1&openid.session_type=DH-
SHA1&openid.dh_consumer_public=KC6IpA00A6SlCikafFSlrTGql9H8+de6GFi5YlKz
4pyDxUMS5Z8pMOM/PtrlgFmCcgAXjFbuxS73ZutDTFJYpADoIntFVrah9eaezMcw6SDR24c
nFjNc14xq0zGt3QcRLXaNTRVKfMW8evDAmLCrvEhU5c7B3eqmk+bMMrbQpcE=&openid.dh
_modulus=ANz5OguIOXLsDhmYmsWizjEOHTdxfo2Vcbt2I3MYZuYe9louJ4mLBX+YkcLiem
OcPym2CBRYHNOyyjmG0mg3BVd9RcLn5S3IHhoxGHblzqdLFEi/368Ygo79JRnxTkXjgmY0r
xlJ5bU1zIKaSDuKdiI+XUkKJX8Fvf8W8vsixYOR&openid.dh_gen=Ag==
```

The above message contains the following parameters.

- The “openid.mode=associate” shows that this is an association request message.
- The “openid.assoc_type=HMAC-SHA1” shows that the Consumer is asking for HMAC-SHA1 type of association. If the OpenID Server does not support this type of association, the association may fail.
- The “openid.session_type=DH-SHA1” shows the Diffie-Hellman key exchange mechanism being used for this association. Again if the OpenID Server does not support this, the association may fail.
- The consumer public key is rather a long string which is shown above as “openid.dh_consumer_public=KC6IpA00A6SlCikafFSlrTGql9H8+de6GFi5YlKz4pyDxUMS5Z8pMOM/PtrlgFmCcgAXjFbuxS73ZutDTFJYpADoIntFVrah9eaezMcw6SDR24cnFjNc14xq0zGt3QcRLXaNTRVKfMW8evDAmLCrvEhU5c7B3eqmk+bMMrbQpcE=”

- The modulus is shown as “openid.dh_modulus=ANz5OguIOXLsDhmYmsWizjEOHTdxfo2Vcbt2I3MYZuYe91ouJ4mLBX+YkcLiemOcPym2CBRYHNOyyjmGomg3BVD9RcLn5S3IHHoXGHblzqdLFEi/368Ygo79JRnxTkXjgmYorxIJ5bU1zIKaS DuKdiI+XUkKJX8Fvf8W8vsixYOr”

Note that both the Consumer and the OpenID server must have a mechanism to store the association handle and related parameters in some way. Most of the times, this is a database or a file residing on a file system.

Following is the association response from the Identity Provider to the Consumer web site. Note that it contains the association handle.

```
HTTP/1.1 200 OK Date: Mon, 26 Mar 2007 02:47:37 GMT Server:
Apache/2.0.54 (Fedora)X-Powered-By: PHP/5.0.4Set-Cookie:
PHPSESSID=f37bm9q0o8n2ahq2qupuk6q9s6Expires: Thu, 19 Nov 1981 08:52:00
GMT Cache-Control: no-store, no-cache, must-revalidate, post-check=0,
pre-check=0Pragma: no-cache Connection: close Content-Length:
337Content-Type: text/html; charset=UTF-8
assoc_handle:{HMAC-SHA1}{4607344a}{oDFF0g==}
assoc_type:HMAC-SHA1
dh_server_public:AIPkx6xJ3b1Wnr1o1WL7suoZnABDc+lJRR9DeNIBolGXQX3W2e+4ud
Y2p+dUcF5jKE6uoZuXLPbimHbndBOYhUDUfkKaAjQtVvONerAjd5RHyT2i2AoYrkjD26tr
aC4jzg7NukZlmrRjfpRg4q3gwW+EZEXvz+ba9JnQfsXx+iH
enc_mac_key:UtQHBswQimAZAp4s/9sfSQSpuq0=
expires_in:1209600
session_type:DH-SHA1
```

You should note the following important things about the above listing, which is the association response from the Identity Provider.

- The response include association handle that will be used for future communication between Consumer and Identity Provider. This handle is “assoc_handle:{HMAC-SHA1}{4607344a}{oDFF0g==}”.

- The message also includes the OpenID server's public key which is used in Diffie-Hellman key exchange method. This is "dh_server_public:AIPkx6xJ3b1Wnr1olWL7suoZnABDc+lJRR9DeNIBo lGXQX3W2e+4udY2p+dUcF5jKE6uoZuXLVPbimHbndBOYhUDUfkKa AjQtVvONerAjd5RHyt2i2AoYrkjD26traC4jzg7NukZlmrRjfPRg4q3gwW +EZEEXvz+ba9JnQfsXx+iH".
- The Message Authentication Code (MAC) is also attached which is "enc_mac_key:UtQHBswQimAZAp4s/9sfSQSpuq0="
- The "expires_in:1209600" shows the time after which the association will expire. This time is in seconds. After this time, the Consumer and the Identity Provider will go through another association.

Another thing to note about the association is that it is up to the Consumer to initiate the association request. The Consumer can perform association at any time it deems necessary. If an association fails, the Consumer may elect to use Dumb mode of communication without any association handle.

5.4 Diffie-Hellman (DH) Key Exchange Mechanism

Diffie-Hellman key exchange mechanism enables two parties to exchange a shared key in a secure fashion. This shared key is used to encrypt further data exchange between these two parties. The two parties may be completely unknown to each other for the key exchange process to take place in a secure fashion.

5.4.1 Basic Process for Generating DH Keys

The algorithm itself is quite simple in its nature. The step-by-step process is as follows:

1. Let us say Bob and Alice want to establish a shared secret. Bob and Alice will start with a large prime number p and another small number, called generator g . Both of these numbers need not be secret and any other party may know these numbers. Both Bob and Alice already know these two numbers.
2. Now Bob will chose a secret number a which will be typically a large number. Bob will compute a number using formula $(g^a \bmod p)$ and will send it to Alice. Let us say this number is x .
3. Alice will pick her own secret b and computer a number on her side using formula $(g^b \bmod p)$ and send it to Bob. Let us say this number is y .
4. Note that Alice and Bob never share their selected secrets a and b . Now that both Bob and Alice have numbers sent by each other (x and y), they will use another formula on their side to come up with a shared secret.
5. Bob will use formula $(y^a \bmod p)$ which will give him a shared secret key, let us say $K1$. Essentially value of $K1$ is $(y^a \bmod p)$, which is $((g^b \bmod p)^a \bmod p)$.
6. Alice will use formula $(x^b \bmod p)$ which will give her the same shared secret key, $K2$. Value of $K2$ is $(x^b \bmod p)$, which is $((g^a \bmod p)^b \bmod p)$. Mathematically $((g^b \bmod p)^a \bmod p)$ and $((g^a \bmod p)^b \bmod p)$ are the same, so both $K1$ and $K2$ are the same.

Now that Bob and Alice have calculated the same shared secret key, they can use it for further communication, or even exchange a stronger key using this shared key. Also note that an eavesdropper can get hold of the prime number p , numbers g , x , and y by sniffing the network. However, the eavesdropper can't calculate key K because numbers a and b are never sent over the network.

Let us take a simple example to explain the whole process. Let us assume that p is 19 and g is 2. Also let us assume that Bob picks a as 3 and Alice picks b as 5. Now let us plug these values to calculate $K1$ and $K2$.

$$K1 = (2^5 \bmod 19)^3 \bmod 19 = (32 \bmod 19)^3 \bmod 19 = 13^3 \bmod 19 = 2197 \bmod 19 = 12$$

$$K2 = (2^3 \bmod 19)^5 \bmod 19 = (8 \bmod 19)^5 \bmod 19 = 8^5 \bmod 19 = 32768 \bmod 19 = 12$$

As you can see, both Bob and Alice reach the same number, which is 12. Now they can use this number as a secret key. In practical cases, the prime number p and numbers a and b will be quite large resulting in a shared key which is long enough to withstand exhaustive search.

5.4.2 Diffie-Hellman Variants

Also note that the basic algorithm is vulnerable to man-in-middle attacks because both parties have no way to authenticate to each other. This problem can be solved using PKI and certificates.

Many variants of Diffie-Hellman key exchange mechanism exist. OpenID uses one of these variants described in RFC 2631²⁴ as well as in OpenID specifications.

OpenID specifications describe default values of p and g to be used by the OpenID Consumer and Identity Providers. The default value of g is 2 and default value of $\text{base64}(\text{btwoc}(p))$ is as follows (as listed in OpenID specifications):

DCF93A0B883972EC0E19989AC5A2CE310E1D37717E8D9571BB7623731866E61E

²⁴ RFC 2631 is available at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc2631.txt.pdf>

F75A2E27898B057F9891C2E27A639C3F29B60814581CD3B2CA3986D268370557
7D45C2E7E52DC81C7A171876E5CEA74B1448BFDFAF18828EFD2519F14E45E382
6634AF1949E5B535CC829A483B8A76223E5D490A257F05BDFF16F2FB22C583AB

The `btwoc` function converts a large arbitrary precision integer into big-endian two's complement form. This is the standard form used in OpenID. Please refer to OpenID specifications for more information.

5.5 Chapter Summary

In this chapter you learned how to install and run an OpenID server of your own. The important things discussed in this chapter are as follows:

- Installing an OpenID server
- Configuring and creating profiles
- Backend database for OpenID server
- XRD and Yadis
- Different messages used in OpenID protocol
- Diffie-Hellman key exchange algorithm

This was the final chapter related to core OpenID protocol. Next chapters will present additional concepts and using OpenID in enterprise environment.

5.6 References

- OpenID at <http://openid.net>
- OpenID Book web site at <http://www.openidbook.com>
- OpenID blog at <http://openid.blogspot.com>

Chapter Six

OpenID Extensions

In addition to the OpenID authentication protocol, which is the base of any OpenID solutions, a number of OpenID extensions are proposed. Some of these extensions are mature while others are in draft phase. This chapter provides information about these extensions that exist today. New OpenID extensions will also be included in this Chapter in future as they become available. Latest information about these extensions can be found at <http://openid.net/specs.bml> where you can find draft as well as final versions of these extensions.

OpenID extensions deal with such things like:

1. User registration which enables Consumers to register new users when they login to a web site for the first time.
2. OpenID service key discovery using Yadis
3. DTP messages which are MIME encoded
4. Attribute exchange or AX
5. Assertion quality

This chapter is to have discussion on these extensions in more detail and enable readers to use extensions that make sense for them.

After reading this chapter, you should be able:

- What OpenID extensions are.
- Understand how different OpenID extension work
- When and where to use these OpenID extensions
- How to submit new extensions to OpenID

You will also be able to appreciate how open protocols can be extended and used for different purposes.

Note that OpenID extensions use additional parameters in OpenID messages to convey extra information.

6.1 Simple Registration or Profile Exchange

The Simple Registration extension²⁵ is a mechanism to convey commonly used parameters associated with a user profile. To be exact, there are eight parameters which are commonly used in a user profile. These profile parameters are as listed below. The work in parenthesis is the keyword used in OpenID messages as you will see shortly:

1. Full name (fullname)
2. Nick name (nickname)
3. Email (email)
4. Date of Birth (dob)
5. User Gender (gender)
6. User Language (language)
7. Country (country)
8. Time Zone (timezone)

For the user registration purpose (or any other purpose), the Consumer web site can request these additional parameters inside the `checkid_immediate` or `checkid_setup` authentication request messages. The Consumer can also specify if a requested parameter is “optional” or “required”. Previously in this book, you have already seen how this extension was used in graded authorization.

²⁵ Note that Verisign PIP identity provider web site that you used in the first chapter, implements simple registration extension and in our example, we requested email address in addition to the authentication.

6.1.1 How It Works

In a typical scenario, when a Consumer receives an OpenID URL from a user, it may check if the user already exists or if this is the first time a user is logging into the web site. If a user already exists, the Consumer may request a simple authentication from the Identity Provider. However, if a Consumer finds that it is the first time a new user is trying to login, it can attach request for these additional profile parameters in the authentication request. By doing so, the Consumer web site can create a more personalized profile for the End User because now it has such things as full name, nick name, etc.

The Consumer can also use these parameters for additional tasks as graded authorization.

6.1.2 Typical Use Cases

From a usability perspective, the benefits of profile exchange are very good. For example, if a Consumer does not know the full name of an end user, it will display a welcome message something like “**Welcome <http://consumer.conformix.com/?user=openidbook>, we hope to serve you better**”, which is not that intuitive or user-friendly. But if the Consumer web site knows the full name, the same message will be something like “**Welcome Rafeeq Rehman, we hope to server you better**” which is more personalized and looks friendlier.

In other cases, a Consumer may also use additional profile messages for graded or risk-based authentication. For example, depending upon how many parameters are returned by an Identity Provider, the Consumer may provide different levels of access.

6.1.3 Message Description

As mentioned earlier, the Consumer will send these additional parameters with the `checkid_immediate` or `checkid_setup` request messages. The following parameters in these messages are used for simple registration.

- The **openid.ns.sreg** parameter shows the location of namespace which is “`http://openid.net/extensions/sreg/1.1`”. With older version 1.0 of the specification, you may use “1.0” instead of “1.1”. However, you should note that this parameter is optional.
- The **openid.sreg.required** is a list of comma separated parameter. If the Identity Provider does not reply back with all of the parameters in the response message, the Consumer has two options: Either not register the End User, or prompt the end user to enter the remaining parameters manually. The understanding is that the Consumer must have all of these parameters for user registration.
- The **openid.sreg.optional** is also a comma separated list of parameters which the Consumer has requested but will not insist on having them.
- The **openid.sreg.policy_url** parameter is a URL where the Consumer will put information about how the parameters will be used by the Consumer. Typically it would be a “privacy policy” statement. When Identity Provider received a request with this parameter, it should display that web page to the End User so that the End User can decide whether or not to send these parameters to the Consumer.

In general, any time a Consumer requests these additional parameters for user registration purpose, the Identity Provider should prompt the End User before sending these parameters to the Consumer. The End User should be given a choice which parameters it wants to send to the Identity Provider. From

examples in the preceding chapters, you have seen how the Identity Provider prompted the End User when the Consumer requested the Email address.

Once an Identity Provider has received a request for Simple Registration message, it will reply back to the Consumer with a response message. In addition to regular response parameters, following additional parameters will be attached to the response message with a successful authentication. If authentication fails, none of these parameters will be attached to the response. Note that here all of the possible parameters are listed but the response will contain only those parameters which are requested by the Consumer.

- The **openid.ns.sreg** is the same as described earlier.
- The **openid.sreg.nickname** in UTF-8 format
- The **openid.sreg.fullname** in UTF-8 format
- The **openid.sreg.postcode** in UTF-8 format
- The **openid.sreg.email** which should have email address of the End User.
- The **openid.sreg.dob** parameter which shows date of birth and will be in YYYY-MM-DD format. The Identity Provider can make some parts as 0 (zero) if you want to send only partial date of birth information.
- The **openid.sreg.gender** in “M” or “F” format where M denotes Male and F for Female.
- The **openid.sreg.country** parameter shows the country code. Country codes are defined by International Standards Organization (ISO) and you can see a list of country codes at <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>. Note that these

country codes are used for many purposes in daily life, like postal services, domain names, stock market trading, and so on.

- The **openid.sreg.language** parameter defines the language codes as defined by ISO. List of languages codes is available at http://www.loc.gov/standards/iso639-2/php/code_list.php web site.
- The **openid.sreg.timezone** parameter shows time zone of the End User. List of time zones can be found at http://en.wikipedia.org/wiki/List_of_tz_zones_by_name where you will see names for time zones for different parts of world.

Note that the Identity Provider may decide to return only a subset of the parameters that a user requested based upon the End User selection.

The following message shows an optional parameter email being requested. The relevant part in the following listing is in boldface.

```
HTTP/1.1 302 Found
Date: Mon, 26 Mar 2007 01:16:42 GMT
Server: Apache/2.2.3 (Fedora)
X-Powered-By: PHP/5.1.6
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Location:
http://idp.conformix.com/index.php/serve?openid.assoc_handle={HMAC-SHA1}{46071e25}{Tt8MwQ==}&openid.identity=http://idp.conformix.com/?use
r=openidbook&openid.mode=checkid_setup&openid.return_to=http://consumer
.conformix.com:80/finish_auth.php?nonce=nC5sKquX&openid.sreg.optional=e
mail&openid.trust_root=http://consumer.conformix.com:80/
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

In the above listing, the “openid.sreg.optional=email” part shows the OpenID extension part.

6.1.4 Simple Registration and Yadis

Note that Simple Registration extension is a service and a Consumer can use Yadis protocol to discover this service. The following is a typical XRD document retrieved by Yadis request that shows Simple Registration as a service.

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS
  xmlns:xrds="xri://$xrds"
  xmlns:openid="http://openid.net/xmlns/1.0"
  xmlns="xri://$xrd*($v*2.0)">
  <XRD>
    <Service>
      <Type>http://openid.net/signon/1.1</Type>
      <Type>http://openid.net/sreg/1.0</Type>
      <URI>http://idp.conformix.com/index.php/serve</URI>

      <openid:Delegate>http://idp.conformix.com/?user=openidbook</openid:Dele
gate>
    </Service>
  </XRD>
</xrds:XRDS>
```

The relevant part is shown in boldface in the above listing under the “Services” node. From this you know that Simple Extension is a service like other services provided by an Identity Provider. It can be discovered using Yadis like any other service.

6.2 OpenID Service Key Discovery

OpenID uses PKI (Public Key Infrastructure) for many purposes. In PKI two parties communicate with each other using public-private key pair. Each party

holds keeps the private key secret (also known as secret key) but make the public key available to everyone who wants to use it.

Public keys are used to establish secure connections in daily life very often. For example, PKI is used in SSL connection to secure communication with a web site. Typically Public/Private key mechanism is used to exchange a shared key. Encryption by a shared key is much cost effective compared to public/private key in terms of computation time.

OpenID key discovery extension is a mechanism where two parties or end points of a communication channel can publish and discover public keys.

This extension is in a draft phase and may change over time.

6.2.1 How it Works

Yadis is the basic mechanism to advertise and discover public keys. Yadis uses XRD document as discussed earlier to discover keys. The XRDS document may have an element like the following in XML format.

```
<PublicKey>  
  https://www.conformix.com/certs/openid.crt  
</PublicKey>
```

Once the Yadis has discovered the key URL, it can be retrieved using a GET request over HTTPS protocol. If the key URL does not contain https, the client may not accept the public key.

6.2.2 Typical Use Cases

The public key is used for message exchange in the S/MIME format as discussed next in DTM Messages next.

Public keys can also be used for many other types of tasks, such as:

- The Consumer and IdP may use public keys for encrypting some “out-of-band” communication in a custom application.
- The mechanism may be useful in distributing keys in a corporate environment.

Since this specification is still in draft mode, the implementers should consider the risk of specification being revised later on.

6.2.3 Message Description

The key discovery and retrieval does not need any OpenID message. Yadis is sufficient to discover a key and regular GET request can be used to retrieve the key because the URL is already known to the Consumer by analyzing the Yadis message.

6.3 How to Submit New Specification

If you are engaged in OpenID work and need additional functionality, you have the options of submitting new specifications. You can submit new specifications for OpenID using specs@openid.net mailing list where there will be discussion on the specification by OpenID community. For more information, please refer to <http://openid.net/specs.bml>.

6.4 Chapter Summary

In this chapter you have learned about OpenID extension and how they work. There was specific discussion on:

- OpenID Simple Registration extension
- OpenID Service Key Discovery extension

There are other specifications as well which are in the draft form. Information about these specifications is available on the OpenID web site.

6.5 References

- ISO Codes <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>
- OpenID Specifications <http://openid.net/specs.bml>

Chapter Seven

OpenID as Enterprise Solution

Enterprise environment is significantly different in many ways than small company environment. In the Enterprise, there are many additional things that need to be considered. OpenID is a suitable solution for a number of applications in the enterprise environment, including cross company authentication, Single Sign On (SSO). This becomes especially useful when OpenID is integrated into enterprise LDAP or other directory solutions.

In this chapter we are going to look into different ways of using OpenID in enterprise. OpenID can be used as an effective mean for cross company authentication (CCA) as well as for single sign on (SSO).

After reading this chapter, you will be able to:

- Understand what options are feasible for using OpenID in the enterprise
- How to use OpenID in cross company authentication environment
- OpenID with digital certificates
- Issues related to digital certificates when using with OpenID

7.1 Cross Company Authentication Solutions and OpenID

Cross company authentication (CCA) is often needed in enterprise environment. This is because any large company has to exchange data, login to partner web sites, work with hosted applications, and so on. No company can afford to have a separate username and password for all these web based applications. The preferred way to use an employee's corporate credentials when the employees logs into an external web site. This is where the idea of cross company authentication comes into picture.

Security Assertion Markup Language (SAML) is an established process for cross company authentication. However, this is a heavy protocol and expensive to manage. OpenID provides a light-weight and cost-effective to implement CCA.

This section provides a simple architecture to implement CAA using OpenID and enterprise directory services. For OpenID to be a feasible solution for CCA, it is a must that it should work with enterprise directory.

User credentials in any enterprise are managed using a directory services solution. Most commonly used solutions are:

- Light Weight Directory Access Protocol or LDAP
- Microsoft Active Directory

- Novell eDirectory
- Oracle directory solution
- Sun Micro Systems directory servers

All of these solutions are compatible with LDAP in one way or the other which has become a de facto enterprise standard. That is the reason that OpenID must be able to talk to LDAP to get accepted in this environment. This can be accomplished by adding LDAP as a “store” in different OpenID APIs.

Multiple solutions exist for cross company authentication (CCA). Most commonly used system is SAML (Security Assertion Markup Language). More information about SAML is available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

However, OpenID can also be used for CCA as described next.

7.1.1 General Architecture for OpenID Cross Company Authentication

In a typical enterprise environment, users are managed with the help of some type of directory service. Microsoft Active Directory, Novell eDirectory, LDAP repository, Sun directory, are some example of enterprise directory services. For OpenID to work in an enterprise environment, it must use these directory services to store user information.

A typical OpenID environment where users in an enterprise are able to login to partner web sites is shown in Figure 7-1.

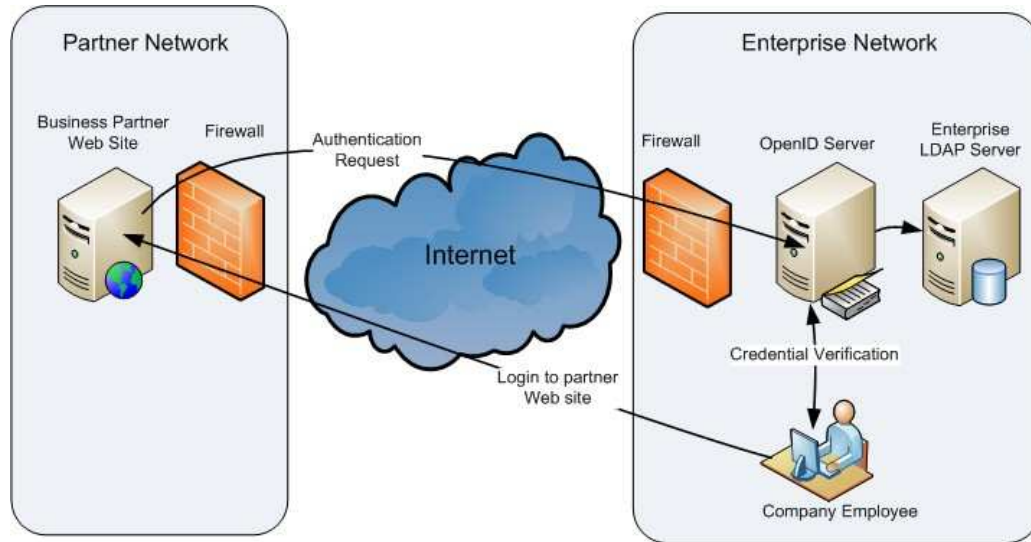


Figure 7-1: A typical enterprise OpenID environment enabling CCA.

In this Figure, a user sitting inside enterprise network is logging in to a partner web site over the Internet. The user enters his enterprise ID. The partner web site converts this ID into an agreed-upon URL and sends it back to the enterprise OpenID server for authentication. The OpenID server may verify user credentials using the backend LDAP repository and then reply the authentication request to the web site in the partner network.

Note that you never send a user credential to the partner web site in this architecture. The only thing a partner web site receives is success or failure of authentication.

Note that the business partner web site must be OpenID enabled and should implement a scheme to convert a regular userID to an OpenID URL. Converting a typical 6-8 characters long userID to an OpenID URL acceptable to the enterprise OpenID server is trivial if you have used examples in Chapter 4.

A number of things have to happen to enable the company employee login to the partner web site using enterprise credentials. The step-by-step process is as follows:

1. The employee goes to the login screen of the web application and enters his/her username used in the enterprise network.
2. The login page on the partner network uses this username and creates an OpenID identity URL and sends it to OpenID server in the enterprise network.
3. The OpenID server can ask the employee to authenticate (if the employee has not already done so).
4. The OpenID server will validate the user using a call to enterprise LDAP systems. Upon success, it will send the positive authentication back to partner web site to complete the login process.

You should note the following items with this arrangement.

- User password is never sent to partner network
- The authentication is done using enterprise LDAP system
- In case an employee is terminated or leaves company, his/her access to partner web sites is automatically terminated as well
- The partner web sites can enable OpenID easily using APIs already available in the open source.

These are very good advantages to use OpenID as a cross company authentication mechanism.

7.1.2 User Interface for OpenID Server

In a typical scenario, the user interface for the OpenID server will be accessible from within the enterprise environment only to improve security.

7.1.3 Partner/Hosted Web Sites

Partner web sites for all business partners or application hosting service providers need to have the following items implemented:

- Partner web sites should be OpenID enabled.
- Partner web sites should have provision for converting a regular userID to OpenID URL for users logging in from the enterprise environment.
- The web sites may also serve requests from users of multiple customers.

7.1.4 Security Controls

For this architecture, different security controls can be put in place. Some of these are as follows:

- The enterprise OpenID server accepts authentication requests only from known partner web sites.
- The firewalls on both the enterprise and partner side allow ports 80 and 443 (HTTP and HTTPS) from known locations only to get to the OpenID server.
- There is no user registration process on the OpenID server. It uses already established backend LDAP systems.
- Users can use their enterprise username and partner web sites can convert these names to OpenID URL, which makes this solution quite seamless for end users.

7.2 Secure OpenID and Digital Certificates

Some identity providers have started providing X.509 certificate based identities. In this case, an identity is attached to a digital certificate installed on a user's computer and available through web browser.

Services like this may be useful in enterprise environment where a PKI solution is already in place.

While implementing any X.509 certificate solution, you should consider things like certificate expiration and revocation and implications of lost certificates on the access control.

7.2.1 Certifi.ca

One of the services providers is Certifi.ca and information is available at its web site <https://certify.ca>. This provider works with digital certificates, i.e. your OpenID identifier is tied to a digital certificate.

You have to get a certificate from another certificate provider. A list of free certificate providers is listed in next section.

Things to note about using Certifi.ca are as follows:

- The web site recognizes you by the certificate. When registering for the first time, the web site will ask you which certificate to you.
- If you don't have a certificate installed in your browser, you can't create your Identity.
- Once the Identity is created, you don't need to authenticate by yourself with the Identity Provider (Certifi.ca). Your browser will do it for you using the certificate.

If your browser has certificate installed, it will act as it is permanently logged in to the web site such that you don't need to authenticate to the OpenID server by entering username/password.

A typical OpenID identifier URL is something like <http://certifi.ca/openidbook>. The neat thing is that you can login from the computer where you installed the

certificate and as long as you protect your computer, you don't need to remember *any* username and password.

Certification expiration may become another issue that need to be considered while implementing solutions like this.

7.2.2 Prooveme

Prooveme provides OpenID Identity Provider services which is based upon digital certificates. When you create you OpenID account, the web site will give you a certificate that you will install in your web browser. This certificate is used for client-side authentication. This means, that whenever you use your OpenID URL to login to a web site, your web browser will use this certificate to authenticate to Prooveme. Since you are not creating any username/password with Prooveme, the authentication process is transparent to you (unless you have configured your browser to prompt to choose a certificate).

While using Prooveme (or similar services), you should keep the following in mind:

- You will not use any username and password to authenticate to the Identity Provider (i.e. Prooveme.com) when you use your Identity URL.
- The authentication will be done using certificate installed in your web browser.
- You should keep a backup copy of the certificate, in case you need to re-install you machine.
- You can login only from the machine where you have installed your certificate. If you want to login from multiple machines, you have to backup your certificate from one machine and install it on other machines.

- If you are using multiple web browsers, you have to install the certificate in each browser. You can do so by backing up the certificate from the original browser which you used to create your account and then install it in other browsers using “import” facility.
- The certificate based authentication is more secure compared to username/password authentication. However, keep in mind that if someone gets physical access to the machine where the certificate is installed, that person will use your account. To overcome this problem, you can password protect the user of your certificate.
- If you are using multiple IDs on the same machine or want to switch IDs, you have to tell your browser which certificate to use. FAQ section on Prooveme web site discusses some steps that you can take for this purpose.

You can create your OpenID URL by visiting Prooveme web site at <https://www.prooveme.com/>.

Once you have installed a certificate in your web browser, you can see it in your web browser. In Internet Explorer, you will go into “Internet Options” in “Tools” menu and then click on the “Content” tab. There you will see a button “Certificates” which will take to the list of certificates you have installed. Figure 7.2 shows a typical window with list of certificates in Internet Explorer.

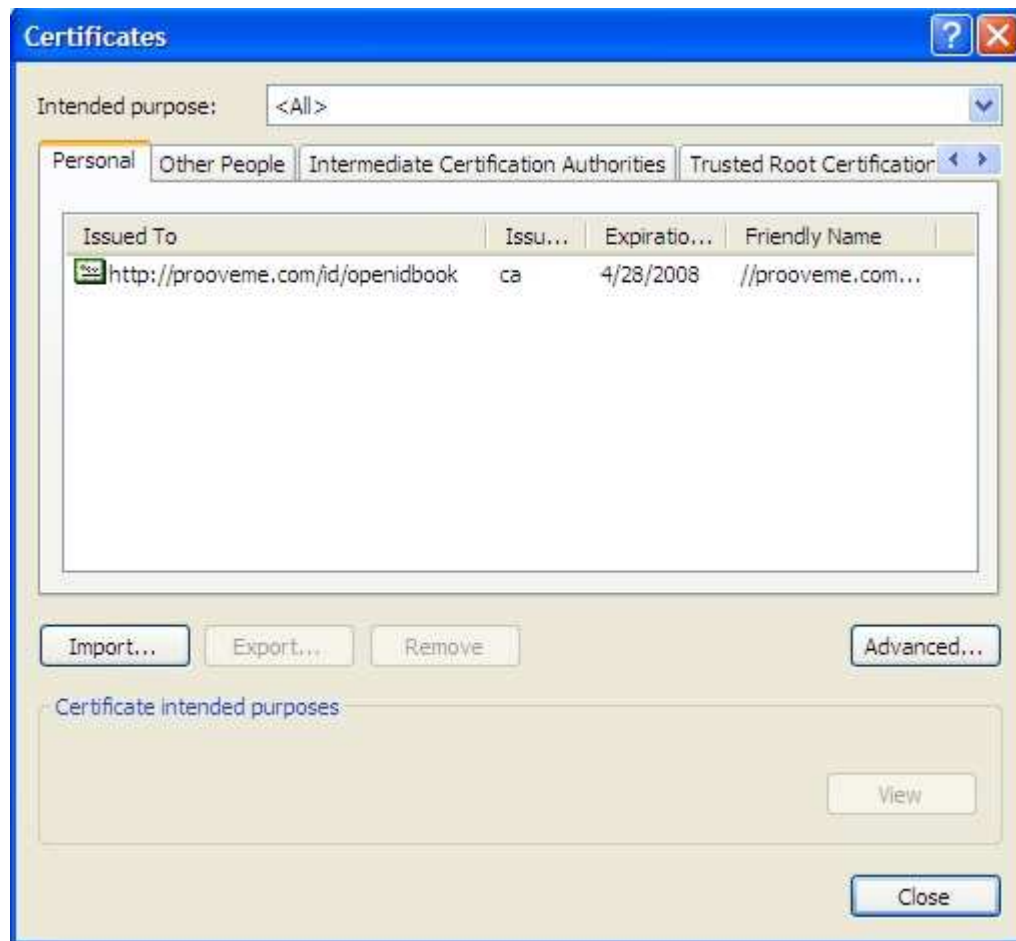


Figure 7.2: List of certificates in Internet Explorer.

In Firefox, you will go to “Tools” menu and then select “Options”. In Options, you will select “Advanced” button and then “Encryption” tab. Now if you click on “View Certificates” button, you should be able to see the certificates. Figure 7-3 shows a typical list of certificates in Firefox browser.

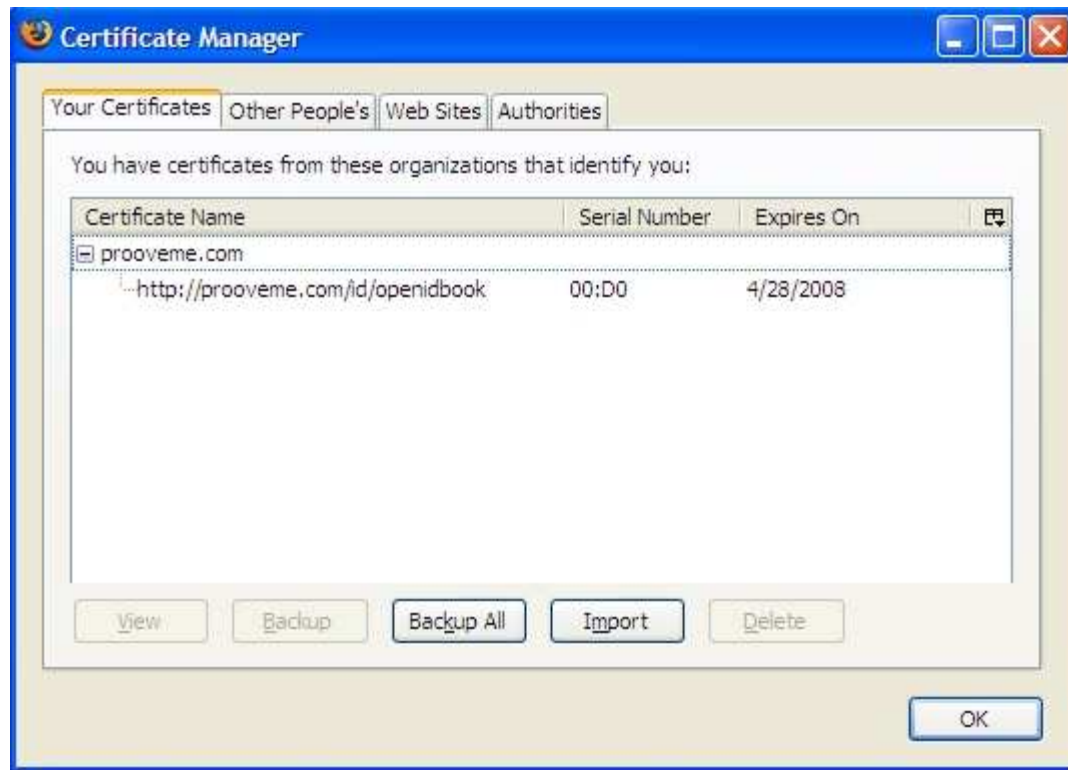


Figure 7-3: List of certificates in Firefox browser.

To view detail of a certificate, you can double click on the certificate name both in Internet Explorer and Firefox. Figure 7-4 shows detail of the Prooveme certificate in Firefox.



Figure 7-4: Detail of Prooveme certificate in Firefox.

Note that I am using Internet Explorer version 7 and Firefox version 2.0. The location may be slightly different in other versions of these browsers.

7.2.3 Getting Free Certificates

There are multiple places on the Internet where you can go and get digital certificates. Some of the locations are listed below:

- CA-Cert at <http://cacert.org>
- Thawte Personal Email Certificate <http://www.thawte.com/secure-email/personal-email-certificates/index.html>
- Startcom Free Certificates <http://cert.startcom.org/>
- Comodo Free Certificates: For personal use, you can get free certificates from Comodo. For more information, have a look at http://www.comodo.com/products/certificate_services/email_certificate.html
- CA Cert www.cacert.org

7.3 OpenID and OpenSSO

There is another effort to integrate OpenID and OpenSSO and more information about this effort is available at the following web site.

<https://opensso.dev.java.net/public/extensions/openid/>

7.4 Chapter Summary

In this chapter, you looked at different ways OpenID can be used in an enterprise environment and how to use certificate based solution for OpenID.

The important thing to remember is that for OpenID to be a viable solution in the enterprise environment, it has to work with solutions like LDAP. In some cases OpenID may be a better solution than existing solutions. It is lightweight compared to SAML and provides adequate security and ease of use as well.

Enterprise environment is more challenging compared to small companies as for as identity management solutions are concerned. In this chapter we looked at two major items important to use of OpenID in the enterprise. These are:

- Use of OpenID as cross company authentication solution.
- OpenID solutions related to digital certificates.

OpenID can be a good solution especially for CCA if implemented properly. An architecture is proposed for implementing CCA using OpenID for business partners and external application hosting.

7.5 References

- SAML http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
- Certifi X.509 based Identity Provider <http://certifi.ca>
- Prooveme X.509 based Identity Provider <https://www.prooveme.com/>
- OpenID and OpenSSO <https://opensso.dev.java.net/public/extensions/openid/>

Chapter Eight

OpenID Protocol: Miscellaneous Topics

We have discussed a number of issues related to OpenID protocol in previous chapters. However, there are some other topics that are not yet discussed. This chapter is kind of “catch all” for all of the remaining chapters.

After reading this chapter, you will be able to:

- Basic security issues related to OpenID

- Privacy
- Use of OpenID protocol for desktop applications
- Send-a-message protocol

8.1 OpenID Security Issues

Like any other protocol, OpenID should be implemented in a secure way. There are some security issues that should be considered. This section provides information about some security issues.

8.1.1 Relay Attacks

In some cases, OpenID may be vulnerable to relay attacks. Probability of relay attacks is minimized with the help of nonce variable in OpenID messages. However, if a Consumer is not actively storing nonce or allows a long lapse for timestamp in the nonce value, it may become vulnerable to relay attacks.

To avoid relay attacks, following steps may be useful.

- All Consumers and Identity Providers should use NTP (Network Time Protocol) to keep the clock synchronized with a standard time source.
- Consumers should discard those messages for which difference between the current time at the consumer and the timestamp in the nonce variable is far off.

8.1.2 Phishing Attack

OpenID is vulnerable to some phishing attacks and there are efforts in progress for making improvements to avoid phishing attacks.

8.1.3 OpenID and Use of SSL

Use of SSL (Secure Socket Layer) is very important in any OpenID system. SSL encrypts all data on the transport level and is very commonly used as a security measure between web browser and web server.

It is highly recommended that all OpenID protocol communication Consumer, Identity Provider, and the User Agent should be SSL encrypted.

8.1.4 OpenID and Browser History

Since a number of OpenID messages are passed using HTTP GET method, there is a possibility that these messages are stored in browser history. If an attacker gets access to the machine, the browser history may provide significant information. This is especially important if a user is using a public computer where multiple people have access to the browser history. Solution to this problem lies in education and awareness of a user. In case of a public computer use, the user should remove the browser history after he/she is done with the web site access.

Another important thing to note about the HTTP GET method is un-intentional posting of query string parameters to marketing services that use methods like transparent GIF files. Many web sites use these services for tracking web site visitor behavior and effectiveness of different web pages. However, these services may use JavaScript in addition other methods to send URLs to the marketing company web sites. If an OpenID-enabled web site developer is using a marketing service and is not careful about marketing services or other tracking mechanisms, the URLs in OpenID protocol messages may be sent to a marketing company inadvertently.

8.2 OpenID and Privacy

In addition to the security of the OpenID protocol itself, there are some privacy related issues that may need to consider. These privacy issues are especially important if you are using an OpenID identity provider from a third party. Some of the major concerns are listed in this section.

The privacy concerns are not a big issue if you use OpenID in the enterprise and have a control over the OpenID identity server.

8.2.1 Saving OpenID Credentials on Identity Provider Web Sites

One of the privacy concerns is saving identity information on an Identity Provider's web site. The security and privacy of this information depends upon the security of the OpenID Identity Provider. In some cases, the data stored on the Identity Provider's location may be quite sensitive. For example, if date of birth, address, and social security numbers are stored at the Identity Provider's servers, any breach of the Identity Provider may result in serious issues like identity theft.

This issue does not exist in the corporate environment where a company is hosting its own identity server.

8.2.2 Identity Providers Logging

Identity providers may log a user's activities by logging to which web site a user goes to, time of visits, and so on. This is very sensitive information related to a user habits and behavior and has serious privacy concerns.

This problem can be solved in multiple ways:

- If you are corporation, you should be able to run your own Identity server

- Before you use an Identity Provider service, read carefully their privacy and information use policy.

8.3 OpenID for Desktop Clients and Miscellaneous Uses of OpenID

Typically OpenID is used with web-based applications. However, there are some interesting discussions about using OpenID for desktop application as well. The following URL shows an introduction to use of OpenID for desktop.

http://blog.wachob.com/2007/03/openid_for_desk.html

There are also some discussions about this topic on OpenID mailing lists.

8.3.1 Send-a-Message Protocol

In regular web-based applications, you can invite other people/friends to your web site or community using their email addresses. However, in OpenID, there is only a URL and not an email address. So how to invite someone using OpenID? The following URL gives you a draft proposal about “send-a-message protocol which may be interesting for some of the readers.

http://openid.net/wiki/index.php/Send_A_Message_Protocol

There is some discussion on OpenID mailing lists as well.

8.4 Chapter Summary

This chapter covered miscellaneous topics about OpenID that were not discussed anywhere else.

Appendix A

Glossary

OTP	One Time Passwords which are randomly generated and used only once.
SSH	Secure Shell is a mechanism to log on to remote servers in a secure way. The protocol establishes a secure communication tunnel between two end points for data transport.
Digital Certificate	Digital Certificate or X.509 Certificate is the electronic information that identifies an entity. The entity may be a person, a server, or any other device to name a few. The certificate may also be used for applications and in many scenarios authentication and

	authorization is done using certificates. Certificates are also used to encrypt internet traffic, like SSL or sending secure email.
IVR	Interactive Voice Response is a system used with traditional telephony to get information over telephone.
SSO	Single Sign On
CCA	Cross Company Authentication
SSL	Secure Socket Layer
Consumer	Consumer is web-enables application that uses the OpenID system for authentication purposes.
Relying Party	See Consumer
XRD	eXtensible Resource Descriptor

Appendix B

References and Useful Links

10.1 References

- Yadis <http://yadis.org>
- Wireshark packet sniffer <http://www.wireshark.org>
- Ethereal packet sniffer <http://www.ethereal.com>
- Smarty Software <http://smarty.php.net/>

- JanRain PHP server
<http://www.openidenabled.com/resources/downloads/php-server/PHP-server-1.1.tar.gz>
- JanRain OpenID PHP Library
<http://www.openidenabled.com/resources/downloads/php-openid/PHP-openid-1.2.2.tar.gz>
- OpenID main page <http://openid.net>
- Diffie-Hellman Key Agreement Method RFC 2631
<http://www.ietf.org/rfc/rfc2631.txt>
- Hypertext Transfer Protocol HTTP/1.1
<http://www.ietf.org/rfc/rfc2616.txt>

10.2 RFCs

OpenID specifications are compliance with many RFC²⁶s (Request For Comment). Following is a list of RFCs that you can refer to for detailed information.

- RFC 2104: Keyed-Hashing for Message Authentication at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc2104.txt.pdf>
- RFC 1750: Randomness Recommendations for Security at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc1750.txt.pdf>

²⁶ RFC or Request For Comment are documents published by the Internet Engineering Task Force (IETF) and they define Internet standards. List of RFCs can be found at www.rfc-editor.org

- RFC 3174: US Secure Hash Algorithm 1 (SHA1) at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc3174.txt.pdf>
- RFC 3548: The Base16, Base32, and Base64 Data Encodings at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc3548.txt.pdf>
- RFC 3330: Date and Time on the Internet: Timestamps at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc3339.txt.pdf>
- RFC 2631: Diffie-Hellman Key Agreement Method at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc2631.txt.pdf>
- RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1 at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc2616.txt.pdf>
- RFC 2119: Key words for use in RFCs to Indicate Requirement Levels at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc2119.txt.pdf>
- RFC 3629: UTF-8, a transformation format of ISO 10646 at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc3629.txt.pdf>
- RFC 3986: Uniform Resource Identifier (URI): Generic Syntax at <ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc3986.txt.pdf>

10.3 OpenID Libraries

- Google Java OpenID Library <http://code.google.com/p/openid4java/>
- PHP5 implementation <http://www.openidforphp.org/>

10.4 OpenID Providers

- MyOpenID <http://www.myopenid.com>

- PIP <http://pip.verisignlabs.com>
- Sun Microsystems Identity Services at <http://openid.sun.com>

10.5 Miscellaneous

- Coder and Decoder <http://meyerweb.com/eric/tools/dencoder/>
- RP Best Practices
http://openid.net/wiki/index.php/Relying_Party_Best_Practices
- ISO Country Codes <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>
- ISO Language Codes http://www.loc.gov/standards/iso639-2/php/code_list.php
- Time Zones List
http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
- OpenID Directory at <http://openiddirectory.com/> provides listing of different resources for OpenID. It is a very useful web site.

Appendix C

Index

CardSpace, 20, 49	OpenSSO, 203
CCA, 19, 193, 194, 211	Privacy, 208
Certifi.ca, 197	<i>Relying Party</i> , 20
<i>Consumer</i> , 20	single sign on, 19
cross-company authentication, 19	SSL, 43, 62, 74, 188, 207, 210, 211
Diffie-Hellman, 176	SSO. <i>See</i> single sign on
<i>Identity Provider</i> , 20	X.509, 43
OpenID, 18	

Appendix D

Advertisement in the Book

In next pages, you can advertise your business in this book to reach your target audience in the information security business. Your advertisement will help continue this project and provide free PDF downloads for all through the OpenID book web site <http://www.openidbook.com>. For more information, contact us at info@conformix.com

CONFORMIX TECHNOLOGIES INC.

Web Site Security Assessment

Web Site Penetration Testing

 **Conformix**
Technologies

Web address at
<http://www.conformix.com>

E-mail: info@conformix.com



Conducted by

Rafeeq U Rehman

CISSP

On-Site Training
for a group of 5 or more

Contact Conformix Technologies Inc
at
info@conformix.com

Advertise Your Business Here

For more information, contact

info@conformix.com

Advertise Your Business Here

For more information, contact

info@conformix.com

Advertise Your Business Here

For more information, contact

info@conformix.com

Advertise Your Business Here

For more information, contact

info@conformix.com

Rafeeq Rehman Other Books

1. HP-UX CSA Book: <http://www.amazon.com/HP-UX-CSA-Official-Reference-Professional/dp/0131448544/>
2. HP Certified Book: <http://www.amazon.com/HP-Certified-HP-UX-System-Administration/dp/0130183741/>
3. Snort Book: <http://www.amazon.com/Intrusion-Detection-SNORT-Advanced-Techniques/dp/0131407333/>
4. Linux Development Platform Book: <http://www.amazon.com/Linux-Development-Platform-Rafeeq-Rehman/dp/0130091154/>
5. Solaris 8 Network Administrator Book: <http://www.amazon.com/Solaris-Network-Administrator-Training-CD-ROM/dp/1578702615/>
6. CISSP Certification Exam Notes at <http://www.conformix.com/books/cissp>