Research on OpenID and its integration within the GravityZoo framework

Jarno van de Moosdijk
jarno.vandemoosdijk@os3.nl

January 2008

UNIVERSITEIT VAN AMSTERDAM

Abstract

This report covers the research on the working of OpenID and its integration into the GravityZoo framework. The GravityZoo framework can be used to deliver applications to a wide variety of devices. Currently, it is only possible to log in with a username and password combination. OpenID authentication eliminates the need for separate username/password combinations bound to each service. One identifier can be used to log in to multiple OpenID enabled services across different providers. The objective of this document is to give an in depth view on the OpenID login procedure. It covers the usability of the most popular OpenID Providers on mobile devices. None of these providers have custom pages for mobile users, which makes them less usable on mobile devices. Next to this, I have looked in to the requirements for integrating OpenID authentication within the GravityZoo framework.

Contents

1 Introduction

In this period I have done research on OpenID [2] and the requirements for its integration into the GravityZoo framework [1]. OpenID is a decentralized user identification standard that makes use of URIs [3]. It eliminates the need for multiple usernames across different services. The URI that identifies the user can be used to log in to all OpenID enabled services.

The GravityZoo framework can be used to deliver applications to a wide variety of devices. There is no application code distributed to the devices itself. The applications run in the back-end on application servers. This is totally transparent to the end user because there is no difference in the feeling of the application.

Personal data generated with (or by) the application is stored on the storage back-end and can be accessed through any device that has the GravityZoo client installed. The only requirement is that you log in to the GravityZoo framework with the same account as before. The mobile and "fat client" version of the GravityZoo client all work on the same back-end. GravityZoo is currently focusing on the mobile version of their application framework. This framework is compatible with phones running the Symbian S60 operating system.

1.1 Research Focus

The focus of my research, which covered the month January, first laid on the OpenID specification itself. I have done research on how the OpenID authentication works and explained the working of the protocol in depth in this report. This was needed to specify the requirements for integrating OpenID in to the GravityZoo framework.

Because GravityZoo is currently focusing on the mobile version of their client, I tested the most popular `OpenID Providers` on their mobile phone friendliness.

After this I looked in to the requirements of integrating OpenID into the GravityZoo framework. The requirements for all steps of the OpenID authentication process are covered in this report. The research can be summarized in the following questions:

- *How does OpenID work?*
- *How mobile phone friendly are the most popular OpenID Providers?*
- *What are the requirements for integrating OpenID into the GravityZoo framework?*

1.2 Structure of this report

The first part of the report gives an overview of the GravityZoo framework. Chapter 3 gives background information about GravityZoo's choice for OpenID, followed by basic terminology you should know when talking about OpenID. This chapter also visualized the steps that an end user has to go through to log in using an OpenID account.

Chapter 4 gives a representation of all steps of the OpenID authentication procedure. This chapter covers all technical details, including explanation of the messages that are exchanged between the different parties. Chapter 5 covers extensions that are currently standardised by the OpenID foundation.

Chapter 6 contains a test of various popular OpenID Providers. This test covers the usability of them on a mobile device. The test is done using several different mobile phones.

Chapter 7 focuses on the requirements that have to be looked at when implementing OpenID. Two potential scenarios are introduced, one of them is chosen. After that I draw my conclusions and give some ideas for further research.

2 The GravityZoo Framework

This chapter provides a global overview of the different layers within the GravityZoo Framework.

The GravityZoo Framework is a framework designed for the development and hosting of any kind of application. It can be used to deliver applications to a wide variety of device types. The applications are available at anytime and anyplace. Figure 1 contains an overview of the different layers of the GravityZoo Framework.

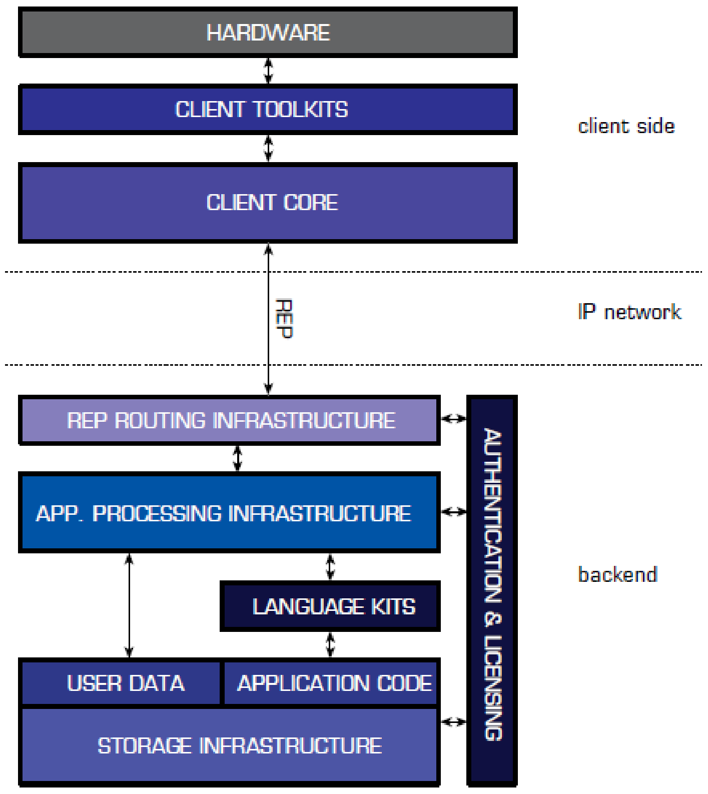


Figure 1: Overview of the GravityZoo Framework

A small software stack is installed on each client. There is no application code distributed to the client devices itself. The client core is a platform independent part of the client which communicates with the back-end. Next to this, the client core maintains the client-side state, processes updates and enforces client side security policies. The client toolkit forms the glue between the client OS and the and the client core.

The GravityZoo Framework uses REP [1]. This is a routing protocol that is designed by GravityZoo itself and works over an IP network. The protocol uses both TCP/IP and UDP/IP. TCP for reliable data transmission and UDP for unreliable real-time data transmission. The main purpose of REP is to act as a “vessel” for object updates, sent amongst client and server. REP is also designed for [1]:

- Bi-directional communication, allowing the client to not only request for new information, but also send updates to clients without a request, that is if the client allows to do so.
- Traversing NAT[1] and most existing firewalls.
- Delivering end-to-end encryption as a default functionality.
- The support of quality of service, link speed and quality monitoring.

The application processing infrastructure does the actual application processing. The state of the application is not stored in the application infrastructure. It is stored, in the Central Object Store (COS) together with the user data. The COS is also known as the storage infrastructure which is a huge intelligent “object database”. The storage infrastructure consists of two layers: The storage nodes and storage handlers. The latter are responsible for handling requests to the COS. They operate on a transaction basis and ensure data integrity and the efficient distribution of data. Storage nodes are responsible for the permanent storage of user and application data. The language kits can be seen as glue between the language in which an application is developed and the GravityZoo framework itself. Currently there are language kits available for Python and C. Language kits for C# and Java will be available in the near future.

The last - and especially for this research project - not least part of the GravityZoo framework is the authentication and licensing module. This module acts as the gatekeeper of the framework. Currently, the only option of logging in to the GravityZoo framework is by supplying a username and password. The authentication request containing the username and password is routed by the REP routers to a server which services the Authentication and Licensing Service (`ALS`) role. An authentication request has the REP packet type: “authentication request”. When a REP router sees this, it knows that the packet has to be routed to an `ALS`. The `ALS` checks the supplied username and password combination. It sends back a reply to the client, confirming the login or stating that the username and password combination is not valid.

[1]NAT stands for Network Address Translator, RFC 1631, http://www.faqs.org/rfcs/rfc1631.html

3 OpenID overview

OpenID is an open, decentralized user identification standard that makes use of URIs [3]. It has arisen from the open source community to solve the problems that could not be easily solved by other existing technologies.

People are using the Internet more and more every year. When someone wants to use a service on a website, he (or she) normally has to register to that service first. The user has to register an account with a corresponding username and password. The username you use as default, may already be taken by somebody else. Different usernames and passwords on different sites make it hard to memorize every login. This is where OpenID comes into play.

OpenID allows a user to log onto an OpenID enabled service with the same identity everywhere. When a user only needs to memorize one password, this password can be more sophisticated. To make use of OpenID, the only thing you have to do is register at an OpenID provider. When registering at an OpenID provider, you receive a username in URL style, for example: `https://logmij.in/als/jarno`. When authenticating at an OpenID enabled service, you prove that you are the owner of the subdomain (or page) you supplied.

3.1 Why choose for OpenID?

Why does GravityZoo want to integrate OpenID into their Cloud OS? The main reason is simple. When integrating OpenID, a user can choose to reuse their own OpenID identifier for authentication purposes within the Cloud OS. The user does not need to register a new username with a password which he needs to remember again. There are more reasons to choose for OpenID.

The OpenID specification is owned by the OpenID Foundation which was formed in June 2007 [4]. The foundation is formed to promote, protect and enable the OpenID technologies and community. This ensures that the OpenID specification will always stay freely implementable. As Brad Fitzpatrick (the father of OpenID) said[5]:

> *“Nobody should own this. Nobodys planning on making any money from this. The goal is to release every part of this under the most liberal licenses possible, so theres no money or licensing or registering required to play. It benefits the community as*

a whole if something like this exists, and were all a part of the community."

Next to the guarantee that OpenID will always be freely implementable, it is gaining support of more and more big companies. Companies like AOL, Google, IBM, Microsoft, MySpace, VeriSign and Yahoo! already act as OpenID providers. Google, IBM, Microsoft, VeriSign and Yahoo! even joined the OpenID Foundation and all have their own corporate board member [4]. Gaining the support of these companies is a big step in the way of becoming the standard authentication method over the Internet. The companies all see the added value of a global identity and accept the OpenID specification as being one.

Over time there have been many other initiatives which were trying to simplify the management around digital identities. Think of Shibboleth [7], A-Select[8] and Windows Cardspace (codename InfoCard) [6]. The latter is now collaborating with OpenID [9]. Everybody who has registered at Microsoft - think of all the hotmail users - already have an OpenID without even knowing it. Same goes for users who registered at the websites of AOL, Flickr, WordPress,Yahoo! and Google[2].

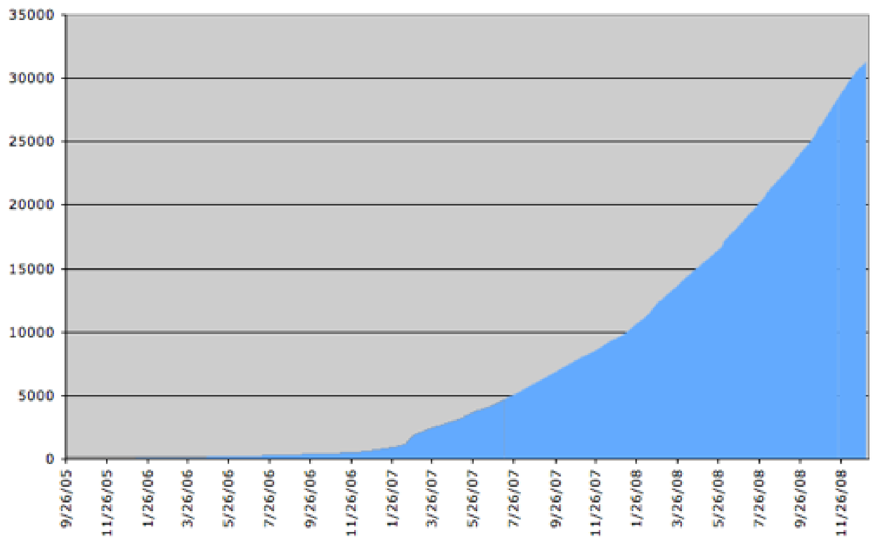


Figure 2: OpenID Relying Parties as of January 1, 2009 [11]

The integration of OpenID into websites has recently seen a steep climb as can be seen in figure 2. In January 2008 there were about 9.600 websites

[2]Google requires an API to support OpenID [10]

supporting OpenID. This increased enormously: to 31.000 websites in January 2009.

When you registered at a website using a regular username and password, you often have the ability to add your OpenID details to your current account. This enables you to log in with both your username/password and through OpenID.

Summarizing the three main reasons to choose for OpenID:

- The user can reuse his own OpenID identifier to log in to GravityZoo.
- The OpenID is accepted by big companies world wide.
- The OpenID will remain intellectual property and will always be freely implementable.

3.2 Basic terminology

To understand how OpenID works, you first have to be familiar with the terms that are frequently used when talking about OpenID [2]:

- **Identifier**. An Identifier is either a "HTTP" or "HTTPS" URI or an XRI (Extensible Resource Identifier).
- **User-Agent**. The user's client software which implements HTTP/1.1.
- **Relying Party** (RP). A Web application that wants proof that the end user controls an Identifier.
- **OpenID Provider** (OP). An OpenID Authentication server on which a Relying Party relies for an assertion that the end user controls an Identifier.
- **OP Endpoint URL**. The URL which accepts OpenID Authentication protocol messages, obtained by performing discovery on the User-Supplied Identifier.
- **OP Identifier**. An Identifier for an OpenID Provider.
- **User-Supplied Identifier**. An Identifier that was presented by the end user to the Relying Party, or selected by the user at the OpenID Provider. During the initiation phase of the protocol, an end user may enter either their own Identifier or an OP Identifier. If an OP Identifier is used, the OP may then assist the end user in selecting an Identifier to share with the Relying Party.

- **Claimed Identifier**. An Identifier that the end user claims to own; the overall aim of the protocol is verifying this claim. The Claimed Identifier is either:

  - The Identifier obtained by normalizing (Normalization) the User-Supplied Identifier, if it was an URL.

  - The CanonicalID (XRI and the CanonicalID Element), if it was an XRI.

- **OP-Local Identifier**. An alternate Identifier for an end user that is local to a particular OP and thus not necessarily under the end user's control.

3.3 OpenID identifier delegation

It is not mandatory to use the URL that is directly registered at an OpenID Provider as your identifier. If you have your own domain name, you can use this domain name as your OpenID identifier. There are several ways to do this. The first one is to run your own OpenID provider on your webserver. The second and easiest way is to link the URL that you registered at an OpenID Provider to your domain name. This can be done by adding the HTML code found below, to the `HEAD` section of your index file. The first line specifies the OpenID Endpoint URL and the second specifies the identifier that should be used.

OpenID identifier delegation

```
<link rel="openid.server" href="https://logmij.in/index.php/serve">
<link rel="openid.delegate" href="https://logmij.in/als/jarno">
```

When - for any reason - you want to switch to another OpenID Provider, the only thing you need to do is change the OpenID provider in your index file. You can keep using the domain name of your choice as your OpenID identifier.

3.4 Example login procedure using OpenID

The following example describes the user experience when logging in with an OpenID `identifier`. An in depth version of this procedure can be found in chapter 4.

In the example, `website X` is the Relying party (`RP`) which is logged in to. This website provides two ways of authentication: `(1)` normal authentication through username and password. `(2)` Authentication through OpenID. `OP Y` is be the OpenID provider (`OP`) in this example. The example is visualized in figure 3.

The process of logging on to `website X` goes as follows:

1. Open `website X` as you would do normally.
2. Enter your OpenID `identifier` in the login box.
3. You are redirected to your `OP` where you will be asked to log in. This proofs that you are the owner of the `identifier` you provided.
4. After logging in at `OP` you get a confirmation dialog that the `RP` requested authentication for your `identifier`. You have several options on this page. `(1)` Accept the request permanently, so the `RP` will be always allowed to request authentication. `(2)` Accept the request only this time. `(3)` Deny the request of the `RP`. After accepting the request you are be redirected back to the `RP` and the authentication procedure is finished.

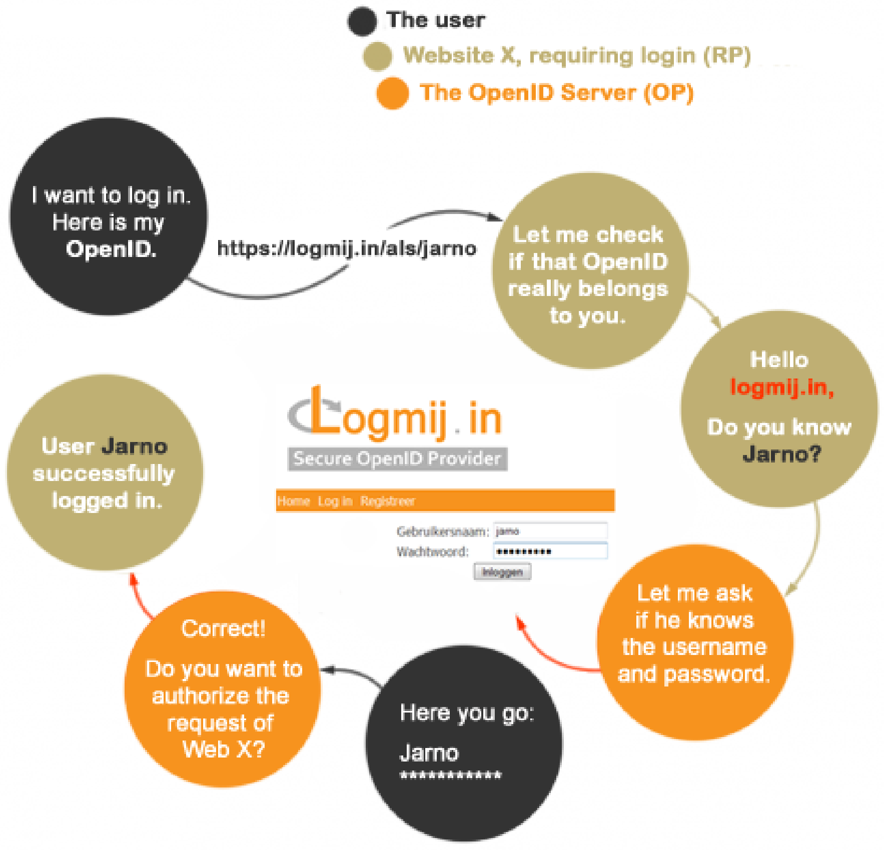

The user
Website X, requiring login (RP)
The OpenID Server (OP)
I want to log in. Here is my OpenID.
https://logmij.in/als/jarno
Let me check if that OpenID really belongs to you.
Hello logmij.in, Do you know Jarno?
Let me ask if he knows the username and password.
Here you go: Jarno ************
Correct! Do you want to authorize the request of Web X?
User Jarno successfully logged in.
Logmij.in
Secure OpenID Provider
Home Log in Registreer
Gebruikersnaam: jarno
Wachtwoord: **********
Inloggen

Figure 3: OpenID: Global overview

4 OpenID in depth

This chapter gives an in depth description of the OpenID login procedure [2], [14]. The procedure in chapter 3.4 just describes the steps which a user sees when logging on to a website.

All steps in figure 4 are be covered in this chapter. Each of the following paragraphs is - according to its number - dedicated to to one of the steps in figure 4.

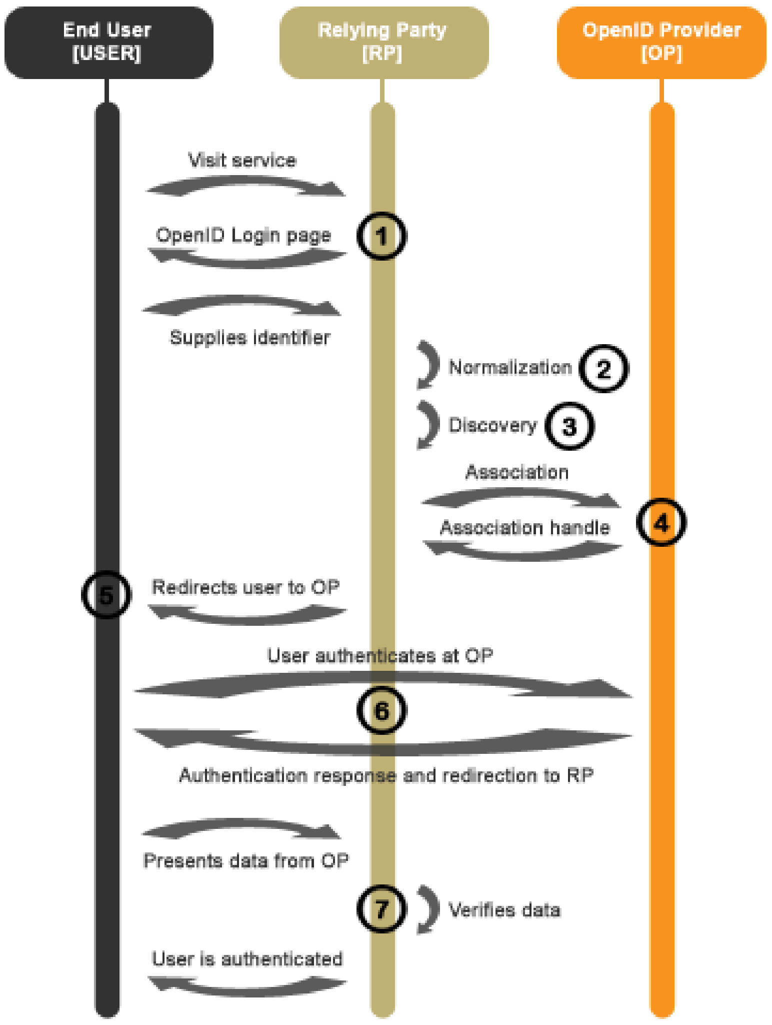


Figure 4: OpenID: In depth

All communication between the parties is serviced by `HTTP` commands. OpenID Parameters are sent by appending them to the used identifiers of the `OpenID Provider` and the `Relying Party`, prefixed with a question mark. Listing 1 contains an example of an URL containing OpenID parameters. All available OpenID parameters are be explained in the upcoming paragraphs.

```
https://logmij.in/index.php/serve?openid.assoc_handle=%7BHMAC-SHA1%7D
    %7B49744372%7D%7BMEOX0w%3D%3D%7D&openid.identity=https%3A%2F%2
    Flogmij.in%2Fals%2Fjarno&openid.mode=checkid_setup&openid.
    return_to=http%3A%2F%2Fopenidenabled.com%2Fresources%2Fopenid-test
    %2Fdiagnose-server%2FTestCheckidSetup%2F%3Faction%3Dresponse%26
    attempt%3D1%26nonce%3DPIX42n6G&openid.trust_root=http%3A%2F%2
    Fopenidenabled.com%2Fresources%2Fopenid-test%2Fdiagnose-server%2
    FTestCheckidSetup%2F
```

Listing 1: An OpenID URL

4.1 The supplied identifier

A user opens the OpenID enabled service and is asked to supply his OpenID identifier. The identifier supplied by the user is called `user-supplied identifier`. The user can supply two types of identifiers:

1. A HTTP or HTTPS URL.
2. A XRI (Extensible Resource Identifier).[3]

4.2 Identifier normalization

The next step of the OpenID procedure is normalization `(2)`. The user may have supplied an incomplete identifier. The `RP` checks and corrects the `user-supplied identifier` during the normalization process.

The input of the normalization process is the `user-supplied identifier`. The output of the process is called the `claimed identifier` if the user supplied a HTTP(S) URL, or `CanonicalID` if the user supplied a XRI address.

The normalization process goes as described in figure 5. First, if present, the XRI prefix is stripped from the `user-supplied identifier`. The normalization process stops here if the remaining string starts with a XRI global context symbol. The XRI global context symbols are: `=`, `@`, `+`, `$` and !.

The remaining string is processed as a HTTP(S) identifier if no XRI global context symbols are found. The string is checked for a HTTP(S) prefix. If it does not have one of both, a `http://` prefix is added. Next to this,

[3]A XRI is a scheme compatible with URI and IRI which looks like `xri://authority/path?query#fragment` [12].

any fragments in the URL are removed. Lastly an URL identifier must be further normalized by both following redirects when retrieving their content and finally applying the rules in section 6 of RFC 3986: URI [3].

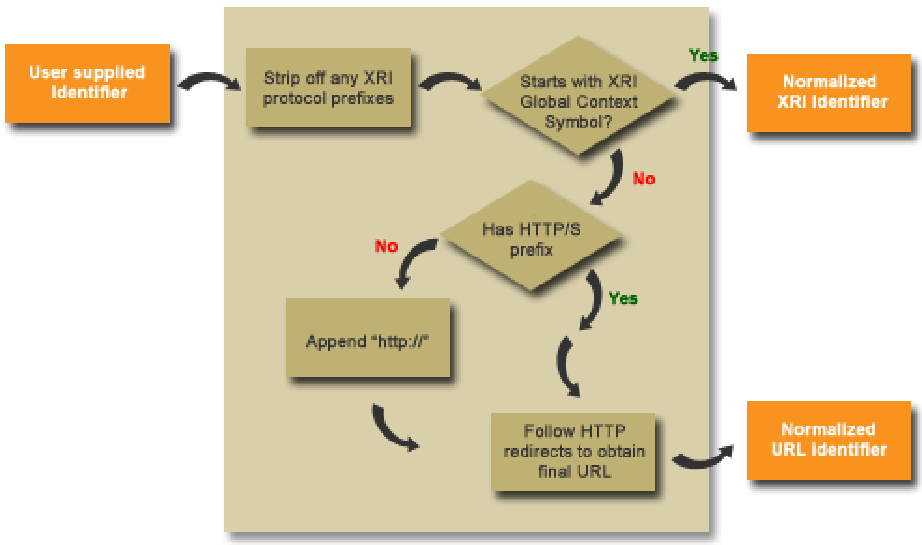

User supplied Identifier
Strip off any XRI protocol prefixes
Starts with XRI Global Context Symbol?
Yes
Normalized XRI Identifier
No
Has HTTP/S prefix
No
Yes
Append "http://"
Follow HTTP redirects to obtain final URL
Normalized URL Identifier

Figure 5: OpenID: Normalization

Table 1 contains examples of `user-supplied identifiers` and the obtained `claimed identifier` or `CanonicalID`. [13].

User's Input	**Identifier**	**Type**	**Comment**
example.com	http://example.com/	URL	missing scheme URI is normalized to a http URI
http://example.com	http://example.com/	URL	An empty path component is normalized to a slash
https://example.com/	https://example.com/	URL	https URIs remain https URIs
http://example.com/user	http://example.com/user	URL	No trailing slash is added to non-empty path components
http://example.com/	http://example.com/	URL	Trailing slashes are preserved when path is empty
xri://$dns*example.com	http://example.com	URL	$dns is replaced with http://
=example	=example	XRI	Normalized XRIs start with a global context symbol
xri://=example	=example	XRI	Normalized XRIs start with a global context symbol

Table 1: User-supplied identifiers and their normalized versions

4.3 Discovery

Discovery is the process where the RP uses the claimed identifier to look up the necessary information for initiating authentication requests at the OP. The following information is discovered during this process:

1. OP endpoint URL
2. Protocol version
3. Claimed identifier
4. OP-Local identifier

If the end user entered an OP Identifier, there is no Claimed Identifier. For the purposes of making OpenID Authentication requests, the value http://specs.openid.net/auth/2.0/identifier_select must be used as both the Claimed Identifier and the OP-Local Identifier when an OP Identifier is entered.

There are three types of discovery. Which type is used process depends on the type of the claimed identifier. The types of resolution mentioned in table 2 are covered in the next three paragraphs.

Identifier type	Resolution Protocol	Resultant document
XRI	XRI resolution	XRDS document
URL	Yadis Protocol	XRDS document
URL	HTML discovery	HTML

Table 2: The different ways of discovery

4.3.1 XRI resolution

XRI resolution resolves an XRI into metadata describing the resource identified by the XRI. This resolution is done through HTTP(S). An XRI can be transformed into a URI by prefixing the XRI with `http(s)://xri.*/`. The URI now refers to a so called proxy resolver, which resolves a URI of this kind to an XRDS document. XRDS is covered in chapter 4.3.4.

Relying parties can take advantage of XRI proxy resolvers. This removes the need for the RP to perform XRI resolution locally. An example of a proxy resolver is http://www.xri.net. The XRI identifier `=example` becomes http://xri.net/=example. The latter is called an HTTP XRI or short: HXRI. The owner of the XRI `=example` can tell the proxy resolver what to do, if the HXRI is called. One possible action is to do a 302 HTTP redirect to a stored URI [15].

4.3.2 Yadis resolution

The Yadis protocol can link URL based identities to an XRDS document. This document contains metadata that is linked to the identity. The owner of a Yadis identifier can manage the content of a XRDS document. Such document can contain several identities. After the `RP` retrieves this document, it can select an appropriate service(s) of the XRDS document. A Yadis XRDS document can be retrieved in one of the following ways [16]:

1. By following a custom HTTP response header called `X-XRDS-Location`.
2. By an equivalent entry in the HTML HEAD section, called:
   `<meta http-equiv="X-XRDS-Location" content="http://example.com/yadis.xml">`
3. By requesting a special mime type called `application/xrds+xml` when performing an HTTP GET on the identity URL.

4.3.3 HTML discovery

HTML discovery is only be used if Yadis resolution fails. With HTML discovery, the HEAD section of the HTML file is checked for the following values:

```
HTML Discovery
<link rel="openid2.server" href="https://logmij.in/index.php/serve">
<link rel="openid2.local_id" href="https://logmij.in/als/jarno"/>
```

The second value of the listing above is optional. It should only be present if the end user is using a delegation. With HTML discovery, the `RP` discovers the OpenID provider endpoint URL and the protocol version that is used. This enabled the `RP` to communicate with the `OP`

4.3.4 XRDS: Extensible Resource Descriptor Set

When using XRI- or Yadis-resolution, metadata is retrieved in a extensible XML document called an XRD (Extensible Resource Descriptor). Multiple XRD documents can be returned inside a single XRDS (Extensible Resource Descriptor Set) container document.

This document is designed to provide information about the identifier. In the case of OpenID, the XRDS specifies the OpenID server(s) that can be used for authenticating the normalized identifier. The file may contain priority parameters that indicate the user's order of preference. The lowest priority number takes precedence. An example XRDS file with multiple entries can be found in listing 2, taken from [16].

The following information can be distilled from this file:

1. The URL is a Yadis identity URL.
2. The URL supports the OpenID protocol, through two servers and two delegates.
3. The URL supports version 1.0 and version 2.0 of the LID protocol, with a delegate.
4. The owner of this identity URL prefers to using sign-on using their LiveJournal account and the OpenID protocol (priority 10). If this is not possible because OpenID is not supported, or because the LiveJournal server is unavailable, the owner would like to use the LID URL `http://mylid.net/liddemouser`. Lastly if the LID URL is not working either, the MyOpenID service with a priority of 50 is tried.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns:xrds="xri://\$xrds" xmlns="xri://$xrd*($v*2.0)"
xmlns:openid="http://openid.net/xmlns/1.0">
  <XRD>
    <Service priority="50">
      <Type>http://openid.net/signon/1.0</Type>
      <URI>http://www.myopenid.com/server</URI>
      <openid:Delegate>http://smoker.myopenid.com/</openid:Delegate>
    </Service>
    <Service priority="10">
      <Type>http://openid.net/signon/1.0</Type>
      <URI>http://www.livejournal.com/openid/server.bml</URI>
      <openid:Delegate>http://www.livejournal.com/users/frank/</
          openid:Delegate>
    </Service>
    <Service priority="20">
      <Type>http://lid.netmesh.org/sso/2.0</Type>
      <URI>http://mylid.net/liddemouser</URI>
    </Service>
    <Service>
      <Type>http://lid.netmesh.org/sso/1.0</Type>
    </Service>
  </XRD>
</xrds:XRDS>
```

Listing 2: XRDS example file

4.4 Establishing an association (Optional)

The fourth step in the authentication process is the establishment of an association between `Relying Party` and `OpenID Provider`. It is not mandatory to form an association, although it is recommended. The association establishes a shared secret between both parties which enables them to communicate securely. Diffie-Hellman key exchange is used to generate a shared secret-key. The security can be implemented either via the transport layer, by using SSL/TLS, HMAC-SHA1 or HMAC-SHA256 [2]. The shared secret is also used to sign/verify subsequent messages and to reduce round trips. The signature generation process is described in chapter 4.8.

If there is no association established between both parties, the `Relying Party` has to request authentication at the `OpenID Provider` itself. Subsequently, the `RP` has to send another request to the `OP` to verify the authentication. This allows the `RP` to be stateless because it would not need to keep track of association handles.

An association is initiated by a direct request from the `RP` to the `OP Endpoint URL` with the `openid.mode` key set to `associate`. The signing algorithm to be used is specified by the `openid.assoc_type` key. The last important key is `openid.session_type`. This key defines the algorithm that is used to encrypt the association's MAC key in transit. Unless using SSL/TLS, the use of `no-encryption` is prohibited.

The result of a successful association request is an `assoc_handle` that the `RP` and `OP` can use as a key to refer to this association in subsequent messages. A full specification of the parameters used when requesting an association can be found in table 3.

Parameter name	**Explanation**
`openid.ns`	Always contains the value `http://specs.openid.net/auth/2.0`.
`openid.mode`	Contains value `associate`.
`openid.assoc_type`	Defines the preferred signing algorithm. Two options: `HMAC-SHA1` or `HMAC-SHA256` (latter is recommended).
`openid.session_type`	Defines the preferred encryption method used. Three options: `no-encryption`, `DH-SHA1`, `DH-SHA256`
`openid.dh_modulus`	Modulus used for Diffie-Hellman. The default modulus can be found in appendix B of [2].
`openid.dh_gen`	`base64(btwoc(g)`. The default `g` = 2.
`openid.dh_consumer_public`	`base64(btwoc( g ^ xa mod p)` [2].

Table 3: Parameters used when requesting an association

The function `btwoc` that is used by the Diffie-Hellman key exchange fields handles integer representation, more information on this function can be found in chapter 4.2 of [2].

Parameters used in the response to a successful association request can be found in table 4.

Parameter name	Explanation
openid.ns	Always contains the value http://specs.openid.net/auth/2.0.
openid.assoc_handle	Association handle that is used as a key to refer to this association in subsequent messages. A string 255 characters or less in length.
openid.session_type	A copy of the value of this field during the request.
openid.assoc_type	A copy of the value of this field during the request.
openid.expirs_in	The lifetime, in seconds, of this association. Integer, base 10 ASCII.
openid.mac_key	The shared secret for this association, Base 64 encoded. Only included when session_type is set to no-ecnryption.
openid.dh_server_public	base64(g ^ xb mod p). The OP's Diffie-Hellman public key.
openid.enc_mac_key	base64(H(btwoc(g ^(xa * xb) mod p)) XOR MAC key). MAC key, encrypted with the DH-secret. H is either SHA1 or SHA256. Not included when session_type is set to no-encryption.

Table 4: Parameters used when responding to a successful association

The OP responds with a unsuccessful response if it receives an association request with a session type or association type which it does not support. The parameters used in such response can be found in table 5.

Parameter name	Explanation
openid.ns	Always contains the value http://specs.openid.net/auth/2.0.
openid.error	A human-readable message indicating why the request failed.
openid.error_code	Contains unsupported-type.
openid.session_type	A valid association session type that the OP supports (optional).
openid.assoc_type	A valid association type supported by the OP (optional).

Table 5: Parameters used when responding to a **un**succesful association

4.5 Redirecting the end user, requesting authentication

The next step is step five from figure 4. The end user is asked to authenticate during this step. The Relying Party asks the OpenID Provider verify authentication of the identifier. The RP does not directly communicate with the OP, but redirects the user to the OP. This has security advantages as it allows the OP to read cookies from the end user. Next to this, it does not leak authentication details to the RP. Table 6 contains all parameters used in an authentication request.

The request can be handled in two different modes, checkid_setup and checkid_immediate. When the latter is used, no interaction between OP and end user is required, which makes the login procedure more user friendly. This can be used when the front-end uses an asynchronous web technique such as AJAX[4]. If authentication in checkid_immediate mode fails, the RP can place a subsequent request in checkid_setup mode. This mode requires interaction between end user and OP.

If the OP wants you to confirm relationships with RP's, then the first time checkid_immediate mode is used fails from any RP. This until an authentication request in checkid_setup mode is completed and the RP is given permission to do future authentication requests from the end user at the OP. The OP may use the realm to allow the end user to configure automatic approval of future authentication requests by that RP.

A realm is a pattern that represents the part of URL-space for which an OpenID Authentication request is valid. A realm is designed to give the end user an indication of the scope of the authentication request. OPs should present the realm when requesting the end user's approval for an authentication request. The realm is used by OPs to uniquely identify RPs. A realm pattern is a URL, with the following changes:

- A realm may not contain any URI fragments.
- A realm may contain a wild-card (*.) at the beginning of the URL authority section.

Overly general realms, like http://*.com/ or http://*.co.uk/ can be dangerous. OPs should protect their users against overly general realms.

The values claimed_id and identity should either be both present or both absent. If neither value is present, the assertion is not about an identifier, and contains other information in its payload, using extensions.

Parameter name	**Explanation**
openid.ns	Always contains the value http://specs.openid.net/auth/2.0.
openid.mode	Has a value of checkid_setup or checkid_immediate.
openid.claimed_id	(optional) The identifier that the end user claims to own.
openid.identity	(optional) If a different OP-Local Identifier is not specified, the claimed identifier must be used as the value for this field.
openid.assoc_handle	(optional) The handle that should be used between RP and OP to sign the response. If not available, transaction will be in stateless mode.
openid.return_to	(optional) URL to which the OP redirects the user to once authentication is finished.
openid.realm	(optional) URL pattern the OP asks the user to trust. Must be present if the return_to parameter is omitted. Default value: The return_to URL.

Table 6: Parameters used when requesting authentication

[4]AJAX: Asynchronous JavaScript and XML (http://nl.wikipedia.org/wiki/Asynchronous_JavaScript_and_XML).

4.6 End user authentication at OP

After receiving the authentication request, the OP must decide whether to allow or reject the user's authentication. This can be done based on whether the end user has previously authenticated with the OP.

The authentication request from the RP has been redirected via the user to the OP. From this point, the OP takes over the control from the RP. The OP responds asynchronously to the authentication request. This means that the OP can have an entire sequence of interactions with the end user before it responds to the request. Most OPs make use of this by asking the end user if the authentication request from the RP should be allowed or denied.

If the RP requested OP-driven identifier selection by setting `openid.identity` to http://specs.openid.net/auth/2.0/identifier_select, and there are identifiers for which the end user is authorized to issue authentication responses, the OP should allow the end user to choose which `identifier` to use.

If the RP supplied an association handle with the authentication request, the OP should attempt to look up an association based on that handle. If the association is missing or expired, the OP should send the `openid.invalidate_handle` parameter as part of the response with the value of the request's `openid.assoc_handle` parameter, and should proceed as if no association handle was specified.

If no association handle is specified, the OP should use a private association for signing the response. The OP must store this association. It must respond to later requests of the RP to check the signature of the response via `Direct Verification`.

If an authorized end user allows the authentication request, the OP should send a positive assertion to the RP. Assertions are covered in the next two paragraphs.

4.6.1 Positive Assertion

If authentication is successful, a positive assertion is sent by the OP. This response is sent via a redirect through the end user to the RP. This ensures that the RP and OP do not communicate directly during the authentication process. A positive assertion message contains the values listed in table 7.

Parameter name	Explanation
`openid.ns`	Always contains the value http://specs.openid.net/auth/2.0.
`openid.mode`	Value: `id_res`. States that this is a response to an authentication request.
`openid.op_endpoint`	The OpenID provider endpoint URL.
`openid.claimed_id`	(optional) The identifier that the end user claims to own.
`openid.identity`	(optional) Same as the `openid.claimed_id`
`openid.return_to`	Verbatim copy of the `return_to` URL parameter sent in the request.
`openid.response_nonce`	A string 255 characters or less in length, that must be unique to this particular successful authentication response. It must contain the current time on the server (in UTC), indicated with a "Z". Example: `2009-01-15T17:14:56ZUNIQUE`.
`openid.invalidate_handle`	(optional) If the `RP` sent an invalid association handle with the request, it should be included here.
`openid.assoc_handle`	The handle for the association that was used to sign this assertion.
`openid.signed`	Comma-separated list of signed fields. (without the `openid.` prefix).
`openid.sig`	Base 64 encoded signature.

Table 7: Parameters used when sending a **positive** assertion

4.6.2 Negative Assertion

The `OP` sends a negative assertion if he is unable to identify the end user or the end user does not or cannot approve the authentication request. As with the positive assertion, the response is sent through indirect communication to the `RP`. When receiving a negative assertion in response to a `checkid_immediate` mode request, Relying Parties should construct a new authentication request using `checkid_setup` mode [2].

The responses to the two modes mentioned in the last paragraph are different. Table 8 contains the parameters used when sending a response to a request in `checkid_immediate` mode. Table 9 contains the response parameters used in response to a request in `checkid_setup` mode.

Parameter name	Explanation
`openid.ns`	Always contains the value http://specs.openid.net/auth/2.0.
`openid.mode`	Value: `setup_needed`.

Table 8: Parameters used in a **negative** assertion (checkid_immediate)

Parameter name	Explanation
`openid.ns`	Always contains the value http://specs.openid.net/auth/2.0.
`openid.mode`	Value: `cancel`.

Table 9: Parameters used in a **negative** assertion (checkid_setup)

4.7 Verifying the authentication response

If the RP receives a positive assertion, it must verify the assertion before accepting it. The user is successfully authenticated after verifying all steps. The verification process consists of the following four steps [2]:

1. The value of openid.return_to matches the URL of the current request.
2. Discovered information matches the information in the assertion.
3. An assertion has not yet been accepted from this OP with the same value for openid.response_nonce.
4. The signature on the assertion is valid and all fields that are required to be signed are signed.

The steps are covered in the next four paragraphs.

4.7.1 Verifying the return URL

To verify that the openid.return_to URL matches the URL that is processing this assertion:

- The URL scheme, authority, and path must be the same between the two URLs.
- Any query parameters that are present in the openid.return_to URL must also be present with the same values in the URL of the HTTP request the RP received.

4.7.2 Verifying the discovered information

The Claimed Identifier in the assertion may be an URL that contains fragments. The fragment part and the fragment delimiter character # may not be used for the purposes of verifying the discovered information.

If the Claimed Identifier is included in the assertion, it must have been discovered by the RP. The information in the assertion must be present in the discovered information. It is not allowed that the Claimed Identifier is an OP-Identifier.

There are several scenario's where the Claimed Identifier is not previously discovered by the RP. This happens if the openid.identity in the request is set to http://specs.openid.net/auth/2.0/identifier_select or a different Identifier, or if the OP is sending an unsolicited positive assertion. If the RP has not discovered the Claimed Identifier in the response, it has to do it during this step. This to make sure that the `OP` is authorized to make assertions about the `Claimed Identifier`.

If no `Claimed Identifier` is present in the response, the assertion is not about an identifier. The `RP` may not use the `User-supplied Identifier` associated with the current OpenID authentication transaction to identify the user. Extensions are used in most scenario's when the `Claimed Identifier` is not present.

4.7.3 Checking the nonce

Nonces are implemented as prevention of replay attacks (although they don't prevent active replay attacks in my opinion). The agent checking the signature keeps track of the nonce values included in `positive assertions`. It never accepts the same value more than once for the same `OP Endpoint URL`.

The `OP` is not allowed to issue more than one successful response to a request with the same value for `openid.response_nonce`. The `RP` should ensure that an assertion has not yet been accepted with the same value for `openid.response_nonce` from the same `OP Endpoint URL`.

Nonces contain time-stamps. They may be used to reject responses that shift too much from the current time. This limits the time that nonces must be stored to prevent attacks. The OpenID specification does not contain recommendations for this. A larger range would require storing more nonces for a longer time. A shorter range increases the chance that clock-skew and transaction time causes a spurious rejection.

4.7.4 Verifying the signatures

There are two ways of validating an authentication response. Which way is used, depends on if there is an association made between the `RP` and the `OP`.

If there is an association between both parties, `indirect verification` is used. The `RP` has stored the association with the association handle specified in the assertion. This association is used to check the signature of the signed fields. The signature generation process is described in chapter 4.8.

The second scenario comes in to play when the `RP` has no association stored under the corresponding handler. The `RP` must request that the `OP` verifies the signature through a `direct request`. It does this by sending a request to the `OP` with `openid.mode` set to `check_authentication`. The rest of the fields are exact copies of the fields from the authentication response.

The `OP` responds with a message containing the parameters listed in table 10.

Parameter name	**Explanation**
`openid.ns`	Always contains the value `http://specs.openid.net/auth/2.0`.
`openid.is_valid`	Value: `true` or `false`. Asserts whether the signature of the verification request is valid.
`openid.invalidate_handle`	(optional) Value: The `invalidate_handle` value sent in the verification request, if the `OP` confirms it is invalid.

Table 10: Parameters used in responding to direct verification request

The following keys should be signed at the minimum, as stated in the specification.

- `op_endpoint`.
- `return_to`.
- `response_nonce`.
- `assoc_handle`.
- `claimed_id` (if present).
- `identity` (if present).

4.8 Generating signatures

This paragraph covers the procedure that is used to generate a message signature. In most scenario's, MAC (Message Authentication Code) is used to sign OpenID authentication messages. Generating signatures goes as follows:

1. Determine the list of keys to be signed, according to the message to be signed. The list of keys is comma-seperated and stored with the key `openid.signed` within the same message.
2. Iterate through the list of keys to be signed in the order they appear in the `openid.signed` list. For each key, find the value in the message whose key is equal to the signed list key prefixed with `openid.`
3. Convert the list of key/value pairs to be signed to an octet string by encoding with Key-Value Form Encoding[5].
4. Determine the signature algorithm from the association type. Apply the specified signature algorithm to the octet string.

[5]A sequence of lines. Each line begins with a key, followed by a colon, and the value associated with the key[2]

5 OpenID Extensions

The OpenID foundation is working on several extensions for the OpenID authentication mechanism. The majority of the `OpenID Providers` have not implemented the extensions yet. According to the OpenID support page [18], `MyVidoop.com` is currently the only `OpenID Provider` which has implemented all three extensions.

The three extensions that reached the final version of their specification are covered in the upcoming paragraphs. The paragraphs do not fully cover the message types used by these extensions. They intend to give a feature overview of each extension.

GravityZoo wants me to focus on the implementation of the OpenID authentication mechanism, and not include the extensions for now. Although to give a complete picture of the possibilities, I have included overviews of the extensions.

5.1 OpenID Simple Registration Extension 1.0

OpenID Simple Registration is an extension that allows for very light-weight profile exchange [19].

When registering at an `OpenID provider`, you have to supply information about yourself. This extension allows `RP`s to request user profile data from the `OpenID provider`. After logging in at the `RP`, the `end user` gets an overview of the data requested by the `RP`. The user can choose which data is allowed to be seen by the `RP`. The `RP` can specify which data fields are mandatory to finish the registration.

The `Simple Registration Extension` can supply `RP`s with the following information that is frequently needed to register at a website:

Field	**Description**
openid.sreg.nickname	Any UTF-8 string that the End User wants to use as a nickname.
openid.sreg.email	The email address of the End User.
openid.sreg.fullname	UTF-8 string free text representation of the End User's full name.
openid.sreg.dob	The End User's date of birth as YYYY-MM-DD.
openid.sreg.gender	The End User's gender, `M` for male, `F` for female.
openid.sreg.postcode	UTF-8 string that conforms to the End User's country's postal system.
openid.sreg.country	The End User's country of residence as specified by ISO3166.
openid.sreg.language	End User's preferred language as specified by ISO639.
openid.sreg.timezone	ASCII string from TimeZone database. Examples: `Europe/Amsterdam` or `America/Los_Angeles`.

Table 11: Simple Registration Extension: Supplied information

5.2 OpenID Provider Authentication Policy Extension 1.0

OpenID Provider Authentication Policy Extension 1.0 (`PAPE`) is one of the newest extensions of OpenID. `PAPE` 1.0 is approved on December 31, 2008 and provides an important security enhancement to OpenID Authentication. This extension is optional, though its use is recommended [20].

`PAPE` enables `RP`s to request that `OP`s employ specified authentication policies when authenticating end users. It also provides a mechanism by which an `OP` can inform a `RP` which authentication policies were used. For example: A `RP` can request that the `OP` employs a phishing-resistant authentication method for authenticating the user. The `RP` can request information from the `OP` to see if this method is really used.

Extension overview [20]:

- As part of the Yadis Discovery process covered in chapter 4.3.2, `OP`s can add the supported authentication policies to an end user's XRDS document. This aids `RP`s in choosing between multiple listed `OP`s depending on authentication policy requirements.
- The `RP` includes parameters in the authentication request describing its preferences for authentication policy for the current assertion.
- The `OP` processes the `PAPE` request, prompting the end user to fulfill the requested policies during the authentication process.
- As part of the `OP`'s response to the `RP`, the `OP` includes `PAPE` information around the end user's authentication.
- When processing the `OP`'s response, the `RP` takes the `PAPE` information into account when determining if the end user should be sent through additional verification steps or if the OpenID login process cannot proceed due to not meeting policy requirements.

`PAPE` can also be used to request multi-factor authentication and to learn what NIST level the authentication conforms to. NIST authentication levels are levels specified by the National Institute of Standards and Technology. An overview of levels can be found in table 12 and 13. The minimum requirement for level 3 and 4 is two-factor authentication, level 1 and 2 require normal single-factor authentication. More in depth information can be found in [21].

At the time of this writing, the specification contains the following three predefined authentication policies [20]. The URLs included with each method are the predefined schema locations. The descriptions are cited from the specification.

1. Phishing-Resistant Authentication.

   http://schemas.openid.net/pape/policies/2007/06/phishing-resistant

   An authentication mechanism where a party potentially under the control of the RP can not gain sufficient information to be able to successfully authenticate to the end user's OP as if that party were the end user. The potentially malicious RP controls where the User-Agent is redirected to and thus may not send it to the end user's actual OP.

2. Multi-Factor Authentication.

   http://schemas.openid.net/pape/policies/2007/06/multi-factor

   An authentication mechanism where the end user authenticates to the OP by providing more than one authentication factor. Common authentication factors are something you know, something you have, and something you are. An example would be authentication using a password and a software token or digital certificate.

3. Physical Multi-Factor Authentication

   http://schemas.openid.net/pape/policies/2007/06/multi-factor-physical

   An authentication mechanism where the end user authenticates to the OP by providing more than one authentication factor where at least one of the factors is a physical factor such as a hardware device or biometric. This policy also implies the Multi-Factor Authentication policy and both policies may be specified in conjunction without conflict. An example would be authentication using a password and a hardware token.

Token type	**Level 1**	**Level 2**	**Level 3**	**Level 4**
Hard crypto token	X	X	X	X
One-time password device	X	X	X	
Soft crypto token	X	X	X	
Passwords and PINs	X	X		

Table 12: Mechanisms used in corresponding NIST levels

Token type	**Level 1**	**Level 2**	**Level 3**	**Level 4**
On-line guessing	X	X	X	X
Replay	X	X	X	X
Eavesdropper		X	X	X
Verifier impersonation			X	X
Man-in-the-middle			X	X
Session hijacking				X

Table 13: Protection against threats based on different levels

5.3 OpenID Attribute Exchange 1.0

The OpenID Attribute Exchange extension services the exchange of identity information between endpoints. An attribute is a unit of personal identity information that is identified by a unique URI. It may refer to any kind of information.

You are able to specify your own attributes, although not all providers will be able to serve those requests. The attributes specified by the simple registration extension described in chapter 5.1 can be exchanged using this extension. Next to this, the website `axschema.org` services a list of semi-standardized attributes. Table 14 contains attributes specified by `axschema.org`.

Name	**Label**
`http://axschema.org/namePerson`	Full name
`http://axschema.org/namePerson/friendly`	Alias/Username
`http://axschema.org/namePerson/prefix`	Name prefix
`http://axschema.org/namePerson/first`	First name
`http://axschema.org/namePerson/last`	Last name
`http://axschema.org/namePerson/middle`	Middle name
`http://axschema.org/contact/phone/home`	Phone (home)
`http://axschema.org/contact/phone/business`	Phone (work)
`http://axschema.org/contact/phone/cell`	Phone (mobile)

Table 14: Attributes specified by `www.axschema.org`

This is only a selection of the attributes. Next to these, there are attributes which specify information about the address of the identity, the preferred language, various websites, instant messaging addresses and lots more[6].

There are two message types defined for exchanging attributes using this extension: `fetch` and `store`. `Fetch` retrieves attribute information from an `OP`, while `store` saves or updates attribute information on the `OP`. Both messages originate from the `RP` and are passed to the `OP` via the user agent.

[6]A full overview can be found at `http://www.axschema.org/types/`

6 OpenID on mobile phones

The primary focus of this research lays on integrating the OpenID authentication standard within the mobile version of the GravityZoo client. A mobile phone is a totally different environment comparing to a normal desktop computer. Differences with the biggest impact are described in paragraph 6.1.

6.1 Notable differences between PCs and mobile phones

A mobile phone has a relatively small screen. This screen is not designed for viewing web pages that are made to be displayed on normal computer screens. When authenticating using OpenID, you are required to log in at the website of the OpenID Provider of your choice. This could be difficult if this page contains large objects or many pictures.

Most mobile phones do not supply you with a full size keyboard which makes typing large quantities difficult. The OpenID provider can help in lowering the amount of data to be typed by automatically filling out the username field. This would enlarge the ease of use on a mobile phone.

Mobile phones come with stripped internet browsers. For example: Most mobile internet browsers contain only a limited size certificate store. Due to this, not all commercially bought certificates that are used on the Internet, are being trusted by default. When viewing pages that are secured by SSL, a user is often confronted with a security warning which indicates that the certificate used to encrypt the session, is not trusted. The user has to manually choose if he trusts the certificate before he can continue to the web page.

Summarizing the above to the following constraints:

- The website of the `OP` should be compatible (as far as possible) with the small screen of a mobile phone.
- The certificate used for the SSL connection should (preferably) be trusted by the mobile internet browser.
- `OP`s should lower the amount of typing needed by for example, automatically fill out the user field on the log in page.

6.2 Tested OpenID Providers

A handful of OpenID providers have been selected from http://openid.net/get/. The selected providers are generally recommended by various OpenID members [17]. Next to the recommended OpenID providers, two dutch ones have been added[7] to the selection, which can be found in table 15. As the OpenID foundation says:

> *In the end you should choose a Provider from a company which you trust.*

Provider name	Website	OpenID "username"
AOL	http://openid.aol.com	openid.aol.com/username
ClaimID	http://claimid.com	claimid.com/username
Google	http://google.com	google.com/accounts/o8/id
LiveID (INT)	http://live-int.com	live-int.com/username
LogMijIn	https://logmij.in	logmij.in/als/username
MijnOpenID	http://mijnopenid.nl	mijnopenid.nl/is/username
MyID	http://myid.net	username.myid.net
MyOpenID	https://www.myopenid.com	username.myopenid.com
MyVidoop	https://myvidoop.com	username.myvidoop.com
Verisign	https://pip.verisignlabs.com	username.pip.verisignlabs.com

Table 15: Tested OpenID Providers

LiveID-INT (http://login.live-int.com) is the Windows Live test cluster. The Windows Live OpenID Provider is still in the beta phase [22]. It is currently not possible to enable your own Windows Live ID for use with OpenID. You have to register a separate account within the Windows Live test cluster.

With the Google OpenID service, you do not receive a unique OpenID URL with which you can identify yourself. You will need to enter the URL google.com/accounts/o8/id in the OpenID login field to use your Google account as OpenID. Google wants to promote its own API, the "Google Federated Login API". This enables the use of Google accounts for authentication on third party websites [23]. I decided to include Google in this test because a lot of people have an account at Google. I'm hoping myself that they change their scenario and give each user a unique OpenID identifier.

6.3 Tested criteria

During the test, all OpenID providers listed in table 15 have been tested on the mobile phones listed in chapter 6.4. Criteria that have been looked in to during the test are summarized on the next page.

[7]The providers www.logmij.in and www.mijnopenid.nl have been added.

1. Does the OpenID provider website use SSL?
   - Do the mobile browsers trust this certificate by default?
2. Does the OpenID provider have a custom page for mobile users?
3. Is the login page easy to use on a mobile phone?
4. Does the OpenID provider automatically fill out the username field?

The test is done with a PHP script supplied by openidenabled.com[8] It can be used to test OpenID servers or OpenID enabled URL's. This script gives you the ability to log in using OpenID in `openid.mode` immediate and setup. The results of the test can be found in chapter 6.5.

6.4 Phones used during the test

The providers have been tested on several mobile phones with a different feature set. The following mobile phones have been used in the test:

1. **Nokia N91**. This phone runs on Symbian S60 and comes with the Nokia Mobile web browser. It does not have touch screen functionality or a full keyboard. The screen size is 2,1 inch and its resolution is 176*208 pixels.
2. **Nokia N96**. This phone runs on Symbian S60 and comes with the Nokia Mobile web browser. It has a bigger screen than the N91: 2,8 inch, it has a resolution of 240*320 pixels.
3. **HTC Touch Cruise**. This phone runs on Windows Mobile 6 and comes with Microsoft Internet Explorer Mobile. It has a touch screen and a full featured on-screen keyboard. Next to this it is equipped with a 2,8 inch screen with a resolution of 240*320 pixels.

The first plan was to test the mentioned OpenID providers with another phone: The **Nokia E71**. This phone has a 2,4 inch screen that has a resolution of 320*240 pixels. It has a This phone has a full size QWERTY-keyboard which should really ease the process of supplying your OpenID identifier and credentials. Sadly, this phone was not available during the time frame of this research project. I used another Nokia model as a replacement: the **Nokia N95**.

[8]The test page is located at http://openidenabled.com/resources/openid-test/.

6.5 The results

This chapter covers the results of the OpenID mobile test. After registering an OpenID at all OpenID providers the test could start. The first thing that leaps out in table 16 is that `MijnOpenID.nl` is not using SSL for session encryption. This is marked with a "-". All other tested providers do use SSL (marked with a "+"), which is recommended in the OpenID specification [2].

The Nokia N91 contains a very small certificate store. This results in that most certificates used by the OpenID Providers cannot be verified. Certificates that could not be verified are marked with a "-", the ones that could successfully be verified are marked with a "+". The other two phones were able to verify most of the used SSL certificates.

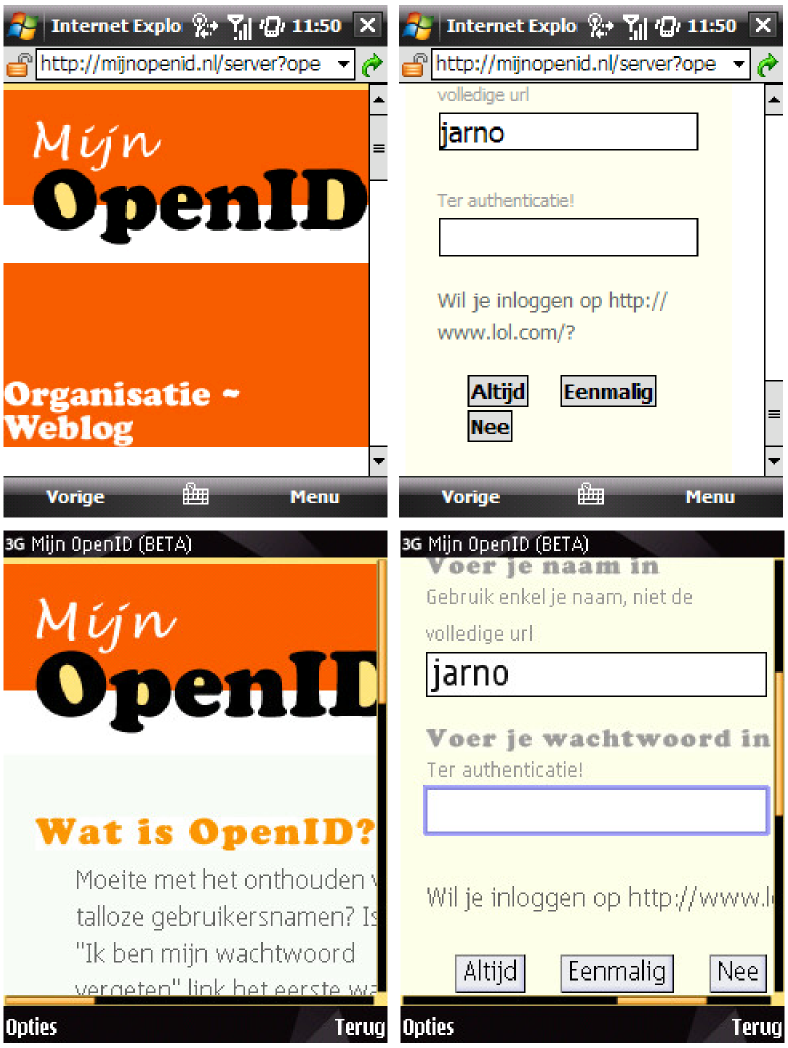
Most OpenID Providers automatically filled the username field after begin redirected to them. This eliminates the need to supply the username a second time, this is useful when authenticating with a mobile phone without a full size keyboard. The only way to use the Google OpenID provider is to specify the supplied general identifier. Due to this, the username field cannot be filled automatically.

Provider	**SSL**	**Certificate verified on**			**Auto fill username field?**
		N91	**N95/N96**	**HTC**	
AOL.com	+	-	+	+	+
ClaimID.com	+	-	+	-	-
Google.com	+	+	+	+	-
LiveID-INT.com	+	-	+	+	+
LogMij.in	+	-	-	+	+
MijnOpenID.nl	-	n/a	n/a	n/a	+
MyID.net	+	+	+	+	+
MyOpenID.com	+	-	+	-	+
MyVidoop.com	+	-	+	+	+
VerisignLabs.com	+	-	+	+	-

Table 16: OpenID provider mobile phone friendliness (part 1)

Concluding from table 17, there is no doubt that OpenID Providers are not focusing on mobile users at all. None of the tested OpenID Providers has a custom website for mobile users. This is marked with a "-" in the table. Due to the lack of not having a custom page for mobile users, the page layout is bad when displaying the page using a mobile browser. Some of the OpenID Providers have big images on the login page. This dramatically increases the page loading time when using your mobile internet connection.

The HTC which runs the mobile version of Internet Explorer manages to display some pages fairly okay. This because the mobile version of Internet Explorer it places all columns of internet pages below each other. A good example of this is `MijnOpenID.nl`. This provider has next to a big image in the banner, a lot of text on the log in page. First there is a lot of text explaining what OpenID essentially does, unluckily, the login field that has to be used is at the bottom of the page. On the Nokia phones there is no scaling at all on this page. You have to scroll a lot in both ways as can be seen in figure 6.5. The two screenshots in the top if the figure are taken on the HTC, the lower two on a Nokia N96.



volledige url

jarno

Ter authenticatie!

Wil je inloggen op http://www.lol.com/?

Wat is OpenID?

Moeite met het onthouden v
talloze gebruikersnamen? Is
"Ik ben mijn wachtwoord
vergeten" link het eerste wa

Voer je naam in

Gebruik enkel je naam, niet de volledige url

jarno

Voer je wachtwoord in

Ter authenticatie!

Wil je inloggen op http://www.l

Figure 6: MijnOpenID.nl in both browsers

A selection of screenshots taken from various `OpenID Providers` on both devices, can be found in appendix A.

The provider ClaimID.com scales reasonable as well on the mobile version of Internet Explorer. Despite the fact that the login page contains a logo beneath the page header, there is no need for a lot of scrolling to be done.

A login page is rated with a "+" if no scrolling at all is needed and it does not contain any large images. Pages on which you can see the login box at first load, but which need small amounts of scrolling in one way are marked with a "+/-". Pages on which you do not see the login box at all at first load, are marked with a "-". Pages which need a large amount of scrolling in both ways are marked with "- -".

The Nokia N91 is not at all suitable for web browsing as can be concluded from this test. Due to its low resolution and small screen, none of the pages scaled well on this phone.

Provider	**Custom page on**			**Page layout**		
	N91	**N95/N96**	**HTC**	**N91**	**N95/N96**	**HTC**
AOL.com	-	-	-	- -	-	-
ClaimID.com	-	-	-	- -	+/-	-
Google.com	-	-	-	- -	-	+/-
LiveID-INT.com	-	-	-	- -	-	+/-
LogMij.in	-	-	-	- -	-	+/-
MijnOpenID.nl	-	-	-	- -	- -	-
MyID.net	-	-	-	- -	-	-
MyOpenID.com	-	-	-	- -	-	-
MyVidoop.com	-	-	-	- -	- -	- -
Verisignlabs.com	-	-	-	- -	+/-	-

Table 17: OpenID provider mobile phone friendliness (part 2)

During the tests, Pip.Verisignlabs.com did not allow a Relying Party to redirect the browser to them for authentication purposes. When begin redirected to Verisign, you get a message that redirection is not allowed. After this, you have to navigate yourself to https://pip.verisignlabs.com/login.do. I tested the providers again in during the last week. During this last test, Pip.Verisignlabs.com allowed redirection again. The Verisign OpenID login page neither scales on the Nokia phones, nor on the HTC.

Logging in by using the MyVidoop.com OpenID Provider did not succeed on any of the phones. When visiting the MyVidoop login page for the first time, it detects that this browser is not used to authenticate before. On a normal computer, it pop ups a overlay window using JavaScript/Ajax technology which thats that the browser needs to be activated first. This can be done by e-mail or SMS. The window did not pop up on any of the phones so the browser can not be activated.

When logging in at `MyOpenID.com` using a Nokia phone, you get redirect to the French version of the page. None of the Nokia phones were actually set to use French as primary language. Due to the latter, `MyOpenID.com` is rated with a "-". Using Internet Explorer Mobile, you are redirected to the normal English version of the login screen. Though, you do not see the login box at first load. Due tho this, the page is rated with a "-" aswell.

6.6 Conclusion on mobile-friendliness

From this test can be concluded that none of the tested OpenID providers is currently focusing on mobile users. None of the login pages is *really* mobile browser friendly. I'm hoping that they will optimize their pages in the near future.

It should not be that difficult for the OpenID providers to create a lightweight version of their page for mobile users. The OpenID providers can check the browser version of the end user visiting the website. Based on this, they can choose to display the lightweight version of the page or the normal page.

7 Integrating OpenID

This chapter covers the implementation requirements of OpenID authentication within the mobile version of the GravityZoo framework.

Until now, OpenID is only used for authentication on internet pages. GravityZoo is the first who actually integrates OpenID authentication within their own application framework. As described in chapter 4, OpenID uses the HTTP protocol for communication between the endpoints. For this reason, the application framework has to be able to send HTTP requests and process incoming HTTP messages.

The GravityZoo company wants me to focus on the implementation of the OpenID authentication mechanism, and not include the extensions for now. Extensions can be looked in to when doing further research on the subject.

There are two main roles in the OpenID authentication process as described in chapter 3.4: The `Relying party` to which the user wants to authenticate and the `OpenID provider` which actually provides the authentication service. GravityZoo wants to enable its users to authenticate with their own OpenID. They currently have no intention to become an `OpenID provider` their selves. This because there is a large chance that users currently already have an OpenID - maybe without even knowing - is big.

7.1 Where to place the Relying Party entity?

The most important question to be answered is: Which server role in the GravityZoo framework should inherit the role of being the `Relying Party`.

The `Relying Party` server role has the following requirements:

- Needs to be able to construct, send and process HTTP messages.
- Needs to be able to communicate with `OpenID providers` on the Internet for establishing associations needed for signing purposes, or to verify signatures if no association could be established.
- Needs to be able to store association information to be able to verify signatures on incoming responses from `OpenID Providers`. Next to this, the associations are reused for future communication with this `OpenID Provider`.

- Should be located in a trusted environment, only accepting incoming authentication requests from GravityZoo clients using the REP routing infrastructure.
- Needs to be able to communicate with the GravityZoo `ALS` which provides the authorization.

Roughly there are the following two scenarios to think of here:

1. Create a new server role for handling OpenID authentication.
2. Extend the current Authentication and Licensing Servers with the `Relying Party` role.

Both scenario's have its pros and cons. The second scenario would mean that the `ALS` has to host a webserver. This makes the `ALS` reachable from the Internet. It would become a point to attack the framework on because it handles all authentication, authorization and licensing for the whole framework. Due to this reason, this scenario is not a good choice.

The other option is to implement the `Relying Party` on a new server within the framework. It has to create associations with `OpenID Providers`. These associations need to be stored in a trusted environment. The role cannot be hosted in the trusted part because of the demand of a webserver.

A better alternative is to split the webserver from the rest of the methods. This scenario is reviewed in chapter 7.2, together with its requirements for each step.

7.2 Relying Party process overview

A `Relying party` needs to go through several steps when authenticating an end user. The steps summarized below, are numbered with the corresponding paragraphs that contain information about this specific part of the procedure.

7.2.1 The end user needs to be able to supply its OpenID identifier to the GravityZoo framework (as covered in chapter 4.1). A field servicing the OpenID identifier should be created on the login screen.

7.2.2 When clicking the login button, the GravityZoo framework should be notified that OpenID is being used for authentication. The authentication request should be routed to the entity within the framework that acts as `Relying Party`.

7.2.3 The `supplied identifier` has to be normalized. After this, the `OpenID Provider` used by the end user has to be discovered (covered in chapters 4.2 and 4.3).

7.2.4 Since the GravityZoo client is not web-based, an Internet browser has to be opened on the end user's device. The browser has to be redirected to the login page of the `OpenID Provider`. The URL that is used to redirect the user to the `OpenID Provider` has to be constructed by the `Relying Party`.
The URL contains the authentication request as well as the URL to which the end user should be redirect to once authentication is finished (covered in chapter 4.5). Under the bonnet, the GravityZoo framework has to create an association with the `OpenID Provider` (covered in chapter 4.4).

7.2.5 In the normal authentication process, the `OpenID provider`'s website redirects the browser back to the website of the `Relying Party` which is requesting authentication. The redirect contains the provider's response to the authentication request (chapter 4.7). This response needs to be processed and verified by the GravityZoo framework.

7.2.6 After verifying the response from the `OpenID Provider`, the end user has to be granted access to all resources he is authorized to use.

7.2.1 Modifications to the login screen

Currently, the only way to log in to the GravityZoo framework is by supplying your username and password through the client. A new field has to be added to the login screen, in which you can supply your OpenID identifier.

OpenID identifiers can be relatively long. My Verisign OpenID identifier is a good example: `jarnovandemoosdijk.pip.verisignlabs.com`. Supplying such an identifier on a mobile phone without a full-size keyboard takes long.

There are two ways to increase the ease of use of this service. The first one is adding a "`remember my OpenID`" option into the GravityZoo client, like it is common practise at username/password logins. This option requires that the mobile browser supports cookies and that cookie support is enabled.

In the second scenario two boxes are being used: One text box and one drop-down list box. The first box is used for entering just the username part of the OpenID identifier. The latter contains the n most popular OpenID providers and an option for "`other provider`", in case your provider is not listed. Statistics of the used identifiers during login, can be used to create a representative selection of the most popular `OpenID Providers` used when signing in to the GravityZoo framework. The "`remember my OpenID`" option from the first scenario, can be added to this scenario too. This substantially cuts down the amount of typing required when logging in.

7.2.2 Supplying the identifier

Currently, when logging with a username and password, the login request is routed through the REP routers to the `ALS`. This request uses a special REP packet type as described in the last part of chapter 2. To fit OpenID into this scheme, an additional packet type has to be specified so that the GravityZoo framework knows that OpenID authentication is being used.

The identifier is used to discover and associate with the `OpenID Provider`. The association has to be created and stored in a trusted environment. Since the newly added webserver is not in a trusted environment, the authentication request has to be forwarded to the `ALS`.

7.2.3 Normalizing the identifier and Discovering the OP

After the end user has supplied its OpenID identifier, the `ALS` has to process the `supplied identifier`. The identifier is first normalized as described in chapter 4.2. The normalization procedure has no specific requirements. The identifier is processed locally.

The next step for the `Relying Party` is to discover the `OpenID Provider` that is servicing authentication for the `supplied identifier`. This information is discovered by using one of the three discovery methods described in chapter 4.3. Discovering the `OpenID Provider` requires internet connection and the ability to send `HTTP GET` requests to the normalized identifier.

7.2.4 Redirecting the user

Before the end user is redirected, an association is established between the `Relying Party` and the `OpenID Provider` (covered in chapter 4.4). The association request is sent by the `ALS` to the discovered `OpenID Provider`. The most important keys of this request are the encryption to be used, the Diffie-Hellman modulus and generator to be used, next to the computed `g^xa mod p`. This where `g` is the generator, `xa` is the secret integer chosen by the `ALS` and `p` is the specified modulo.

The `OpenID Provider` responds to this request with the encrypted shared secret and the value of `g^xb mod p`. The computed shared secret is used to encrypt the association's MAC key in transit. Details of this process are covered in chapter 4.4. This step requires processing power for the Diffie-

Hellman computations and a communication path to the `OpenID Provider`.

The association has to be stored together with the corresponding handle, shared secret and the lifetime of the shared secret. If the value is not stored, it is not possible to verify any signed messages sent by the `OpenID Provider`. The association is used for all sessions to this provider until its expiration time invalidates the handle.

After establishing an association, the URL to which the end user is redirected is constructed from several keys. Background information on all keys can be found in 4.5. One of these keys is the `openid.mode`. This key specifies one of the following two modes that can be used: `checkid_immediate` or `checkid_setup`.

The most userfriendly scenario is to try `checkid_immediate` mode first. In immediate mode, no interaction with the end user is needed. This only succeeds if the user is already logged in to its `OpenID Provider`, and permanently authorized the GravityZoo framework to request authentication for its OpenID identifier. The `OpenID Provider` is required to answer the request immediately. If this mode fails, a new request has to be placed in `checkid_setup` mode. After the user has logged in at the `OpenID Provider` and granted GravityZoo permanent permission to request authentication, the user does not have to supply its password to log in to the framework, until the login at the `OpenID Provider` for the corresponding session expires.

Another important key used in the request is `openid.return_to`. The end user is redirected to this URL after having authenticated at the `OpenID Provider`'s website. This is covered in chapter 7.2.5.

As described in chapter 7.2, the GravityZoo framework is not web-based. During the normal OpenID login procedure, the end user's internet browser is redirected to the website of his `OpenID Provider`. An internet browser needs to be launched on the end user's device. This browser needs to be redirected to the login page of the `OpenID Provider`.

This scenario has an advantage over normal OpenID authentication. Some `OpenID Providers` do not allow a `Relying Party` to redirect the end user the their login page (covered in chapter 6.5). There is no browser in the first place, so the user can normally log in using the spawned browser, without manually surfing to the `OpenID Provider`'s login page.

7.2.5 Intercepting and verifying the response from the OP

The end user now has to supply its credentials at the `OpenID Provider` and authorize the GravityZoo framework to receive the authentication response. After this, the user is redirected back to the URL specified in the `return_to` key.

The `return_to` key should contain the URL of the webserver installed to receive the response from the `OpenID Provider`, as covered in chapter 7.1. The URL to which the user is redirected contains keys that complete the authentication request done by the `ALS`. These keys, also known as the assertion from the `OpenID Provider`, need to be verified. Verification can be done in two ways, as covered in chapter 4.7: Indirect or direct. Indirect verification is used when an association is established with the `OpenID Provider`.

The webserver that received the keys has no access to the trusted environment where the assertion is stored. This assertion is needed to verify the signature on the keys signed by the `OpenID Provider`. The keys need to be forwarded to the `ALS` for verification. The `ALS` can then verify the keys and their signatures. Direct verification requires the `ALS` to send a signature verification request to the used `OpenID Provider` (covered in chapter 4.7.4).

After the verification process has ended successfully, the `ALS` should inform the webserver that the process has been finished. The user should be redirect to a web page stating something like the quoted text below.

"You now have been authenticated successfully, click [here] to close this window and return to the GravityZoo client."

To optimize this page for mobile usage, it should not contain any large images or other large objects. Authorization which is needed to access the applications in the framework is covered in the next paragraph.

7.2.6 Authorizing the end user

OpenID just covers authentication and no authorization. Authorization is handled by the `ALS` server role within the GravityZoo framework.

To authorize users that are using their OpenID instead of a username/-password combination, an extra record has to be added to each user in the user database. This entry should contain the OpenID corresponding to that user.

Since the `ALS` is verifying the response from the `OpenID Provider`, authorization can then be handled in the same way as with username/password authentication. The authorization response to the client should refresh the client, this gives the user the ability to see which programs he is authorized access.

7.3 The code

After research I found out that there are full feature OpenID libraries available for a wide variety of programming languages. An overview of the available libraries can be found at [24].

GravityZoo is programmed in Python. Python libraries are available at `openidenabled.com`. They come with documentation and implementation examples created by the OpenID community. These libraries are used for the integration of OpenID into a website. The GravityZoo system is not a website: it's an application framework. Though, some code can be reused, e.g. the code for discovering the `OpenID Provider` and creating the association can be reused.

Due to time limitations I haven't been able to start with a POC integration of the protocol. Next to this, I would need to have some support of a GravityZoo programmer. I have been told that they are currently on a very tight schedule, so this was not possible.

8 Conclusion

Reviewing my research questions as stated in the last part of chapter 1.1:

How does OpenID work?

The working of the OpenID protocol is covered in detail in chapter 4 of this report. This was needed to specify the requirements for integrating OpenID in to the GravityZoo framework, and for my personal interest in the concept.

How mobile phone friendly are the most popular OpenID Providers?

From the test results that can be found in chapter 6.5 can be concluded that none of the `OpenID Providers` is currently focusing on its mobile users. None of the providers have a custom lightweight page for mobile users. The providers should create a lightweight version of their page to service its mobile users.

The user friendliness of providers like Google, LiveID and LogMij.In is fair when using a phone with the mobile version of Internet Explorer, When using a Nokia N95 or N96, ClaimID and Verisign provide reasonable user friendliness, though, most providers require scrolling in both browsers.

What are the requirements for integrating OpenID into the GravityZoo framework?

GravityZoo wants its users to be able to log in with their OpenID identifier. This means that the OpenID Relying Party role needs to be implemented. Detailed information on the integration requirements can be found in chapter 7. In short, the following needs to be done.

The Relying Party role needs to store session information like shared secret keys in a trusted location and it has to be reachable from the internet. The latter makes it impossible to integrate the whole role on the GravityZoo Authentication and Licensing server role. This because the web server creates an easy way to reach and possibly attack the ALS role in the GravityZoo framework. If this role is compromised, all authentication and authorization can be stolen or modified.

Due to the above reason, the web server has to be separated from the rest of the OpenID Relying Party role. This way, the web server can be placed outside the trusted environment. The rest of the Relying Party role can be placed on the Authentication and Licensing server. The `ALS` processes the supplied OpenID identifier and discovers the OpenID provider. After this, it places an authentication request at the OpenID Provider. It specifies the address of the web server in the `return_to` key of the request. The user will be redirected to the web server after authenticating at his OpenID Provider. The only task of the web server is to relay the authentication response that comes from the OpenID Provider, to the Authentication and Licensing server. The `ALS` can then verify the received response and authorize the specific user. A new field which stores the OpenID identifier has to be added to each user in the database. This way the `ALS` can authorize users that use OpenID to authenticate, the same way as users that log in with a combination of username and password.

9 Future considerations

In this report I focussed on the working of OpenID and on the requirements for its implementation into the GravityZoo framework. I have not spend any time on the ethics involved with the protocol or with trusting an `OpenID Provider`.

I have not done any in depth research on how secure the protocol itself is. Although, during the first week when reading about OpenID, I came across an enormous amount of news items and forum threads about phishing attacks on OpenID [25] [26] [27] [28] [29]. Future research can be done on the security of the protocol, including looking in to the implementation of the extensions.

10 Acknowledgments

I would like to thank the following people for their help while conducting this research project.

- **Marcel van Birgelen**, for his guidance throughout the research project.
- **Dirk Moors**, for sharing his view and knowledge about the GravityZoo framework.
- **Mahdi Abdulrazak**, for bringing me into contact with GravityZoo.
- And lastly, those who gave their feedback on this document.

References

[1] Marcel van Birgelen, *GravityZoo Technical Overview*
http://www.gravityzoo.com/support/GZF_techoverview_2_1_rc2.pdf

[2] Brad Fitzpatrick, David Recordon, Josh Hoyt, *OpenID terms*
http://openid.net/specs/openid-authentication-2_0.html

[3] T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier (URI)*
http://www.ietf.org/rfc/rfc3986.txt

[4] OpenID Foundation, *OpenID Foundation*
http://openid.net/foundation/

[5] OpenID Foundation, *OpenID Intellectual Property*
http://openid.net/foundation/intellectual-property/

[6] David Chappell, *Introducing Windows CardSpace*
http://msdn.microsoft.com/en-us/library/aa480189.aspx

[7] Internet2 Intellectual Property Framework, *The Shibboleth System*
http://shibboleth.internet2.edu/

[8] Bart Kerver, *The innovative authentication system*
http://a-select.surfnet.nl/

[9] Dick Hardt, *CardSpace / OpenID Collaboration Announcement*
http://www.identityblog.com/?p=668

[10] Google, *Federated Login for Google Account Users*
http://code.google.com/intl/nl/apis/accounts/docs/OpenID.html

[11] Larry Drebes, *Relying Party Stats as of Jan 1st, 2009*
http://blog.janrain.com/2009/01/relying-party-stats-as-of-jan-1st-2008.html

[12] Drummond Reed, Dave McAlpin, *Extensible Resource Identifier (XRI) Syntax V2.0*
http://www.oasis-open.org/committees/download.php/15376

[13] Brad Fitzpatrick, David Recordon, Josh Hoyt, *OpenID Authentication 2.0 - Final - Appendix A*
http://openid.net/specs/openid-authentication-2_0.html#normalization_example

[14] Justen Stepka, *Using OpenID*
http://www.theserverside.com/tt/articles/article.tss?l=OpenID

[15] Gabe Wachob, Reed Drummond, *Extensible Resource Identifier (XRI) Resolution Version 2.0*
http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.html

[16] Joaquin Miller, *Yadis*
http://yadis.org/papers/yadis-v1.0.pdf

[17] OpenID Foundation, *How do I get an OpenID?*
http://openid.net/get/

[18] Will Norris, *OpenID Support*
http://willnorris.com/openid-support/

[19] OpenID Foundation, *OpenID Simple Registration Extension 1.0*
http://openid.net/specs/openid-simple-registration-extension-1_0.html

[20] OpenID Foundation, *OpenID Provider Authentication Policy Extension 1.0*
http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html

[21] W. Burr, D. Dodson, W. Polk, *Electronic Authentication Guideline*
http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf

[22] Dare Obasanjo, *Windows Live is now an OpenID Provider*
http://www.25hoursaday.com/weblog/2008/10/27/WindowsLiveIsNowAnOpenIDProvider.aspx

[23] Google, *Federated Login for Google Account Users*
http://code.google.com/intl/nl/apis/accounts/docs/OpenID.html

[24] OpenID Foundation, *OpenID Libraries*
http://wiki.openid.net/Libraries

[25] Marco Slot, *Beginner's guide to OpenID phishing*
http://marcoslot.net/apps/openid

[26] Simon Willison, *Solving the OpenID phishing problem*
http://simonwillison.net/2007/Jan/19/phishing

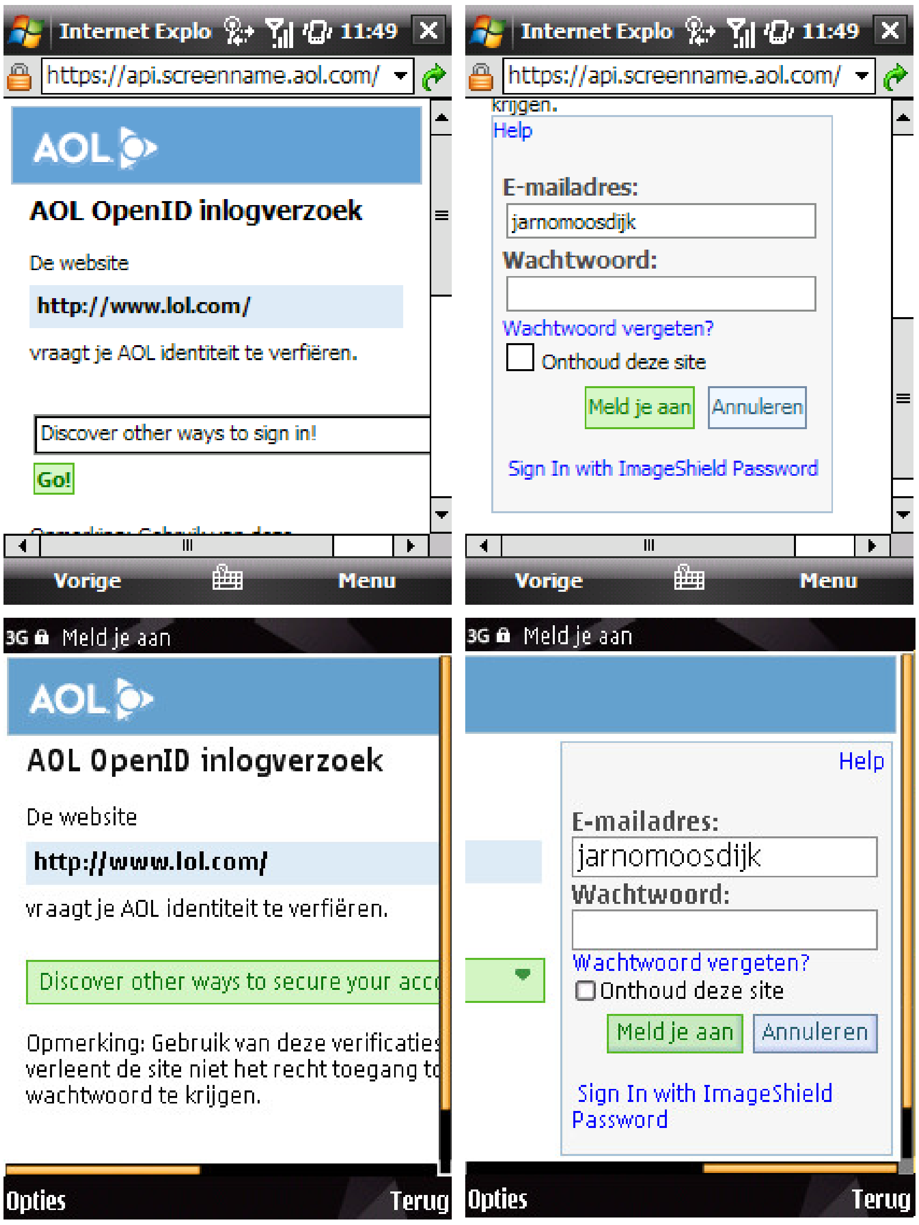
[27] Links.org, *OpenID: Phishing Heaven*
http://www.links.org/?p=187

[28] Links.org, *OpenID and Phishing: Episode II*
http://www.links.org/?p=188

[29] Yoeri Lauwers, *OpenID beschikbaar in Nederland*
http://tweakers.net/nieuws/47158/openid-beschikbaar-in-nederland.html

A Screenshots taken from OpenID Providers

This appendix contains screenshots of the tested `OpenID Providers` taken on the HTC and the Nokia N96. The two first screenshots in each figure are taken on the HTC, the lower two are taken on the Nokia N96.


Internet Explo 11:49
https://api.screenname.aol.com/
AOL OpenID inlogverzoek
De website
http://www.lol.com/
vraagt je AOL identiteit te verfiëren.
Discover other ways to sign in!
Go!
Vorige Menu

Internet Explo 11:49
https://api.screenname.aol.com/
krijgen.
Help
E-mailadres:
jarnomoosdijk
Wachtwoord:
Wachtwoord vergeten?
Onthoud deze site
Meld je aan Annuleren
Sign In with ImageShield Password
Vorige Menu

3G Meld je aan
AOL
AOL OpenID inlogverzoek
De website
http://www.lol.com/
vraagt je AOL identiteit te verfiëren.
Discover other ways to secure your acc
Opmerking: Gebruik van deze verificaties
verleent de site niet het recht toegang to
wachtwoord te krijgen.
Opties Terug

3G Meld je aan
Help
E-mailadres:
jarnomoosdijk
Wachtwoord:
Wachtwoord vergeten?
Onthoud deze site
Meld je aan Annuleren
Sign In with ImageShield Password
Opties Terug

Figure 7: AOL.com in both browsers

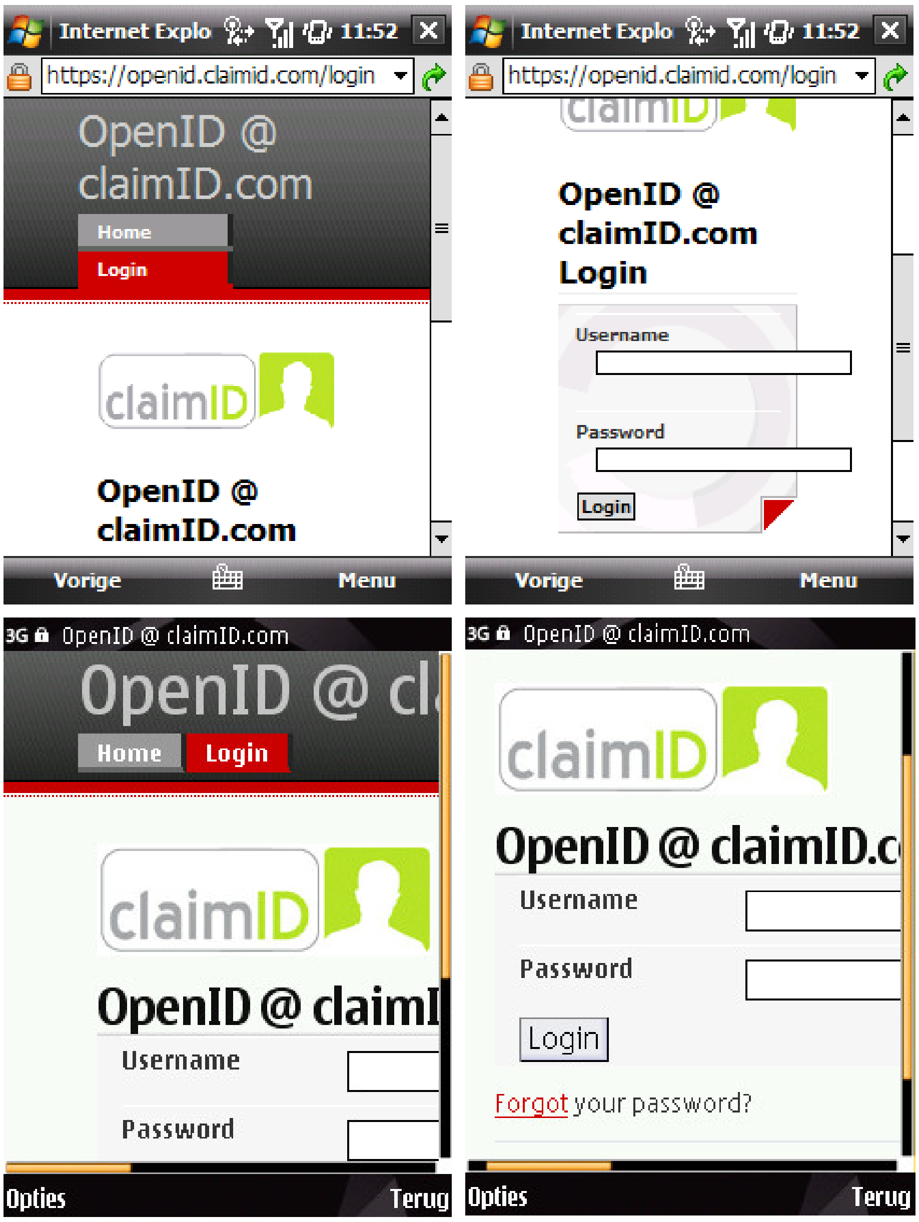


Figure 8: ClaimID.com in both browsers

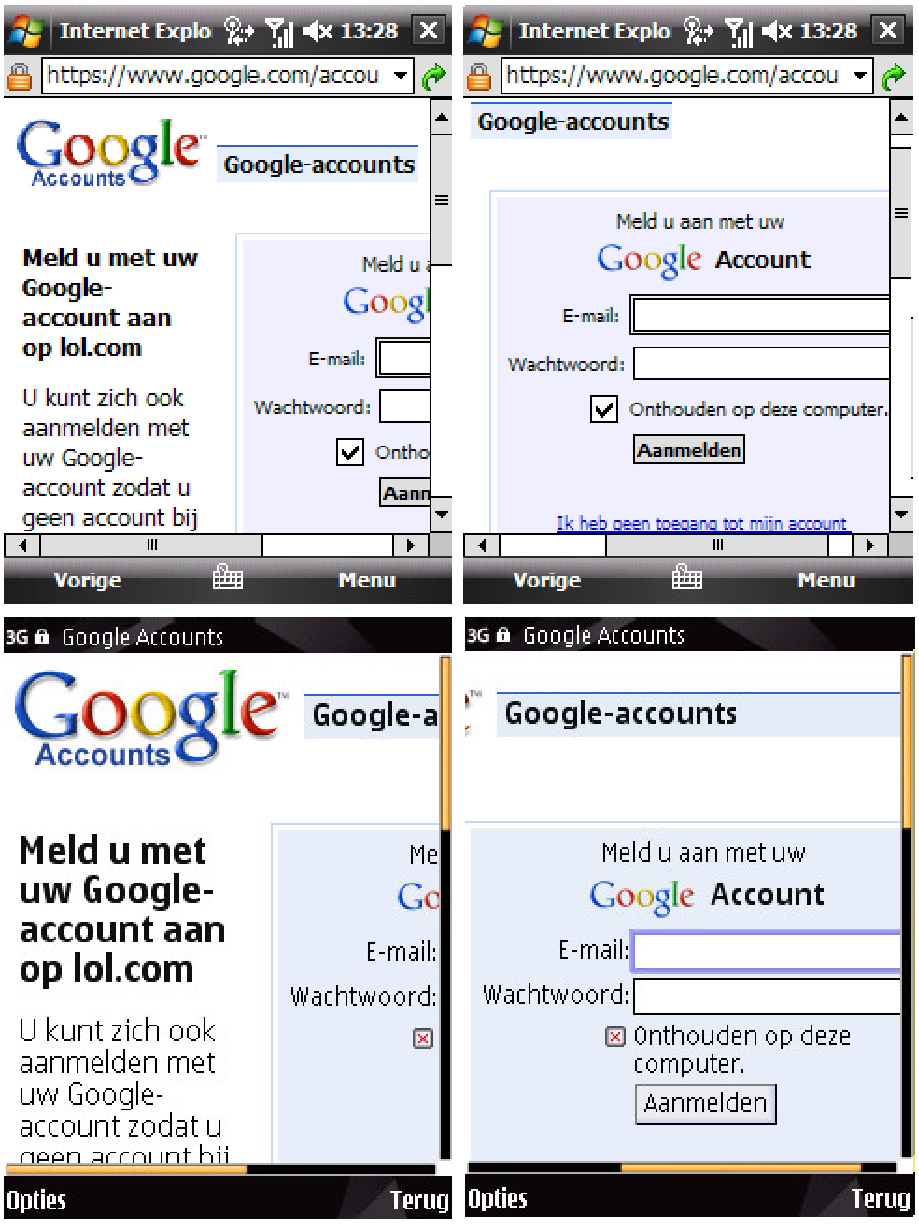
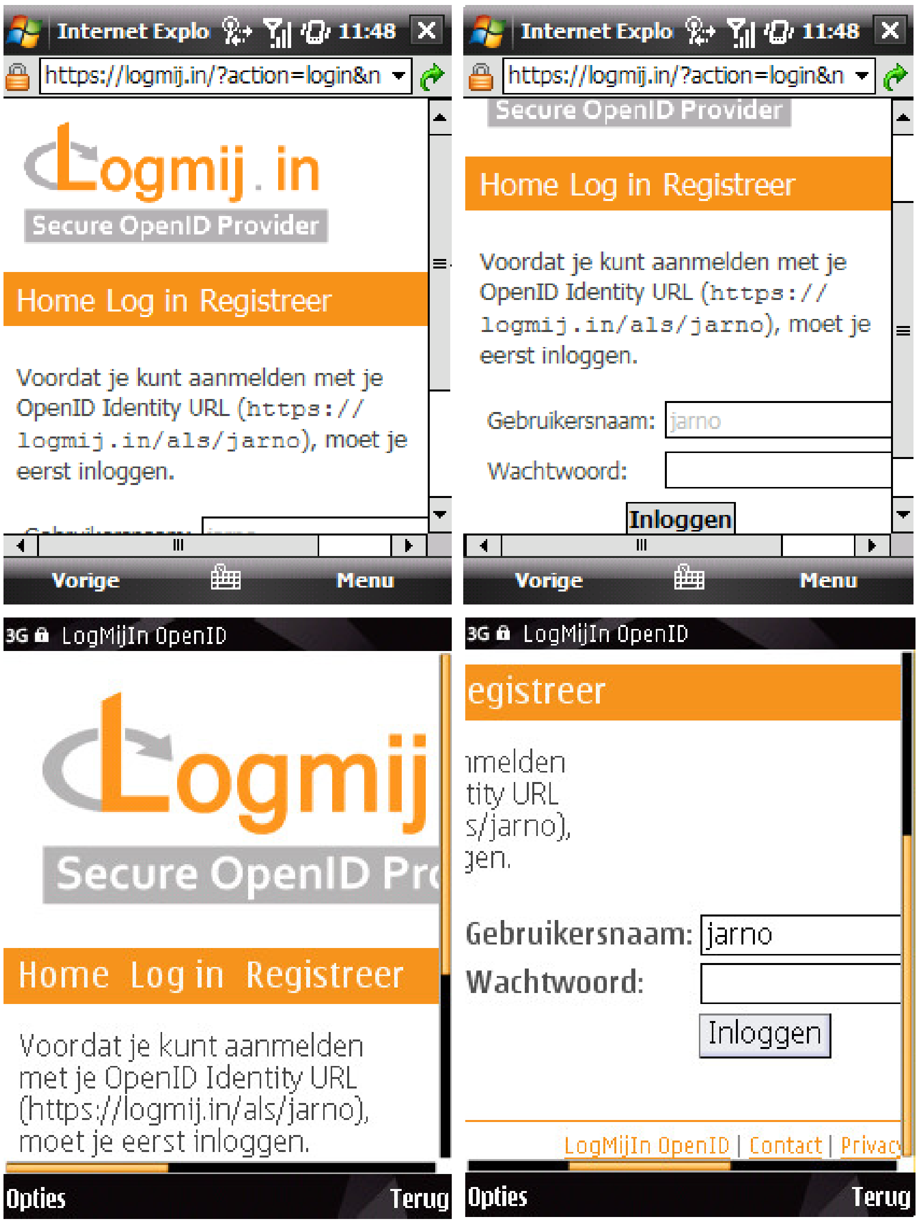

Internet Explo 13:28
https://www.google.com/accou
Google Accounts
Google-accounts
Meld u met uw Google-account aan op lol.com
U kunt zich ook aanmelden met uw Google-account zodat u geen account bij
Meld u
E-mail:
Wachtwoord:
Ontho
Aann
Vorige
Menu
Internet Explo 13:28
https://www.google.com/accou
Google-accounts
Meld u aan met uw Google Account
E-mail:
Wachtwoord:
Onthouden op deze computer.
Aanmelden
Ik heb geen toegang tot mijn account
Vorige
Menu
3G Google Accounts
Google Accounts
Google-a
Meld u met uw Google-account aan op lol.com
U kunt zich ook aanmelden met uw Google-account zodat u geen account bij
Me
E-mail:
Wachtwoord:
Opties
Terug
3G Google Accounts
Google-accounts
Meld u aan met uw Google Account
E-mail:
Wachtwoord:
Onthouden op deze computer.
Aanmelden
Opties
Terug

Figure 9: Google.com in both browsers



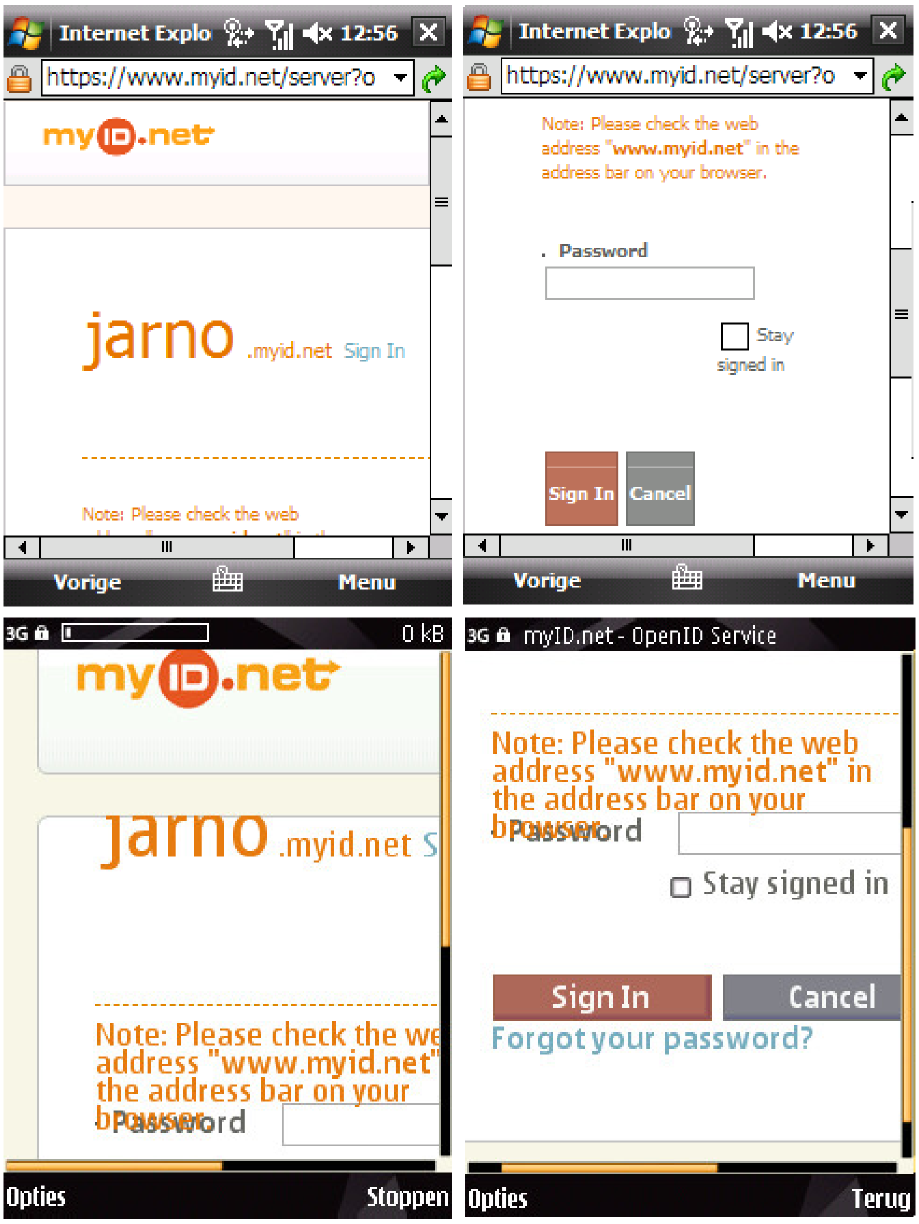
Voordat je kunt aanmelden met je OpenID Identity URL (`https://logmij.in/als/jarno`), moet je eerst inloggen.

Gebruikersnaam: jarno

Wachtwoord:

Inloggen

Figure 10: LogMij.in in both browsers



Note: Please check the web address "www.myid.net" in the address bar on your browser.

Figure 11: MyID.net in both browsers