Problem Description

User-centric identity management (IdM) have several benefits compared to traditional service-centric IdM. Among them, OpenID allows a user to manage own identities from a privately hosted OpenID Identity Provider (OP). That way, information data associated with an identity is stored and governed at the user's discretion, thus eliminating dependencies on the alternative third party OP. However, for the average user, the level of difficulty is too high for setting up such an OP, something which limits most people to enjoy this feature. The appearance and functionalities of modern smartphones have opened up new possibilities of hosting such an OP as a native mobile application, making acquisition and use very easy. Despite its simple use, the implementation itself needs to deal with challenges such as the authentication method, network reachability, security issues such as phishing and identity backup capabilities.

The master thesis will address the problem of hosting a private OP by implementing it as a smartphone application. The work will be a continuation of an in-depth study on OpenID and the resulting proposals to solve the network reachability problem, that took place in the Fall semester of 2010. The work will include user testing and analysis.

Assignment given: 24.01.2011
Supervisor: Do van Thanh

Summary

In the area of identity management OpenID is an identity system allowing users to log in to OpenID-enabled web sites by proving ownership of an OpenID Identifier by authenticating with its controlling OpenID Identity Provider. A user can choose to host an OpenID Identity Provider herself or trust in existing third-party providers such as Google. Technical skill is required for the former, leaving it unavailable for the average user.

This thesis simplifies the matter by implementing an OpenID Identity Provider as a smartphone application, making use of the traditional server-like features inherent in such devices. New possiblities for authenticating the user arise as she is enabled to physically interact with the OpenID Identity Provider, which in the traditional scheme is performed through the web browser. As a result from these new possiblities, phishing attacks are claimed to be avoided and identity attributes are exempted from being controlled and possibly exploited by any third-party.

One of several technical challenges include enabling the smartphone to receive inbound connections as this is required by the OpenID Authentication protocol, but restricted by telecom operators by default. Functionality must be in place to backup identity repositories stored on the smartphone in order not to lose possession of the established OpenID identities if the device becomes lost or damaged. Lastly, focus is given to make the solution easily applicable for even the novice consumer.

Preface

The following Master Thesis is the result from five interesting and fun months marking my final chapter of a five year study program in Communication Technology at the Department of Telematics at the Norwegian University of Science and Technology (NTNU).

The thesis consists of two parts

1) this report
2) an archive file (i.e. zip-file) containing the implementations that makes up the system

The greater part of the work has been spent on the actual implementation.

Acknowledgements

Firstly, I would like to thank my family and good friends for great support during my years of study.

Secondly, I would like to thank my supervisor, professor Do van Thanh, for giving me freedom to choose this topic within the identity management domain, as well as his support.

Eirik Stien
June 2011

Contents

i. List of Figures

ii. List of Tables

iii. List of Abbreviations

AES	Advanced Encryption Standard
APN	Access Point Name
CSRF	Cross-Site Request Forgery
EAP-SIM	Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules
EDGE	Enhanced Data rates for GSM Evolution
GGSN	Gateway GPRS Support Node
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IdM	Identity Management
JSON	JavaScript Object Notation
MAC	Message Authentication Code
MS	mOpenID Server
OP	OpenID Identity Provider
PII	Personal Identifiable Information
QoS	Quality of Service
RP	Relying Party
RSA	Rivest, Shamir, and Adleman (public key encryption technology)
SA	Smartphone Application
SAML	Security Assertion Markup Language
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module
SP	Service Provider
SSL	Secure Sockets Layer
SSO	Single Sign-On
TLS	Transport Layer Security
UA	User Agent
UMTS	Universal Mobile Telecommunications System

URL	Uniform Resource Locator
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity
WS	Web service(s)
XRDS	Extensible Resource Descriptor Sequence
XRI	eXtensible Resource Identifier

1. Introduction

Digital identity is a concept all Internet users touch upon, knowingly or not, and in a larger or smaller scale. Most users register for various web services, and during this process give up a lot of personal information required by the service provider. As such information is stored with the service provider, it contributes to a user's identity for that service alone. Further, users have to authenticate themselves, typically with a username and password, to the service provider in order to access a particular service. As the list of service providers a user registers to grows, so does (or at least should) the list of different usernames and passwords associated with different services. Unfortunately, users often tend to use the same or tightly related usernames and passwords for new services they sign up for, hence protection towards identity theft is weakened, among others [23].

All of the above resemble the traditional relation people have with using Internet services. The problems are several. Seen from a user's perspective it is inconvenient to sign up for new services, where each are requiring registration of the same or similar array of personal information, including a unique identifier accompanied with a secret password. Scalability becomes a problem, as humans are not capable of remembering an ever-growing list of identifiers and secure passwords. Also, managing one's identity seems unfeasible as one small change in the identity characteristics must be manually updated at each account with the service providers.

There have been several proposals to enhance usability and security with online identity management in means of both simplifying the authentication process and strengthening its security. Among others, OpenID is a highly user-centric identity management solution, which has enjoyed widespread adoption since its launch in 2005 [29]. This thesis aims to make possible some of the benefits gained from using OpenID for a larger audience.

In OpenID identity management an End User needs only trust an OpenID Identity Provider (OP) with her digital identity. Any OpenID-enabled service provider (SP), henceforth a Relying Party (RP), will accept assurances from the OP associated with a given OpenID Identifier. The End User presenting the OpenID Identifier to the RP needs only authenticate with the OP, which makes the details concerning this procedure irrelevant for the RP. The detailed functioning of the protocol will be described in section 3.2.2.

A user can choose to trust any OP they wish, but in doing so the user must trust the selected OP with a lot of personal identifiable information (PII). A more demanding but user controlled solution is for an End User to implement and manage an OP herself, as the OpenID protocol is an open specification allowing for this. To achieve the OP setup the End User is must possess technical skills of a certain degree, leaving it as too big of a challenge for the average Internet user.

This thesis will address the issue of the current procedures and technical dif-

ficulties associated with setting up private OPs. By making this setup procedure simple enough for any average Internet user, one can achieve an enhanced user-centric identity management (IdM) solution. This is argued as even the attributes associated with a certain OpenID Identifier are entirely managed and stored by equipment owned and physically carried by the user on a daily basis. By taking advantage of the processing power, communication technologies, user friendliness and increased adoption of modern smartphones, this thesis will suggest and implement a solution that satisfies the necessary requirements for an OP.

This thesis will start by introducing the reader to the concept of digital identity. Afterwards, a thorough study on the OpenID framework will be presented, as well as an elaboration of current challenges and weaknesses. As the OpenID authentication procedure and requirements are entirely left up to the OP to decide, the thesis will study some authentication mechanisms suggested in previous work. Based on the above this thesis will include the design and implementation of a smartphone application fulfilling the requirements of a private OP. In the end limitations and security issues of the solution will be discussed.

2. Digital identity

Since this thesis's main concern revolves around digital identity management, it is important to understand what digital identity is. This section, derived from [38] and [10], defines different terms and properties that most commonly take part in the concept of digital identity.

2.1. Entity

An entity constitutes the physical person or organization in an identity system and is thus unique. It is typically who or what humans visualize as being in possession of an identity. Shared entities may exist as well as e.g. a family can act as one entity to an IdM system, but still consist of several individuals.

2.2. Identity

An identity acts as a representation of an entity in a specific application domain, hence it must be unique within that domain. An entity can have several identities both in one and across several domains. For instance can a person both study and teach classes at a university, thus having two distinct identities registered in the university's IdM system.

2.3. Identifier

An identity consists of a set of attributes that are characteristic for the entity such as gender, date of birth and home address. If these are unique and used for identification purposes they are referred to as identifiers for a specific identity. A home address counts as a characteristic for a person, but not as an identifier as it may not be unique within the identity domain. On the other hand, a mobile phone number or a social security number would due to their uniqueness apply as an identifier for a person.

2.4. Digital Identity

Digital identity is defined as a traditional identity existing or being represented in a computer network system such as the Internet or a corporate intranet [38]. These can be issued or certified by various parties such as governments or banks, or they could be user generated, the latter being the most common approach.

2.5. Service Provider-Centric versus User-Centric Identity Management

IdM systems can be split into two different categories, namely service provider-centric and user-centric. Service providers (SPs) have traditionally positioned themselves as both an identity and credential provider, as it allows for them to enjoy reduced cost and complexity, and more control of their IdM systems. One of the downsides from an SP's perspective, although not highly significant, is the absence of new user registrations due to human indolence. Of course, if the

service provided is valuable enough, a user will not hesitate creating yet another account. However, the unavoidable problem seen from a user's perspective, is to remember all the identifiers and credentials issued or created for authenticating with the various services.

Another aspect worth taking notice of is data governance, hence ultimately identity governance. As users have to trust more SPs with personal characteristics such as date of birth, address etc., the risks of such information being lost, tampered with or leaked to undesired parties grow.

There are specifications in place to support identity federation between SPs. This allows for a user to enjoy services from distinct SPs after only having signed in with one of them beforehand. This concept is known as Single Sign-On (SSO) and could be obtained using a variety of protocols and standards such as the SAML protocol [27] and WS-* [41] specifications, respectively.

The fundamental difference between a user-centric IdM system and SP centric ones, is that in the former it is required to ensure that the user is the same as the one registered but not necessarily *who* she is, as in some SP centric IdM. User-centric IdM is suitable for social communities and e-commerce but not secure enough for banking, enterprise and government applications. A user-centric IdM system allows for the user to have her identity located in and managed from one centralized entity in the network, typically called the Identity Provider. From here the user can, when authenticated,

- confirm a given identity to SPs
- manage and grant SP access to identity attributes
- revoke access given to SPs
- manage identity attributes

SSO is commonly achieved by keeping a session between the user and the Identity Provider. On a subsequent SP login, the existing session will relieve the user from having to reauthenticate. A slightly different approach to SSO, although user-centric, is promoted by Microsoft and termed Simplified Sign-On with Microsoft Cardspace (code named "InfoCard") [3]. Its functioning minimizes user intervention, whereas a digital identity card is presented to an SP for authentication, asserted by the user herself or a third party identity issuer.

SPs supporting user-centric IdM have a trust relationship to the Identity Provider in charge of a particular identity, be it the End User herself or a third party. This way, a user needs only authenticate herself to her Identity Provider to prove ownership of a claimed identity. This relieves the SPs from having to deal with user management and authentication as this is completely decentralized and delegated to the Identity Provider.

The user-centric scheme benefits the user in both usability and security as it lessens the burden of having to manage or remember a multitude of usernames and passwords. User familiarity with an Identity Provider's visual interfaces contributes to increased security when authenticating. Having the possibility to transfer

the identity trust relationship to other Identity Providers is an additional advantage.

3. OpenID

OpenID is one of several user-centric identity management systems freely available on today's market. With its openness, flexibility and widespread adoption it has risen to become one of the most popular decentralized IdM systems for the consumer market. The current section of this thesis aims to give the reader a good understanding of how the OpenID framework operates and how the different entities interact to provide the functionalities needed in an identity management system. Unfortunately as with any other data system human interaction and dependence introduces security weaknesses to OpenID, which will also be elaborated.

3.1. Concept and Adoption

One can gain several benefits by exploring and adopting the OpenID concept, both as a user and as a service provider. The latter is more commonly, and for the remainder of this thesis, referred to as a Relying Party (RP) in an OpenID context. An interesting approach taken by OpenID benefits the users by letting an already existing web site URL, in the user's possession, act as their OpenID Identifier. Moreover, a user can have any URL under her control to be considered as her digital identifier. If no such controlled URL exists, a user can register with an OpenID Identity Provider (OP) to obtain one. OPs are freely available on the Internet; myOpenID [25], claimID [5] and myID.net [24] are just a few of the biggest ones. In fact, users might unknowingly have an OpenID already, as both Google and Yahoo! are OPs themselves [1]. These two companies contribute to a large user base resulting in as many as 1 billion OpenID-enabled users worldwide and more than 9 million RPs at the end of 2009 [18]. At the time of writing this thesis the most recent significant example of adoption is for registration of a Flickr account using an existing Google account, made possible by OpenID [19]. Even Facebook allows users to automatically sign in with OpenID given that the web browser have an already authenticated session with the OP associated with the Facebook account [35].

3.2. OpenID Framework

An OpenID Identifier can be presented to an RP for user registration or login, whereas the subsequent authentication process is delegated to the OP by the OpenID framework. The initiation of authentication follows the specifications given by the OpenID 2.0 Authentication protocol [30], but the actual mechanism itself is dictated by the OP currently in charge of confirming ownership of an OpenID Identifier. Ultimately the mechanism is based on user preference as she decides what OP to trust. Section 3.2.2. studies this in detail.

3.2.1. Entities

Next, with referral to Figure 2, a complete picture of the entities and their roles in the OpenID framework will be presented.

3.2.1.1. End User

The End User is the physical human being in control over one or more OpenID identities. Hence the End User corresponds to the entity described in section 2.1. It is the End User's task to manage and maintain the OpenID identity in means of granting or revoking access to RPs, as well as keeping identity attributes updated. All of the previous actions require prior authentication of the End User to the OP, which is normally done by providing a credential such as a password. It has been done extensive work on finding ways for more secure and easier authentication which the thesis will come back to later in section 4. The End User is sometimes referred to as 'the (female) user' in this thesis.

3.2.1.2. OpenID Identifier

An OpenID Identifier is a unique string in the OpenID domain, acting as the identifier an End User uses for logging in to an OpenID-enabled web site, i.e. an RP, as shown in step 1 in Figure 2. It is used in the discovery phase by RPs, with the goal of returning a pointer to the OP in charge of it. The particular pointer is called the OP endpoint, and is the URL which accepts OpenID Authentication Protocol messages. It is most often given in the form of an HTTP or HTTPS URL as shown in Figure 1, where the latter severely reduces the risk of a phishing or man-in-the-middle attack due to the strengths of X.509 certificates. Another option supported by the OpenID Authentication 2.0 protocol [30] is the use of Extensible Resource Identifiers (XRI) [6] as identifiers, where an XRDS document is used for OP discovery.

```
https://idexample.myopenid.com
```

Figure 1: Example of an OpenID Identifier

3.2.1.3. User Agent

The User Agent (UA) is an Internet browser supporting the HTTP/1.1 protocol specification. This entity works on behalf of the End User's input and handles requests, responses and redirects between both the RP and OP. SSL/TLS must be supported if such is enforced either by the RP or the OP.

3.2.1.4. Relying Party

The Relying Party (RP) provides services to users and allows for authentication through the OpenID Authentication 2.0 protocol. An RP is interchangeable with a user-centric oriented service provider (SP), a term used previously in this thesis. When the RP has been presented with the OpenID Identifier, it is responsible for performing a discovery of the OP before redirecting the UA to the discovered destination for authentication. The RP has an additional option of establishing an association directly with the OP.

3.2.1.5. OpenID Identity Provider

The OpenID Identity Provider (OP) is an identity provider designed to host the OpenID identity on behalf of the End User. A user can also trust the OP with managing several OpenID identities. The OP must be able to authenticate a user if one claims ownership of an OpenID Identifier, and relay assertions of this if requested by an RP. How the authentication takes place is out of scope for the OpenID Authentication 2.0 protocol, although an RP can enforce the OP to use a policy extension known as OpenID Provider Authentication Policy Extension 1.0 [31], to ensure a certain level of authentication reliability.

3.2.2. Authentication Protocol

The authentication protocol's goal is to prove to the RP that an End User controls an OpenID Identifier. A small set of messages exchanged between the OP, the UA and the RP realizes this goal given that the user successfully authenticates with the OP. In the following, deducted from mainly [30] but also [20], the various messages are explained as well as which underlying techniques and protocols are used to make the OpenID protocol as solid and secure as possible. Although there are more than one protocol version in use on the Internet today, we will exclusively focus on the OpenID Authentication 2.0 protocol in this thesis, as this is the latest, thus proper version to be implemented.

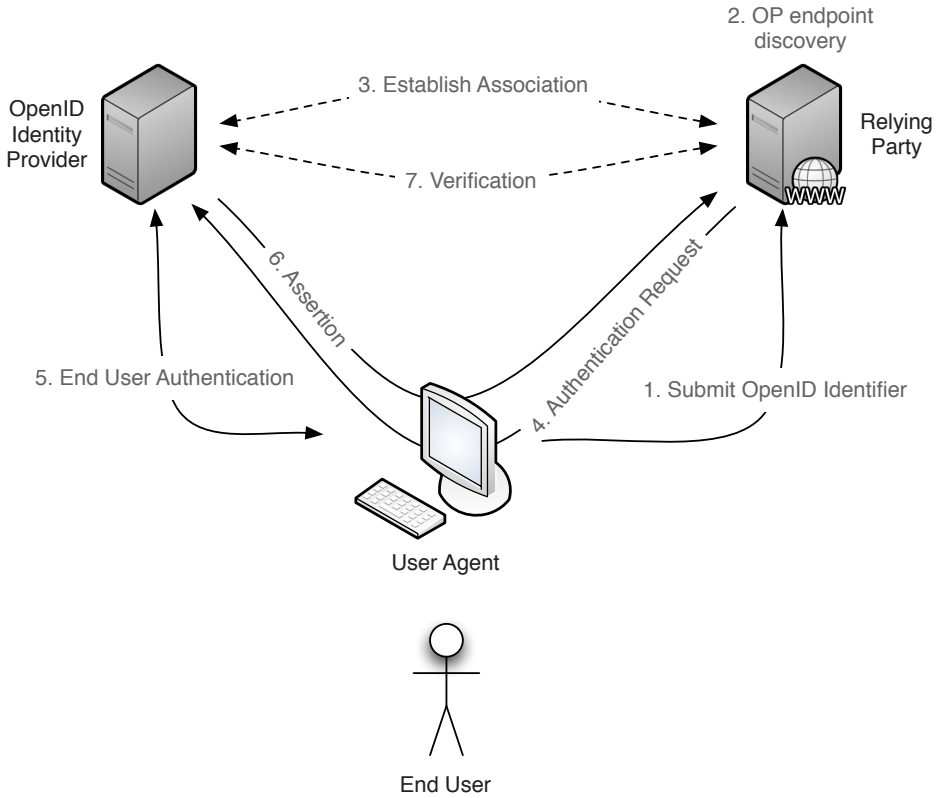

Figure 2: OpenID Authentication Protocol 2.0 overview

As shown in Figure 2 there are mainly three entities involved in the authentication procedure; the User Agent (UA), the OpenID Identity Provider (OP) and the Relying Party (RP). The UA is controlled by the End User and is usually a

web browser. In the following subsections the UA and the End User is interchangeable. The UA wishes to access some service provided by the RP, but authentication is demanded by the latter. In step 1 the UA presents its OpenID Identifier to the RP through an HTML form. The identifier is sent using HTTP GET method and optionally encrypted using SSL/TLS.

3.2.2.1. OpenID Identity Provider Discovery

In step 2 the RP performs a discovery of the OP based on one of three different discovery methods:

a) If the user has entered an eXtensive Resource Identifier (XRI) an eXtensive Resource Descriptor Sequence (XRDS) document will be used

b) If the user has entered an URL the Yadis protocol [21] will be used

c) If the user has entered an URL and there is lack of support for the Yadis protocol, the HTML document at the given URL will be used

If none of the above are available the RP can not go through with the authentication protocol. In order to support former versions of the OpenID Authentication protocol the returned document from the OpenID Identifier can contain pointers to multiple OPs running different protocol versions. Additional benefits gained from this is load balancing, in the unlikely case where one of the OPs in charge are unavailable.

In case a) when an XRI is used as an identifier, the RP performs a similar lookup of the desired OP as done in a Domain Name System (DNS) resolution when looking up IP addresses from domain names. An XRI is ultimately something called an i-number, but can have several i-names which acts as long-lived (but reassignable) identifiers for an i-number. An i-number is a permanent identifier dedicated to a resource in a system be it e.g. a person, organization or an OP, but distinguishes itself from an i-name in being less human readable as it consists of a non-logical mix of numbers, symbols and letters. A typical i-name belonging to a user could be `=myself*username`, and can be obtained for free from several providers on the Internet. The submitted XRI is resolved to an XRDS document which contains the pointers to the OP associated with the XRI.

In case b) the document returned from the URL could either be the XRDS document containing the necessary information about the OP, or it could be an HTML document with a header link to the sought after XRDS document. In the latter, the RP needs to perform an additional HTTP request using the provided link. The link should be found inside a HTML meta tag having the *http-equiv* attribute set to `X-XRDS-Location` and the value of the *content* attribute set to the location of the XRDS document. When the RP has received the XRDS document, the Yadis protocol has completed.

In case c) neither an XRDS or a link to an XRDS document is returned. In order for the given URL to be a valid OpenID URL the returned HTML document must contain links to the OP allowing the RP to perform HTML-Based Discovery. To achieve this, an HTML link tag containing the correct attributes

must be placed within the HTML `head` tag. The *rel* attribute must be set to `openid2.provider` and the *href* attribute must be set to the OP endpoint URL. Some OPs implement backward compatibility supporting the older OpenID Authentication 1.1 protocol by adding an alternative link tag pointing to the desired OP.

After successful discovery, the RP is ready to initiate communication with the OP to complete the authentication protocol.

3.2.2.2. Association

Step 3 in Figure 2 establishes an association between the RP and the OP. In the figure it is shown as a dashed line meaning that it is optional. An example of an Association Request message is shown in Figure 3. The association allows for the generation of a shared secret key used for message encryption in subsequent communication between the two parties. It is an optional but recommended feature to implement at OPs, as its use benefits from not having to generate new keys for every authentication request. Additionally, the assertion message returned in step 6 can be verified locally at the RP without an extra request to the OP.

The two parties collaborate to establish a shared secret using the Diffie-Hellman Key Agreement Method [33]. The RP sends an Association Request message directly to the OP. This message, as well as all the others, are using HTTP as an application layer transport protocol. The use of Transport Layer Security (TLS) or Secure Sockets Layer (SSL) will strengthen the protocol in terms of security, but will also have some influence on a specific parameter in the messages exchanged in the OpenID protocol. The Association Request message has the form **AssocReq(**t,s,p,g,DH_{RP}**)**, where t is the preferred association type describing what algorithm to be used to sign subsequent messages, s is the preferred session type describing how or if to encrypt the MAC key in transit, p and g is the modulus prime number and generator, respectively, used in the Diffie-Hellman algorithm, and DH_{RP} is the RP's public key. If using "no-encryption" as session type it is required to use transport layer security to prevent the MAC key from being transmitted in the clear.

```
POST /mopenid/server HTTP/1.1
HOST: 188.32.23.359:8080
Content-Type: application/x-www-form-url-encoded
Content-Length: 341

openid.ns=http://specs.openid.net/auth/2.0
&openid.mode=associate
&openid.assoc_type=HMAC-SHA256
&openid.session_type=DHSHA256
&openid.dh_modulus=p
&openid.dh_gen=g
&openid.dh_consumer_public=#DHRP#
```

Figure 3: Example of an Association Request message

The OP answers with the Association Response message which has the form

AssocRes(*AH,t,s,n,h,*DH_{OP}**)**. An example of an Association Response message is shown in Figure 4. *AH* is the association handle which both the RP and the OP should use as a key to refer to the association in subsequent message exchanges, *t* and *s* are the same values for the association type and session type, respectively, as was in the request. The *n* parameter is the association's lifetime in seconds and DH_{OP} is OP's public key. *h* is either the shared secret in plain text or the secret encrypted by XORing the MAC key with the secret Diffie-Hellman value, depending on the session type. If either of *t* or *s* is unsupported by the OP a unsuccessful response message is sent to the RP.

```
HTTP/1.1 200
Content-Type: text/plain
Content-Length: 172

ns:http://specs.openid.net/auth/2.0
assoc_handle:2j30f9j3ffa
assoc_type:HMAC-SHA256
session_type:DHSHA256
expires_in:n
enc_mac_key:#MAC_key XOR secret_DH#
dh_server_public:#DHOP#
```

Figure 4: Example of an Association Response message

3.2.2.3. Authentication Request Message

Step 4 in Figure 2 shows the Authentication Request message, which differs from the association messages in that the OP is indirectly rather than directly requested by the RP, through the UA. This is accomplished by using the HTTP status code `302 Found` in the response packet returned to the UA after the initial request in step 1, which is a HTTP redirect packet with the OP endpoint as destination. An example of this message is shown in Figure 5. The Authentication Request message has the form **AuthReq(***m,AH,cID,ID,rt,r***)** where *m* is the request mode deciding whether any user interaction with the OP should be possible or not[1], *AH* is the association handle that is subject to have been established between the RP and the OP in step 3, and *cID* is the OpenID Identifier the user claims to be in control of. If delegation is used, the *ID* is the OP-local identifier, otherwise it is the same as the claimed identifier. Also, setting the ID to the special value `http://specs.openid.net/auth/2.0/identifier_select` the OP is expected to choose the OpenID Identifier that belongs to the End User. The *rt* parameter is the return to destination where the RP wants the OP to redirect the UA after the authentication process. At last the *r* parameter is the URL pattern, or realm, the OP asks the End User to trust. At the moment the RP sends the Authentication Request it also stores the *rt* and *cID* values to be able to lookup the request when the Assertion message is received in step 5, as described next.

1 If not then a asynchronous JavaScript typically performs this request

```
HTTP/1.1 302 Found
Location:http://188.32.23.359:8080/mopenid/server
?openid.ns=http://specs.openid.net/auth/2.0
&openid.mode=checkid_setup
&openid.claimed_id=https://mopenid.item.ntnu.no/id/idexample
&openid.identity=https://mopenid.item.ntnu.no/id/idexample
&openid.assoc_handle=2j30f9j3ffa
&openid.return_to=http://rp.example.com/signin
&openid.realm=http://rp.example.com
```

Figure 5: Example of an Authentication Request message

3.2.2.4. Positive Assertion Message

Step 5 in Figure 2 is out of scope of the OpenID protocol. The details and requirements on how the user proves ownership of a claimed OpenID Identifier is entirely left up to the OP. Say that the user authenticates successfully with the OP, then a Positive Assertion message is sent indirectly to the RP, through the UA, as shown in step 6. The Positive Assertion message has the form **PosAssert(**m,OP,cID,ID,rt,n,AH,{s},s**)** and sent as an `HTTP 302 Found` message to the UA with the OP as destination. *m* is the mode parameter with its value set to `id_res` and *OP* is the URL of the originator of the message. *cID* and *ID* must either be both present or absent. The value of the former is the claimed identifier or optionally the OP-local identifier. The latter's value is the OP-local identifier, thus possibly equal to the *cID*. The *rt* parameter is an exact copy of the *rt* parameter in the Association Request message. The *n* parameter is a response nonce formatted in a particular way, containing the OP's current date and time, appended by a random string aiming to make the nonce unique. The *AH* parameter is the association handle identifier used for signing the assertion message. The {*s*} parameter is a comma-separated list of certain parameter values part of this message which are signed. The *s* parameter holds the resulting signature in the format decided by the association type in the Authentication Request message. An optional parameter can be sent, namely the *invalidate handle* indicating to the RP that its value, i.e. handle, should be considered invalid for future interactions.

```
HTTP/1.1 302 Found
Location:http://rp.example.com/signin
?openid.ns=http://specs.openid.net/auth/2.0
&openid.mode=id_res
&openid.op_endpoint=http://188.32.23.359:8080/mopenid/server
&openid.identity=https://mopenid.item.ntnu.no/id/idexample
&openid.claimed_id=https://mopenid.item.ntnu.no/id/idexample
&openid.return_to=http://rp.example.com/signin
&openid.responce_nonce=2010-11-21T21:12:30ZD3R6F%
&openid.assoc_handle=2j30f9j3ffa
&openid.signed=op_endpoint,return_to,responce_nonce,claimed_
id,identity
&openid.sig=#signature#
```

Figure 6: Example of a Positive Assertion message

3.2.2.5. Assertion Message Verification

Assertion message verification happens in one of two ways. Either the RP can use the already negotiated MAC key which can be indexed using the association handle, or the RP can send an additional direct message to the OP requesting verification of the Positive Assertion message. The latter case is illustrated in step 7 in Figure 2. The RP needs to verify several of the returned parameters in the Positive Assertion message against parameters originally sent in the Authentication Request message. The *rt* parameter needs to match the URL of the current request. Further, all the information gathered in the discovery process in step 2 needs to be verified by checking certain parameters in the Positive Assertion message. Table 1 shows the mapping between the discovered values and the fields returned in the assertion message, which are required to have equal values for the RP to approve the login.

Discovered value	**Response field**
Claimed Identifier	openid.claimed_id
OP-Local Identifier	openid.identity
OP Endpoint URL	openid.op_endpoint
Protocol Version	openid.ns

Table 1: Discovered information to Positive Assertion Response message [30]

Further, if the Association Request message delegated to the OP to choose an identifier for the user, the Positive Assertion message will contain this identifier on return. In this case the RP must perform discovery on the returned identifier to make sure that the same OP controls it. The nonce must be checked to verify its uniqueness among nonces returned from the specific OP. To ensure that no values have been tampered with on the way, the signature should be checked. If an association exists between the OP and the RP verification is easily performed using the pre-negotiated key and the format specified as the association type. If no association exists the RP needs to request the OP to verify the positive assertion message by sending a direct message to the OP. The message is a copy of the Positive Assertion message received, with mode parameter altered to the value `check_authentication`. The OP returns a message containing two parameters, *openid.is_valid* and *openid.invalidate_handle*. The first parameter is set to `true` if the signature is correct and `false` otherwise. The second and last parameter is optional, but if included the given association handle should not be used by the RP anymore.

All of the above messages contain a common parameter which specifies the protocol version of which the contents are formatted by. This is the *openid.ns* parameter and is set to `http://specs.openid.net/auth/2.0` stating that only the 2.0 specification of the OpenID Authentication protocol is respected.

3.3. Weaknesses

There are arguably weaknesses to the OpenID Authentication 2.0 protocol if the necessary precautions are not being taken. Some of the major weaknesses and recommended steps to prevent them are presented in the end of the protocol specification [30]. There have been several independent analysis performed on the overall security of OpenID which in some cases have revealed security threats [8] [20]. We will in the following subsections look into some of these.

3.3.1. Phishing

A phishing attack involves tricking an End User into giving up her credentials for some protected resource to some perceived trusted entity in a network [23]. Such an attack requires no more than a rogue RP and lack of user attention. The essential step of this attack is to redirect the UA to a malicious OP with similar or identical visual looks as the genuine OP. A low-observant user miss to see that the URL she has been redirected to does not match the URL belonging to the genuine OP. Perceived to authenticate with the real OP, the user instead reveals her credentials to the fake OP, leaving its owner capable of controlling the OpenID identity managed by the genuine OP. The attack can be avoided if the user is more aware of where she is being redirected to, and optionally choosing an OP with SSL/TLS support for server authentication purposes. Additionally, the user can initiate an authenticated session with the OP before trying to log in with RPs. When the UA is redirected in step 4 in Figure 2 it will use the session it already shares with the OP, thus there is no need to once again authenticate by e.g. entering a password. However, user knowledge about unnecessary reauthentication when a session already exists is needed.

3.3.2. Cross-Site Scripting

Cross-site Request Forgery (CSRF) is a type of cross-site scripting and a potential attack given that the user already has an authenticated session with the OP. If the UA encounters malicious HTML code as shown in Figure 7 at some visited website, unwanted actions could take place. The code exploits the existing session between the OP and the UA, triggering the UA to send certain HTTP requests to other RPs without the user knowing. If the user already has an account with a given RP, the code can log in the user and, secretly, perform harmful actions [39].

```
<iframe id="login"
src="http://bank.com/login?openid_url=http://idexample.myopenid.
com" width="0" height="0"></iframe>
...
<iframe id="transfer"
src="http://bank.com/transfer_money?amount=100&to=attacker"
width="0" height="0"></iframe>
```

Figure 7: Malicious HTML code for realizing a CSRF attack. Source: [39]

3.3.3. Denial of Service

The OP is subject to a denial of service attack if a malicious RP, or another entity in the network, decides to launch a large batch of association, authentication or verification requests at it [30]. The most severe threat happens in the case of an association request as this forces the OP to perform a discrete exponentiation based on parameters received from the attacker. No built-in protection from this attack is provided by the OpenID protocol.

3.3.4. Data Governance Issues

An End User needs to fully trust its OP. But how does a user know that the information stored with the OP is safe and treated as careful as required with respect to security. [43] states that

> *"Data governance is the term used for knowing what happens to the data that is stored, particularly when that data has any PII (personally identifiable information), which the OpenID IdP does."*

Not only does the user allow for the OP to store PII such as date of birth, name, e-mail address and other contact information, but she also lets the OP manage the list of approved services associated with the OpenID identity. Usage patterns can easily be derived from observation. As an End User, one have the option of trusting third party OPs or host one yourself. Concerning the former, which by far is the most common approach, how certain can a user be that the PII is kept secret or that credentials are stored in an absolute secure manner, e.g. that they are stored as hashed values rather than plain text? Is the PII that is stored with the OP encrypted? If no, consequences are big if the system is breached or the disk containing the database physically stolen, as it allows the intruders to access and exploit sensitive information.

Having an account with an OP is usually provided at no cost to End Users, which leads to the question of how OP administrators make enough money to cover the expenses associated with providing the service. OP service requirements are, among others, high availability and strong security, something which does not come for free, rather the opposite. The Privacy Policy page belonging to the OP myid.net [24] states the following about how personal information is used:

> *"In connection with the delivery of our services, we may share the Personal Information with third-party partners or vendors who help deliver or administer the services, but they are only permitted to use the information in connection with the delivery or administration of our services."*

The current practices differ between various OPs of course, but there is a reason to believe that the above condition allows for the OP to make profit from e.g. selling customized advertisements based on the information shared to third parties, as the term *service* is rather ambiguous.

OpenID is referred to as a highly user-centric IdM solution, but the user identity is, however, in most cases managed from a trusted third party. Presenting an even more user-centric solution is the purpose of this thesis, which allows more

control to the users who simply do not want third parties controlling their PII and access to services using OpenID.

3.3.5. Feasibility of setting up a Private OpenID Identity Provider

The OpenID Foundation web site [29] motivates advanced users to set up and host their own private OPs. A number of libraries on the foundation's developer web site are listed and available for supporting OP (and RP) implementations on various platforms written in different programming languages. A great deal of technical insight, time, server access with administrative rights, and not to mention motivation is required in order for an End User to host the OP herself. How feasible is this for the average user? Obviously, not very. But should the lack of technical skill and interest keep those users incapable of enjoying the control and security of hosting their own private OP? In modern times, relatively cheap consumer electronics, possessing the necessary hardware and software support, are available, namely smartphones. With this in mind, a solution to solve this problem is presented in section 5.

4. Prior Work for enhancing OpenID Authentication

As we have learned from last section, the implementation and process of the authentication between the OP and the user is out of scope of the OpenID Authentication protocol. Based on what the user prefers she selects the OP with the adequate authentication mechanisms. Some of the weaknesses facing OpenID have been tried to dealt with through various enhancements of the authentication procedure between the OP and the End User. The implementation in this thesis introduces new possibilities for enhancing this process, but we will first look into some of the work that previously has been done in this area. The following subsections describe two distinct authentication techniques between an OP and an End User.

4.1. An OpenID Identity Provider based on SSL Smart Cards

A proposal for a stronger authentication solution using SSL smart cards is suggested in [40]. It does so by enforcing mutual authentication between the user and the OP using X.509 certificates on both ends. The smart card is plugged in a USB key also holding a flash drive, which together constitute the authentication token. While the flash drive includes a small piece of software used for proxying HTTP messages between the token and the OP through the UA, the smart card holds the RSA private key and the X.509 certificate associated with the OpenID identity. The user experience is argued to be hassle free as most of the setup and communication is performed under the hood, moreover anonymously, to the user. The idea is to relieve the user of having to remember a username and a password at all, which is traditionally used for authentication in OpenID. However, if the user is to lose the authentication token, or lack administrative rights on a public computer needed to run the small piece of software on the flash drive, limitations to this solution is evident.

4.2. OpenID and SIM

A solution for how telecommunication companies can participate as facilitators for the adoption of OpenID for their already existing subscribers is proposed in [2]. The solution seems profitable for the mentioned companies as it requires the ubiquitous SIM as a key part in the authentication procedure. The OP is operated by and located at the telecom operator, and is tightly connected with an authenticator which receives and handles EAP-SIM [13] over HTTP messages from the user as part of the authentication scheme. On the user side a Java Applet running in an Internet browser is able to communicate with both the SIM through a USB dongle or a Bluetooth capable mobile phone, and the server-side authenticator, thus fulfilling the EAP-SIM over HTTP authentication messages required from the user side as defined in the paper. The strength of this authentication scheme relies on the cryptographic algorithms and shared secret keys stored in both the SIM and with the authenticator. Benefits are that the user does not need to remember a traditional user name and password at all, as the SIM handles the authentication requirements by itself, thus eliminating the potential insecure passwords that users tend to have. However, the drawbacks are the needs for the user to have software installed to support the Java Applet, not to mention the possibility to connect through a USB or Bluetooth interface for communicating with the SIM. Although both of the communication interfaces are common on most laptop and desktop computers today, it should not be taken for granted that the user have sufficient permissions to use them or even run the Java Applet if using a different computer than her own. In addition, trusting a third party, in this case the telecom operator, to manage a user's OpenID identity and make it reliant on a SIM, counteracts the goal to further enhance the user-centric approach for managing digital identities.

In this context a similar, but not identical solution has in fact recently been implemented by the largest mobile operator in Japan, NTT docomo. Essentially, the implementation allows for their subscribers to use their issued subscription ID as an OpenID identity as well [34]. The company subscribes approximately 50% of Japan's population, which amounts to over 55 million OpenID-enabled customers. This clearly illustrates the power certain companies with a large user base possess in regards to increasing use of identity management frameworks such as OpenID, and should inspire other telecom operators to do the same.

5. mOpenID – Mobile OpenID Identity Provider

5.1. Idea

As discussed previously in section 3.3. OpenID although popular, suffers from weaknesses. One of the most severe is the data governance insecurities related to the identity data being stored by an OP. The OP dictates the various privacy policies in which the user has no other option than to accept. Furthermore, the user may experience unexpected withdrawal of services or policy changes decided by third parties. Last but not least there is overall weak security due to authentication through the UA as phishing attacks may occur. To remedy these weaknesses the following work presents the implementation of a private OP running on a smartphone, named the *Mobile OpenID Identity Provider* application. The solution will address the technical challenge of setting up and hosting a private OP and introduce new and arguably easier and more secure authentication possibilities.

5.2. Network Reachability Models

There are certain constraints that apply when introducing reliance on a smartphone to host an OP rather than confessing to the traditional fixed server. One big challenge is to enable inbound TCP/IP connections to the smartphone, as this is required for any OP. IP connectivity between a mobile phone and the Internet is made possible by the gateway GPRS support node (GGSN) in the GPRS core network infrastructure, which handles functions including access point name (APN) processing, IP address allocation, tunneling technologies, and QoS management [4]. In most cases, inbound data traffic is disabled by default by the network operator's GGSN firewall. In the following subsections three network architectures are presented to enable for a connection to be established with a smartphone from either the UA or the RP.

5.2.1. Port forwarding a Wi-Fi access point

Most Wi-Fi access points provide port forwarding functionality as a standard feature. By connecting the smartphone to an access point in possession of a public IP and routing the incoming traffic on a given port on the access point to the smartphone, one can access the OP running on the smartphone from the Internet. Figure 8 illustrates how the OP is made accessible on port 8080 of the public IP address belonging to the Wi-Fi access point. This might come in handy for testing purposes during application development. However, as it brings configuration demands on the End User, it increases the solution's complexity, which in turn violates the core purpose of this thesis, namely making the solution applicable for the novice user. Lastly, always having to depend on a Wi-Fi access point being accessible when desiring to log in with OpenID is inappropriate.

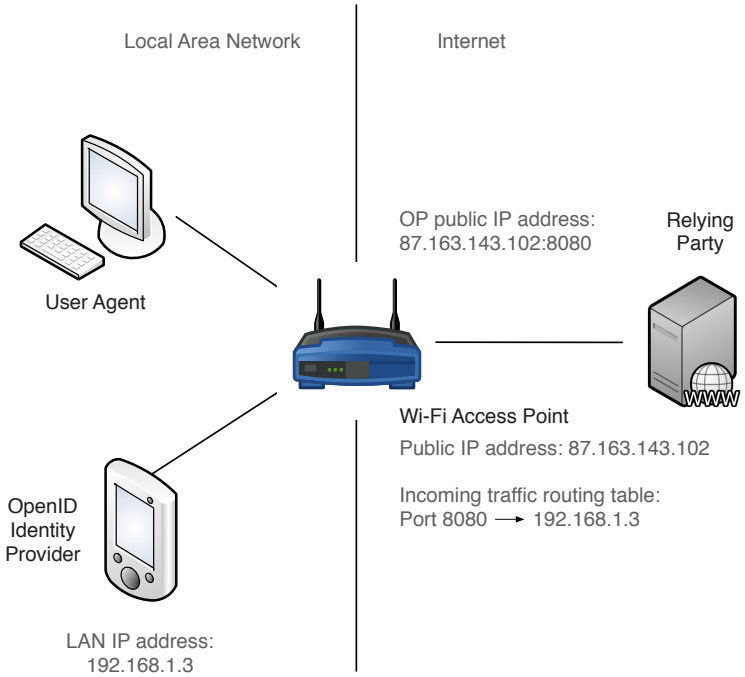

Figure 8: Architecture when a Wi-Fi access point uses port forwarding to make the OP accessible

5.2.2. Proxying through a Managed Server

Another solution is to introduce a controlled server acting as an intermediary between the cellular network connected OP and the RP and UA, which together make up the entities in the OpenID framework. The controlled server is shown in Figure 9 as the OpenID Proxy entity and has the URL `https://op-proxy.com`. The appended `/userid` string is unique to a given OP that connects to and initializes with the OpenID Proxy, which happens at the moment the OP application is launched. The initialized session is maintained between the OP and OpenID Proxy, making the OP reachable for any User Agent or Relying Party by having the proxy forward to it any Authentication or Association messages requested to the URL `https://op-proxy.com/userid`. Likewise, the OP responds with the correct protocol messages to the proxy, which in turn ensures final delivery to the initial requester. Note that the URL acts as, and in fact is, the OpenID Identifier. Thus, yet another dependence is introduced to the solution, namely having to rely on the OpenID Proxy server. A problem arises in the initialization phase between the OP and the OpenID Proxy server as authentication between the two has to be realized in some way to prevent a rogue party from stealing control of the OpenID identity. Thus, this solution suffers from increased complexity and inconvenience as all messages must go through an intermediate server not managed by the End User. Despite these disadvantages, it is a plausible solution to be realized.

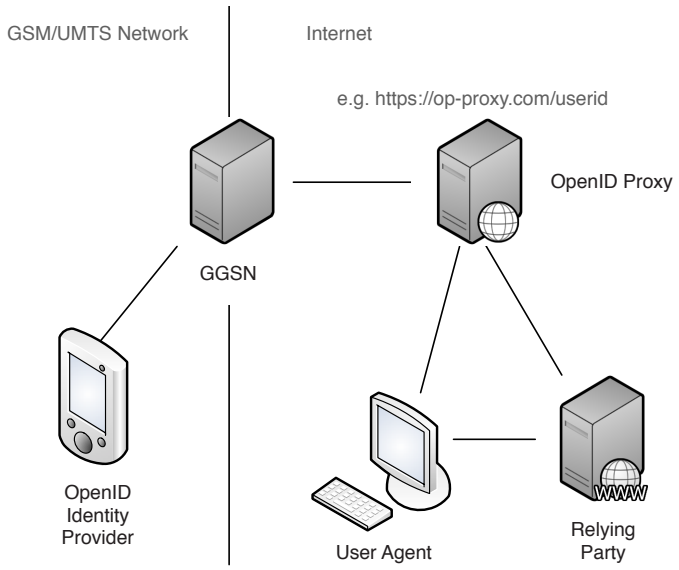

Figure 9: Architecture when using an intermediary proxy between the OP and UA and RP

5.2.3. Changing the Access Point Name

Every smartphone connected to the mobile data network are subscribed to one or more access point names (APNs), which act as reference points for other Internet nodes communicating with the smartphone [4]. If the smartphone have subscribed to an APN that allows for inbound TCP/IP connections it can be accessed directly by other Internet nodes as shown in Figure 10. Hence, the OP application needs to configure the smartphone on runtime to use the particular APN offered by the operator supporting this scheme, if such exists. For security reasons the OP application should revert to the initial APN settings on shutdown, in order to minimize the time period for which it is allocated a public IP address. This final architecture is made use of in the ultimate implementation.

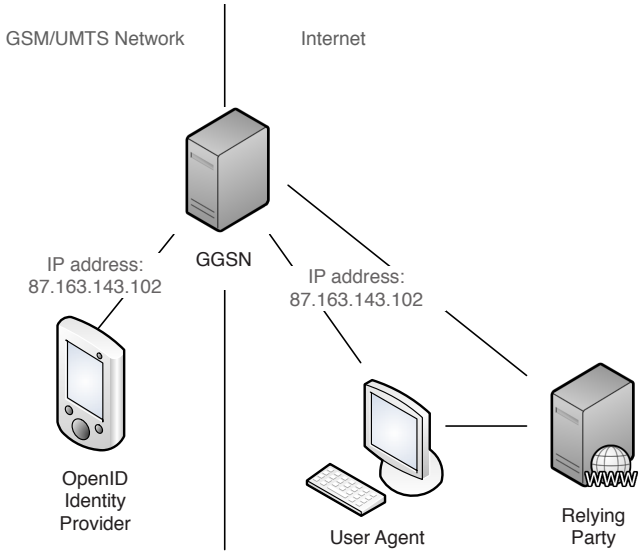

Figure 10: Architecture when changing to an APN which allocates a public IP address to the smartphone

5.3. Methodology

The three subsequent sections comprising the analysis, design and implementation phases are carried out according to the ADPO Development Methodology [37]. Initially, use case diagrams are created and explained to capture the functional requirements of the system. In the consecutive design phase, collaboration diagrams are established illustrating how various entities in the system interact to achieve the defined functional requirements. The collaboration diagrams illustrate the order of execution, hence the same as sequence diagrams would do. Consequently sequence diagrams are omitted. Class diagrams are defined next, creating the basis for the following implementation. The implementation is documented by showing how key functionalities are realized and which OpenID constraints that need to by taken into consideration and why. Lastly, the implementation is validated by going through and visualizing each of the functionalities.

A moderate amount of time has been spent getting familiar with existing frameworks, both the Android operating system itself and other packages used in the implementation. To obtain a public IP address for the smartphone a special SIM was initially acquired from Telenor R&D and tested. As the SIM did not work as expected, unforeseen research on how to obtain a public IP address arose. Before the eventual solution was discovered, work was done on how alternative realizations was to be designed and implemented.

5.4. Development Tools and Platforms

The development of the Smartphone Application was done in Eclipse aided by the Android Development Tools (ADT) plugin [12]. The HTC Desire smartphone running Android version 2.2 was used during the implementation phase, while the final application also was successfully tested and run on both the Samsung Galaxy S and the Sony Ericsson Xperia X10. Certain parts of the Smartphone Application was compiled using Apache Maven. Netbeans was used for implementing the webservices that is part of mOpenID server.

6. Analysis

From the gained understanding of the OpenID entities and protocol, use case diagrams are constructed to derive the functional requirements for the system. To successfully understand each part of the use case diagrams, Figure 11 gives more information about the entities needed to realize the system, and their interaction.

For convenience the abbreviation *SA* will be used as a single term for the Smartphone Application and the OpenID Identity Provider combined. These entities are essentially the same entity in the system. The abbreviation *MS* is used for mOpenID server.

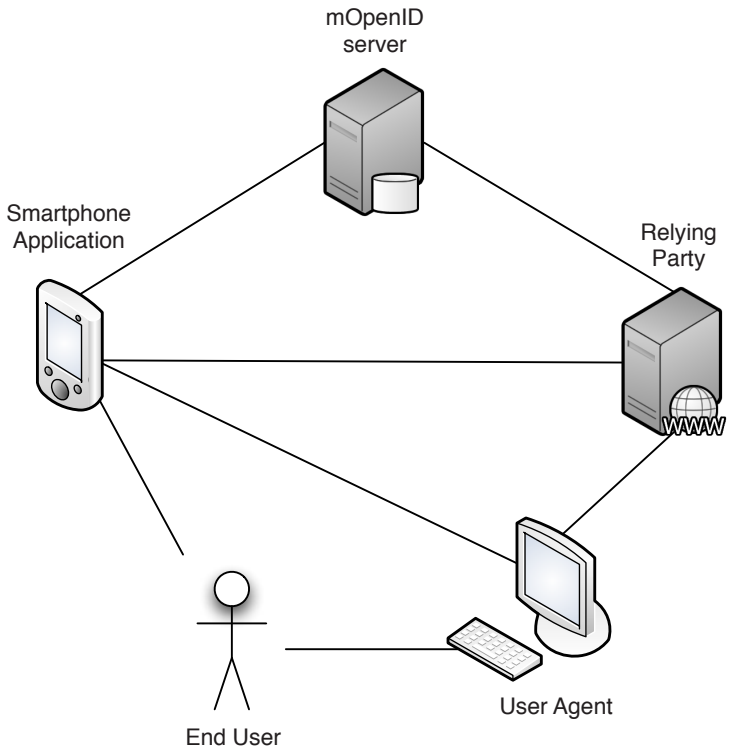

Figure 11: The figure shows the interaction relationships between the entities in the system. All the entities (except the End User of course) are connected to the Internet

As illustrated in Figure 11 the End User has physical interaction with both the SA and the User Agent. As later described in section 6.1.2., the End User can choose between two different modes when using the SA. The interaction between the MS and both the SA and RP is only realized when run in *mOpenID mode,* which is assumed to be the most frequently used mode.

6.1. Use Case Diagrams

6.1.1. End User - Smartphone Application Actions

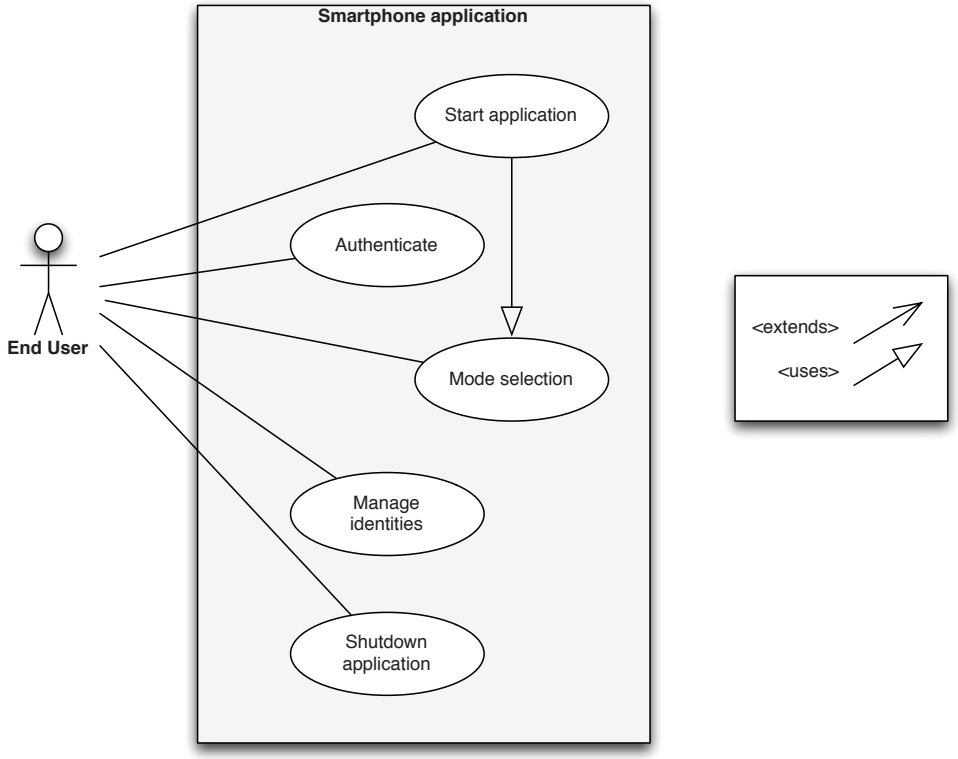

Figure 12: Use case showing application functionalities designated for the End User

The above use case diagram briefly shows the application functionalities that must be provided to the End User in order to perform a login with OpenID. Obviously, the application must be started before use – an action that includes obtaining a public IP address if not possessed already. On shutdown, the phone's configuration state prior to application launch must be restored if changed. When started, the End User must select between two different modes, each providing distinct variants of logging in with OpenID. More about this is presented in the more detailed diagram in Figure 13. The authentication functionality provides the End User the possibility to approve or reject authentication requests of a particular OpenID Identifier. Finally, management of identities include creation and deletion of identities as well as export and import options. Exporting, i.e. backing up the identity repository created in the SA saves the End User from losing possession of the identities in case the smartphone is e.g. lost or damaged.

6.1.2. End User - Smartphone Application Mode Selection Actions

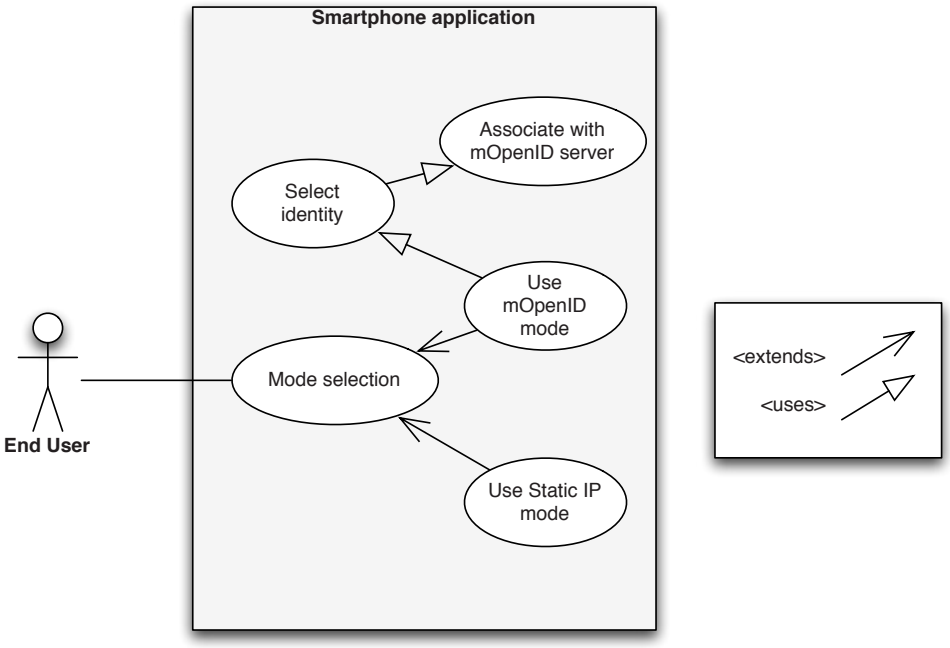

Figure 13: The figure shows the actions associated with Smartphone Application mode selection

When using the system for OpenID login, the End User can choose between two different modes, namely *mOpenID mode* and *Static IP mode*. mOpenID mode is reliant on the mOpenID server (MS) to administer the OpenID Identifier, and when selected an association with the MS is created. Static IP mode is only supposed to be used if the End User possess a special SIM that, when connected to the mobile data network, always is issued the same IP address (hence static). This makes the mode capable of operating without dependence to an extra third-party entity on the Internet (the MS). On the other hand, it makes the OpenID Identifier less human readable as it essentially is the smartphone's IP address concatenated with the "http://" prefix (e.g. `http://188.23.21.6:8080/mopenid/server`).

6.1.3. Smartphone Application And Relying Party - mOpenID Server Actions

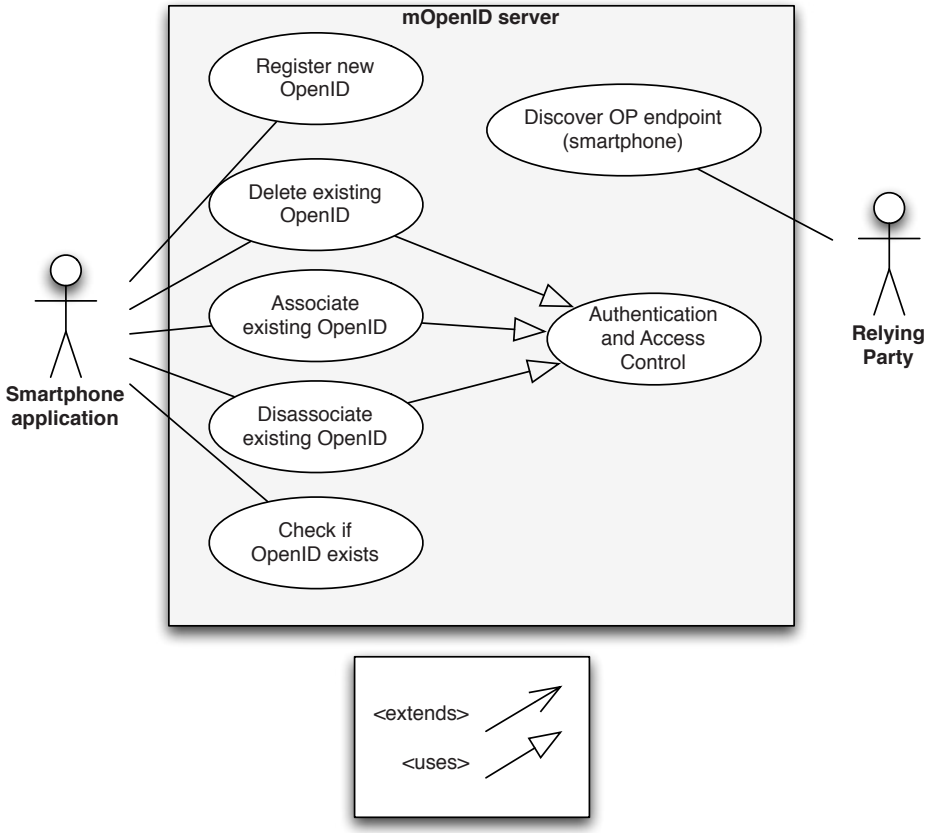

Figure 14: The above figure shows functionalities provided to the Relying Party and Smartphone Application by the mOpenID server

When the SA is run in mOpenID mode, the mOpenID server play a vital role. Specifically, it administers and stores all the OpenID Identifiers registered and managed by SA users. Its main responsibility is allowing Relying Parties to discover the OP endpoint in charge of a particular OpenID Identifier, as described previously in section 3.2.2.1. As for the SA, functionalities must be provided for registering, deleting and associating OpenID Identifiers, the latter being configuring an OpenID Identifier to point to the SA in charge of authentication when used for a login. Also, when a user want to register a particular OpenID Identifier, a functionality must be provided to check if the desired identifier is already taken. As seen in the figure, the use cases where the SA is involved, requires authentication and access control to prevent incorrect configuration of OpenID Identifiers administered by the MS.

6.1.4. User Agent and Relying Party - Smartphone Application Actions

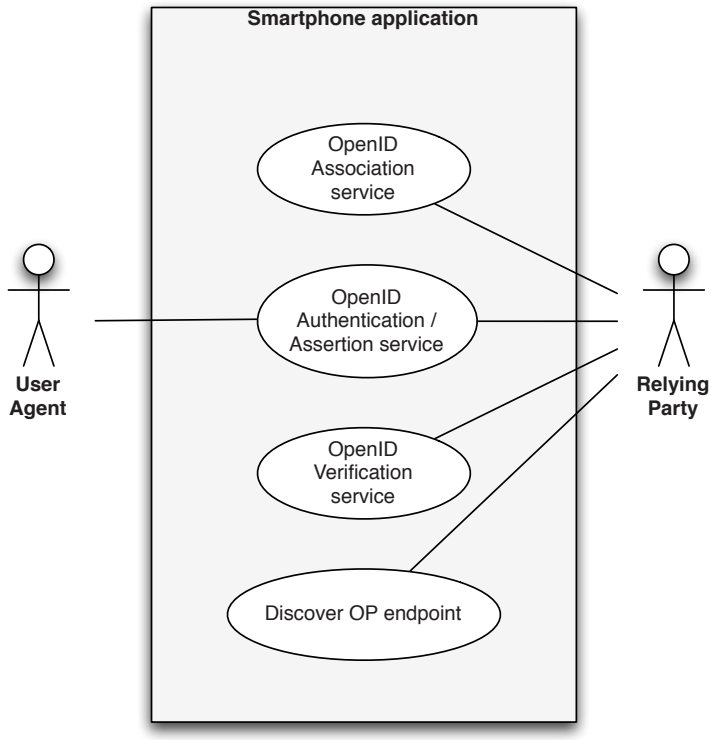

Figure 15: Use case showing basic OpenID functionalities provided by the Smartphone Application to the User Agent and the Relying Party

The use cases shown in Figure 15 are all required functionalities of any OP supporting the OpenID Authentication 2.0 Protocol [30], with exception of the Discover OP endpoint functionality used by the RP actor. This use case is needed in the case when the SA is run in *Static IP mode* and an ID page needs to be rendered on request by an RP. The ID page contains a pointer to where the OP endpoint is, which in Static IP mode is the very same location as the OpenID Identifier URL.

6.1.5. End User - Smartphone Application Identity Management Actions

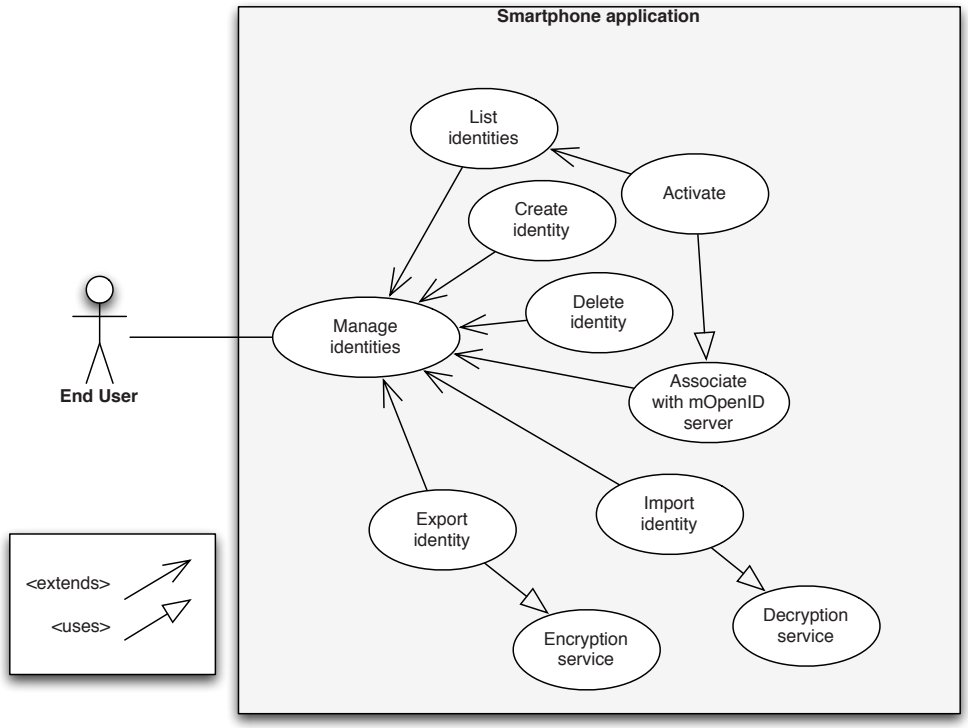

Figure 16: The figure shows actions related to identity management by the End User

The management of OpenID identities is all done from the SA by the End User. The SA must support creating, deleting, exporting and importing an OpenID identity, if requested by the End User. If one have multiple OpenID identities, a list must first be presented to the End User before selecting which one of them that should be activated for use when performing an OpenID login. All of the actions in Figure 16 is only current when the SA is running in mOpenID mode.

6.1.6. End User - Smartphone Application Authentication Actions

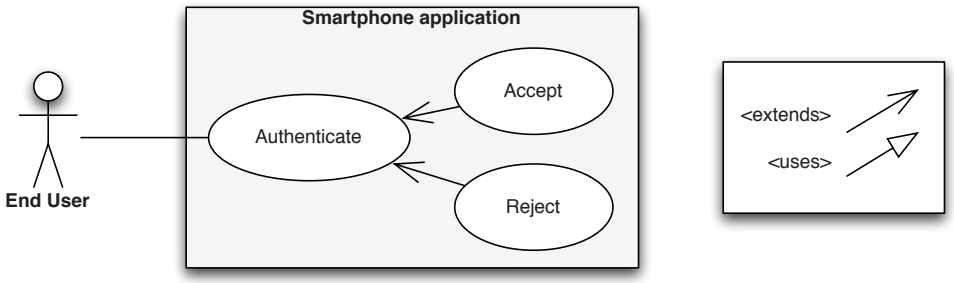

Figure 17: The use case describes how the End User must authenticate an OpenID Authentication Request Message received from an RP

The SA must provide an interface allowing the End User to accept or reject

received OpenID Authentication Requests from RPs as presented in section 3.2.2.3.

6.2. Functional Requirements

Based on the use case diagrams in the previous section, we get the following two tables specifying the functional requirements of the SA and the MS.

6.2.1. Smartphone Application Requirements

ID	Abbreviation	Description
SA-1	Start	The user must be able to start the application. In this step, the application will automatically obtain a public IP address and turn off the Wi-Fi connection. If the application is not able to perform the initial setup, an error message will be displayed.
SA-2	Shutdown	When the application is shut down, it must restore its initial settings prior to first launch. This includes making its IP address non-public and turning the Wi-Fi connection back on again if it was on initially.
SA-3	ModeSelection	The user must be able to select which mode to use when a login with OpenID is desired. The modes must show appropriate functionalities when selected, e.g. list the current OpenID identities stored in the application if mOpenID mode is selected.
SA-4	Authenticate	A user wishing to authenticate an OpenID Authentication Request Message must be allowed to do so from a GUI. Accept and reject buttons are required.
SA-5	OpenID2.0	The application must support requests and responses in accordance to the OpenID Authentication 2.0 protocol. Hence, support must exist for interacting with both a User Agent and a Relying Party.
SA-6	Manage	The application must support a various number of ways to manage OpenIDs when running in mOpenID mode, as detailed in the following sub requirements.
SA-6.1	IDcreate	The application must provide a way for a user to easily create new OpenIDs. The OpenID Identifier is constrained by a minimum character length of five, and must only contain letters and numbers. Security tokens proving ownership of the created OpenID must be securely stored in the application for later association with the mOpenID server. The application must check if a desired OpenID already is taken by someone else on the mOpenID server.

ID	Abbreviation	Description
SA-6.2	IDdelete	The application must provide a way to easily delete an OpenID and any traces of it, both locally and on the mOpenID server. The identities are deleted either completely (remotely too) or only locally. Deletion of a particular OpenID is initiated by long-pressing on it when the list over stored SA OpenID identities is shown.
SA-6.3	IDassociate	When in mOpenID mode, the functionality to associate with the mOpenID server must exist. The resulting effect of this should leave a particular OpenID Identifier residing on the mOpenID server to point to the smartphone's current IP address where the OpenID Identity Provider is running and awaiting to either reject or accept an OpenID Authentication Request received from an RP.
SA-6.4	IDdisassociate	Once a OpenID Authentication Request has been either rejected or accepted, the application must disassociate itself with the mOpenID server.
SA-6.5	IDexport	Whenever the user wants to backup its currently stored identities, the application must support this. Encryption must be used so that the exported identity repository can not be read. It should be possible to share the encrypted file to other application installed on the smartphone such as an e-mail application or Dropbox [9].
SA-6.6	IDimport	The application must support importing previously exported identity repositories. The correct decryption key must be provided by the user for a successful import. If some of the identities being imported is already present in the application, they will be skipped.
SA-6.7	IDlist	The SA must be able to show a list over the OpenIDs stored in the SA application. These are either created using SA-6.1, or imported using SA-6.6.
SA-7	Discover	When the application is run in Static IP mode, it needs to host an identity page itself, containing pointers to where the OP endpoint is. Generation of an XRDS document used for the Yadis protocol is included.

Table 2: Functional requirement specification for the Smartphone Application

6.2.2. mOpenID Server Requirements

ID	Abbreviation	Description
MS-1	IDexists	This functionality is used by the SA to check if a particular OpenID identifier is already registered. Verification of allowed characters is also included in this step.
MS-2	IDregister	Provides the SA the functionality to register a particular OpenID identifier. A randomly generated security token must be created by the mOpenID server and shared with the SA. The functionality is typically preceded by the MS-1 functionality. Verification of allowed characters is also included in this step.
MS-3	IDremove	Provides the SA the functionality to remove (delete) a particular OpenID. This will only be performed if the parameterized security token is correct.
MS-4	IDass	This is the association action. The SA uses this functionality prior to an OpenID login. The mOpenID server should activate the ID page of the particular OpenID identifier and initialize the necessary pointers to point to the SA, i.e. the OP endpoint. This will only be performed if the parameterized security token is correct.
MS-5	IDdisass	This is the disassociation action. The SA will not use this functionality unless an association has been established already. It is used to close the identity page and remove any trace of the IP where the SA application is located. This is essentially a functionality to disable the OpenID after a successful or unsuccessful login attempt.
MS-6	OPDiscover	This is the discover action performed by the RP prior to when the OpenID Authentication Protocol is exercised. This is only a functionality needed when the SA runs in mOpenID mode. Essentially, it provides a web page to the RP containing pointers to where the OP endpoint is located. Generation of an XRDS document must be supported as well.

Table 3: Functional requirement specification for the mOpenID server

7. Design

7.1. Android Platform Characteristics

The SA is developed to run on the Android operating system [11], and the following creation of collaboration and class diagrams bear resemblances of this fact. Hence, some key characteristics of the Android framework are worth highlighting first and is done over the next subsections.

7.1.1. Activities

An Android application is built up of one or more activities, each run and being active one at a time. An activity provides functionalities for the user and handles all user input performed on the smartphone, as well as providing the GUI. The SA to be developed is going to have multiple activities, each providing different functionalities to the End User, e.g. the functionality to create a new OpenID Identifier.

7.1.2. Services

A service is started (and stopped) from an activity and is capable of operating in the background of any running activity. The service can do just about anything an activity can do, except displaying GUI or handling direct user input. Activities can communicate with services through intents.

7.1.3. Intents

Intents are messages broadcasted internally across a running Android operating system. These are sent and received by activities, services and other parts of the operating system. An intent can trigger actions if any receiver is listening for the given intent. Intents provide a way of communicating between different parts of the system, and is frequently used in the SA.

7.1.4. Execution Environment

As for application's execution environment, Android provides a dedicated Java Virtual Machine for any running application. This increases security as application code runs in isolation from the code of all other applications running on a smartphone.

7.2. Collaboration Diagrams

To understand how the entities in the system interact with each other to achieve a particular goal, key collaboration diagrams are constructed next. As every collaboration diagram explain the order of message execution, sequence diagrams are not constructed as they would describe the exact same interaction.

7.2.1. Start Application in mOpenID Mode

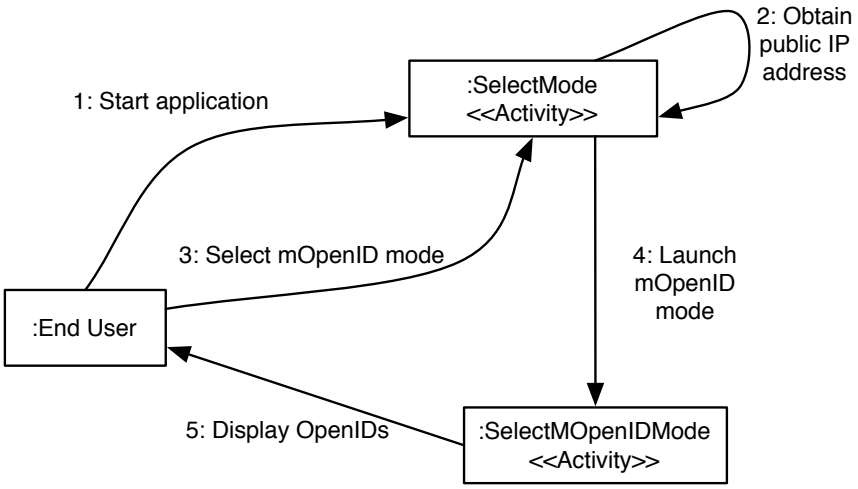

Figure 18: End User starting the SA in mOpenID mode

The collaboration diagram in Figure 18 shows the course of action when the End User starts the SA and selects mOpenID mode.

7.2.2. Creating a new OpenID in mOpenID Mode

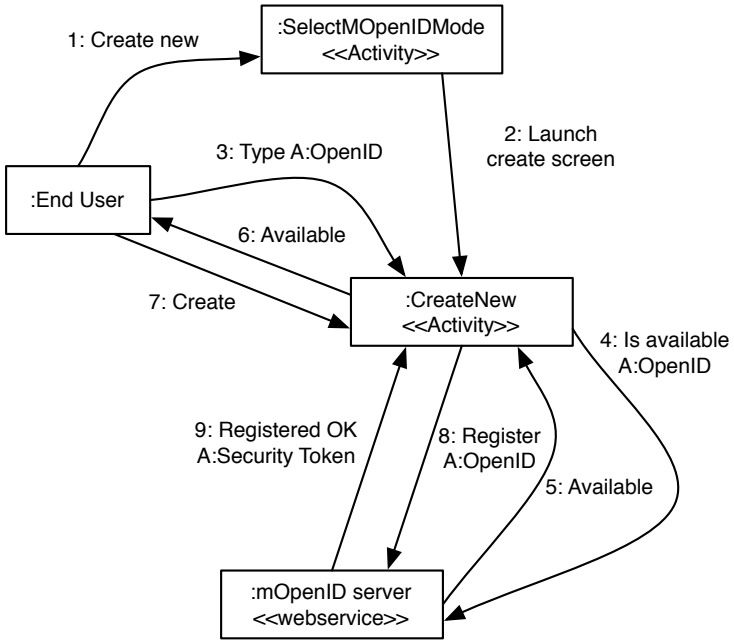

Figure 19: End User creating a new OpenID using the SA which then interacts with the mOpenID server

Creation of a new OpenID using the SA, first requires interaction with the mOpenID server in order to verify that the desired OpenID Identifier is available as shown in step 3-6. Lastly, as shown in step 7-9, the OpenID is established. On creation, the mOpenID server generates a security token that is sent to the SA,

which is stored and later used to administer the OpenID from the SA.

7.2.3. Associating with the mOpenID Server in mOpenID Mode

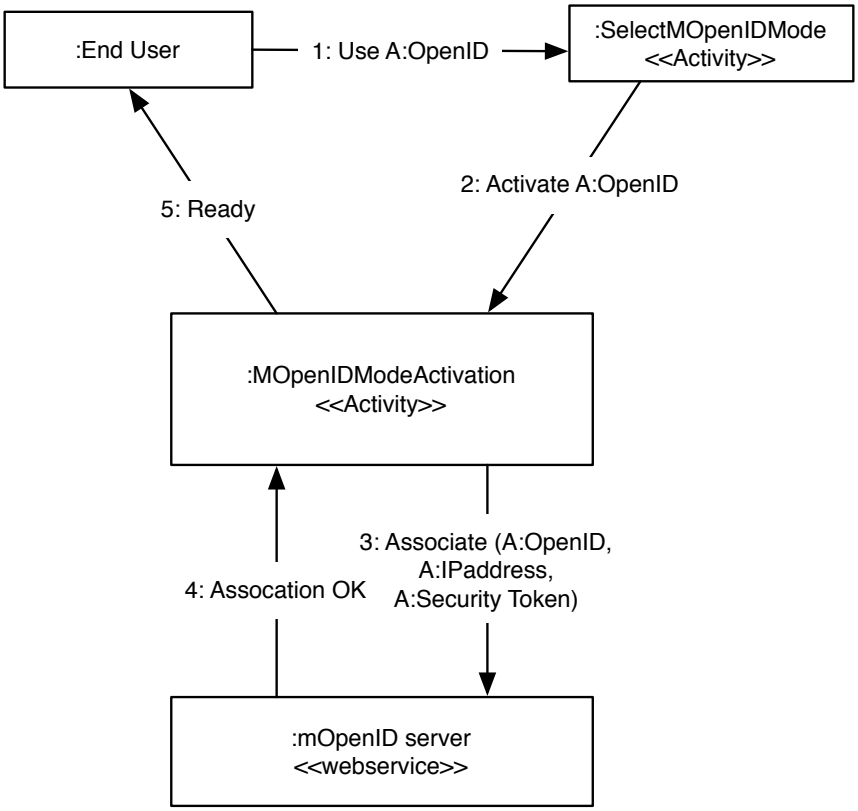

Figure 20: Association of an OpenID with the mOpenID server in mOpenID mode

Say that an OpenID (A:OpenID) has already been created on the SA. If an End User would like to log in with this particular identifier, the SA needs to associate this OpenID with the mOpenID server, resulting the mOpenID server to point to the SA as the OP endpoint. In step 3, the mOpenID server is provided with the smartphone's IP address as well as the correct security token required to perform the pointer configuration.

7.2.4. Logging in while in mOpenID Mode

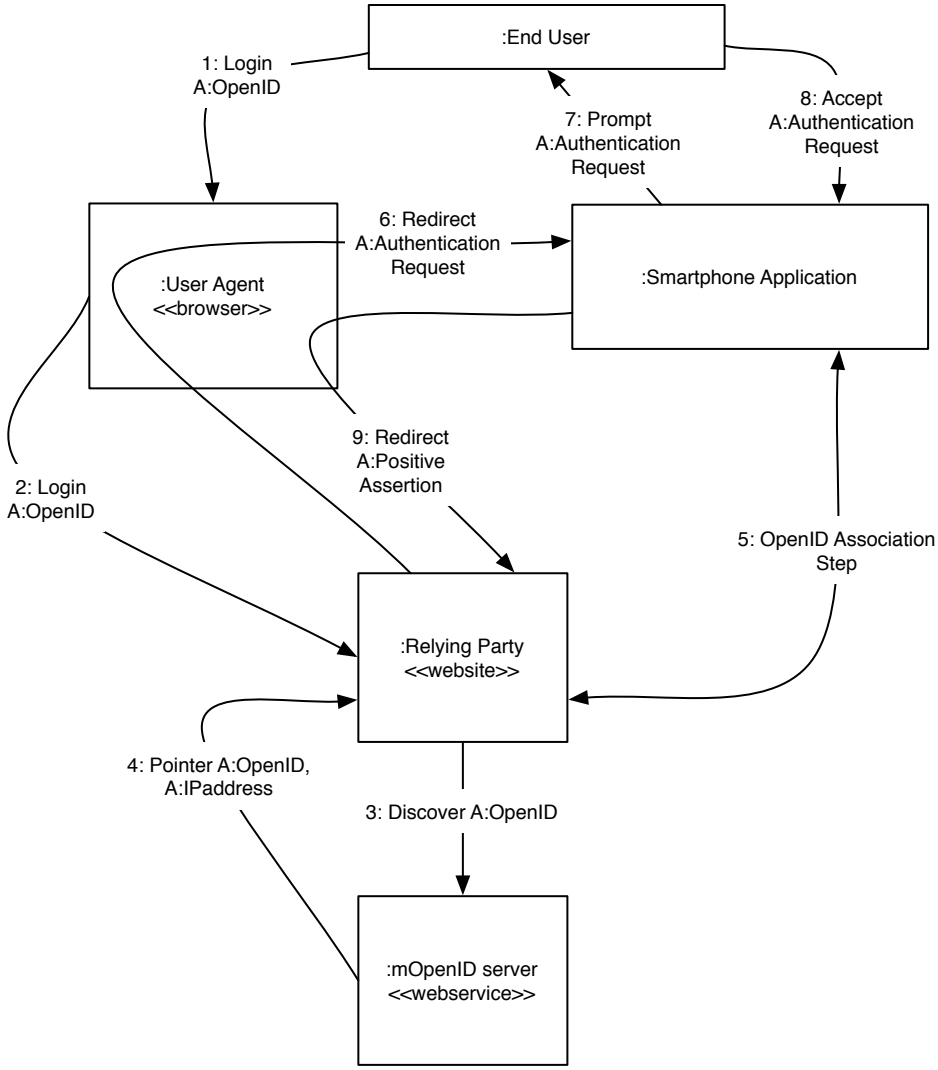

Figure 21: Showing a successful login in mOpenID mode and how the entities interact

The procedure in Figure 21 takes place right after where Figure 20 leaves off, as an association with the mOpenID server must be existing at the time of login. The message flow is initiated by the End User providing an RP with the associated OpenID through an HTML form (1). On submission through the User Agent (2), the RP discovers the OP endpoint by requesting the mOpenID server for the particular OpenID (3), resulting in the IP address of the SA (4). With the returned information about the OP in charge of the OpenID Identifier, the RP can optionally initiate an OpenID association (5) with the SA (i.e. OP) as described in section 3.2.2.2. Note that this association is different from the association described in section 7.2.3. The HTTP response to the User Agent is a `302 Found` message forwarding an Authentication Request Message to the awaiting OP running as part of the SA (6). Next, the End User will be prompted with information about the Authentication Request Message (7) on the SA. At the same time an HTML response is sent to the UA, including a button that when clicked will trigger a last request to the SA, although this is not illustrated in the

figure. As the End User accepts the request (8) on the SA and clicks the button shown in the UA, the Positive Assertion Message is created by the SA and sent back through the User Agent to the RP (9). If step 5 was skipped by the RP, a verification of the received message is it this point performed directly between the RP and the OP, although not illustrated in the figure. This completes a successful OpenID login in mOpenID mode.

7.2.5. Start Application in Static IP Mode

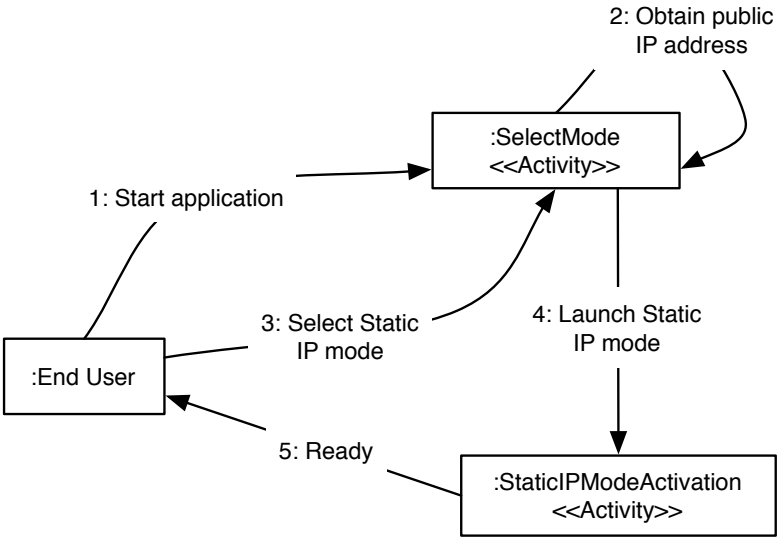

Figure 22: End User starting the SA in Static IP mode

The steps of procedure are similar to what is shown in section 7.2.1. The difference is only that the End User selects Static IP mode instead of mOpenID mode, and that no list of OpenIDs is displayed to the user (5). This is self-explanatory since the (static) IP address assigned to the smartphone, is functioning as the OpenID Identifier, hence no option for selecting a particular one is provided.

7.2.6. Logging in while in Static IP Mode

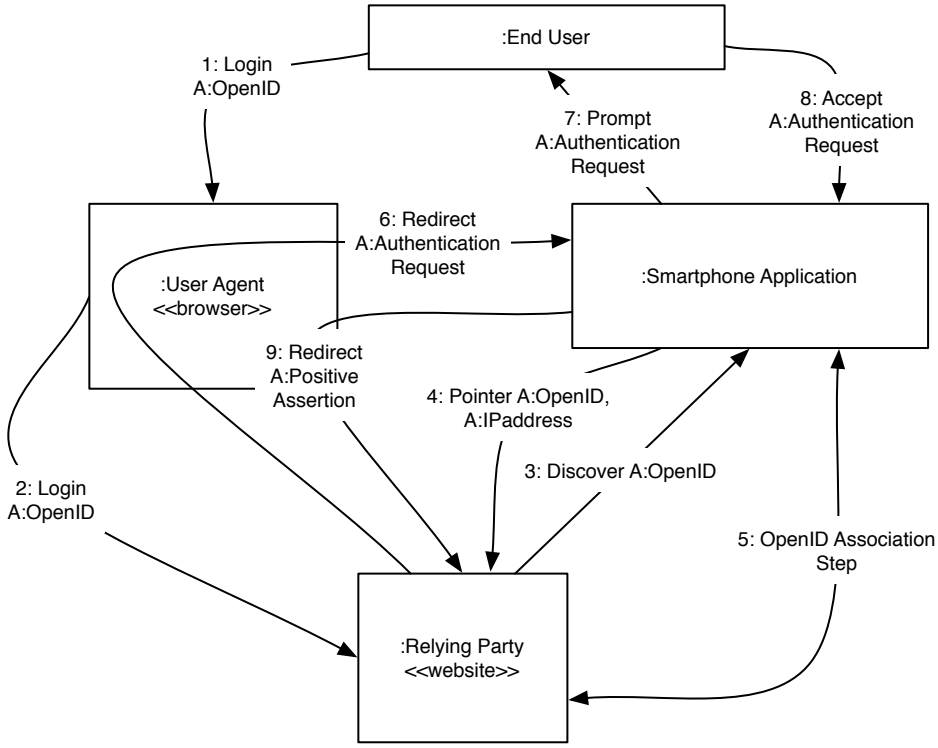

Figure 23: Showing a successful login in Static IP mode and how the entities interact

The login process is as always initiated by the End User who provides the OpenID Identifier through her UA, which in turn submits it to the RP (1-2). The discovery step (3) is, different from what happens in mOpenID mode, directed to the SA itself, hence dependence to any third party server is eliminated. The remaining steps are equal to those described in section 7.2.4. and Figure 21.

7.3. Class Diagrams

The next subsections describe what classes that are needed to implement the SA.

7.3.1. Activity Classes

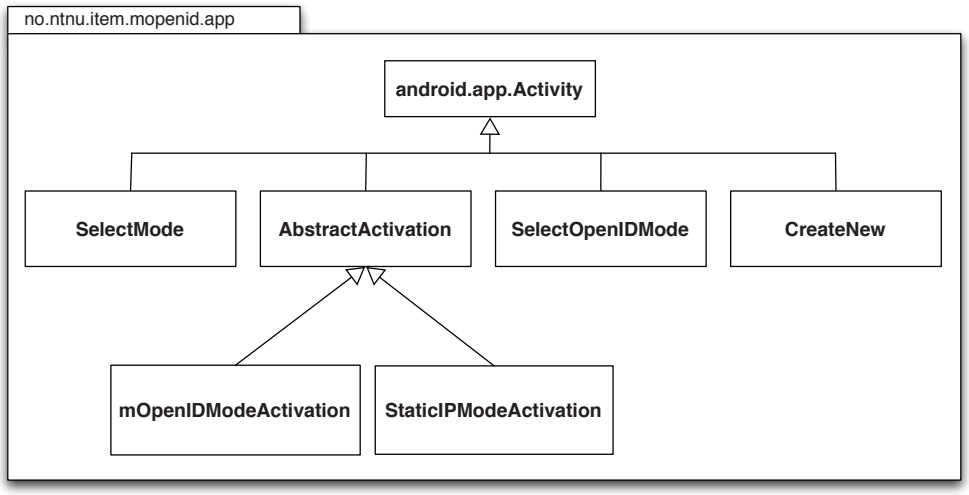

Figure 24: Above are the activity classes making up the front-end mOpenID Application

The activity classes do not provide any functionality to any of the other classes, hence no methods are visible in the class diagram in Figure 24. Instead, they use functionality provided by other classes in the `no.ntnu.item.mopenid.util` package in addition to interacting with other components through intents, as both listeners and broadcasters. Each activity class provide a GUI and triggers actions based on user input. Some of the class names can be recognized in some of the previously described collaboration diagrams.

7.3.2. Webapp Servlet Class

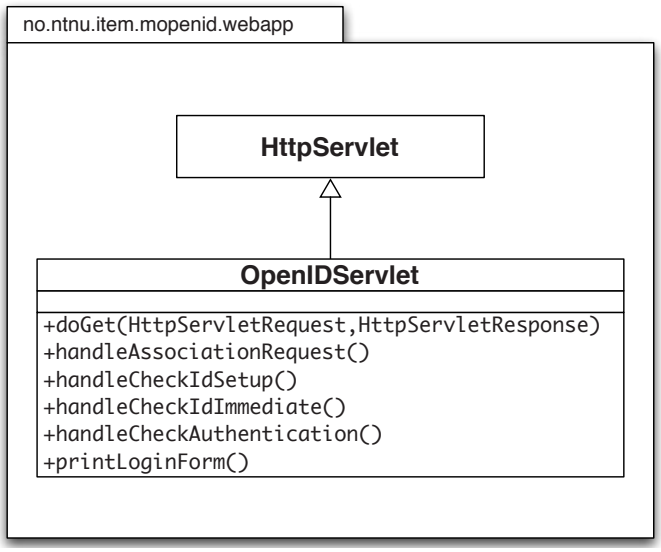

Figure 25: Servlet class handling HTTP requests required as an OP

As an OP needs to support HTTP requests and responses, a web server is required to run as part of the SA. Java Servlet Technology [14] will be used for this to process the HTTP requests. Figure 25 shows the OpenIDServlet class which will respond to HTTP requests according to the OpenID Authentication protocol. The servlet will interact with the activity classes shown in Figure 24 through intents whenever user interaction is required, e.g. when an Authentication Request Message from an RP must be approved or rejected.

7.3.3. Util Classes

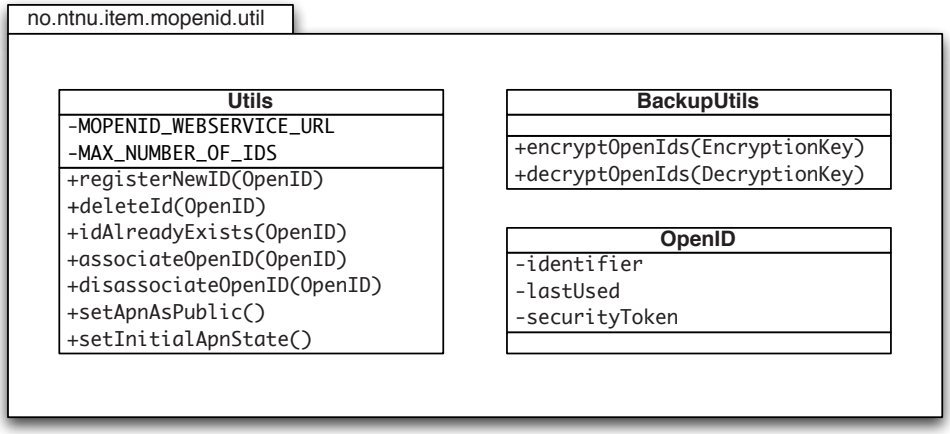

Figure 26: Util classes used by the activity classes in Figure 24

The classes in the `no.ntnu.item.mopenid.util` package provide useful functionalities for the activity classes described in Figure 24. The fields and methods in the Utils and BackupUtils classes are all defined static, hence they can be used instantly from any other class. As the method names are reasonably descriptive, no further elaboration is given. The OpenID class is a data container containing important values such as the OpenID Identifier and the security token used for configuring the MS when in mOpenID mode.

7.4. Non-Functional Requirements

As with any software system, certain non-functional requirements must be met. Current for this system are the requirements listed below

- Any sensitive data of an OpenID created on the SA must be securely stored
- The system must be easy to use
- The application responsiveness must be good
- The application must not use more resources than necessary as battery life is a constraint

8. Implementation

8.1. Deployment

The deployment diagram in Figure 27 shows how the different software pieces run on various hardware and in which way they communicate with each other. Note that the Relying Party is not implemented as part of the system, but is included in the diagram to clarify interaction with the MS and the SA.

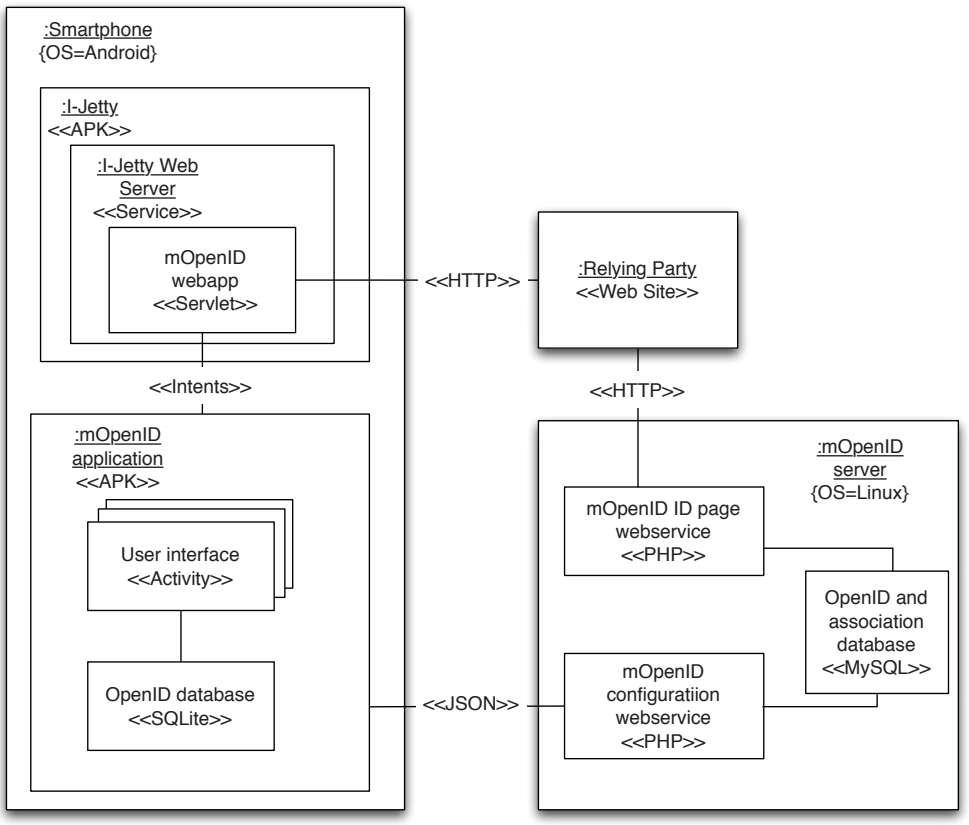

Figure 27: Deployment diagram of the various software pieces of the solution

8.2. mOpenID Server

The mOpenID Server (MS) is only in use when the SA is running in mOpenID mode. It is basically a dedicated Ubuntu (v10.04.1) server running Apache (v2.2.14), PHP (v5.3.5) and MySQL (v5.1.41). The domain name for this server decides the prefix of the OpenID Identifiers obtainable through use of the SA. For this thesis the domain name is `mopenid.item.ntnu.no`. The Apache server is equipped with a valid X509 certificate, allowing for encrypted communication between the server and both the SA and RPs. There are two independent web services running on the server; "mOpenID configuration web service" and "mOpenID ID page web service". They both share the same database containing information about the OpenIDs registered through the SA. These three components are described in the following subsections.

8.2.1. Database

The MS uses the database to store all the OpenIDs registered from user's SAs as well as to keep track of active associations at the time a particular OpenID is used for logging in at an RP. Two tables are needed for this.

Field	Type	Null	Key	Default
id	varchar(30)	NO	PRI	NULL
security_token	varchar(200)	NO		NULL
enabled	tinyint(2)	NO		1
created	timestamp	NO		CURRENT_TIMESTAMP
ip	varchar(20)	YES		NULL

Figure 28: Structure of the table `links`

The table shown in Figure 28 holds all the necessary information about the OpenID identifiers registered by users of the SA. The `id` field stores the postfix value of the OpenID Identifier, hence it must be unique. Figure 29 shows how the value is used to construct the OpenID Identifier that applies when running in mOpenID mode.

```
https://mopenid.item.ntnu.no/id/identifierexample
```

Figure 29: An OpenID Identifier having the `id` field's value set to "identifierexample"

The `security_token` field stores a random string generated by the MS when creating a new OpenID using an SA. The correct value is required for administering the OpenID at the MS. The `enabled` field tells if particular OpenID is enabled or not. The last two fields, `created` and `ip`, tells when and from which IP address the OpenID was created, respectively.

Field	Type	Null	Key	Default
id	int(11)	NO	PRI	NULL
identifier	varchar(30)	NO		NULL
destination	varchar(255)	NO		NULL
active	tinyint(4)	NO		1
last_updated	timestamp	NO		CURRENT_TIMESTAMP

Figure 30: Structure of the table `active_links`

The above table contains records of all the active associations between an OpenID Identifiers and the SAs in control of them. Rows are inserted by providing the correct security token and IP address, as demonstrated in step 3 in Figure 20. The `id` field is an auto incremented integer for each row inserted into the table, while the `identifier` corresponds to the particular OpenID Identifier for which the row applies. The `destination` field contains the URL of the OP endpoint, i.e. the location to where the RPs can find the OP in charge of this particular OpenID Identifier. It always points to a running instance of the SA

when the `active` field is set to 1. Finally, the `last_updated` field contains the timestamp when the record was last modified.

8.2.2. mOpenID Configuration Web Service

The mOpenID configuration web service provides the functionalities MS-1 to MS-5 listed in Table 3. The SA triggers these functionalities via HTTP GET requests, which are handled by scripts written in PHP running on the MS. The responses returned from the requests are JSON [7] formatted messages indicating whether a request was successful or not. Two PHP files are used for this purpose, `exists.php` and `associate.php`, and behaves as follows.

Functionality	**Request**	**JSON response**
Check if an OpenID Identifier already exists.	`exists.php?id=exampleidentifier`	if exists: `{"message":"yes"}` if not exists: `{"message":"no"}`

Table 4: Description of the behavior of exists.php

The simple code of `exists.php` makes sure to strip the id value of any harmful characters, before performing a lookup in the database to see if the OpenID Identifier already has been registered. The appropriate JSON response is returned as shown in Table 4.

Functionality	**Request**	**JSON response**
Register a new OpenID Identifier	`associate.php?a=1&id=exampleidentifier`	`{"message":"inserted", "security_token": "examplesecuritytoken"}`
Delete an OpenID Identifier	`associate.php?a=2&id=exampleidentifier&st=examplesecuritytoken`	`{"message":"ok"}`
Associate an OpenID Identifier	`associate.php?a=3&id=exampleidentifier&st=examplesecuritytoken&d=exampleOPendpoint`	`{"message":" Link associated"}`
Disassociate an OpenID Identifier	`associate.php?a=4&id=exampleidentifier&st=examplesecuritytoken`	`{"message":" Link disassociated"}`

Table 5: Description of the behavior of associate.php

From the first row in Table 5 one can see that the MS responds with a security token generated for the particular registration request. This is used in subsequent actions for controlling the OpenID Identifier, as seen in the three remaining rows. The security token value sent to the SA is merely a seeded SHA-512 hash digest of the generated value. This way, the security token stored in the SA is somewhat shorter as well as not identical to the one stored at the MS.

Note the `d` parameter (destination) in the association request which specifies where the OP endpoint is located in terms of an IP address with the `http://` prefix. In case any of the requests fail, the MS will return the JSON response `{"message":"failed"}`.

8.2.3. mOpenID ID Page Web Service

The ID page is the web page returned when submitting a regular HTTP GET request on an OpenID Identifier (Figure 29) when running in mOpenID mode. Whether a valid ID page is displayed on request or not depends on if the particular OpenID Identifier exists and is currently activated from a controlling SA. An OpenID Identifier is activated if it has an association record in the `active_links` table (Figure 30). When activated, the MS will generate an ID page as an HTML document with the necessary head tags pointing to the SA (i.e. the OP endpoint) in charge of the requested OpenID Identifier. This way, an RP will during the discovery phase (step 2 in Figure 2) find out which network entity (i.e. SA) to continue the OpenID Authentication protocol with.

```
<html>
<head>
  <link rel="openid2.provider" href="http://
188.149.199.175:8080/mopenid/server">
  <link rel="openid.server" href="http://188.149.199.175:8080/
mopenid/server">
  <meta http-equiv="X-XRDS-Location" content="https://mopenid.
item.ntnu.no/userXrds/exampleidentifier" />
  <meta http-equiv="cache-control" content="no-cache" />
</head>
<body>
  This is the identity page for the user
<code>exampleidentifier</code>.
</body>
</html>
```

Figure 31: HTML source code of an active ID page

The first `link` tag in Figure 31 points to where the RPs exercising version 2.0 of the OpenID Authentication Protocol can find the OP in charge of the current OpenID Identifier. The second `link` tag does the same for RPs only supporting the older 1.1 version of the protocol. If the RP applies the Yadis protocol when performing OP endpoint discovery, it will read the XRDS document by requesting the URL pointed to in the first `meta` tag (`X-XRDS-Location`). The returned document is shown in Figure 32. The second `meta` tag enforces requesters not to cache the document, as an outdated document can contain faulty OP endpoint URLs.

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS
    xmlns:xrds="xri://$xrds"
    xmlns="xri://$xrd*($v*2.0)">
  <XRD>
    <Service priority="0">
      <Type>http://specs.openid.net/auth/2.0/signon</Type>
      <Type>http://openid.net/signon/1.1</Type>
      <URI>http://188.149.199.175:8080/mopenid/server</URI>
    </Service>
  </XRD>
</xrds:XRDS>
```

Figure 32: XRDS document used for OP discovery if exercising the Yadis protocol

The mOpenID ID page web service fulfills the necessary functionalities for the MS-6 requirement in Table 3. For complete implementation details, the attached code coming with this report should be studied. The ID page and XRDS document generation has been somewhat aided by the Apache 2.0 licensed framework, PHP OpenID [16].

8.3. mOpenID Application

The mOpenID Application is an Android application, hence written in the Java programming language. It consists of Android activities, each providing a set of functionalities in order to fulfill the requirements in Table 2, with the exception of SA-5 as this is covered by the mOpenID Webapp as described in section 8.4. Over the next subsections, key implementation details are described. The full implementation details are included in the attached code that comes with this report.

8.3.1. Activity Overview

Table 6 shows the activities and how the functionality implementation is distributed between them.

Activity name	**Functionalities implemented**
SelectMode	Start (SA-1), Shutdown (SA-2), ModeSelection (SA-3), IDexport (SA-6.5), IDimport (SA-6.6)
SelectMOpenIDMode	IDlist (SA-6.7), IDdelete (SA-6.2)
CreateNew	IDcreate (SA-6.1)
MOpenIDModeActivation	Authenticate (SA-4), IDassociate (SA-6.3), ID-disassociate (SA-6.4)
StaticIPModeActivation	Authenticate (SA-4), Discover (SA-7)

Table 6: Android activities and the functionalities they implement

8.3.2. Maximizing Application Responsiveness

In order to make the application behave as smooth as possible, all of the I/O calls for the MS interaction and other computational demanding work is done asynchronous to the UI thread. This is achieved by extending and tailoring task-specific classes from the `android.os.AsyncTask` class included in the Android Java library.

8.3.3. Obtaining a Public IP Address.

The process of obtaining a public IP address for the SA is of highest importance for the OP to function. The solution to this problem was presented in section 5.2.3. and the smartphone is configured as follows:

1) Disable the Wi-Fi connectivity by turning it off
2) Change the Access Point Name (APN) used for connecting to the Internet through the mobile network to one that does not restrict incoming connections. Moreover, the new APN must not include the standard firewall protection and allocate a public IP address to the smartphone.

Disabling the Wi-Fi is performed easily when starting the application and the SelectMode activity is launched. In this same start-up procedure, the application switches the current APN to a public one (if it is supported) after first saving the initial APN configuration. The current implementation of the SA only supports one APN name, namely "internet". This APN has been tested to work across the main Norwegian telecom operators. Obtaining a public IP address is achieved by appending the string ".public" to the APN name. The resulting "internet.public" APN allocates the smartphone a public IP address on the network, making the SA ready for use. The described functionality is implemented in the `no.ntnu.item.mopenid.util.Util` class, illustrated by the class diagram in Figure 26. When shutting down the application, the initial configuration is restored.

8.3.4. Database

An SQLite database is used for storing the either created or imported OpenIDs in the mOpenID Application. The structure is presented in Figure 33.

Field	Type	Null	Key	Default
_id	text	NO	PRI	NULL
security_token	text	NO		NULL
description	text	YES		NULL
created	text	NO		CURRENT_TIMESTAMP
last_used	text	YES		NULL

Figure 33: Structure of the table `links` residing in the mOpenID Application

The `_id` field stores the postfix of the OpenID Identifier, hence it is defined as the primary key of the table. The `security_token` field stores the hashed digest

received from the MS when creating an OpenID. This value is regarded as the most sensitive attribute as it essentially controls to which OP endpoint the OpenID Identifier residing on the MS should point. The `description` field could contain textual information about a particular OpenID (e.g. areas of use, confidentiality level) but is not used in this thesis. The field `created` and `last_used` contains the timestamp for when a particular OpenID was created and last used for authentication, respectively.

8.4. mOpenID Webapp

As argued in section 7.3.2., a vital functionality of an OpenID Identity Provider is to be able to receive and produce responses to HTTP requests. For this, a web server must run on the smartphone at the time when an End User needs to authenticate a particular OpenID Identifier. The I-Jetty Web Server [15], a lightweight, open source web server ported from the original Jetty Web Server [22] to the Android platform, is used for this purpose. It supports the deployment and running of webapps, which are allowed access to the Android API. The Webapp implementation will use a third-party Java library called OpenID4Java [28], which provides functionalities required as an OpenID Identity Provider. Both I-Jetty and OpenID4Java are open source projects carrying the Apache Licence 2.0.

8.4.1. Intent Communication

As described in Figure 27 the Webapp communicates with the mOpenID Application through intents, allowing it to modify the latter's application state. This happens in the occasions described in the following subsections.

8.4.1.1. Static Mode Resolving

To find out if the mOpenID Application is running in Static IP mode or mOpenID mode, the Webapp sends the following intent to the mOpenID Application.

```
private static final String INTENT_ACTION_SET_STATIC_IP_MODE_
REQUEST = "no.ntnu.item.mopenid.ACTION_STATIC_IP_MODE";
...
Intent intent = new Intent(INTENT_ACTION_SET_STATIC_IP_MODE_
REQUEST);
androidContext.sendBroadcast(intent);
```

Figure 34: The Webapp requesting the mOpenID Application state by sending an intent

The mOpenID Application returns an intent with a boolean value confirming the current application state, which is read by the Webapp. The Webapp needs to be aware of the application state in order to produce (or not produce) a local ID page if receiving a discovery request from an RP.

8.4.1.2. Authentication Request

When receiving an Authentication Request Message, login approval or rejection by the End User is needed. To display a dialog asking for this, an intent is sent from the Webapp to the mOpenID Application, containing the current OpenID Identifier and RP realm that is requested.

```
private static final String INTENT_ACTION_SEND_AUTH_REQUEST =
"no.ntnu.item.mopenid.ACTION_AUTH_REQUEST";
...
Intent infoIntent = new Intent(INTENT_ACTION_SEND_AUTH_REQUEST);
infoIntent.putExtra("openid.realm", request.
getParameterValue("openid.realm"));
infoIntent.putExtra("openid.claimed_id", request.
getParameterValue("openid.claimed_id"));
androidContext.sendBroadcast(infoIntent);
```

Figure 35: The Webapp sending an authentication request intent to the mOpenID Application

As the End User accepts or rejects the request, the mOpenID will respond to the Webapp with another intent containing the result.

8.4.1.3. After Login Complete

The Webapp convey the mOpenID Application information regarding whether a login attempt was successful or not through two additional intents. One of them is used by the mOpenID Application to update the `last_used` field in the `links` table belonging to the particular OpenID that is being authenticated. The final intent is used to close the running activation activity and trigger the smartphone to go to its initial network configuration state. Details are not described here, but can be found by studying the code that comes with this report.

8.4.2. Ensure Identical Session on Login Complete

When the End User has approved (or rejected) the request on the SA, her final step is to press the "Complete login" button showing in the UA. This request will trigger the SA to produce the Positive Assertion message (given that the user has accepted the request) and return it indirectly to the RP through the UA as an HTML `302 Found` message. The Positive Assertion message is only produced and returned if the SA receives a request with the identical HTTP session that was current in the Authentication Request message. This is an extra security check to prevent an attacker from obtaining the Positive Assertion message by session hijacking [32].

9. Validation of Implementation

In this section the work will be validated by showing how the implementation fulfill the functional requirements defined in section 6.2. The first five subsections demonstrate how a corresponding collaboration diagram from section 7.2. is carried out. The last subsections describe how management functions of the stored OpenIDs are applied.

9.1. Start Application in mOpenID Mode

In the following, a test on how to start the SA in mOpenID mode is performed. The application's starting point is as if it just has been installed. Moreover, no OpenIDs have been created beforehand using the particular SA. When an End User starts the SA by clicking its application icon, the SelectMode activity is launched as shown in Figure 36.

Figure 36: SelectMode activity showing a progress bar dialog immediately after startup, making the network connection ready

Figure 37: SelectMode activity ready for the user to select which mode to run the application in

In Figure 36 the SA tries to obtain a public IP address indicated by the dialog box. Performed in the background, the Wi-Fi connection is being turned off and the APN name changed to one that will allocate a public IP address. The procedure will fail if it has not succeeded within 18 seconds, or if the initial APN name is unsupported by default. Note that nor a Wi-Fi or mobile data network connectivity icon is shown in the status bar (top). If the procedure is successful the progress dialog will disappear and the SelectMode activity screen in Figure 37 is shown, ready for user input.

Figure 38: SelectMOpenIDMode activity showing an empty OpenID list

Next, when the user presses the mOpenID mode button displayed in Figure 37 the activity switches to the SelectMOpenIDMode activity displayed in Figure 38. Note the I-Jetty icon in red and green in the top left corner of the figure, as the SelectMOpenIDMode activity starts the mOpenID Webapp (the web server) on creation. Since no OpenIDs have been established yet, the list supposed to show the stored OpenIDs is empty. The next procedure will explain how creation of a new OpenID is carried out when the user presses "Create new".

The described procedure satisfy the SA-1 and SA-3 requirements.

9.2. Create a new OpenID in mOpenID Mode

Shown in Figure 39 is the CreateNew activity awaiting for the user to input text to produce a desired OpenID Identifier. As soon as the user starts typing, the red text above the input field switches color to orange, indicating that the current OpenID Identifier is too short. Immediately when the user have entered five or more characters, the orange OpenID Identifier turns yellow at the same time as the "Check availability" button is enabled. When the button is pressed a request is made to the MS to find out whether the desired OpenID Identifier is available or not. Figure 41 shows that a requested OpenID Identifier is available and ready to register, and as the user presses the register button a final registration request is sent to the MS. The security token is extracted from the successful response and stored in the SA's database. Finally, the CreateNew activity is closed and the SelectMOpenIDMode activity is shown, presenting the newly created OpenID.

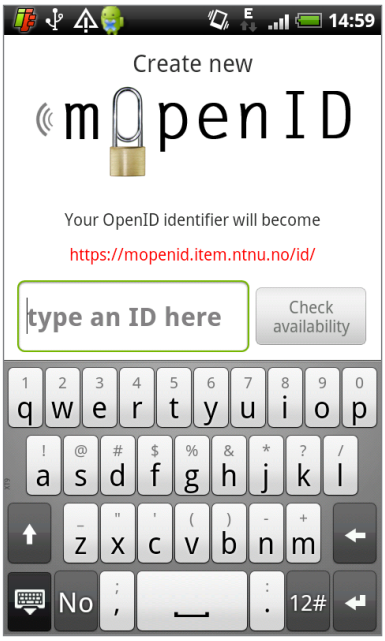

Figure 39: CreateNew activity ready for the user to type a desired OpenID Identifier

Figure 40: The user have now entered a sufficiently long enough OpenID Identifier, and can check its availability by pressing the button

Figure 41: The response from the MS confirms that the queried OpenID Identifier is available

Figure 42: After creation, the SelectMOpenIDMode activity contains the newly established OpenID

The described procedure satisfy the SA-6.1, SA-6.7, MS-1 and MS-2 requirements.

9.3. Associating with the mOpenID Server

The first step of a login process is to select the desired OpenID. This is done by choosing from the OpenID list shown in Figure 42 (only "idexample" showing). This will bring up the MOpenIDActivation activity, initially showing a progress dialog while associating, or linking, the particular OpenID with the MS as shown in Figure 43.

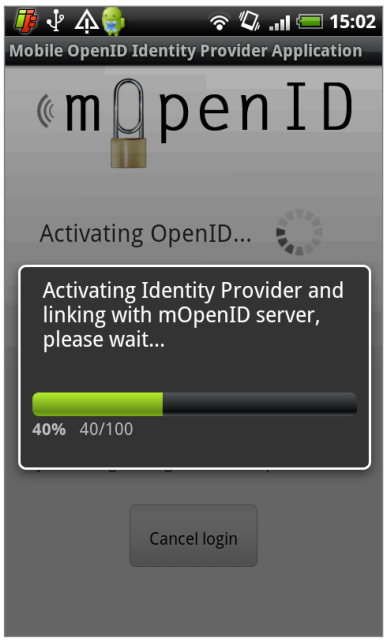

Figure 43: The MOpenIDModeActivation activity initially associating with the MS

Figure 44: MOpenIDModeActivation activity ready for an RP to request authentication of the particular OpenID Identifier (green)

After the association step has finished, the SA shows the MOpenIDModeActivation activity as presented in Figure 44. At this point of time, the MS will generate the correct ID page if the OpenID Identifier is requested by an RP, containing a pointer to the SA as its controlling OP endpoint. A record has been inserted into the `active_links` table in the database that is part of the MS as shown in Figure 45.

```
+----+------------+-----------------------------------------+
| id | identifier | destination                             |
+----+------------+-----------------------------------------+
| 46 | idexample  | http://188.149.192.182:8080/mopenid/server |
+----+------------+-----------------------------------------+
```

Figure 45: The newly inserted record in the active_links table in the MS database

This record is used in building the ID page when the OpenID Identifier is requested by an RP, in order to point to the controlling OP, namely the associated smartphone running the SA. At this point the SA is ready to be used for authenticating the particular OpenID Identifier.

The described procedure satisfy the SA-6.3 and MS-4 requirements.

9.4. Logging in when the SA is running mOpenID Mode

It is time to perform the actual login process by using the OpenID at a Relying Party. For this example Stack Overflow [36], a highly regarded and serious questions and answers web site for computer programming, will be used as the Relying Party.

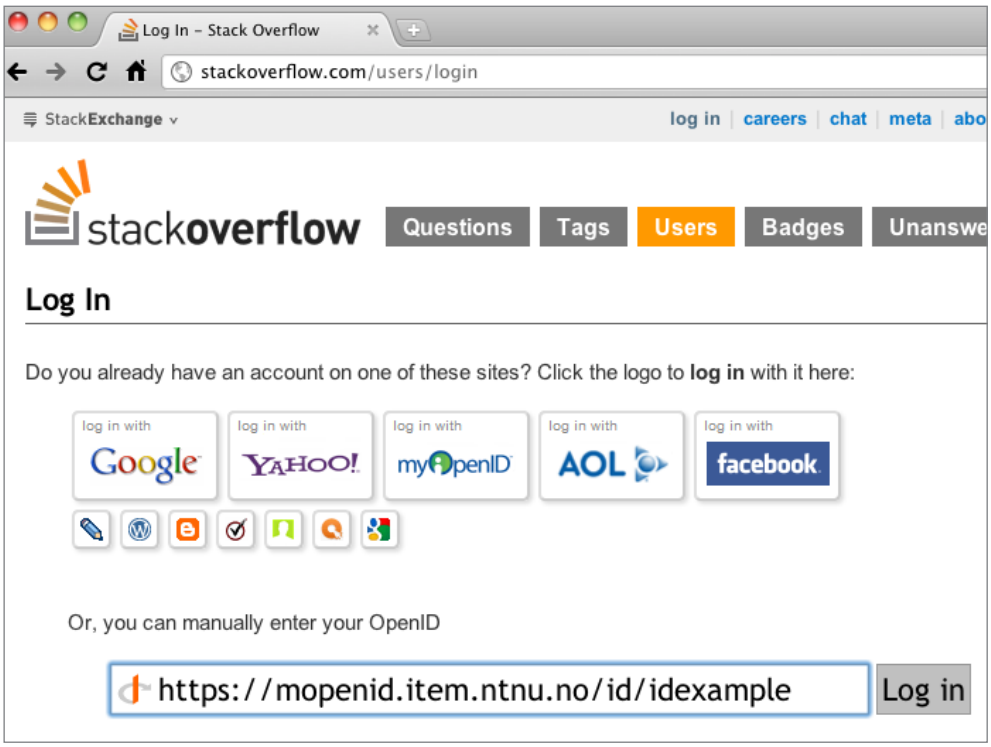

Figure 46: The RP login web page, where the user have entered its OpenID Identifier

Before pressing the login button shown in Figure 46 it is evident that the SA is awaiting an Authentication Request message as indicated in Figure 44. When the button is pressed the RP performs discovery on the user-provided OpenID Identifier to locate which OP that controls it. The MS renders the ID page for the particular OpenID Identifier, which contain a pointer to the current controlling SA. This pointer, or URL, is used by the RP to route the UA to the SA with the Authentication Request message. As the SA receives the message, a dialog is shown with the options to either confirm or reject the request as shown in Figure 47. Concurrently, the SA responds with the web page shown in Figure 49 to the UA.

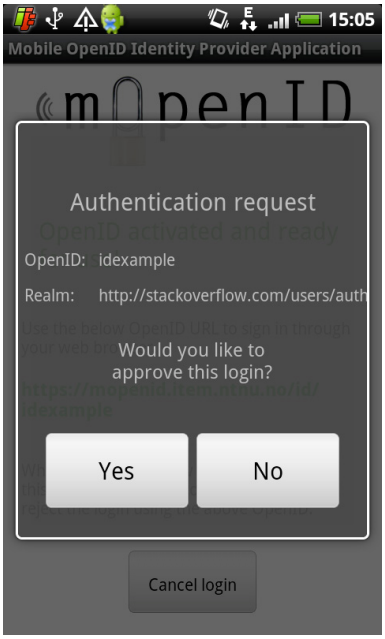

Figure 47: The confirmation dialog on whether to accept or reject the Authentication Request message

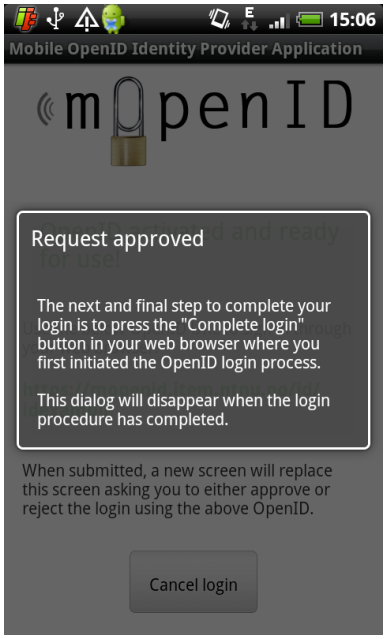

Figure 48: The final confirmation dialog after the user have accepted the request in Figure 47

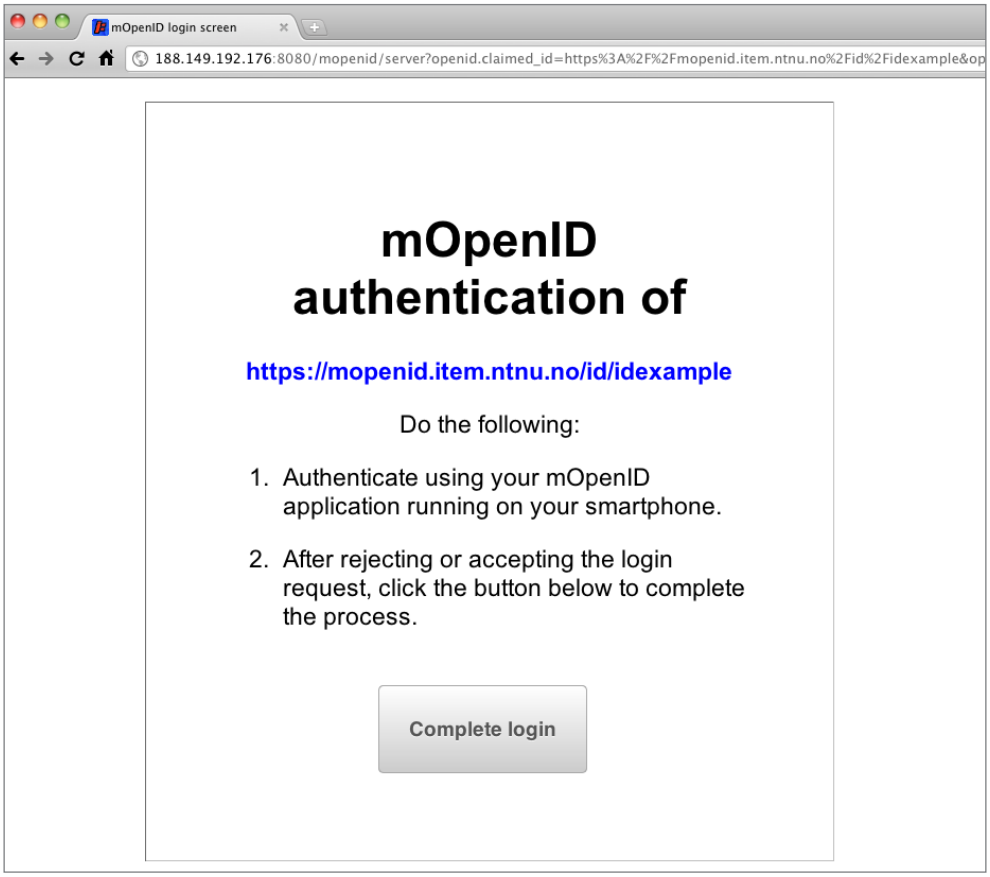

Figure 49: Showing the HTML response received from the SA after requesting the Authentication Request message

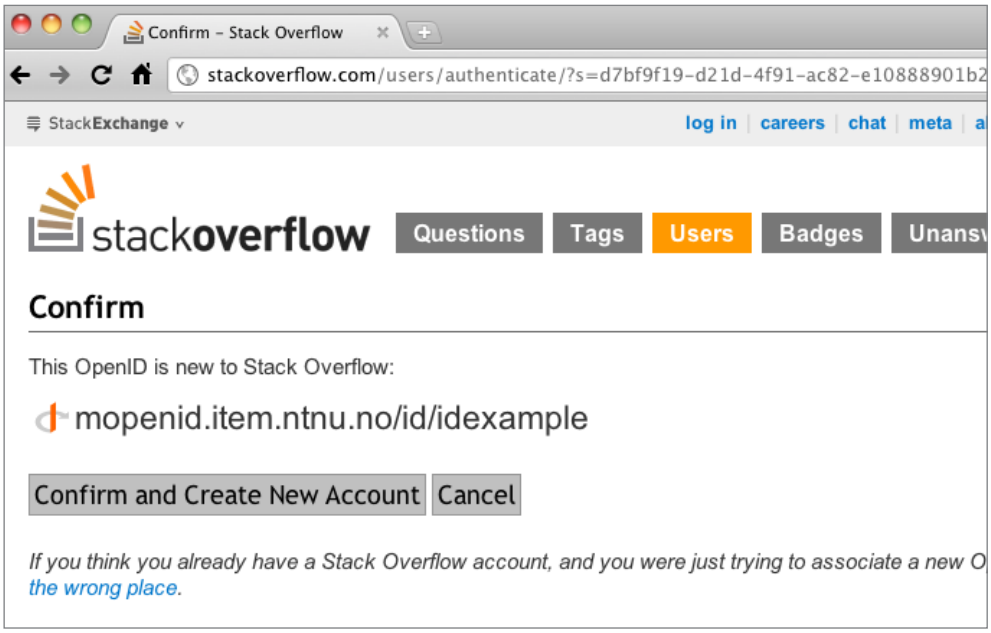

Figure 50: Showing the RP having accepted the authenticated OpenID after receiving the Positive Assertion message from the SA

After the user confirms the login request on the SA, and presses the "Complete login" button in the UA shown in Figure 49, the Positive Assertion message is generated at the SA and responded to the RP through the UA. Figure 50 shows the RP having approved the login of the provided OpenID Identifier. Meanwhile, the SA has disassociated the OpenID Identifier with the MS, restored the smartphone's initial network configuration, shutdown the local web server and returned to the SelectMode activity. This concludes the login procedure when running the SA in mOpenID mode.

The described procedure satisfy the SA-2, SA-4, SA-5, SA-6.4, MS-5 and MS-6 requirements.

9.5. Logging in when the SA is running Static IP Mode

The steps in this procedure are almost identical to those applied in the previous section. Only the minor differences are presented next.

By selecting "Static IP mode" button when the SelectMode activity is showing, the StaticIPModeActivation activity is started as displayed in Figure 51. From the figure one can see the OpenID Identifier derived from the static IP address assigned to the SIM. Additionally, the running web server knows that it is running in Static IP mode, hence it will generate an ID page locally when the OpenID Identifier is requested by an RP in the discovery phase.

The functionality provided fulfills the SA-2, SA-3, SA-4, SA-5 and SA-7 requirement.

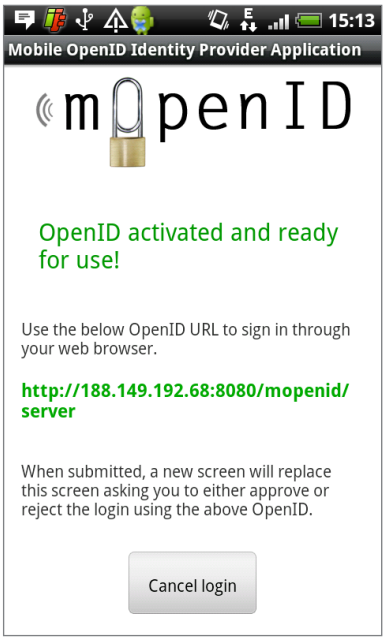

Figure 51: StaticIPModeActivation activity ready for Static IP mode authentication

9.6. Delete an OpenID Identifier

When the list over OpenIDs is showing in the SelectMOpenIDMode activity, a user can long-press any of them in order to delete it. The context menu in Figure 52 shows two options for deleting the OpenID "idexample". One can either delete the OpenID completely, removing both the local instance as well as the instance stored on the MS, or it can be deleted locally only. The latter is mostly used for testing related to the export and import functionalities, as it simulates a loss of OpenID control.

Figure 52: Context menu showing two ways of deleting an OpenID Identifier

The functionality to delete an OpenID fulfills the SA-6.2 and MS-3 requirement.

9.7. Exporting and importing OpenID Repositories

The export functionality comes in handy in order to backup the established OpenIDs on the SA to prevent loss of authority over the OpenID Identifiers stored on the MS. The user can bring up the export and import options by pressing the Android native menu button on the smartphone from the SelectMode activity as displayed in Figure 53. If pressing the "Export OpenID(s)" button, an encryption key ("secret") must be provided by the user as displayed in Figure 54.

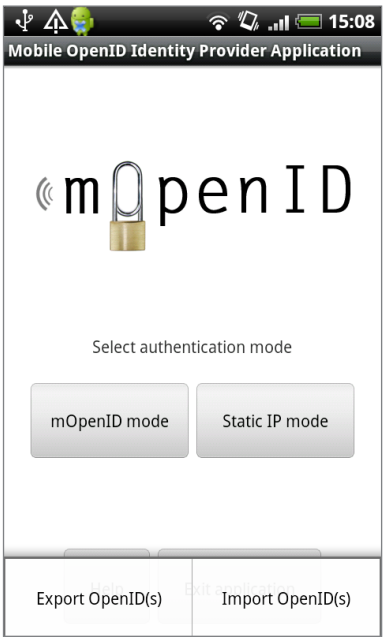

Figure 53: The SelectMode activity when the Android native menu button is pressed

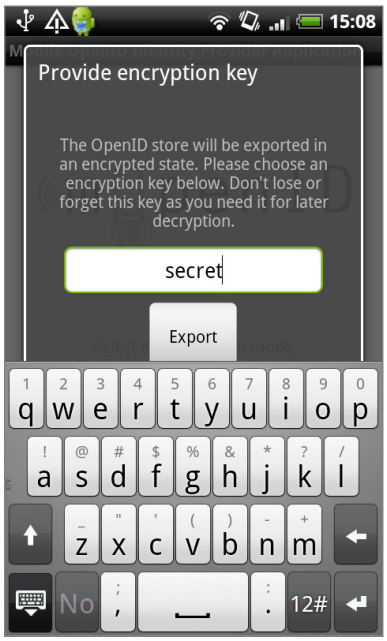

Figure 54: A dialog allowing the user to enter an encryption key during the export procedure

The OpenIDs are encrypted using AES-128 encryption and saved by default as plain text on the external storage medium of the smartphone. Additionally, the user is asked to export the encrypted file to other applications on the smartphone which accepts the `text/plain` application type. Figure 55 shows the particular menu for this functionality, having the Dropbox application selected.

When importing an encrypted OpenID repository this can be performed in two ways as displayed in Figure 56. One can either import the encrypted file from a URL, be it a local file URL or one resolving to an Internet location, or one can paste the contents of the decrypted file in a text field provided by the SA. In both cases the correct decryption key must be provided for the import to be successful.

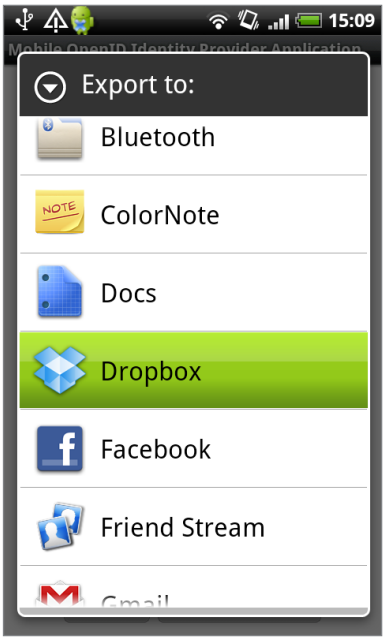

Figure 55: Menu allowing to export the encrypted OpenID(s) to other applications on the smartphone

Figure 56: Two ways of importing OpenIDs to the SA

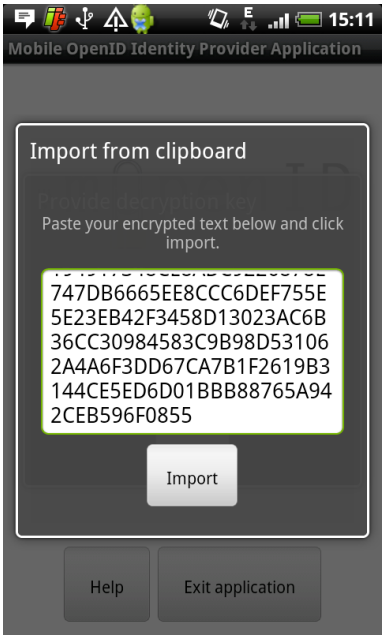

Figure 57: Pasted encrypted text awaiting to be decrypted to one or more OpenIDs

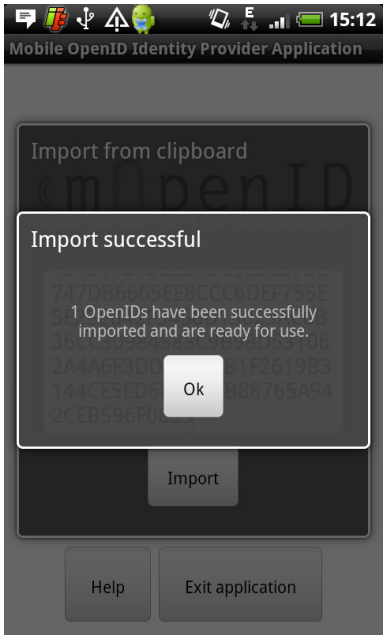

Figure 58: A dialog confirming a successful import of an OpenID

The described export and import functionalities fulfill the final SA-6.5 and SA-6.6 requirements.

10. Limitations and Security Issues

Just as any other software system, the implementation suffers from security issues and limitations. As the software solution, from its own perspective, depends on both external and internal variables, the report will elaborate on drawbacks split into two categories. Firstly, the external technical factors embracing the SA, such as cellular network availability, will be presented. Secondly, the internal technical factors of the application, such as weaknesses related to the actual implementation, will be addressed. Finally, a few other limitation inadequate to be placed in the former two categories are briefly discussed.

10.1. External Technical Factors

The external technical factors are those factors independent of the application itself. Moreover, they are thought of as the embracing entities, environment and processes outside of the application code's domain, which operation are purely limited by themselves or other entities they have a relation to.

Firstly, a security issue with using a smartphone application as an OP, is addressed. The benefit of allowing smartphone owners to install third party applications such as the proposed OP application, has a counteractive side effect as well. Other applications can be installed on the smartphone, also those with malicious purposes. Imagine the threat a malicious application could pose if it unconditionally responds positively to any Authentication Request message received. The execution of this application could happen unknowingly to the smartphone user, and concurrent or separate to the execution of the genuine OP application. If carrying out the OpenID Authentication protocol correctly, it would go ahead and authorize any RP authentication request sent to the reportedly legit OP running on the smartphone. The described situation is not particularly feasible if the OP was running on a closed system or server such as in traditional schemes.

Secondly, unfavorable implications arise from being dependent on yet another network entity, namely the MS. As the mOpenID mode is likely to be the most used mode, the solutions is useless if the MS becomes inaccessible, as SAs are unable to configure their OpenID Identifiers for logging in at an RP. In addition to this, if the database residing on the MS is compromised, all its controlled OpenID Identifiers can be configured to point to any OP owned by an attacker. Also, if using mOpenID mode the user must trust the owner of the MS.

Thirdly, the proposed solution will be useless in the rare case when lacking a cellular network supporting data traffic. This will fail because the OP application has no way of communicating with Internet nodes to carry out the OpenID Authentication protocol.

Fourthly, successful administering of the OP application involves that proper routines are maintained for backing up its identity repository. This must be performed manually, hence it relies on the user to remember to do so. But failing to

do so has consequences in the case when the End User either loses her phone or has her identity repository accidentally deleted. Any relation between the lost OpenID identity and authorized RPs is hard to reestablish as the security tokens are lost.

Fifthly, there is an odd occasion when an RP could fail in interacting with the OP due to the following occurring steps:

1. Association is performed between the SA and the MS
2. The smartphone is allocated a new IP address due to e.g. switching from GRPS to UMTS
3. RP-OP interaction fails as the RP does not find a valid OP endpoint at the outdated IP address retrieved from the MS in the discovery phase

This clarifies the weakness of using the SA in mOpenID mode when a change of IP address is likely to occur.

Lastly, common external factors exists. Among these, insufficient battery lifetime will limit the operation of the OP. Further, the smartphone's operating system and hardware may not be able to provide the OP application the necessary system resources to either operate or even be installed at all.

All of the above constitute limitations hindered by external technical factors for the implemented system to fully, or in part, fulfill the necessary functioning as valid entities in the OpenID framework.

10.2. Internal Technical Factors

The internal technical factors are interchangeable with the factors of the application's internal components, processes or functionalities. During the system development process decisions have been made that reduces system security. However, these are acknowledged in the following and should in further work be dealt with orderly.

Firstly, the SA stores sensitive information about each established OpenID Identifier in an insecure manner, as the database which keeps system critical values such as the security token is left unencrypted. However, some security constraints exist by default as only the application process itself is allowed to read and write from the file that constitute the database. If rooting the smartphone users and applications are allowed privileged control within Android's Linux subsystem [42]. If such conditions apply, other applications can access the private folder structure allocated to the SA, possibly revealing the security token used to control the identity. A simple solution to this could be to temporary decrypt the database file using a user-provided decryption key while the application is open. However, problems arise if the application is closed in an unexpected way leaving the temporary decrypted file persistent in the memory of the smartphone. A more advanced and highly secure solution would be to store the keys in hardware such as a smart card chip inherent in the smartphone designed for

the purpose. An example of such a chip is the PN65K from NXP which in addition to triggering a self-destruction mechanism if it is tampered with, provides secure storage of sensitive data [26].

Secondly, the interaction between the SA and the MS when establishing new OpenID Identifiers and associating existing ones is performed using JSON/HTTP. Most preferably, this should happen through an encrypted channel such as HTTPS. However, when this was tried in the implementation phase, the Java `SSLException: Not trusted server certificate` was thrown. The outcome from strenuous debugging was unsuccessful.

Further, having the possibility of starting and stopping the SA on demand has until now been presented purely as beneficial. However, as OpenID supports Single Sign-On (SSO) by keeping an authenticated HTTP session between the End User and the SA, this session will be destroyed should the End User shut down the SA. Obviously, the destruction of the session is done for increased security so that no undesired person could exploit the previously authenticated session between a UA and the SA, if the concerned person gain control of the smartphone. The point is however, that the capability of shutting down the SA limits the End User to enjoy the benefits of SSO.

There are drawbacks using intents for interaction between the Webapp and the mOpenID Application as these are broadcasted throughout the Android operating system. The intents are subject to be sent and picked up by other applications installed on the smartphone, hence undesired SA behavior could occur. Preferably, the Webapp and mOpenID Application should be implemented as one single entity.

This thesis have previously studied existing work on how to enhance authentication between an OP and an End User. Besides the possession of the smartphone, the implementation lacks an actual authentication before allowing a user to approve the Authentication Request messages. However, it is believed that despite its simplicity the implementation should eliminate the risks of phishing attacks, at least when compared to the way attacks are carried out when authenticating purely through a web browser. Although not implemented in the final solution, little work must be carried out for a minimal authentication feature to be realized.

These are probably not all internal technical factors that bring about security issues and limitations to the solution. In software development, weaknesses are identified over time, which likely will happen if the work is studied and by third-parties

10.3. Other Factors

Non-technical factors such as economical measures may put limitations on the End User preventing her from taking use of mOpenID.

As argued in section 6.1.2. a special SIM is required for the Static IP mode to function. Naturally, this comes as an expense to the End User, as the issuing telecom operator is likely to charge money for the service. Hence, limitations on economical grounds might prevent a user to utilize the solution.

11. Conclusion

The purpose of the previous work has been to make the OP setup procedure simple enough to be performed by even the novice user. The work has made possible an enhanced user-centric approach of utilizing the login functionalities provided by the OpenID identity system. Not only is private control of an OP made possible for an increased number of users, but more secure authentication can be performed enabled by the phishing-resistant, out-of-band and physical interaction with the OP. As the product of this thesis is merely a proof of concept, further work should look into applying more advanced authentication of the user managing OpenID logins through the SA.

Weaknesses and security issues concerning the solution have been identified and discussed. Although they vary in gravity, many of them are believed solvable if spending more time and gaining experience with the Android operating system. Should the area of application enabled by obtaining a public IP address be inspiring to others, application developers must possess a good understanding of the risks involved with publicly exposing a smartphone on the Internet.

The most recent statistics published by OpenID Foundation in 2009 counts over 9 million Relying Parties on the Internet. The SA that is implemented as part of this thesis is believed to add a new and interesting approach to how regular users can sign in to these web sites using their smartphone.

References

[1] Allen, T. (2009, September 25). *OpenID: Now more powerful and easier to use!* Retrieved February 11, 2011, from OpenID web site: http://openid.net/2009/09/25/more-powerful-and-easier-to-use/

[2] Bakken, E., Jørstad, I., Eliasson, C., Fielder, M., & Thanh, D. v. (2009). Releasing the potential of OpenID and SIM. *In: Proceedings of the 11th International Conference on Intelligence in Next Generation Networks (ICIN 2009)*. IEEE Press.

[3] Chappell, D. (2006, April). *Introducing Windows CardSpace*. Retrieved February 11, 2011, from Microsoft Developer Network web site: http://msdn.microsoft.com/en-us/library/aa480189.aspx

[4] Chen, Y.-K., Lin, Y.-b. (2005, February). *IP connectivity for gateway GPRS support node*. Wireless Communications, IEEE , vol.12, no.1, pp. 37- 46. Retrieved May 23, 2011, from IEEE Xplore web site: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1404571&isnumber=30466

[5] claimID. (2010). *claimID.com – Manage your online identity*. Retrieved February 11, 2011, from claimID web site: http://claimid.com

[6] Cordance, D., & Epok, D. (2005, November 14). *Extensible Resource Identifier (XRI) Syntax V2.0*. Retrieved April 22, 2011, from XRI Syntax Specification web site: http://www.oasis-open.org/committees/download.php/15376/xri-syntax-V2.0-cs.html

[7] Crockford, D. (2006). *The application/json Media Type for JavaScript Object Notation (JSON)*. Retrieved May 15, 2011, from IETF web site: http://www.ietf.org/rfc/rfc4627.txt

[8] Delft, B. v., & Oostdijk, M. (2010). A Security Analysis of OpenID. *POLICIES AND RESEARCH IN IDENTITY MANAGEMENT*, 343, 73-84.

[9] Dropbox. (2010). *Online backup, file sync and sharing made easy*. Retrieved April 2, 2011, from Dropbox web site: https://www.dropbox.com

[10] Glässer, U., & Vajihollahi, M. (2010). Identity Management Architecture. *Security Informatics, Annals of Information Systems*, 9, pp. 97-116.

[11] Google. (2010, December 7). *The Developer's Guide*. Retrieved May 2, 2011, from Android web site: http://developer.android.com/guide

[12] Google. ADT Plugin for Eclipse. Retrieved May 30, 2011, from Android web site: http://developer.android.com/sdk/eclipse-adt.html

[13] Haverinen, H., & Salowey, J. (2006, January). *Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM) RFC 4186*. Retrieved February 11, 2011, from IETF web site: http://datatracker.ietf.org/doc/rfc4186/

[14] Hunter, J., Crawford, W. (2001). *Java servlet programming*. Sebastopol, CA: O'Reilly Media.

[15] I-Jetty. (2008, January 5). *i-jetty: webserver for the android mobile platform*. Retrieved April 21, 2011, from Project Hosting on Google Code web site: http://code.google.com/p/i-jetty/

[16] JanRain Inc (2011). *PHP OpenID library*. Retrieved May 15, 2011, from their GitHub web site: https://github.com/openid/php-openid

[17] Jøsang, A., & Pope, S. (2005). User Centric Identity Management. *AusCERT Conference 2005*. Brisbane: Proceedings of AusCERT 2005.

[18] Kissel, B. (2009, December 19). *OpenID 2009 Year in Review*. Retrieved May 5, 2011, from OpenID Foundation web site: http://openid.net/2009/12/16/openid-2009-year-in-review/

[19] Leung, D. (2010, October 28). *Sign up for Flickr with your Google Account! Flickr Blog*. Retrieved February 12, 2011, from Flickr Blog web site: http://blog.flickr.net/en/2010/10/28/sign-up-for-flickr-with-your-google-account/

[20] Lindholm, A. (2009). *Master of Science Thesis: Security Evaluation of the OpenID Protocol*. KTH Royal Institute of Technology, School of Computer Science and Communication. Stockholm: Royal Institute of Technology.

[21] Miller, J. (2006, March 18). *Yadis 1.0*. Retrieved February 12, 2011, from Yadis.org web site: http://yadis.org/wiki/Yadis_1.0_(HTML)

[22] Mort Bay Consulting. (2011). *jetty - Jetty Webserver*. Retrieved May 25, 2011, from codehaus foundation web site: http://jetty.codehaus.org/jetty/

[23] Myers, S., & Jakobsson, M. (2006). *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wiley-Interscience.

[24] myID.net. (2010). *myID.net – OpenID Service*. Retrieved February 11, 2011, from myID.net web site: http://www.myid.net/

[25] myOpenID. (2008). *Welcome to myOpenID*. Retrieved April 15, 2011, from myOpenID web site: https://www.myopenid.com

[26] NXP. (2006, October 26). PN65K Objective short data sheet. Retrieved June 2, 2011, from AdvanIDe web site: http://www.advanide.com/datasheets/sfs_pn65k_rev1_3.pdf

[27] OASIS Standard. (2005, March 15). *SAML Specifications*. Retrieved October 5, 2010, from SAML XML.org web site: http://saml.xml.org/saml-specifications

[28] OpenID4Java Library. (2011). *openid4java - OpenID 2.0 Java Libraries*. Retrieved May 23, 2011, from Project Hosting on Google Code web site: http://code.google.com/p/openid4java/

[29] OpenID Foundation. Retrieved May 25, 2011, from OpenID Foundation web site: http://www.openid.net

[30] OpenID Foundation. (2007, December 5). *OpenID Authentication 2.0 - Final*. Retrieved February 15, 2011, from OpenID Foundation web site: http://openid.net/specs/openid-authentication-2_0.html

[31] OpenID Foundation. (2008, December 30). *OpenID Provider Authentication Policy Extension 1.0*. Retrieved April 19, 2011, from OpenID.net web site: http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.txt

[32] OWASP. (2009, May 27). *Session hijacking attack*. Retrieved June 2, 2011, from The Open Web Application Security Project web site: https://www.owasp.org/index.php/Session_hijacking_attack

[33] Rescorla, E. (1999, June). *Diffie-Hellman Key Agreement Method*. Retrieved March 2, 2011, from The Internet Engineering Task Force web site: http://www.ietf.org/rfc/rfc2631.txt

[34] Sakimura, N. (2009, March 9). *NTT docomo is now an OpenID Provider*. Retrieved February 11, 2011, from OpenID Foundation web site: http://openid.net/2010/03/09/ntt-docomo-is-now-an-openid-provider/

[35] Shepard, L. (2009, May 18). *Facebook Supports OpenID for Automatic Login*. Retrieved May 29, 2011, from Facebook developer web site: https://developers.facebook.com/blog/post/246/

[36] Stack Overflow. Retrieved May 20, 2011, from Stack Overflow website: http://stackoverflow.com/users/login

[37] Thanh, D. v. (2000). ADPO Project Development Methodology. R&D Telenor, Oslo.

[38] Thanh, D. v., Jørstad, I. (2007). The Ambiguity of Identity. *Telektronikk* , 103 (3/4), 3-10.

[39] Tsyrklevich, E., & Tsyrklevich, V. (2007). OpenID – Single Sign-On for the Internet: A Security Story. *Blackhat Conference 2007*. Las Vegas: Blackhat USA.

[40] Urien, P. (2010). An OpenID Provider based on SSL Smart Cards. *Consumer Communications and Networking Conference (CCNC)* (pp. 1-2). Las Vegas: IEEE

[41] Wikipedia. (2010, October 25). *List of web service specifications*. Retrieved May 14, 2011, from Wikipedia web site: http://en.wikipedia.org/wiki/List_of_Web_service_specifications

[42] Wikipedia. (2011, May 15). *Rooting (Android OS)*. Retrieved May 29, 2011, from Wikipedia web site: http://en.wikipedia.org/wiki/Rooting_(Android_OS)

[43] Wood, L. (2009, September 21). *Sun's OpenID IdP: Data Governance.* Retrieved February 11, 2011, from Anyway – meandering thoughts from Lauren Wood web site: http://www.laurenwood.org/anyway/2007/09/suns-openid-idp-data-governance/