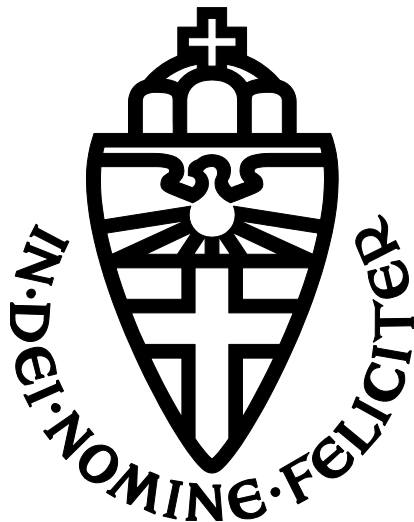


Master Thesis

How to use the CAcert infrastructure within an OpenID context?



Auteurs: B.J. Derlagen (s0829692), O.M. Berfelo (s0756067)
Educational institution: Radboud University
Faculty: Faculty of Science
Institute for Computing and Information Sciences
Information Science
Education:
Thesis number:
Supervisors: Prof.dr. B.P.F. Jacobs, dr.ir. E. Poll
Date:



Preface

We wrote this master thesis for our education Information Science at the Radboud University in Nijmegen. We have done a research of how we can use the CAcert infrastructure in an OpenID context. This is the result of our research.

We would like to thank some people who have made a contribution to this master thesis that led us to the final result. We would like to thank our supervisors Prof.dr. B.P.F. Jacobs and dr.ir. E. Poll for their feedback, knowledge, ideas and advice. We also would like to thank Esther Makaay and Marco Davids of SIDN for the conversation we had with them. Finally we would like to thank Bernard van Gastel and Marc Overmeer for the ideas and advice they gave us.

Barthold Derlagen & Onno Berfelo. Warnsveld, 3 September 2010.

Abstract

These days more and more people have access to an internet connection. These people spend an increasing amount of time on the web. On the web you'll encounter many websites on which you'll need to authenticate yourself. In many cases you'll do this with a username and password combination. Using the same combination on every site is unwise. This is where Single Sign On (SSO) (section 2.3.1) gets into the picture. SSO is an Identity Management system (section 2.3). SSO allows users to use a single account on one site to gain access to multiple other websites. Even in the scenario of SSO there are still some pitfalls in using a username and password combination. An attacker has several options to gain access to your account. He could use a keylogger to log your credentials. He could also attempt to guess your password or use brute force. Using a certificate from a Certificate Authority (CA) based on the Public Key Infrastructure (PKI) (section 2.2) would resolve the security issues of logon credentials.

OpenID is a SSO solution but it has several security problems. Most of these problems would be solved by requiring the usage of Secure Sockets Layer (SSL). The usage of a username and password as logon credentials is common in OpenID implementations and it could be a problem if an attacker were to obtain these credentials. This might be resolved by using different login credentials.

CACert is a CA that issues certificates based on a Web of Trust (WoT). CACert failed a management audit and because of this its root certificate has not been included in any of the major browsers. This results in error and warning messages whenever someone visits a website with a certificate that has been signed by CACert. This has halted the deployment of CACert certificates. Using CACert client certificates as a logon credential would give it more purpose. Perhaps it could even stimulate the growth of CACert.

OpenID (chapter 3) is an open standard, it is open source. It is a software framework which means that apart from the core code most of the code can be altered. There are therefore many different implementations of the OpenID protocol. OpenID is decentralized which means that authentication doesn't need to take place on the site that offers the service. Within OpenID there are three parties, the User, Identity Provider (IdP) and Relaying Party (RP). The IdP provides the user with an identity and an identifier. The user can provide his identifier to the RP. The RP will then redirect the user to the IdP. The user will authenticate himself to the IdP. The IdP redirects the user back to the RP. The RP then accepts that the user has identified himself. OpenID has some security and trust problems (section 3.2).

CACert (chapter 5) is not unlike a common CA (section 2.2). It does, however, use a WoT to verify the identity of their users. CACert has assurers which are users with 100 or more assurance points who have successfully taken an assurer test. Assurers can grant users from ten up to thirty-five points depending on their rank. Their rank is determined by the amount of people they have assured or being active during the start-up process of CACert. A user needs to print a verification form. He then needs to provide this form and show his identification papers to an assurer. The assurer can then grant the user points. Once a user has 50 or more points he is deemed assured which will unlock various options in generating certificates. User can then obtain class 3 certificates and include his name in his certificates.

During the creation of this thesis an idea arose to develop a proof of concept (chapter 6). This proof of concept was to use a CACert client certificate as a logon credential on an IdP. The



proof of concept is based on a package named SimpleID. This package was selected after comparing it with several other packages. Some of the scripts of the SimpleID were altered to enable user login with a CAcert client certificate. The credentials of users are stored in a database.

Table of Contents

1	Introduction.....	8
1.1	Problem statement	8
1.1.1	Research questions	9
1.2	Method.....	9
1.3	Thesis structure.....	10
2	Theoretical framework.....	11
2.1	Security Goals.....	11
2.1.1	Confidentiality.....	11
2.1.2	Integrity	11
2.1.3	Availability.....	12
2.1.4	Authentication	12
2.1.5	Accountability	13
2.2	Public Key Infrastructure (PKI)	13
2.2.1	X.509 (Public-key certificate)	15
2.3	Identity Management	16
2.3.1	Single Sign On (SSO)	16
3	OpenID.....	17
3.1	Protocol.....	17
3.1.1	Enter OpenID identifier.....	20
3.1.2	Discovery	22
3.1.3	Association request	23
3.1.4	Association response	24
3.1.5	Authentication request.....	25
3.1.6	Request authentication and authorization	25
3.1.7	Authenticate and authorize.....	26
3.1.8	Positive assertion.....	27
3.1.9	Verification.....	28
3.2	Problems	29
3.2.1	Trust problems.....	31
3.3	Alternative OpenID solutions	31
3.3.1	OpenID Plus	31
3.3.2	Personal Identity Portal	32
3.3.3	MyOpenID	32
3.4	Protocol in other research	32
4	Single Sign On comparison	34



- 4.1 Kerberos..... 34
 - 4.1.1 Protocol 35
 - 4.1.2 Comparison with OpenID 39
- 4.2 DigiD 41
 - 4.2.1 Protocol 41
 - 4.2.2 Comparison with OpenID 44
- 4.3 Microsoft Windows LiveID..... 45
 - 4.3.1 Protocol 46
 - 4.3.2 Comparison with OpenID 47
- 4.4 Google Federated Login 48
 - 4.4.1 Protocol 49
 - 4.4.2 Comparison with OpenID 50
- 5 CAcert..... 52
 - 5.1 What is CAcert 52
 - 5.1.1 Certification Authority 52
 - 5.1.2 Web of Trust..... 57
 - 5.2 How to use CAcert 57
 - 5.2.1 Client Certificate 58
 - 5.2.2 Server Certificate..... 58
- 6 Proof of Concept 62
 - 6.1 Motivation for our solution..... 62
 - 6.2 Design Decisions 63
 - 6.2.1 Database 63
 - 6.2.2 Script 64
 - 6.3 Package selection..... 65
 - 6.3.1 OpenID Implementations 65
 - 6.3.2 Chosen package..... 67
 - 6.4 CAcertID 68
 - 6.4.1 Credential Database..... 68
 - 6.4.2 Background 69
 - 6.5 Results 69
 - 6.5.1 Advantages 70
 - 6.5.2 Disadvantages..... 70
 - 6.5.3 Notes..... 70
 - 6.5.4 Future work 70
- 7 Conclusion 71
 - 7.1 OpenID 71
 - 7.2 CAcert..... 71



7.3 Proof of Concept..... 72

7.4 Future Work..... 72

Bibliography..... 73

Attachment A: Setting up a Webserver with SSL enabled 75

Attachment B: Database..... 83

Attachment C: Proof of Concept Code 92

1 Introduction

More and more people are using the Internet. The usage of the Internet and websites is growing by the day. Websites are getting more personalized. This asks for the need to authenticate users. The problem with this is that users need to register on many websites. Another problem that occurs is that a lot of times the username that you want to choose is already in use, so you need to come up with another username. So users need to remember a lot of usernames and password combinations. For this problem there is a solution that is called Identity management systems. Identity Management Systems allow users to use a single identity on multiple websites. One example of an Identity management system is OpenID.

CAcert is a Certificate Authority that allows people to register certificates for free. They sign server certificates and generate client certificates for people that request them. People that join CAcert need to be assured by assurers. When an assurer assures an assuree, he signs the client certificate of the assuree. Assurers can grant from 10 to 35 points depending on their status. The status of an assurer is determined by the amount of people they have assured and in some cases being active in the startup process of CAcert. Once an assuree has received 100 points, he can become an assurer after taking a test. Services like signing a server certificate do not require an assuree to have 100 points.

1.1 Problem statement

Various sources take note of the security issues that exist within the OpenID protocol. [TSY07], [OH08], [MOS09] and [MCD08] all mention several security flaws in the OpenID protocol. The issue that is mentioned most is its susceptibility to phishing attacks, followed by man in the middle attacks. The problem underlying is that the use of SSL is not required by the protocol. This problem is, however, noticeably not the highest priority in this thesis. Most OpenID identity providers use a username and password combination as login credentials. This means that when an attacker obtains these, he will have access to all the websites that were accessed by the user with this OpenID. Our aim is to see whether it is prudent to replace the username and password combo with a more secure credential in the form of a certificate.

CAcert has an entirely different problem which frustrates its deployment. The root certificate of CAcert is not included in the bigger browsers. This is due to issues in the management structure which made it fail an audit [MOZI07]. This failure leads to the abandonment of the inclusion request in the Mozilla browser (Firefox, Seamonkey). Therefore certificates that are handed out by CAcert can be used but will generate a warning in most applications. In short our aim is to give the CAcert certificates another purpose. This purpose will be a logon credential for OpenID.

1.1.1 Research questions

Our main research question is:

How to use the CAcert infrastructure within an OpenID context?

We also have some sub research questions to help to achieve our main goal:

1. How does OpenID work?
 - a. What are the differences and similarities between DigiID and OpenID?
 - b. What are the differences and similarities between Kerberos and OpenID?
 - c. What are the differences and similarities between Windows LiveID and OpenID?
 - d. What are the differences and similarities between Googles ID and OpenID?
2. How does CAcert work?
3. How can OpenID and CAcert complement each other?
 - a. What are the advantages and disadvantages of authentication with CAcert?

1.2 Method

In essence our thesis is mostly based on a literature study. We started by searching and studying literature about CAcert, OpenID and the security problems with OpenID. When we had written the chapters about these subjects we continued by studying literature about protocols that we were planning to compare with OpenID. We also developed a proof of concept to show how people would be able to login on an OpenID Provider with the use of a CAcert client certificate. We have written down how to setup a webserver to be able to accomplish this in Attachment A. We selected a package to base our proof of concept on by comparing a few packages. We then modified the selected package to make it possible to sign in with a CAcert client certificate. We have done this by editing files and added some new scripts. To be able to do this we have searched for information about how this can be done.

1.3 Thesis structure

This thesis contains the following chapters:

1. Introduction
2. Theoretical framework
3. OpenID
4. Single Sign On comparison
5. CAcert
6. Proof of Concept
7. Conclusion
8. Reflections
9. Bibliography

Chapter one is an introduction to our thesis. In the second chapter we discuss some general theories and research behind our thesis. The third chapter describes the OpenID protocol and some of its flaws. The fourth chapter describes some other Single Sign On protocols that are compared with OpenID. The fifth chapter describes what CAcert is and how it can be used. In the sixth chapter we describe our proof of concept. The seventh chapter contains the conclusion of our thesis. In chapter eight you can find the bibliography.

2 Theoretical framework

In this chapter we will discuss some of the underlying theory behind our research. We will start by describing general security aspects in the first section. In that section we will elaborate on confidentiality, integrity, availability, authentication and accountability. In the second section we will give a short summary of the Public Key Infrastructure (PKI). PKI and specifically the X.509 protocol is what the CAcert protocol is based on. The third and last paragraph of this chapter is about Identity Management. It will briefly explain Identity Management and Single Sign-On (SSO). We will name some of the solutions available. These solutions have their own chapter in which examples such as Kerberos and DigiD will be described (chapter 4). OpenID based Alternatives for our own project, CAcertID are mentioned in the OpenID chapter (chapter 3).

2.1 Security Goals

The most important goals of security are confidentiality (sometimes referred to as secrecy), integrity (occasionally referred to as accuracy) and availability. These goals as well as other goals, Authentication and Accountability, will be discussed further in the following paragraphs. In this thesis we will in various chapters talk about the assurance level of a protocol. By a high assurance level we mean that it is safe to be used in online banking and other vital services. A low assurance level means that it is best to be used only for chatting, blogging, etc.

2.1.1 Confidentiality

Confidentiality can be described as ensuring that only those authorized to have access have access. It is one of the most important aspects of information security [GAN91]. In other words, confidentiality is a restriction of access to information. It resembles the need-to-know policy used in military organizations and by governments [GAN91]. Organizations have a lot of confidential data from development plans to trade secrets. Disclosure of these data could be disastrous [TAN03]. Organizations other than companies have confidential data as well. Patient data is highly confidential and this is a factor in the development of the Electronic Patient Dossier (EPD). Just think of the implications that would arise if your patient files were visible for a future employer.

2.1.2 Integrity

Integrity is about the message not being altered along the way. In other words the message the sender sent is equal to the message the receiver receives. An analogy would be that the letter you sent is not altered along the way. Non-repudiation is stronger than Integrity as it means that a sender can not deny sending the message. To the earlier analogy this would mean the envelope is sealed with your personal seal or you hand your letter over in person. Online you would be able to achieve this with the use of a hash function and digital signature.

2.1.3 Availability

The uptime of a computer shows how long it has been available for use. The availability in information security is similar; it is the degree to which something is available for use, when it needs to be up. You can even calculate this by dividing the uptime with the total time passed. Availability is also one of the more pronounced aspects of information security. In the OpenID protocol the Identity Provider (IdP) needs to be available to be able to logon to a Relaying Party (RP). An analogy of this could be the copyright protection of new single player games. Some of the recently released games like Command and Conquer 4 require players to be online while playing. The game is authenticated by an authentication server. When the authentication server is down you can't play. With both you are dependent on a third party (authentication server/ IdP) to be able to use the service (game/ RP). In both cases the service can no longer be used if the IdP/ authentication server shuts down permanently.

2.1.4 Authentication

Ensuring that someone is who he claims to be is in short the goal of authentication. During our daily lives we authenticate ourselves often and not just when we go online. When we meet an individual we know, in person or in a video conference, we can authenticate him by his appearance and voice. The voice can also be used to identify someone whilst speaking on the phone or when we use a voice over IP (VoIP) application. When we logon to our computer we authenticate ourselves to the local system. When we logon to our Windows LiveID account we authenticate ourselves to Microsoft's Windows LiveID servers. Whilst logging in to gmail, youtube or any other google product we authenticate ourselves to one of Google's servers. In [KAU02] authentication is defined as the manner in which you can trustworthily identify a person's or object's identity. You can authenticate yourself with:

- Something you know (username and password)
- Something you are (biometrics)
- Something you have (key, certificate or smartcard)
- Somewhere you are (location of payment, IP, etc.)

The username and password combination is perhaps the most used authentication method on the web and it is also the simplest one to set up. It is however not a very secure authentication method, even if the protocol setup for it is secure. Users tend to use the same password for various uses. Some only have one password they use for all their accounts on the web. If an attacker were to obtain your credentials he would be able to logon to any of the sites you use them on. An attacker could create a malicious site in which you're actually giving up your credentials freely. Another attack the attacker could employ would be a keylogger. A keylogger logs all keys pressed so if you login to a site the keylogger will have your credentials. A different issue is the man in the middle attack. The attacker intercepts your message and sends it himself which makes it appear as if he is the one logging in. A replay attack might also be possible. During a replay attack an attacker resends a previously sent message by the user. A replay attack can be prevented with the use of a nonce.

2.1.5 Accountability

Accountability means that actions made by some party (who is responsible for their actions) can be traced back and those responsible can be found. Most often traceability is created by logfiles or other forms of methods that can record actions. A stronger form of accountability is non-repudiation. In this case the party cannot deny that he did not do it [TAN03]. Authorization is a tool that extends the execution of accountability and improves it. This means that authorization in a way makes it possible to trace back the actions that have been made by some party. Authorization can be described as the process which authorizes parties to access resources.

2.2 Public Key Infrastructure (PKI)

Public key infrastructure (PKI) is a method for digital security. PKI makes it possible to verify the identity of parties (sender and receiver) involved in electronic communication. PKI provides the users with a public and private key. The difference between the two keys is that the private key needs to be kept private and the public key needs to be made public. These two keys match as a pair (mathematically). This means that if you use one key to create an electronic message, it can be read by using the other key. An example of how PKI works in practice (Figure 1): Alice (sender) wants to send an electronic message to Bob (receiver). Alice creates an electronic message. To make sure that Bob knows that the message is from Alice and hasn't changed she needs to do a few things. The first thing that Alice needs to do is to create a hash value of the electronic message. When the hash value is created Alice needs to encrypt the hash with her private key. By encrypting the hash Alice's identity or digital signature is assigned to the message. The encryption makes it impossible for others to change the message without raising red flags. Alice sends the signed message (plaintext electronic message and digital signature) to Bob.

Bob receives the two messages. Now Bob needs to do a check to make sure that the received message hasn't changed and does come from Alice. The first thing that Bob needs to do is to decrypt the digital signature with the public key of Alice. The result of this action is a hash value. The next step is that the Bob must create a hash value of the plaintext electronic message. When this is done Bob can check if the two hashes values are the same. If this is the case then Bob can be sure that the message is from Alice and hasn't been changed [LAN03].

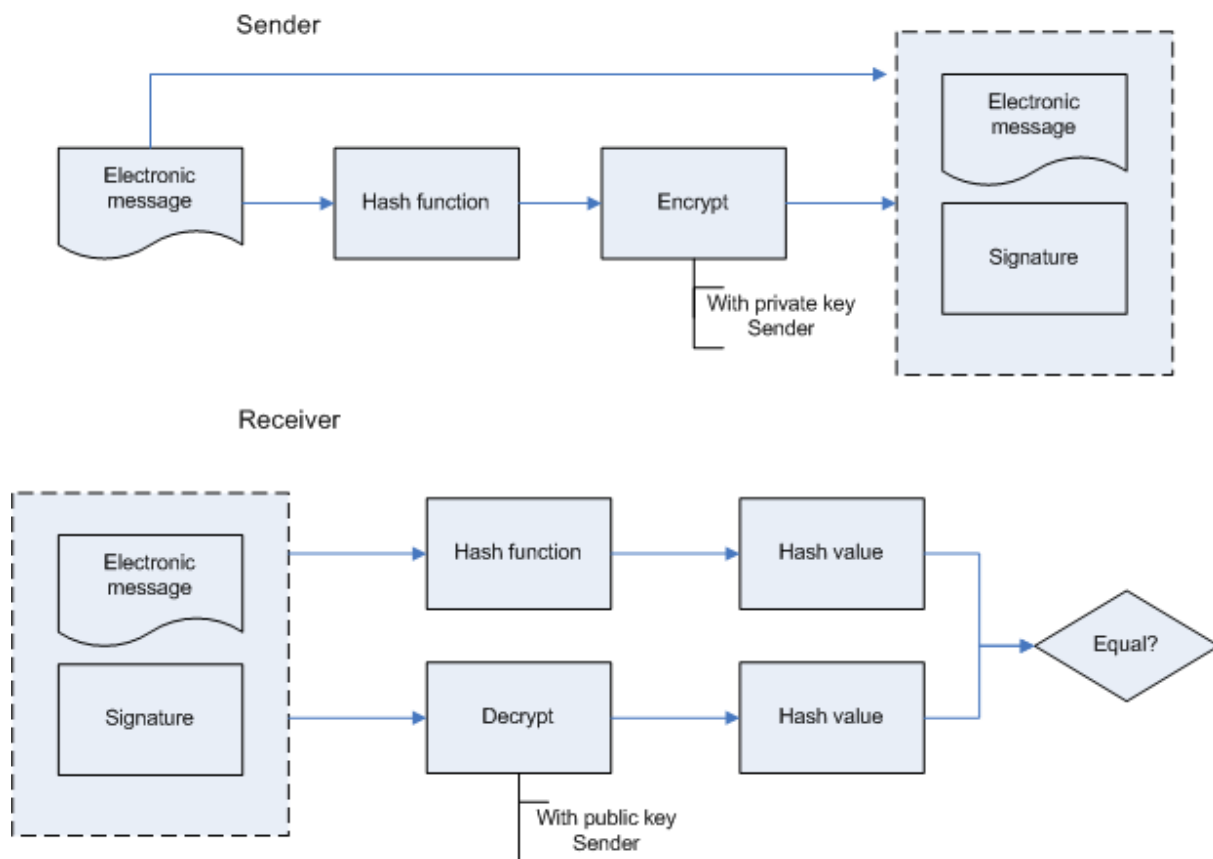


Figure 1: Example signing and verifying electronic message

PKI in the sense of digital certificates is a way of structuring and defining standards for the distribution and control of digital certificates. In a standard PKI setup certificates are used to bind the public key of the user to the user’s identity. An organization that does this is called a Certification Authority (CA). The CA doesn’t need to know the user’s private key [BES06]. For example if a user wants to prove to another user that he is that certain person, the user can go to the CA with his public key and passport and ask if he can be certified.

A commonly used component of PKI is Registration Authorities (RA). RA performs several administrative tasks of a CA. One of the tasks that a RA performs is to verify the entity’s identity and determine if the entity is entitled to have a public key certificate issued. Another task that the RA performs is to enforce all policies and procedures that are defined. Other tasks could also be assigned to the RA, it differs from one CA to the next [WEI01].

Certificate directory is also a commonly used component of PKI just like RA. The purpose of a certificate directory is publishing certificates in a directory that is controlled by the CA or the RA. This could also be done by a different organization but that makes it more difficult. If it is handled by the CA or RA, it is easier to keep it up to date. A requirement of a certificate directory is that it must be publicly readable. In the certificate directory the Certificate Revocation List (CRL) is also published. CRL is a list where the certificates of entities are published that are no longer valid [WEI01].

The functions that are involved with PKI are:

- Public key cryptography: This is the creation, distribution, administration and control of the cryptographic keys.
- Certificate issuance: Bind the public key to an entity or person. It could also be used to bind a public key to an attribute [TAN03].
- Certificate validation: Validating that a trust relationship or binding exists and that the certificate is valid.
- Certificate revocation: Canceling certificates and publish them on the CRL

Example of how PKI works when creating a certificate:

1. A user generates a public/private key pair. (this could be done by the user himself or by the CA or the RA)
2. The user provides his public key, name and other information that is required to the RA.
3. The RA checks if the credentials are valid and sends this information to the CA.
4. The CA will then generate a certificate for the user's public key and other information and signs the certificate with the CA private key.
5. End result is that the user has a public and private key and a public key certificate.

2.2.1 X.509 (Public-key certificate)

X.509 is the standard for certificates (PKI for Single Sign On). The International Telecommunication Union (ITU) and International Organization for Standardization (ISO) published the standard in 1988 [JOO99]. Since the publishing of the standard there have been three versions. The latest version V3 was released in 2008 ([RFC 5280](#)). A public-key certificate is a certificate that is signed by a CA (a person or entity) to confirm the identity or other information of the holder of the certificate [JOO99]. An X.509 certificate consist of the following elements:

- Version of certificate (version of the X.509 certificate V1,V2 or V3)
- Serial number certificate (unique to identify the certificate)
- Signature algorithm (the algorithm that is used to sign the certificate)
- Issuer's X.500 name (name of the certificate authority)
- Validity period (start and end date of the validity of the certificate)
- Subject name (the entity or person that is being certified)
- Subject's public key (the public key of the subject)
- (optional) Issuer ID (unique identifier of the CA issuer, option since version V2)
- (optional) Subject ID (unique identifier of the CA subject, option since version V2)
- Extensions (there are many extensions defined in the certificate, possible since version V3)
- Signature (the signature of the certificate authority)

Source: [TAN03], [JOO99], [NIS01].

The addition of extensions field (since version V3) makes the certificates more open. An advantage of this is that the certificate issuers can add their own extension types [JOO99].

2.3 Identity Management

On the web an identity is something one must have to perform certain tasks. Whether you are tweeting, commenting on a Youtube video or replying to what your friend said on Facebook. You will need to identify yourself. An identity should always point to a single user but a user can have multiple identities [HEL09]. This is however not always the case as more and more corporations have an identity which is handled by several employees. We disagree that an identity should point to a single user. An identity should point to a single entity. This entity could be a person, organization or a group.

The identity you claim must be authenticated in some way. Usually this is done by using a username and password combination which was chosen by the user. This is in essence the same as printing your own passport [HEL09]. You can create a Twitter or Facebook account under the name of a VIP but this doesn't mean that you are this VIP. Identity Management is in short the process of managing identities. To us this means from both the client side as well as the server side.

2.3.1 Single Sign On (SSO)

These days people do more and more online and have a vast number of accounts. Each account has logon credentials, which mostly consist of a password and username combination. Most users use the same credentials for various accounts. Other users will find it hard to avoid this as the number of accounts they have grows. This is where SSO comes in.

"SSO stands for Single Sign-On and in its simplest form means a way where a user can access several applications using centrally managed account information and performing authentication only once." [HEL09]

In other words SSO can help users to eliminate the reuse of logon credentials for various accounts, thereby preventing that one maleficent site is able to use your logon credentials and use them to access your other accounts. The SSO concept has been around for a few decades, but it was usually a proprietary solution [VOL01]. It was first thought of as a way to reduce cost as it was removing the identity management from all separate applications and let it be handled by a dedicated application.

People already use several single sign-on accounts probably without realizing it. Notable examples of this are OpenID, Windows LiveID, Google account, Kerberos and in the Netherlands DigiD. OpenID will be described in the next chapter (chapter 3). The other examples will be described in the Single Sign On comparison chapter (chapter 4).

3 OpenID

In this chapter OpenID will be described in some detail. First we will start with an introduction to OpenID. In the first paragraph the OpenID protocol will be described. In the second paragraph we will describe some of the problems of OpenID. In the third paragraph we will briefly describe some OpenID implementations. The fourth and last paragraph looks at the way the OpenID protocol has been described in other research.

OpenID is an open, decentralized, free framework for User-centric digital identity management [OPEN10a]. OpenID is open because it is an open standard. It is decentralized because there are numerous Identity providers (IdP). It is free because anyone can implement or use it free of charge. OpenID is a software framework, which means that it allows for many flavors of implementations. A framework has a core which all applications that are based on it need to adhere to but the rest of the code can be altered and extended. User-centric digital identity management means that it manages the identity data of a user [OPEN10a]. OpenID is based on the SSO mechanism. SSO is a mechanism which enables a user to authenticate and authorize to all participating systems where the user has access, without the need of having more than one identity [OH08]. OpenID is being used by companies such as Google, Yahoo, AOL, MySpace, Verisign, etc. Anyone who has an account on any of the sites of these companies has an OpenID Identity.

OpenID uses an OpenID identifier to identify a user (for example: <http://name.example.com> or <http://example.com/name>). There are some terms in OpenID that need to be explained to understand the working of OpenID. The Relaying party (RP) is a website that requires users to provide an OpenID identifier and uses OpenID as a method to authenticate users. Another important term of OpenID is OpenID provider (OP) also called Identity provider (IdP). The IdP is the authentication server within the OpenID protocol. An RP will contact the IdP to check whether the identifier that the user provides is really his identifier. A user can choose on which IdP he wants to register an account and will be able to use the corresponding identifier on websites which allow OpenID logon.

3.1 Protocol

The architecture of the OpenID protocol is described in Figure 2 and 3. The description in this thesis is based mainly on the OpenID specification [OPEN10c] as well as [HEL09], [LEE08], [LIN09], [OH08] and [TSY07]. The way OpenID has been described differs greatly in these papers. The last paragraph in this chapter has been dedicated to this variation in modelling.

OpenID is an open standard; parties have choices in how to set up their IdP or RP. Implementations of IdPs and RPs therefore vary. Our description includes most of these implementations.

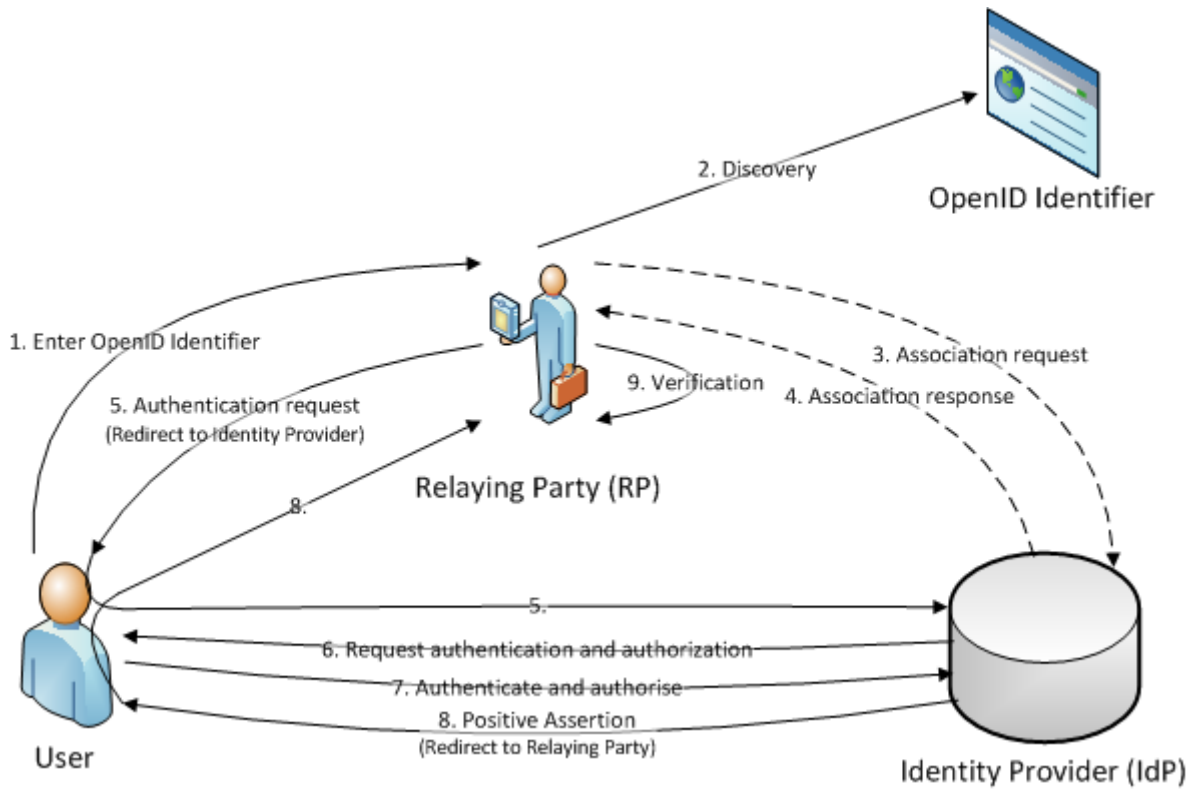


Figure 2: OpenID architecture

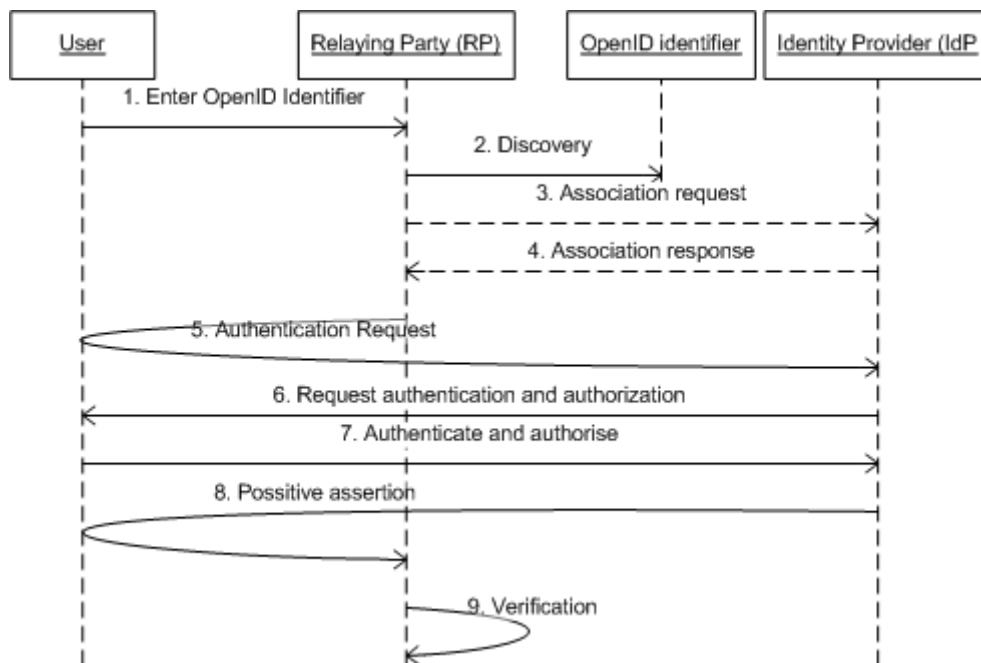


Figure 3: OpenID UML Sequence Diagram

The common variables that are sent in messages of the OpenID protocol are described in table 1. You will also be able to see in which steps these variables can be found.



Variable	Description	Commonly found in steps								
		1	2	3	4	5	6	7	8	9
ns	This value determines the version of the protocol the sender is using. In the 2.0 version of OpenID this value is always ' http://specs.openid.net/auth/2.0 '			X	X	X				
mode	This value is used to determine the current step. If mode is associate, the step is association request, if it's checkid_setup, the current step is authentication request, etc.			X		X			X	
assoc_type	The value of this field defines the key algorithm to be used to sign subsequent messages [OPEN10c].			X	X					
assoc_handle	This value is used to refer to an association.				X	X			X	
session_type	In the association request message this value is the preferred association type. In the association response message it should be equal to the value the IdP received.			X	X					
mac_key	The MAC key/ shared secret for an association.				X					
realm	In realm there needs to be a URL (of the RP) that the IdP asks the user to trust, default value is the return_to value [OPEN10c].					X				
claimed_id	OpenID identifier claimed by the user					X			X	
identity	OpenID identifier or identity					X			X	
expires_in	The lifetime of an association in seconds				X					
return_to	The value needs to be a URL to which the IdP can return the User with the response indicating the status of the request. The return to URL is generated by the RP. Parameters of this URL are not defined within the OpenID specification.						X		X	
signed	Comma-separated list of the signed fields [OPEN10C]								X	
sig	Signature of the values of the signed fields. Sig is equal to {signed.values} _{mac_key} . The session key has been used to encrypt the values of the fields specified in signed.									X
op_endpoint	Identity Provider endpoint URL, the website address of the IdP								X	
response_nonce	A string of 255 or less characters that is unique to a certain positive assertion								X	
invalidate_handle	If an RP sent an invalid association handle it should be included in this value.								X	

Table 1: Common variables in the OpenID protocol

In the next couple of subsections (2.1.1 - 2.1.9) we describe and discuss the various steps of figure 2 and 3. The numbers as well as the titles of the following subsections match the steps as seen in Figure 2 and 3.

In our reference example the IdP onno.uni.cc uses the SimpleID package which will be discussed further in section 6.3.1.2. The RP we used to logon to in our running example is livejournal.com.

3.1.1 Enter OpenID identifier

This subsection corresponds to step 1 of figure 2 and 3.

User -> Relaying Party: OpenID identifier

The user launches a web browser and types in a URL of a Relaying Party (RP) he wants to visit. When the user has loaded the website, he will have the choice to login or register by presenting an OpenID identifier to the RP. The first step in this protocol is input from the user. The user enters his OpenID identifier in a designated field of the website of a Relaying Party. This identifier can be a Uniform Resource Identifier (URI), which could be a name (URN), a locator (URL) or both. By both we mean that a URI can be both a URN and a URL as there is a slight overlap between the two.

Another identifier that can be used is an Extensible Resource Identifier (XRI). “The goal of XRI is a standard syntax and discovery format for abstract, structured identifiers that are domain-, location-, application-, and transport-independent, so they can be shared across any number of domains, directories, and interaction protocols.” [WIKI10] The identifier that is mostly used is undoubtedly the URL because it is used in most standards.

Websites that offer users the ability to sign in with an OpenID usually have a field similar to the one in Figure 4. Some RP have implemented the Janrain Engage/ RPX software. This package offers more than just OpenID logon. It also provides users with the ability to logon with a Facebook, Twitter, Windows Live or LinkedIn account [ELLI10]. These accounts can not be accessed with the OpenID protocol. The Janrain Engage package has included proprietary protocols of the companies behind Facebook, Twitter, Windows Live and LinkedIn. RP's that have implemented the Janrain Engage package have a login form similar to the one in Figure 5. A user needs to click the button of an account he wishes to logon with. An RP is able to add or remove buttons from the login form. The OpenID button can be removed thereby disabling the ability of the user to enter an identifier. All IdP implementations have identifiers but some IdPs have the same identifier for all users. For example all Google users have the identifier (<https://www.google.com/accounts/o8/id>). In our reference example we will logon to the RP <http://www.livejournal.com> with the identifier <http://onno.uni.cc> on the IdP <http://onno.uni.cc/id>.

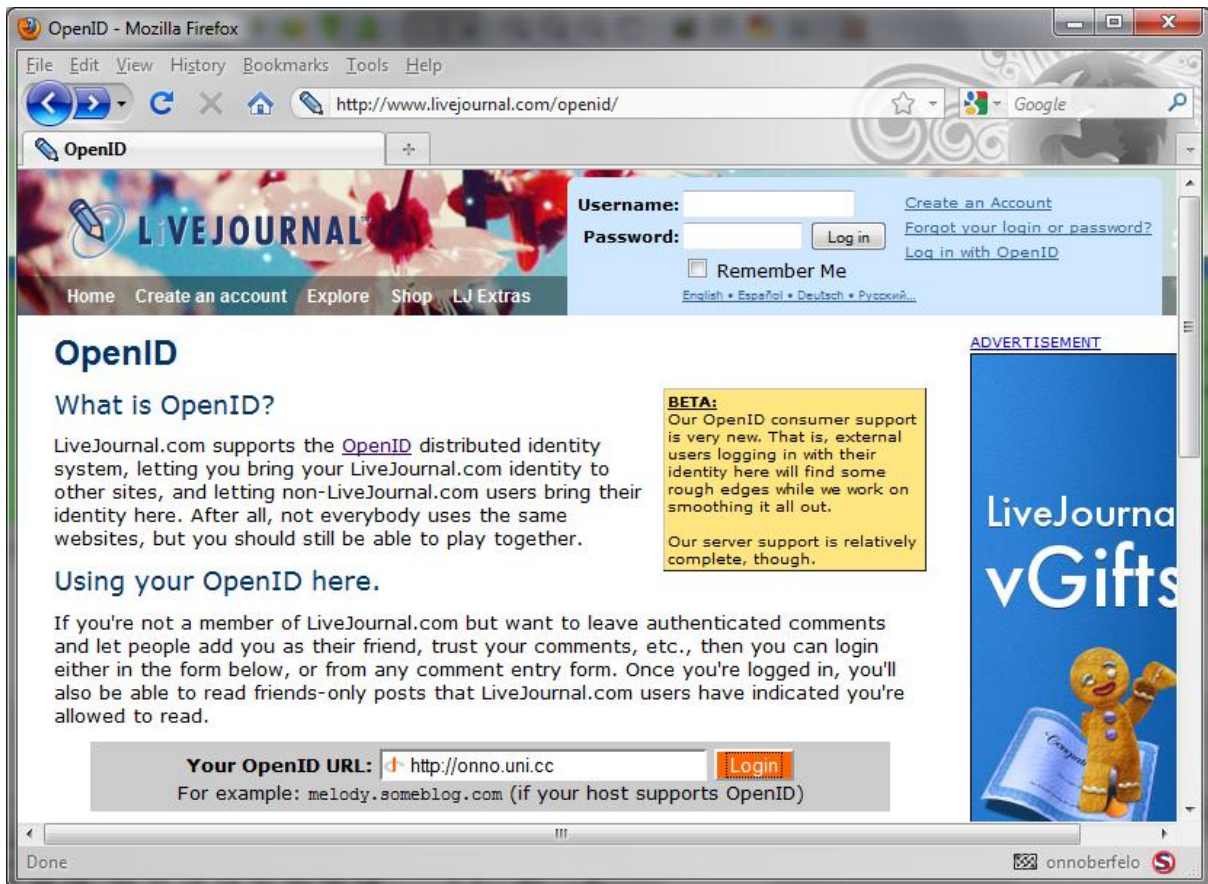


Figure 4: OpenID sign in on livejournal.com



Figure 5: OpenID sign in Janrain Engage



3.1.2 Discovery

This subsection corresponds to step 2 of figure 2 and 3.
Relaying Party -> Identity provider

The RP will normalize the User-Supplied Identifier (this is the OpenID identifier). Normalization means in this context converting the input of the user to a normalized identifier. If a user for example supplies an XRI identifier, the 'xri:/' at the beginning will be stripped off so it has the simplest form. The following table lists all normalization rules (Table 2).

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

Table 2: Example user input to identifier normalization [OPEN10c]

If the RP wants to limit the use of certain IdP then this could be checked during Normalization. The RP could for example check the identifier to see if the IdP is one that it trusts.

After normalization of the identifier the RP performs a discovery process on the normalized identifier to establish the IdP endpoint URL that the user uses for authentication. The IdP endpoint URL is a URL of the IdP that accepts the OpenID authentication protocol messages. The discovery process can be executed in various ways. HTML discovery is the last attempted and most common method used by RP. When the identifier is an XRI, its resolution will result in an eXtensible Resource Descriptor Sequence (XRDS) document. When the identifier is a URL the Yadis protocol [YADI06] will be attempted and if successful, this too will result in an XRDS document. If the Yadis protocol has failed to retrieve an XRDS document HTML based discovery will be attempted to retrieve the endpoint URL and the OpenID protocol version. The protocol version is important due to the differences in the OpenID protocols versions 1.0, 1.1 and 2.0. Some RP will allow an IdP that makes use of the 2.0 version of the OpenID protocol.

In our running example the IdP server registers a GET request of the root directory from the RP. The IdP server then responds with the content of the autoload file in that directory, in this case 'index.html'. This file contains the information with regard to the directory of the IdP on the server:

```
<html>
<head>
<link rel="openid2.provider" href="http://onno.uni.cc/id/" />
<link rel="openid2.local_id" href="user" />
</head>
<body></body>
</html>
```

The endpoint URL (<http://onno.uni.cc/id/>), the identity (user) as well as the version of the protocol (openid2) has now been discovered by the RP. Older versions of the protocol use different attributes on their identity page.

3.1.3 Association request

This subsection corresponds to step 3 of figure 2 and 3.

Relaying Party -> Identity Provider: ns, mode, assoc_type, session_type

The association steps are optional for the execution of the OpenID protocol [LIN09]. This is mainly because these steps are not necessary when the communication takes place with the use of the Secure Socket Layer (SSL) or rather HyperText Transfer Protocol Secure (HTTPS). Nonetheless the specifications recommend that a shared secret is established between the RP and IdP; if this step is not executed the protocol requires the execution of a different set of verification steps. The association can be created with either no additional encryption (in case of HTTPS), DH-SHA1 or DH-SHA256. The RP commonly sends the extension alias ns (value: "<http://specs.openid.net/auth/2.0>"), OpenID mode (value: associate) and OpenID session type. If the ns value is missing or set to a different value the protocol should be dealt with using the 1.1 compatibility specification. If the association steps are skipped the ns parameter will be sent with the Authentication Request. When the Diffie Hellman protocol (DH) is used the RP also sends an OpenID association type (value: HMAC-SHA1/ HMAC-SHA256).

In our running example the association steps were skipped. The RP did not initiate association. But when it does take place, you will be able to see a POST message on the IdP server from the RP requesting the endpoint that was obtained during the discovery step. The following POST message is a different example of an association request which uses an older version of the OpenID specification. Note that the ns variable is missing due to it being a new parameter in the 2.0 specification of OpenID.

```
POST http://example.com/op/index.php
openid.assoc_type=HMAC-SHA1&openid.mode=associate
```

- `assoc_type`: The value of this field defines the algorithm to be used to sign subsequent messages [OPEN10c].
- `mode`: The value for this field needs to be associate [OPEN10c].

3.1.4 Association response

This subsection corresponds to step 4 of figure 2 and 3.

Identity Provider -> Relaying Party: ns, assoc_type, assoc_handle, session_type, expires_in, mac_key

An Association response is a direct response to the Association request from the IdP to the RP. The IdP responds with a lifetime of the association after which the association is no longer valid and has to be renewed. This means that the association steps will have to be executed again the next time the user logs onto this RP while using the same identifier.

In our running example the RP did not initiate association; the IdP therefore did not need to send an association response. The following message is an example of an association response. server:

```
-----  
assoc_type:HMAC-SHA1\n  
expires_in:1440\n  
assoc_handle:6rn55rlh449dm6dv77ngkoa6d3\n  
mac_key:L//32JdjUJaZ+6gok1tRjhw7+OA=\n  
-----
```

- **assoc_type:** This is the value of the assoc_type of the association request message. Or if the IdP is unwilling or unable to support this assoc_type, then the value of the message will be an unsuccessful response [OPEN10c].
- **expires_in:** The lifetime, in seconds, of this association. The RP must not use the association response after the lifetime has passed [OPEN10c]. HTTP messages themselves also contain a timestamp which is what is used by the RP to compare it with the lifetime.
- **assoc_handle:** The value of this field is used as a key to refer to this association in subsequent messages [OPEN10c].
- **mac_key:** The value is the MAC key (shared secret) for this association encoded (BASE 64) [OPEN10c].

3.1.5 Authentication request

This subsection corresponds to step 5 of figure 2 and 3.

Relaying Party -> User -> Identity Provider: ns, mode, return_to (optional), claimed_id (optional), identity (optional), realm (optional), assoc_handle (optional)

The RP will redirect the user to the IdP with an OpenID authentication request. The return to value in our running example contained a number 1273495345 and a ciphertext 82ea675f7d08da6386b7. The number represents Unix Time or the time passed in seconds since 01/01/1970. In this case this value is equal to the date Mon, 10 May 2010 12:42:25 GMT. The ciphertext is most likely a keyed hash of the Unix Time. These parameters are not specified in the OpenID specification. The parameters in our example are unique to the implementation of the RP. The HTTP redirect from our running example looks like this:

```
GET onno.uni.cc/id/?openid.ns=http://specs.openid.net/auth/2.0&
openid.return_to=http://www.livejournal.com/openid/login.bml%3Foic.time%3D1
273495345-82ea675f7d08da6386b7&
openid.claimed_id=http://onno.uni.cc/&
openid.identity=user&
openid.mode=checkid_setup&
openid.realm=http://www.livejournal.com/&
openid.assoc_handle=4be6aa61000646801eaffe45
```

- ns: The ns value determines the OpenID version that the RP uses. The value of this parameter needs to be 'http://specs.openid.net/auth/2.0'. If the ns is missing or has another value the IdP will treat the request in the OpenID 1.1 compatibility mode.
- return_to: The value needs to be a URL which the IdP can return the User to with the response indicating the status of the request. The return to URL is generated by the RP. Parameters of this URL are not defined within the OpenID specification.
- claimed_id: OpenID identifier claimed by the user
- identity: is the OpenID identity of the user (or claimed identifier).
- mode: The value checkid_setup in mode enables the end-user to interact with the IdP
- realm: In realm there needs to be a URL (of the RP) that the IdP asks the user to trust, default value is the return to value [OPEN10c].
- assoc_handle:(optional) is used to establish the way responses need to be signed.

3.1.6 Request authentication and authorization

This subsection corresponds to step 6 of figure 2 and 3.

Identity Provider -> User

The IdP checks whether the user is authorized to perform OpenID authentication; the user needs to enter his password (or other required information). And the user is asked if he wants to proceed with the OpenID authentication and which data the user wants to send to the RP. This is not always the case. Some implementations of IdPs don't support that. This means that the user is not always asked which data is sent to the RP.

The way in which a user has to authenticate himself to the IdP is not specified in the OpenID protocol and it therefore differs in many of the interpretations. The IdP gives the user the opportunity to authenticate himself and authorize the RP. The IdP can use any sort of authentication mechanism.

3.1.7 Authenticate and authorize

This subsection corresponds to step 7 of figure 2 and 3.
User -> Identity Provider

The user sends his login data and other user data the user chooses to send. The user is able to choose which information (data) to send if the RP requests it. OpenID itself doesn't specify any user information other than the identity. The user information is added to OpenID with an extension like SREG. SREG stands for Simple Registration. It is an extension to the OpenID protocol. It is made to exchange some profile information with an RP when a user registers a new account. In Figure 6 you can see the authentication of a user in our running example. Note that the user has been redirected to the website of IdP (Figure 7). In this case you logon first (Figure 6) after that you get a request for authorizing the IdP to redirect you back to the RP (Figure 7). In our running example the RP did not request any user information.

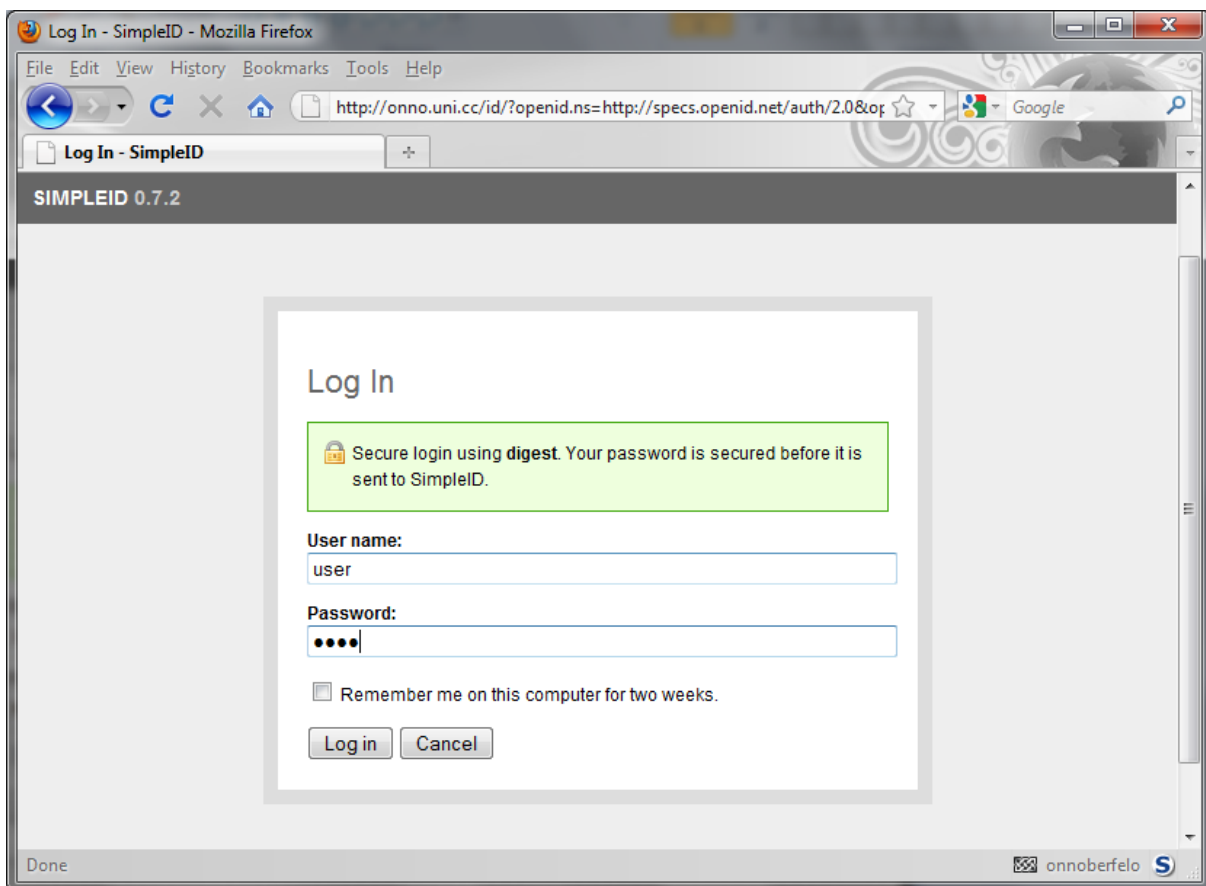


Figure 6: OpenID Authentication

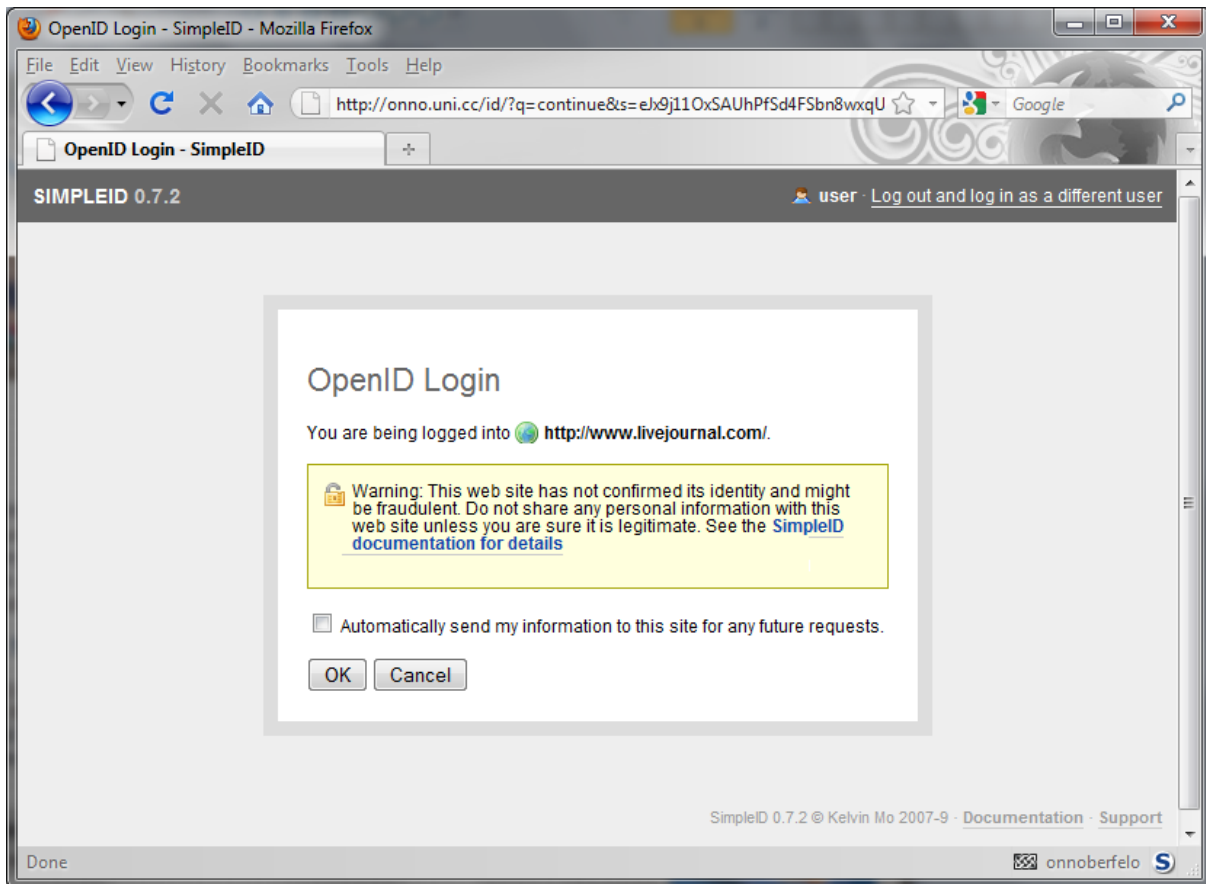


Figure 7: OpenID Authorization

3.1.8 Positive assertion

This subsection corresponds to step 8 of figure 2 and 3.

Identity Provider -> User -> Relaying Party: mode, op_endpoint, response_nonce, assoc_handle, return_to, signed, sig = {signed.values}_{mac_key}, identity (optional), claimed_id (optional), invalidate_handle (optional)

When the user has authenticated himself and authorized the IdP to pass on data to the RP, the IdP redirects the user back to the RP with a positive assertion. This positive assertion is a message in which the IdP states that it has authenticated the user. The positive assertion in our running example looks like this:

```
"Location: http://www.livejournal.com/openid/login.bml?
oic.time=1273495345-82ea675f7d08da6386b7&
openid.mode=id_res&
openid.op_endpoint=http%3A%2F%2Fonno.uni.cc%2Fid%2F&
openid.response_nonce=2010-05-09T12%3A54%3A53Zcba9f18e&
openid.assoc_handle=4be6aa61000646801eaffe45&
openid.identity=user&
openid.return_to=http%3A%2F%2Fwww.livejournal.com%2Fopenid%2Flogin.bml%3Foi
c.time%3D1273495345-82ea675f7d08da6386b7&
openid.claimed_id=http%3A%2F%2Fonno.uni.cc%2F&openid.ns=http%3A%2F%2Fspecs.
openid.net%2Fauth%2F2.0&
openid.signed=op_endpoint%2Creturn_to%2Cresponse_nonce%2Cassoc_handle%2Cide
ntity%2Cclaimed_id&
openid.sig=GMABNRWg%2Bv2t92egBqgBSOzkuhI%3D"
```

- `oic.time`: RP Parameter resembling unix time and a ciphertext. This parameter is not specified in the OpenID standard.
- `mode`: The value of mode needs to be `id_res`, there is no extra information about why it should be `id_res` or what it means.
- `op_endpoint`: IdP endpoint URL.
- `response_nonce`: Is a string of 255 characters or less and has to be unique to this particular positive assertion. There are also some other obligations the nonce must start with the current time of the server and may contain ASCII characters [OPEN10c]. This nonce is used for the purpose that the RP doesn't accept the same positive assertion from that certain IdP. It will be verified by the RP during the verification step.
- `assoc_handle`: The value of this field is the algorithm that is used for signing the assertion.
- `identity`: OpenID identifier or identity
- `return_to`: Is a verbatim copy of the return to value that is sent in the request.
- `claimed_id`: OpenID identifier claimed by the user
- `signed`: Is used to determine which fields need to be covered by the signature (`openid.sig`), the fields need to be named without the "openid."
- `sig`: Is the calculated encoded (64) signature.

When the user has not authorized the RP, a negative assertion is sent as opposed to the positive one. This will terminate the execution of the protocol.

3.1.9 Verification

This subsection corresponds to step 9 of figure 2 and 3.
Relaying Party -> Identity Provider

The last step is that the RP verifies the information received from the IdP:

- Check if the returned value of "openid.return_to" matches the URL of the current assertion
- Check if the signature on the current assertion is valid and the fields that need to be signed, are signed
- Check if the discovered information matches the information of the current assertion
- Check if the assertion has not yet been accepted from this IdP with the same value for "openid.response_nonce"[OPEN10C]

If the association steps are not performed during the process, the verification of the signature needs to be done in a different way. The RP then sends a direct request to the IdP. To respond to this message the IdP uses the private association that was generated when it issued the positive assertion.

In our running example the RP did not initiate the association steps. Therefore it needed to send a direct request; it requested the identity page. The IdP replied by sending the identity page. When all the steps are done, the user has identified himself to the RP and the authentication protocol is finished. The user will then be sent to a page that states that the authentication was successful (Figure 8).

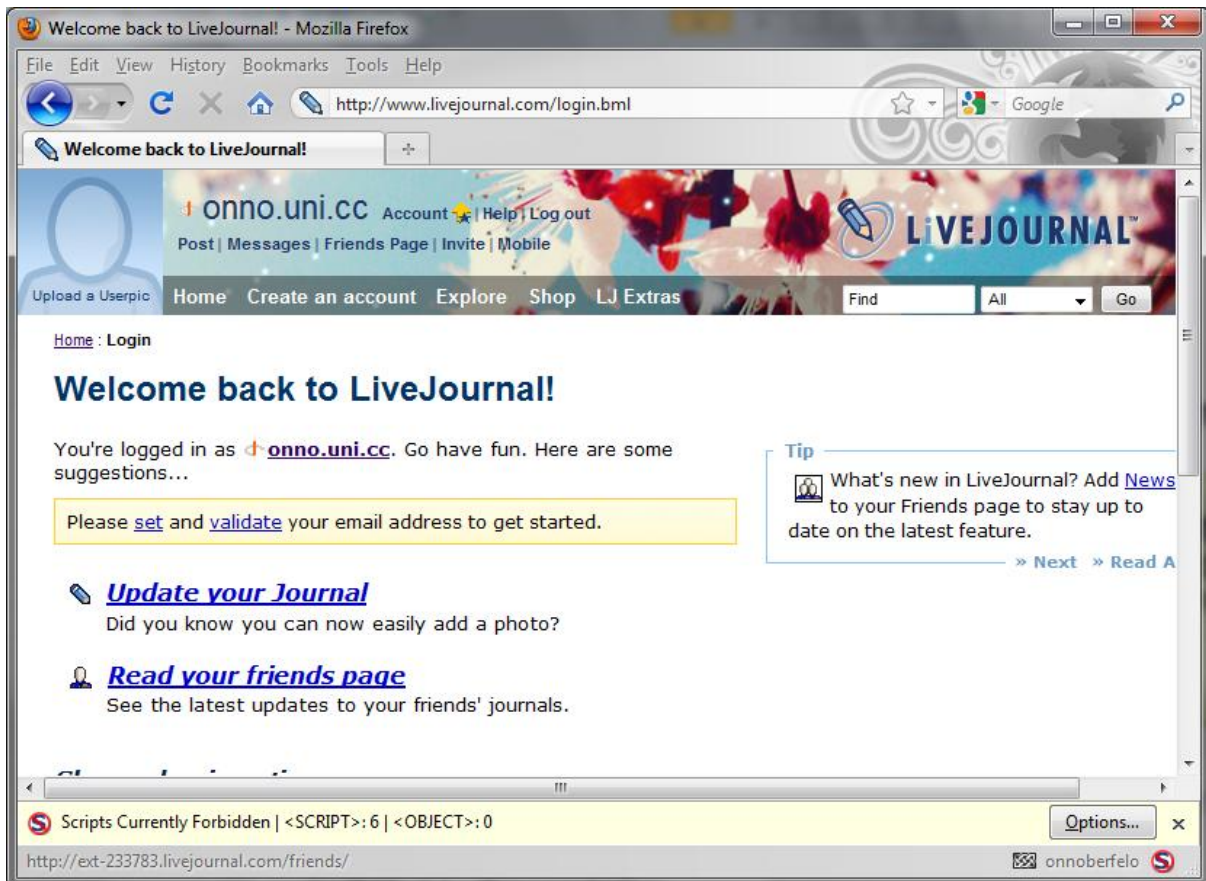


Figure 8: Logged on with OpenID

3.2 Problems

Ever since the release of the OpenID Authentication 2.0 in 2007 there have still been some security problems. In this paragraph some of these security problems in OpenID will be discussed in relation to the described OpenID protocol above. First the security goals of all the involved parties (user, RP and IdP) are described. Secondly the security problems are described. Thirdly some trust problems are described.

The security goals:

User:

- Login information needs to be kept confidential
- Login information must not be altered along the way
- Browsing history needs to be kept confidential
- IdP should redirect the user to the right RP
- RP should redirect the user to the right IdP
- IdP should be available when the user needs it

RP:

- The communication between the RP and the IdP needs to be confidential
- The communication between the RP and the IdP must not be altered along the way
- The communication between the RP and the user needs to be confidential
- The communication between the RP and the user must not be altered along the way
- The RP needs to protect itself against malicious identifiers.

IdP:

- The communication between the IdP and the RP needs to be confidential
- The communication between the IdP and the user needs to be confidential
- The communication between the IdP and the RP must not be altered along the way
- The communication between the IdP and the user must not be altered along the way

Now some problems (also linked to the steps of the protocol) are described which can occur when using OpenID:

- When the RP performs discovery on an identifier, the RP needs to extract the information to know which IdP it needs to establish a connection with. If an attacker were to enter a malicious identifier, this could have some consequences for the RP. A few example consequences are:
 - Denial of Service
If the URL is a link to a large file, the RP server could become unavailable.
 - Port Scan
If you add a port to the identifier (<http://www.example.nl:11>), the RP will Port Scan that host [TSY07]. An RP can prevent this by checking whether the identifier is a local address. Another measure needs to be reporting the same error message for each of the failures that can occur during the discovery step. Thus an attacker will not be able to tell if a port is open or closed.
- The association steps are used by the RP and IdP to negotiate a shared secret key using the Diffie-Hellman Key Exchange (DH). The problem with DH is that it is vulnerable to man-in-the-middle attack. A man-in-the-middle attack is an attack where the attacker is eavesdropping the connection between RP and the IdP. The attacker intercepts the messages between the RP and IdP and sends them himself. For the IdP he will now seem to be the RP and the RP will think the attacker is the IdP. A solution that OpenID gives is to use HTTPS instead of HTTP to avoid this problem but when you use HTTPS, DH isn't necessary anymore because you can exchange information securely over the SSL connection [TSY07].
- During the authentication request step the RP redirects the user to the IdP. The problem that can occur in this situation is that the RP is malicious and redirects the user to a fake IdP. This can be done because the RP can redirect a user to any website it wants. This form of attack is called phishing (fake websites which look like the original, and then steal the personal information of the user). With this attack the user can lose his identity and an attacker can use this data to cause damage. This is the best known attack against OpenID [TSY07].
- After the user is redirected to the IdP and has successfully been authenticated (step six and seven) the user can automatically login on other RPs. The problem with this is that an IdP (a malicious one) can track which websites the user visits. At the moment there isn't a good solution for this problem [TSY07].

- When a user is already logged in, a malicious website is able to grab the user information. This vulnerability is called clickjacking. Bart van Delft recently demonstrated this problem during the Digital Security Lunch Colloquium on June 11th 2010. With clickjacking a page steals your click by hiding an iframe element on another page. The page you see will request your OpenID identifier. Whilst you will log on, the page will say it failed. The retry button on the page is actually the authorize button on the IdP website. When you click the retry button the website will have retrieved the information from your OpenID.
- During the positive assertion step the user is redirected back to the RP. The problem with this is that if, for example, an attacker is eavesdropping and gets in possession of the URL, the attacker will be able to replay that URL. When the attacker replays the URL he will be logged in on the RP as the user. There is a solution for this and that is that IdP uses a nonce (a random number used only once). With a nonce implemented in the URL the attacker will not be able to replay the URL. If the attacker is the first one to use the URL, he will still be logged in on the RP [TSY07].

3.2.1 Trust problems

Another big issue of OpenID is trust. Everyone can create an OpenID identifier and use it to logon to websites (RP). In the context of OpenID the information sent by the IdP needs to be trusted by the RPs. When there is no trust between those parties then the information that is sent from the IdP to RP is of lesser value, only the identifier can be trusted. This leads to a certain problem and that is that large IdPs such as Google, Microsoft, Yahoo and others only act as IdPs and not as RPs, because they would not want to depend on another maybe less trustworthy IdP to provide information. Another problem that could occur when you choose an IdP which isn't reliable and goes down, is that you can't access your websites (RPs) where you used that OpenID identity [HEL09]. So the trust between IdPs and RPs isn't always there and the trust for the user, that the IdP is stable, isn't certain either.

3.3 Alternative OpenID solutions

There are some problems in the standard protocol of OpenID, as described above. To solve these problems organizations are trying to create an improved version of OpenID. They are adding extra security mechanisms and are adding conditions to parties to make OpenID more secure. In this paragraph some of these solutions have been described.

3.3.1 OpenID Plus

OpenID plus (since 2009) is an initiative of ECP-EPN where potential IdPs and RPs can work together to create a "trusted identity". One of the great issues of standard OpenID is trust. OpenID plus has as goal to create a trust layer on top of OpenID. They intend to have multiple levels of assurance within OpenID. The plus in OpenID plus stands for agreements between parties to preserve the interests of the customers (users) and to improve the quality of the OpenID based identities. OpenID plus works like this: The RPs which are in the network of OpenID plus must meet up to certain conditions. An example of a condition is that the RPs don't abuse the information that they get of the IdP. If they do, there will be consequences for them. All of these conditions are a contribution to user experience, so that users will be able to trust a RP. This strays from the mission of OpenID and if implemented will resemble Security Assertion Markup Language (SAML) rather than OpenID. OpenID is open and by putting up boundaries it would not be as open as it is now.

3.3.2 Personal Identity Portal

Personal Identity Portal (PIP) is a VeriSign Labs product. It is not just an OpenID solution but it does incorporate it. As an OpenID provider PIP allows users to logon with a username and password combination, a client SSL certificate and an Information Card. The Information Card logon is odd as in itself it is an alternative to OpenID. We have tried to generate a client SSL certificate on the website with both Internet Explorer 8 and Firefox 3.6.3 but it simply failed to work. With Internet Explorer we received a message that the browser wasn't supported. This message stated that all versions of Internet Explorer 6 and newer and all versions of Firefox 2 and newer were supported. In Firefox it took ages after which it crashed the browser. The method used to generate the key appears graphically to be the same as the one used by CAcert, the latter working in both of the browsers mentioned. While PIP adds certificate logon, the information in this certificate has not been validated. This in contrast to what we aim to achieve with our project.

PIP also contains a way for users to store logon credentials online, named One Click Sign-In. This has some synergy with browser synchronization. In most browser synchronization applications the data that is sent to the server is encrypted whilst in this scenario you may just be handing your authentication credentials over to VeriSign. This seems odd to us as you would not want your logon credentials for numerous sites in the hands of a third party.

3.3.3 MyOpenID

MyOpenID is an OpenID Identity provider that allows users to identify themselves with either a username and password combination, an SSL client certificate or an Information Card. We have not been able to generate an SSL Certificate on the site of MyOpenID. In Internet Explorer 8 the option to create an SSL client certificate is not available. In Firefox 3.6.3 it reports that an unknown error has occurred while generating a key. This leaves the Information Card as the alternative for logging on. While this may be more secure, this still leaves the need for the identity to be matched to an individual. The information in the certificate or information card has not been validated.

3.4 Protocol in other research

The architecture of the OpenID protocol has been described in various sources such as [HEL09], [LEE08], [LIN09], [OH08] and [TSY07]. The architectures in these scientific articles all have similarities but also slight differences. The model in [LIN09] is very close to the specifications of the OpenID 2.0 authentication protocol [OPEN10c] as mentioned on the OpenID website. Most other models are more abstract and use terminology which is less similar to the terminology used in the OpenID specifications. The models mostly start out the same when the user enters an OpenID URL as the first step. The second step in most models is the discovery of the OpenID URL, but for instance in [LEE08] this step has been merged with the association step(s). In some models e.g. [LIN09], two steps are used for the association process whilst others e.g. [TSY09] and [HEL09], use only one step for it. There is even a model in [OH08] where the association steps are missing, which could be due to these steps being optional according to [LIN09] and [OPEN10c].

The redirect steps can be found in most models, but they are named differently in [LIN09]. In [LIN09] they are named 'Authentication request' and 'Positive assertion' which is in line with the terminology used in the OpenID 2.0 specifications [OPEN10c]. For clarity we have



chosen to split the redirect step into two connected parts due to the fact that in a redirect three parties are involved namely the forwarder, user and the target. This is not much unlike the way it is modeled in [TSY07] or [LIN09].

The next few steps are about authenticating the User to the Identity Provider, but these steps are not specified by the OpenID standard. This is mentioned in [LIN09] and obviously due to it not mentioned in the OpenID specifications [OPEN10c]. Most literature assumes that a username-password system is in place. It should not be surprising that the authentication steps in the models vary a lot. In [TSY07] the only step mentioned is 'Enter Password' followed by 'Authorize RP'. The model in [LIN09] has three steps: the first is an authentication request that is actually a redirect to the IdP, the second one is the actual authentication step whilst the last one is the process of sending a positive assertion to the relaying party after the successful authentication. The last step in [LIN09] is actually a redirect as well so you could say that the authentication consists of just one step. The protocol described in [OH08] has two authentication steps; they are divided into a request and response between the Identity Provider and the User. In [HEL09] only one step has been mapped named 'Authenticate'.

When the authentication of the User to the Identity Provider is concluded the User will be redirected back to the Relaying Party. This has been defined differently in the models mentioned but it is apparent in all of them. Then the Relaying Party may proceed with verification, during which it verifies the assertion from the User. The papers [HEL09], [LEE08], [LIN09], [OH08] and [TSY07] all described the OpenID protocol accurately but from a different level of abstraction or point of view. Some simplified the protocol others merged some steps. With our interpretation we chose to be as close to the specification as possible. This meant that the descriptions in the papers were quite brief while ours is more extensive.

4 Single Sign On comparison

In this chapter we compare the OpenID protocol with other SSO mechanisms, such as Kerberos, DigiD, Microsoft Windows LiveID and Google Federated Login. In each paragraph we will first introduce the protocol. After the introduction a description of the protocol follows. The last section that is described is a comparison with the OpenID protocol.

4.1 Kerberos

The Kerberos protocol is based on the Needham and Schroeder protocol. The Needham and Schroeder protocol dates back to the end of the seventies. What has changed the least within Kerberos compared to Needham and Schroeder is password control. In either protocol both the client and the server share a secret password. Another similarity is the exchange of one-time identifiers between clients and servers [ZHA03]. The identifier used in the Kerberos protocol (timestamp) is stronger than the one used in Needham and Schroeder (nonce) [ZHA03]. These one-time identifiers are used to prevent replay attacks. The timestamp used in Kerberos is generalized time which is based on the UTC timezone [KOHL93]. KerberosTime does not keep track of fractional seconds [NEUM05]. KerberosTime is notated as YYYYMMDDHHNNSSST. The NN refers to minutes and T refers to timezone. July 12th, 2010, 12:10:22.41 (UTC) is notated as 20100712121022Z.

Kerberos is named after a Greek mythological three headed dog. These heads resemble the Client, Service and the Key Distribution Center (KDC). The KDC consists of the Authentication Server (AS) and the Ticket Granting Server (TGS). The KDC is responsible for maintaining a database of secret keys. The KDC also generates keys for sessions between parties. Since both the TGS and the AS require the same information such as the keys, they usually run on the same system.

Many Unix and Linux distributions as well as Windows 2000 and later use Kerberos as the default authentication method to identify the identity of the user or host.

4.1.1 Protocol

The Kerberos protocol can be divided into two parts: one part in which the identity of the user is determined and one part in which a session is established. We will discuss both parts of the protocol in the following two subsections. The abbreviations we use in our depictions of the Kerberos protocol are described in table 3.

Abbreviation	Description
U	User / username
C	Client
S	Server
AS	Authentication Server
TGS	Ticket Granting Server
P_X	Password of X
$h[X]$	Hash of X
K_X	Secret key of X
$K_{X,Y}$	Session key for X and Y
$\{Z\}_{K_X}$	Data Z encrypted with the secret key of X
$\{Z\}_{K_{X,Y}}$	Data Z encrypted with the session key of X and Y
A_X	Authenticator of X
$T_{X,Y}$	Ticket of X to use Y

Table 3: Kerberos abbreviations

Kerberos specifies two credentials: a ticket and an authenticator. Both credentials are timestamped. A ticket is a record that can be used to authenticate yourself to a service. All tickets have an encrypted part and an unencrypted part. The unencrypted part of a ticket contains the Version number, Realm and PrincipalName. The encrypted part contains TicketFlags, EncryptionKey, Realm, PrincipalName, TransitedEncoding, Authtime (KerberosTime), Starttime(KerberosTime) (optional), Endtime(KerberosTime), Renew-till(KerberosTime) (optional), HostAddresses (optional) and AuthorizationData (optional). KerberosTime is Generalized time based on the UTC timezone (section 4.1.1). The attributes of a ticket are described in table 4.



Variable	Unencrypted	Encrypted	Optional	Description
Version number	X			Version number of the protocol.
Realm	X	X		Realm of the ticket/ authenticator issuer. Realm is equal to the uppercase domain of the issuer.
PrincipalName	X	X		The unique name of a user or service
TicketFlags		X		TicketFlags are a set of flags that indicate various attributes of the ticket. An example is the 'initial' flag which, if true, means that the ticket is an initial ticket granting ticket (ticket sent in step 2 of figure 11).
EncryptionKey		X		Keytype and keyvalue
TransitedEncoding		X		List of Realms that took part in authenticating the user to whom this ticket was issued
Authtime		X		Authtime specifies when the original authentication took place
Starttime		X	X	Start time of the ticket
Endtime		X		End time of the ticket
Renew-till		X	X	Renew-till time that specifies the latest time the ticket can be renewed. A ticket can only be renewed when the renew flag is set to true.
HostAddresses		X	X	HostAddresses is array that contains zero or more host addresses. These are addressees from which the ticket can be used. If the array is empty the ticket can be used from any address.
AuthorizationData		X	X	"The authorization-data field is used to pass authorization data from the principal on whose behalf a ticket was issued to the application service. If no authorization data is included, this field will be left out." [NEUM05]

Table 4: Kerberos ticket attributes

"An authenticator is a record sent with a ticket to a server to certify the client's knowledge of the encryption key in the ticket, to help the server detect replays, and to reach an agreement on a specific session key used with the particular session." [ZHA03]. An authenticator is fully encrypted. An authenticator contains Version number, Realm, PrincipalName, Checksum (optional), Microseconds, CurrentTime(KerberosTime), EncryptionKey (optional), SequenceNumber (optional) and AuthorizationData (optional). The attributes of an authenticator are described in table 5.

Variable	Optional	Description
Version number		Version number of the protocol.
Realm		Realm of the ticket/ authenticator issuer. Realm is equal to the uppercase domain of the issuer.
PrincipalName		The unique name of a user or service
Checksum	X	Checksum of application data that accompanies the authenticator
Microseconds		Microsecond addition to the CurrentTime attribute
CurrentTime		The current time in the KerberosTime standard (section 4.1.1)
EncryptionKey	X	Encryption key chosen by the client to be used for the application session. If the field is empty the session key from certificate will be used instead.
SequenceNumber	X	Sequence number that can be used to detect replays.
AuthorizationData	X	"The authorization-data field is used to pass authorization data from the principal on whose behalf a ticket was issued to the application service. If no authorization data is included, this field will be left out." [NEUM05]

Table 5: Kerberos authenticator attributes

4.1.1.1 Determining the identity

In Figure 9 a UML Sequence diagram of a Kerberos process has been depicted. In this case it is the process that determines the identity of a user. In Table 4 you can see the messages that are sent in the Kerberos protocol. The numbers in the description refer to the ones in figure 9 and table 6.

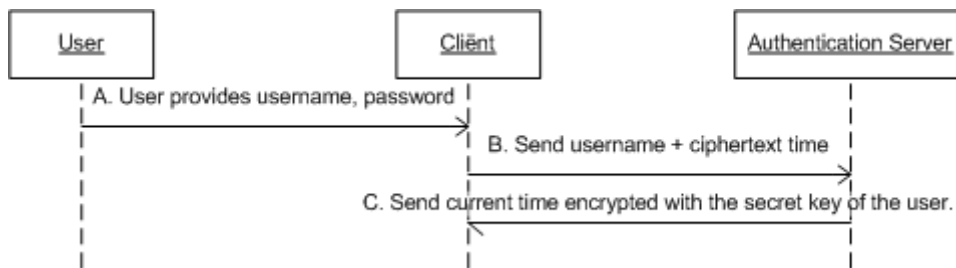


Figure 9: Kerberos UML Sequence diagram - Determining identity

Step	Sender	Receiver	Message
A	User	Client	U, P _U
B	Client	Authentication Server	U, {Current Time}K _U
C	Authentication Server	Client	{Current Time}K _U

Table 6: Overview Kerberos Messages - Determining identity

* The hash of the password of a user is equal to the secret key of the user. $h[P_U] = K_U$



Before a user can gain access to services, he must be authenticated. In the Kerberos protocol this is done with the use of a username and password combination. The hash of the password is the secret key of the user. The user provides his username and password (A). The client hashes the password; this hash is the secret key of the user. The client encrypts the current time with the secret key of the user. The client sends his username together with the ciphertext of the current time to the Authentication Server (AS) (B). The AS retrieves the secret key of the user from the KDC database. The AS decrypts the ciphertext that was sent by the client with the secret key of the user. If the plaintext matches the current time or is within a certain margin of it, the AS can assume that the password is correct. If the password is correct the AS will encrypt the current time with the secret key of the user and sends it to the Client (C). The Client decrypts the received ciphertext if it matches the current time to a certain margin. If it matches the Client and the AS will now have authenticated each other.

4.1.1.2 Session Setup

Setting up a session is something the user will do after he has authenticated himself to the AS. In Figure 10 the main UML Sequence diagram of the Kerberos protocol has been depicted. The names of these messages have been interpreted from the protocol. The names aren't used in the specification itself. The specification uses 8 character codes for the messages. Some literature only describes 3 party's [LIN09]. In that case they merge the AS and the TGS. In Table 5 you can see the messages that are sent in the Kerberos protocol. Note that timestamp is also part of a ticket and an authenticator. The numbers in the description refer to the ones in figure 10 and table 5.

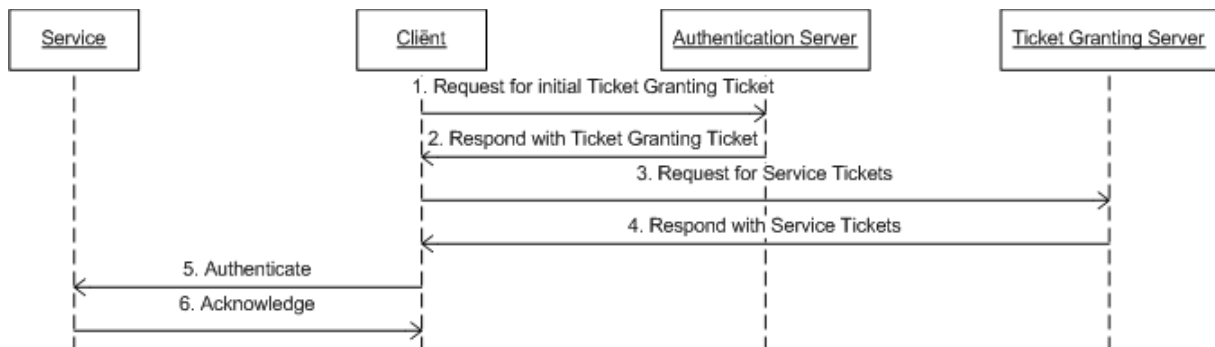


Figure 10: Kerberos UML Sequence diagram - Session Setup

Step	Sender	Receiver	Message
1	Client	Authentication Server	U, TGS
2	Authentication Server	Client	$\{K_{U, TGS}, \{T_{U, TGS}\}K_{TGS}\}K_U$
3	Client	Ticket Granting Server	S, $\{T_{U, TGS}\}K_{TGS}, \{A_U\}K_{U, TGS}$
4	Ticket Granting Server	Client	$\{K_{U, S}, \{T_{U, S}\}K_S\}K_{U, TGS}$
5	Client	Service	$\{T_{U, S}\}K_S, \{A_U\}K_{U, S}$
6	Service	Client	$\{\text{timestamp}+1\}K_{U, S}$

Table 5: Overview Kerberos Messages - Session Setup

The client sends the username of the user and the name of the TGS to the AS(1). The AS responds by sending an initial ticket granting ticket to the user (2). The initial ticket granting ticket is encrypted with the secret key of the User (K_U). This initial ticket granting ticket consists of a session key of the User and the TGS ($K_{U, TGS}$) and ticket of the User to use the

TGS ($T_{U, TGS}$) encrypted with the secret key of the TGS (K_{TGS}). The User continues by sending a request for service tickets (3). The user sends the address of the server, his authenticator encrypted with the session key and the encrypted ticket he received from the AS. The TGS responds with the service ticket which is encrypted with the session key of the User and the TGS (4). This ciphertext consists of the ticket of the User to use the Service ($T_{U, s}$) encrypted with the secret key of the Service (K_S) and the session key of the User and the Server ($K_{U, s}$). The User will now authenticate himself to the Server (5). He does this by sending the ticket to use the Server encrypted with the secret key of the Server $\{T_{U, s}\}K_S$ and his authenticator encrypted with the session key $\{A_U\}K_{U, s}$. The server will acknowledge this authentication by sending the timestamp from the authenticator incremented with 1 encrypted with the session key.

4.1.2 Comparison with OpenID

While comparing OpenID with Kerberos we wondered what they had in common. Apart from the fact that both are Identity Management systems, Single Sign On protocols and use Diffie-Hellman to generate session keys they are distinctly different. Kerberos has a high assurance level (section 2.1) while the assurance level of OpenID is low. Kerberos is a computer network authentication protocol that authenticates users and clients while OpenID is an Open Framework for User-Centric Identity Management that authenticates entities. In Kerberos all parties involved are authenticated, OpenID only authenticates the user. Kerberos is pretty much ironclad, its current version Kerberos V5 has existed since 1993 ([RFC 1510](#)). It was last altered in 2005 ([RFC 4120](#)). OpenID on the other hand is a much newer protocol and the current 2.0 version has been in existence since late 2007. Kerberos has no security problems worth mentioning and OpenID has many.

In the Kerberos protocol you could compare the service with an RP in OpenID. Both the service and the RP require the user to be authenticated. You could also compare the IdP in OpenID with the KDC (AS + TGS) of Kerberos. Both the IdP and the KDC enable the user to be authenticated towards the RP/ service. Kerberos has 4 parties (Server, User, Authentication Server, Ticket Granting Server), OpenID has 3 parties (RP, User, IdP). You could argue that Kerberos has 3 as well since both the AS as the TGS are part of the KDC.

You could say that step 2 (Respond with Ticket Granting Ticket) and step 3 (Request for Service Tickets) combined are somewhat similar to the Authentication Request step (section 3.1.1.5) in the OpenID protocol. In the OpenID protocol the RP redirects the user to the IdP with information with regard to what information is requested, RP address, user identity, etc. In Kerberos during the second step the AS sends the user the session key for the user and TGS and ticket of the user to be used with TGS encrypted with the secret key of TGS. During the third step the user forwards the encrypted ticket to the TGS and includes his authenticator.

In the same manner you could compare steps 4 (Respond with Service Tickets) and 5 (Authenticate) with the Positive Assertion step (section 3.1.1.8) in OpenID. These steps have in common that after these steps the user has been authenticated to the service/ RP. During the positive assertion step the IdP forwards the user to the RP, thereby acknowledging that the user is who he claims to be. During step 4 the TGS sends the Client the necessary information (session key and ticket encrypted with the secret key of the service) for the user to authenticate himself to the service. During step 5 the user uses this information to authenticate himself.

From a low level of abstraction it is tough to look beyond the differences but from a higher level of abstraction it is far more similar than it appeared at first sight. There is one thing that remains very different in Kerberos from what is specified in OpenID, which is Authentication of the User. In OpenID this is done about halfway during the process steps Request authentication and authorization (section 3.1.1.6) and the Authenticate and authorize steps (section 3.1.1.6). The manner in which these steps need to be executed is not specified. In the Kerberos protocol authentication of the user towards the authentication server takes place during steps A, B and C (section 4.1.1.1). These steps are specified and need to be executed prior to any of the other steps (4.1.1.2).

4.1.2.1 Similarities

- Both are Identity Management systems;
- Both are SSO protocols;
- Both Kerberos and OpenID use Diffie-Hellman to generate session keys;
- The Service in Kerberos has a role similar to the RP in OpenID;
- The KDC in Kerberos has a role similar to the IdP in OpenID;
- Step 2 (Respond with Ticket Granting Ticket) and step 3 (Request for Service Tickets) combined from the Kerberos protocol are similar to the Authentication Request step (section 3.1.1.5) from the OpenID protocol;
- Steps 4 (Respond with Service Tickets) and 5 (Authenticate) combined from the Kerberos protocol are similar to the Positive Assertion step (section 3.1.1.8) in OpenID.

4.1.2.2 Differences

- Kerberos has a high assurance level, OpenID has a low assurance level (section 2.1);
- Kerberos is a proven technology, OpenID on the other hand is a new protocol and has yet to prove itself;
- Kerberos has no significant security problems, OpenID has several security problems;
- Kerberos does not allow many choices in the implementation of the protocol, OpenID does allow many implementations;
- Kerberos has 4 parties (Server, User, Authentication Server, Ticket Granting Server), OpenID has 3 parties (RP, User, IdP);
- Kerberos specifies how a user should logon, OpenID doesn't and allows freedom for the IdP to implement its own solution;
- In Kerberos the Service, AS and TGS are authenticated to the user, in OpenID this is not the case for neither the RP nor the IdP;
- In Kerberos the Authentication of the user towards the authentication happens prior to any of the other steps, in OpenID these steps take place about halfway during the process.

4.2 DigiD

DigiD is an authentication system that most if not all Dutch government departments use for their electronic services. DigiD stands for digital identity. DigiD is based on the standard product called A-select. A-select is an authentication system that is based on Kerberos (section 4.1). The advantage of A-select is that the authentication mechanism (such as username and password) is separate of the application (such as a website). This allows you to change the authentication mechanism without having to change the application itself. This is an advantage of SSO in general. Citizens can use electronic services with the help of DigiD. The government departments can check the identity of the citizens with the DigiD protocol.

4.2.1 Protocol

There are three authentication levels within DigiD; basic, medium and high. The authentication levels basic and medium are implemented. Authentication level high will be available in the near future. Every authentication level uses its own set of authentication methods. The basic level of DigiD uses a username and password. The medium level of DigiD uses a username, password and a transaction code sent by SMS. A citizen can only use the medium authentication level if his mobile telephone number is known to DigiD. This mobile telephone number can only be used by one citizen. In the future the highest level of DigiD will use the electronic Dutch identitycard (eNIK). This authentication level will be going to use the PKIgovernment (PKIoverheid) certificates. Every government department can choose which authentication level is required for their electronic services.

The assurance level (section 2.1) of the protocol depends on the authentication level that is used. If the authentication level is high or medium, the assurance level is high and very high. But when the basic authentication level is used, the assurance level is medium. A DigiD can be matched to a unique citizen identifier, the Burgerservicenummer (an equivalent to the American Social Security Number).

As you can see figure 11 represents the Sequence diagram of the DigiD protocol. The steps in this protocol will be discussed in the description underneath. The numbers in the description refer to the ones in the diagram.

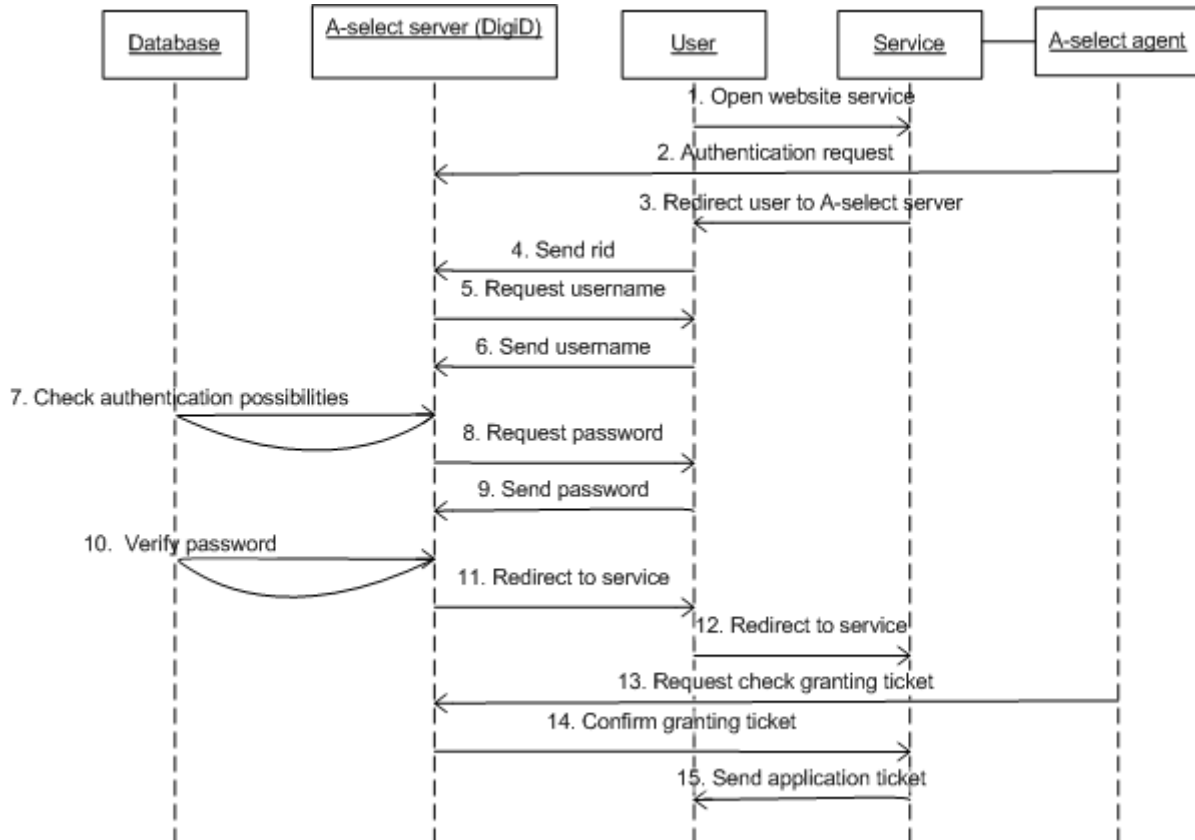


Figure 11: DigiD UML Sequence diagram

The first step is that the user visits a website of a government institution (1). Then the service concludes that the user has yet to identify himself. The A-select agent then sends a message to the A-select server (DigiD server) that the user will contact the A-select server to authenticate himself (1).

```
https://as.digid.nl/was/server?
request=authenticate&
app_url=https%3A%2F%2Fwww%2Ezutphen%2Enl%2Fsmartsite.dws&
app_id=zutphen_digid_portal&
shared_secret=123456-kd2s-s3kg-72kf-k2f3-mk2eaoe3&
a-select-server=DigiD1
```

- app_url: This is the URL of the service.
- app_id: This is the service id.
- shared_secret: This is a shared secret that the service and DigiD use to communicate with each other.
- a-select-server: This is the DigiD identifier.

Another thing that the A-select agent does in this step is that it reserves a session called a request identifier (rid) for the A-select server (2).

```
rid=A77C582B33C03912&
as_url=https://as.digid.nl/aselectserver/server?
request=login1&
aselect-server=DigiD1&
result_code=0000
```

- rid: This is a unique number.
- as_url: This is the URL of the DigiD server.
- a-select-server: This is the DigiD identifier.

The A-select agent will now redirect the user to the A-select server (3). The user sends the rid to the A-select server (4).

```
https://as.digid.nl/aselectserver/server?
request=login1&
rid=A77C582B33C03912&
a-select-server=DigiD1
```

- as_url: The URL of the DigiD server.
- rid: The unique number for this session.

The A-select server will then send a message to the user, in which he asks for the user's username (5). Then the user needs to send his username to the A-select server (6). When the A-select server receives the username it will check in the database which authentication mechanisms are available for that particular user (7). After this check has been done the A-select server asks the user for his password (8). The user replies with his password (9). The A-select server verifies the password in the database to verify whether it is the correct one (10). If there is a mobile telephone number of the user available, steps 8 to 10 could be repeated to use the medium level authentication (basic + SMS authentication). When the user is authenticated the A-select server will redirect the user back to the service (11) (12). In this step the A-select server sends a granting ticket to the service to confirm that the user really is who he says he is (based on burgerservicenummer) and at what authentication level the user has been authenticated.

```
https://www.zutphen.nl/smartsite.dws?aselect_credentials=X&
rid=A77C582B33C03912&
a-select-sever=DigiD1
```

- app_url: The URL of the service.
- aselect_credentials: Encrypted granting ticket.
- rid: The unique number for this session.
- a-select-server: This is the DigiD identifier.

After the service has received the granting ticket it will send it back through the A-select agent to the A-select server (13).

```
https://as.digid.nl/was/server?
request=verify_credentials&
aselect_credentials=X&
rid=A77C582B33C03912&
shared_secret=123456-kd2s-s3kg-72kfk2f3-mk2e-aoe3&
a-select-server=DigiD1
```

- as_url: The URL of the DigiD server.
- aselect_credentials: Encrypted granting ticket.
- rid: The unique number for this session.
- shared_secret: The shared secret that the service and DigiD uses to communicate with each other.
- a-select-server: This is the DigiD identifier.

The A-select server then checks the granting ticket if it is the same granting ticket that it had sent to the service and if the granting ticket is valid and has the right authentication level. If this is the case, the A-select server will send a confirm message to the service (14).

```
rid=A77C582B33C03912&
uid=190382582&
app_id= zutphen_digid_portal&
auth_level=10&
r_org=DigiD&
a-select-server=DigiD1&
result_code=0000
```

- rid: The unique number for this session.
- uid: The Burgerservicenummer of the user
- app_id: The service id.
- auth_level: Which authentication level is used.
- a-select-server: This is the DigiD identifier.

The final step is that the service sends the user an application ticket so that the user can use the service (15) [JOC07]. We based our examples of the messages between the involved parties on the following source [MON08].

4.2.2 Comparison with OpenID

OpenID and DigiD have some general things in common but there are also a lot of differences. The A-select server has three authentication levels which the service can choose to be used. In OpenID the authentication method (section 3.1.1.6) for the IdP isn't specified in the protocol. This means that there is no guarantee for a certain security level on that part of the OpenID protocol, because every IdP can choose which authentication method it wants to use, a good one or a bad one. OpenID is an open framework so that could be one of the main reasons why they didn't specify it, as an open framework means that it is open for people to create their own version of it, so most of the time they don't specify all the things so that people can creatively create their own version of it. But to give the users of OpenID a guarantee that there is a certain basic level of security would be good. An other difference is that in DigiD the service (sort of RP) makes in a certain way sure to the A-select server (sort of IdP) that it is talking to the user that it was intent to. This is done by using a rid (session)

(see protocol steps 2 & 4). The main difference with OpenID is that DigiD is more detailed in security aspects. One of the main reasons of that is that DigiD isn't an open framework; it is a proprietary solution. Another reason is that DigiD is used for privacy sensitive purposes. And misuse of this has more consequences than it has with OpenID. OpenID isn't used for privacy sensitive purposes because those websites don't support login with OpenID. It is also up to the user where he uses OpenID to login (more or less privacy sensitive). The OpenID implementations use an identifier that looks like a URL. This is because most of the time the username you want to choose has already been taken. In DigiD the identifier is a username. OpenID gives people the opportunity to create an IdP. The result of this is that there are many OpenID IdPs. DigiD only has one authentication server (A-select), but of course there are some redundant A-select servers. These servers are there to make sure that there is enough capacity available so that the authentication can be done and to see to it that, if one server crashes, the whole system does not go down. DigiD and OpenID are both SSO protocols but in practice the only SSO part of DigiD is that it uses one single username and password combination. In OpenID there are more SSO features such as: if a user has authenticated and authorized himself to the IdP, the user doesn't have to authenticate and authorize himself to all the RPs which he has access to. This only works for as long as the session takes. In DigiD this feature is blocked because all the services use 'forced logon' of A-select. This means that each time a user is forwarded to DigiD he needs to re-authenticate himself. Another reason why this doesn't work is that DigiD doesn't create cookies in any form. So DigiD cannot track the authentication state of its users [MON08]. OpenID supports user data, that is stored on the IdP, to be sent to the RP. A-select also supports this feature but it is not implemented in DigiD. The user data that is stored on the IdP could be fake so the RP cannot really trust the information. With DigiD this is not the case. There the user data is adequate and valid so this could be a big plus for the service [MON08]. DigiD and OpenID share the same security attack that is possible and that is phishing.

4.2.2.1 Similarities

- Both are Identity Management systems;
- Both are SSO protocols;
- DigiD has 3 parties (A-select (Database, A-Select server, A-select agent) User, Service), OpenID has 3 parties (IdP, User, RP);
- Phishing attack is possible.

4.2.2.2 Differences

- DigiD can be used by Dutch citizens, OpenID can be used by anyone;
- DigiD specifies which authentication mechanism is used, OpenID doesn't;
- DigiD has more security measures than OpenID;
- DigiD has a medium/ high/ very high assurance level, OpenID has a low assurance level;
- DigiD has one authentication server, OpenID has many;
- DigiD is a proprietary solution, OpenID is an open framework.

4.3 Microsoft Windows LiveID

Microsoft Windows LiveID is an identity people use to logon to Windows Live Messenger, Bing and many other services provided by Microsoft. The logon credentials of this identity consist of an emailaddress and a password. The emailaddress does not necessarily need to be

a hotmail (*@hotmail.com) or live (*@live.com) address. Microsoft allows people to use their own emailaddresses to register an account. Windows LiveID has existed for several years and is relatively new compared to other single sign-on solutions. However, it was formerly known as Microsoft .NET Passport which dates back to the mid nineties. Windows LiveID as well as the .NET Passport before is a proprietary system [MYL06]. Proprietary systems are not aimed at becoming web standards. Instead they service a selected set of service providers [MYL06].

Microsoft is planning to become an OpenID Identity provider [MICR08]. Tests in the Community Technology Preview (CTP) environment have been concluded. There is no schedule for the the release of OpenID functionality but Microsoft is actively working on this feature [MICR09].

4.3.1 Protocol

As you can see, figure 12 represents the Sequence diagram of the Windows LiveID protocol. The steps in this protocol will be discussed in the description underneath. The numbers in the description refer to the ones in the diagram.

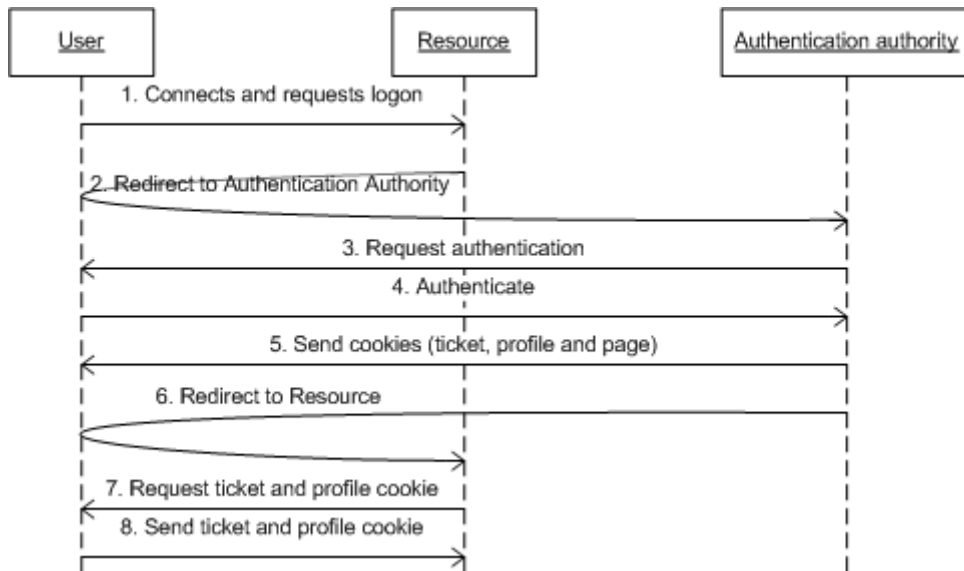


Figure 12: Microsoft Windows LiveID UML Sequence diagram

The first step is that the user is connecting to a resource and decides to login (1). The resource will then proceed by redirecting the user to one of Microsofts Authentication authorities (2). The authentication authority will then request the authentication credentials (3). The user will then logon with his emailaddress and password (4). When the credentials are correct the authentication authority will send three cookies (ticket, profile and page) to the user (5). The profile cookie stores all the pages the user is logged onto [LIN09]. The ticket cookie contains a unique identifier and a timestamp which will be used to verify the user by the resource [LIN09]. User information can be found in the profile cookie [LIN09]. The authentication authority will then continue by redirecting the user back to the Resource (6). The resource will now request the ticket and profile cookie from the user (7). The page cookie is only used for signing of [LIN09]. The user will supply these tickets and after the resource has verified them, the user will have logged on successfully (8).

4.3.2 Comparison with OpenID

The most easily recognizable similarity between Windows LiveID and OpenID is that both are Identity Management Systems and SSO protocols. They also share a low assurance level (section 2.1). Both are not supposed to be used to transmit important data. A Resource in Windows LiveID resembles the Relaying Party (RP) in OpenID. Both the Resource and the RP require a user to be authenticated. What in OpenID is called an Identity Provider (IdP) can be compared with an Authentication Authority in Windows LiveID. Both the Authentication authority and the IdP authenticate the user. The big difference between the two is that there is only one type of Authentication Authority and all servers which play this role are owned and managed by a single party, namely Microsoft. OpenID has many different Identity Providers that are owned by numerous parties.

Windows LiveID is a proprietary solution whilst OpenID is an open framework which is free to be used by anyone. Since Windows LiveID is a proprietary solution, the authentication authority is known. This means that there is no need for a discovery step or entering an identifier by a user. For Windows LiveID this also has the advantage that there is far less trust needed between the Resource and Authentication Authority. Both are mostly controlled by the same entity, namely Microsoft.

In OpenID the user needs to authorize the RP, but in Windows Live the Resource is authorized by the Authentication Authority. When in Windows LiveID the user has authenticated himself to the authentication authority, the user will receive a set of cookies and will then be redirected to the resource. In OpenID, after successfully authenticating himself, the user is redirected to the RP with the needed information inside this redirect link. The user will not need to visit the Authentication Authority when logging in to other Resources. The user will just send the ticket and profile cookies he received earlier. In OpenID the user will need to revisit the IdP for each new RP he wishes to authorize.

Step 2 (Redirect to Authentication Authority) from the Windows LiveID protocol can be compared with the Authentication Request step (section 3.1.1.5) of the OpenID protocol. Both of these steps redirect the user to the Authentication Authority/ IdP.

The implementations of step 3 (Request Authentication) and step 4 (Authenticate) are in most cases quite similar to the implementations of the Request Authentication and Authorization step (section 3.1.1.6) and Authenticate and Authorize step (section 3.1.1.7) of the OpenID protocol. OpenID does not specify how these steps should be implemented but in most implementations of this the IdP requests a username and password. The user will then supply his credentials after which he can authorize the RP. Authorization is missing in the Windows LiveID counterpart steps. Both sequences of steps have the same goal which is to authenticate the user towards the Authentication Authority/ IdP.

The Windows LiveID steps 5 (Send Cookies), 6 (Redirect to Resource), 7 (Request ticket and profile cookie) and 8 (Send ticket and profile cookie) combined serve the same purpose as the Positive Assertion step (section 3.1.1.8) of the OpenID protocol. During the Positive Assertion the IdP sends the user back to the RP with redirect link. This redirect link contains all the necessary information for the user to be authenticated towards the RP. After the Positive Assertion step has been completed the user will have been authenticated towards the user. Steps 5, 6, 7 and 8 of the Windows LiveID protocol accomplish the same as the Positive Assertion step. The user first receives the necessary information to authenticate himself. The user is then redirected to the Resource. The resource requests authentication and the user

proceeds by sending the data he received earlier. Now the user will have been authenticated towards the Resource.

4.3.2.1 Similarities

- Both are Identity Management systems;
- Both are SSO protocols;
- Both have a low Assurance Level;
- Resource and RP have similar roles;
- Authentication authority and IdP have similar roles;
- Windows LiveID step 2 (Redirect to Authentication Authority) is similar to the Authentication Request step (section 3.1.1.5) of the OpenID protocol;
- Windows LiveID step 3 (Request Authentication) is somewhat similar to the implementation of the Request Authentication and Authorization step (section 3.1.1.6) of the OpenID protocol; ¹
- Windows LiveID step 4 (Authenticate) is somewhat similar to the implementation of the Authenticate and Authorize step (section 3.1.1.7) of the OpenID protocol; ¹
- Windows LiveID steps 5 (Send Cookies), 6 (Redirect to Resource), 7 (Request ticket and profile cookie) and 8 (Send ticket and profile cookie) combined have the same goal as the Positive Assertion step (section 3.1.1.8) of the OpenID protocol.

¹ OpenID does not specify how these steps take place but implementations are in most cases quite similar apart from the authorization.

4.3.2.2 Differences

- Windows LiveID is a proprietary solution, OpenID is an open framework;
- Windows LiveID requires the use of cookies, OpenID does not;
- In Windows LiveID the resource is authorized by the authentication authority, in OpenID the user authorizes the RP;
- After being logged on a user will not need to revisit the authentication authority when he wants to use a different Resource. In OpenID a user needs to revisit the IdP for each RP to authorize them.

4.4 Google Federated Login

Google offers users many services. Federated login is the authentication mechanism of Google which is based on OpenID. Google works as an OpenID provider. Google supports the OpenID 2.0 protocol but also other extensions such as:

- OpenID Attribute Exchange 1.0 (This allows web developers to access user information (username and e-mail address) that is stored with Google. This can only be done if the user allows it.)
- OpenID User Interface 1.0 (This makes it possible to change to an alternative user experience for the authentication process. The default experience for the user is that he is being redirected away from the web application site to the Google authentication page. This extension makes it possible to stay on the web application site and create a popup for the authentication page.)

- OpenID+OAuth Hybrid protocol (This extension can be used to combine OpenID and OAuth (see [OAUTH10] for more information) authentication request. This can be useful for web developers who already use OAuth.)

Google provides the relaying party (web application) with an identifier that the web application can use to recognize the user. This identifier stays the same. The advantage of this is that the web application (RP) can recognize the user across multiple sessions.

4.4.1 Protocol

As you can see, figure 13 represents the Sequence diagram of the Google Federated Logon. The steps in this protocol are based on [GOOG10] and will be discussed in the description underneath. The numbers in the description refer to the ones in the diagram.

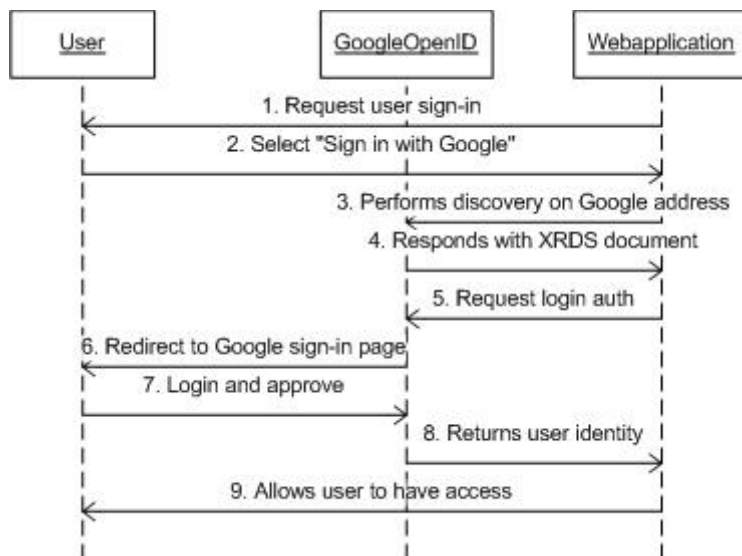


Figure 13: Google OpenID UML Sequence diagram

The federated login protocol is triggered when the user wants to use an option or function of a web application that requires the user to be authenticated. The first step is that the web application requests the user to login with his Google account (1). The next step is that the user decides to login with his Google account (2). The web application sends a “discovery” request to GoogleOpenID (IdP) to get information on the Google login authentication endpoint URL (3). Google responds to the web application with an XRDS document (section 3.1.2) that contains the endpoint URL (4). When the web application has discovered the endpoint URL in the XRDS document, it sends an authentication request to the endpoint URL (5). The user is then redirected to the Google federated login page (either in popup window or the same browser window) and is asked to login (6). The user logs in on the federated login page with his email address and password. When the user has done this, the user gets a message if he wants to proceed signing in to that web application (7). When the user chooses to approve signing in he will be returned to the web application (8). The final step is that the web application uses the Google-supplied identifier to recognize the user and allows this user to have access to certain services (9)[GOOG10].

4.4.2 Comparison with OpenID

The federated login protocol is similar to the OpenID protocol because the federated login protocol is based on the OpenID 2.0 protocol. There are some differences. One of the biggest differences is that it is a proprietary solution. It will only accept Google OpenID identifiers. This means that other OpenID identifiers from other IdP's don't work. Another great difference is that the federated SSO solutions use the concept Circle of Trust (CoT) and OpenID does not. There needs to be a trust relationship between the RP and the IdP. This has to be there because the IdP is responsible for the authentication process. And the other way around, the IdP needs to trust the information that it gets from the RP. In OpenID this relationship isn't there because everyone can create an IdP or RP. The lack of a CoT in OpenID has as a result that the level of trust moves from the application level to a social level [HEL09]. As mentioned before, everyone could create a IdP or RP and they could make a malicious one (section 3.2). This means that the user cannot trust the "application" but he needs to trust the person or organization that has created it.

The discovery step in Google federated login is slightly different from the discovery step (section 3.1.1.2) in OpenID. In the federated login protocol the RP sends a message directly to the GoogleOpenID with a request to get information of the Google login authentication endpoint URL. GoogleOpenID responds with an XRDS document which contains this Google login authentication endpoint URL (location of the IdP). In the OpenID protocol the RP first performs a normalization on the identifier before requesting information about the endpoint URL. This difference is there because the RP's that use OpenID can receive a lot of different types of identifiers. The federated login protocol only recognizes one.

As you can see in the federated login protocol, it misses the Association request (section 3.1.1.3) and response steps (section 3.1.1.4) that are present in the OpenID protocol. These steps are optional in the OpenID protocol. This isn't an issue for security because the federated login protocol uses HTTPS for communication. In the described documentation of federated login protocol they don't state that they only use HTTPS for communication. But since the Google login authentication endpoint URL is an HTTPS address (<https://www.google.com/accounts/o8/ud>) we can assume that they use only HTTPS for communication. We have also found HTTPS being used during our logging in in every Google application (Gmail, Youtube, etc.).

Step 5 Request login auth in the Google Federated login protocol is similar to the Authentication request step (section 3.1.1.5) from the OpenID protocol. In these steps the web application/RP redirects the user to the authentication mechanism (GoogleOpenID/IdP). Step 6 Redirect to Google sign-in page and step 7 Login and approve in the Google Federated login protocol are quite similar to the Request authentication and authorization step (section 3.1.1.6) and Authenticate and authorize step (section 3.1.1.7) from the OpenID protocol. A difference in these steps is that the Google federated login protocol specifies which authentication mechanism is used to login. The federated login protocol prescribes that a user has to log on with an email address and password. The similarity these steps have is that they have the same goal which is to authenticate the User. The OpenID protocol doesn't specify which authentication mechanism (section 3.1.1.6) needs to be used. Step 8 Return user identity of the Google federated login protocol is similar to the Positive Assertion step (section 3.1.1.8) from the OpenID protocol. The similarity is that they redirect the user to the RP, if the user has authenticated himself successfully. The difference is that in the Positive Assertion step a message is sent to the RP with some information. Step 9 Allows user to have access of the Google federated login protocol has some similarity with the Verification step

(section 3.1.1.9) from the OpenID protocol. In both these steps they provide the user with access. A great difference between these steps is that in the OpenID protocol the information (of the user) sent from the IdP to the RP is being checked by the RP if it is valid.

The federated login protocol also offers the users the possibility to use Open Authorization (OAuth). OAuth can be used in the federated login protocol to exchange user-specific (calendar, address book, etc) data with a Google service. In general the OAuth allows users to share user-specific data from one website to another website without having to hand out their username and password [OAUTH10]. In the OpenID protocol step 6 Request authentication and authorization (section 3.1.1.6) and step 7 Authenticate and authorize (section 3.1.1.7) it is possible to send information of the user to the RP. But this isn't specified in the OpenID specifications. There are some extensions for OpenID that handle this like the Simple Registration Extension (SREG)(section 3.1.1.7) but they are not as elaborate as OAuth. The steps of OAuth are integrated in the federated login protocol. In step 5 of the federated login protocol the web application requests a token. GoogleOpenID responds in step 8 with a token to the web application. In step 9 the web application uses the token to gain access to the user's Google services.

4.4.2.1 Similarities

- Both are Identity Management systems;
- Both are SSO protocols;
- Both have a low Assurance Level;
- Web application and RP have similar roles;
- GoogleOpenID and IdP have similar roles;
- Google Federated Login step 5 (Request login auth) is similar to the Authentication Request step (section 3.1.1.5) of the OpenID protocol;
- Google Federated Login step 6 (Redirect to Google sign-in page) and step 7 (Login and approve) is quite similar to the steps Request authentication and authorization step (section 3.1.1.6) and Authenticate and authorize step (section 3.1.1.7) of the OpenID protocol;
- Google Federated Login step 8 (Return user identity) is similar to the Positive Assertion step (section 3.1.1.8) of the OpenID protocol.

4.4.2.2 Differences

- Google Federated Login uses the concept of Circle of Trust;
- Google Federated Login is a proprietary solution based on OpenID, OpenID is an open framework;
- The discovery step in Google Federated Login is slightly different (no normalization);
- Google Federated Login has no Association steps, OpenID does have association steps but they are optional;
- Google Federated Login specifies which authentication mechanism it uses (step 7);
- Google Federated Login doesn't have a check of validation of the information sent from the IdP to the RP in step 8, which OpenID has in its Verification step (section 3.1.1.9).

5 CAcert

In this chapter CAcert will be described. First we will introduce CAcert. We will describe how CAcert is different from other Certificate Authority or Certification Authority (CA) in the first section. In the second section we will explain how you can use CAcert.

CAcert is a non-profit association that exists since July 2003 and has as a goal “promote awareness and education on computer security through the use of encryption, specifically with the X.509 family of standards“ [CACE10a]. CAcert gives users the option to create x.509 certificates on the basis of ‘web of trust’ for free. CAcert signs server and client certificates and generates client certificates for users that request them. Web of trust means that people confirm that you are the person that you say you are. The more independent people confirm that you are the person you claim to be the more likely it is that you are that person.

A problem of CAcert is that their root certificate is not by default included in any of the major web browsers (section 1.1). This means that people that want to use CAcert certificates need to manually install the root certificate into their web browser. It also means that everyone that visits your website will be confronted with a warning if they haven't got the root certificate installed.

5.1 What is CAcert

CAcert is a CA that uses a Web of Trust (WoT) to confirm the identities of parties it issues certificates to. CAcert issues both client certificates and server certificates to users. The features a user has access to grow in number as he gains more assurance points.

5.1.1 Certification Authority

A CA is a party that issues certificates to other parties. A CA is a trusted third party, it facilitates between parties that trust it. CA is an integral part of a Public Key Infrastructure (PKI). Descriptions of both the CA and the PKI can be found in section 2.2.

CAcert is different compared to most other CA due to the fact that CAcert makes use of a WoT (section 5.1.2) to verify the identity of its users. Some CAs require their users to make a physical appearance at an office of the CA. This method is costly because the CA will need to have offices near their users. Some other CAs only require a copy of passport to be sent by email or mail. This method is less costly but it doesn't authenticate (section 2.1.4) the user. All the CA knows is that someone sent a copy of a passport of somebody. There is no way to know if the sender and the owner of the passport are the same individual. With a WoT CAcert reaps the benefits of both mentioned methods. A WoT is cost efficient because there is no need for numerous offices. Authentication still happens face to face with an assurer.

5.1.1.1 Classification of certificates

Each CA can classify its certificates. CAcert uses two classes, class 1 and class 3. There are two matching root certificates. If you create a client certificate which is signed by class 1 you will need to have a root class 1 certificate installed in the browser. If you have a class 3 certificate you need to have either the class 1 or class 3 root certificate installed [CACE10b].

The class 3 root certificate was signed with the class 1 root certificate. The difference between them is that class 3 is primarily used for certificates including the names of assured members (and is for assured members only) and for class 1 this is not the case. In our proof of concept CAcertID (chapter 6) we use class 1 certificates that include the owner's name. These certificates that contain a name can only be owned by assured users.

5.1.1.2 Client Certificate

A client certificate is a certificate that a user can use to login on websites and to securely send email (section 2.2). In the case of securely sending email the user will publish his certificate. People will then be able to use the public key in the certificate to encrypt the message. The user can then use his private key to decrypt it. In figure 14 you can see an example of a client certificate as it is shown in a browser. The attributes of a client cert are the same as the ones used in the server certificate (section 5.1.1.3).

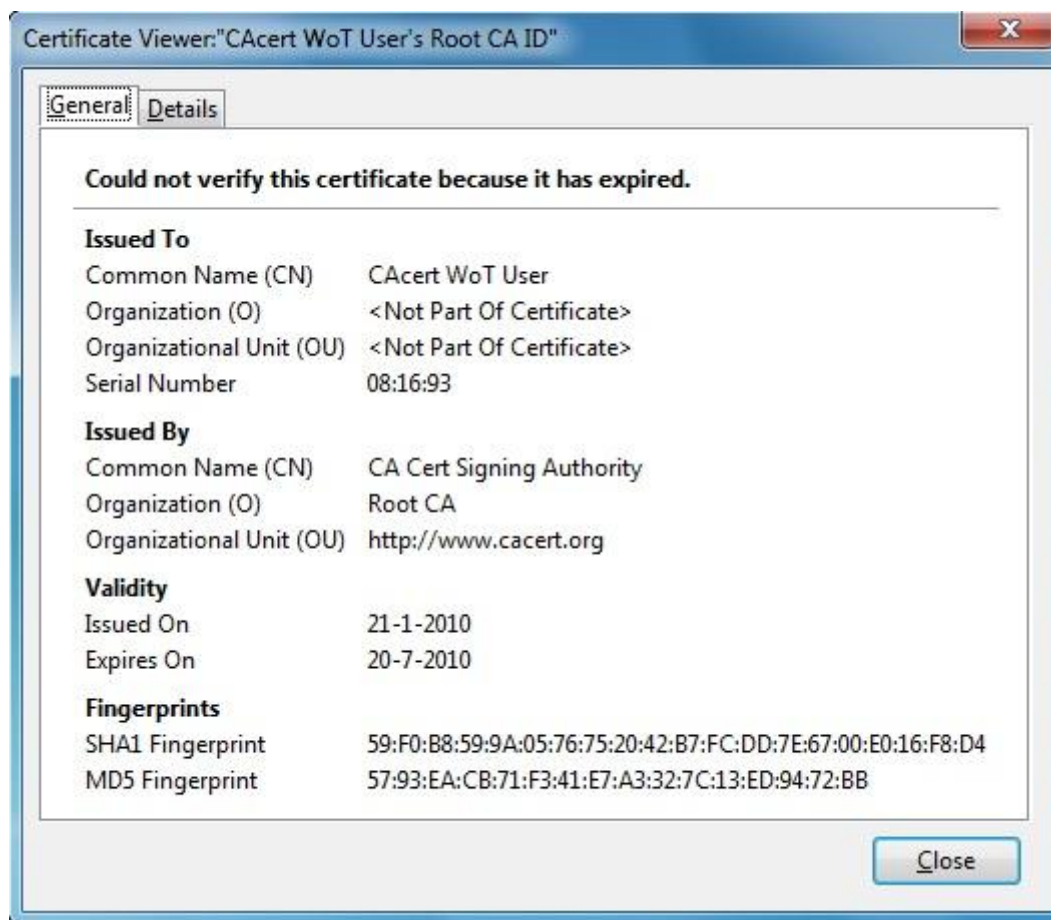


Figure 14: Client certificate

If you have 50 or more CAcert assurance points you can choose if you want your name included in the client certificate. This client certificate can also be used to logon to the CAcert website (optional). It is also possible to add Single Sign On (SSO) ID information to your client certificate, which could be useful if you use your certificate for SSO purposes. Our solution CAcertID (chapter 6), however, does not use this information.



5.1.1.3 Server Certificate

A server certificate is a certificate that is installed on servers. Servers that have a certificate installed, can use it to set up secure connections with clients. Servers, that have a certificate, have had to generate a secure key pair. The responsibility for generating a secure key pair lies in the hand of the server not the CA. This key pair can then be used to generate a Certificate Signing Request (CSR). The CSR can then be sent to a CA which signs it and provides you with your certificate. CAcert only includes the CommonName (web-address) and SubjectAltName from your CSR in your certificate automatically. A user will need take an extra step on top of having more than 50 assurance points to get more information in his certificate. He will need to send a letter of incorporation to include the name of his organization. View section 2.2.1. for more detailed information about certificates. The following text block is an example of a server certificate.

```
-----BEGIN CERTIFICATE-----
MIERTCCAI2gAwIBAgIDCIJXMA0GCSqGSIb3DQEBBQUAMHkxEDAObgNVBAoTB1Jv
b3QgQ0ExHjAcBgNVBAsTFWh0dHA6Ly93d3cuY2FjZXJ0Lm9yZzEiMCAGAlUEAxMZ
Q0EgQ2VydCBTawduaW5nIEF1dGhvcml0eTEhMB8GCSqGSIb3DQEJARYSc3VwcG9y
dEBjYWN1cnQub3JnMB4XDTEwMDQxNjE1MDQxMjE0MTI1XjAUMDQxMTI1MDQxMjE0
MTI1MBIGAlUEAxMLb25ub3Y5b251b251bmkuY2MwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGB
AKWOCH1loIAud/HhBdqolFhd7uist3GJM04FE8UhvSghWVqo8NE3b/ccwKDmdMv4
1Xell8Mm2/S7xw9vDiu2eJwAbh4rhDjzCjj8QJHNz4sBnxv/PS4gTbs0p8pTtRbd
xkfTCd/XalEwCxaNU0JD85zuSe0H2YauXkUGVmI12j/BAGMBAAGjgbwgbkwDAYD
VR0TAQH/BAIwADA0BgNVHSUELTAarBggrBgEFBQcDAGYIKwYBBQUHAwEGCWCGSAGG
+EIEAQQYKKwYBBAGCNwoDAzALBGNVHQ8EBAMCBaAwMwYIKwYBBQUHAQEJEZAlMCMG
CCsGAQUFBzABhhhdodHRwOi8vb2NzcC5jYWN1cnQub3JnLzAxZgBGNVHREEKjAoggtv
bm5vLnVuaS5jy6AZBggrBgEFBQcIBaANDAtvbm5vLnVuaS5jyZANBgkqhkiG9w0B
AQUFAAOCAgEAOYRFQ1Jvq7g2zvSolQWUPcS8WM806C1I7r6OpMCUnriT3TyBkKOG
Y8SQ4A2QyVKpZgSIbdePQFxd3SWHh1LK8n45ZiORr6QmGxK6IOMjo+ZU6S0ymTD+
WMbEio7rg9gYIF4deHUNjaPZFFnybn4ExwMhVufc43921oHfRNajwNL8AblHpQbB
DURcqrBfpQ3bXRCOCGV/v4IqyidpSDB6607MTop6eb49P7KqP7+DXWzcely97Sg
g6ZA8z1bJyWxed5ZThYePjrIDU5NGL6GoU0ax9XqUPrGzQpfYJm0tVLLXkESmy9K
s/R6JZmETXaKd5uilxUGZijUpoiEnztacgd1mN8xpPI/Vfh2qVnLmWoiuc/jwGjZ
YMU6l9QrDkqAV7Pfn7KBBCjIKcl6aFxxgGIM2sDV2tImXu3CSM3J+va2DimHh19f+
vk1gNYqpuv7MYqJ01kmKq41y0KthadLjGxeJtTAg0CrbiN0N7mQ2pReCwuKjI18m
mzVvyernBmIGDP5LT2tX9gC9CK09ngt1Lk9q3SE4OyBryVdvDvaM1oBgX3IRpDWG
PzNE4ztAlWbGY3Ge2gH2m3T0ReHqMz3lUi623+Xo/LAlaJYW7WagB3APvhz1xs5q
A3dYD7G/gLgyfJcPGGDT6Qu/jzTj8W/hrlNzX8p01sdVWAqQcliNok=
-----END CERTIFICATE-----
```

The certificate contains information about the issuer, validity and subject of the certificate. It also lists the version, serial number, X509v3 extensions, key and signature. The version is a reference to the version of the protocol X509v3. The serial number is a unique identifier provided by the CA. The X509v3 extensions are an array of attributes that confine the usage of the certificate. If you decoded the certificate, you would get a text block similar to the one described underneath.

```
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 557655 (0x88257)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer:
        emailAddress = support@cacert.org
        commonName = CA Cert Signing Authority
        organizationalUnitName = http://www.cacert.org
        organizationName = Root CA
```




```

Validity
    Not Before: Apr 16 00:42:16 2010 GMT
    Not After : Oct 13 00:42:16 2010 GMT
Subject:
    commonName          = onno.uni.cc
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
        Modulus (1024 bit):
            00:a5:8e:08:7d:65:a0:80:2e:0f:f1:e1:05:da:a8:
            95:f1:dd:ee:e8:ac:b7:71:89:31:0e:05:13:c5:21:
            bd:2a:a1:59:5a:a8:f0:d1:37:6f:f7:1c:c0:a0:e6:
            74:cb:f8:d5:77:a5:97:c3:26:db:f4:bb:c7:0f:6f:
            0e:2b:b6:78:9c:00:6e:1e:2b:84:38:f3:0a:38:fc:
            40:91:cd:cf:8b:01:9f:1b:ff:3d:2e:20:4d:bb:34:
            a7:ca:53:b5:16:dd:c6:47:d3:09:df:d7:6b:51:30:
            0b:16:8d:53:42:43:f3:9c:ee:49:ed:07:d9:86:ae:
            5e:45:06:56:62:35:da:3f:c1
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Extended Key Usage:
        TLS Web Client Authentication, TLS Web Server
Authentication, Netscape Server Gated Crypto, Microsoft Server Gated Crypto
    X509v3 Key Usage:
        Digital Signature, Key Encipherment
    Authority Information Access:
        OCSP - URI:http://ocsp.cacert.org/
    X509v3 Subject Alternative Name:
        DNS:onno.uni.cc, othername:<unsupported>
Signature Algorithm: sha1WithRSAEncryption
39:84:45:42:52:6f:ab:b8:36:ce:f4:a8:95:05:94:3d:c4:bc:
58:cf:0e:e8:29:48:ee:be:8e:a4:c0:94:9e:b8:93:dd:3c:81:
90:a3:86:63:c4:90:e0:0d:90:c9:52:a9:66:04:88:6d:d7:8f:
40:5c:5d:dd:25:87:87:52:ca:f2:7e:39:66:23:91:af:a4:26:
1b:12:ba:20:e3:23:a3:e6:54:e9:2d:32:99:30:fe:58:c6:c4:
8a:8e:eb:83:d8:18:20:5e:1d:78:75:27:8d:a3:d9:15:f9:f2:
6c:de:04:c7:03:21:56:e7:dc:e3:7f:76:d6:81:df:44:d6:a3:
c0:d2:fc:01:b9:47:a5:06:c1:0d:44:5c:aa:b6:c5:a5:b4:37:
6f:1a:c2:38:21:95:fe:fe:08:ab:28:9d:a5:20:c1:eb:ad:3b:
31:3a:29:e9:e6:f8:f4:fe:ca:a8:fe:fe:0d:75:b3:71:e9:72:
f7:b4:a0:83:a6:40:f3:3d:5b:27:25:b1:79:de:59:4e:16:1e:
3e:3a:c8:0d:4e:4d:18:be:86:a1:4d:1a:c7:d5:ea:50:fa:c6:
cd:0a:5f:60:99:b4:b5:52:cb:5e:41:12:9b:2f:4a:b3:f4:7a:
25:99:84:4d:76:8a:77:9b:a2:d7:15:06:66:28:d4:a6:88:84:
9f:3b:5a:72:07:75:98:df:31:a4:f2:3f:55:f8:76:a9:59:cb:
99:6a:22:b9:cf:e3:c0:68:d9:60:c5:3a:97:d4:2b:0e:4a:80:
57:b3:df:9f:b2:81:04:28:c8:29:c9:7a:68:5c:60:18:83:36:
b0:35:76:b4:89:97:bb:70:92:33:72:7e:bc:0d:83:8a:61:e1:
d7:d7:fe:be:4d:60:35:8a:a9:ba:fe:cc:62:a2:4e:d6:49:8a:
ab:8d:72:d0:ab:61:69:d2:e3:19:77:89:4e:d0:20:d0:2a:db:
88:dd:0d:ee:64:36:a5:17:82:c2:e2:a3:22:5f:26:9b:35:6f:
c9:ea:e7:06:62:06:0c:fe:4b:4f:6b:57:f6:00:bd:08:ad:3d:
9e:0b:75:2e:4f:6a:dd:21:38:3b:20:6b:c9:57:6f:0e:f6:8c:
d6:80:60:5f:72:11:a4:35:86:3f:33:44:e3:3b:40:95:66:c6:
63:71:9e:da:01:f6:9b:74:f4:45:e1:ea:33:3d:e5:52:2e:b6:
df:e5:e8:fc:b0:25:68:96:16:ed:66:a0:07:70:0f:be:1c:f5:
c6:ce:6a:03:77:58:0f:b1:bf:80:b8:18:7d:f2:5c:3c:61:83:
4f:a4:2e:fe:36:53:8f:c5:bf:86:b9:4d:cd:7f:29:3b:5b:1d:
55:60:2a:41:cd:62:36:89

```

5.1.1.3.1 Certificate Signing Request

A Certificate Signing Request (CSR) is a file that a user sends to request his certificate to be signed by CA. The CSR is generated after the user has generated an asymmetric key pair. The CSR contains the public key, email address, web-address, the location information of the user and a signature. The signature consists of the entire CSR apart from the signature itself. The signature is created with the private key of the user. The following text block is an example of a CSR.

```
-----BEGIN CERTIFICATE REQUEST-----
MIIC2zCCAcMCAQAwwZUxCzAJBgNVBAYTAK5MMRMwEQYDVQQIEwphZWxkZXJ5YW5k
MRIwEAYDVQQHEwlYXJuc3ZlbGQxITAFBgNVBAoTGEIudGVybmV0IFdpZGdpdHMg
UHR5IEEx0ZDEUMBIGA1UEAxMLb25uby51bmkuY2MxJDAiBgkqhkiG9w0BCQEFW9u
bm9iZXJmZWxvQHhZNGFsbC5ubDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC
ggEBANJ8j0hu5AtPPic7+1H23AWxi2d0SieQh+vI+D2XvNpk6ZLJQL9U+5yjAqUm
C1VgDwvjACVR5chD4B6fSRKQhSTYDtozuaDmvEyWo/tpo9BdWg13wuqlmSD749Me
98OrKphGtpVzrDmrbyGeINNm+FKwOlG37NiczcCb4Vrc1CfH1m6y22I3pnw7Xjae
i+X4/vk85kqFTN2GexvD5FbehifkTb8/HHHVdUQciNs1zcAohTDExd8C/2+rWXTF
HS7jrvDkFAF4EeryKuZ4hiXbjvPeUcEkwNoPyyVmRf4426724VXR4kRHhyQ0jj
eRf1V7KNP+qUJyAfVbsqdUVO/1UCAwEAaAAMA0GCSqGSIb3DQEBAQUAA4IBAQBUs
PVTonsQg1PeUgGklvHVRk6+mC9p1HLtdMABsgANKb/lq8jQRsDOElT+rKjqcEe
r6clFcKhxXz8vQczDc8vyfQK2MPeaSRqxBlYOv2yk7X+4F+6ErTr3prq2i4gwphY
1muQg77Mj9Cc0cZYVB/8fRzR5UuJf1O2vUx0JKDpaqlcKd1qdNFiZM41D8Ie18q
+L9H5sIgYSN6qa1W+KIw8w+Z+IilmqE/XUzM66/rBJHKxy2CwBoBoKBkVKmTr1Ba
SzTdRlqeODR7bKYuFNIj6djY70o6QL+2xF6OqNmGYXjflZ6npH8QVCXZaUvS/oGk
LSFMoy4FsR6ADX29A9Ps
-----END CERTIFICATE REQUEST-----
```

If you were to decode this CSR, you would get something similar to the following text block.

```
Certificate Request:
Data:
  Version: 0 (0x0)
  Subject:
    emailAddress      = onnoberfelo@xs4all.nl
    commonName        = onno.uni.cc
    organizationName  = Internet Widgits Pty Ltd
    localityName      = Warnsveld
    stateOrProvinceName = Gelderland
    countryName       = NL
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (2048 bit)
      Modulus (2048 bit):
        00:d2:7c:8f:48:6e:e4:0b:4f:3c:87:3b:fb:51:f6:
        dc:05:b1:8b:67:74:4a:27:90:87:eb:c8:f8:3d:97:
        bc:da:64:e9:92:c9:40:bf:54:fb:9c:a3:02:a5:26:
        0b:55:60:0f:0b:e3:00:25:51:e5:c8:43:e0:1e:9f:
        49:12:90:85:24:d8:0e:da:33:b9:a0:e6:bc:4c:96:
        a3:fb:69:a3:d0:5d:5a:0d:77:c2:ea:a5:99:20:fb:
        e3:d3:1e:f7:c3:ab:2a:98:46:b6:95:73:ac:39:ab:
        6d:81:9e:20:d3:66:f8:52:b0:3a:51:b7:ec:d8:9c:
        ce:a0:9b:e1:5a:dc:d4:27:c7:d6:6e:b2:db:62:37:
        a6:7c:3b:5e:36:9e:8b:e5:f8:fe:f9:3c:e6:4a:85:
        4c:dd:86:7b:1b:c3:e4:56:de:86:21:64:4d:bf:3f:
        1c:71:d5:75:44:1c:88:db:35:cd:c0:28:85:30:c4:
        c5:df:02:ff:6f:ab:59:74:c5:1d:2e:e3:ae:f0:e4:
        14:01:78:11:ea:d8:2a:e6:78:86:25:db:8e:f3:de:
        51:c1:24:a6:ec:0d:a0:fc:ae:56:64:5f:e3:8d:ba:
```

```

ef:6e:15:5d:1e:24:44:78:72:43:48:e3:79:17:f5:
57:b2:8d:3f:ea:94:27:20:1f:55:bb:2a:75:45:4e:
fe:55
Exponent: 65537 (0x10001)
Attributes:
  a0:00
Signature Algorithm: sha1WithRSAEncryption
6e:4b:e3:d5:4e:89:ec:42:0d:4f:79:48:06:92:5b:c7:55:19:
3a:fa:60:bd:a7:51:cb:b5:d3:00:06:c8:00:36:46:ff:96:af:
23:41:1b:03:38:49:53:fa:b2:a3:a9:c1:1e:af:a7:25:15:c2:
a1:c5:7c:fc:bd:07:33:0d:cf:2f:c9:f4:0a:d8:c3:de:69:24:
6a:c4:1d:58:39:5d:b2:93:b5:fe:e0:5f:ba:12:b4:eb:de:9a:
ea:da:2e:20:c2:98:58:d6:6b:90:83:be:cc:8f:d0:9c:d1:c6:
58:54:1f:fc:7d:1c:d1:e5:4b:89:7f:53:b6:bd:4c:74:24:a0:
e9:6a:a9:5c:a8:a7:75:a9:d3:45:89:93:38:94:3f:08:7b:5f:
2a:f8:bf:47:e6:c2:20:61:23:7a:a9:ad:56:f8:a2:30:f3:0f:
99:f8:88:b5:9a:a1:3f:5d:4c:cc:eb:af:eb:04:91:ca:c7:2d:
82:c0:1a:01:a0:a0:64:54:a9:93:af:50:5a:4b:34:dd:46:5a:
9e:38:34:7b:6c:a6:2e:14:d2:23:e9:d8:d8:ef:4a:3a:40:bf:
b6:c4:5e:8e:a8:d9:86:61:78:df:95:9e:a7:a4:7f:10:54:25:
d9:69:4b:d2:fe:81:a4:2d:21:4c:a3:2e:05:b1:1e:80:0d:7d:
bd:03:d3:ec

```

We do not know how or why the public key length was reduced from 2048 bit in the CSR to 1024 in the certificate.

5.1.2 Web of Trust

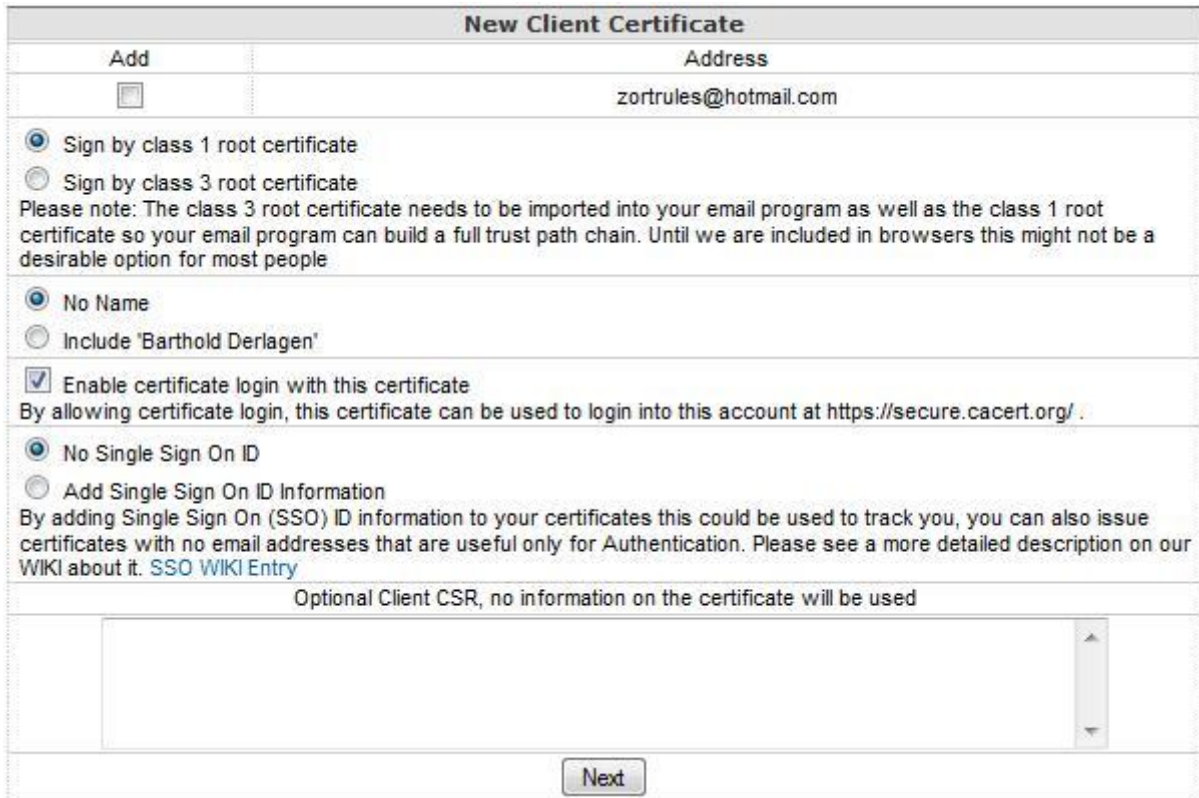
Web of trust (WoT), sometimes known as friend of a friend (FoaF), is a decentralized trust model. To get people to confirm your identity CAcert works with assurers. This works as follows: when an assurer assures a user, he signs the client certificate of the user. Assurers can grant from 10 to 35 points depending on their status. The status of an assurer is determined by the amount of people he has assured and is in some cases active in the startup process of CAcert. To get your identity assured by an assurer you need to show your passport and give an identity verification form to the assurer. You can find an assurer who lives nearby on the website of CAcert and send a message to meet somewhere. Once an user has received 100 points, he can take a test to become an assurer. Services like signing a server certificate do not require a user to have 100 points. Users that have 50 or more points have more functions at their disposal. Assurers gain points for each person they assure and will be able to grant more points when they have assured a certain amount of users.

5.2 How to use CAcert

The first thing that a new user needs to do is register on the CAcert website (www.cacert.org). The next thing you need to do is install the root certificate of CAcert into your web browser. You can find these root certificates on the website of CAcert. You will need to install these root certificates to be able to generate a client certificate. Another purpose of these root certificates is that you will not receive an error when visiting a website that has a CAcert server certificate. If you generate a client or server certificate you can choose by which root certificate it will be signed.

5.2.1 Client Certificate

When you are registered you can create a client certificate. If you have installed a root certificate you can proceed by creating a client certificate. The section client certificates on the website helps you step by step to create a client certificate. While creating a client certificate you can decide which email address you would like to use for the client certificate. You can also choose by which root certificate class it needs to be signed. You can choose whether or not you want to be able to use the certificate to logon to the CAcert website. You can optionally choose to include SSO ID information.



The screenshot shows a web form titled "New Client Certificate". It has two columns: "Add" and "Address". The "Address" field contains "zortrules@hotmail.com". Below the form, there are several sections of options:

- Signing Options:**
 - Sign by class 1 root certificate
 - Sign by class 3 root certificate

Please note: The class 3 root certificate needs to be imported into your email program as well as the class 1 root certificate so your email program can build a full trust path chain. Until we are included in browsers this might not be a desirable option for most people
- Name Options:**
 - No Name
 - Include 'Barthold Derlagen'
- Enable certificate login:**
 - Enable certificate login with this certificate

By allowing certificate login, this certificate can be used to login into this account at <https://secure.cacert.org/>.
- SSO ID Options:**
 - No Single Sign On ID
 - Add Single Sign On ID Information

By adding Single Sign On (SSO) ID information to your certificates this could be used to track you, you can also issue certificates with no email addresses that are useful only for Authentication. Please see a more detailed description on our WIKI about it. [SSO WIKI Entry](#)
- Optional Client CSR:**

Optional Client CSR, no information on the certificate will be used

At the bottom of the form is a "Next" button.

Figure 15: Client certificate options

5.2.2 Server Certificate

Getting a server certificate takes a bit more effort than getting a client certificate. To be able to have your certificate signed you must first generate one yourself. You can do this on either a linux or windows based system, if your server installation includes OpenSSL. The examples in the following paragraphs do not need to be followed to the letter but it is in line with section 2 in Attachment A. You will need to have a certain emailaddress bound to your domain to register your domain with CAcert. When you have obtained your server certificate you can install it in the config file of your webserver (section Attachment A.2).

5.2.2.1 Registrating a domain

You can add domains of websites you own to create server certificates for it. CAcert will verify if you own that domain by sending an email to a specific emailaddress account of that domain. This e-mail contains a link which you need to go to, to confirm that you own that specific domain.

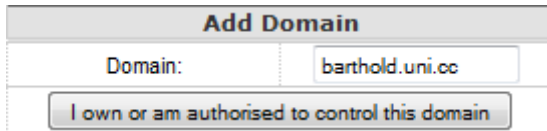


Figure 16: Adding a domain on the CAcert website

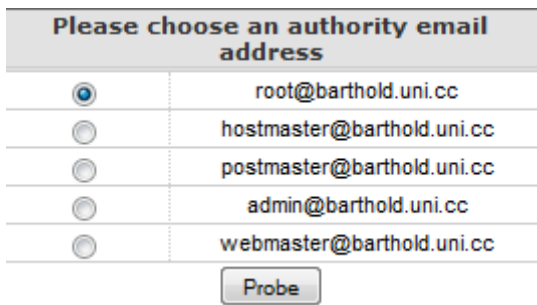


Figure 17: Choosing an emailaddress to receive a confirmation email.

5.2.2.2 Generating a Server Certificate

When you have added your domain to your account and the domain has been verified, you can have your server certificate signed by CAcert. The first thing that you need to do is to generate a key pair and a Certificate Signing Request (CSR). You can also decide by which root certificate the server certificate needs to be signed. There are two classes available 1 and 3, the same as with the client certificate.

5.2.2.2.1 Linux

On a linux system it is slightly less complicated to generate a key or certificate signing request. Open the terminal and execute the following command:

```
openssl genrsa -out server.key 2048
```

You have generated your private key and with this you can create a certificate signing request with the execution of the following command:

```
openssl req -new -key server.key -out server.csr
```

You can now find both the key as well as the certificate signing request in your local home directory. Open server.csr and copy the contents to the textbox on cacert.org (see figure 18) to get a signed certificate. Save this certificate as server.crt. Copy your certificate and key to /etc/apache2.

5.2.2.2.2 Windows

On a Windows system generating the key takes some more effort than on a linux based system. Create a batch file (*.bat) with the following content [SMIT10]. The directories mentioned are the default ones, change them if necessary.

```

@echo off

if not defined apache_dir set apache_dir=C:\Program Files\Apache Software
Foundation\Apache2.2
if not defined apache_conf_dir set apache_conf_dir=%apache_dir%\conf
if not defined openssl_conf set openssl_conf=%apache_conf_dir%\openssl.cnf
if not defined openssl_opts set openssl_opts=-config "%openssl_conf%"
if not defined openssl set openssl=%apache_dir%\bin\openssl.exe

if not exist "%apache_dir%" (
    echo Directory not found: "%apache_dir%"
    goto :eof
)

if not exist "%apache_conf_dir%" (
    echo Directory not found: "%apache_conf_dir%"
    goto :eof
)

if not exist "%openssl_conf%" (
    echo File not found: "%openssl_conf%"
    goto :eof
)

if not exist "%openssl%" (
    echo File not found: "%openssl%"
    goto :eof
)

pushd "%apache_conf_dir%"

"%openssl%" req %openssl_opts% -new -out server.csr || goto :eof
"%openssl%" rsa -in privkey.pem -out server.key || goto :eof
"%openssl%" x509 -in server.csr -out server.crt -req -signkey server.key -
days 3650

popd

```

Source: [SMIT10]

Start command prompt and browse to the directory in which you have placed your batch file. Execute the batch and answer the questions. If you have doubts on what you should answer to each question, note that most are optional.



5.2.2.3 Getting your Server Certificate Signed

Now that you have generated your asymmetric keys and CSR you can get your public key signed by supplying the certificate signing request file (server.csr) (Figure 18). On the CAcert website you can copy the contents of this file into a text box. You can do this by opening it with your favorite text editor. Rename the file you get back from the certificate signer and rename it to server.crt. Copy the file to the Apache configuration directory.

*** Please Note. All information on your certificate will be removed except the CommonName and SubjectAltName field, this is because it's an automated service and cannot automatically verify other details on your certificates are valid or not. If you are a valid organisation and would like more details to appear on certificates, you will need to have at least 50 assurance points and you need to send us a copy of your document of incorporation. Then we can add those details to your certificates. Contact us for more information on our organisational services. ***

- Sign by class 1 root certificate
- Sign by class 3 root certificate

Please note: The class 3 root certificate needs to be setup in your webserver as a chained certificate, while slightly more complicated to setup, this root certificate is more likely to be trusted by more people.

Paste your CSR(Certificate Signing Request) below...

Figure 18: Creating server certificate.

6 Proof of Concept

In this chapter our proof of concept is described. Our proof of concept is to ascertain whether it is possible to develop an OpenID provider where you can sign-in with your CAcert client certificate. In section 6.1 we describe the idea behind our solution. Section 6.2 describes the design choices we have made. Section 6.3 describes which OpenID package we have chosen and why. Section 6.4 describes which modifications we have made in the package, to enable sign-in with your CAcert certificate. We end this chapter with section 6.5 in which we describe the conclusion derived out of developing the proof of concept and future work that needs to be done.

6.1 Motivation for our solution

We set out to find a solution for the problems described in the introduction (section 1.1) of this master thesis; these are briefly described again underneath. There are a lot of security problems in OpenID (section 3.2). We wanted to find a way to improve the security of OpenID. The main advantage of OpenID is that it is an open framework. An open framework is highly adaptable. This means that we can change a lot of the aspects of OpenID. This is the main reason why we have chosen OpenID.

CAcert (chapter 5) has the advantage that it gives users the opportunity to create X.509 certificates (section 2.2.1) on the basis of a Web of Trust (WoT) (section 5.1.2) for free. There are a lot of certificate authorities that charge a fee to users for providing a certificate. CAcert offering certificates without charge is a great advantage. This is the main reason why we chose CAcert. But, as mentioned before in the introduction of this master thesis, the root certificate of CAcert isn't included in the biggest browsers.

By combining OpenID and CAcert we might solve some of the problems and take advantage the strong points of both. Our idea is to improve security of the IdP by making use of CAcert client certificates. Concretely this means that a user has to logon with his CAcert client certificate on the IdP.

The advantage of this is that the CAcert client certificates are used for another purpose. The CAcert client certificates are now mainly used for e-mail. So using CAcert client certificates for another purpose could result in some publicity which could eventually result in brand awareness. The advantage of OpenID is that the authentication mechanism for the IdP is more secure. CAcert client certificates are based on a WoT, the IdP and RP can indirectly trust that the user is who he claims to be.

We have developed a proof of concept to ascertain whether it was possible to develop an OpenID provider where you can sign-in with your CAcert client certificate. We have done this by first searching for standard OpenID implementation packages. Then we selected one of the OpenID implementation packages which we figured was best for our proof of concept. Our criteria for a good OpenID implementation package were:

- Easy to install
- Complete package
- Supports OpenID 2.0
- Easy to edit the code

After we selected the OpenID implementation package and installed it we browsed the Internet for ways to allow sign-in with a X.509 certificate. We found some example scripts and other information of how it can be done. After we had found the information, we started to edit files and add scripts to the installed package, to make it possible to sign in with a CAcert client certificate. We used trial and error to develop the installed package, until it worked. For storing all the user data we created a database and implemented queries in the installed package.

The goals of making a proof of concept are:

- Find out if it is possible to create the idea (show that it is possible)
- See what the advantages and disadvantages are
- Notice which problems can occur
- Find out how much effort it costs to create it
- Test if the idea works as you thought it would work

We had to setup a web server that allows login with an SSL client certificate; the description of how we did it can be found in [Attachment A](#).

6.2 Design Decisions

In this paragraph we will describe the design decisions that we have made during the development of the proof of concept. First the design decisions of the database will be described. After that the design decisions of the script will be described.

6.2.1 Database

The email address of a user is the primary key in the user table

We have chosen for the email address to be the primary key because the email address is embedded in the client certificate. The email address is used to verify the identity of the user when the user signs in with his certificate. This means that the email address in the client certificate is checked if it matches the email address in the database of that particular user.

Nickname in the user table must be unique

The nickname is unique because if a user wants to login with username and password you need to use the nickname and password. If the nickname isn't unique it cannot be used as the identity. The nickname is used in the identifier (<http://onno.uni.cc/~nickname>).

Full name in the user table is not unique

The full name of a user isn't unique because there is a possibility that another user has the same full name.

Timezone, language, country, date of birth, gender and postcode are optional values

The values of these objects are optional in the OpenID SREG extension (section 3.1.1.7). This is why we have made them optional.

Timezones, languages and countries all have their own table in the database.

The information in the tables of timezone, language and country are static. Because of the amount of all the values it is useful to put them in the database. It reduces the amount of codes necessary to have, users being able to select their country, timezone or language from hundreds of lines of code to several lines of code. We could loop through all the languages timezones and countries as opposed to having long static lists in our template like:

```
<select name="country" id="edit-country">
  <option value=""></option>
  <option value="AF">Afghanistan</option>
  <option value="AX">Aland Islands</option>
  <option value="AL">Albania</option>
  ...
  etc.
</select>
```

The timezone table contains a countrynumber column

Numerous countries have more than just one timezone. Some countries have more than twenty timezones. We added numbers to these rows to be able to select one of these timezones after a user has selected a country.

The country code does not match the language codes in the country and language tables.

The codes we used are not just random; we took them from the ISO639 and ISO3166 standards. These standards are used by the OpenID SREG extension (section 3.1.1.7) specifically the two letter codes.

6.2.2 Script

Only CAcert client certificates are allowed.

The CAcert Web of Trust ensures that the name in a client certificate is accurate. If the same can be said about other CA they can be added in the server (the IdP) settings. Another reason for only accepting CAcert client certificates is that it was part of the formulated thesis assignment.

Only class 1 client certificates are accepted.

At the moment we decided that for the proof of concept only class 1 certificates needed to be accepted. This can be changed in the server settings. The script should not need to be altered to allow class 3 client certificates.

Only client certificates with a full name in them are allowed.

Only users that have 50 or more assurance points can add their name to a client certificate. The name in the client certificate has been verified by a Web of Trust. Therefore the name in the client certificate adds something to this IdP. The name in the client certificate can safely be considered to be true.

Only client certificates with one or more emailaddresses in them are allowed.

An emailaddress is the only information in the client certificate that is unique. Without one we would be unable to tell apart individuals with the same name. The nickname is unique as well but is not part of the client certificate.

The script checks whether a certificate is meant to be used for logging on to a website.

If it was not the intention of the user to use the generated certificate to use it to log on to a website, why would we allow it to be used in such a way?

Some functions have names that no longer suit them.

We decided to leave function names as they were such that if a new version of SimpleID were to be released, it would be easier to update our proof of concept. It also makes it easier to compare our script to the original.

After a country has been selected the script automatically selects a timezone and language.

This is a usability feature that should make it a bit easier for a user. It is not perfect due to mismatching country and language codes. For instance when someone picks the American Virgin Islands as a country the language Vietnamese (VI) is selected. This is perhaps a feature that should be improved at a later time.

6.3 Package selection

Before we started developing a proof of concept we first searched the web to find standard OpenID packages. These are packages with OpenID implementations with almost the same functionality. We quickly found a few of them and after examining them selected one to be used.

6.3.1 OpenID Implementations

There are numerous packages which can provide you with the OpenID functionality. In this paragraph we will look at three of them (Php-openid, SimpleID and PhpMyID) and asses whether they are fit to be used in our proof of concept. We have tested these packages to a certain extent. We also describe how these packages can be installed and what you should look at. We only describe how it can be installed on Linux because we used a Linux server.

6.3.1.1 Php-openid

Php-openid is a library that can be used on a Linux/ Unix server that runs Apache as well as PHP. Php-openid supports OpenID protocol version 1.1 and 2.0. This library is used by many

OpenID plugins, for instance the one for Wikipedia. Installing this library takes some effort as opposed to the packages (SimpleID and PhpMyID) which were merely copy and paste work. These additional packages are needed to be installed for this library to work:

- PEAR DB
- Crypt_HMAC2
- Crypt_DiffieHellman
- Services_YADIS

6.3.1.1.1 Installation

You can install these packages by executing the following lines of code in the terminal:

```
-----  
sudo pear install DB-1.7.14RC1  
sudo pear install Crypt_HMAC2-1.0.0  
sudo pear install Crypt_DiffieHellman-0.2.3  
sudo pear install Services_Yadis-0.4.0  
-----
```

Download the latest version of the Php-openid library from <http://www.openidenabled.com/>. In this compressed file you will find a readme as well various directories and documentation. To get started you should move the Auth directory of the installation to one of the directories in your include path. If you don't know where that is, use the phpinfo function. In our linux system these directories are /usr/share/php and /usr/share/pear. The Auth directory is placed in the include path so it can not be reached from the outside. In other words, files in the include directory are in a secure area. Note that it should still work if the Auth directory is placed in the web root of the IdP. If you want to look at the examples that came with the library you can copy them to your document root.

The Php-openid library comes with some very limited examples of both an IdP and an RP. Using this package in our proof of concept would have meant having to spend a lot of time on getting basic functions to work.

6.3.1.2 SimpleID

SimpleID is an IdP package written in PHP and, as the name says, it is simple. SimpleID can be used on any platform. On the server where you install the package Apache and PHP should be running. SimpleID supports OpenID protocol version 1.1 and 2.0. Files of this package are limited in size. Unlike PhpMyID, these files do not contain 2000 lines of code but contain mostly just a few hundred lines.

6.3.1.2.1 Installation

To install SimpleID download the latest version from sourceforge.net. Extract the compressed archive. Move the www directory from the archive to your web root on your server. Move the cache and identities directories to a secure folder that is accessible by the webserver. On our linux system these are include directories namely /usr/share/pear and /usr/share/php. But you can find these directories with the use of the php function phpinfo. Make sure the cache directory is writable by the server and identities directory readable. You could leave these directories in the web root while developing but should realize that it would create security issues. Rename the www directory in your web root if you like as it will be your OpenID provider endpoint. Now to create your own identity you will need to alter the identity file that

came with the package. The file `example.identity.dist` can be found in the `identities` directory. Rename the file to match your username and remove the `.dist` (`username.identity`). Open the identity file and alter the identity and pass. The identity must match the username in the filename. The pass is an MD5 hash of your password. Optionally you can alter the OpenID SREG information in the file. You will now need to create an html file which will point to the OpenID endpoint and will tell your identity. This file will need to look like this:

```
<html>
<head>
<link rel="openid2.provider" href="http://onno.uni.cc/id/" />
<link rel="openid2.local_id" href="username" />
</head>
<body></body>
</html>
```

Now you will be able to logon with this OpenID. In this example the file is located in the web root and is named `index.html`. So your identity URL is now <http://onno.uni.cc>.

6.3.1.3 PhpMyID/ PhpMyOpenID

PhpMyID is a lot like SimpleID, but it lacks support for version 2.0 of the OpenID protocol. It is also easy to set up and platform independent. PhpMyID is less easy to modify due to the way it is programmed. Some files of PhpMyID contain up to several thousand lines of code. With PhpMyID it is easy to set up an IdP. PhpMyOpenID is based on phpMyID. The difference between them is that phpMyOpenID contains a script for generating an identity whilst you have to alter a textfile for phpMyID. They share all the other advantages and disadvantages.

There is one big disadvantage to using this application and that is due to it using version 1.1 of the OpenID protocol. The other disadvantage of using PhpMyID is the way it is coded. Most of the code is to be found in single file which doesn't make it any more legible.

6.3.1.3.1 Installation

You'll need a webserver with php installed, but it is not necessary to have it running on a specific platform or webserver. Basically all that is needed is copying the files to the root of your webserver. Then for your identity you will need to edit the identity file with a text editor. After you've completed this, your IdP is ready for use and you can use it to log on to any RP.

6.3.2 Chosen package

It will not come as a surprise to find that we selected SimpleID as the basis for our proof of concept. It is easier to modify than PhpMyID, supports OpenID 2.0 and is complete enough, so we could get started quickly. In our opinion it was the best candidate (based on the criteria we had drawn up) to base our proof of concept on.

6.4 CAcertID

We have called our proof of concept CAcertID because it's a combination of OpenID with CAcert. With CAcertID it is possible to log in with a CAcert client certificate on the IdP (CAcertID). Underneath we will describe what we have changed in SimpleID to make it work.

We have modified several files of SimpleID to make it possible to sign in with a CAcert client certificate. The modifications to the SimpleID files are described in [Attachment C](#). We have created a script (server side) that makes it possible to sign in with a CAcert client certificate. We have also created a dynamic identity page which means that every user will have an identity page without the need to manually create one. Normally you should make an identity page for every user. The dynamic identity page is located in the CAcertID root directory (on the IdP server) and contains the following code:

```
<html>
<head>
<link rel="openid2.provider" href="http://onno.uni.cc/id/" />
<link rel="openid2.local_id" href="<?php echo $_GET["user"];?>" />
</head>
<body></body>
</html>
```

File: idpage.php

Whenever the identity page is requested, the “user” variable (<?php echo \$_GET["user"];?>) will be the requested OpenID identity. So for example the result for Barthold would be:

```
<link rel="openid2.local_id" href="Barthold" />
```

In the .htaccess file that can be found or needs to be created in the web root (of the IdP) we add the following lines:

```
RewriteEngine on
RewriteRule ^\~([^\/]*)$ /id/idpage.php?user=$1 [L]
```

File: .htaccess

This rewrite rule will forward anyone that requests a page similar to <http://example.com/~username> to <http://example.com/id/idpage.php?user=username>. In our case this would mean that Barthold would need to use <http://onno.uni.cc/~Barthold> as his OpenID identifier.

For storing the user credentials we have created a database. This database will be described in the next section.

6.4.1 Credential Database

As described above we have created a database to store the credentials of the user. We have created a database because otherwise all the user data will be stored in separate files. With a database the user data is more manageable. All the data is being used by scripts and by the user himself. We have created four tables: user, timezone, country and language. The table user is created to store the data when a user creates an account on the IdP. The following user data will be stored: Username, Password, E-mail, Fullname, Date of birth (optional), Gender

(optional), Postcode (optional), Country (optional), Language (optional) and Timezone (optional). The other three tables, timezone, country and language, have been created to use predefined values. This is useful because some people, for example, write their country's name with capital letters or in their native tongue. The advantage of this is that the values of the same timezone, country and language are the same. More information about the database is described in Attachment B: Database

6.4.2 Background

In this section we will shortly describe what happens in the background when you sign in with a CAcert client certificate. The first thing that is done is that the client certificate is being parsed and checked if it's a CAcert client certificate. This is done by checking some values (Attachment C. Proof of Concept Code) of the client certificate. The next step is that the email address that is included in the client certificate is being grabbed out of it. Then the email address of the client certificate is used to look for a match with an email address in the database. If there is a match, the user will be successfully logged on.

6.5 Results

Our goal was to create an IdP that allows the users to sign in with their CAcert client certificate. We have reached our goal. We have modified SimpleID so that it is possible to sign in with a CAcert client certificate. You can see a screenshot of what it looks like, if you are signing in with a certificate in figure 18.

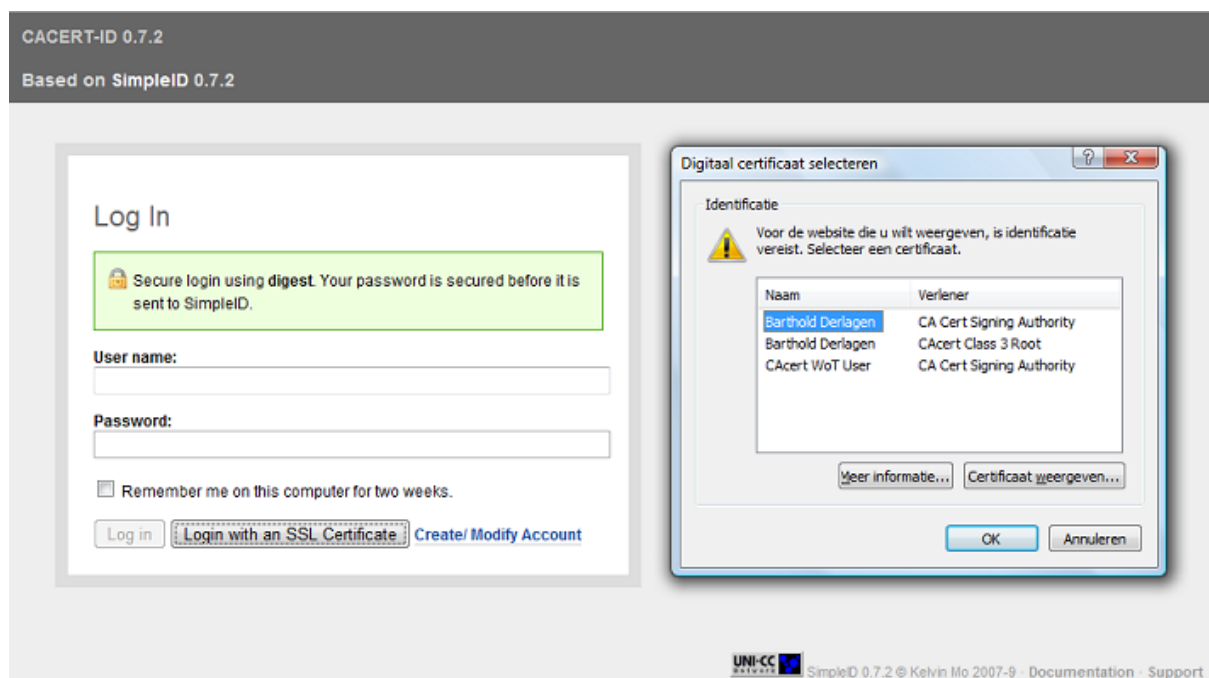


Figure 18: Signing in with a certificate.

6.5.1 Advantages

- Users can authenticate themselves with something they have as opposed to something they know.
- RP will receive a full name and email address that have been validated by a Web of Trust
- CAcert certificates have gained another purpose
- The trust problems between the involved parties could improve because of the Web of Trust.

6.5.2 Disadvantages

- Assurance level (section 2.1) remains low. The reason for this is: that using a CAcert client certificate to sign in doesn't solve the OpenID security problems. If only https were used, most security problems of OpenID would be solved. A solution for this could be to develop an IdP that only allows connections from RP through https.

6.5.3 Notes

- The server running the OpenID provider will require a server certificate that is signed by a trusted party. This is needed so that the OpenID provider can setup secure connections with the users.
- The full name of a user only matters when the RP requests the full name of the IdP

6.5.4 Future work

- Documentation: There needs to be documentation for the users to know how they can use this product.
- Promotion: This solution needs to be promoted so that people are going to use it. Promotion can be done by placing articles on community websites and in magazines.
- HTTPS: For security reasons only HTTPS should be used for communication not HTTP. (section 3.2)
- Law: The credentials of the users are being stored in a database. When you store user credentials then the law specifies some rules. In the Netherlands this law is called: Wet bescherming persoonsgegevens. Every country has its own laws for this. If this product is going to be used, the laws that are relevant need to be looked at.
- Protocol version: For security reasons only OpenID protocol version 2.0 should be used. This version is more secure than older versions (section 3.2).

7 Conclusion

"How to use the CAcert infrastructure within an OpenID context?"

The CAcert infrastructure can be used as an authentication mechanism for the OpenID identity provider (IdP). Concretely this means that a user will be able to authenticate himself with his CAcert client certificate. The CAcert infrastructure adds a validated name and email address to the OpenID identity. We have created a proof of concept to prove that it is possible to use this authentication mechanism. We accomplished this by modifying a standard OpenID package.

7.1 OpenID

"How does OpenID work?"

OpenID (chapter 3) is based on Single Sign On (SSO) (section 2.3.1). Concretely this means that a user can authenticate and authorize himself with one identity to all participating systems to which the user has access. There are three parties within OpenID: a user, Relaying Party (RP) and Identity Provider (IdP). The IdP is the authentication server within OpenID. The IdP provides the user with an OpenID identifier. The RP is a website that requires users to provide an OpenID identifier and uses OpenID as a method to authenticate users. An RP will contact the IdP to check whether the identifier that the user provides is really his identifier. In the current version of OpenID there are some security problems. These attacks are possible man-in-the-middle attack, phishing and clickjacking. There are also some trust problems between the different parties.

We compared OpenID with other SSO mechanisms like Kerberos, DigiD, Microsoft Windows LiveID and Google federated login (chapter 4). Windows LiveID and Google Federated Login are most similar to OpenID. Google Federated Login is based on OpenID and Windows LiveID has a similar purpose as OpenID. The roles that are played in Kerberos, Microsoft Windows LiveID and Google federated login are similar to the roles in OpenID. Microsoft Windows LiveID, Google Federated Login and OpenID have a low assurance level, the others have a higher assurance level because most of them have better or more security measures. OpenID has not specified which authentication mechanism is used for the user to authenticate himself. The other SSO mechanisms have specified which authentication mechanism they use. DigiD, Microsoft Windows LiveID and Google federated login are proprietary solutions, Kerberos is a standard and OpenID is an open framework.

7.2 CAcert

"How does CAcert work?"

CAcert (chapter 5) is a non-profit association that promotes the awareness and education on computer security through the use of encryption, specifically with the X.509 family of standards. CAcert gives users the option to create x.509 client certificates on the basis of 'web of trust' for free. CAcert signs server and client certificates and generates client certificates for users that request them. Web of trust means that people confirm that you are the person that you say you are. CAcert uses a network of assurers to assure the identity of new users. A user will need to print a verification paper. This paper has to be filled in by an assurer after he



verified your identity by your identification papers (passport, driver's license, etc.). An assurer can grant a user 10-35 points depending on his rank. Once a user has 50 points he can be considered assured. When a user gets 100 points he can take a test to become an assurer.

7.3 Proof of Concept

"How can OpenID and CAcert complement each other?"

OpenID does not specify how a user should be authenticated. CAcert on the other hand issues client certificates that can be used by users to authenticate themselves. CAcert client certificates can be used by an IdP to authenticate users. We developed a proof of concept (chapter 6) to test this hypothesis.

To be able to develop a proof of concept we set to work by setting up a webserver. We installed Linux on the server with the following packages installed: Apache 2, MySQL and PHP 5. We then searched for standard OpenID packages that we could modify to become our proof of concept CAcertID. We found several standard OpenID packages which we looked at and tested. We chose the package SimpleID and installed it on the server. We modified the package to make it possible to sign in with a CAcert client certificate.

7.4 Future Work

Future work may include any project with research questions similar to:

- How can CAcertID become more beneficial to users?
- How should the security issues in OpenID be addressed?
- etc.

Bibliography

Scientific literature

- [BES06] Bessie, C. Hu; Duncan, S. Wong; Zhenfeng, Zhang; Xiaotie, Deng, Certificateless signature: a new security model and an improved generic Construction, Paper, Springer Science and Business Media, 2006
- [GAN91] Gangemi G.T., Computer Security Basics, Sebastopol (California, USA), O'Reilly & Associates, ISBN: 0937175714, 2010
- [HEL09] Helenius, Kari, OpenID and identity management in consumer services on the Internet, Paper, Helsinki University of Technology, 2009
- [JOC07] Jochems, Marc, DigiD and Privacy, Master Thesis, Radboud University Nijmegen, 2007
- [JOO99] Joon, S. Park; Ravi, Sandhu, Smart Certificates: Extending X.509 for Secure Attribute Services on the Web, paper, The Laboratory for Information Security Technology Information and Software Engineering Department George Mason University, 1999
- [KAU02] Charlie Kaufman, Radia Perlman, and Mike Spencer, Network Security Private Communication in a Public World, Second Edition ed. New Jersey, USA: Prentice Hall, 2002, ISBN: 0-13-046019-2
- [LAN03] Lancaster, Sean; Yen, C. David; Huang, Shi-Ming, Public key infrastructure: a micro and macro analysis, Paper, Elsevier Science, 2003
- [LEE08] Lee, HwanJin; Jeun, InKyung; Chun, Kilsoo; Song, Junghwan, A New Anti-phishing Method in OpenID, Proceedings of the second International Conference on Emerging Security Information, Systems and Technologies (Pages 243-247), 2008
- [LIN09] Lindholm, Alexander, Security Evaluation of the OpenID Protocol, Thesis Paper, School of Computer Science and Engineering - Royal Institute of Technology, 2009.
- [MCD08] McDonald, Tony, Facilitating online integrity using OpenID, Proceedings ascilite Melbourne 2008, Faculty of Medical Sciences - The Medical School - Newcastle University - United Kingdom, 2008
- [MOS09] Mostarda, Michele; Palmisano, Davide; Zani, Federico; Tripoldi, Simone, Towards an OpenID-based solution to the Social Network Interoperability problem, Position paper for the W3C Workshop on the Future of Social Networking, 2009
- [MON08] Moniava, Giorgi, Extending DigiD to the Private Sector (DigiD-2), Master Thesis, Technische Universiteit Eindhoven, 2008
- [MYL06] Myllyniemi, Annu, Identity Management Systems A Comparison of Current Solutions, Paper, Helsinki University of Technology, 2006
- [NIS01] NIST (National Institute of Standards and Technology), Introduction to Public Key Technology and the Federal PKI Infrastructure, SP 800-32, U.S. Government publication, 2001
- [OH08] Oh, Hyun-Kyung; Jin, Seung-Hun, The Security Limitations of SSO in OpenID, Research Paper, Information Security Engineering, Korea University of Science and Technology(KUST); Digital ID Security Research Team - Electronics and Telecommunication Research Institute(ETRI), 2008
- [TAN03] Tanenbaum, Andrew S., Computer Networks 4th Edition, Upper Saddle River (New Jersey, USA), Pearson Education Inc., ISBN: 0-13-066102-3, 2003
- [TSY07] Tsyklevich, Eugene; Tsyklevich, Vlad, Single Sign-On for the Internet: A Security Story, Whitepaper, BlackHat USA - Las Vegas, 2007

- [VOL01] Volchkov, A., Revisiting single sign-on: A pragmatic approach in a new context, IT Professional, Volume 3, Issue 1, Pages 39–45, 2001
- [WEI01] Weise, Joele, Public Key Infrastructure Overview, Sun BluePrints™ OnLine, 2001
- [ZHA03] Zhang, Xiaolan, A Comprehensive Study on Kerberos, Paper, University of Illinois at Urbana-Champaign, 2003

Other sources

- [CACE10a] CAcert, CAcert, 2010 (<http://www.cacert.org>) [accessed February 2nd 2010]
- [CACE10b] CAcert, CAcert wiki, 2010 (<http://wiki.cacert.org>) [accessed February 2nd 2010]
- [ELLI10] Ellin, Brian, OpenID Technology Summit 2010; Challenges faced implementing OpenID in RPX (<http://wiki.openid.net/f/Janrain+-+RPX-OpenID-Tech-Summit-2010.pdf>) [accessed June 25th 2010]
- [GOOG10] Google, Federated Login for Google Account Users, 2010 (<http://code.google.com/intl/nl/apis/accounts/docs/OpenID.html>) [accessed May 23th 2010]
- [KOHL93] Kohl, J. (Digital Equipment Corporation); Neuman, C. (ISI), RFC1510 - The Kerberos Network Authentication Service (V5), 1993 (<http://www.ietf.org/rfc/rfc1510.txt>) [accessed July 12th 2010]
- [MICR08] Microsoft, Windows Live ID Becomes an OpenID Provider, October 2008, (<http://winliveid.spaces.live.com/Blog/cns!AEE1BB0D86E23AAC!1791.entry?wa=wsignin1.0&sa=159627916>) [accessed 19 April 2010]
- [MICR09] Microsoft, Windows Live ID OpenID CTP Status Update, August 2009, (<http://winliveid.spaces.live.com/Blog/cns!AEE1BB0D86E23AAC!1791.entry>) [accessed 19 April 2010]
- [MOZI07] Mozilla, Bugtracker - CAcert root cert inclusion into browser, 2010 (https://bugzilla.mozilla.org/show_bug.cgi?id=215243) [accessed February 1st 2010]
- [NEUM05] Neuman, C. (USC-ISI); Yu, T.; Hartman, S.; Raeburn, K. (MIT), RFC4120 - The Kerberos Network Authentication Service (V5), 2005 (<http://tools.ietf.org/html/rfc4120>) [accessed July 12th 2010]
- [OPEN10a] OpenID, OpenID (<http://openid.net>) [accessed February 5th 2010]
- [OPEN10c] OpenID, OpenID authentication 2.0 specifications (http://openid.net/specs/openid-authentication-2_0.html) [accessed March 9th 2010]
- [OAUT10] OAuth, OAuth documentation (<http://oauth.net/>) [accessed June 21st 2010]
- [SMIT10] Smittii.com, How to generate OpenSSL keys for Apache for Windows (<http://smithii.com/node/117>) [accessed April 29th 2010]
- [WIKI10] Wikimedia, Extensible Resource Identifier, 2010 (http://en.wikipedia.org/wiki/Extensible_Resource_Identifier) [accessed March 23rd 2010]
- [YADI06] Yadis, Specifications 1.0, 2006 (http://yadis.org/wiki/Yadis_1.0_%28HTML%29) [accessed March 26th 2010]

Attachment A: Setting up a Webserver with SSL enabled

In this attachment we describe how to set up a server that is capable of running as an OpenID provider. In the first paragraph we explain how to set up a webserver and in the second paragraph we describe how you can enable SSL on your webserver.

Setting up a Webserver

When running a webserver you can pick either a Windows based Operating System or a Linux/ Unix based one, the latter being freely available whilst the first requires a licence. We'd recommend using the Apache 2 webserver on either system.

Linux

On a Linux distribution that is based on the Debian distribution like Ubuntu you can use the 'apt-get install' command from the terminal to install applications. You can either launch the terminal from applications/ accessories/ Terminal or by pressing Ctrl + Alt + F1/ F2/ F3/ F4/ F5/ F6. In the latter case you can return to your desktop by pressing Ctrl + Alt + F7. In the terminal you will see the following line 'User@servername:~\$'. You can type after this line. To install Apache 2.x type:

```
sudo apt-get install apache2
```

The 'sudo' means that you are executing the script with elevated permissions. You will be asked for logon credentials, if you have not supplied them already. When you have logged on, Apache 2.x will be installed. This seems like a lot of work but most packages can be installed with this method.

We should proceed by installing and configuring PHP. Open the terminal and execute the following lines:

```
sudo apt-get install php5  
sudo apt-get install libapache2-mod-php5
```

You've now installed PHP 5 and configured it to work with Apache 2. The changes will take effect after you have restarted Apache. You can restart Apache by entering the following line in the terminal:

```
sudo /etc/init.d/apache2 restart
```

This will restart Apache and notify you of any startup errors, warnings and notices. If you are considering storing data in an SQL database, it could prove to be wise to install MySQL and configure PHP to work with mysql. This can easily be achieved by executing the following commands in the terminal.


```
sudo apt-get install mysql-server
sudo apt-get install php5-mysql
sudo /etc/init.d/apache2 restart
```

We have now installed MySQL and configured PHP to inquire data from it. Due to restarting the server all these changes have now been applied. You could also install the MySQL Workbench 5.2 despite it being beta at the time of writing. Version 5.1 also exists but it lacks a lot of functionality. MySQL Workbench is a graphical interface that can be used to administrate a MySQL server. If you want to install this application run the following line in the terminal.

```
sudo apt-get install mysql-workbench-oss
```

This should install the latest stable version of the application. Alternatively you could download a version from the MySQL website.

Windows

On a Windows system you can visit <http://httpd.apache.org> to download the latest version. This is a binary which you can execute. If you go through all the steps it will successfully install Apache on your server.

Now that Apache has been installed we should proceed to installing and configuring PHP. Visit <http://www.php.net/downloads.php> to download one of the newest versions of PHP. You could try to use the binary installer but we'd recommend downloading the zip and configure it yourself. Make sure the version you download supports OpenSSL. Most versions have native support for OpenSSL but some of the older versions lack it. The zip file comes with documentation on how to configure PHP to work with your webserver. If you follow the documentation it should not prove to be hard to get PHP to work.

You may also want to install MySQL for data storage. Point your browser at <http://www.mysql.com/downloads/>. Download the latest MySQL Community Server and optionally the MySQL Workbench (version 5.2 or later). Execute the installer to install MySQL. You will then need to have another look at the php.ini document that is loaded by Apache. If you are not sure where this file is, create a php file with the following contents:

```
<?php phpinfo() ?>
```

Place this file within your webroot and use your browser to browse to its location. On the page shown to you the value of 'Loaded Configuration File' is the file you need to modify to get PHP to work with MySQL. If you are unsure what to modify, take another peek at the documentation that came with PHP.

Enabling SSL

To be able to check client certificates the connection must take place over a secure connection. The user will then be able to access the page through <https://domain.extension> instead of <http://domain.extension>. It is still possible to access the site with http. This paragraph describes how you can get your website to use a server certificate and accept class 1 client certificates. To enable SSL you must generate a certificate. This certificate can then be signed by a CA like CAcert. This has been described in the CAcert chapter.

Linux

Download the class 1 and class 3 root certificates from CAcert and name them cacert1.pem and cacert3.pem. Copy your certificate and key as well as the root certificates of CAcert to /etc/apache2. Browse to the directory /etc/apache2/sites-enabled and open default-ssl. As opposed to the configuration on a windows platform this file already contains a great deal of ssl related settings. The file is also very well documented and you might want to put the lines of code with their corresponding documentation. Set the ServerAdmin to your email address and docroot to /var/www. Make sure the 'Directory /var/www' brackets consist of at least the following lines:

```
Options Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all
SSLOptions +StdEnvVars
SSLOptions +ExportCertData
```

Make sure the following lines are contained within the 'VirtualHost _default_:443' brackets:

```
SSLVerifyClient require
SSLVerifyDepth 1
SSLCertificateFile /etc/apache2/cacert1.pem
SSLCertificateChainFile /etc/apache2/cacert3.pem
SSLCertificateFile /etc/apache2/server.crt
SSLCertificateKeyFile /etc/apache2/server.key
```

Your file should be close to this:

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
    ServerAdmin username@domain.com

    DocumentRoot /var/www
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>

    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>

    ErrorLog /var/log/apache2/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
```

```
# alert, emerg.
LogLevel warn

CustomLog /var/log/apache2/ssl_access.log combined

Alias /doc/ "/usr/share/doc/"
<Directory "/usr/share/doc/">
    Options Indexes MultiViews FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
    Allow from 127.0.0.0/255.0.0.0 ::1/128
</Directory>

#    SSL Engine Switch:
#    Enable/Disable SSL for this virtual host.
SSLEngine on

#    A self-signed (snakeoil) certificate can be created by installing
#    the ssl-cert package. See
#    /usr/share/doc/apache2.2-common/README.Debian.gz for more info.
#    If both key and certificate are stored in the same file, only the
#    SSLCertificateFile directive is needed.
SSLCertificateFile /etc/apache2/server.crt
SSLCertificateKeyFile /etc/apache2/server.key

#    Server Certificate Chain:
#    Point SSLCertificateChainFile at a file containing the
#    concatenation of PEM encoded CA certificates which form the
#    certificate chain for the server certificate. Alternatively
#    the referenced file can be the same as SSLCertificateFile
#    when the CA certificates are directly appended to the server
#    certificate for convinience.
SSLCertificateChainFile /etc/apache2/cacert3.pem

#    Certificate Authority (CA):
#    Set the CA certificate verification path where to find CA
#    certificates for client authentication or alternatively one
#    huge file containing all of them (file must be PEM encoded)
#    Note: Inside SSLCACertificatePath you need hash symlinks
#           to point to the certificate files. Use the provided
#           Makefile to update the hash symlinks after changes.
#SSLCACertificatePath /etc/ssl/certs/
SSLCACertificateFile /etc/apache2/cacert1.pem

#    Certificate Revocation Lists (CRL):
#    Set the CA revocation path where to find CA CRLs for client
#    authentication or alternatively one huge file containing all
#    of them (file must be PEM encoded)
#    Note: Inside SSLCAREvocationPath you need hash symlinks
#           to point to the certificate files. Use the provided
#           Makefile to update the hash symlinks after changes.
#SSLCAREvocationPath /etc/apache2/ssl.crl/
#SSLCAREvocationFile /etc/apache2/ssl.crl/ca-bundle.crl

#    Client Authentication (Type):
#    Client certificate verification type and depth. Types are
#    none, optional, require and optional_no_ca. Depth is a
#    number which specifies how deeply to verify the certificate
#    issuer chain before deciding the certificate is not valid.
SSLVerifyClient require
SSLVerifyDepth 1
```

```

# Access Control:
# With SSLRequire you can do per-directory access control based
# on arbitrary complex boolean expressions containing server
# variable checks and other lookup directives. The syntax is a
# mixture between C and Perl. See the mod_ssl documentation
# for more details.
#<Location />
#SSLRequire (    %{SSL_CIPHER} !~ m/^(EXP|NULL)/ \
#    and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \
#    and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"} \
#    and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
#    and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20
#    or %{REMOTE_ADDR} =~ m/^192\.76\.162\. [0-9]+$/
#</Location>

# SSL Engine Options:
# Set various options for the SSL engine.
# o FakeBasicAuth:
# Translate the client X.509 into a Basic Authorisation. This
means that
# the standard Auth/DBMAuth methods can be used for access control.
The
# user name is the `one line' version of the client's X.509
certificate.
# Note that no password is obtained from the user. Every entry in
the user
# file needs this password: `xxj31ZMTZzkVA'.
# o ExportCertData:
# This exports two additional environment variables:
SSL_CLIENT_CERT and
# SSL_SERVER_CERT. These contain the PEM-encoded certificates of
the
# server (always existing) and the client (only existing when
client
# authentication is used). This can be used to import the
certificates
# into CGI scripts.
# o StdEnvVars:
# This exports the standard SSL/TLS related `SSL_*' environment
variables.
# Per default this exportation is switched off for performance
reasons,
# because the extraction step is an expensive operation and is
usually
# useless for serving static content. So one usually enables the
exportation for CGI and SSI requests only.
# o StrictRequire:
# This denies access when "SSLRequireSSL" or "SSLRequire" applied
even
# under a "Satisfy any" situation, i.e. when it applies access is
denied
# and no other module can change it.
# o OptRenegotiate:
# This enables optimized SSL connection renegotiation handling when
SSL
# directives are used in per-directory context.
#SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
<FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
    SSLOptions +ExportCertData
</FilesMatch>

```

```

<Directory /usr/lib/cgi-bin>
    SSLOptions +StdEnvVars
    SSLOptions +ExportCertData
</Directory>

#   SSL Protocol Adjustments:
#   The safe and default but still SSL/TLS standard compliant shutdown
#   approach is that mod_ssl sends the close notify alert but doesn't
wait for
#   the close notify alert from client. When you need a different
shutdown
#   approach you can use one of the following variables:
#   o ssl-unclean-shutdown:
#   This forces an unclean shutdown when the connection is closed,
i.e. no
#   SSL close notify alert is send or allowed to received. This
violates
#   the SSL/TLS standard but is needed for some brain-dead browsers.
Use
#   this when you receive I/O errors because of the standard approach
where
#   mod_ssl sends the close notify alert.
#   o ssl-accurate-shutdown:
#   This forces an accurate shutdown when the connection is closed,
i.e. a
#   SSL close notify alert is send and mod_ssl waits for the close
notify
#   alert of the client. This is 100% SSL/TLS standard compliant, but
in
#   practice often causes hanging connections with brain-dead
browsers. Use
#   this only for browsers where you know that their SSL
implementation
#   works correctly.
#   Notice: Most problems of broken clients are also related to the
HTTP
#   keep-alive facility, so you usually additionally want to disable
#   keep-alive for those clients, too. Use variable "nokeepalive" for
this.
#   Similarly, one has to force some clients to use HTTP/1.0 to
workaround
#   their broken HTTP/1.1 implementation. Use variables "downgrade-
1.0" and
#   "force-response-1.0" for this.
BrowserMatch "MSIE [2-6]" \
    nokeepalive ssl-unclean-shutdown \
    downgrade-1.0 force-response-1.0
# MSIE 7 and newer should be able to use keepalive
BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
</VirtualHost>
</IfModule>

```

Save the the file. If you want to be able to rewrite requests (for example <http://example.com/news/104> -> <http://example.com/news.php?id=104>) you can enable a module to do so. Open /etc/apache2/httpd.conf and copy paste the following line into it:

```
LoadModule rewrite_module /usr/lib/apache2/modules/mod_rewrite.so
```

After you have restarted Apache you should now have Apache running with support for SSL client certificates.

Windows

You will need to change the Apache configuration so it will use your certificate. But first download the class 1 and class 3 root certificates of CAcert and save them as cacert1.pem and cacert3.pem in the Apache configuration directory. Open httpd.conf from the configuration directory in a text or hex editor. Look for the following code:

```
<IfModule ssl_module>
*
</IfModule>
```

The contents may seem different with each version of Apache but alter the contents to make them seem more similar to the following example. Take special note of the data between the VirtualHost brackets mentioning the keys (SSLCACertificateFile, SSLCertificateChainFile, SSLCertificateFile, SSLCertificateKeyFile).

```
<IfModule ssl_module>
  Listen 443
  NameVirtualHost *:443
  SSLRandomSeed startup builtin
  SSLRandomSeed connect builtin
  AddType application/x-x509-ca-cert .crt
  AddType application/x-pkcs7-crl .crl
  SSLPassPhraseDialog builtin
  SSLSessionCache "shmcb:C:/Program Files/Apache Software
Foundation/Apache2.2/logs/ssl_scache(512000)"
  SSLSessionCacheTimeout 300
  SSLMutex default
  SSLCACertificateFile "C:/Program Files/Apache Software
Foundation/Apache2.2/conf/cacert1.pem"
  SSLCertificateChainFile "C:/Program Files/Apache Software
Foundation/Apache2.2/conf/cacert3.pem"
  SSLCertificateFile "C:/Program Files/Apache Software
Foundation/Apache2.2/conf/server.crt"
  SSLCertificateKeyFile "C:/Program Files/Apache Software
Foundation/Apache2.2/conf/server.key"
  SSLCipherSuite
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

  BrowserMatch ".*MSIE.*" \
    nokeepalive ssl-unclean-shutdown \
    downgrade-1.0 force-response-1.0

  CustomLog "C:/Program Files/Apache Software
Foundation/Apache2.2/logs/ssl_request_log" \
    "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"

<VirtualHost *:443>
  SSLEngine on
  <FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars +ExportCertData
    SSLVerifyClient require
    SSLVerifyDepth
  </FilesMatch>
</VirtualHost>
</IfModule>
```



When you restart Apache it should now use the certificate you specified and request a client certificate, when someone accesses the server over https.

Attachment B: Database

In this attachment there will be a description of how we have set up the database structure behind the IdP (of the proof of concept). First there will be a description of which tables we have created and why. After that there will be a description of every table and the code will be displayed.

We have created four tables: user, timezone, country and language. The table user is created to store the data when a user creates an account on the IdP. The other three tables, timezone, country and language, have been created to use predefined values. This is useful because some people write for example their country name with capital letters or in their native language. The advantage of this is that the values of the same timezone, country and language are the same.

Table: user

The table user consists of the following columns:

- nickname (the user's nickname, the value of the field needs to be filled, the user's nickname needs to be unique)
- pass (the user's password, the value of the field needs to be filled)
- email (the user's e-mail address, the value of the field needs to be filled, this is the primary key of the table)
- fullname (this is the user's first and last name, the value of the field needs to be filled)
- dob (the user's date of birth, this does not have to be filled, default value is NULL)
- gender (the user's gender, this does not have to be filled, default value is NULL)
- postcode (the user's postcode, this does not have to be filled, default value is NULL)
- country (the country where the user lives, this does not have to be filled, default value is NULL, is a foreign key of the table)
- language (the language of the website that the user prefers, this does not have to be filled, default value is NULL, is a foreign key of the table)
- timezone (the timezone of where the user lives, this does not have to be filled, default value is NULL, is a foreign key of the table)

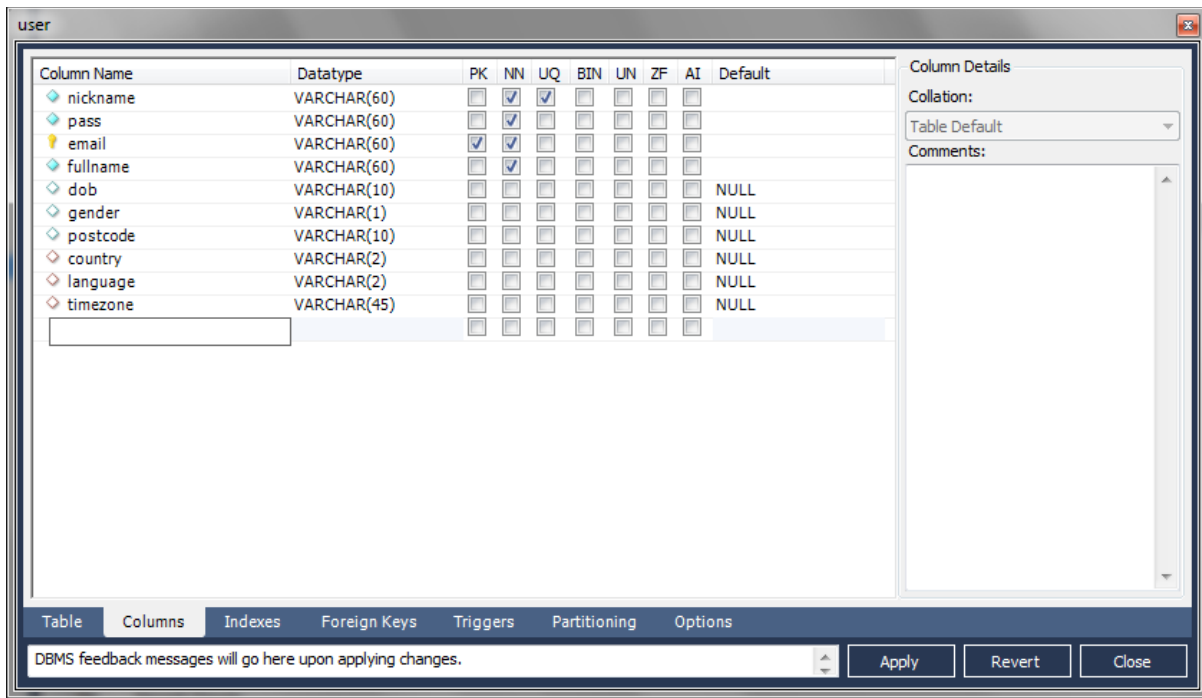


Figure B.1: Table user

Code:

```
CREATE TABLE `user` (
  `nickname` varchar(60) NOT NULL,
  `pass` varchar(60) NOT NULL,
  `email` varchar(60) NOT NULL,
  `fullname` varchar(60) NOT NULL,
  `dob` varchar(10) DEFAULT NULL,
  `gender` varchar(1) DEFAULT NULL,
  `postcode` varchar(10) DEFAULT NULL,
  `country` varchar(2) DEFAULT NULL,
  `language` varchar(2) DEFAULT NULL,
  `timezone` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`email`),
  UNIQUE KEY `nickname_UNIQUE` (`nickname`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Table: timezone

The table timezone consists of the following columns:

- country (the id of the country, the value of the field needs to be filled, this is a foreign key of the table)
- timezone (the options of all the timezones, this is the primary key of the table, the value of the field needs to be filled)
- countrynr (nr of the country, the value of the field needs to be filled, default value is '1')

Constraints:

- The foreign key 'country' references to 'id' in the country table.

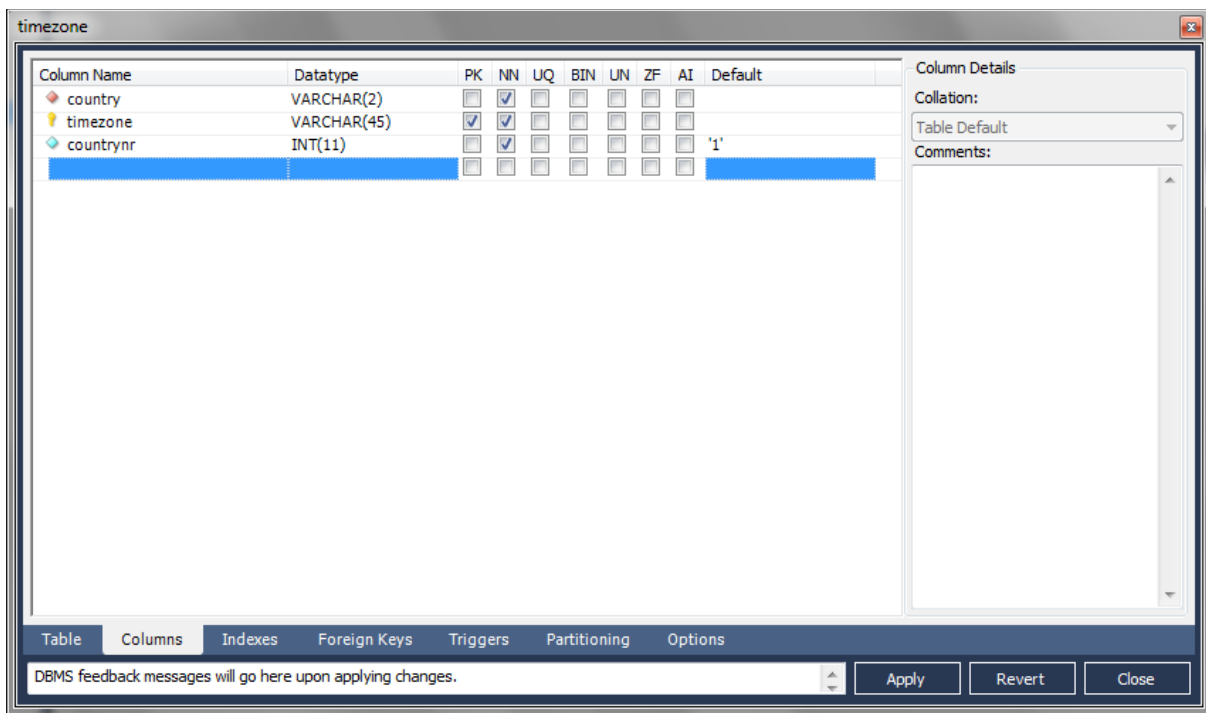


Figure B.2: Table timezone

Code:

```

CREATE TABLE `timezone` (
  `country` varchar(2) NOT NULL,
  `timezone` varchar(45) NOT NULL,
  `countrynr` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`timezone`),
  UNIQUE KEY `index3` (`country`,`countrynr`),
  CONSTRAINT `fk_timezone` FOREIGN KEY (`country`) REFERENCES `country`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `timezone` VALUES
('AD','Europe/Andorra',1), ('AE','Asia/Dubai',1), ('AF','Asia/Kabul',1), ('AG',
,'America/Antigua',1), ('AI','America/Anguilla',1), ('AL','Europe/Tirane',1),
('AM','Asia/Yerevan',1), ('AN','America/Curacao',1), ('AO','Africa/Luanda',1)
, ('AQ','Antarctica/Casey',1), ('AQ','Antarctica/Davis',2), ('AQ','Antarctica/
DumontDURville',3), ('AQ','Antarctica/Mawson',4), ('AQ','Antarctica/McMurdo',
5), ('AQ','Antarctica/Palmer',6), ('AQ','Antarctica/Rothera',7), ('AQ','Antarc
tica/South_Pole',8), ('AQ','Antarctica/Syowa',9), ('AQ','Antarctica/Vostok',1
0), ('AR','America/Argentina/Buenos_Aires',1), ('AR','America/Argentina/Catam
arca',2), ('AR','America/Argentina/Cordoba',3), ('AR','America/Argentina/Juju
y',4), ('AR','America/Argentina/La_Rioja',5), ('AR','America/Argentina/Mendoz
a',6), ('AR','America/Argentina/Rio_Gallegos',7), ('AR','America/Argentina/Sa
lta',8), ('AR','America/Argentina/San_Juan',9), ('AR','America/Argentina/San_
Luis',10), ('AR','America/Argentina/Tucuman',11), ('AR','America/Argentina/Us
huaia',12), ('AS','Pacific/Pago_Pago',1), ('AT','Europe/Vienna',1), ('AU','Aus
tralia/Adelaide',1), ('AU','Australia/Brisbane',2), ('AU','Australia/Broken_H
ill',3), ('AU','Australia/Currie',4), ('AU','Australia/Darwin',5), ('AU','Aust
ralia/Eucla',6), ('AU','Australia/Hobart',7), ('AU','Australia/Lindeman',8), (
'AU','Australia/Lord_Howe',9), ('AU','Australia/Melbourne',10), ('AU','Austra
lia/Perth',11), ('AU','Australia/Sydney',12), ('AW','America/Aruba',1), ('AX',
,'Europe/Mariehamn',1), ('AZ','Asia/Baku',1), ('BA','Europe/Sarajevo',1), ('BB',
,'America/Barbados',1), ('BD','Asia/Dhaka',1), ('BE','Europe/Brussels',1), ('B
F','Africa/Ouagadougou',1), ('BG','Europe/Sofia',1), ('BH','Asia/Bahrain',1),

```

('BI', 'Africa/Bujumbura', 1), ('BJ', 'Africa/Porto-
 Novo', 1), ('BL', 'America/St_Barthelemy', 1), ('BM', 'Atlantic/Bermuda', 1), ('BN',
 'Asia/Brunei', 1), ('BO', 'America/La_Paz', 1), ('BR', 'America/Araguaina', 1), ('
 BR', 'America/Bahia', 2), ('BR', 'America/Belem', 3), ('BR', 'America/Boa_Vista', 4
), ('BR', 'America/Campo_Grande', 5), ('BR', 'America/Cuiaba', 6), ('BR', 'America/
 Eirunepe', 7), ('BR', 'America/Fortaleza', 8), ('BR', 'America/Maceio', 9), ('BR', '
 America/Manaus', 10), ('BR', 'America/Noronha', 11), ('BR', 'America/Porto_Velho'
 , 12), ('BR', 'America/Recife', 13), ('BR', 'America/Rio_Branco', 14), ('BR', 'Ameri
 ca/Santarem', 15), ('BR', 'America/Sao_Paulo', 16), ('BS', 'America/Nassau', 1), ('
 BT', 'Asia/Thimphu', 1), ('BW', 'Africa/Gaborone', 1), ('BY', 'Europe/Minsk', 1), ('
 BZ', 'America/Belize', 1), ('CA', 'America/Atikokan', 1), ('CA', 'America/Blanc-
 Sablon', 2), ('CA', 'America/Cambridge_Bay', 3), ('CA', 'America/Dawson', 4), ('CA'
 , 'America/Dawson_Creek', 5), ('CA', 'America/Edmonton', 6), ('CA', 'America/Glace
 _Bay', 7), ('CA', 'America/Goose_Bay', 8), ('CA', 'America/Halifax', 9), ('CA', 'Ame
 rica/Inuvik', 10), ('CA', 'America/Iqaluit', 11), ('CA', 'America/Moncton', 12), ('
 CA', 'America/Montreal', 13), ('CA', 'America/Nipigon', 14), ('CA', 'America/Pangn
 irtung', 15), ('CA', 'America/Rainy_River', 16), ('CA', 'America/Rankin_Inlet', 17
), ('CA', 'America/Regina', 18), ('CA', 'America/Resolute', 19), ('CA', 'America/St
 _Johns', 20), ('CA', 'America/Swift_Current', 21), ('CA', 'America/Thunder_Bay', 2
 2), ('CA', 'America/Toronto', 23), ('CA', 'America/Vancouver', 24), ('CA', 'America
 /Whitehorse', 25), ('CA', 'America/Winnipeg', 26), ('CA', 'America/Yellowknife', 2
 7), ('CC', 'Indian/Cocos', 1), ('CD', 'Africa/Kinshasa', 1), ('CD', 'Africa/Lubumba
 shi', 2), ('CF', 'Africa/Bangui', 1), ('CG', 'Africa/Brazzaville', 1), ('CH', 'Europ
 e/Zurich', 1), ('CI', 'Africa/Abidjan', 1), ('CK', 'Pacific/Rarotonga', 1), ('CL', '
 America/Santiago', 1), ('CL', 'Pacific/Easter', 2), ('CM', 'Africa/Douala', 1), ('C
 N', 'Asia/Chongqing', 1), ('CN', 'Asia/Harbin', 2), ('CN', 'Asia/Kashgar', 3), ('CN'
 , 'Asia/Shanghai', 4), ('CN', 'Asia/Urumqi', 5), ('CO', 'America/Bogota', 1), ('CR',
 'America/Costa_Rica', 1), ('CU', 'America/Havana', 1), ('CV', 'Atlantic/Cape_Verd
 e', 1), ('CX', 'Indian/Christmas', 1), ('CY', 'Asia/Nicosia', 1), ('CZ', 'Europe/Pra
 gue', 1), ('DE', 'Europe/Berlin', 1), ('DJ', 'Africa/Djibouti', 1), ('DK', 'Europe/C
 openhagen', 1), ('DM', 'America/Dominica', 1), ('DO', 'America/Santo_Domingo', 1),
 ('DZ', 'Africa/Algiers', 1), ('EC', 'America/Guayaquil', 1), ('EC', 'Pacific/Galap
 agos', 2), ('EE', 'Europe/Tallinn', 1), ('EG', 'Africa/Cairo', 1), ('EH', 'Africa/El
 _Aaiun', 1), ('ER', 'Africa/Asmara', 1), ('ES', 'Europe/Madrid', 1), ('ES', 'Atlanti
 c/Canary', 2), ('ES', 'Africa/Ceuta', 3), ('ET', 'Africa/Addis_Ababa', 1), ('FI', 'E
 urope/Helsinki', 1), ('FJ', 'Pacific/Fiji', 1), ('FK', 'Atlantic/Stanley', 1), ('FM'
 , 'Pacific/Kosrae', 1), ('FM', 'Pacific/Ponape', 2), ('FM', 'Pacific/Truk', 3), ('F
 O', 'Atlantic/Faroe', 1), ('FR', 'Europe/Paris', 1), ('GA', 'Africa/Libreville', 1)
 , ('GB', 'Europe/London', 1), ('GD', 'America/Grenada', 1), ('GE', 'Asia/Tbilisi', 1
), ('GF', 'America/Cayenne', 1), ('GG', 'Europe/Guernsey', 1), ('GH', 'Africa/Accra
 ', 1), ('GI', 'Europe/Gibraltar', 1), ('GL', 'America/Danmarkshavn', 1), ('GL', 'Ame
 rica/Godthab', 2), ('GL', 'America/Scoresbysund', 3), ('GL', 'America/Thule', 4), ('
 GM', 'Africa/Banjul', 1), ('GN', 'Africa/Conakry', 1), ('GP', 'America/Guadeloupe
 ', 1), ('GQ', 'Africa/Malabo', 1), ('GR', 'Europe/Athens', 1), ('GS', 'Atlantic/Sout
 h_Georgia', 1), ('GT', 'America/Guatemala', 1), ('GU', 'Pacific/Guam', 1), ('GW', 'A
 frica/Bissau', 1), ('GY', 'America/Guyana', 1), ('HK', 'Asia/Hong_Kong', 1), ('HN',
 'America/Tegucigalpa', 1), ('HR', 'Europe/Zagreb', 1), ('HT', 'America/Port-au-
 Prince', 1), ('HU', 'Europe/Budapest', 1), ('ID', 'Asia/Jakarta', 1), ('ID', 'Asia/J
 ayapura', 2), ('ID', 'Asia/Makassar', 3), ('ID', 'Asia/Pontianak', 4), ('IE', 'Europ
 e/Dublin', 1), ('IL', 'Asia/Jerusalem', 1), ('IM', 'Europe/Isle_of_Man', 1), ('IN',
 'Asia/Kolkata', 1), ('IO', 'Indian/Chagos', 1), ('IQ', 'Asia/Baghdad', 1), ('IR', 'A
 sia/Tehran', 1), ('IS', 'Atlantic/Reykjavik', 1), ('IT', 'Europe/Rome', 1), ('JE', '
 Europe/Jersey', 1), ('JM', 'America/Jamaica', 1), ('JO', 'Asia/Amman', 1), ('JP', 'A
 sia/Tokyo', 1), ('KE', 'Africa/Nairobi', 1), ('KG', 'Asia/Bishkek', 1), ('KH', 'Asia
 /Phnom_Penh', 1), ('KI', 'Pacific/Enderbury', 1), ('KI', 'Pacific/Kiritimati', 2),
 ('KI', 'Pacific/Tarawa', 3), ('KM', 'Indian/Comoro', 1), ('KN', 'America/St_Kitts'
 ', 1), ('KP', 'Asia/Pyongyang', 1), ('KR', 'Asia/Seoul', 1), ('KW', 'Asia/Kuwait', 1),
 ('KY', 'America/Cayman', 1), ('KZ', 'Asia/Almaty', 1), ('KZ', 'Asia/Aqttau', 2), ('KZ'
 , 'Asia/Aqtobe', 3), ('KZ', 'Asia/Oral', 4), ('KZ', 'Asia/Qyzylorda', 5), ('LA', 'As
 ia/Vientiane', 1), ('LB', 'Asia/Beirut', 1), ('LC', 'America/St_Lucia', 1), ('LI', '
 Europe/Vaduz', 1), ('LK', 'Asia/Colombo', 1), ('LR', 'Africa/Monrovia', 1), ('LS', '
 Africa/Maseru', 1), ('LT', 'Europe/Vilnius', 1), ('LU', 'Europe/Luxembourg', 1), ('

LV', 'Europe/Riga', 1), ('LY', 'Africa/Tripoli', 1), ('MA', 'Africa/Casablanca', 1), ('MC', 'Europe/Monaco', 1), ('MD', 'Europe/Chisinau', 1), ('ME', 'Europe/Podgorica', 1), ('MF', 'America/Marigot', 1), ('MG', 'Indian/Antananarivo', 1), ('MH', 'Pacific/Kwajalein', 1), ('MH', 'Pacific/Majuro', 2), ('MK', 'Europe/Skopje', 1), ('ML', 'Africa/Bamako', 1), ('MM', 'Asia/Rangoon', 1), ('MN', 'Asia/Choibalsan', 1), ('MN', 'Asia/Hovd', 2), ('MN', 'Asia/Ulaanbaatar', 3), ('MO', 'Asia/Macau', 1), ('MP', 'Pacific/Saipan', 1), ('MQ', 'America/Martinique', 1), ('MR', 'Africa/Nouakchott', 1), ('MS', 'America/Montserrat', 1), ('MT', 'Europe/Malta', 1), ('MU', 'Indian/Mauritius', 1), ('MV', 'Indian/Maldives', 1), ('MW', 'Africa/Blantyre', 1), ('MX', 'America/Mexico_City', 1), ('MX', 'America/Merida', 2), ('MX', 'America/Mazatlan', 3), ('MX', 'America/Matamoros', 4), ('MX', 'America/Hermosillo', 5), ('MX', 'America/Chihuahua', 6), ('MX', 'America/Cancun', 7), ('MX', 'America/Monterrey', 8), ('MX', 'America/Ojinaga', 9), ('MX', 'America/Santa_Isabel', 10), ('MX', 'America/Tijuana', 1), ('MY', 'Asia/Kuala_Lumpur', 1), ('MY', 'Asia/Kuching', 2), ('MZ', 'Africa/Maputo', 1), ('NA', 'Africa/Windhoek', 1), ('NC', 'Pacific/Noumea', 1), ('NE', 'Africa/Niamey', 1), ('NF', 'Pacific/Norfolk', 1), ('NG', 'Africa/Lagos', 1), ('NI', 'America/Managua', 1), ('NL', 'Europe/Amsterdam', 1), ('NO', 'Europe/Oslo', 1), ('NP', 'Asia/Kathmandu', 1), ('NR', 'Pacific/Nauru', 1), ('NU', 'Pacific/Niue', 1), ('NZ', 'Pacific/Auckland', 1), ('NZ', 'Pacific/Chatham', 2), ('OM', 'Asia/Muscat', 1), ('PA', 'America/Panama', 1), ('PE', 'America/Lima', 1), ('PF', 'Pacific/Tahiti', 1), ('PF', 'Pacific/Marquesas', 2), ('PF', 'Pacific/Gambier', 3), ('PG', 'Pacific/Port_Moresby', 1), ('PH', 'Asia/Manila', 1), ('PK', 'Asia/Karachi', 1), ('PL', 'Europe/Warsaw', 1), ('PM', 'America/Miquelon', 1), ('PN', 'Pacific/Pitcairn', 1), ('PR', 'America/Puerto_Rico', 1), ('PS', 'Asia/Gaza', 1), ('PT', 'Europe/Lisbon', 1), ('PT', 'Atlantic/Madeira', 2), ('PT', 'Atlantic/Azores', 3), ('PW', 'Pacific/Palau', 1), ('PY', 'America/Asuncion', 1), ('QA', 'Asia/Qatar', 1), ('RE', 'Indian/Reunion', 1), ('RO', 'Europe/Bucharest', 1), ('RS', 'Europe/Belgrade', 1), ('RU', 'Europe/Moscow', 1), ('RU', 'Asia/Anadyr', 2), ('RU', 'Asia/Irkutsk', 3), ('RU', 'Asia/Kamchatka', 4), ('RU', 'Asia/Krasnoyarsk', 5), ('RU', 'Asia/Magadan', 6), ('RU', 'Asia/Novokuznetsk', 7), ('RU', 'Asia/Novosibirsk', 8), ('RU', 'Asia/Omsk', 9), ('RU', 'Asia/Sakhalin', 10), ('RU', 'Asia/Vladivostok', 11), ('RU', 'Asia/Yakutsk', 12), ('RU', 'Asia/Yekaterinburg', 13), ('RU', 'Europe/Kaliningrad', 14), ('RU', 'Europe/Samara', 15), ('RU', 'Europe/Volgograd', 16), ('RW', 'Africa/Kigali', 1), ('SA', 'Asia/Riyadh', 1), ('SB', 'Pacific/Guadalcanal', 1), ('SC', 'Indian/Mahe', 1), ('SD', 'Africa/Khartoum', 1), ('SE', 'Europe/Stockholm', 1), ('SG', 'Asia/Singapore', 1), ('SH', 'Atlantic/St_Helena', 1), ('SI', 'Europe/Ljubljana', 1), ('SJ', 'Arctic/Longyearbyen', 1), ('SK', 'Europe/Bratislava', 1), ('SL', 'Africa/Freetown', 1), ('SM', 'Europe/San_Marino', 1), ('SN', 'Africa/Dakar', 1), ('SO', 'Africa/Mogadishu', 1), ('SR', 'America/Paramaribo', 1), ('ST', 'Africa/Sao_Tome', 1), ('SV', 'America/El_Salvador', 1), ('SY', 'Asia/Damascus', 1), ('SZ', 'Africa/Mbabane', 1), ('TC', 'America/Grand_Turk', 1), ('TD', 'Africa/Ndjamena', 1), ('TF', 'Indian/Kerguelen', 1), ('TG', 'Africa/Lome', 1), ('TH', 'Asia/Bangkok', 1), ('TJ', 'Asia/Dushanbe', 1), ('TK', 'Pacific/Fakaofu', 1), ('TL', 'Asia/Dili', 1), ('TM', 'Asia/Ashgabat', 1), ('TN', 'Africa/Tunis', 1), ('TO', 'Pacific/Tongatapu', 1), ('TR', 'Europe/Istanbul', 1), ('TT', 'America/Port_of_Spain', 1), ('TV', 'Pacific/Funafuti', 1), ('TW', 'Asia/Taipei', 1), ('TZ', 'Africa/Dar_es_Salaam', 1), ('UA', 'Europe/Kiev', 1), ('UA', 'Europe/Simferopol', 2), ('UA', 'Europe/Uzhgorod', 3), ('UA', 'Europe/Zaporozhye', 4), ('UG', 'Africa/Kampala', 1), ('UM', 'Pacific/Johnston', 1), ('UM', 'Pacific/Midway', 2), ('UM', 'Pacific/Wake', 3), ('US', 'America/Adak', 1), ('US', 'America/Anchorage', 2), ('US', 'America/Boise', 3), ('US', 'America/Chicago', 4), ('US', 'America/Denver', 5), ('US', 'America/Detroit', 6), ('US', 'America/Indiana/Indianapolis', 7), ('US', 'America/Indiana/Knox', 8), ('US', 'America/Indiana/Marengo', 9), ('US', 'America/Indiana/Petersburg', 10), ('US', 'America/Indiana/Tell_City', 11), ('US', 'America/Indiana/Vevay', 12), ('US', 'America/Indiana/Vincennes', 13), ('US', 'America/Indiana/Winamac', 14), ('US', 'America/Juneau', 15), ('US', 'America/Kentucky/Louisville', 16), ('US', 'America/Kentucky/Monticello', 17), ('US', 'America/Los_Angeles', 18), ('US', 'America/Menominee', 19), ('US', 'America/New_York', 20), ('US', 'America/Nome', 21), ('US', 'America/North_Dakota/Center', 22), ('US', 'America/North_Dakota/New_Salem', 23), ('US', 'America/Phoenix', 24), ('US', 'America/Shiprock', 25), ('US', 'America/Yakutat', 26), ('US', 'Pacific/Honolulu', 27), ('UY', 'America/Montevideo', 1), ('UZ', 'Asia/Samarkand', 1), ('UZ', 'Asia/Tashkent', 2), ('VA', 'Europe/Vatican', 1), ('VC', 'America/St_Vincent', 1), ('VE', 'America/Caracas', 1), ('VG', 'Am

```
erica/Tortola',1), ('VI', 'America/St_Thomas',1), ('VN', 'Asia/Ho_Chi_Minh',1),
('VU', 'Pacific/Efate',1), ('WF', 'Pacific/Wallis',1), ('WS', 'Pacific/Apia',1),
('YE', 'Asia/Aden',1), ('YT', 'Indian/Mayotte',1), ('ZA', 'Africa/Johannesburg',
1), ('ZM', 'Africa/Lusaka',1), ('ZW', 'Africa/Harare',1);
```

Table: country

The table country consists of the following columns:

- id (the id of the country, this is the primary key of the table, the value of the field needs to be filled)
- name (the name of the country, the name of the country needs to be unique, default value is NULL)

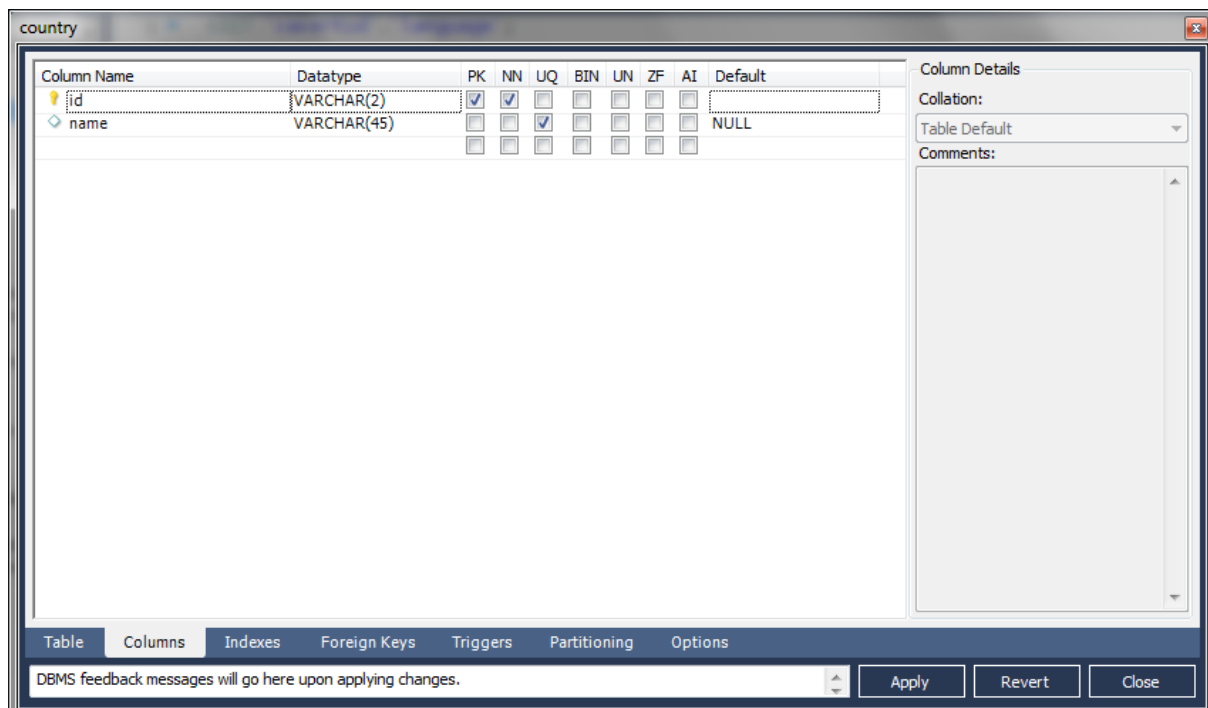


Figure B.3: Table country

Code:

```
CREATE TABLE `country` (
  `id` varchar(2) NOT NULL,
  `name` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `country` VALUES ('AF','Afghanistan'),('AX','Aland
Islands'),('AL','Albania'),('DZ','Algeria'),('AS','American
Samoa'),('AD','Andorra'),('AO','Angola'),('AI','Anguilla'),('AQ','Antarctic
a'),('AG','Antigua and
Barbuda'),('AR','Argentina'),('AM','Armenia'),('AW','Aruba'),('AU','Austral
ia'),('AT','Austria'),('AZ','Azerbaijan'),('BS','Bahamas'),('BH','Bahrain')
,('BD','Bangladesh'),('BB','Barbados'),('BY','Belarus'),('BE','Belgium'),('
BZ','Belize'),('BJ','Benin'),('BM','Bermuda'),('BT','Bhutan'),('BO','Bolivi
a'),('BA','Bosnia and Herzegovina'),('BW','Botswana'),('BV','Bouvet
Island'),('BR','Brazil'),('IO','British Indian Ocean
```

Territory'), ('BN', 'Brunei Darussalam'), ('BG', 'Bulgaria'), ('BF', 'Burkina Faso'), ('BI', 'Burundi'), ('KH', 'Cambodia'), ('CM', 'Cameroon'), ('CA', 'Canada'), ('CV', 'Cape Verde'), ('KY', 'Cayman Islands'), ('CF', 'Central African Republic'), ('TD', 'Chad'), ('CL', 'Chile'), ('CN', 'China'), ('CX', 'Christmas Island'), ('CC', 'Cocos (Keeling) Islands'), ('CO', 'Colombia'), ('KM', 'Comoros'), ('CG', 'Congo'), ('CD', 'Congo, The Democratic Republic of The'), ('CK', 'Cook Islands'), ('CR', 'Costa Rica'), ('CI', 'Cote D'Ivoire'), ('HR', 'Croatia'), ('CU', 'Cuba'), ('CY', 'Cyprus'), ('CZ', 'Czech Republic'), ('DK', 'Denmark'), ('DJ', 'Djibouti'), ('DM', 'Dominica'), ('DO', 'Dominican Republic'), ('EC', 'Ecuador'), ('EG', 'Egypt'), ('SV', 'El Salvador'), ('GQ', 'Equatorial Guinea'), ('ER', 'Eritrea'), ('EE', 'Estonia'), ('ET', 'Ethiopia'), ('FK', 'Falkland Islands (Malvinas)'), ('FO', 'Faroe Islands'), ('FJ', 'Fiji'), ('FI', 'Finland'), ('FR', 'France'), ('GF', 'French Guiana'), ('PF', 'French Polynesia'), ('TF', 'French Southern Territories'), ('GA', 'Gabon'), ('GM', 'Gambia'), ('GE', 'Georgia'), ('DE', 'Germany'), ('GH', 'Ghana'), ('GI', 'Gibraltar'), ('GR', 'Greece'), ('GL', 'Greenland'), ('GD', 'Grenada'), ('GP', 'Guadeloupe'), ('GU', 'Guam'), ('GT', 'Guatemala'), ('GG', 'Guernsey'), ('GN', 'Guinea'), ('GW', 'Guinea-Bissau'), ('GY', 'Guyana'), ('HT', 'Haiti'), ('HM', 'Heard Island and McDonald Islands'), ('VA', 'Holy See (Vatican City State)'), ('HN', 'Honduras'), ('HK', 'Hong Kong'), ('HU', 'Hungary'), ('IS', 'Iceland'), ('IN', 'India'), ('ID', 'Indonesia'), ('IR', 'Iran, Islamic Republic of'), ('IQ', 'Iraq'), ('IE', 'Ireland'), ('IM', 'Isle of Man'), ('IL', 'Israel'), ('IT', 'Italy'), ('JM', 'Jamaica'), ('JP', 'Japan'), ('JE', 'Jersey'), ('JO', 'Jordan'), ('KZ', 'Kazakhstan'), ('KE', 'Kenya'), ('KI', 'Kiribati'), ('KP', 'Korea, Democratic People's Republic of'), ('KR', 'Korea, Republic of'), ('KW', 'Kuwait'), ('KG', 'Kyrgyzstan'), ('LA', 'Lao People's Democratic Republic'), ('LV', 'Latvia'), ('LB', 'Lebanon'), ('LS', 'Lesotho'), ('LR', 'Liberia'), ('LY', 'Libyan Arab Jamahiriya'), ('LI', 'Liechtenstein'), ('LT', 'Lithuania'), ('LU', 'Luxembourg'), ('MO', 'Macao'), ('MK', 'Macedonia, The Former Yugoslav Republic of'), ('MG', 'Madagascar'), ('MW', 'Malawi'), ('MY', 'Malaysia'), ('MV', 'Maldives'), ('ML', 'Mali'), ('MT', 'Malta'), ('MH', 'Marshall Islands'), ('MQ', 'Martinique'), ('MR', 'Mauritania'), ('MU', 'Mauritius'), ('YT', 'Mayotte'), ('MX', 'Mexico'), ('FM', 'Micronesia, Federated States of'), ('MD', 'Moldova, Republic of'), ('MC', 'Monaco'), ('MN', 'Mongolia'), ('ME', 'Montenegro'), ('MS', 'Montserrat'), ('MA', 'Morocco'), ('MZ', 'Mozambique'), ('MM', 'Myanmar'), ('NA', 'Namibia'), ('NR', 'Nauru'), ('NP', 'Nepal'), ('NL', 'Netherlands'), ('AN', 'Netherlands Antilles'), ('NC', 'New Caledonia'), ('NZ', 'New Zealand'), ('NI', 'Nicaragua'), ('NE', 'Niger'), ('NG', 'Nigeria'), ('NU', 'Niue'), ('NF', 'Norfolk Island'), ('MP', 'Northern Mariana Islands'), ('NO', 'Norway'), ('OM', 'Oman'), ('PK', 'Pakistan'), ('PW', 'Palau'), ('PS', 'Palestinian Territory, Occupied'), ('PA', 'Panama'), ('PG', 'Papua New Guinea'), ('PY', 'Paraguay'), ('PE', 'Peru'), ('PH', 'Philippines'), ('PN', 'Pitcairn'), ('PL', 'Poland'), ('PT', 'Portugal'), ('PR', 'Puerto Rico'), ('QA', 'Qatar'), ('RE', 'Reunion'), ('RO', 'Romania'), ('RU', 'Russian Federation'), ('RW', 'Rwanda'), ('BL', 'Saint Barthélemy'), ('SH', 'Saint Helena'), ('KN', 'Saint Kitts and Nevis'), ('LC', 'Saint Lucia'), ('MF', 'Saint Martin'), ('PM', 'Saint Pierre and Miquelon'), ('VC', 'Saint Vincent and The Grenadines'), ('WS', 'Samoa'), ('SM', 'San Marino'), ('ST', 'Sao Tome and Principe'), ('SA', 'Saudi Arabia'), ('SN', 'Senegal'), ('RS', 'Serbia'), ('SC', 'Seychelles'), ('SL', 'Sierra Leone'), ('SG', 'Singapore'), ('SK', 'Slovakia'), ('SI', 'Slovenia'), ('SB', 'Solomon Islands'), ('SO', 'Somalia'), ('ZA', 'South Africa'), ('GS', 'South Georgia and The South Sandwich Islands'), ('ES', 'Spain'), ('LK', 'Sri Lanka'), ('SD', 'Sudan'), ('SR', 'Suriname'), ('SJ', 'Svalbard and Jan Mayen'), ('SZ', 'Swaziland'), ('SE', 'Sweden'), ('CH', 'Switzerland'), ('SY', 'Syrian Arab Republic'), ('TW', 'Taiwan, Province of



```
China'), ('TJ', 'Tajikistan'), ('TZ', 'Tanzania, United Republic
of'), ('TH', 'Thailand'), ('TL', 'Timor-
leste'), ('TG', 'Togo'), ('TK', 'Tokelau'), ('TO', 'Tonga'), ('TT', 'Trinidad and
Tobago'), ('TN', 'Tunisia'), ('TR', 'Turkey'), ('TM', 'Turkmenistan'), ('TC', 'Turk
s and Caicos
Islands'), ('TV', 'Tuvalu'), ('UG', 'Uganda'), ('UA', 'Ukraine'), ('AE', 'United
Arab Emirates'), ('GB', 'United Kingdom'), ('US', 'United
States'), ('UM', 'United States Minor Outlying
Islands'), ('UY', 'Uruguay'), ('UZ', 'Uzbekistan'), ('VU', 'Vanuatu'), ('VE', 'Vene
zuela'), ('VN', 'Viet Nam'), ('VG', 'Virgin Islands, British'), ('VI', 'Virgin
Islands, U.S.'), ('WF', 'Wallis and Futuna'), ('EH', 'Western
Sahara'), ('YE', 'Yemen'), ('ZM', 'Zambia'), ('ZW', 'Zimbabwe');
```

Table: language

The table language consists of the following columns:

- id (the id of the language, this is the primary key of the table, the value of the field needs to be filled)
- name (the name of the language, the name of the language needs to be unique, default value is NULL)

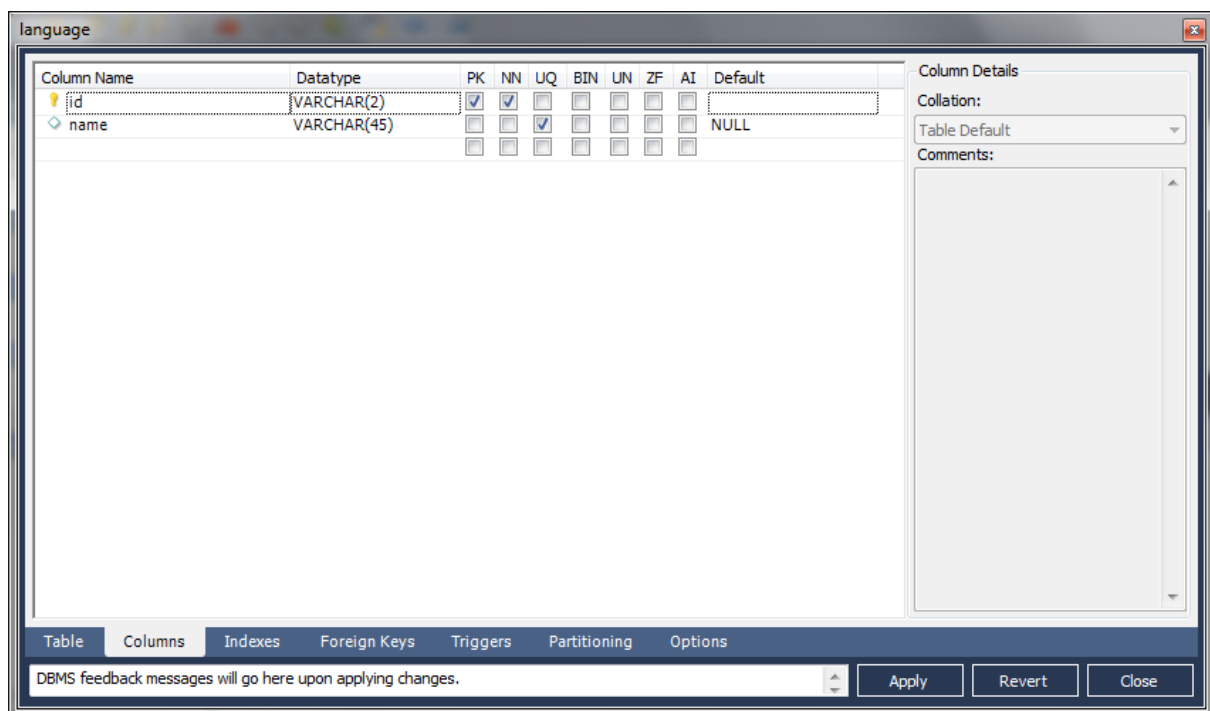


Figure B.4: Table language

Code:

```
CREATE TABLE `language` (
  `id` varchar(2) NOT NULL,
  `name` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `language` VALUES
('AB', 'Abkhazian'), ('AA', 'Afar'), ('AF', 'Afrikaans'), ('AL', 'Albanian'), ('AM'
```

, 'Amharic'), ('AR', 'Arabic'), ('HY', 'Armenian'), ('AS', 'Assamese'), ('AY', 'Aymara'), ('AZ', 'Azerbaijani'), ('BA', 'Bashkir'), ('EU', 'Basque'), ('BN', 'Bengali/Bangla'), ('DZ', 'Bhutani'), ('BH', 'Bihari'), ('BI', 'Bislama'), ('BR', 'Breton'), ('BG', 'Bulgarian'), ('MY', 'Burmese'), ('BE', 'Byelorussian'), ('KM', 'Cambodian'), ('CA', 'Catalan'), ('CN', 'Chinese'), ('CO', 'Corsican'), ('HR', 'Croatian'), ('EZ', 'Czech'), ('DA', 'Danish'), ('NL', 'Dutch'), ('EN', 'English/ American English'), ('EO', 'Esperanto'), ('ET', 'Estonian'), ('FO', 'Faeroese'), ('FJ', 'Fiji'), ('FI', 'Finnish'), ('FR', 'French'), ('FY', 'Frisian'), ('GD', 'Gaelic/ Scots Gaelic'), ('GL', 'Galician'), ('GG', 'Georgian'), ('DE', 'German'), ('GR', 'Greek'), ('KL', 'Greenlandic'), ('GN', 'Guarani'), ('GU', 'Gujarati'), ('HA', 'Hausa'), ('IW', 'Hebrew'), ('HI', 'Hindi'), ('HU', 'Hungarian'), ('IS', 'Icelandic'), ('IN', 'Indonesian'), ('IA', 'Interlingua'), ('IE', 'Interlingue'), ('IK', 'Inupiak'), ('GA', 'Irish'), ('IT', 'Italian'), ('JA', 'Japanese'), ('JW', 'Javanese'), ('KN', 'Kannada'), ('KS', 'Kashmiri'), ('KK', 'Kazakh'), ('RW', 'Kinyarwanda'), ('KY', 'Kirghiz'), ('RN', 'Kirundi'), ('KO', 'Korean'), ('KU', 'Kurdish'), ('LO', 'Laothian'), ('LA', 'Latin'), ('LV', 'Latvian/ Lettish'), ('LN', 'Lingala'), ('LT', 'Lithuanian'), ('MK', 'Macedonian'), ('MG', 'Malagasy'), ('MS', 'Malay'), ('ML', 'Malayalam'), ('MT', 'Maltese'), ('MI', 'Maori'), ('MR', 'Marathi'), ('MO', 'Moldavian'), ('MN', 'Mongolian'), ('NA', 'Nauru'), ('NE', 'Nepali'), ('NO', 'Norwegian'), ('OC', 'Occitan'), ('OR', 'Oriya'), ('OM', 'Oromo/ Afan'), ('PS', 'Pashto/ Pushto'), ('FA', 'Persian'), ('PL', 'Polish'), ('PT', 'Portuguese'), ('PA', 'Punjabi'), ('QU', 'Quechua'), ('RM', 'Rhaeto-Romance'), ('RO', 'Romanian'), ('RU', 'Russian'), ('SM', 'Samoan'), ('SG', 'Sangro'), ('SA', 'Sanskrit'), ('SR', 'Serbian'), ('SH', 'Serbo-Croatian'), ('ST', 'Sesotho'), ('TN', 'Setswana'), ('SN', 'Shona'), ('SD', 'Sindhi'), ('SI', 'Singhalese'), ('SS', 'Siswati'), ('SK', 'Slovak'), ('SL', 'Slovenian'), ('SO', 'Somali'), ('ES', 'Spanish'), ('SU', 'Sudanese'), ('SW', 'Swahili'), ('SV', 'Swedish'), ('TL', 'Tagalog'), ('TG', 'Tajik'), ('TA', 'Tamil'), ('TT', 'Tatar'), ('TE', 'Tegulu'), ('TH', 'Thai'), ('BO', 'Tibetan'), ('TI', 'Tigrinya'), ('TO', 'Tonga'), ('TS', 'Tsonga'), ('TR', 'Turkish'), ('TK', 'Turkmen'), ('TW', 'Twi'), ('UK', 'Ukrainian'), ('UR', 'Urdu'), ('UZ', 'Uzbek'), ('VI', 'Vietnamese'), ('VO', 'Volapuk'), ('CY', 'Welsh'), ('WO', 'Wolof'), ('XH', 'Xhosa'), ('JI', 'Yiddish'), ('YO', 'Yoruba'), ('ZU', 'Zulu');

Attachment C: Proof of Concept Code

Altered Files

index.php

We added the following lines to include files that contain new functions.

```

/* Added several additional functions for logging on with a certificate */
include_once "cacert.inc";
/* Added MySQL scripts and added database functionality */
include_once "mysql.inc";

```

We modified the routes array. The routes array is used to call a function when a certain page is requested.

```

/* Added "'modifyuser' => 'user_modify', 'updateuser' =>
'user_update', 'createuser' => 'user_create',
'deleteuser' => 'user_delete'," to the $routes array */
$routes = array(
    'continue' => 'simpleid_continue',
    'send' => 'simpleid_send',
    'autorelease' => 'simpleid_autorelease',
    'modifyuser' => 'user_modify',
    'updateuser' => 'user_update',
    'createuser' => 'user_create',
    'deleteuser' => 'user_delete',
    'login' => 'user_login',
    'logout' => 'user_logout',
    'my/dashboard' => 'page_dashboard',
    'my/sites' => 'page_sites',
    'my/profile' => 'page_profile',
    'user' => 'user_public_page',
    'user/(.+)' => 'user_public_page',
    'xrds/(.*)' => 'user_xrds',
    'xrds' => 'simpleid_xrds',
    '.*' => 'simpleid_index'
);

```

config.inc

We added the MySQL parameter to the config file. The config file is the only one that contains variables that need to be changed when the application is moved to another website.

```

/**
 * MySQL parameters.
 *
 * You can specify MySQL parameters that will be used to logon to the
mysql database
 *
 * Examples:
 * <code>
 *   define('CACERTID_MYSQL_SERVER', 'localhost');
 *   define('CACERTID_MYSQL_DATABASE', 'db');
 *   define('CACERTID_MYSQL_USER', 'username');

```

```

*   define('CACERTID_MYSQL_PASSWORD', 'password');
* </code>
*
*/
define('CACERTID_MYSQL_SERVER', 'localhost');
define('CACERTID_MYSQL_DATABASE', 'cacertid');
define('CACERTID_MYSQL_USER', 'root');
define('CACERTID_MYSQL_PASSWORD', 'klomp21');

```

We also added the Identity URL, better known as the OpenID identifier to the config.inc file. This value is only displayed to users to help them see what their Identity is. It has no influence on the identity url itself.

```

/**
 * Cacert identity URL.
 *
 * You can specify the URL of the OpenID identifiers.
 * This value is only displayed in create and modify user forms
 *
 * Examples:
 * <code>
 *   define('CACERTID_IDENTITY_URL', 'http://onno.uni.cc/~identity');
 *   define('CACERTID_IDENTITY_URL', 'http://onno.uni.cc/~username');
 *   define('CACERTID_IDENTITY_URL', 'http://onno.uni.cc/~nickname');
 * </code>
 *
 */
define('CACERTID_IDENTITY_URL', 'http://onno.uni.cc/~nickname');

```

filesystem.store.inc

The changes that have been made are such that it uses our database instead of an identity file. The names of these functions have remained the same.

Altered Functions

store_user_exists(\$uid)

```

/**
 * Returns whether the user name exists in the user store.
 *
 * @param string $uid the name of the user to check
 * @param bool whether the user name exists
 */
function store_user_exists($uid) {
    // Start database connection
    dbconnect();
    // Checking if there is an account with the respective nickname.
    $userexists=false;
    $result = mysql_query("SELECT * FROM user WHERE nickname =
''. $uid.'"); or die(mysql_error());
    while($row = mysql_fetch_array($result))
    {
        $userexists=true;
    }
    // Closing the database connection
    dbdisconnect();
    return ($userexists);
}

```

store_user_load(\$uid)

```

/**
 * Loads user data for a specified user name.
 *
 * The user name must exist. You should check whether the user name
exists with
 * the {@link store_user_exists()} function
 *
 * @param string $uid the name of the user to load
 * @return mixed data for the specified user
 */
function store_user_load($uid) {
    $store_file = SIMPLEID_STORE_DIR . "/" . $uid . ".usrstore";

    if (file_exists($store_file)) {
        $data = unserialize(file_get_contents($store_file));
    } else {
        $data = array();
    }

    // Start database connection
    dbconnect();
    // Creating an array with all the user data
    global $uidinfo;
    $result = mysql_query("SELECT * FROM user WHERE nickname =
'" . $uid . "'") or die(mysql_error());
    while($row = mysql_fetch_array($result))
    {
        $uidinfo["identity"] = $row['nickname'];
        $uidinfo["pass"] = $row['pass'];
        $uidinfo["sreg"]["nickname"] = $row['nickname'];
        $uidinfo["sreg"]["email"] = $row['email'];
        $uidinfo["sreg"]["fullname"] = $row['fullname'];
        $uidinfo["sreg"]["dob"] = $row['dob'];
        $uidinfo["sreg"]["gender"] = $row['gender'];
        $uidinfo["sreg"]["postcode"] = $row['postcode'];
        $uidinfo["sreg"]["country"] = $row['country'];
        $uidinfo["sreg"]["language"] = $row['language'];
        $uidinfo["sreg"]["timezone"] = $row['timezone'];
    }
    // Closing the database connection
    dbdisconnect();
    // Merging the array (standard functionality)
    $data = array_merge($data, $uidinfo);

    return $data;
}

```

user.inc***Altered Functions****user_login()*

```

/**
 * Attempts to log in a user, using the user name and password specified
in the
 * HTTP request.
 */

```

```

function user_login()
{
    global $user, $GETPOST;

    //If the user is already logged in, return
    if (isset($user["uid"]))
openid_indirect_response_redirect(simpleid_url(), '');

    $destination = $GETPOST['destination'];
    $state = (isset($GETPOST['s'])) ? $GETPOST['s'] : '';
    $query = 'q=' . $destination;
    $query .= ($state) ? '&s=' . rawurlencode($state) : '';

    if ($_POST['op'] == 'Cancel') {
        global $version;

        $request = unpickle($state);
        $version = openid_get_version($request);

        if (isset($request['openid.return_to'])) {
            $return_to = $request['openid.return_to'];
            $response = simpleid_checkid_error(FALSE);
            redirect_form($return_to, $response);
        } else {
            indirect_fatal_error('Login cancelled without a proper
OpenID request. ');
        }
        return;
    }

    // We allow legacy login if the connection is via HTTPS or if
SIMPLEID_ALLOW_LEGACY_LOGIN is true
    $allow_legacy_login = (is_https() || SIMPLEID_ALLOW_LEGACY_LOGIN);

    if (!isset($_POST['name'])) $_POST['name'] = '';
    if (!isset($_POST['pass'])) $_POST['pass'] = '';
    if (!isset($_POST['digest'])) $_POST['digest'] = '';
/*
*/
    Start of addition to the login script
*/
    // This sets the account name when logging in with the certificate
    if (isset($_SERVER['HTTPS']))
    {
        // Start database connection
        dbconnect();

        // Parsing the client certificate.
        $cert_data = openssl_x509_parse($_SERVER['SSL_CLIENT_CERT']);

        // Grabbing the emailaddress(es) from the certificate
        If (isset($cert_data['subject']['emailAddress']) &&
is_array($cert_data['subject']['emailAddress']))
        {
            global $email, $nr;
            $emails = count($cert_data['subject']['emailAddress']);
            $nr = 0;
            for ($i=0; $i<=($emails-1); $i++)
            {
                If
(substr_count($cert_data['subject']['emailAddress'][$i], "@")==1 &&
substr_count($cert_data['subject']['emailAddress'][$i], ".")>=1)
            {

```

```

        $email[$nr] =
$cert_data['subject']['emailAddress'][$i];
        $nr = $nr + 1;
    }
}
If (count($email) == 0)
{$emails = 0;}
ElseIf (count($email) == 1)
{$emails = 1;}
Else
{$emails = count($email);}
}
elseif (isset($cert_data['subject']['emailAddress']))
{
    $email[0] = $cert_data['subject']['emailAddress'];
    If (substr_count($email[0], "@"==1 &&
substr_count($email[0], ".")>=1)
        {$emails = 1;}
    Else
        {$emails = 0;}
}
Else
{$emails = 0;}

global $emailquery, $name;
//Building part of the SQL query, the WHERE part.
If ($emails == 0)
{
    // Each certificate needs to contain at least one
emailaddress
    set_message('The certificate you supplied does not contain
an emailaddress');
    cache_delete('user-nonce', $_POST['nonce']);
    user_login_form($destination, $state);
    return;
}
ElseIf ($emails == 1)
{
    $emailquery = "email = '". $email[0]. "'";
}
Else
{
    $emailquery = "";
    for ($i=0; $i<=(count($email)-1); $i++)
    {
        If ($emailquery==""){$emailquery=" email =
'".$email[$i]."'";}
        Else {$emailquery = $emailquery." OR email =
'".$email[$i]."'";}
    }
}

// Checking if there is an account on the database linked with
any of the emailaddresses from the certificate
If ($emails > 0)
{
    // Checking if there is an account with any of the emails in
the certificate.
    $result = mysql_query("SELECT nickname, pass FROM user WHERE
'".$emailquery) or die(mysql_error());
    while($row = mysql_fetch_array($result))
    {

```



```

        $name = $row['nickname'];
    }
}
// Setting $_POST['name']
If ($name != ''){$_POST['name'] = $name;}
Else
{
    set_message('None of the emails found in your certificate
can be matched to an account which is stored on the server. ');
    cache_delete('user-nonce', $_POST['nonce']);
    user_login_form($destination, $state);
    return;
}
}

/*
End of addition to the login script
*/

// Added !isset($_SERVER['HTTPS']) && to prevent failure due to
missing password while login in with a certificate
if (
    ($_POST['name'] == '')
    || !isset($_SERVER['HTTPS']) && (
        ($allow_legacy_login && ($_POST['pass'] == '') &&
($_POST['digest'] == ''))
        || (! $allow_legacy_login && ($_POST['digest'] == ''))
    )
) {
    if (isset($_POST['destination'])) {
        // User came from a log in form.
        set_message('You need to supply the user name and the
password in order to log in. ');
    }
    cache_delete('user-nonce', $_POST['nonce']);
    user_login_form($destination, $state);
    return;
}

if (!isset($_POST['nonce'])) {
    if (isset($_POST['destination']))
    {
        // User came from a log in form.
        set_message('You seem to be attempting to log in from
another web page. You must use this page to log in. ');
    }
    user_login_form($destination, $state);
    return;
}

$time = strtotime(substr($_POST['nonce'], 0, 20));
// Some old versions of PHP does not recognise the T in the ISO 8601
date. We may need to convert the T to a space
if (($time == -1) || ($time === FALSE)) $time =
strtotime(strtr(substr($_POST['nonce'], 0, 20), 'T', ' '));

if (!cache_get('user-nonce', $_POST['nonce'])) {
    log_warn('Login attempt: Nonce ' . $_POST['nonce'] . ' not issued
or is being reused. ');
    set_message('SimpleID detected a potential security attack on
your log in. Please log in again. ');
    user_login_form($destination, $state);
}

```

```

        return;
    } elseif ($time < time() - SIMPLEID_LOGIN_NONCE_EXPIRES_IN) {
        log_notice('Login attempt: Nonce ' . $_POST['nonce'] . '
expired.');
```

expired.');

```

        set_message('The log in page has expired. Please log in
again.');
```

again.');

```

        user_login_form($destination, $state);
        return;
    } else {
        cache_delete('user-nonce', $_POST['nonce']);
    }

    $test_user = user_load($_POST['name']);
    if ($test_user == NULL) {
        set_message('The user name or password is not correct.');
```

The user name or password is not correct.');

```

        user_login_form($destination, $state);
        return;
    }
}
/*
Start of altered login script
*/
// Checks if a client certificate has been set
if (isset($_SERVER['SSL_CLIENT_CERT']))
{
    // You can add more unauthorized common names (CN) if that is
necessary. it is also possible to remove this clause altogether
    /* Normally the CN should be the name of the certificate owner.
Some CAcert certificate have CAcert WoT User as CN */
    $unauthorized_cn = array("CAcert WoT User");

    // Checks whether the cn is in the list of unauthorized cn if the
variable has been set.
    // The certificate must contain the name of the person.
    if (isset($unauthorized_cn) &&
in_array($cert_data['subject']['CN'], $unauthorized_cn))
    {
        // The user is forwarded back to the http website
        header('Location: '.httplocation().'/index.php?nn=1');
        exit;
    }
    else
    {
        // Checks whether the certificate is intended to be used on
the client side of an SSL connection.
        If (!openssl_x509_checkpurpose($_SERVER['SSL_CLIENT_CERT'],
X509_PURPOSE_SSL_CLIENT) == 1)
        {
            // The user is forwarded back to the http website
            header('Location: '.httplocation().'/index.php?nn=2');
            exit;
        }
    }
}
// standard username and password logon
else
{
    if ($_POST['digest'] && !_user_verify_digest($_POST['digest'],
$_POST['nonce'], $test_user))
    {
        set_message('The user name or password is not correct');
```

The user name or password is not correct');

```

        user_login_form($destination, $state);
        return;
    }
}
```



```

    }
    elseif ($allow_legacy_login && !$_POST['digest'] &&
(md5($_POST['pass']) != $test_user['pass']))
    {
        set_message('The user name or password is not correct. ');
        user_login_form($destination, $state);
        return;
    }
}
// Closing the database connection
dbdisconnect();
/*
End of altered login script
*/
log_info('Login successful: ' . $test_user['uid']);
_user_login($test_user);

if (isset($_POST['autologin']) && ($_POST['autologin'] == 1))
user_autologin_create();

openid_indirect_response_redirect(simpleid_url(), $query);
}

```

user_login_form ()

```

/**
 * Displays a user login form.
 *
 * @param string $destination the SimpleID location to which the user is
directed
 * if login is successful
 * @param string $state the current SimpleID state, if required by the
location
 */
function user_login_form($destination = '', $state = NULL) {
    global $xtpl;

    if ($state) {
        $xtpl->assign('state', htmlspecialchars($state, ENT_QUOTES, 'UTF-
8'));
        $xtpl->parse('main.login.state');
    }

    cache_gc(SIMPLEID_LOGIN_NONCE_EXPIRES_IN, 'user-nonce');
    $nonce = openid_nonce();
    cache_set('user-nonce', $nonce, 1);

    $xtpl->assign('javascript', '<script src="html/md5.js"
type="text/javascript"></script><script src="html/user-login.js"
type="text/javascript"></script>');
    // Checking if the user came from a failed certificate logon
    If (isset($_GET['nn']))
    {
        If ($_GET['nn']==1)
        {
            set_message('The certificate you attempted to logon with
does not contain your name. ');
        }
        ElseIf ($_GET['nn']==2)
        {
            set_message('The certificate you used to logon with is not
intended to be used as a client certificate. ');
        }
    }
}

```

```

        Else
        {
            set_message('Something went wrong while logging on with your
certificate');
        }
    }

    if (is_https()) {
        $xtpl->assign('security_class', 'secure');
        $xtpl->assign('security_message', 'Secure login using
<strong>HTTPS</strong>.');
    } else {
        if (!SIMPLEID_ALLOW_LEGACY_LOGIN) {
            $xtpl->assign('security_class', 'unsecure login-digest');
            $xtpl->assign('security_message', 'Your SimpleID
configuration does not allow you to log in unsecurely. Please enable
JavaScript and try again, or see <a
href="http://simpleid.sourceforge.net/documentation/using-
simpleid/logging-simpleid">the SimpleID documentation</a> for more
details.');
```

```

            $xtpl->assign('security_disabled', 'disabled="disabled"');
        } else {
            $xtpl->assign('security_class', 'unsecure login-digest');
            $xtpl->assign('security_message', '<strong>WARNING:</strong>
Your password will be sent to SimpleID as plain text.');
```

```

        }
    }
}
/*
    Start of addition to the script
    The following addition to the script enforces that the user keeps on
using the secure https connection.
*/
    // If a user is not accessing the page through https he must be forced
to do so while logging on with a certificate.
    $xtpl->assign('rootdir', httplocation());
    $xtpl->assign('srootdir', httpslocation());
/*
    End of addition to the script
    The following addition to the script enforces that the user keeps on
using the secure https connection.
*/

    extension_invoke_all('user_login_form', $destination, $state);

    $xtpl->assign('title', 'Log In');
    $xtpl->assign('page_class', 'dialog-page');
    $xtpl->assign('destination', htmlspecialchars($destination,
ENT_QUOTES, 'UTF-8'));
    $xtpl->assign('nonce', htmlspecialchars($nonce, ENT_QUOTES, 'UTF-8'));

    $xtpl->parse('main.login');
    $xtpl->parse('main.framekiller');
    $xtpl->parse('main');
    $xtpl->out('main');
}

```

New functions

user_modify()

```
// Create a form in which a user can create, modify or delete his account.
function user_modify() {
    global $xtpl;
    cache_gc(SIMPLEID_LOGIN_NONCE_EXPIRES_IN, 'user-nonce');
    $nonce = openid_nonce();
    cache_set('user-nonce', $nonce, 1);
    $xtpl->assign('javascript', '<script src="html/md5.js"
type="text/javascript"></script><script src="html/user-login.js"
type="text/javascript"></script>');

    // Forcing usage of https
    If (!is_https())
    {
        header('Location: '.httpslocation().'/index.php?q=modifyuser');
    }
    Else
    {
        $xtpl->assign('security_class', 'secure');
        $xtpl->assign('security_message', 'Connection uses HTTPS');
    }
    // Start database connection
    dbconnect();
    // Parsing the client certificate.
    $cert_data = openssl_x509_parse($_SERVER['SSL_CLIENT_CERT']);

    $fullname = $cert_data['subject']['CN'];
    If ($fullname == "CAcert WoT User")
    {
        $xtpl->assign('security_class', 'secure');
        $xtpl->assign('security_message', 'Your name needs to be in your
client certidicate');
        $xtpl->assign('security_disabled', 'disabled="disabled"');
    }

    // Grabbing the emailaddress(es) from the certificate
    If (isset($cert_data['subject']['emailAddress']) &&
is_array($cert_data['subject']['emailAddress']))
    {
        global $email, $nr;
        $emails = count($cert_data['subject']['emailAddress']);
        $nr = 0;
        for ($i=0; $i<=($emails-1); $i++)
        {
            If
(substr_count($cert_data['subject']['emailAddress'][$i],"@")==1 &&
substr_count($cert_data['subject']['emailAddress'][$i],".")>=1)
            {
                $email[$nr] = $cert_data['subject']['emailAddress'][$i];
                $nr = $nr + 1;
            }
        }
        If (count($email) == 0)
        {$emails = 0;}
        ElseIf (count($email) == 1)
        {$emails = 1;}
        Else
        {$emails = count($email);}
    }
}
```

```

}
elseif (isset($cert_data['subject']['emailAddress']))
{
    $email[0] = $cert_data['subject']['emailAddress'];
    If (substr_count($email[0], "@")==1 &&
substr_count($email[0], ".")>=1)
    {$emails = 1;}
    Else
    {$emails = 0;}
}
Else
{$emails = 0;}
    global $emailquery;
If ($emails == 0)
{
    $xtpl->parse('main.modifyuser.oneemail');
    $xtpl->assign('security_class', 'secure');
    $xtpl->assign('security_message', 'The certificate you selected
does not contain an emailadres');
    $xtpl->assign('security_disabled', 'disabled="disabled"');
}
ElseIf ($emails == 1)
{
    $emailquery = "email = '". $email[0]. "'";
    $xtpl->assign('email', $email[0]);
    $xtpl->parse('main.modifyuser.oneemail');
}
Else
{
    $emailquery = "";
    for ($i=0; $i<=(count($email)-1); $i++)
    {
        If ($emailquery==""){$emailquery=" email =
'". $email[$i]. "'";}
        Else {$emailquery = $emailquery." OR email =
'". $email[$i]. "'";}
    }
}

$status = "createuser";
$title = "Create an account";
$pass = "*";
global $usercountry, $userlanguage, $usertimezone, $usernickname,
$nicknames, $emailid;
If ($emails > 0)
{
    // Checking if there is already an account with any of the emails
in the certificate.
    $result = mysql_query("SELECT * FROM user WHERE ".$emailquery) or
die(mysql_error());
    while($row = mysql_fetch_array($result))
    {
        $status = "updateuser";
        $title = "Modify your account";
        $pass = "";
        $fullname = $row['fullname'];
        If (isset($row['nickname'])){$usernickname =
$row['nickname']; $xtpl->assign('nickname', $row['nickname']);}
        If (isset($row['gender']) && $row['gender'] =='M'){ $xtpl-
>assign('gendermselected', ' selected="true"');}
        ElseIf (isset($row['gender']) && $row['gender']
=='F'){ $xtpl->assign('genderfselected', ' selected="true"');}

```

```

        Else {$xtpl->assign('nogenderselected', '
selected="true"');}
        If (isset($row['email']) && $row['email']
<>''){$emailid=$row['email']; $xtpl->assign('emailid', $row['email']);}
        If (isset($row['dob']) && $row['dob'] <>''){$xtpl-
>assign('dob', $row['dob']);}
        If (isset($row['postcode']) && $row['postcode'] <>''){$xtpl-
>assign('postcode', $row['postcode']);}
        If (isset($row['country'])){$usercountry = $row['country'];}
        If (isset($row['language'])){$userlanguage =
$row['language'];}
        If (isset($row['timezone'])){$usertimezone =
$row['timezone'];}
        $xtpl->parse('main.modifyuser.delete');
    }
}
If (!isset($usernickname) || $usernickname == ''){$query = '';}
Else {$query = " WHERE nickname != '". $usernickname. "'";}
$result = mysql_query("SELECT nickname FROM user ".$query);
while($row = mysql_fetch_array($result))
{
    If
($nicknames==''){$nicknames="'" .strtolower($row['nickname'])."'";}
    Else
{$nicknames=$nicknames.",'" .strtolower($row['nickname'])."'";}
}
$nicknames = "'" . $nicknames. "'";
$xtpl->assign('nicknames', $nicknames);

If ($emails > 1)
{
    for ($i=0; $i<=(count($email)-1); $i++)
    {
        $xtpl->assign('email', $email[$i]);
        $xtpl->assign('emailnr', $i);
        If ($email[$i] == $emailid){$xtpl->assign('emailselected', '
selected=true');}
        Else {$xtpl->assign('emailselected', '');}
        $xtpl->parse('main.modifyuser.multiplemail.selectemail');
    }
    $xtpl->parse('main.modifyuser.multiplemail');
}

// Setting $notime and $nolanguage variables as globals
global $notime, $nolanguage;
$counter = 0;
// Retrieving countrycodes that can not be found in the list of
timezones
$result = mysql_query("SELECT * FROM country WHERE id NOT IN (SELECT
country FROM timezone)");
while($row = mysql_fetch_array($result))
{
    $notime[$counter]=$row['id'];
    $counter = $counter + 1;
}

$counter = 0;
// Retrieving countrycodes that can not be found in the list of
languages
$result = mysql_query("SELECT * FROM country WHERE id NOT IN (SELECT
id FROM language)");
while($row = mysql_fetch_array($result))

```



```

{
    $nolanguage[$counter]=$row['id'];
    $counter = $counter + 1;
}

// Retrieving and parsing the list of countries
$result = mysql_query("SELECT * FROM country");
while($row = mysql_fetch_array($result))
{
    If (isset($usercountry) && $usercountry == $row['id']){$xtpl-
>assign('countryselected', ' selected=true');}
    Else {$xtpl->assign('countryselected', '');}
    $xtpl->assign('country', $row['id']);
    $xtpl->assign('countryname', $row['name']);
    $xtpl->assign('countryupdate', checkcountry($row['id'], $notime,
$nolanguage));
    $xtpl->parse('main.modifyuser.selectcountry');
    $xtpl->parse('main.modifyuser.selectedcountry');
}
If (!isset($usercountry) || $usercountry == ''){$xtpl-
>assign('nocountryselected', ' selected=true');}
Else {$xtpl->assign('nocountryselected', '');}

// Retrieving and parsing the list of languages
$result = mysql_query("SELECT * FROM language");
while($row = mysql_fetch_array($result))
{
    If (isset($userlanguage) && $userlanguage == $row['id']){$xtpl-
>assign('languageselected', ' selected=true');}
    Else {$xtpl->assign('languageselected', '');}
    $xtpl->assign('language', $row['id']);
    $xtpl->assign('languagename', $row['name']);
    $xtpl->parse('main.modifyuser.selectlanguage');
}
If (!isset($usercountry) || $usercountry == ''){$xtpl-
>assign('nocountryselected', ' selected=true');}
Else {$xtpl->assign('nocountryselected', '');}

// Retrieving and parsing the list of timezones
$result = mysql_query("SELECT * FROM timezone ORDER BY timezone");
while($row = mysql_fetch_array($result))
{
    If (isset($usertimezone) && $usertimezone ==
$row['timezone']){$xtpl->assign('timezoneselected', ' selected=true');}
    Else {$xtpl->assign('timezoneselected', '');}
    $xtpl->assign('timezone', $row['timezone']);
    $xtpl->assign('tcountry', $row['country']);
    $xtpl->assign('tcountrynr', $row['countrynr']);
    $xtpl->parse('main.modifyuser.selecttimezone');
}
If (!isset($usertimezone) || $usertimezone == ''){$xtpl-
>assign('notimezoneselected', ' selected=true');}
Else {$xtpl->assign('notimezoneselected', '');}

If (!isset($destination)){$destination="";}
If (!isset($state)){$state="";}
extension_invoke_all('user_modify', $destination, $state);
// If a user is not accessing the page through https he must be forced
to do so while logging on with a certificate.
$xtpl->assign('rootdir', httplocation());
$xtpl->assign('srootdir', httpslocation());
$xtpl->assign('status', $status);

```

```

$xtpl->assign('fullname', $fullname);
$xtpl->assign('title', $title);
$xtpl->assign('enterpass', $pass);
$xtpl->assign('identityurl', CACERTID_IDENTITY_URL);
$xtpl->assign('page_class', 'dialog-page');
$xtpl->assign('nonce', htmlspecialchars($nonce, ENT_QUOTES, 'UTF-8'));

$xtpl->parse('main.modifyuser');
$xtpl->parse('main.framekiller');
$xtpl->parse('main');
$xtpl->out('main');
// Closing the database connection
dbdisconnect();
}

```

user_create ()

```

// Insert a row in the user table
function user_create()
{
    // Start database connection
    dbconnect();
    // Checking if there is a name amongst the post data
    If (!isset($_POST['nickname']) || $_POST['nickname'] == '')
    {
        user_modify();
        exit("No nickname has been received");
    }
    // Checking if there is an email address amongst the post data
    If (!isset($_POST['email']) || $_POST['email'] == '')
    {
        user_modify();
        exit("No email address has been received");
    }
    // hashing the password if set to MD5
    If (isset($_POST['pass']) && $_POST['pass'] <>
''){$_POST['pass']=MD5($_POST['pass']);}
    $values = "('" . $_POST['nickname'] . "',
'" . $_POST['pass'] . "', '" . $_POST['email'] . "', '" . $_POST['name'] . "', '" . $_POST['
dob'] . "', '" . $_POST['gender'] . "', '" . $_POST['postcode'] . "', '" . $_POST['country
'] . "', '" . $_POST['language'] . "', '" . $_POST['timezone'] . "')";
    mysql_query("INSERT INTO user (nickname, pass, email, fullname, dob,
gender, postcode, country, language, timezone) VALUES ".$values) or
die(mysql_error());
    header('Location: '.httplocation().'index.php');
    // Closing the database connection
    dbdisconnect();
}

```

user_update ()

```

// Updating the row in the user table
function user_update()
{
    // Start database connection
    dbconnect();
    global $nr, $update;
    $nr = 0;
    // Validating post variables
    If (isset($_POST['nickname']) && !$_POST['nickname'] == ''){$update =
"nickname = '" . $_POST['nickname'] . "'";}
    Else {user_modify();exit("No nickname has been received");}
    If (isset($_POST['email']) && $_POST['email'] <>

```

```

    '$update=$update.', email = '".$_POST['email'].''";}
    If (isset($_POST['pass']) && $_POST['pass'] <> ''){$update=$update.",
pass = '".MD5($_POST['pass']).''";}
    If (isset($_POST['name']) && $_POST['name'] <> ''){$update=$update.",
fullname = '".$_POST['name'].''";}
    $update=$update.", dob = '".$_POST['dob'].''";
    $update=$update.", gender = '".$_POST['gender'].''";
    $update=$update.", postcode = '".$_POST['postcode'].''";
    $update=$update.", country = '".$_POST['country'].''";
    $update=$update.", language = '".$_POST['language'].''";
    $update=$update.", timezone = '".$_POST['timezone'].''";

    mysql_query("UPDATE user SET ".$update." WHERE email =
'".$_POST['account'].''");
    header('Location: '.httplocation().'/index.php');
    // Closing the database connection
    dbdisconnect();
}

```

user_delete ()

```

// Delete a user from the database
function user_delete()
{
    // Start database connection
    dbconnect();
    If (isset($_POST['account']) || $_POST['account'] != '')
    {
        mysql_query("DELETE FROM user WHERE
email='".$_POST['account'].''");
        header('Location: '.httplocation().'index.php');
    }
    Else
    {
        header('Location: '.httpslocation().'index.php?q=modifyuser');
    }
    // Closing the database connection
    dbdisconnect();
}

```

template.xtpl

Altered Functions

We have added a few scripts that verify form data. We've placed these scripts in a separate file which we filled with the following line of code.

```

<!-- Added javascript for logging on with certificate -->
<script src="html/cacert.js" type="text/javascript"></script>

```

Login

We have modified the login part of the website template. We created a button for logging on with a certificate. We also added a link to create or modify an account.

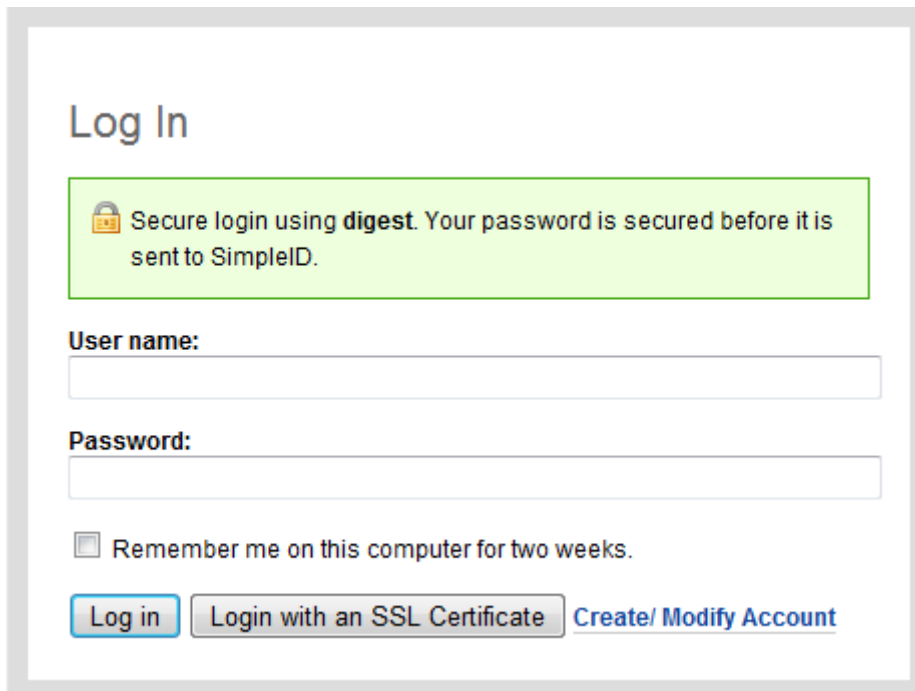


Figure C.1: Added button and link

```

<!-- Start of alteration to the template for account login -->
  <!-- BEGIN: login -->
    <div class="login-security {security_class}">
      <p>{security_message}</p>
    </div>

    <form action="{rootdir}/index.php" method="post"
  enctype="application/x-www-form-urlencoded" id="login-form" name="login-
  form">
      <input type="hidden" name="q" value="login"/><input
  name="destination" type="hidden" value="{destination}"/>
      <input type="hidden" name="nonce" id="edit-nonce"
  value="{nonce}" />

      <input type="hidden" name="digest" id="edit-digest" value=""
  />

      <div class="form-item">
        <label for="edit-name">User name:</span></label>
        <input type="text" maxlength="60" name="name" id="edit-
  name" size="60" value="" class="form-text required" {security_disabled}
  />
      </div>
      <div class="form-item">
        <label for="edit-pass">Password:</label>
        <input type="password" name="pass" id="edit-pass"
  size="60" class="form-text required" {security_disabled} />
      </div>
      <div class="form-item">
        <label class="option">
          <input type="checkbox" name="autologin" value="1" />

```

```

Remember me on this computer for two weeks.
    </label>
</div>
<input type="submit" name="op" id="edit-submit" value="Log
in" class="form-submit" {security_disabled} />
<!-- BEGIN: state -->
    <input type="submit" name="op" id="edit-cancel"
value="Cancel" class="form-submit" />
    <input type="hidden" name="s" value="{state}"/>
<!-- END: state -->
<!-- Added button for logging on with a certificate -->
<input type="submit" name="op" value="Login with an SSL
Certificate" onclick="ssllogin('{srootdir}');" {security_disabled} >
<!-- Added link for modifying and creating an account --
>
    <a href="{srootdir}/index.php?q=modifyuser">Create/ Modify
Account</a>
    </form>
<!-- END: login -->
<!-- End of alteration to the template for account login -->


```

New Functions

Modify user

As described above, we have added a link on the login page to "Create/Modify Account". We have created a page that makes it possible to create or modify your account. The code that we have created, is described after a screenshot of the "Modify your account" page.

Modify your account

 Connection uses HTTPS

Nickname:

Password:

Full name:

Email:

Date of birth (dd-mm-yyyy):

Gender:

ZIP/ Postcode:

Country:

Language:

timezone:

Figure C.2: Modify your account

```

<!-- Start of addition to the template for modifying/ creating an account
-->
  <!-- BEGIN: modifyuser -->
    <div class="login-security {security_class}">
      <p>{security_message}</p>
    </div>

    <form action="{srootdir}/index.php" method="post"
enctype="application/x-www-form-urlencoded" id="user-form" name="user-
form">
      <input type="hidden" name="q" id="q" value="{status}"/>
      <input type="hidden" name="nonce" id="edit-nonce"
value="{nonce}" />
      <input type="hidden" name="digest" id="edit-digest"
value="" />
      <input type="hidden" name="account" id="account"
value="{emailid}" />
      <div class="form-item">
        <label for="edit-nickname">Nickname: *
{identityurl}</label>
        <input type="text" name="nickname" id="edit-
nickname" maxlength="60" size="60" value="{nickname}"
onchange="checknickname();" class="form-text required" {security_disabled}

```

```

/>
        </div>
        <div class="form-item">
            <label for="edit-pass">Password:
{enterpass}</label>
            <input type="password" name="pass" id="edit-pass"
maxlength="60" size="60" class="form-text required" {security_disabled} />
        </div>
        <div class="form-item">
            <label for="edit-name">Full name: *</span></label>
            <input type="hidden" name="name" id="edit-name"
value="{fullname}" />
            <input type="text" maxlength="60" size="60"
value="{fullname}" class="form-text required" disabled="disabled"/>
        </div>
        <div class="form-item">
            <label for="edit-email">Email: *</label>
            <!-- BEGIN: oneemail -->
            <input type="hidden" name="email" id="edit-email"
value="{email}" />
            <input type="text" value="{email}" maxlength="60"
size="60" class="form-text required" disabled="disabled"/>
            <!-- END: oneemail -->
            <!-- BEGIN: multiemail -->
            <select name="email" id="edit-email"
{security_disabled}>
                <!-- BEGIN: selectemail -->
                <option id="e_{emailnr}" name="e_{emailnr}"
value="{email}" {emailselected}>{email}</option>
                <!-- END: selectemail -->
            </select>
            <!-- END: multiemail -->
        </div>
        <div class="form-item">
            <label for="edit-dob">Date of birth (dd-mm-
yyyy):</label>
            <input type="text" name="dob" id="edit-dob"
maxlength="10" size="60" value="{dob}" onchange="checkdob();" class="form-
text required" {security_disabled} />
        </div>
        <div class="form-item">
            <label for="edit-gender">Gender:</label>
            <select name="gender" id="edit-gender"
{security_disabled}>
                <option value="" {nogendersselected}></option>
                <option
value="M" {gendermselected}>Male</option>
                <option
value="F" {genderfselected}>Female</option>
            </select>
        </div>
        <div class="form-item">
            <label for="edit-postcode">ZIP/ Postcode:</label>
            <input type="text" name="postcode" id="edit-
postcode" maxlength="10" size="60" value="{postcode}" class="form-text
required" {security_disabled} />
        </div>
        <div class="form-item">
            <label for="edit-country">Country:</label>
            <select name="country" id="edit-country"
onchange="countryupdate();" {security_disabled}>
                <option id="c 00" name="c 00"

```



```

value=""{nocountryselected}></option>
        <!-- BEGIN: selectcountry -->
        <option id="c_{country}" name="c_{country}"
value="{country}"{countryselected}>{countryname}</option>
        <!-- END: selectcountry -->
    </select>
</div>
<div class="form-item">
    <label for="edit-language">Language:</label>
    <select name="language" id="edit-language"
{security_disabled}>
        <option id="l_00"
value=""{nolanguageselected}></option>
        <!-- BEGIN: selectlanguage -->
        <option id="l_{language}"
value="{language}"{languageselected}>{language}</option>
        <!-- END: selectlanguage -->
    </select>
</div>
<div class="form-item">
    <label for="edit-timezone">timezone:</label>
    <select name="timezone" id="edit-timezone"
{security_disabled}>
        <option id="t_00"
value=""{notimezoneselected}></option>
        <!-- BEGIN: selecttimezone -->
        <option id="t_{tcountry}{tcountrynr}"
value="{timezone}"{timezoneselected}>{timezone}</option>
        <!-- END: selecttimezone -->
    </select>
</div>
    <input type="button" name="op" id="edit-submit"
value="{title}" class="form-submit" onclick="return validateuserform();"
{security_disabled} />
    <!-- BEGIN: delete -->
    <input type="submit" name="op" id="edit-submit"
value="Delete your account" class="form-submit" onclick="return
deleteuser();" {security_disabled} />
    <!-- END: delete -->
</form>
    <input type="hidden" name="nicknames" id="nicknames"
value={nicknames} />
    <input type="hidden" name="cu_00" id="cu_00" value="" />
    <!-- BEGIN: selectedcountry -->
    <input type="hidden" name="cu_{country}" id="cu_{country}"
value="{countryupdate}" />
    <!-- END: selectedcountry -->
    <!-- END: modifyuser -->
<!-- End of addition to the template for modifying/ creating an account --
>

```

New Files

cacert.js

This file contains the functions that are used to verify form data when a user creates or modifies his account.

ssllogin (x)

If the user clicks on the button “Login with an SSL Certificate” on the login page, the function `ssllogin(x)` will be triggered.

```
// login with a certificate
function ssllogin(x)
{
    document.forms["login-form"].action = x;
}
```

countryupdate ()

When a user selects his country the timezone and language for that country will be automatically selected. There are exceptions due to countries not having their own language or having multiple timezones.

```
// Select timezone and language automatically after the user selected his
country
function countryupdate ()
{
    var x = document.getElementById("edit-country").value;
    var y = document.getElementById("cu_"+x).value;
    if (y=="a")
    {
        document.getElementById("l_00").selected=true;
        document.getElementById("t_00").selected=true;
    }
    else if (y=="b")
    {
        document.getElementById("l_"+x).selected=true;
        document.getElementById("t_00").selected=true;
    }
    else if (y=="c")
    {
        document.getElementById("l_00").selected=true;
        document.getElementById("t_"+x+"1").selected=true;
    }
    else if (y=="d")
    {
        document.getElementById("l_"+x).selected=true;
        document.getElementById("t_"+x+"1").selected=true;
    }
}
```

checknickname ()

This function checks if the nickname is unique. It also checks if there are illegal characters or a space in the nickname.

```
// Check if nickname is unique
function checknickname()
{
    var x = document.getElementById("edit-nickname").value;
    var y = document.getElementById("nicknames").value;
    x = x.toLowerCase();
    y = y.toLowerCase();
    if (y.indexOf("'" + x + "'") >= 0)
    {
        document.getElementById("edit-nickname").value = '';
        alert('The nickname you have chosen is already in use!');
    }
    else if ((x.indexOf("'") >= 0) || (x.indexOf('"') >= 0) ||
(x.indexOf("+") >= 0) || (x.indexOf("=") >= 0) || (x.indexOf("(") >= 0) ||
(x.indexOf(")") >= 0) || (x.indexOf("\\") >= 0) || (x.indexOf("/") >= 0) ||
(x.indexOf(",") >= 0) || (x.indexOf(".") >= 0) || (x.indexOf("!") >= 0) ||
(x.indexOf("@") >= 0) || (x.indexOf("#") >= 0) || (x.indexOf("$") >= 0) ||
(x.indexOf("%") >= 0) || (x.indexOf("^") >= 0) || (x.indexOf("&") >= 0) ||
(x.indexOf("*") >= 0) || (x.indexOf("{") >= 0) || (x.indexOf("}") >= 0) ||
(x.indexOf("[") >= 0) || (x.indexOf("]") >= 0) || (x.indexOf("|") >= 0) ||
(x.indexOf("`") >= 0))
    {
        document.getElementById("edit-nickname").value = '';
        alert('The nickname you chose contains illegal characters. (\' \' "
+ = ! @ # $ % ^ * ( ) { } [ ] | \ / . , `')');
    }
    else if ((x.indexOf(" ") >= 0))
    {
        document.getElementById("edit-nickname").value = '';
        alert('Please do not use spaces in your nick name. You can use -
and _ instead!');
    }
}
}
```

checkdob ()

This function checks if the date of birth is valid. We use the following date of birth registration “dd-mm-yyyy”. So this means that if a user enters “mm-dd-yyyy”, it isn’t valid when the month is bigger than 12.

```
// Checking if the date of birth is valid
function checkdob()
{
    var x = document.getElementById("edit-dob").value;
    if (isDate(x))
    {
    }
    else
    {
        document.getElementById("edit-dob").value = '';
    }
}
}
```

isDate (dateStr)

This function checks if the given date is valid.

```
// Checking if the date is valid
function isDate(dateStr)
{
    var datePat = /^(\d{1,2}) (-) (\d{1,2}) (-) (\d{4})$/;
    var matchArray = dateStr.match(datePat); // is the format ok?

    if (matchArray == null)
    {
        alert("Please enter date as dd-mm-yyyy.");
        return false;
    }
    month = matchArray[3]; // parse date into variables
    day = matchArray[1];
    year = matchArray[5];

    if (month < 1 || month > 12) // check month range
    {
        alert("Month must be between 1 and 12.");
        return false;
    }
    if (day < 1 || day > 31)
    {
        alert("Day must be between 1 and 31.");
        return false;
    }
    if ((month==4 || month==6 || month==9 || month==11) && day==31)
    {
        alert("Month "+month+" doesn't have 31 days!");
        return false;
    }
    if (month == 2) // check for february 29th
    {
        var isleap = (year % 4 == 0 && (year % 100 != 0 || year % 400 ==
0));
        if (day > 29 || (day==29 && !isleap))
        {
            alert("February " + year + " doesn't have " + day + "
days!");
            return false;
        }
    }
    return true; // date is valid
}
```

deleteuser ()

If the user wants to delete his account and clicks on the button “Delete your account” on the “Modify your account” page, this function will trigger the function.

```
// Alter the form query so the user will be deleted
function deleteuser()
{
    var r=confirm("Are you sure you want to delete your account?");
    if (r==true)
    {
        document.getElementById("q").value = 'deleteuser';
        return true;
    }
    else
    {return false;}
}
```

validateuserform ()

When the user clicks to modify, this script will validate the required fields.

```
// Validate the attributes that were filled in by the user
function validateuserform()
{
    var user = document.getElementById("edit-nickname").value;
    var pass = document.getElementById("edit-pass").value;
    var action = document.getElementById("q").value;
    if (user=='')
    {
        alert("Please enter a nickname before continuing");
        return false;
    }
    else if(pass=='')
    {
        if(action=='createuser')
        {
            alert("Please enter a password before continuing");
            return false;
        }
    }
    else
    {return true;}
}
```

cacert.inc

httplocation ()

SIMPLEID_BASE_URL is a superglobal that is set in config.inc

```
// Retrieve HTTP location
function httplocation()
{
    // The default http location
    return(SIMPLEID_BASE_URL);
}
```

httpslocation ()

This function converts the SIMPLEID_BASE_URL superglobal from an http to an https address.

```
// Retrieve HTTPS location
function httpslocation()
{
    return(substr_replace(SIMPLEID_BASE_URL, 's', 4, 0));
}
```

checkcountry (\$x, \$y, \$z)

The returned values in this script are used in a javascript function that will automatically select the language and timezone after selecting the country.

```
// Checks whether the the countrycode can be found in the list of
timezones and languages
function checkcountry($x, $y, $z)
{
    if ((isset($y) && in_array($x, $y)) && (isset($z) && in_array($x,
    $z)))
    {
        $cu="a";
    }
    elseif (isset($y) && in_array($x, $y))
    {
        $cu="b";
    }
    elseif (isset($z) && in_array($x, $z))
    {
        $cu="c";
    }
    else
    {
        $cu="d";
    }
    return($cu);
}
```

mysql.inc

These are some standard functions for connecting and closing the connection to the database.

dbconnect ()

```
-----  
// Connecting to the mysql database  
function dbconnect()  
{  
    $con = mysql_connect(CACERTID_MYSQL_SERVER, CACERTID_MYSQL_USER,  
CACERTID_MYSQL_PASSWORD);  
    if (!$con)  
    {  
        die('Could not connect: ' . mysql_error());  
    }  
    mysql_select_db(CACERTID_MYSQL_DATABASE, $con);  
}  
-----
```

dbdisconnect ()

```
-----  
// Closing the mysql database connection if its still open.  
function dbdisconnect()  
{  
    if (isset($con))  
    {  
        mysql_close($con);  
    }  
}  
-----
```