Identification and Authentication in Networks enabling Single Sign-On

Thomas Gert Roessler
thomas.roessler@iaik.at

Institute for Applied Information
Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a
A-8010 Graz, Austria

Diploma Thesis

supervised by
O.Univ.-Prof. Dr. Reinhard Posch

October, 2002

I hereby declare that the work presented in this thesis is my own work and that to the best of my knowledge it is original except where indicated by reference to other authors.

Thomas Gert Roessler

Acknowledgements

The knowledge this thesis is based on has been conducted at the Institute for Applied Information Processing and Communications (IAIK) of the Graz University of Technology. Therefore, I would like to thank all people of the institute who instructed me in various topics of IT Security. Furthermore, I would like to thank Arno Hollosi who gave me a very interesting insight into the concept and into the difficulties in context with the Austrian e-Government, and Herbert Leitold for his great support of my work. In particular, I would like to thank Andrea Pfundner who was a valuable help by polishing my English knowledge.

Abstract

Identification and authentication is every days business for users in todays Internet. Therefore, every user has to tackle with an increasing number of usernames and passwords. Usually every username and password belongs to an isolated account which can be considered as a security domain. As a result, users are stressed with handling so many credentials. On the other hand, to realize the cooperation of services located in different domains there is a need to have some authentication mechanism which does not require the user to authenticate and enter his user credentials several times. Single Sign-On is used therefore. In this thesis, the most important Single Sign-On solutions in use are discussed depending on their architecture (centralized or federated). In particular Microsoft Passport and the Liberty Project are viewed closely. As Single Sign-On systems allow to establish trust relationships between several independent security domains, the question arises how to treat trust in a technical manner. Therefore, one chapter in this work presents a possible approach to calculate trust in a situation of interoperating security domains. Especially in e-government applications—where trustworthiness is a major issue—the determination of trust and the possibilities to react are crucial. The practical part of this thesis consists of the analysis and the realization of some main components of a federated Single Sign-On system based on the specifications published by the Liberty Alliance. Therefore, the last section of the thesis demonstrates a possible architecture for such a system based on technologies like SAML (Security Assertion Markup Language).

Keywords: Single Sign-On, MS Passport, Kerberos, Project Liberty, Trust Algebra, SAML

Abstract

Identifikation und Authentisierung ist heutzutage tägliches Brot für Internet-Anwender. Eine Vielzahl von Benutzernamen und Passwörtern müssen vom Anwender gemerkt oder sicher verwaltet werden. Jedes der Benutzernamen-Passwort-Paare gehört zu einem meist isoliertem Benutzerkonto, das in einer meist unabhängigen Domäne angesiedelt ist. Um den Anwender einerseits von der Flut von Kontodaten zu befreien und es andererseits zu ermöglichen, dass die verschiedenen Services aus den einzelnen Sicherheitsdomänen zusammenwirken können ohne dass der Anwender unzählige Male aufgefordert wird einen seiner Namen und Passwörter anzugeben, werden sogenannte Single Sign-On Lösungen eingeführt. Am Beginn dieser Arbeit werden die gängigsten Single Sign-On Lösungen vorgestellt und anhand deren Architektur unterschieden (zentralisiert oder verteilt). So wird zum Beispiel Microsoft Passport und das Project Liberty detailliert betrachtet. Eine der Möglichkeiten die Single Sign-On bietet ist das Zusammenwirken mehrerer vormals unabhängiger Domänen in Sachen Authentisierung. Deshalb stellt sich die Frage, wie in diesem Zusammenhang technisch gesehen die Vertrauenswürdigkeit bewertet und berechnet werden kann. Im zweiten Teil der Arbeit wird deshalb eine Metrik vorgestellt die dies und darüber hinaus auch die Bewertung verketteter Situationen ermöglicht. Dies ist besonders in sensiblen Bereichen wie dem e-Government wichtig. Einige der in diesen Kapiteln vorgestellten Massnahmen und Vorgehensweisen zielen speziell auf solche Anwendungsbereiche ab, in denen auch Datenschutz eine wesentliche Anforderung ist. Zum Abschluss wird in einem praktischen Teil die Realisierung der wichtigsten Komponenten eines Single Sign-On Systems, basierend auf den Liberty Alliance Spezifikationen und unter Verwendung von Technologien wie SAML (Security Assertion Markup Language), gezeigt und deren Architektur demonstriert.

Keywords: Single Sign-On, MS Passport, Kerberos, Project Liberty, Trust Algebra, SAML

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

Identification and authentication are a big topic, especially in systems of federated services and entities. Therefore, there exist many different requirements depending on the properties of the service where user authentication is needed. Of course, identification and authentication are also a considerable threat for the privacy of a user and his behavior. This is why it is crucial to be careful with identifying an entity everywhere and for any reason. On the other hand, having a great number of different user accounts for one single person is not very comfortable regarding convenience. For this aspect it would be a good practice to initiate only one or a few accounts which are used to log-in at all services. Furthermore, it is practicable to log in at one portal only once and automatically all other related services are useable for the authenticated user. Such a system is called a Single-Sign-On solution. Depending on the technical realization working in the background, these systems could be split up into centralized or federated systems. Each of them has some advantages and also some weak-points, especially under the aspect of harming users' privacy.

1.1 Motivation

At the beginning of working on this thesis there was the problem of user authentication within an e-governmental scenario. In e-government various autonomous services are cooperating in order to fulfill some task which may be initiated by a citizen's request. The idea was that a citizen who enters such a request has to be identified and authenticated only once and all cooperating services rely on this initial authentication. As this thesis points out in a later chapter relying on an authentication result unconditionally is not sufficient, especially in such a sensitive environment as e-government. In the course of the investigations on this topic, the possibilities of how to introduce *trust* in such an environment were examined. Furthermore, by working through the possibilities an applicable way to grasp trust and to determine the trustworthiness of such an authentication result, also called assertion, was gathered. One part of this work presents a mathematical framework and a possible algebra to calculate this. Within this part, also possible reactions on the situation of distrusting an assertion were developed. Thus, mainly some re-authentication mechanisms are shown as well. Moreover, the risks of harming a user's privacy by calculating

the trustworthiness of an assertion are pointed out because keeping the privacy of a user is mandatory for all tasks and applications in this context.

On the other hand, various existing Single Sign-On solutions were surveyed, e.g. Microsoft Passport, Kerberos, Project Liberty. Furthermore, the approach of the Liberty Alliance and of their Project Liberty Single Sign-On solution fulfills the needs of the requirements given by the introductional problem most likely. Thus, in the practical part of this work a prototype of a framework enabling Project Liberty functionalities were developed in order to gain a deeper understanding of the Liberty Alliance specifications.

1.2 Introduction to Single Sign-On

The Open Group defines Single Sign-On as [24]:

> *a mechanism whereby a single action of user authentication and authorization can permit a user to access all computers and systems where that user has access permission, without the need to enter multiple passwords*

In this chapter, this wide-range definition is focused on domains, services and applications situated in the web. In this means, a Single Sign-On system (SSO) supports the user with the advantage of accessing several sites and services within multiple security domains after authenticating himself only once. The first log-in happens preferably at a portal-site. From this point, other sites and other security domains—where normally a separate sign-on is necessary—obtain the user's credentials by contacting the portal-site used by the client. Furthermore, user profiles may be supported too by such a portal-service. Without any further action from the client every service and web-page will be presented according to the user's preferences stored in his profile.

The following chapters give an introduction for such systems and they differ between centralized and federated solutions for these systems, depending on their architecture. The following sections give also a short survey over the most important standards and products for Single Sign-On solutions.

Users normally have to sign-on to several distributed systems where user accounts are needed. Many of them are induced for the reason of convenience, but an increasing number of accounts is used to access services with higher security restrictions. Recently, distributed systems were built up in a way that they act independently as isolated security domains. Therefore, each user who wants to enter one of these domains has to authenticate himself at each of these security domains by the use of separate credentials (figure 1.1).

Therefore, an end-user who wants to use services and accounts housed in different domains has to sign-on multiple times. This results in the difficulty that users today have to remember much more than one username and about the same number of different passwords. Moreover, because of duplicate names a client is used to select different user names for his accounts. Therefore, up to now it is necessary either to write the different user data into a secret book,

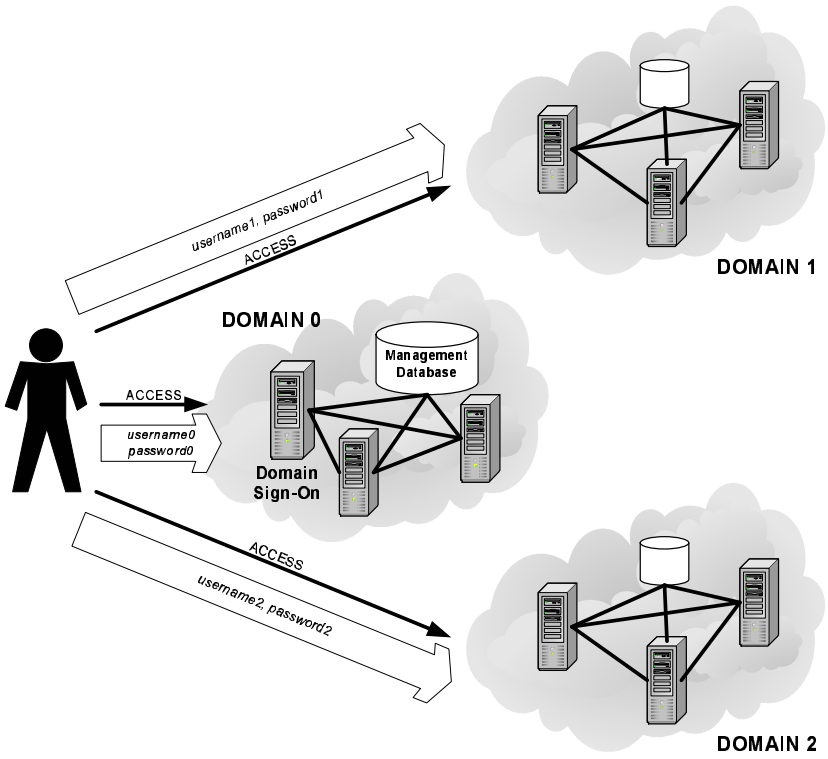

Figure 1.1: Multiple domains require multiple usernames and passwords

or to have an encrypted file or a kind of account-manager on your computer to handle the various account-information.

A new approach to tackle this problem is to introduce a Single Sign-On mechanism in all different security domains. The introduction of Single Sign-On into these security domains increases the usability and the security as well. Furthermore, the effort for the management of user accounts may be reduced. A system which provides an integration and coordination of different accounts, of this kind brings some benefits for users and providers. For example:

- reduction of time exposure for users if users have to sign-on to different domains
- improved usability by reducing the number of account information that a user has to remember

Using this system enables an end-user to access other, secondary security domains after signing-on to a primary domain. Between these domains there exists a trust relationship. Therefore, the second domain obtains user credentials through the sign-on service from the first domain where the user is already authenticated. In other words, the primary security domain supports the other domains by the user's credentials assumed that the user is logged in correctly

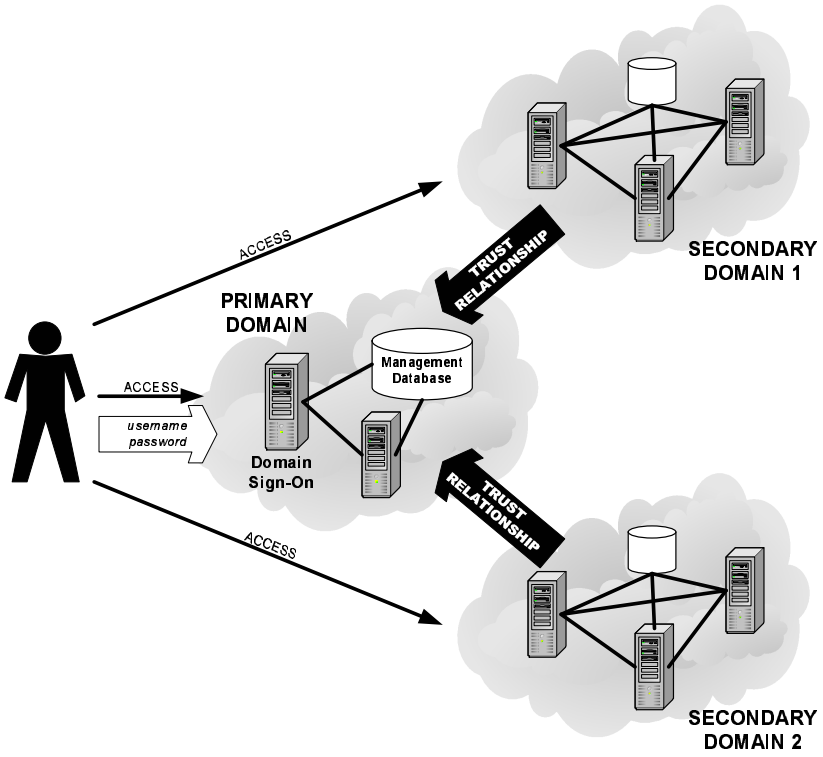

Figure 1.2: Using SSO only one username and password is required

(as depicted in figure 1.2). Depending on the architecture there are mainly two different approaches to SSO-systems:

- centralized architecture
- federated architecture

The centralized architecture uses a central authority and a central user database working behind. Each domain relying on the authentication decisions of the authority has to establish a trust-relationship with it. Contrarily the federated architecture allows to combine several independent authorities, so called identity providers. The user's identities distributed across these providers are collected in one overall identity—the so called network identity. Both of them have their advantages and disadvantages. In chapter 2 the centralized solution will be discussed in detail. In this course, the most important established standards and implementations of centralized SSO-systems are introduced. Chapter 3 describes the federated Single Sign-On technology and its standardized specifications and implementations. Especially the Liberty Alliance Project is treated very deeply.

1.3 About this Document

The result of the work initiated by the motivation stated above is condensed and summarized in this thesis which is organized as follows:

As the introduction above points out the existing solutions can be separated into centralized and federated systems. Therefore, chapter 2 goes on with an examination of systems following the centralized approach. Thus, Kerberos as one of the first attempts to enable Single Sing-On within networks is introduced. The functionalities and main components of the Kerberos model are described. On the other hand, one of the most important representatives of such systems is Microsoft Passport. That is why Passport's architecture is treated in the second section of chapter 2. To round up this introductional chapter, the last section gives a short overview of some problems concerning central Single Sign-On systems. Contrarily, chapter 3 deals with the other approach to this problem, namely the federated Single Sign-On architecture. In this chapter, the quite new Project Liberty and the Liberty Alliance itself are presented. By examining an exemplary sign-on process the basics of this architecture are figured out. At the end of chapter 3, it will be stated that this federated approach deals best with the scenarios similar to those given in the motivation.

As mentioned above, receiving an assertion based on a previously happened authentication process is not sufficient. Part II and its chapters deal with the situation of having various services cooperating in a chain. Moreover, in this scenario the first service requires an authentication only and all other involved services are relying onto the assertions of the first service. This situation is similar to the situation in a Single Sign-On system. An authentication authority asks the user to sign on and the other services are consumers of the assertions provided by the authority. These chapters assume that the trustworthiness will decrease along such a chain of services. Therefore, to cover this fact by some metric chapter 5 introduces a so called *Trust Algebra* mainly based on the work of Audon Jøsang. Furthermore, at the end of this chapter a possibility is introduced to determine the trustworthiness of such an assertion, or more generally spoken of a message, by applying a recursive calculation. In chapter 6 various ways of how to react in the case if an assertion or message is considered as not trustworthy are drafted. The eventuality of re-authentication is mainly treated. All results of this part of the thesis are considered and discussed under the aspect of a user's privacy according to the requirements of sensitive environments.

Because of that the federated approach meets the requirements given in the motivation best, the last part of this thesis covers the description of a sample implementation of some facilities given by the Liberty Alliance specifications. A main objective for this implementation is to gain a deeper understanding of this preferable technology. At first, chapter 8 gives a short motivation for this implementation. Furthermore, the Liberty Alliance specifications build up on basic technologies such as the *Security Assertion Markup Language* (SAML) and the *Simple Object Access Protocol* (SOAP). In the introduction an overview on SAML and SOAP is given as well. Chapter 9 deals with the juicy bits of this part. It introduces the chosen architecture of Java classes underlaying the implementations, enabling basic functionalities of Project Liberty (Single Sign-On

by using the Liberty Browser Artifact Profile). Finally, to round up this part of the thesis a complete Single Sign-On process is demonstrated by the use of the resulting sample implementation.

Chapter 2

Centralized Single Sign-On Systems

Due to the way identities and user-credentials are distributed and managed Single Sign-On solutions can be distinguished between centralized and federated systems. In the centralized approach user's authentication information and his profile are centrally managed, for example by an Internet portal. No matter if there are more than one portal-sites which are synchronized periodically, the core behind is still the same. One central database holds the client's authentication-, identification- and profile-information, or more generally spoken his identity-information. This approach is a very pragmatical one. Because of the maintenance and security aspect it is easier to manage one central database keeping the user data. Also, access-control depending on these data is quite comfortable and transparent. Therefore, controlling access can be done by a portal-site where every user has to be identified and authenticated with the help of his data stored in the database. On the other hand, setting up such a system implies that every participating system and security domain has to cooperate with the portal site and its primary security domain.

The first system described in this chapter is Kerberos. Kerberos is using cryptographic algorithms especially symmetric encryption to control access. After signing on once, the user gets so called tickets to prove that he was authenticated by the central authority. With the help of these tickets the user can enter a restricted area or use a protected resource. Such a ticket is always dedicated to a special receiver, e.g. a server, and it is encrypted by using just created session keys. The main application area of Kerberos are closed networks. For the Internet, Microsoft Passport is introduced which is representing a central authority as well. Similar to Kerberos, Passport holds the user's data in a central database. Gaining access to protected sites and resources is controlled by the use of cookies. This means, that the user receives some encrypted cookies after he was authenticated successfully by the Passport Sign-On server. From this point, the possession of these cookies is sufficient to gain access to the desired services and sites. The next two sections give a deeper look into the details and mechanisms used by these two systems.

2.1 Kerberos

2.1.1 Introduction

Kerberos was developed in the early eighties at the Massachusetts Institute of Technology (MIT) with the intent to allow users and services to authenticate themselves to each other unequivocally. With Kerberos, the MIT attempts to replace the common way of authentication along networks of this time, which is known as *authentication by assertion* [30]. With this, the user at first has to log in at his client-program and from this moment, the client program asserts to other services that the user is authenticated. Obviously, this solution is not very secure. Imagine that such a client programm is corrupted and is working without requesting any password authentication all services and sites inside the affected net are accessible. Consequently the idea to develop Kerberos was born. This system was designed to prevent the necessity of proving the possession of secret data, e.g. the password, by disclosing this private secret itself. This is why Kerberos works on the basis of key distribution and cryptographic operations, namely decryption and encryption. Actually, Kerberos is available in its fifth version and it is widely used and supported by many products.

2.1.2 Architecture and Functionality

Figure 2.1 illustrates the basic information flow used in Kerberos. This architecture uses the following components:

- Client
- Authentication Server (AS)
- Ticket Granting Ticket (TGT)
- Ticket Granting Service (TGS)
- Requested Service

Very often, the *Authentication Server* and the *Ticket Granting Service* are realized as one central system, called *Key Distribution Center (KDC)*. In some papers the terms Authentication Server and Ticket Granting Service are preferred, others are talking about the Key Distribution Center only. In this section, the terms Authentication Server and Ticket Granting Service are used separately.

Credentials

Generally, Kerberos uses the following two types of credentials:

- tickets
- authenticators

A ticket contains various information about the user, such as the name and the IP-address of the client. Furthermore it includes a time-stamp, a lifetime value and a random session key which is used for secure communication. Such a ticket is issued by the Authentication Server and is sent to the desired end

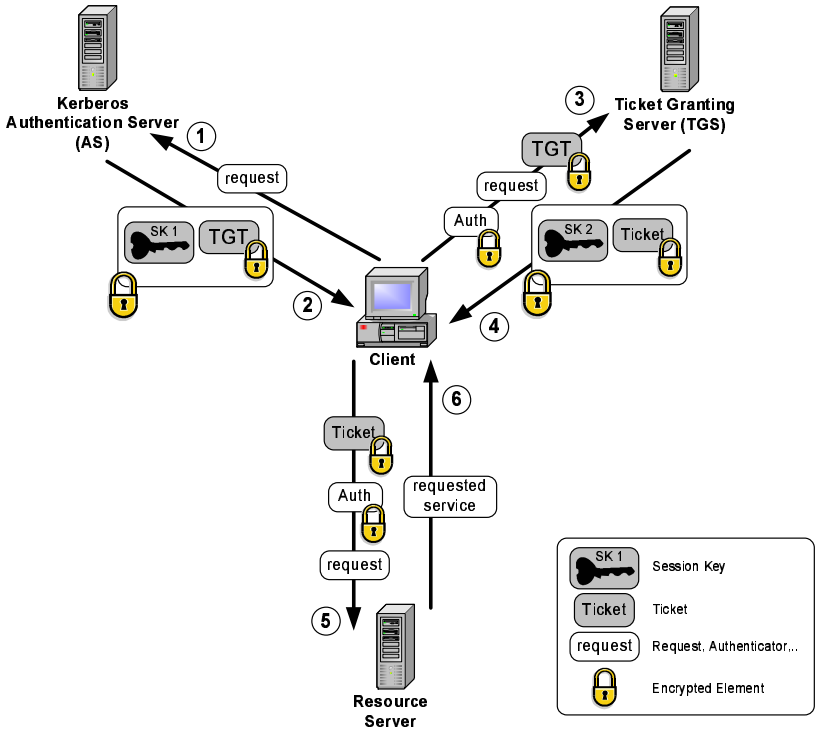

Figure 2.1: Single Sign-On with Kerberos

server, called the Resource Server. With the help of tickets the receiving server gets information about the user. They allow the server to prove whether the user representing the ticket is the user to which the ticket belongs to or not. If proving was successful, the user gains access to the desired resource. The ticket has to be encrypted by using the key of the receiving server in order to secure the content of the ticket. A ticket can be used several times depending on its lifetime.

On the other hand, the authenticator is needed to prove that the user presenting a ticket is the same one who the ticket was issued to. This can be done because the authenticator contains again the user's name and the Internet address of his workstation as well as the current system time at the user's workstation. Therefore, the authenticator is generated at the user's computer by the user's client program. By comparing this additional information with the information included in the ticket it is possible to ensure that the owner of the ticket is the same as the owner of the authenticator. Contrarily to the ticket, the authenticator cannot be used multiple times. The client has to build up a new authenticator for each usage. The authenticator has to be encrypted, namely by using the session key related to the session between the client and the service receiving the authenticator.

Kerberos keeps a client-database working in the background storing the user's

data and their associated private keys. The Authentication Server maintains this database. Kerberos is working with symmetrical keys, which means that the same key is used for both encryption and decryption. Therefore, the distribution of the secret and symmetric keys is very crucial. In practice, these keys are generated at one side of the communication channel and they are distributed using a secure media. First of all these keys are needed for user authentication when a client wants to use the Kerberos system. Based mainly on the work by [28] and [30], the following steps describe an authentication process by the use of Kerberos. For example: Alice wants to authenticate to the Authentication Server in order to get access to the Ticket Granting Server:

1. **Request at the Authentication Server**

   Alice creates a message containing her name and the current time-stamp and the request for accessing the Ticket Granting Server and sends it to the Authentication Server. The Authentication Server receives her message and tries to resolves Alice's name and to verify the time-stamp. If verification was successful and Alice is known to the Authentication Server, the server is able to get Alice's secret key from its central database.

   The Authentication Server is able to generate temporary keys, so called session keys, for symmetrical encryption in order to secure messaging between two parties. Therefore, after the user's request was proved successfully by the Authentication Server, a session key is generated which is needed for the communication between the user and the requested Ticket Granting Server. One copy of this key is sent to the user within an encrypted message using the user's secret key. Another copy of this key is dedicated to the Ticket Granting Server. The Authentication Server has to create the so called Ticket Granting Ticket for the Ticket Granting Server as well. The Ticket Granting Ticket contains the name of the user, e.g. Alice, the name of the desired Ticket Granting Server, a time-stamp to prevent reply attacks, a lifetime value defining the ticket's expiration, the client's Internet address and the random session key just having been created. At last, this ticket is encrypted whereby the key has to be known to Ticket Granting Server and the Authentication Server only.

   Kerberos supports also mutual authentication, the Authentication Server will authenticate itself to Alice by sending an encrypted message containing the server's name and a time-stamp. The included time-stamps are very important to prevent replay attacks by hackers whereby a eavesdropped authentication message is simply sent to the Kerberos service a second time. With the introduction of time-stamps, though, each entity involved in the process is able to determine the time of message creation and is able to decide about the trustworthiness of a message.

2. **Getting a Ticket Granting Ticket**

   The Authentication Server sends the ticket and the user's copy of the session key included in a message which is encrypted with the user's secret key back to the user. By using the user's secret key it is ensured that only he is able to access the content of this response. Depending on the implementation, the user's secret key may be derived from his password

which he is asked to enter by the login client on the user's workstation. Note that the Authentication Server never asks for the user's password. The Authentication Server answers to a simple request as described in the previous step with an encrypted response containing the ticket and the session key. Only the real user is able to decrypt this response after having entered his password in order to regenerate his secret key. If the response could be decrypted the sign-on client stores the received session key and the Ticket Granting Ticket for later use. From this moment on, the client's secret key (master key) is not longer needed during this session. Just with the help of this Ticket Granting Ticket, the user is allowed to access the Ticket Granting Server in order to obtain tickets needed for the access to other services. Therefore, this Ticket Granting Ticket is used to get specialized tickets during a user session. After a certain time, the Ticket Granting Ticket will expire.

3. **Accessing a Service**

   So far the user has got the Ticket Granting Ticket which allows accessing the service of the Ticket Granting Server. This server is responsible for issuing separate tickets for accessing other individual sites or services. Therefore, the user has to send a request to the Ticket Granting Server. This request has to contain the name of the desired server which the user wants to access. Furthermore, this request has to include the Ticket Granting Ticket received introductorily and an authenticator built as described above by using the previously generated session key. The Ticket Granting Server verifies the authenticator and the Ticket Granting Ticket and if both are valid, the server generates a new session key which has to be used for the communication between the client and the new desired server.

4. **Obtaining the new Ticket**

   The Ticket Granting Server generates a new ticket for the desired server. This ticket contains again the user's name, his IP address, the name of the server or service which the ticket is issued for, a time-stamp, the session key (SK 2) just created and a lifetime value which is limited by the remaining lifetime of the Ticket Granting Ticket and a predefined maximum lifetime value for the desired server. This ticket and the new session key is wrapped into a message which is encrypted by using the first session key (SK 1) that was generated previously during the first step of the procedure. The encrypted message has to be sent back to the user.

5. **Accessing the desired Server**

   With the help of the received ticket, the user is now able to access the desired server. Additionally, to prove that the user is really the one who the ticket was issued to, the client program has to generate an authenticator as mentioned earlier. This authenticator has to be encrypted by using the session key (SK 2) received together with the ticket for the desired server from the Ticket Granting Server as described before. The authenticator and the previously obtained ticket has to be sent to the desired server which decrypts the authenticator by using the session key (SK 2) that is contained inside the ticket. If the user described by the authenticator is

the same given in the ticket, the server allows the user to access the service.

Finally, if mutual authentication is desired by the client, the server has to send a response containing a new time-stamp. The value of this new time-stamp is the value of the time-stamp which was included inside the client's authenticator increased by one. This response is encrypted by using the session key (SK 2). This way, the client is ensured that the server is genuine too.

Cross Realm Authentication

The previous example is based on the scenario of having one Authentication Server and one Ticket Granting Server. In other words, the user was authenticated inside one realm only. In "The Moron's Guide to Kerberos" [30], Brian Tung states that as long as the number of users and therefore the number of requests remain low, there may be no problem considering the performance of the Authentication and Ticket Granting Server. As Tung points out, if the number of requests increases these servers would become a bottleneck in the authentication process. Therefore he summarizes, that this system does not scale, which is bad for an authentication system like Kerberos .

Dividing the network into several realms instead of having only one is an attempt to solve this problem. Furthermore, by having more than one realm authentication across the boundaries of realms is necessary. To allow this, Kerberos introduces the so-called Remote Ticket Granting Server (RTGS) into each realm to allow cross realm authentication. The authentication process described above has to be extended by one additional step. If a user wants to use a service of a foreign realm he has to contact his Authentication Server to obtain his ticket and to gain access to his local Ticket Granting Server. So far, everything is similar to a normal authentication process. To access the foreign service the user has to contact the Remote Authentication Server housed in the foreign realm. To do so, the user has to request a ticket for this Remote Ticket Granting Server from his own local Ticket Granting Server. Normally, the user obtains the ticket and with the help of this the user can access the Remote Ticket Granting Server in order to request a ticket which is needed for entering the foreign service. The only step inserted into the normal authentication process is to contact the Remote Ticket Granting Server by using a ticket obtained from the local Ticket Granting Server.

In case there are many realms it is not efficient to register each realm in every other realm. Therefore, it is advantageous to introduce a hierarchy of realms. Thus, it is sufficient to contact Remote Ticket Granting Servers in one or more intermediate realms only. Their names have to be recorded in the tickets. Despite the introduction of cross realm authentication, the Kerberos system remains not very scalable.

2.2 Microsoft Passport

2.2.1 Introduction

Today's most popular Single Sign-On solution available is Microsoft Passport. The Passport model consists of three entities:

- the client
- the merchant
- the Passport Sign-On server

Whereby the client at his browser is usually the consumer who has previously registered with the Passport service, the merchant is generally an online store. The merchants respectively the online stores wish to market with the client and the Passport Sign-On server plays the role of the central authority. This server handles the user's authentication information and his profile data, which allows him to interact with the online merchant. Furthermore, the Passport model splits up the client's data into pure profile information and the so called *wallet* containing the client's payment information such as his credit card data. For correctness, wallet itself is already an application based on Passport. Therefore, this section deals with the core of Passport only. Over all, Microsoft Passport was developed to allow Single Sign-On in the web in order to realize a basis for secure transactions in context with online shopping.

2.2.2 Architecture and Functionality

The core of Passport's architecture is a centralized database which contains all the registered users and their according data and credentials. For each user, Passport generates a unique identifier, the so-called *Passport Unique Identifier* (PUID). With the help of this, every user can be identified unambiguously.

Microsoft tries to solve the problems concerning Single Sign-On by using standard Web technologies such as Secure Socket Layer (SSL), cookies and JavaScript. Furthermore, Passport allows the administrator of the participating merchant site to choose among three possible security levels of authentication suiting the needs:

- Standard Sign-On
- Secure Channel Sign-On
- Strong Credential Sign-On

Standard Sign-On is preferably used for common application cases without extraordinary security restrictions. The Secure Channel Sign-On enhances the Standard Sign-On profile by using SSL. Strong Credential Sign-On supports the top level of security by introducing a further sign-on stage in adaptation to the Secure Channel Sign-On.

Microsoft's Single Sign-On solution never sends a user's password to the participating merchant sites. Furthermore, the user's profile information are always

sent encrypted. Common to all levels mentioned above is that they do not require the user to install any client application beside the ordinary Internet browser such as Microsoft's Internet Explorer, Netscape Navigator or Opera. On the server side, the administrator of the participating site has to install Microsoft's Passport Manager [20] which is the only additional software Passport requires. This manager is responsible for encrypting and decrypting cookies. It is used for user authentication, for handling a user's profile data, for storing the user's authentication and profile information in cookies and finally for treating cookies in order to reverify a user's authentication status respectively to manage the user's session.

A user follows a link to a web site where authentication by using Passport is required. For example, this site may be a home page of an e-shop and it allows the client to buy something online. If the user wishes to make a deal with this online shop, the user has to sign-on using a well known Passport Sign-On server. Thus, the user's browser is redirected to this Passport server in order to login. From this point, each of the three possible security levels provided by Passport differ a little bit. The following section describes the basics of Passport by explaining the sign-on procedure using the Standard Sign-On profile.

Standard Sign-On

Standard Sign-On represents the lowest security level available within the Passport model. In this profile, SSL is used only when the username and the password are transferred to the Sign-On Server. Therefore, sites and services with higher security restrictions and where the user gains access to sensitive data, e.g. account information at a home banking service, protection by the Standard Sign-On level will not be sufficient. Despite of this, to show the principles working behind Passport this section describes the authentication process (figure 2.2) of the Standard Sign-On level (according to [20]).

1. **Redirection to MS Passport Server**

   First, the client follows the sign-on link (commonly represented by a standardized Microsoft Passport symbol) at the participating merchant site to the Passport Sign-On server.

2. **Accessing the Passport Sign-On Server**

   The user follows the link and his browser is redirected to the sign-on page of the dedicated Passport server. Within this request a unique identifier is added, which identifies the participating site from where the user is redirected. This ID as well as a unique encryption key is initially assigned to a site when the site registers as a participating site. The return-URL of the requested resource at the participating site is enclosed within the request as well, in order to redirect the client's browser to the desired page when sign-on was successful.

3. **Entering the User's Credentials**

   After having checked successfully if the site ID which was sent within the request corresponds to a registered site, the Passport server asks the user to enter his username and password in order to sign-on. While there is

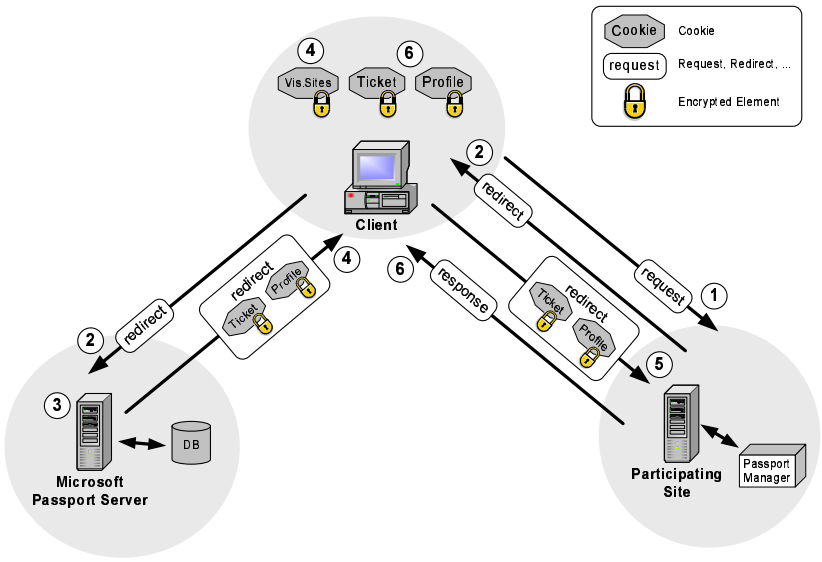

Figure 2.2: Single Sign-On with Microsoft Passport (Standard Sign-On)

no restriction to show the login page of the Passport Sign-On server by using HTTP or HTTPS, the form to enter the user's credentials is always presented by using the POST-method and by using HTTPS to send the credentials to the server. Therefore, when the user submits his data to the login server they are always transferred by using the SSL protocol as mentioned before.

4. **Obtaining the Cookies**

   The Passport Sign-On server proves the received authentication information. If they match with the information stored in the server's database, the user is considered as signed on. Furthermore, the Passport Manager at the server retrieves the user's Passport Unique Identifier and additional profile information if it is needed. With the help of this, the server's Passport Manager is able to create cookies representing the user's status. Overall, the Passport SDK documentation [21] states, that there are five cookies written into the authority domain and two more cookies written into the participating site domain. For the moment and for explaining the Passport's principle it is sufficient to deal with the following three cookies (according to MS Passport Technical Overview [20]):

   - the Ticket Cookie
     ...containing the PUID and a time-stamp
   - the Profile Cookie
     ...containing the user profile information
   - the Visited Sites Cookie
     ...containing a list of sites the user has signed on to

Each of these three most important cookies is encrypted by using the Triple Data Encoding Standard (3DES) algorithm. The key used for encryption was initially created and assigned when a site registers to participate with the Passport system. The server encrypts the content of these cookies and returns the ticket and profile data by adding them as query string to the return URL of the participating site which the user wants to access as declared inside the authentication request. Finally, the client's browser is redirected according to this return URL and the Passport Sign-On server stores the Visited Site Cookie at the client's browser.

5. **Accessing the Participating Site**

   The user's browser was redirected back to the desired site which the user wants to access by following the return URL provided by the Passport server. The Passport Manager running at the participating site's server extracts the return URL in order to obtain the containing ticket and profile information. After the information has been decrypted, the manager obtains the PUID, the time-stamp and the profile data. This way, the participating site recognizes that the user is authenticated at the Passport Sign-On server. If desired the site can be configured to enforce re-authentication after a specific time (when the time elapsed, the Passport Sign-On server's login screen is displayed again and the user is asked to re-enter his password in order to renew the cookies). As a result, the participating site stores the Ticket and the Profile Cookie containing the ticket and profile information received within the URL on the client's browser.

6. **Using the Participating Site**

   The participating site is now able to display a customized page to the client by using the profile information coming along with the Profile Cookie. Furthermore, the site can use this information to add it into its own profile-database or to create its own cookies. While displaying the desired resource to the client the participating site always supports a link to a sign-out service. This link is commonly represented by a standardized Microsoft button.

7. **Sign Out**

   As mentioned above, the user is able to sign out at every time by clicking on the sign-out symbol displayed on all involved sites. In this case the client's browser follows the link behind the symbol and it is redirected to the Passport server. After having proved the ID of the site originating the logout request, the server uses the Visited Sites Cookie in order to delete all cookies created at sign-on for the visited sites. For every cookie to delete the server launches a script at the corresponding participating site, which removes the cookie. Due to the fact that only the site which has created the cookie is able to delete it, every participating site has to provide the URL of the script which has to be called in order to delete their cookies. The registration of these URLs happens during the initial participation process. Additionally, all the Passport cookies are temporary. This means, that these cookies are deleted anyway when the browser session is closed without having signed out from the Passport system previously.

Furthermore, these cookies are time sensitive and so they expire after a certain period of time specified by the Passport server or the participating sites. After a timeout, the user is enforced to re-validate his session at the Passport server by re-entering his password.

The Passport model does not use direct communication between the Passport Sign-On server and the participating site. Instead of this, the communication is realized by the use of cookies and redirects. Only the Passport Manager installed on the server of the participating site has to retrieve a configuration file (presented by an XML document) periodically, which is stored at the Passport Sign-On server. This XML file contains current information of the URLs of the Passport Sign-On servers available and the actual Passport profile configuration.

In the case that the user is already signed-on and authenticated by the Passport server and he enters another participating site, he will be redirected once again to the Passport server by clicking onto the sign-on link represented by the Microsoft symbol. Now the client's browser is redirected to the login server by using the participating site ID and the return URL addressing the resource the user wants to access. Again, the Passport Sign-On server has to prove the incoming request namely the correctness of the site ID and it also proves the validity of the user's ticket data containing the user's PUID and the time-stamp. Afterwards, the Passport server creates new cookies and it returns the encrypted tickets and profile data to the requesting site added to the return URL as query string. Thus, it is possible to enter another site after an initial authentication at the Passport Sign-On server. It is possible, though, that the participating site requires to prove a recent authentication for higher security. In this case the Passport server is enforced to re-authenticate the user by asking his credentials again.

2.2.3 Secure Channel Sign-On and Strong Credential Sign-On

As mentioned in the introduction of this chapter MS Passport supports two other sign-on profiles suiting the needs of higher security levels. These are the Secure Channel and the Strong Credential Sign-On profile extending the Standard Sign-On profile.

Secure Channel Sign-On

As the name of this profile says already, the Secure Channel Sign-On profile extends the Standard Sign-On profile by the use of a secure end-to-end SSL channel. This SSL channel is used during the whole authentication process. If no such channel is used, an attacker is able to eavesdrop the communication between the client's browser and the Passport Sign-On server. Of course, the content of the cookies is encrypted, though, even by stealing these encrypted cookies an attacker is able to impersonate the owner of this cookies by replying them to the affected participating site. The Standard Sign-On profile is vulnerable to such a reply attack because of unprotected transport of cookies by using ordinary HTTP. To go into detail, while the attacker is eavesdropping the network traffic he is able to recognize when an encrypted cookie is sent.

The content of the cookie is not in danger because of encryption, though, the attacker can capture the cookie and can use it to send it to the participating site in behalf of the real user. As a result, the attacker gains access to the protected site for the life time of the stolen cookie. By using the SSL channel it is not possible to detect the transmission of cookies either because the whole traffic is encrypted.

Moreover, this profile defines a new ticket format as well. In order to prevent that a malicious client may change his received cookies to impersonate another user. Thus, the format used in this profile ensures that the cookies received by the Passport Server can not be manipulated without detecting the manipulation at the participating site. Shortly, the Secure Channel Sign-On is very similar to the Standard Sign-On procedure except that SSL is used to secure the communication.

Strong Credential Sign-On

Both, the Secure Channel and the Standard Sign-On profile are limiting the number of incorrect attempts of guessing the client's password during sign-on. In other words, if an attacker tries to guess the password of a user's Passport account, the Passport Sign-On server will block the affected account for a few minutes after a certain number of incorrect attempts. Despite this security mechanism, there remains a risk to crack the password.

To reduce this risk, Microsoft Passport extends the Secure Channel Sign-On profile by introducing a second stage where the user has to authenticate himself again. If a user wants to access a resource or site which is protected by the Passport's Strong Credential Sign-On profile, he has at first to sign on similarly such as it is necessary in the Secure Channel Sign-On profile. In other words, the user is asked for his Passport credentials by using a SSL secured form. If the user is authenticated successfully, he is allowed to enter the second stage of authentication. Here the Passport Sign-On server requires to enter an additional four-digit security code which the user has to choose during his initial Passport registration process. All the communication during the authentication is done through a secure SSL channel as stated in the Secure Channel Sign-On profile. Contrarily to the first stage of authentication where the user account is never blocked forever, the number of incorrect attempts to enter the additional security code is strictly limited with five. After five incorrect attempts, the Strong Credential Sign-On level is blocked until the according user generates a new security key by answering three secret questions which were chosen among ten possible questions during his registration. The counter which is responsible for blocking the account is restated after every successful sign on process. It is important to mention, that only the security key necessary for the Strong Credential Sign-On profile is blocked in the case of five incorrect attempts. The Standard Sign-On profile still remains usable for the user by using his standard Passport credentials. Therefore, sites and resources which require lower levels of Passports Sign-On are still accessible for the user. As a result, this profile supports the highest level of security a participating site can request from Passport. With this, the system is no longer vulnerable to a dictionary attack.

2.3 Risks

This section gives a short survey of some crucial risks and possibilities for attacks of such centralized Single Sign-On architectures. This section paraphrases some points out of the work of David Kormann and Aviel Rubin resulting in their paper "Risks of the Passport Single Signon Protocol" [18] which gives an interesting overview of some weak points of the Microsoft Passport solution. Furthermore, this chapter refers to the problems concerning the use of cookies as described in the article "Microsoft Passport to Trouble" [27] by Mark Slemko.

2.3.1 Attacking the Core

The characteristics common to all solutions which are built up on a central architecture are that they are very vulnerable at their central point. Therefore, Kormann and Rubin state in their paper that the reliance of all participating resources and sites onto the authoritarian decisions of one central system brings a potential target for attacks into the architecture. It seems likely that if the central authority, e.g. the Passport Sign-On server and its database, is vulnerable to denial of service attacks it cannot be guaranteed that all participating servers are accessible for users all the time. Beyond this, if a big number of merchant sites rely on the Single Sign-On system and if the central authority fails, the affected e-shopping sites are no longer in business and they lose their income. This may be also welcomed by a concurrent merchant which enforces the authority to fail by simply flooding the central authority with bogus requests.[18]

On the other hand, as Kormann and Rubin describe, if the central database which contains all the user's data and authentication credentials is compromised and all the information may be accessible for an attacker, this is tremendous as well. Therefore, the central approach brings some big disadvantages by its nature. From the administrator's point of view, to maintain only one central authority and only one central database is more easier than handling a distributed system. However, to tackle the problem of avoiding a denial of service attack and to reduce the danger of failures of merchant sites it is necessary to duplicate the central authority and the central database as well. The Kormann-Rubin paper professes that the replication of the core of such a central Single Sign-On System is quite difficult. This is because it implies that the cryptographical keys which are stored in the central database have to be replicated too. By duplicating the secret keys shared between the central authority and the participating sites the number of existing keys increase, though, the risk to compromise a key increases with the number of existing copies. After all, the key replication is another problem by itself.[18]

2.3.2 MS Passport related Risks

In this section, some hot spots and risky points of the Passport protocol and architecture are spotted. The problems stated here are not only restricted to Passport. These problems are also valid for several other systems using similar technologies. As before, even this section bases on the work of Kormann and Rubin [18] and on the article by Mark Slemko [27].

Problems concerning Cookies

As the papers mentioned above state, the Microsoft Passport solution has an additional disadvantage, namely the use of cookies. These cookies are in other words encrypted credentials belonging to a user. The risk of gaining the content of a cookie is quite low because of the 3DES encryption Passport is using. There exists the possibility to steal a cookie, though. This can be done by impersonating the Microsoft Passport Sign-On server. Then it is possible for an active attacker to delete all cookies which relate to the Passport server's domain. This attack leads to a denial of service for the affected user.[18][27]

As Rubin and Kormann declare, it is obvious that only non-persistent cookies are used by Passport. As long as a user can close the browser session without having signed out before, there is a need for a way to delete the Passport cookies after the browser window has been closed. This can be ensured by the use of non-persistent cookies because they are deleted in the same moment as the affected browser window is closed automatically. If the user leaves his computer without having closed all the browser sessions, though, every other person who has access to the computer is able to abuse the Passport account of the user without any effort. In this situation, the use of non-persistent cookies will not improve the situation either. The unclosed user session keeps alive until the time out of the cookies expires. This is more a human problem, though, the problem of having a careless user is not limited to Passport.[18]

The Use of Persistent Cookies

As the previous section advises, only non-persistent cookies should be used. Kormann, Rubin [18] and Mark Slemko [27] affirm this in their papers as well. They describe the following risks concerning the usage of persistent cookies. Microsoft Passport allows the user to choose whether to use persistent cookies for storing his credentials or not in order to remain signed on even when the user's computer is turned off. This should bring the convenience of never being asked for the Passport's password. By the way, to ask a user this security relevant question and assuming that a normal user knows the difference between the usage of persistent cookies and non-persistent ones is quite hard. Nevertheless, if the user answers to this question with "yes", the Passport Sign-On server will use persistent cookies to leave the user's authentication data on his machine. This affects especially to the *Ticket Cookie*, the *Visited Sites Cookie* and the *Profile Cookie*. Without any further action, every person who has access to the user's computer has access to the user's Passport account and to the participating sites as well. Moreover, every person can access the user's wallet account and may get the user's credit card data if they are available. Therefore, the only cookies which should be used are non-persistent cookies. Persistent cookies are too dangerous for the reasons mentioned above.[18][27]

Kormann and Rubin's paper points out one fundamental weakness of Microsoft Passport, namely the lacking of authenticators as they are used in Kerberos. Thus, in Passport the possession of the cookie is sufficient to access a Passport account or the participating sites. As described in section 2.1.2, the authenticator contains mainly a time-stamp which is encrypted using the clients secret

key. If the authenticator can be decrypted correctly at the server, it is proved that the client has to have the correct key. In Passport there exists no further evidence to ensure the binding between the user and the owner of the cookie.[18]

Other Attacks and Risks

In [18] there are very interesting attacks described concerning a *bogus merchant*, *rewriting and proxying requests* and *DNS attacks*. The principles of these offences are not limited to Passport only. Furthermore, the bogus merchant attack is possible because of the social behavior of the today's users concerning the Internet. On the other hand, the other two attacks described in [18] confess that of course the environmental technologies are points of attacks as well. These are very interesting points and worth to mention. Thus, this section paraphrases the ideas of Kormann and Rubin [18].

The fundamental problem behind a bogus merchant is that the user is redirected to a faked Passport site with the characters of the original Passport login page. This may be possible by offering some service on a website which requires to sign-on using a Passport account. By following a faked login link, the user is then redirected to a bogus login page at a faked site. Additionally the user may be confused by using `pasport.com` (note the type) for the name of this bogus login site. But as Kormann and Rubin state in their paper, users tend to inherently trust the web and so the name is not the big point. The `pasport.com` page displays all conventional symbols and images like the original Passport site. Furthermore, to get the certificate used by `pasport.com` to establish the SSL connection is not a problem either since there are so many root certification authorities, that the existence of one vulnerable to deception seems likely. Beyond this, the knowledge of certificates and SSL of a normal user is quite limited and so representing any valid certificate will work as well. At the end, the user is entering his Passport username and the password which will be stored on the attacker's server. Now the attacker can use this information to enter the user's real Passport account and furthermore, after logging in to Passport, the attacker may also enter the user's *Microsoft Wallet* account in order to get his credit card details.[18]

The *active attack* described in the Kormann-Rubin paper presupposes that a hacker is able to compromise for example a routing device of a Internet service provider in order to detect when a user is redirected to a Passport Sign-On server. If such a redirection is detected, the compromised device rewrites the target URL of the redirection in order to guide the user's browser to a faked Passport Sign-On server which is controlled by the attacker. This is possible, because this redirection is not yet protected by a SSL channel assuming the Standard Sign-On profile is used. Again the intruder may create a faked Passport server as described above. As a result, the attacker's faked sign-on server acts as a proxy between the user and the real Passport Sign-On server (as depicted in figure 2.3). Therefore, the user is asked for his authentication data by the proxying server like the real Passport Sign-On server would do. Behind the scenes, this server contacts the real Passport server in order to establish the session in the behalf of the user. But the user's authentication information is known by the attacker already and so he can get access to sensitive information

such as the user's credit card data.[18]

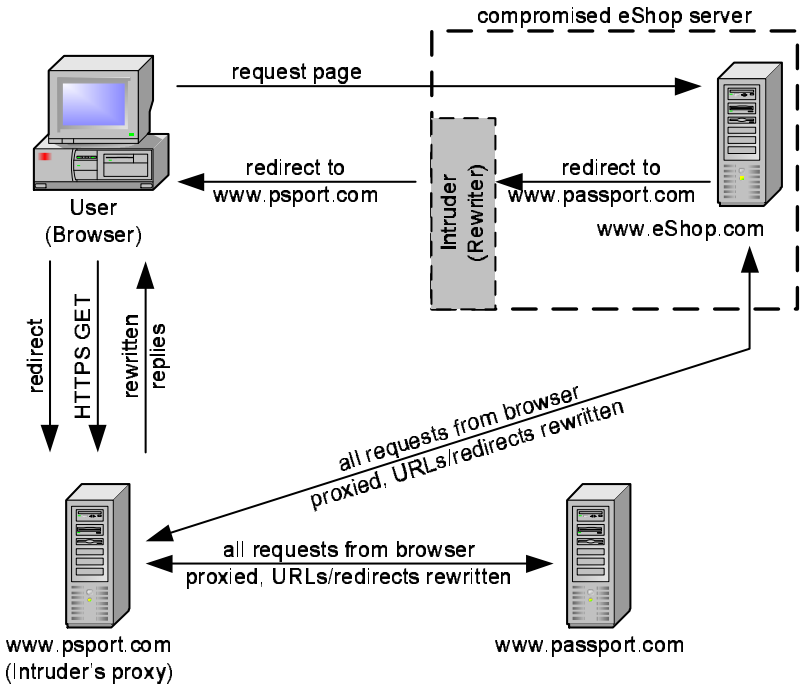

Figure 2.3: Attack by rewriting and proxying requests [18]

The same problem arises when an intruder knows how to control the Domain Name System (DNS) record of a compromised DNS server. Because of the fact that the target of a redirection is given in form of a DNS name, aliasing the IP address of the Passport login server to another bogus server will result in redirecting the user to a faked login page. Again, the attacker gains the username and the password as described above. As the paper [18] states, the use of digitally signed DNS entries (DNSEC) will improve this situation.

Chapter 3

Federated Single Sign-On Systems

Contrarily to the centralized Single Sign-On approach using a central authority, the federated approach tends to deal with several identity fragments maintained by various services, the so called *identity providers* (the term "identity provider" was introduced by the Liberty Alliance [10]). Such an identity provider may be the Internet service of a user's employer or for example any other Internet portal, various communities or business services. Furthermore, a user's identity consists not only of one user account which contains all information necessary for all possible authentication tasks. A user today possesses many identities represented by different accounts fragmented over several identity providers. This approach suits more the needs and the philosophy of today's networks, especially of the Internet.

The information of a user such as his personal profile data, his buying habits etc. are considered as very sensitive. Therefore, users want to decide by themselves how to share which information with the organization of their choice. The global identity of a today's user has to be considered as consisting of several independent identities represented by a number of user accounts located at different identity providers. This way the user can decide to store critical information only at appropriate and trusted providers and has not to trust one single central authority uncompromisingly. A federated Single Sign-On system such as the *Liberty Alliance Project* [10] as described in the next sections is used to combine these distributed identities. Moreover, the user has the opportunity to decide which identity fragment is used with a dedicated service or merchant site to login.

3.1 Liberty Alliance—Project Liberty

The *Liberty Alliance* is a cooperation of various organizations and companies that aims to realize a new level of trust and a new approach to introduce trust in the Internet by using the existing infrastructures. Therefore, the *Liberty Alliance Project* realizes to enable business interactions based on trust relationships among various services which are so far working independently. As

a result, the Liberty Alliance targets to establish open technical specifications ([6],[25]) which delivers the basis for the following considerations. The Liberty Alliance states their Mission as follows [10]:

- *A basis for new revenue opportunities that economically leverage their relationships with consumers and business partners.*
- *A framework within which the businesses can provide consumers with choice, convenience and control when using any device connected to the Internet.*

3.1.1 Introduction

To understand the motivation and the idea behind the Liberty Alliance Project it is necessary to introduce some new aspects concerning identities and trust. Therefore, this section gives introductorily the objectives stated by the Liberty Alliance and describes some new aspects and terms given by the Liberty Alliance specifications, namely *Network Identity* and *Circles of Trust*.

Liberty Objectives

The Liberty Alliance defines in its specifications the goals and main objectives as follows [10]:

- *Enable consumers to protect the privacy and security of their network identity information.*
- *Enable businesses to maintain and manage their customer relationships without third-party participation.*
- *Provide an open Single Sign-On standard that includes decentralized authentication and authorization from multiple providers.*
- *Create a network identity infrastructure that supports all current and emerging network access devices.*

Network Identity

Today, sites and services on the Internet often personalize their appearance suiting the needs of the user. Therefore, the user gets an account with an according username for storing his profile. Furthermore, for e-commerce it may be convenient to store the user's credit card data, his shipping address and his telephone number too. Shortly, the overall global set of these attributes distributed over various accounts all over the net represents a user's *network identity* (figure 3.1) as defined in Liberty Alliance specifications [10].

Circles of Trust

The specifications of the Liberty Alliance defines so called *Circles of Trust* which cover a number of services whereby a user has several isolated accounts. One or maybe more services inside a circle act as a kind of central authority because the other services have trust relationships established with them. The authorities housed in such a circle are called *Identity Providers*. The other remaining

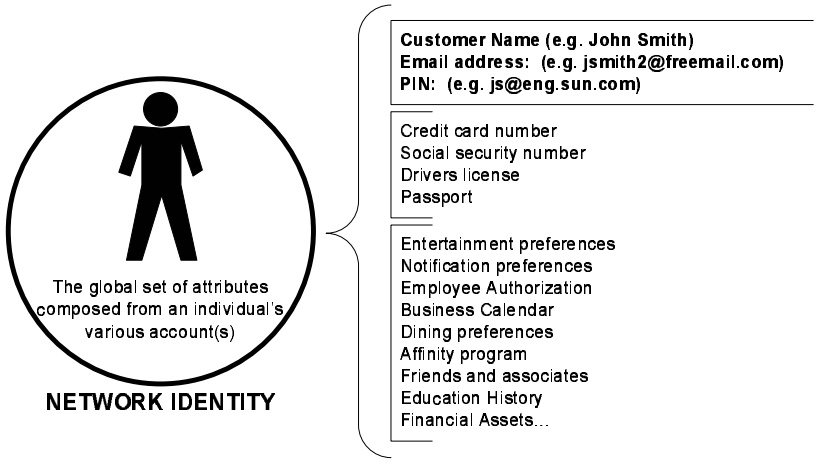

Figure 3.1: Network Identity as considered by the Liberty Alliance [10]

services are called *Service Providers*. The idea of the Project Liberty is to combine and affiliate the various normally independent user accounts—known as the user's *local identity*—inside the circle and allow to use the services inside such a circle after having signed on at an identity provider. In other words, an identity provider acting as authority provides the user's identity to the affiliated services. To sum up, the Liberty architecture consists of three main components:

- Service Providers
- Identity Providers
- Users

Service Providers are entities that simply offer services. Nearby all services offered in the web today can be considered as service providers, such as Internet portals, merchant sites, online banking services, governmental services, etc. [10]

Identity Providers are special service providers. An identity provider and the other affiliated providers build a circle of trust based on established trust relationships as illustrated in figure 3.2. The task of such a provider is to pick the fragmented identities of a user in a circle of trust to a single network identity. Therefore, the identity provider enables network identities. For example, as depicted in figure 3.2 there are two circles of trust. The first one covers services which belong to my business activities. The identity provider of the user's company acts as the identity authority within this circle. The second cluster of services is affiliated with the identity provider which is supported by the user's bank institution. All services within these two circles have established trust relationships with the according identity provider.[10]

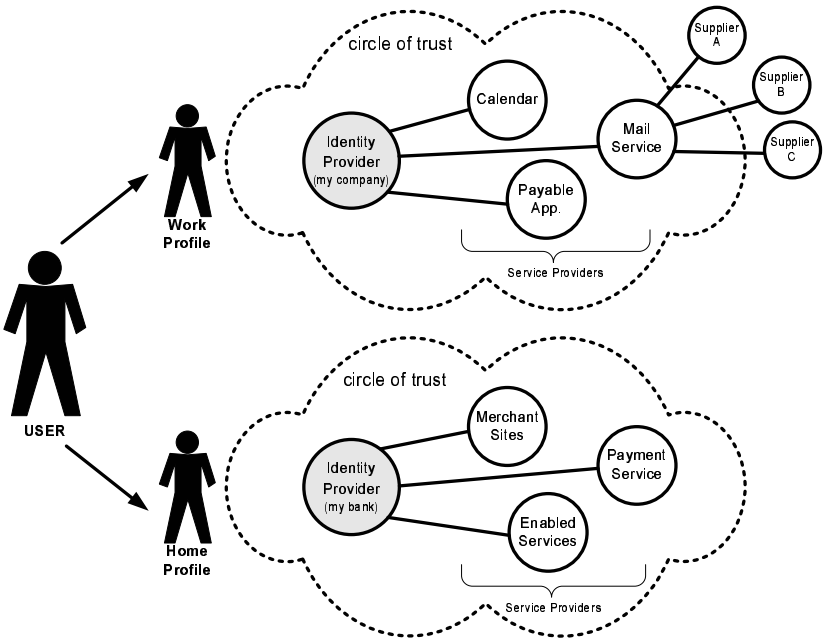

Figure 3.2: Federated network identity and circles of trust [10]

3.1.2 Architecture and Functionality

Actually, the Liberty Alliance Project has two main intentions [10]:

- Identity Federation
- Single Sign-On

The federation of identities allows the user to affiliate his accounts at his identity and service providers which are normally isolated. This linkage of accounts is called *Identity Federation*. This happens based on agreements between the affected providers. The user of the federated identities has to consent this agreement before his account is federated with other providers. Furthermore, the user's agreement has to be recorded and has to be auditable. The Liberty Alliance specifications require the implementors of service and identity providers to ensure this. The second objective of this project is to enable Single Sign-On by signing on to one provider only once which is situated inside a circle of trust. Thus, the user is allowed to use all other providers inside this circle of trust.

The following user experience example gives a deeper look in to the interactions and in to the architecture of Project Liberty's federated identities and Single Sign-On solutions. In this example, which bases on the examples given in the Liberty Alliance specifications [10], the following entities and actors are used:

- John Do: the user
- National Bank Portal: the Internet portal of a bank institute (acts as Identity Provider)
- eMerchant: the merchant site of some e-store (acts as Service Provider)

At the beginning of this example scenario, the user John Do has two separate user accounts. One at the National Bank Portal and the other one at the eMerchant site. Each of these accounts has got an own username and password. Through the following example, these two accounts are instructed to federate John Do's identities.

According to figure 3.3 and to the Liberty Alliance specifications [10], the Single Sign-On process including the federation of identities step by step is presented:

1. **User Accesses his Identity Provider**
   The user (John Do) enters the National Bank's Internet portal by using his local username, e.g. `jodo2308`. After his password has been proved, the user gets access to this site. The identity provider may ask the user if he wishes to federate his local identity with another service provider, e.g. eMerchant site. After a short notice about identity federation, the user has to agree explicitly to federate his local account with the other provider. The local identity provider has to record this agreement.

2. **User Accesses a Service Provider**
   After a while, the user follows a link which redirects him to the website of an affiliated provider, for example to the eMerchant site which is also a member of this circle of trust. The eMerchant website recognizes that the user was redirected by the National Bank Portal and that the user is signed-on to this identity provider. Therefore, this service provider offers the user to federate the eMerchant's identity with the National Bank Portal. Assuming that the user agrees again, the eMerchant site asks him to sign on by using his local eMerchant identity (e.g. by using the local username `johnnyd`).

3. **Federation of Identities**
   As a result, the eMerchant service and the National Bank Portal service federate the user's identity. This does not happen by interchanging the local usernames, but by exchanging a locally unique and opaque user handle (for the user) generated by both providers. As illustrated in figure 3.3, the eMerchant service provider generates `djkjiwu2we2` as local name identifier for the user. Vice versa, `x4dsdic2` is the name identifier which was determined by the identity provider, namely by the National Bank Portal service. If the service provider and the identity provider do communicate, they always have to use the foreign name identifier that is used by the affiliated service provider to identify a federated identity. Of course, if any error during the federation process arises, the provider has to inform the user.

4. **Using the Service Provider**
   After the providers have established the federation of the user's iden-

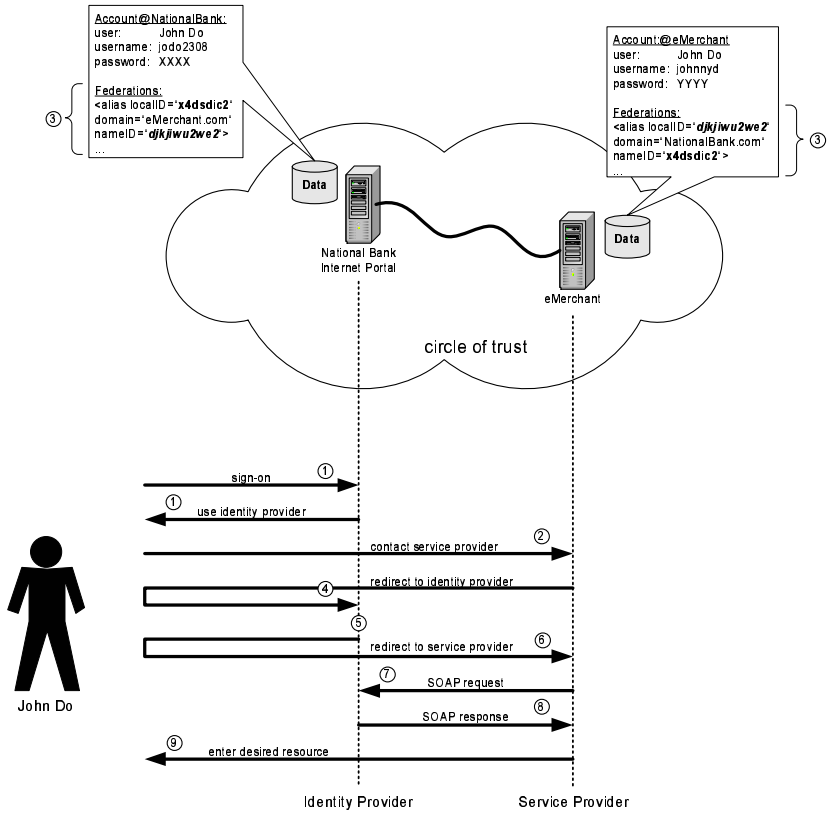

Figure 3.3: Example of Single Sign-On using Project Liberty (SOAP binding) [10].

tity, the eMerchant site notices when the user has signed on to the National Bank Portal service. Therefore, if the user signs on to the identity provider, e.g. to the National Bank Portal site, he may have access to the affiliated service provider, e.g. the eMerchant site, without any further sign on activity. Both providers still use their local user account information. Therefore, the user at the National Bank service is welcomed as `jodo2308` and for the eMerchant site the user is known as `johnnyd`.

Assuming that the user in this example follows a link to the affiliated eMerchant service provider, the service provider contacts the identity provider for information about his authentication status, based on the knowledge of the federated identity. For this communication the Liberty Alliance specifies in its Project Liberty Specifications a response-request protocol which is based on the Security Assertion Markup Language (SAML) [19] [23]. Thus, the eMerchant site which is acting as the requesting service provider generates an authentication request based on this specifications. Such an authentication request is basically an XML document including the user's name identifier that was generated for the federation of the user's identity with the National Bank Portal, e.g. `x4dsdic2`. This request must be dig-

itally signed by using some XML digital signature algorithm. Finally, the eMerchant site redirects the user to the Single Sign-On URL specified by the National Bank Portal identity provider. The authentication request is added to this URL string.

5. **Requesting the Identity Provider**
On the other hand, the identity provider receives the authentication request and after proving the signature and the content of this request it prepares an according authentication response. The provider has to prove if the included user's name identifier corresponds to an existing user and whether the requesting service provider is known by the identity provider or not. Assuming that the request is correct and the user is signed on already, the identity provider can start to prepare the according response. The main part of this response is the so called *assertion*. This assertion element within the authentication response contains beside other details the following information:

   - Audience Restrictions
   ... defining the audience for whom the assertion is dedicated
   - Authentication Statement
   ... declaring the type of authentication, e.g. based on the password or by the use of digital signatures. This element also defines when the service provider has to force the user for re-authentication.
   - Subject information
   ... the subject element holds the name identifiers from the user to identify whom this assertion belongs to

   Of course, this authentication response has to be digitally signed by the identity provider, e.g. the National Bank Portal. Now, based on the Liberty profile one uses, there are several possibilities to send this authentication response to the service provider. In the very simple case, the identity provider answers to the requesting provider with an HTTP-response that contains this authentication response. An alternative is to send this response to the service provider by using a SOAP (Simple Object Access Protocol) [7] message. The scenario in this example uses the SOAP based communication as defined in the Liberty Alliance bindings [25].

   Depending on the implementation, the identity provider may store the prepared response. According to the stored response, the identity provider generates a so called artifact that refers to this prepared authentication response. An artifact is a unique text string which allows the identity provider to identify such an authentication response. Therefore, this artifact is returned to the service provider by redirecting the user's browser (the artifact is attached to the return-URL as parameter) to the service provider (eMerchant).

6. **Request an Assertion**
Back at the eMerchant service, the provider extracts the incoming artifact from the URL. The received artifact can be fragmented into three parts. The first part is constant and defines the type of artifact. The second

fragment is an identifier of the sending identity provider, namely a hash value of its name identifier. The third part is the previously created handle for the authentication response. With the help of the second fragment, the service provider can identify the identity provider sending this artifact. Thus, the eMerchant service knows the corresponding URL which it has to contact by sending a SOAP request. This SOAP request contains a digitally signed SAML request that carries the received artifact in order to request the prepared authentication response.

7. **Provide an Assertion**
   The identity provider, namely the National Bank Portal service in our example, receives the SOAP request including the SAML request. It has to prove the validity of the signature and if the artifact which is contained inside the SAML request points to a prepared authentication response. If there is no problem concerning the incoming request the identity provider resolves the included artifact in order to regenerate the prepared authentication response. This digitally signed response is wrapped into a SOAP response message and this SOAP message is returned to the requesting service provider.

8. **Service Provider processes the Assertion**
   The eMerchant service which acts in the role of the service provider in this example, extracts the authentication response from the incoming SOAP message. After the authentication response which contains the assertion has been proved and evaluated successfully, the service provider accepts the user as authenticated. There is no difference between signing on in this way and logging in by using the local authentication mechanism of the service provider. The result is the same. Depending on the local user management at the service provider, the user will be welcomed with his local username, e.g. as mister `johnnyd` at the eMerchant service.

9. **Gain Accesses**
   Finally, the user gains access to the desired resource.

This example describes a standard situation within one circle of trust only. The used technologies like SAML and SOAP and the Liberty Alliance Project itself base on interchanging of XML documents. Moreover, the usage of XML based protocols allows to integrate this Single Sign-On solution into existing web services and web applications very smoothly. In comparison with the Microsoft Passport model, the specifications of the Liberty Alliance Project do not deal with questions of how the user remains signed on and how to handle Single Sign-On after this request-response-communication. It depends on the provider implementation how to keep track of the user session. It is likely that the session management is realized by the use of cookies, especially non-persistent cookies.

The third part of this thesis deals with the practical implementation of some parts of the Liberty Alliance Project. Particularly, this implementation covers the Single Sign-On procedure based on the *Liberty Browser Artifact Profile* as defined in [25]. Therefore, more details about the protocol and the specification of the XML requests and responses which were mentioned above are covered by the implementation's chapters 8 and 9.

Logout Mechanism

Even for logging out, the Liberty Alliance specification concentrates on describing the protocol which is necessary for the communication between the affected providers. Here too, there are some possibilities to sign-out depending on the implementation of the providers and depending on where the sign-out process has been started [25]. This logout procedure can be started either at the identity provider or at any of the service providers which have received an authentication assertion of the particular identity provider. As a result, after having started the logout procedure the affected identity provider has to contact every service provider that maintains a user session based on the identity provider's assertion. To each of them, the identity provider has to send a *logout notification* which contains the name identifier of the user who wishes to logout and the identifier of the provider who receives this notification. The profiles available as defined in the Project Liberty's specifications [25] for this mechanism are:

- *Single Logout initiated at the Identity Provider*
  - ***HTTP-Redirected-Based:***
    The identity provider uses HTTP redirects to deliver logout notifications to the service providers.
  - ***HTTP-GET-Based:***
    This is using HTTP Get requests to communicate logout notifications from an identity provider to the service providers
  - ***SOAP/HTTP-Based:***
    Relies on SOAP over HTTP messaging to distribute the logout notifications.
- *Single Logout initiated at the Service Provider*
  - ***HTTP-Redirected-Based:***
    The identity provider uses HTTP redirects to deliver logout notifications to the service providers.
  - ***SOAP/HTTP-Based:***
    Relies on SOAP over HTTP messaging to distribute the logout notifications.

To illustrate a basic single logout procedure, the following example uses the SOAP/HTTP based profile (figure 3.4) according to the Liberty Alliance specifications [25]. The process is initiated at the service provider.

1. **User Accesses the Single Sign-Out Service**
   At first, the user follows a sign-out link displayed on the service provider's site. The user's browser accesses the single logout service at the service provider he is currently using.

2. **Sending a Notification**
   As a consequence of this, the service provider generates a logout-notification according to the Project Liberty specifications [6]. This notification is wrapped into a SOAP message which the service provider sends to the SOAP endpoint address of the affected identity provider.

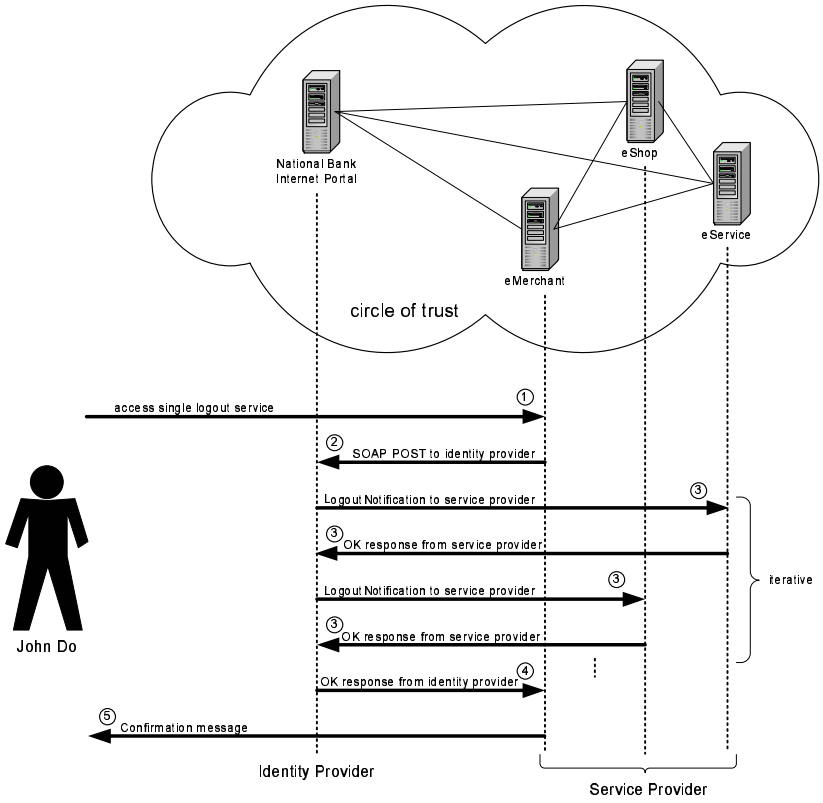

Figure 3.4: Example of single logout using Project Liberty (SOAP binding) [6].

3. **Processing the Notification**
   The identity provider receives the notification inside the incoming SOAP message and has to prove the signature on the notification. Furthermore, the identity provider has to check if an authentication assertion was sent to the service provider which was sending the notification. If the validation was successful, the identity provider has to send a logout notification to each service provider as well that the identity provider has provided an authentication assertion to, belonging to the affected user during this session. Thus, this step is iterative because the identity provider has to contact several service providers by using the logout-profile which the affected service provider prefers. Thus, it may be possible that the identity provider has to send various logout messages by using SOAP messages or by using HTTP-redirections. As a result, all sessions of the current user on the other service providers should be closed. Finally, the identity provider terminates the session for the user and stops providing authentication assertions for him.

4. **Response to Notification**
   The identity provider answers to the SOAP message of the service provider with an according `HTTP-204 OK` response, unless no error occurs during

the session termination.

5. **Confirmation**
   At last, the service provider which has initiated the logout process receives the OK response and has to notify the user about the successful logout process. Now, this service provider can terminate its session with the user as well. The user's browser receives an HTTP response which confirms that the requested action of single logout was completed.

Of course, beside the provider communication drafted above, each of the involved providers has to make an effort to terminate the user's session depending on the local environment, namely depending on how the user and session management is implemented. Therefore, the providers have to delete cookies or have to take other adequate measures in order to ensure that the user's session is terminated irrevocably. The Project Liberty specifications make no restriction in this question either.

3.1.3 Extension by federating identity providers

So far, the previous example deals only with the federation of the user's identity among one identity provider and several service providers. Because the cores of an identity provider and a service provider are similar, identity federation among various identity providers is possible too. Only with this feature, the term Network Identity gets its full meaning. Thereby it is possible to combine identities across several circles of trust. Furthermore, even if there exist more than one identity provider within one circle of trust, the user has to sign-on only once assuming that there exists an adequate trust relationship between the participating identity providers.

In other words, it is possible to combine and federate identities among all providers no matter if it is a service or an identity provider. Thus, it is possible to link several identity providers to one service provider. For example, in the case that a user is registered at the identity provider supported by his company and the same user is a member of a second identity provider as well which is primarily used for private purposes. Though, if the user switches from his office to his home he switches his identity provider too. The service providers he is using are still the same. Furthermore, by using different identity providers it is easy to differ between a user's business identity and his private identity. The Project Liberty specifications do not put any limits to the federation of identities. The basis for all federations are the business and relationship agreements and the circle of trust policies. Based on this and on the federation of the user's local identities the providers of a circle of trust can communicate with each other about or on behalf of the user [10]. There is no limit for creating long chains of identity and service providers. On account of this, even if two circles of trust or especially two identity providers have no direct relationship based on federation, there may exist a path between them involving some other providers. Along this path, the user's identity is federated in each link of the chain. Thus, there is no need for a global unique user identifier of the kind Microsoft Passport requires. This is a substantial advantage of this federated architecture in comparison with the Passport approach.

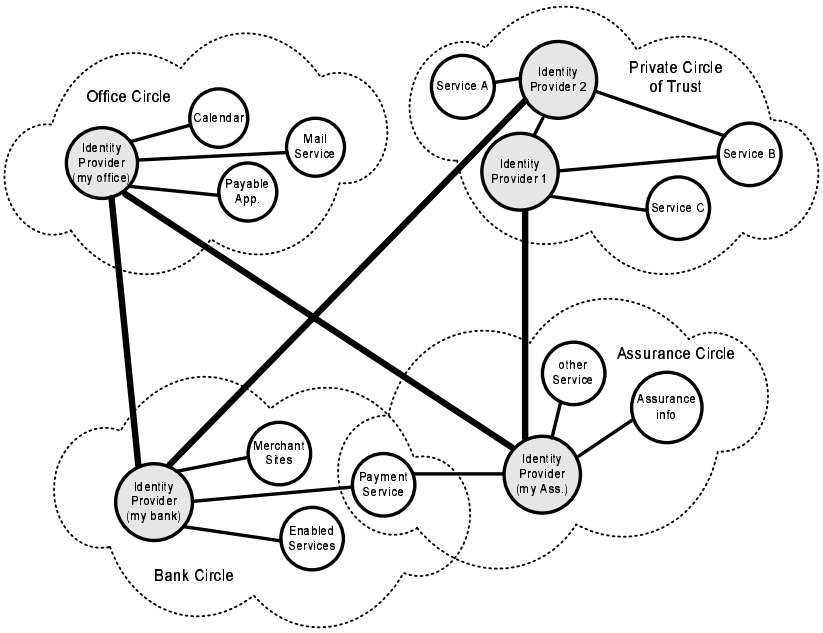

Figure 3.5: Example of federated identity providers housed in different circles of trust.

By linking multiple providers housed in several circles of trust it is possible to cover all accounts of a particular user. As depicted in figure 3.5, there exist several identity providers, but not all of them are affiliated with all the other ones. This is not necessary because it is possible to construct a chain of federated providers from any identity provider through other identity providers to every desired service provider. For example, there does not exist any direct relationship between the identity provider of "my office" and the second identity provider placed in the "private" circle of trust. Even though it is possible to authenticate the user with his office identity provider across the "my bank" identity provider to the desired second identity provider in the "private" circle of trust. By using this principle it is possible to summarize all accounts and local user identities of a particular person. This model suits more the needs and condition of today's Internet than a centralized solution.

Part II

Privacy and Trust in Distributed Networks

Chapter 4

Privacy and Trust in Distributed Networks

Today distributed service frameworks play an ever more important role. Transitive trust is of great importance in such frameworks and is well researched. Although there are many solutions for building and transmitting trust in distributed networks, impacts on privacy are often neglected. In the following chapters various privacy issues, threats, and possible solutions to these problems are analyzed. Based on trust metrics it will be shown why insufficient trust is eventually inevitable if a request or message pass through a chain of services. Depending on the reaction of the service, privacy critical information may leak to other entities in the chain. It is shown that even simple error messages pose a privacy threat and that proper re-authentication methods should be used instead. Several methods of re-authentication and their impacts on privacy are discussed.

4.1 Motivation

In the course of working on this thesis the question arises of how to deal with trust in situations of Single Sign-On. In other words, if a user has been authenticated to a service provider by the use of some authority this authority asserts that the user is authenticated correctly. In terms of the Liberty Alliance, an identity provider offers assertions to affiliated service providers. If such an assertion is used for authentication purposes several times along a chain of services (e.g. if several providers are affiliated as depicted in figure 3.5) it is likely that the trustworthiness of the assertion decreases with each step by which this assertion is used. This question becomes a hot topic in sensitive environments such as e-government. In this context, to keep a user's privacy is very substantial as well. Thus, motivated by such scenarios this part of the thesis deals with a mathematical framework which allows to calculate trustworthiness of an assertion or more generally spoken of a message or a request which is processed along a chain of services. Assuming there are various affiliated identity providers as depicted in figure 3.5 a user has to be authenticated at one of them only. Furthermore, relying on the assertions supported by the identity provider authenticating the user the other identity providers generate assertions as well. It is conceivable

that a service provider has to rely on an assertion which bases on various other assertions. In other words, the affected service provider has to trust a chain of assertions. This part of the thesis states that the trustworthiness along such a chain is not unconditionally transitive and the resulting assertion may not be as trustworthy as the first one because of the increasing number of involved providers. With an increasing number of providers it is more likely that one of them may be compromised. The following chapters show a more abstract and generic approach to this problem. There, not only assertions and providers are considered but also generic requests and messages processed along a chain of services and entities. At the end of this part, there are some possibilities drafted describing how to react if a service in this chain considers an incoming request or assertion as not trustworthy.

The following example shall illustrate the problems concerning privacy and trust relationships in a distributed network of services once more. Here, in order to process a special task, the cooperation of several autonomous services is needed.

4.2 Introductional Example

Assume the following situation: A client would like to buy something which is offered by an online shop where the client is registered in the customer data base. Therefore, the buyer signs a purchase order through the shop's online portal. The process involves several services of other instances such as a service for checking the inventory of the chosen product or a service for doing the payment. Figure 4.1 depicts the constellation of services used in this example.

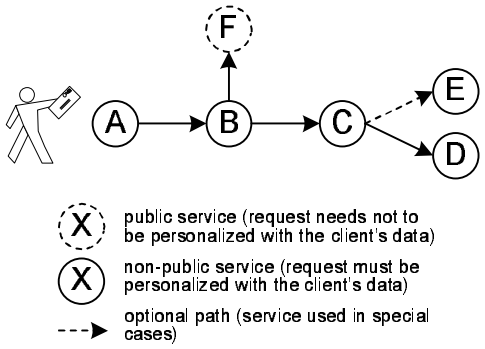

Figure 4.1: example of distributed services

After the user has been authenticated to the portal of the online shop (*service A*), he has to fill in some forms and enter some personal data. When this first step is completed, service A contacts the service which processes a new order (*service B*). This service validates and verifies the content of the order. First, it checks if the desired product is still available for sale. This will be done by sending a request to the inventory service (*service F*). This is an "anonymous" request, as service B does not need to personalize this request with the client's or the agent's data. Next, service B initiates the payment service (*service C*) to process the payment of the product. The payment is preferably done by

the client's customer account which allows the client to buy products within a certain credit line. Thus, service C contacts a service of the internal account system (*service D*). Assuming, that the user has not enough money on his account, the payment service tries to make the payment by a direct debit to the client's credit card institution by requesting the corresponding *service E*.

At this point, some problems concerning privacy and transitive trust will arise:

- Because of too low security restrictions in the chain of services, it might be possible that service E does not accept the request.
- Depending on how service E reacts to this situation, private data about the client may be disclosed to other services and the client's privacy may be harmed.

If service E does not trust the request from service C, it is likely that an error message is sent back to the client. Passing this message back to the user through the service chain harms the user's privacy because each service learns about the error. In the worst case, the error message contains the information of its origin, service E. Thus, every service in the chain gains knowledge that the client has not enough money and needs to use his credit card. Otherwise it would not have been necessary to involve service E into the process. Moreover, by recording a client's habit it would be possible for instance to recognize that his customer account is overdrawn every end of month. Instead of sending an error message in response to insufficient trust into the authentication, service E can request a re-authentication of the client. Again, similar problems and privacy threats arise in this case.

In the next section a trust model and applicable trust algebra is described. This is based on the work of Audun Jøsang ([12],[13],[14],[15]). Based on this a metrics for determining the trustworthiness of a request inside a chain of services is discussed. By introducing and adapting trust values ([1],[2]) based on established criteria ([11],[9]) it is possible to decide either to trust or distrust an incoming request. This will lead to the necessity of re-authenticating requests if a service does not trust the incoming request. In the last chapter, privacy and threats on privacy are described and categorized. Error messages and re-authentication requests are considered critical to privacy and so they are discussed in detail. In this context, several re-authentication mechanisms are possible. They can be divided into synchronous and asynchronous mechanisms. For example, contacting the user through an asynchronous way can happen by sending an e-mail. Such an asynchronous re-authentication causes some delay which is undesirable in most cases. Therefore, a synchronous re-authentication is preferable, for example through a trusted web server or with the help of encrypted messages. Several mechanisms will be discussed under the aspect of privacy.

Chapter 5

Trust Algebra

Determining trust and how to calculate resulting trustworthiness of coupled entities is crucial for distributed services. It suits the needs to determine whether to trust or distrust a chain of services. Therefore, some mechanism is necessary to evaluate the trustability of a message or a request processed successive through a row of services whereby each entity has an according level of security, an according level of trustworthiness, respectively. The algebra introduced in this chapter gives a possible solution for this problem. In this chapter, first a metrics which allows to assign some level of trust to some entity is defined. This is the base for calculations later on. In the second part, the so called *trust algebra* [14] is introduced. Within this algebra the subjective logic allows to examine joined entities under the aspect of trust. Finally, with the result of considerations using this trust algebra, it is possible to express the trustworthiness of coupled entities by a resulting trust value.

5.1 The Opinion Triangle

5.1.1 Definitions

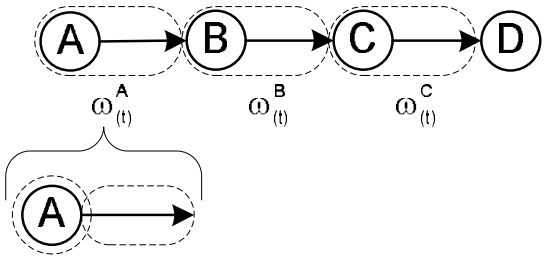

Figure 5.1: an example of chained trust

Assuming the situation illustrated in figure 5.1. Here, an initial request was started at service A. This leads finally to a request to service D. Service D, due to its high security requirements, has to decide if the request and the chain of services associated with this request is trustworthy or not. In [14], an applicable

trust model and an algebra for calculating trust in chained trust relationships is given.

Initially, the trustworthiness of each service has to be determined. Therefore, every entity can be divided in two parts. On the one hand, the connection between two services has to be evaluated under the aspect of security. For example, a normal TCP/IP connection will result in a lower level of security than an SSL-connection with client certificates. On the other hand, the service itself has to be evaluated. For this purpose, some established criteria already exist, e.g. the *Common Criteria* [11]. Such criteria not only consider the technical infrastructure and the system itself. They also take the technical and nontechnical environment into account. After evaluating each service, the level of trust must be expressed in an applicable metrics. Therefore, in [15] and [14] the term *opinion* (ω) was defined as:

$$t + d + u = 1, \{t, d, u\} \in [0, 1]^3 \tag{5.1}$$

> **Definition Opinion:** *Let $\omega = \{t, d, u\}$ be a triplet satisfying (5.1) where the first, second and third component correspond to trust, distrust and uncertainty respectively. Then ω is called an opinion.*

Corresponding to this definition, several levels of trustworthiness are mapped accordingly to different opinions. Equation 5.1 defines a triangle which is depicted in figure 5.2. Every opinion ω can be described as a point $\{t, d, u\}$ in the triangle. For example, there are five trust levels to distinguish (according to table 5.1) and each trust level can be found in the opinion triangle (fig. 5.2).

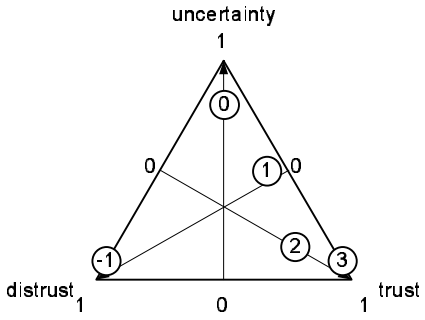

Figure 5.2: trust levels inserted into the opinion triangle

The advantage of using the opinion based trust model instead of a simple trust level based model is that there are three parameters expressing trust instead of only one value. As it will turn out in the next section, these three separate values are not treated equally. For example, it is possible to weigh the *uncertainty*-value more than the others. This will result in a real world adequate model for distributed trust relationships.

t	d	u	trust level	
0.00	0.95	0.05	distrust	(-1)
0.10	0.10	0.80	ignorance	(0)
0.40	0.10	0.50	minimum trust	(1)
0.70	0.15	0.15	medium trust	(2)
0.95	0.00	0.05	maximum trust	(3)

Table 5.1: example of mapping between trust levels and opinions $\omega = \{t, d, u\}$

5.2 Subjective Logic

The algebra for determining trust will be based on a framework for artificial reasoning called *Subjective Logic*, which has already been described in Audun Jøsang's papers [13] and [14]. It defines various logical operators for combining opinions. In this section, only the most important definitions will be quoted, e.g. the *Recommendation* and *Consensus* operator. Firstly, this section gives only a formal definition of them. Afterwards this algebra will be applied on the problem of distributed web services mentioned in the introduction of this part of the thesis.

5.2.1 Definition: Conjunction

If some entity has two different opinions about another entity, then the conjunction ($\wedge$) of these opinions may be useful.

> *Let $\omega_p^A = \{t_p^A, d_p^A, u_p^A\}$ and $\omega_q^A = \{t_q^A, d_q^A, u_q^A\}$ be entity A's opinions about two distinct binary statements p and q. Then the conjunction of ω_p^A and ω_q^A, representing A's opinion about both p and q being true is defined by* [14]:

$$\begin{aligned} \omega_{p\wedge q}^A &= \omega_p^A \wedge \omega_q^A \\ &= \{t_{p\wedge q}^A, d_{p\wedge q}^A, u_{p\wedge q}^A\} \end{aligned} \tag{5.2}$$

where

$$\begin{aligned} t_{p\wedge q}^A &= t_p^A t_q^A, \\ d_{p\wedge q}^A &= d_p^A + d_q^A - d_p^A d_q^A, \\ u_{p\wedge q}^A &= t_p^A u_q^A + u_p^A t_q^A + u_p^A u_q^A. \end{aligned} \tag{5.3}$$

Later in this chapter, a special case of the conjunction operator is used—the conjunction of two similar opinions:

$$\begin{aligned} \omega_{p\wedge p}^A &= \omega_p^A \wedge \omega_p^A \\ &= \{t_{p\wedge p}^A, d_{p\wedge p}^A, u_{p\wedge p}^A\} \end{aligned} \tag{5.4}$$

where

$$\begin{aligned} t_{p\wedge p}^A &= (t_p^A)^2, \\ d_{p\wedge q}^A &= 2d_p^A - (d_p^A)^2, \\ u_{p\wedge q}^A &= 2t_p^A u_p^A + (u_p^A)^2. \end{aligned} \tag{5.5}$$

5.2.2 Definition: Recommendation

Recommendation ($\otimes$) is needed if an entity A decides about the trustworthiness of something (p) based on trust-recommendations given by a third party B. More formally:

Let A and B two entities where $\omega_B^A = \{t_B^A, d_B^A, u_B^A\}$ is A's opinion about B's recommendation, and let p be a binary statement where $\omega_p^B = \{t_p^B, d_p^B, u_p^B\}$ is B's opinion about p expressed in a recommendation to A. Then A's opinion about p as a result of the recommendation from B is defined by [14]:

$$\begin{aligned} \omega_p^{AB} &= \omega_B^A \otimes \omega_p^B \\ &= \{t_p^{AB}, d_p^{AB}, u_p^{AB}\} \end{aligned} \tag{5.6}$$

where

$$\begin{aligned} t_p^{AB} &= t_B^A t_p^B, \\ d_p^{AB} &= t_B^A d_p^B, \\ u_p^{AB} &= d_B^A + u_B^A + t_B^A u_p^B. \end{aligned} \tag{5.7}$$

It must be mentioned that ω_p^B is actually only the opinion that B recommends to A and it is not necessarily B's real opinion. The opinion about an entity's recommendation, e.g. ω_B^A, results of the conjunction of two separate opinions. On the one hand, there is entity A's opinion about the trustworthiness of entity B by itself, called $\omega_{KA(B)}^A$. On the other hand, entity A's opinion about the trustworthiness of the recommendations (recommendation trust) made by entity B has to be considered, given as $\omega_{RT(B)}^A$. Applying the conjunction operator as defined above results in:

$$\omega_B^A = (\omega_{KA(B)}^A \wedge \omega_{RT(B)}^A) \tag{5.8}$$

This term is also known as the *conjunctive recommendation term* [14].

5.2.3 Definition: Consensus

The consensus ($\oplus$) operator is used to combine several independent opinions about the same statement. As a result the certainty should increase.

Let $\omega_p^A = \{t_p^A, d_p^A, u_p^A\}$ and $\omega_p^B = \{t_p^B, d_p^B, u_p^B\}$ be opinions respectively held by entities A and B about the same statement p. Then the consensus opinion held by an imaginary entity [A,B] representing both A and B is defined by [14]:

$$\begin{aligned} \omega_p^{A,B} &= \omega_p^A \oplus \omega_p^B \\ &= \{t_p^{A,B}, d_p^{A,B}, u_p^{A,B}\} \end{aligned} \tag{5.9}$$

where

$$\begin{array}{rcl} t_p^{A,B} & = & (t_p^A u_p^B + t_p^B u_p^A)/(u_p^A + u_p^B - u_p^A u_p^B), \\ d_p^{A,B} & = & (d_p^A u_p^B + d_p^B u_p^A)/(u_p^A + u_p^B - u_p^A u_p^B), \\ u_p^{A,B} & = & (u_p^A u_p^B)/(u_p^A + u_p^B - u_p^A u_p^B). \end{array} \tag{5.10}$$

The effect of the consensus operator is to reduce the uncertainty. Opinions containing zero uncertainty can not be combined.

Equipped with these three basic operation, it is possible to form a model for determining distributed trust in the web service scenario.

Chapter 6

Chained Trust

As a basis for any calculation, each service must already have assigned an opinion ω about its security level—preferable by an independent authority. This opinion will be determined initially, during setting up the service, and has to be kept up-to-date. With some precautions, for example wrapping the opinion value into a signed certificate, the trust value could be sent within the requests. Anyway, trust values, opinions about the trustworthiness of an entity, respectively, have to be propagated in the network.

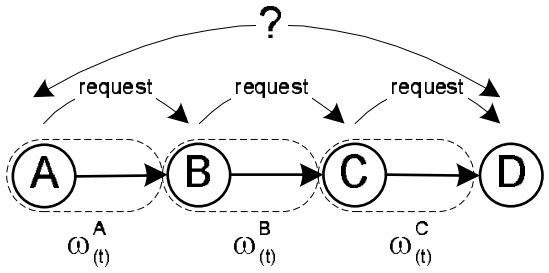

Figure 6.1: the problem of chained trust

Figure 6.1 depicts the stated problem: a request originating from service A will be propagated through service B, C to D. Because of the security requirements of service D, there must be a mechanism to decide whether the request is trustworthy or not. This question is similar to determining service D's opinion about the trustworthiness of service A. Because of the indirect relationship between service A and D, the principle of recommendation is used. Let us consider the chained situation step by step. At first, service B has to decide about the trustworthiness of service A. This can easily be done by evaluating the opinion of service A's trustworthiness ω_t^A, which preferable was attached to the request. Because of the direct trust relationship between A and B, ω_t^A is the value which enables a decision. In the next step, assuming that service B considers A's request as trustworthy, service B sends a request to service C. At this point, service C has to decide whether to trust or distrust the whole chain. Therefore, the subjective logic is needed. A direct trust relationship exists between B and C and the opinion ω_t^B is received by service C through the request. However,

between service A and C there is no such direct relationship. In order to decide about the trustworthiness of the chain, respectively about the trustability of service A, the recommendation operator is applied. In this case, the opinion ω_t^A about service A's trustworthiness is recommended to service C by the preceding service B.

In the definitions stated by Audon Jøsang [14] there is a difference between the opinion ω_t^A about the trustability of an entity A and the opinion about recommendations of an entity. Therefore, the recommendation operator as introduced in section 5.2.2 requires the so called conjunctive recommendation term (equation 5.8), e.g. ω_B^A, which combines the opinion of the trustworthiness about a service itself and the opinion about its recommendation by applying the conjunction operator (as stated in 5.2.2). In this work, these two opinions are considered as equal. This assumption is legitimate in this context because in this scenario, if a service is not trustworthy and its trust value is respectively low, then the recommendations of this service should also be considered as not trustworthy and vice versa. Thus, the conjunctive recommendation term is built by the use of only one opinion and the term can be reduced to (equ. 6.1):

$$\omega_B^A \quad = \quad (\omega_{KA(B)}^A \wedge \omega_{KA(B)}^A) = (\omega_t^B \wedge \omega_t^B) = \omega_{t\wedge t}^B \tag{6.1}$$

Therefore, it is not necessary to define a separate opinion for recommendations which simplifies the application of the recommendation operator. With this assumption, the trust relationship at service C can be calculated as follows:

$$\begin{aligned} \omega_{t(A)}^{CB} &= \omega_B^C \otimes \omega_t^A \\ &= (\omega_t^B \wedge \omega_t^B) \otimes \omega_t^A \\ &= \omega_{t\wedge t}^B \otimes \omega_t^A \end{aligned} \tag{6.2}$$

In (6.2), C's opinion about the trustworthiness of service A consists of:

- the conjunction ($\wedge$) of C's opinion about B's recommendations and B's authenticity. In this matter, they are the same, namely ω_t^B.
- B's opinion about the trustworthiness of service A (ω_t^A)

With this result, service C is able to decide about the trustability of the chain. The same problem arises in the next step. Then, service D receives the request from the preceding service and it has to decide whether to trust or to distrust the chain of services. Based on recommendations as mentioned above, service D will calculate the opinion $\omega_{t(A)}^{DCB}$ in order to determine the trustability of service A through the chain of recommendations.

$$\begin{aligned} \omega_{t(A)}^{DCB} &= \omega_C^D \otimes \omega_B^C \otimes \omega_t^A \\ &= (\omega_t^C \wedge \omega_t^C) \otimes (\omega_t^B \wedge \omega_t^B) \otimes \omega_t^A \\ &= (\omega_t^C \wedge \omega_t^C) \otimes \omega_{t(A)}^{CB} \\ &= \omega_{t\wedge t}^C \otimes \omega_{t(A)}^{CB} \end{aligned} \tag{6.3}$$

The pattern of calculation is always the same. Moreover, it can be shown that this determination is recursive. With the opinion about the trustability of the chain so far, which was determined at the preceding service, and with the opinion about the trust relationship between the actual and the previous service,

the chain can be evaluated. This recursive approach to this calculation is stated in the following lemma (6.4):

Lemma: Recursive Trust *Let $A_1 \ldots A_n$ be a chain of services, where service A_{n-1} makes some request to service A_n. A_{n-1}'s opinion about the trustworthiness of the chain so far is given by $\omega_{t(A_1)}^{A_{n-1}\ldots A_2}$ and it is attached to the request. A_n's opinion about the trustworthiness of the whole chain is:*

$$\begin{aligned} \omega_{t(A_1)}^{A_n\ldots A_2} &= \omega_{A_{n-1}}^{A_n} \otimes \omega_{t(A_1)}^{A_{n-1}\ldots A_2} \\ &= \omega_{t\wedge t}^{A_{n-1}} \otimes \omega_{t(A_1)}^{A_{n-1}\ldots A_2} \end{aligned} \tag{6.4}$$

Tracing the components of the opinion about the trustworthiness of the whole chain during its propagation (based on recommendation) leads to the conclusion that the trust-component will never increase and the uncertainty generally becomes higher. Furthermore, at the end of the chain, depending on the particular opinions during the propagation, the uncertainty about a request may be too high for a service with sophisticated security restrictions. This is the reason why services either decline to act on the request and return an error message, or need the possibility to re-authenticate the client. In the next section we are going to look at the privacy threats that arise from this situation.

Chapter 7

Privacy and Re-authentication

As described in the previous chapter, insufficient trust leads to either declining a request or forcing a re-authentication. In this section we are going to look at the impact of these actions on the user's privacy. Webster's dictionary defines privacy as "freedom from unauthorized intrusion" which is also an adequate definition for this situation. Here, privacy means that the involved services should not gain more knowledge than it is necessary.

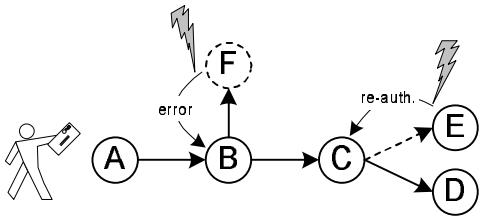

Figure 7.1: errors in distributed services

In the example given in the introduction, the error message disclosed enough information to conclude about the customer's financial situation. Therefore, error messages can act as side-channels. By analyzing similar processes initiated by different people it is possible to establish the standard workflow. But some client's request causes an error message or a re-authentication request due to a too low security level in the chain of services (service E in our example). With this information and with enough knowledge about the process it is possible to conclude about the involved services. In addition it is possible to gain information about user's request and about the user himself. This is why it is crucial to react carefully in such a situation. There are two possible reactions:

- replying with an error message
- starting a re-authentication procedure

Chain history The question arises how and when information is propagated in order to reach previous services or the original client directly. One possibility is adding a chain history of preceding services to every request. The benefits of this approach are obvious: the original client is known to every service and each service can decide to trust the request based on the history of the request instead of calculating a level of trust. But adding a history of all involved services to every message not only increases the header of such messages, it also harms the user's privacy. Every involved service learns about all preceding services. In our example, service E (credit card institute) would learn about the user's contact to the online shop portal represented by service A, and that the user is going to buy something but does not have enough money (service B and service C respectively.) It seems that harming privacy is a too high price for the benefits of a chain history. Thus, a request or message should contain information of the sending and the receiving service only. No service should get more information about the process automatically, or more generally: a service should get as much information as needed but not more than absolutely necessary.

Generic services vs. personalized services The behavior and privacy impact of each service depends on its properties. For example, a generic service like a time-service, which simply sends back a message containing the current time, does not need any personal data of the original client. Thus, such a service only generates an error message if an internal error occurs. If the fact, that the generic service has been called does not disclose any information about the user, generic services pose no threat to the user's privacy.

On the other hand, personalized services may and should act differently depending on whom is requesting their service. These differences leak information about the user to other services in the chain. A benefit of personalized, well informed services is that they can react in more sophisticated ways, for example with a re-authentication request. In this chapter we will focus on personalized services, as generic services usually are not a privacy threat as they do not process personal data.

7.1 Error Messages

The minimum reaction is to return an error message to the preceding service. In the situation depicted in fig. 7.1, service E rejects the request from service C due to security considerations. Therefore, service C will receive an according message. Depending on how detailed the error message is, the receiving service will react. In the worst case, the error will be reported backwards through the whole chain to the original client. On the one hand, in order to give the user as much help as possible, the error message should be very detailed. On the other hand, a detailed error message, which in the worst case passes through every entity in the chain, gives all desirable information about the whole process and the user himself. For this reason, such messages should be encrypted with the client's public key. This prevents disclosing detailed information to any third party. But already the occurrence of an error message brings enough information. Nevertheless, there must be at least a message about the unsuccessfully terminated process that has to be sent to every involved service in order to

stop the process. It is preferable to use a solution where services try to fulfill the request without rejecting an error message. The usage of re-authentication is an attempt to do so. In same cases it may be the only practical way in a distributed service framework.

7.2 Re-authentication Requests

A re-authentication request is sent to the user in order to authenticate the request for a dedicated service. It is also possible that instead of the user himself a trusted third party is allowed to sign the request on behalf of the user. A re-authentication request has to contain at least the following data:

- a pseudo-random stream or a digital finger-print of the sending service (hash-value)
- a time-stamp or nonce to prevent replay-attacks
- an explanation of the receiving service and the purpose of this service in plain text (readable for the user who has to sign it)
- a signature over the request with the private key of the sending service in order to prove the origin of this request

The time-stamp or nonce is needed to prevent manipulation of a service with a replayed re-authentication request in order to gain confidential information about a client. This component is essential for security and is common practice in security technologies. The additional text of the request has to contain detailed information about the service which wants the user to re-authenticate himself. The user must be able to recognize the circumstances of this request in order to decide correctly. Furthermore, the explanation in the request must point out the consequences and results of signing and executing the request. At last, the whole re-authentication request has to be signed by the sending service in order to prove the origin of the message. With such a detailed request, the client will be well informed and will be able to decide whether to grant the permission by re-authenticating or not to grant permission.

Re-authentication requests can be split up into synchronous and asynchronous requests. In a synchronous request, the re-authentication request has to be fulfilled in time. That means that the requesting service is waiting until the request is sent back. This is a viable option only if it can be assumed that this will happen within a certain time frame. Contrarily, the asynchronous request leads to a temporary interruption of the process, because it is not predictable when the re-authentication request will be sent back. If too much time elapsed between starting the process and answering to the re-authentication request, some problems may arise concerning time restrictions for some outstanding requests in the chain. Therefore, a re-authentication through a synchronous way should be the first choice.

The following mechanisms and possibilities are discussed in the following section. The re-authentication can be carried out by four means, as follows:

- an out-of-band mechanism
- a roll-back mechanism
- using a ticket-server
- using a trusted web server

7.2.1 Out-of-Band Re-authentication

When using this method, every service contacts the user directly. It implies the requirement that every service has to get information about how to reach the user in an out-of-band way. Therefore, it is necessary to add some information like the client's e-mail address to the request.

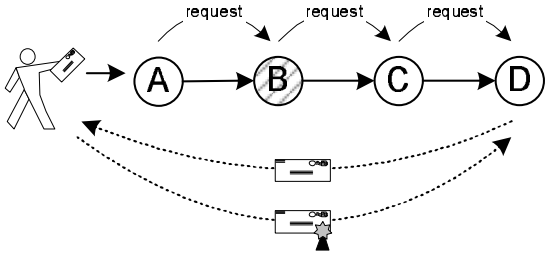

Figure 7.2: An example of an out-of-band re-authentication

From the point of view of privacy this is the best solution, as personalized services do not gain any more information by adding (possibly temporary) contact information to the requests. And in case of errors or re-authentication no information is disclosed to other services in the chain (see figure 7.2.) Assuming that the client signs the request immediately, no other service in the chain will recognize the re-authentication. The drawback is that a re-authentication in this way will most likely be asynchronous (for example using e-mail). It is not very likely for all users to have their own server running which services can contact for re-authentication.

The re-authentication request itself has to contain at least the fields mentioned above. Also, the request should be encrypted by the sending service with the public key of the user.

7.2.2 Roll-Back Re-authentication

Using this method the re-authentication request is passed step by step back to the client. Beginning from the last service, e.g. service D, a re-authentication request will go through the whole chain backwards until a trustworthy service or the user itself is reached. The request is then signed and sent back. (fig.7.3). Each entity in the chain will learn that something is going on, and depending on the content of the request private information may be disclosed.
Using this method it is very important to encrypt the content of the re-authentication request. Otherwise, every involved service which transmits

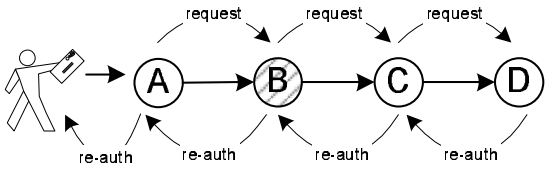

Figure 7.3: Roll-back re-authentication

the request will gain additional information about the process. But even if the content of the request is encrypted and the identity of the requesting service is masked through some session-id privacy is threatened. In our introductory example, when service E issues service A an encrypted and "anonymous" re-authentication request, service A can reason that the request originated from service E, as no other service in the workflow would issue such a request. Thus, service A learns that the user has to have some problems in context with his financial situation. From the aspect of privacy roll-back re-authentication is not the best solution.

Contrarily to the out-of-band mechanism, this re-authentication is a synchronous possibility to get in contact with the user. This is because, the client is already logged in to service A and the re-authentication request is eventually presented to the user through service A.

7.2.3 Ticket-Server Solution

Similar to MS-Passport or the Kerberos authentication system ([20],[17]), this solution uses an additional ticket server (TS). As depicted in figure 7.4, in parallel to accessing the agent's portal (service A) the user signs in at the ticket server. Whenever an authentication is needed the service in question contacts the ticket server.

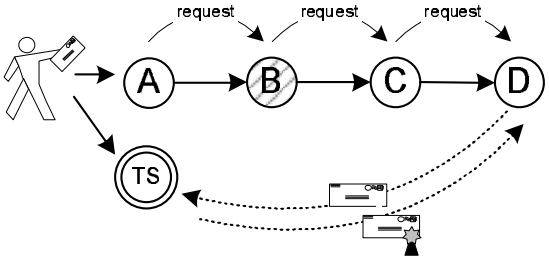

Figure 7.4: Re-authentication by using a ticket server

After the user is successfully signed in at the ticket server, the ticket server is allowed to perform re-authentication requests by signing these requests on behalf of the user. Therefore, the server replies with tickets to the requesting services. The re-authentication request that is sent to the ticket server should look like the other requests described earlier. The ticket itself is signed by the ticket

server using its private key. In order to prevent unrelated services requesting an authentication ticket from the server, the server has to generate a session-id which the user will tie to his request to service A by using cryptographic techniques. Otherwise the ticket server would have to make plausibility checks on the re-authentication requests which is impractical in most but the trivial cases. Furthermore, the ticket server would need more information than a session-id. This in turn may create new privacy problems.

The main advantage of this solution is that in the case of a required re-authentication no other service will learn about it. Also, the ticket server itself has no idea about the other involved services which do not require re-authentication or the whole process as such. Moreover, this solution is very comfortable for the user because he is not burdened with the re-authentication. This is why there will be no additional time-delay caused by the client while answering the request. And so, this re-authentication can be made synchronous. The drawback is that the user has to have absolute trust in the ticket server itself. After all, the ticket server acts on his behalf. Therefore this server must be maintained by a trustworthy independent party. Of course, such a server will be a prime target for attacks.

7.2.4 Communication Server Solution

This solution is similar to the ticket server solution. However, here the so-called communication server (CS) does not act on behalf of the user but is only used as contact and communication point (fig.7.5). This scenario does not suffer the drawbacks of the previous solution.

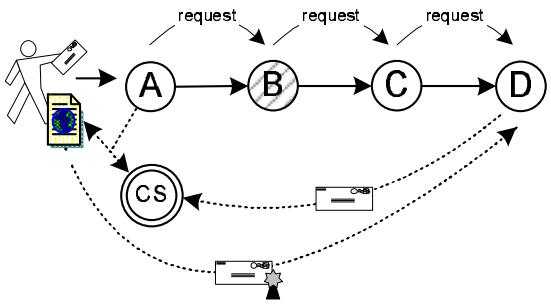

Figure 7.5: Re-authentication through a web-server

The user contacts the CS and is reachable through the CS during the time of the process. For example, if the services have web-interfaces the communication server can be a special website. If a service wants to communicate with the client, e.g. because a re-authentication request is needed, it sends a request to the communication server. This implies that the services need to be aware of the IP-address of this web server. Thus, its address has to be propagated within the requests. Beside the IP-address, the requests may also contain some session-id in order to make it easier for the communication server to classify incoming requests. Both information do not provide additional private information about the user and are thus not privacy critical. The communication

server passes on the request to the user who then signs the re-authentication request. The signed response is sent back directly to the requesting service.

This solution is similar to the out-of-band re-authentication. Here, the request is sent to a communication server instead of the client's mailbox. The request will be displayed directly through for example a web-page and the client can sign the request immediately. Therefore, the re-authentication is synchronous which is the difference to a common out-of-band solution. Apart from this difference, the content of the re-authentication request itself will be quite similar to the other solutions. The communication server could also be used to inform the client about the actual status of the process or to send him error messages. Generally, the communication server allows to communicate with the client without harming his privacy. The only precondition is that the communication server is trustworthy and is run by some reliable party.

7.3 Practical Aspects

The models and problems described in this chapter are crucial for applications in an e-governmental environment. Here too, distributed services are used to process transactions initiated by a client. Furthermore, because of the sensitive data involved with governmental transactions, protecting the client's privacy is of paramount importance.

From the discussed solutions in the previous chapter, out-of-band re-authentication cannot be used as some services require a synchronous re-authentication possibility. Roll-back re-authentication discloses too much information and should not be used in a privacy sensitive environment. That leaves the ticket server and communication server methods as options. However, such a server acts as central authentication authority for whichever governmental service the user accesses. Thus, such a server could be used to profile the user's actions. In addition, data protection laws may forbid running such a service in the context of governmental processes. How can this situation be resolved? Depending on the circumstances three solutions exist:

1. **using a private communication server**
2. **using different authentication authorities**
3. **minimize the need for re-authentication**

Ad. 1) If it can be assumed that the user has access to a private communication server (possibly run by a third party), this server can be used to solve the problem. Again, this is a central approach, but the point is that this server is unrelated to the accessed services and it is a conscious decision on the user's part to use and trust that server.

Ad. 2) The second solution is using many different authentication authorities, instead of only one. For example, the first service in a process could act as an authority. Furthermore, the user might choose different authorities for accessing

different services as it is intended by the Liberty Alliance specifications and the federated Single Sign-On approach. This stands out against a central approach clearly, but has the drawback of possible privacy violations mentioned earlier.

Ad. 3) The third solution is to minimize the need for re-authentication. This can be achieved by digitally signing the request and binding it to the current session. The signature can be verified by every service in the chain and thus the user can reliably be authenticated at each service. To prevent sending the complete request and disclosing too much information to each service the request could be split into separate signed parts. Alternatively, the parts could be partially encrypted with the targeted service's public key. However, signing and splitting the request is not possible in all cases.

To sum up, every solution has its benefits and drawbacks and should be applied according to the situation at hand. Based on our experience, a sensible combination of the proposed solutions may solve most problems.

Part III

Practical Work and Implementation

Chapter 8

Introduction and Motivation for this Implementation

The practical part of this thesis deals with the implementation of some main functionalities as defined in the Liberty Alliance specifications. Especially the core functions for the Single Sign-On mechanisms using SOAP are realized. Therefore, the following chapters introduce the related technologies used for the implementation of Project Liberty, e.g. SAML and SOAP. Moreover, chapter 9 summarizes the chosen architecture which underlays the implementation and illustrates with the help of a Single Sign-On process the functionalities of the resulting sample implementation.

8.1 Objectives

The objectives of this implementation are to get a better understanding of the Project Liberty specifications [6] and try to handle the needed technologies, e.g. SAML, SOAP, etc. The whole implementation should be developed by using the Java programming language and Java related technologies. The conceptional design of the Java class hierarchy is very important. Because of the structure of the protocol and the messages defined in the Project Liberty specifications which is merely an extension of the SAML specification, the resulting class hierarchy should represent this fact. Therefore, a SAML class framework should be the basis for all other classes. The Project Liberty related classes have to extend the SAML basic framework. At the end, the Liberty Alliance classes are on top of the SAML related class hierarchy and both, the SAML framework and the resulting Project Liberty framework can be used independently. Of course, not all functionalities can be realized. The resulting Liberty Alliance implementation should allow to demonstrate one Single Sign-On cycle. For this demonstration it is necessary to implement some identity and service providers. These providers are used for demonstration purpose only. Thus, they are implemented by using some trivial servlets. Once again, the main objective of this implementation is to gain a better understanding of the Liberty Alliance

specifications and its surrounding technologies. Therefore, the product of the implementation is a class framework and a sample application demonstrating the Single Sign-On functionalities of the resulting framework.

To sum up, here the objectives of this practical work in catch phrases:

- **aims:**
  - realization of the Single Sign-On process as defined in the *Liberty Browser Artifact Profile*—the main functionalities are: [25],[16]
    - generation of a digitally signed authentication request
    - processing of an incoming authentication request
    - generation of an assertion corresponding to a request
    - generation of SAML artifacts
    - obtaining an assertion by the use of a SAML artifact
    - processing an incoming authentication response including an assertion
  - gaining a deeper experience with underlaying technologies such as SAML, SOAP, etc.
  - creation of a class framework which can be separated into SAML and Project Liberty related classes
- **requirements:**
  - use Java programming language
  - use Java related technologies

8.2 The Liberty Browser Artifact Profile

Chapter 3 gives an outlining generic introduction of the Project Liberty specifications. Moreover, the stepwise description of the application given in chapter 3 deals with the Browser Artifact Profile as stated in the Liberty Alliance specifications. In short, the use case in figure 8.1 illustrates the application of the Browser Artifact Profile. The characteristic of this profile is that the identity provider does not wrap an assertion into an authentication response which is sent to the requesting service provider by adding it to the return URL. In this profile, the service provider which requests for an authentication assertion receives a SAML artifact instead of the authentication response. By using this SAML artifact within a SAML request encapsulated into a SOAP request, the service provider may receive the according authentication assertion wrapped into a SOAP response from the requested identity provider. Therefore, the Browser Artifact Profile is an adaption of the "*Browser/artifact profile*" for SAML as specified in [23],[22].

As mentioned earlier, the main technologies used for the implementation of this Browser Artifact Profile are SAML and SOAP. Therefore, the rest of this chapter gives a brief introduction into both standards. For more details on these technologies beyond the following introduction refer to the specifications supported by the according organizations.

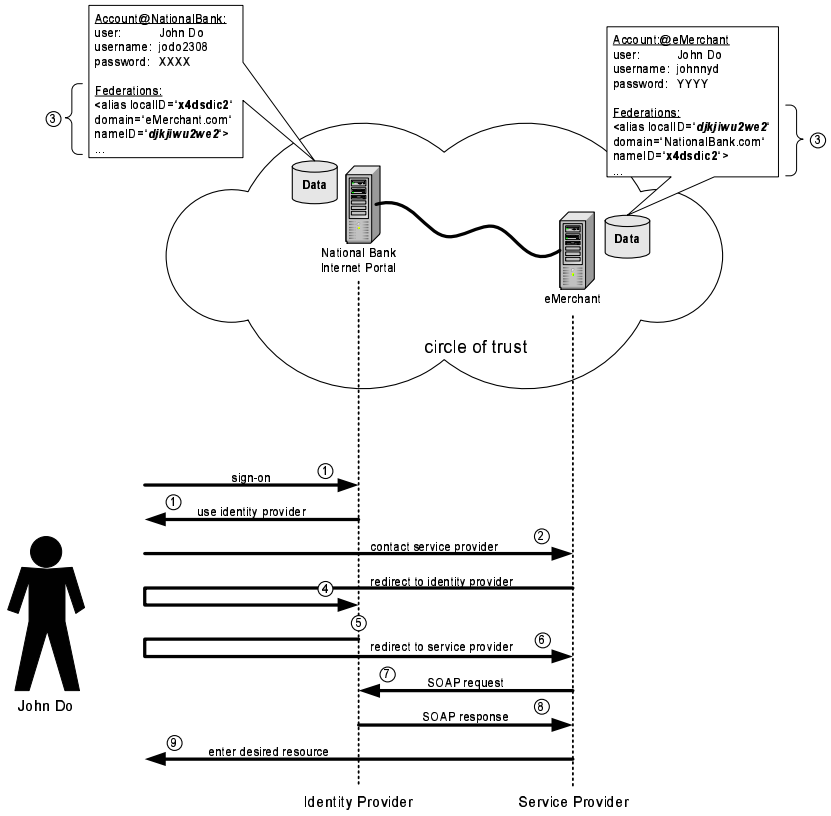

Figure 8.1: Example of Single Sign-On using Project Liberty's Browser Artifact Profile (SOAP binding)[10].

8.3 The Security Assertion Markup Language (SAML)

As mentioned above, the Liberty Alliance protocol bases heavily on the Security Assertion Markup Language. As the SAML specifications [19] state, SAML uses XML to build up a framework which is intended for exchanging security related information. Such an information which is wrapped into a SAML document is called an *assertion* and it belongs to an entity, which may be a human person or a computer or any other subject as well. According to the specifications an assertion may carry information about [19]:

- *authentication acts performed by the subject*
- *attributes of the subject*
- *authorization decisions relating to the subject*

Since SAML builds up on XML, assertions are XML documents whereby an assertion's structure can be nested. An assertion itself is built up from several so called statements. Such a statement may carry data about an entity's authentication acts, attributes or authorization decisions whether the subject is

allowed to enter a special resource or not. These assertions are issued by so called SAML authorities. Figure 8.2 depicts three different kinds of authorities which are defined in the SAML specifications [19].

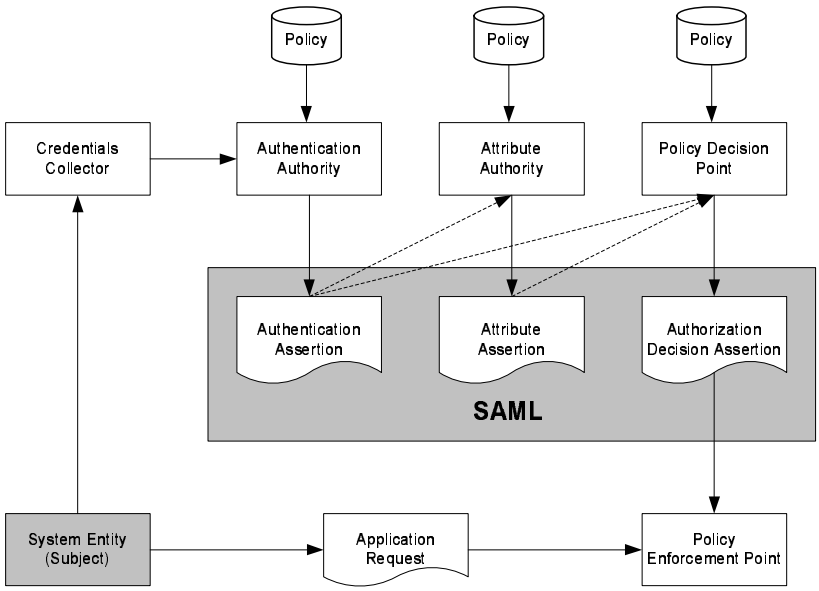

Figure 8.2: The SAML domain model (conceptual) [19]

SAML was mainly intended to allow building up Single Sign-On solutions by reducing the effort of realizing the necessary communication between the authorities and the assertion consuming services. Thus, the authentication authority provides assertions of authentication processes of the user. In figure 8.2, the authentication authority is supported with the entity's credentials by the so called credentials collector. Based on these credentials, the authority is able to generate authentication assertions according to the result of the authentication. With the help of the authentication assertion an attribute authority or the policy decision point can succeed to generate their assertions. For example, assuming that the authentication assertion is presented by an authentication authority, the attribute authority is allowed to provide an attribute assertion containing some further information about the subject belonging to the authentication assertion. The third authority, the policy decision point, acts in the same way. Based on an authentication assertion or on an attribute assertion this authority provides an authorization decision assertion stating whether the entity is allowed to access a certain resource or not. On the one hand, SAML authorities are producers of such an assertion based on various sources such as an external policy store. On the other hand, the authorities may generate an assertion based on a received assertion an authority can act as an assertion consumer as well.

Beside the definition of message formats namely the format of the assertion the SAML specifications constitute an XML based request-response protocol

for interchanging the assertions. By the use of this protocol an assertion consuming service can request a specified assertion belonging to a given subject or entity. This protocol can be based on various existing transport protocols, such as HTTP or SOAP over HTTP. Because of that the Liberty Alliance specifications are extensions of the SAML specifications, the requests and responses depicted in chapter 9 can be considered as examples for SAML messages. Thus, the process flow in the demonstration scenario corresponds to the Browser Artifact Profile as stated in the SAML specifications. For more details on SAML refer to the specifications and documents provided by the OASIS organization [19].

8.4 The Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol is recommended by the World Wide Web Consortium (W3C) and it is intended to give an XML based framework for inter-application communication [7]. Therefore, within SOAP an encoding format and a communication protocol are defined in order to potentiate to realize simple messaging and to support Remote Procedure Calls (RPC) as well. The SOAP definitions include [26]:

- *Message Envelope*
- *Encoding Rules*
- *RPC Convention*
- *Binding with underlaying transport protocols*

8.4.1 The SOAP Message Exchange Model

SOAP bases on a simple request-response dialogs between entities which are also called nodes. Each of these nodes can act as an *Initial SOAP Sender* or an *Ultimate SOAP Receiver* or an *Intermediary*. A node which receives a SOAP message has to process the incoming message. Depending on the result of processing the incoming message, the receiving node has to generate an adequate SOAP response or an appropriate SOAP Fault message if some error occurs.

8.4.2 The SOAP Message Format

A SOAP message is encapsulated within a SOAP envelope which contains an optional header part and a mandatory body block (figure 8.3).

SOAP Header

The optional header is built by one or more header entries. Each of these entries is addressed to a node. The information inside the header entries inform the receiving node how to handle the message. These entries data may be used to enable intermediate nodes to transport the message to the ultimate receiver correctly. Beside the other header entries one header entry element may be dedicated to the ultimate SOAP receiver itself.

The SOAP Header XML element has two important attributes, namely the

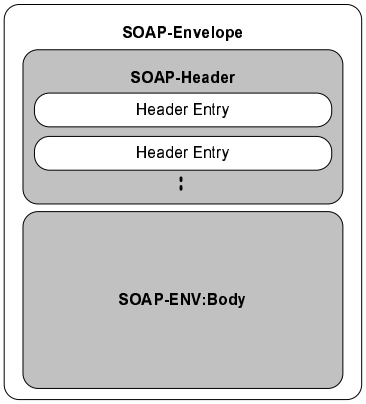

Figure 8.3: SOAP message structure [26]

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope
xmlns:a="http://someurl.com/payment" xmlns:b="http://someurl.com/costumer"
xmlns:c="http://someurl.com/something">
  <env:Header>
    <a:payment
    env:actor="http://www.w3.org/2001/12/soap-envelope/actor/next"
    env:mustUnderstand="true">
      <a:identifier>payid:11332633900-11-4</a:identifier>
      <a:valid>2002-12-31T23:59:59.999-05:00</a:valid>
    </a:payment>
    <b:costumer
    env:actor="http://www.w3.org/2001/12/soap-envelope/actor/next"
    env:mustUnderstand="true">
      <b:id>2176030292</b:id>
      <b:name>John Do</b:name>
    </b:costumer>
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

header block — header entry — header entry

Figure 8.4: Example of a SOAP Header block containing two header entries.

`mustUnderstand` and the `actor` attribute. Depending on the `mustUnderstand`-attribute, the receiving node has to process all SOAP Header blocks addressed to the node and the `mustUnderstand`-attribute is set to `true`. If the node cannot handle it, it must generate a SOAP Fault message. Other blocks without a `mustUnderstand` attribute may be processed or ignored. On the other hand, the `actor` attribute indicates the receiver of a SOAP Header entry. The value of this attribute is presented by a Uniform Resource Identifier (URI). There may exist one header block without an actor attribute. This block is dedicated to the ultimate receiver of the SOAP message as described above. An example of a SOAP Header which consists of two header entries is given in figure 8.4.

SOAP Body

The SOAP Body is made of several body blocks. Body blocks may contain application data, RPC methods and RPC parameters or SOAP Fault messages.

Figure 8.5 gives an example of a SOAP Body element which contains some simple messaging data.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope
xmlns:a="http://someurl.com/payment" xmlns:b="http://someurl.com/costumer"
xmlns:c="http://someurl.com/something">
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    <c:account>
      <c:item>
        <c:article>Book</c:article>
        <c:price>39</c:price>
      </c:item>
      <c:item>
        <c:article>CD-Rom</c:article>
        <c:price>12</c:price>
      </c:item>
    </c:account>
  </env:Body>
</env:Envelope>
```

body block

Figure 8.5: Example of a SOAP Body block.

The body block is dedicated to the ultimate receiver only (that is the reason why it is called an ultimate receiver). This ultimate receiver has to be able to understand the body block and for this receiver processing the body content is mandatory as well. For the implementation task described in this chapter only the messaging capability of SOAP is used. Therefore, the body element contains either application data or, if necessary, a SOAP Fault block. Such a SOAP Fault block is used to convey error information, status information, or both. It consists of four sub-elements: the *faultcode* element, *faultstring* element, *failtactor* element and *detail* element. Moreover, the SOAP specification predefines the following five *faultcode* values [7]:

- *Client*
- *Server*
- *MustUnderstand*
- *VersionMismatch*
- *DataEncodingUnknown*

The *Client* and the *Server* faultcode indicate that the error was caused either on the server side or on the client side. A *MustUnderstand* fault code is sent if the receiver is not able to handle a mandatory header block. The *VersionMismatch* faultcode indicates that there are some troubles concerning the namespaces or element names inside the SOAP envelope. And for completeness, the *DataEncodingUnknown* faultcode value informs that the faulting node could not process the header block, body block or both because of an unsupported data encoding format.

8.4.3 SOAP RPC and SOAP Messaging

SOAP was specified for RPC and it can be used for messaging purposes as well. As mentioned before, in context with SAML, SOAP is used for messaging only.

For completeness, though, this section spots the differences between these two techniques.

Remote Procedure Calls using SOAP can be characterized as follows [26]:

- *used for procedure calls*
- *synchronous*
- *marshaling and unmarshaling of parameters between Java objects and XML*
- *used for simple point-to-point calls*

Remote Procedure Calls are essential for web services. The client calls a function of a web service by sending a SOAP request containing the name of the function and additional parameters. This XML SOAP message is easily generated by the client software. There exist some automatic development tools which are able to generate appropriate Java classes based on the so called *Web Service Description Language (WSDL)* file describing the functionalities and methods of a web service, in order to use the remote procedures very conveniently. The parameters transported by SOAP can be simple, built-in datatypes such as `string`, `integer`, `decimal` and several derived datatypes. Beyond this, even complex Java objects can be passed by SOAP. To do this, SOAP marshals the object into an according XML block. This is done merely by some automatically generated classes. The result of the procedure called remotely is sent back to the client within a SOAP response. The response may contain a single data value using the SOAP datatypes or a whole object marshaled as described. Figure 8.6 depicts an example of remote procedure calls. It illustrates the SOAP messages needed to use a simple Java function to concatenate two given text strings by a remote procedure call initiated at the client.

On the other hand, messaging by using SOAP can be characterized as follows [26]:

- *document-driven*
- *asynchronous and synchronous*
- *used when data is large and fluid*

To send an XML document wrapped into a SOAP message the XML document has to be packed into the body content. Contrarily to SOAP RPC where only a synchronous communication is possible, SOAP messaging allows asynchronus communication as well. This means, that the requesting node must not block until the according response is received. Figure 8.5 illustrates an XML document which is transmitted by using SOAP messaging.

8.4.4 SOAP Binding and SOAP with Attachments

Actually, the SOAP specifications define a binding for HTTP in their request-response model. Beyond it the usage of other transport protocols such as Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), etc. is possible as well [26]. In this implementation, SOAP is used in context with its HTTP binding as required in the SAML and Project Liberty specifications.

RPC call to webservice

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope ...>
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    <concatenate env:encodingStyle="http://www.w3.org/2001/06/soap-encoding">
      <paramA xsi:type="string">Concatenate this </paramA>
      <paramB xsi:type="string">and this.</paramB>
    </concatenate>
  </env:Body>
</env:Envelope>
```

response from webservice

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope ...>
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    <concatenateResponse
    env:encodingStyle="http://www.w3.org/2001/06/soap-encoding">
      <return xsi:type="string">Concatenate this and this.</return>
    </concatenateResponse>
  </env:Body>
</env:Envelope>
```

According Java Method: `String concatenate(String paramA, String paramB)`

Figure 8.6: Example of a SOAP RPC.

SOAP also defines some security measurements. On the one hand, there is the possibility to secure SOAP by using transport level security such as SOAP over HTTPS. On the other hand, the W3C defines some SOAP level security as well (*SOAP SEC*, [8]). The so called SOAP Security Extension secures SOAP messages by using XML Digital Signatures (XML-DSig) which are wrapped into special header entries. These new `Signature` header entries are inserted into the conventional SOAP Header. Thus, depending on the application it is possible to sign one or more arbitrary elements of the SOAP envelope.

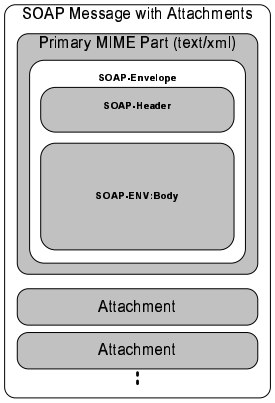

Figure 8.7: SOAP message with attachments [26]

Finally mentioned for completeness, the Simple Object Access Protocol allows to extend a simple SOAP message by attachments. It uses the Multipurpose

Internet Mail Extensions (MIME) type “multipart/related” as container for the SOAP envelope and arbitrary attachments. The envelope and payload can refer to the attachments via relative Uniform Resource Locators (URL) in the SOAP envelope (fig. 8.7).

Chapter 9

The Implementation

9.1 Conceptional Design

As mentioned in the previous chapter, the architecture of the class hierarchy is one of the important objectives in this implementation. Therefore, this section gives an introduction in the chosen design and outlines the concept of the resulting framework. In the course of this chapter, for the Liberty Alliance Project the abbreviated form *"LAP"* is introduced. Therefore classes and packages related to the Liberty Alliance Project are prefixed with LAP. On the other hand, classes and packages related to the SAML specifications are prefixed with the term SAML.

9.1.1 Protocol Classes

Project Liberty is an adaption of the SAML. Thus, the class hierarchy of the protocol related classes were drafted in order to represent these two layers. Figure 9.1 depicts the chosen hierarchy. These protocol classes enable to generate requests and responses of the kind the SAML and Liberty Alliance specifications define. In other words, the methods and members of these classes enable to create SAML-requests and SAML-responses—respectively LAP-requests and LAP-responses.

The base of all classes is the `SAMLProtocolBase` class. This abstract class holds the underlaying XML document which presents the resulting object, e.g. a SAML-request or a SAML-response. Furthermore, some basic functionalities are implemented by this class as well, e.g. a display-function to output the underlaying XML document. Calling the basic constructor of this class by the use of derived constructors, the basic XML document will be initialized and several namespaces are defined. The primary settings and configurations such as the according namespace-URIs and the namespace-prefixes are loaded from a so called `properties`-file. This file contains pairs of keys and values defining the configuration settings needed. Derived from this base class, the `SAMLAbstractRequest` class and the `SAMLAbstractResponse` class go more into detail. They insert into the basic XML document generated by the super class more specified elements and attributes according to the specifications of the desired request or

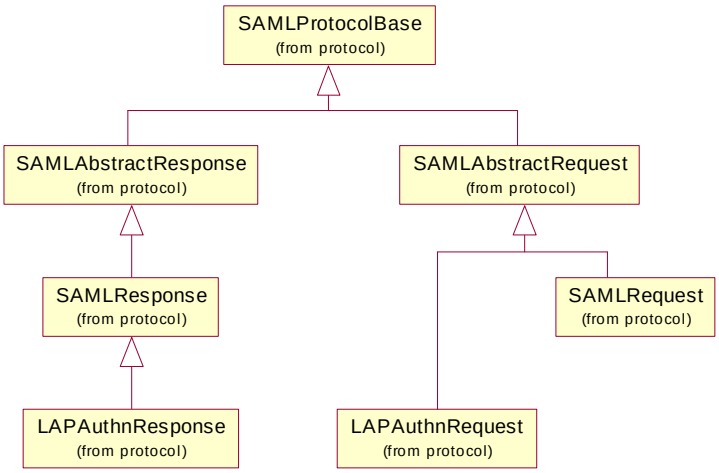

Figure 9.1: Hierarchy of the protocol classes.

response. Finally, the `SAMLResponse` and the `SAMLRequest` classes which are extending the underlaying abstract classes allow to generate proper requests and responses. Therefore, the methods offered by these classes supply the user with all functions needed to insert all required and optional elements and attributes. During the implementation of theses classes special attention was given to prevent the user of building incorrect requests and responses. Despite of this it is necessary to have sufficient knowledge about the SAML specifications in order to understand how to use this framework comprehensively. As mentioned in the introduction of these implementation's chapters, the intention was to implement some parts of Project Liberty. Once again, these parts are:

- generation of a digitally signed authentication request
- processing of an incoming authentication request
- generation of an assertion corresponding to a request
- generation of SAML artifacts
- obtaining an assertion by the use of a SAML artifact
- processing an incoming authentication response including an assertion

Thus, the SAML classes were realized only with functionalities needed to generate LAP-requests and LAP-responses. Further definitions covered by the SAML specifications are not considered and therefore the corresponding functionalities are not supported.

Based on the SAML framework realized by the classes mentioned above the LAP functionalities are possible. Thus, the `LAPAuthnRequest` class is derived from the `SAMLAbstractRequest` class. In a similar way such as used by constructing a `SAMLRequest` object, this class allows to generate authentication requests

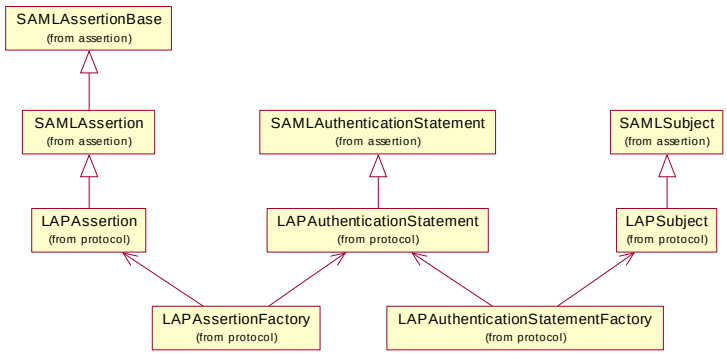

Figure 9.2: Structure of classes used to generate a proper assertion.

according to Liberty Alliance specifications. Beside the basic functionalities supported by the SAML framework additional methods are used to construct well formed requests. On the other hand, the `LAPAuthnResponse` class does not base on the `SAMLAbstractResponse` class but on the `SAMLResponse` class for reasons of implementation. The idea behind this class is the same as before namely to set up an authentication request very comfortably. Therefore, beside the basic functions of the underlaying classes this class offers methods to the user to allow to generate and to custom a proper authentication response. Contrarily to the authentication request, an authentication response contains elements which are more complex and therefore not very easy to generate. One of these complex components of an authentication response is the assertion element containing an authentication statement which contains a subject element. Thus, a kind of tool is needed which allows to construct such an assertion easily. This tool is represented by the class structure surrounding the `LAPAssertionFactory` class (depicted in figure 9.2).

As stated in the SAML and Liberty Alliance specifications one main element in an assertion is the authentication statement element containing a so called subject element. Therefore, this aiding framework of classes contains a hierarchy of subject related classes. The basis is the `SAMLSubject` class which covers the main functionalities corresponding to the SAML specifications. Based on this, the `LAPSubject` class extends the underlaying SAML subject by adding the required `IDPProvidedNameIdentifier` element. With the help of the resulting subject element describing the entity which the assertion belongs to the desired authentication statement can be generated. The whole work of generating a subject element and inserting them to a resulting authentication statement is done by using the `LAPAuthenticationStatementFactory` class. By using the methods of this class it is possible to create a subject element. After the subject element is finalized by the `LAPAuthenticationStatementFactory` the user has to call corresponding methods to set the remaining elements and attributes of the authentication statement. As a result, the factory delivers a `LAPAuthenticationStatement` object corresponding to the specifications. The object itself is derived from the SAML authentication statement and extends this base object by adding further attributes and elements as stated in

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthnRequest ID="request" IssueInstant="2002-10-04T13:45:42+02:00" MajorVersion="1"
MinorVersion="0" RequestID="cDwyFnmqcrXcX3A8MhZ5qnK13F8=" xmlns="http://www.projectliberty.org/
schemas/core/2002/05">
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
         <ds:Reference URI="">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
               <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>ZRyOt9iei/+HpkjlKYXVD6C4S5s=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>CzghjZ8rNgbzPBqD05KNJofJW7pxjSYm1r/BMXKh02KgM9M/1e50+Q==</
ds:SignatureValue>
   </ds:Signature>
   <ProviderID>http://ServiceProvider.com</ProviderID>
   <IsPassive>0</IsPassive>
   <ForceAuthn>0</ForceAuthn>
   <Federate>0</Federate>
   <ProtocolProfile>http://projectliberty.org/profiles/brws-art</ProtocolProfile>
   <AuthnContext>
      <AuthnContextMinimumClassRef>http://projectliberty.org/profiles/brws-post</
AuthnContextMinimumClassRef>
   </AuthnContext>
   <RelayState>23564</RelayState>
</AuthnRequest>
```

Figure 9.3: Example of an authentication request.

the Liberty Alliance specifications, such as the `SessionIndex` attribute or the `ReauthenticateOnOrAfter` attribute. On the basis of this authentication statement object the rest of the assertion can be created. For this task another factory is introduced, namely the `LAPAssertionFactory`. The resulting `LAPAssertion` object represents the desired assertion element which is the core of an authentication response. Even the structure of the `LAPAssertion` class follows the two level architecture. The basis for the assertion classes is given by the `SAMLAssertionBase` class and the `SAMLAssertion` class which is derived from this base class. Again, the `LAPAssertion` class extends the assertions supported by SAML but this derivation does not bring any further functionalities.

Figure 9.3 and 9.4 show an authentication request and the according authentication response generated by using the framework as described in this section. The data in these examples are purely fictive.

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthnResponse IssueInstant="2002-10-04T13:45:54+02:00" MajorVersion="1" MinorVersion="0"
ResponseID="e3xxsORlnGc45Ht8cbDkZZxnOOQ=" xmlns="http://www.projectliberty.org/schemas/core/2002/
05" xmlns:saml="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd"
xmlns:samlp="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-protocol-01.xsd">
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
         <ds:Reference URI="">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
               <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>Va1Qe/j2LZEoALI7vxZLCpyveak=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>Ayrc32fb7g4LQyDQeQox1CTYAN5dKLySZlbtxXPZauMwq1cZvpKvTg==</
ds:SignatureValue>
   </ds:Signature>
   <samlp:Status>
      <samlp:StatusCode Value="saml:Success" xmlns="http://www.projectliberty.org/schemas/core/
2002/05" xmlns:saml="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-
01.xsd" xmlns:samlp="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-protocol-
01.xsd"/>
   </samlp:Status>
   <saml:Assertion AssertionID="8Rc5saZa9EGm//EXObGmWvRBpv8="
InResponseTo="cDwyFnmqcrXcX3A8MhZ5qnK13F8=" IssueInstant="2002-10-04T13:45:54+02:00"
Issuer="www.identityprovider.com" MajorVersion="1" MinorVersion="0" xsi:type="AssertionType">
      <saml:Conditions>
         <saml:AudienceRestrictionCondition>
            <saml:Audience>http://ServiceProvider.com</saml:Audience>
         </saml:AudienceRestrictionCondition>
      </saml:Conditions>
      <saml:AuthenticationStatement AuthenticationInstant="2002-10-04T13:45:54+02:00"
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password" ReauthenticateOnOrAfter="2002-10-
04T13:45:54+02:00" xsi:type="AuthenticationStatementType">
         <saml:Subject xsi:type="SubjectType">
            <saml:NameIdentifier>dk349fkj5yc</saml:NameIdentifier>
            <IDPProvidedNameIdentifier>mnck2kewqy23mq</IDPProvidedNameIdentifier>
         </saml:Subject>
      </saml:AuthenticationStatement>
   </saml:Assertion>
</AuthnResponse>
```

Figure 9.4: Example of an authentication response according to the request of figure 9.3.

9.1.2 Provider and Servlet Classes

By using the framework described in the previous section an identity and a service provider are able to generate the messages needed for this implementation. Figure 9.5 illustrates the framework of classes required for building some primitive providers.

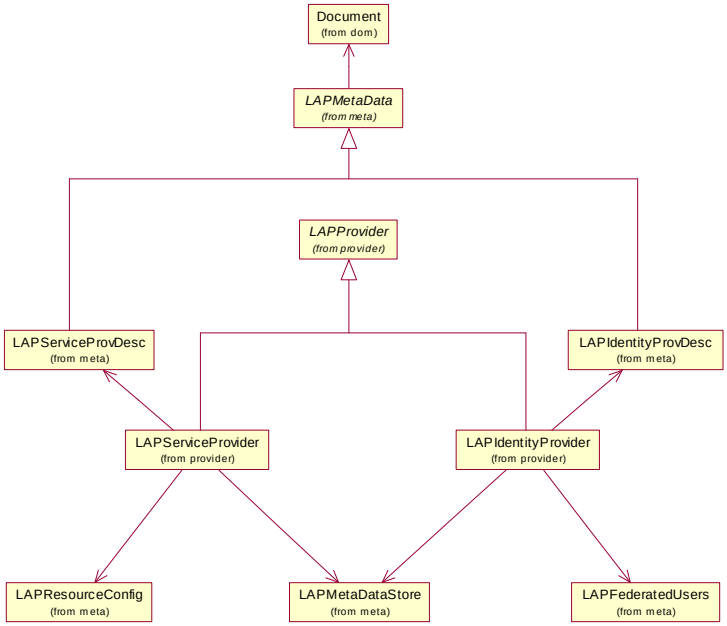

Figure 9.5: Structure of the identity and service provider classes and their most important supporting classes.

The basic class for both the identity provider and the service provider is the `LAPProvider` class. This class implements basic methods such as loading the configuration file (properties file) and loading other necessary XML files determining the characteristics of the provider itself and holding a list of affiliated other providers.

Identity Provider

The whole identity provider used for this implementation is given by an object instantiated of the `LAPIdentityProvider` class. The `LAPIdentityProvider` is derived from the basic `LAPProvider` class. The behavior of the identity provider object is mainly determined by the given configuration file which contains settings such as the location and name of some XML files or of the keystore-file. As illustrated in the schema of the classes (fig. 9.5), the identity provider refers to a so called `LAPIdentityProvDesc` object which contains the identity provider's descriptor. This `LAPIdentityProvDesc` object is an extension of the common `LAPMetaData` class which is used to access information stored in an underlaying XML file very conveniently. Therefore, the identity provider descriptor bases on an XML file containing all properties of the provider, such as:

- type of provider (identity provider or service provider)
- name identifier of the provider (e.g. http://www.IdentityProvider.com)
- accepted profiles
- various URLs
  (e.g.: `SingleSignOnServiceURL`, `SingleLogoutServiceURL`, etc)
- signed requests are required or not

In other words, the XML file containing the provider's description holds XML-entries the kind of *Identity Provider Descriptors* as defined in the Liberty Alliance specifications [6]. Therefore, the content of this file respectively the resulting `LAPIdentityProvDesc` object determines the configuration of the identity provider from the Liberty Alliance's point of view. The second very important supporting object for the identity provider is the `LAPFederatedUsers` object. This class enables to handle all known users and their according information of affiliated accounts at other providers. Therefore, this class supports the identity provider with the local and foreign name identifier of a user if the user's identity is federated. Actually for this implementation an XML file is used for storing the users information. In a real world application a professional database would be used instead. Similarly, the `LAPMetaDataStore` object used by the identity provider stores information about all affiliated providers. As indicated by the name, the `LAPMetaDataStore` class is built up similar to the `LAPMetaData` class. Even the meta data store is using a XML file to manage the information about the affiliated providers. Contrarily, the `LAPMetaDataStore` holds an XML file containing a sequence of several provider descriptor entries. These descriptor entries are based on the provider descriptors as defined in the Liberty Alliance specifications. Thus, the meta data store allows the identity provider to gain information about an affiliated service provider very easily by using the functionalities given by the `LAPMetaDataStore`.

The resulting identity provider object performs the following main tasks in order to fulfill the basic steps needed for Single Sign-On:

- receiving and processing of authentication requests
- generating an according authentication response
- generation of a SAML artifact and temporary storing of authentication responses
- returning of authentication responses by the use of a given SAML artifact

This all happens depending on the given configuration file (properties file) and the given identity provider descriptor. A whole Single Sign-On cycle is demonstrated in the last section of this chapter, thus a qualitative description of these tasks is given as well.

Service Provider

As shown in figure 9.5, the `LAPServiceProvider` class is based on the `LAPProvider` class similar to the `LAPIdentityProvider` class. Again, the behavior of the provider object is determined by both the configuration file and the service provider descriptor. Very similar to the identity provider the service provider descriptor is hold by the `LAPServiceProvDesc` object which is derived from the `LAPMetaData` class. Also the service provider descriptor is loaded from an XML file containing the descriptor data. This descriptor holds the same entries as the descriptor of the identity provider beside one URL entry, namely the `SingleSignOnServiceURL` entry is omitted. This is not the only similarity between a service provider and an identity provider. The `LAPServiceProvider` is supported by a `LAPMetaDataStore` object as well. This time the meta data store contains descriptor entries of affiliated identity providers. Thus, the `LAPMetaDataStore` is used similar as it is done by the identity provider. The `LAPResourceConfig` class is needed to handle the information about the resources of the service provider which requires an authentication by an identity provider. Each site and service a user can request is listed in the underlaying XML file. Therefore, this file or an entry in this file defines the restrictions for accessing the specified resource. Thus, it is possible to define that every user has to be authenticated by some specified Liberty Alliance profile or that it is required to ask for re-authentication, e.g. every hour. This information is needed to generate a corresponding authentication request and to prove if an incoming authentication response achieves the requirements and whether the assertion contained in the response fulfills the restrictions of the desired resource or not.

The resulting service provider object performs the following main tasks in order to support the basic steps needed for Single Sign-On:

- generation of authentication request
- requesting for an authentication response by the use of a SAML artifact
- receiving and processing an incoming authentication response

Servlet Framework

The identity and service provider classes drafted above are responsible for the main work and for processing the Liberty Alliance protocol. Beyond this, there is a need to deploy these providers into the Internet. For the demonstration of this implementation a few very simple servlets are used. As depicted in figure 9.6, there exist two circles of servlets. One is related to the identity provider and the other one to the service provider.

The servlets belonging to the service provider have one central `LAPServiceProvider` object which supports the core functionalities. In the same way, the servlets dedicated to the identity provider integrate a `LAPIdentityProvider` object. The following list describes the basic functionalities of these servlets:

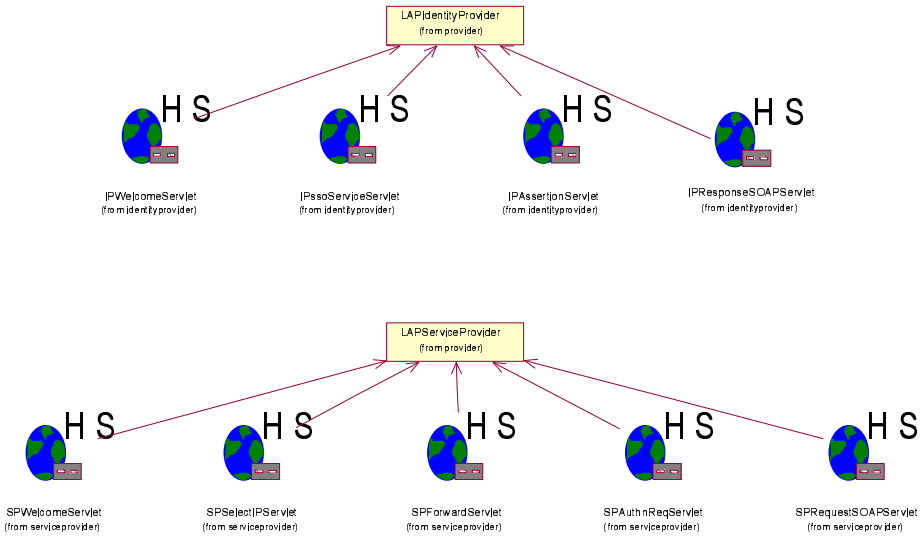

Figure 9.6: Structure of the servlet classes related to the provider objects.

- Service Provider Servlets:
  - `SPWelcomeServlet`
    This servlet welcomes a user at the service provider. At this point the user can chose among several resources to access by following a link. The access to this servlet does not require any authentication.
  - `SPForwardServlet`
    If the user wants to access a restricted resource which is presented by a corresponding link at the welcome-servlet, this servlet checks if the user is authenticated already. Actually, this servlet is not in use in this sample implementation.
  - `SPSelectIPServlet`
    Assuming that the user has to be authenticated for accessing a resource of this service provider this servlet asks the user to chose one of the affiliated identity providers for authentication purpose.
  - `SPAuthnRequestServlet`
    Depending on the result of the `SPSelectIPServlet` this servlet is responsible for generating an according authentication request. This servlet sends this request to the Single Sign-On service of the chosen identity provider.
  - `SPRequestSOAPServlet`
    This servlet is contacted by an identity provider and receives a SAML artifact. This artifact is later used in order to request an assertion by the service provider. Therefore, this servlet generates an according SAML request and sends this request by the use of the SOAP protocol to the identity provider. On the other hand, this servlet is waiting for the SOAP response corresponding to the SOAP request. Further-

more, this servlet processes the incoming SOAP response containing an assertion or a fault message.

- Identity Provider Servlets:
  - `IPWelcomeServlet`
    This servlet presents a standard welcome screen and it is accessible without authentication. It informs a user about the affiliated providers and allows him to sign-on for later use.
  - `IPssoServiceServlet`
    This servlet is accessed by an affiliated provider in order to request an assertion. Therefore, this servlet processes incoming authentication requests and verifies it according to the processing rules stated in the Liberty Alliance specifications. If the request does not match the requirements and rules the servlet does not proceed and provides an according error message.
  - `IPAssertionServlet` This servlet is directly called by the `IPssoServiceServlet`. If no error occurs during the procession of the incoming request the servlet prepares a corresponding assertion. As a result, this servlet returns a SAML artifact. With the help of this artifact, an affiliated provider can ask for the prepared assertion by the use of a SOAP request. Note that this servlet and the `IPssoServiceServlet` may be realized in one servlet only, but for convenience processing the incoming request and preparing an assertion is separated in this sample implementation.
  - `IPResponseSOAPServlet`
    This servlet implements the SOAP endpoint where an affiliated provider may request an assertion by the use of a SAML artifact. If the incoming SOAP request containing a SAML request which includes the artifact corresponds to a prepared assertion this servlet returns a SOAP response including the desired assertion. Otherwise this servlet answers with an error message.

The last section 9.3 of this chapter describes a complete Single Sign-On authentication cycle by using this implementation. During this stepwise description the usage of the servlets is illustrated in a more detailed way.

9.1.3 Exception Classes

Exception classes have been implemented in order to react in the case of errors or faults. Figure 9.7 depicts the hierarchy of exception classes used in this implementation. All exception classes are derived from the `Exception` class as defined in the Java framework. Even the exception classes are separated in SAML related exceptions and exceptions dedicated to Project Liberty according classes. As the names of the exceptions state they are thrown in the following situations:

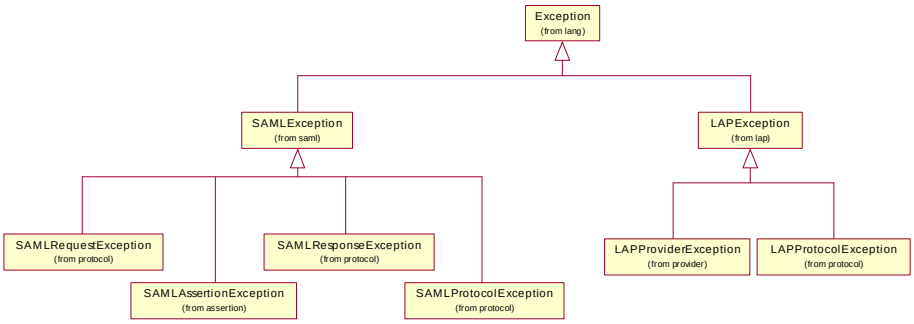

Figure 9.7: Hierarchy of exception classes.

- `SAMLException`
  This exception class is the base class of all exceptions used in the SAML framework. All SAML exceptions are derived from this exception.
  - `SAMLProtocolException`
    These exceptions are used by all protocol classes such as the `SAMLProtocolBase` class.
  - `SAMLRequestException`
    This class of exceptions covers new exceptions occurring in connection with the generation of SAML requests.
  - `SAMLAssertionException`
    This class of exceptions covers new exceptions occurring in connection with the generation of SAML assertions.
  - `SAMLResponseException`
    This class of exceptions covers new exceptions occurring in connection with the generation of SAML responses.
- `LAPException`
  This is the base class for all Project Liberty related exceptions such as the following two.
  - `LAPProtocolException`
    These exceptions are thrown from all LAP classes which are involved in the creation of authentication requests, authentication responses and assertions.
  - `LAPProviderException`
    These exceptions are used by provider classes only namely by the `LAPProvider` class, `LAPIdentityProvider` and `LAPServiceProvider` class.

9.2 Used Technologies and Packages

For this implementation, a number of Java packages and technologies are used. The environment used to implement this example was the *Java Development*

Kit (JDK) version 1.4.0. Based on this, the following supportive packages are in use:

- **Apache Xerces 2 Java Parser**
  This parser supports a convenient interface for working with XML documents. [4]
- **Apache Tomcat 4.0 Servlet Engine**
  The providers in this implementation are realized by the use of servlets. Thus, the Tomcat servlet engine and their related packages are used to implement these servlets. The sample implementation uses the built-in session and user management facilities offered by Tomcat.[3]
- **Apache XML Security 1.0.4**
  This package provides the interface and functionalities desired in order to sign and verify XML documents such as requests and responses.[5]
- **SUN Java Webservice Developer Pack 1.0**
  The `JAXM` package included in this pack supports convenient functions to realize SOAP messaging as required in this implementation.[29]

These technologies build the underlaying basis for this sample implementation. For more information about these packages refer to the documentation offered by the corresponding references.

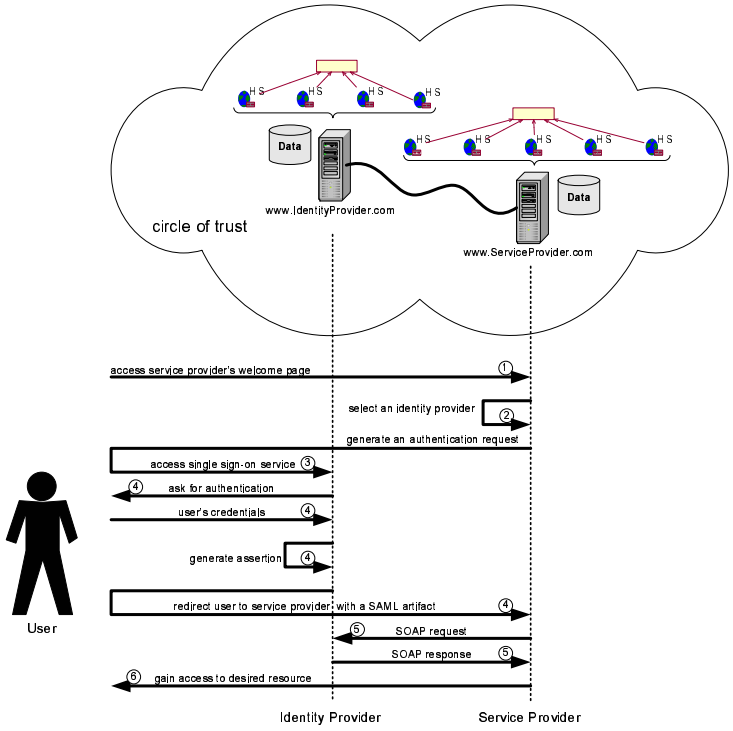

Figure 9.8: Process flow of a Single Sign-On cycle in this sample implementation.

9.3 Demonstration Scenario

To complete this chapter, this section presents a complete Single Sign-On authentication process by the use of this sample implementation. It shows the main tasks of the SAML and LAP framework and it illustrates the cooperation of the servlets used for this demonstration scenario. This description assumes that a user accesses the welcome-page of a service provider and wants to access some resources with restricted access conditions at this provider. Therefore the user is engaged to authenticate himself by the use of an affiliated identity provider. Figure 9.8 and the following enumeration illustrate this process step by step. This example follows the basic Liberty Alliance profile as described in chapter 3.1.2. The explanation in this section is limited to some technical details in order to complete the understanding of the concept of this implementation.

1. **User Accesses the Service Provider**
   First, the user accesses the welcome site of the service provider (www.serviceprovider.com), which is realized by the `SPWelcomeServlet`. If the user follows the link to the restricted area of the service provider, which is displayed on the right margin of the page, the user's browser is redirected to the `SPSelectIPServlet` running on the service provider's server (servlet engine). If this is the very first action calling a servlet of the service provider, within this servlet the underlaying `LAPServiceProvider` object is initialized for the first time. All the other servlets are using this object. Figure 9.9 depicts a screen shot of this welcome page.

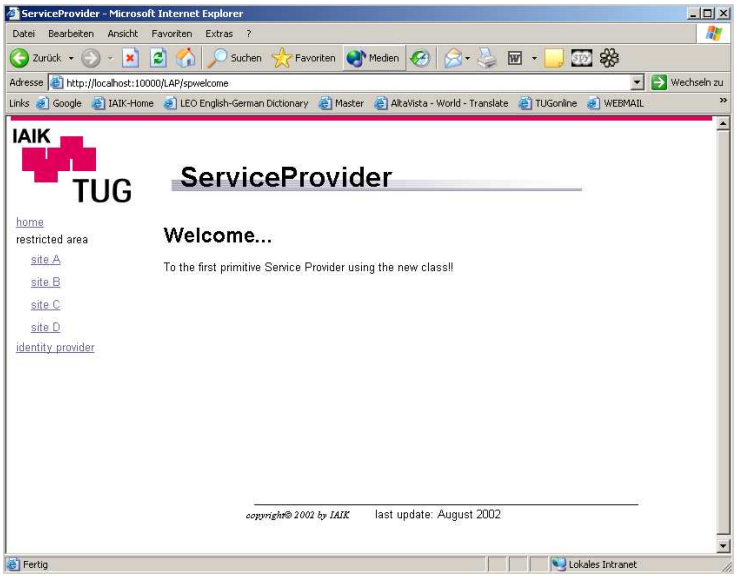

Figure 9.9: Screenshot of the identity provider's welcome servlet.

2. **Chose an affiliated Identity Provider**
   At the `SPSelectIPServlet` the user is enforced to chose among several affiliated identity providers. In this example, the federation of identi-

ties and the affiliation of providers are assumed to having been established already. This means that the identity providers displayed in order to select one of them are affiliated already and that both, the identity and the service provider have a unique name identifier associated with the current user. In other words, the provider affiliation and the identity federation are not touched by this sample implementation. After the user has selected one of the displayed identity providers the user's selection and his chosen resource which he wants to access are passed to the `SPAuthnRequestServlet`. Figure 9.10 depicts a screen shot of the page displayed by the `SPSelectIPServlet`.

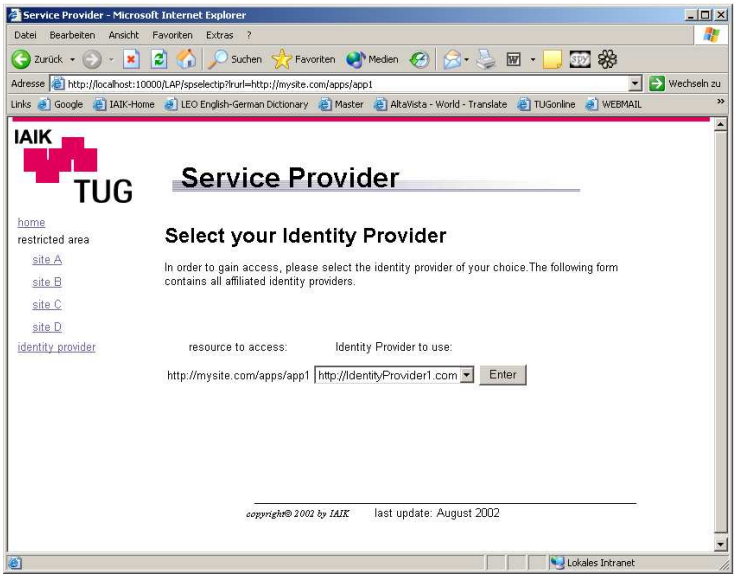

Figure 9.10: Screenshot of selecting among various affiliated identity providers.

3. **Generate an Authentication Request**
   The `SPAuthnRequestServlet` running at the service provider's servlet engine is responsible for generating an authentication request according to the selections of the user. Thus, the content of this request depends on the authentication requirements of the requested resource. Therefore, elements such as `ForceAuthn`, `ProtocolProfile` and `AuthnContext` are given by the properties defined in the resource XML file of the current service provider. Figure 9.3 depicts such a request. Finally, the request has to be signed digitally by the use of the service provider's private key. The whole request generation process is done by calling the `genAuthnRequest` and signing method of the underlying service provider object. The created request is sent to the well known identity provider's Single Sign-On service URL by using a SSL connection.

4. **Access the Identity Provider's SSO Service**
   At the identity provider, the incoming request is checked for correctness and if the signature of the authentication request is valid and whether the

requesting provider is affiliated with the identity provider or not. This task is performed by the `processAuthnRequest` method of the underlaying `LAPIdentityProvider` object. If the request has been accepted the user is asked for his authentication information. Actually in this implementation, very rudimentary providers are realized only. Thus, the session and user management are done by the Tomcat servlet engine. Therefore, verifying the incoming request is done by the `IPssoServiceServlet` and creating the assertion is done by the `IPAssertionServlet` which is protected by a security realm requiring user authentication. Thus, for processing the incoming request no authentication is required. Only if the request is valid, the user is asked for his credentials because the user is redirected to the `IPAssertionServlet`. By using a random index which refers to the temporarily stored request this servlet generates an assertion. If the user is already authenticated the `generateAssertion` method of the identity provider is used to generate an assertion according to the request and to the user's authentication profile. Moreover, this assertion is stored at the identity provider and the according SAML artifact is returned to the requesting service provider. Figure 9.11 shows the Single Sign-On servlet of this implementation.

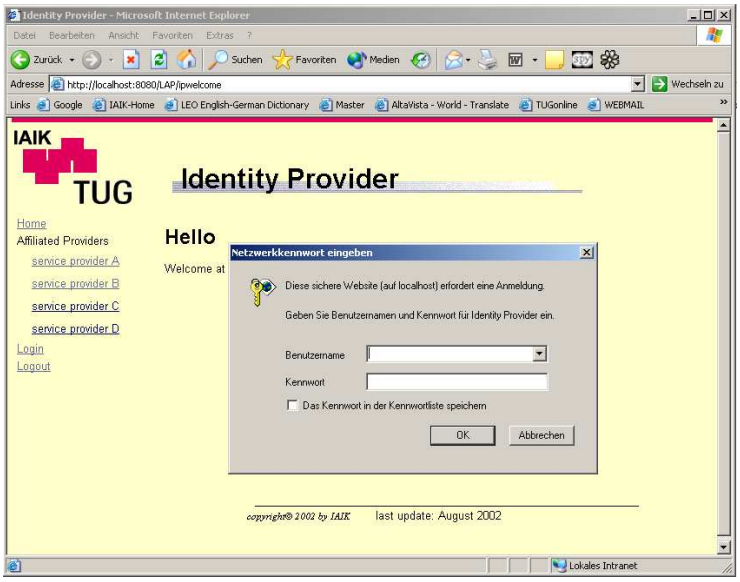

Figure 9.11: Screenshot of the sign-on service presented by the identity provider.

5. **Request for an Assertion**
   The generated SAML artifact is sent to the `SPRequestSOAPServlet` of the requesting service provider. As described in chapter 3.1.2 the second part of the received artifact presents a hash value of the name identifier of the sending identity provider. Therefore, by the use of this fragment of the artifact the service provider is able to identify the corresponding identity provider. Thus, this serlvet generates a SOAP request including a SAML request which contains the SAML ar-

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header/>
  <soap-env:Body>
    <samlp:Request xmlns:samlp="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-
protocol-01.xsd" IssueInstant="2002-10-07T11:57:15+02:00" MajorVersion="1" MinorVersion="0"
RequestID="f6uXcMBYfKU7OH+rl3DAWHylO9A=">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:Reference URI="">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
               <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>BP638ZP1OQRonNaQoTI/LSuGWBU=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>KKLh1Jh0gKyt5EhENuV/QHx2UAVec/G2ZJTfO5vCgKhTimpDffhj0g==</
ds:SignatureValue>
      </ds:Signature>
      <samlp:AssertionArtifact>AAMCrV%2FS5xHYh5e9rUUmsm0Fa37gdOCnYbG%2B1nlZ3Dcbki2k411L2cNW</
samlp:AssertionArtifact>
    </samlp:Request>
  </soap-env:Body>
</soap-env:Envelope>
```

Figure 9.12: Example of a SOAP request containing a SAML artifact.

tifact in order to ask for the prepared assertion. After having signed the request it is sent to the SOAP node of the identity provider. Figure 9.12 depicts a SOAP request containing a SAML artifact wrapped into a SAML request. At the identity provider, the `IPResponseSOAPServlet` reacts to an incoming SOAP request by calling the `onMessage` method of the underlaying SOAP-servlet class. Therefore, the incoming request is proved and if the included artifact can be resolved into a corresponding temporarily stored assertion this assertion is wrapped into a SOAP response. This response is sent back to the requesting `SPRequestSOAPServlet`. If an error occurs during the processing of the request or if the requested assertion can not be found, a fault message is returned to the asking service provider.

6. **Processing an Incoming Assertion**
   Back at the service provider, the `SPRequestSOAPServlet` is waiting for the SOAP response. If the received response is not a SOAP Fault message, this servlet has to process the assertion included in the response (figure 9.13 depicts an incoming SOAP response containing an assertion). This task is done by the `processReceivedAssertion` method of the service provider object working in the background. Within this method various proofs can be made, e.g. signature validation, proving various elements wrapped in the assertion etc. If the incoming assertion suits the needs in order to access the required resource the user gains access (figure 10.1).

If the user wants to access another site requiring an authentication the user may not be asked to enter his authentication information at the identity provider once again. The user is authenticated already and if the affiliated service provider requests for an assertion the user is not involved in the process of assertion generation. Requesting for a new assertion will be done silently in the same way.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header/>
  <soap-env:Body>
    <samlp:Response xmlns:samlp="http://www.oasis-open.org/committees/security/docs/cs-sstc-
schema-protocol-01.xsd" xmlns:saml="http://www.oasis-open.org/committees/security/docs/cs-sstc-
schema-assertion-01.xsd" IssueInstant="2002-10-07T11:57:14+02:00" MajorVersion="1"
MinorVersion="0" ResponseID="KMdeOV6x23nxoijHXjlesdt58aI=">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:Reference URI="">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
              <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>m+QM53RUVbxDdqLKd4M4SIZK4yg=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>IOibEabwzTrG4IjYg8ZIB41RE7AMgdeVHvMsZZSynpHR8I3t+zkQ9Q==</
ds:SignatureValue>
      </ds:Signature>
      <samlp:Status>
        <samlp:StatusCode Value="saml:Success"/>
      </samlp:Status>
      <saml:Assertion AssertionID="L7y5TABermly4i+8uUwAXq5pcuI=" InResponseTo="n+UxgUPQE6k/t5/
lMYFDOBOpP7c=" IssueInstant="2002-10-07T11:57:13+02:00" Issuer="http://IdentityProvider.com"
MajorVersion="1" MinorVersion="0" xmlns:xsi="http://www.oasis-open.org/committees/security/docs/
cs-sstc-schema-assertion-01.xsd" xsi:type="AssertionType">
        <saml:Conditions>
          <saml:AudienceRestrictionCondition>
            <saml:Audience>http://ServiceProvider.com</saml:Audience>
          </saml:AudienceRestrictionCondition>
        </saml:Conditions>
        <saml:AuthenticationStatement AuthenticationInstant="2002-10-07T11:57:12+02:00"
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password" ReauthenticateOnOrAfter="2002-10-
07T12:57:12+02:00" xsi:type="AuthenticationStatementType">
          <saml:Subject xsi:type="SubjectType">
            <saml:NameIdentifier>dfvvmmvfkmklgg</saml:NameIdentifier>
            <IDPProvidedNameIdentifier xmlns="http://www.projectliberty.org/schemas/core/2002/
05">lkjeioekvmksdfei</IDPProvidedNameIdentifier>
          </saml:Subject>
          <saml:SubjectLocality/>
        </saml:AuthenticationStatement>
      </saml:Assertion>
    </samlp:Response>
  </soap-env:Body>
</soap-env:Envelope>
```

Figure 9.13: Example of a SOAP response containing an authentication assertion.

Chapter 10

Conclusion

In this thesis, I gave an introduction to authentication systems enabling Single Sign-On. I categorized them into centralized and federated systems. Therefore, in chapter 2 of this thesis there was an overview on existing solutions which are following the centralized approach, such as Kerberos and Microsoft Passport. Moreover, section 2.3 outlined some weak points of the centralized architecture. By paraphrasing the work of Kormann and Rubin [18], especially some weak points in context with Passport were pointed out. In this section, the use of cookies was discussed as well. There I summarized, that only non-persistent cookies are usable in matters of security. Contrarily, chapter 3 dealt with Single Sign-On solutions corresponding to the federated architecture. In course of this, the Liberty Alliance and their Project Liberty was introduced. I gave a detailed description of what is a federated identity and I showed how to establish identity federations between various so called providers. Moreover, the details and principles of this federated architecture were presented by explaining a Single Sign-On process using Project Liberty.

In the second part of this thesis the question arose how to deal with trust in situations of Single Sign-On. In other words, the question was how to determine the trustworthiness of an assertion or a request, which is handled along a chain of services. Therefore, in chapter 5 I introduced a metrics to calculate trust and security in such frameworks. The term opinion was defined and an algebra which is based on recommendation relationships was given. With this it was possible to determine the trustworthiness of an assertion or a request without having information of all involved services as shown in chapter 6. Determining trust stepwise through a chain of services led to the conclusion that the trustworthiness of a request will decrease while the value of uncertainty will become higher. Thus, eventually a request may be considered as not trustworthy. In order to complete the request successfully, the necessity for some re-authentication mechanism arose. Furthermore, I have shown how the user's privacy can be harmed by simple error messages. The mere fact of the existence of error messages combined with a knowledge of the workflow may disclose private information to others. Next, in chapter 7 several methods of re-authentication and their impact on privacy have been discussed. A consequence of this analysis was that messages should only contain the absolute minimum information necessary for the services to function correctly. Any more data may harm the client's privacy.

Furthermore, encryption should be used wherever sensible, so that information can be passed to services further down the chain without disclosing it to intermediate nodes. It has also been shown that introducing a trusted third party may have substantial benefits from the point of view of privacy.

Finally, in the third and last part of this thesis I described a sample implementation of some parts of the Liberty Alliance specifications, especially of the Browser Artifact Profile. Thus, I developed a Java framework which allows to generate some simple requests and responses in order to process a whole Single Sign-On procedure. Moreover, after a short introduction to the underlaying technologies such as SAML and SOAP given in chapter 8, chapter 9 drafted the chosen class architecture. To round up, a demonstration scenario given in section 9.3 showed the usage of the resulting implementation by demonstrating a simple Single Sign-On cycle which hopefully results in a successful authentication (figure 10.1). In the course of this implementation, I gained a better understanding of this architecture and of the federated Single Sign-On approach in order to conclude, that the federated architecture is the better solution for the realization of a Single Sign-On systems. Thus, even Microsoft is currently working on a federated authentication system called "TrustBridge".

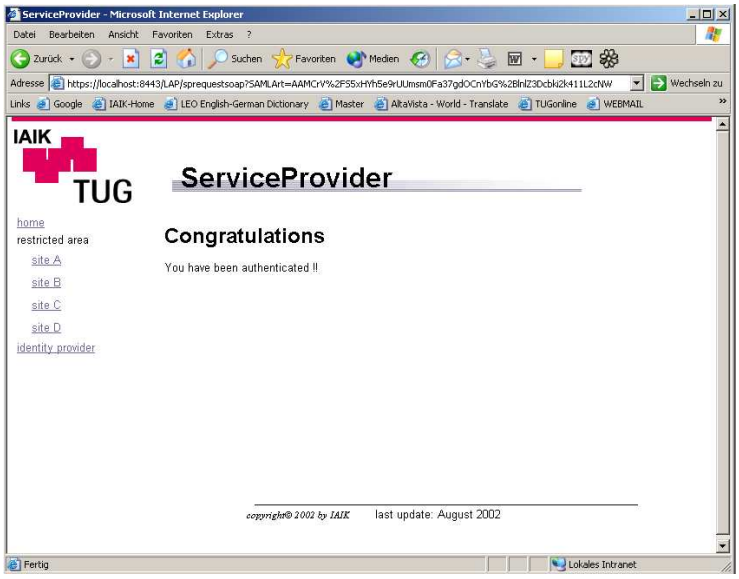

Figure 10.1: Screenshot of gaining access to the desired resource of the service provider.

Appendix A

Abbreviations used in this thesis

3DES	Triple Data Encoding Standard
AS	Authentication Server
DNS	Domain Name System
DNSEC	Domain Name System Security Extension
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
KDC	Key Distribution Center
MIME	Multipurpose Internet Mail Extension
PUID	Passport Unique Identifier
RPC	Remote Procedure Call
RTGS	Remote Ticket Granting Server
SAML	Security Assertion Markup Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SOAP SEC	SOAP Security Extension
SSL	Secure Socket Layer
SSO	Single Sign-On
TGS	Ticket Granting Service
TGT	Ticket Granting Ticket
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XML	Extensible Markup Language
XML-DSig	XML Digital Signatures

Bibliography

[1] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *Proceedings of the New Security Paradigms 97*, 1997.

[2] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Proceedings of the Hawaii Int. Conference on System Sciences 33 , Maui, Hawaii*, 2000.

[3] The Apache Software Foundation. *The Jakarta Project-Apache Tomcat*, 2002. (http://jakarta.apache.org/tomcat/index.html).

[4] The Apache Software Foundation. *Xerces2 Java Parser 2.2.0 Release*, 2002. (http://xml.apache.org/xerces2-j/).

[5] The Apache Software Foundation. *XML Security*, 2002. (http://xml.apache.org/security/index.html).

[6] J.D. Beatty et al. *Liberty Protocols and Schemas Specification 1.0*. Liberty Alliance, 2002.

[7] D. Box et al. *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web Consortium Note, May 2000.

[8] A. Brown et al. *SOAP Security Extensions: Digital Signature*. World Wide Web Consortium Note, February 2001.

[9] ECSC-EEC-EAEC. *Information Technology Security Evaluation Criteria (ITSEC)*, 1991.

[10] J. Hodges et al. *Liberty Architecture Overview 1.0*. Liberty Alliance, 2002.

[11] International Standardization Organisation (ISO). *Evaluation criteria for IT security (ISO/IEC 15408:1999)*, 1999.

[12] A. Jøsang. The right type of trust for distributed systems. In C. Meadows, editor, *Proceedings of the 1996 New Security Paradigms Workshop*, 1996.

[13] A. Jøsang. Artificial reasioning with subjective logic. In Abhaya Nayak, editor, *Proceedings of the Second Australian Workshop on Commonsense Reasioning*, 1997.

[14] A. Jøsang. An algebra for assessing trust in certification chains. In J.Kochmar, editor, *Proceedings of the Network and Distributed Systems Security (NDSS'99) Symposium*, 1999.

[15] A. Jøsang. Trust-based decision making for electronic transactions. In L.Yngstrm and T.Svensson, editors, *Proceedings of the Fourth Nordic Workshop on Secure IT Systems (NORDSEC'99), Stockholm, Sweden*, 1999.

[16] L. Kannappan et al. *Liberty Architecture Implementation Guidelines 1.0.* Liberty Alliance, 2002.

[17] J. Kohl and C. Neuman. *The Kerberos Network Authentication Service (V5).* RFC 1510, 1993.

[18] D. Kormann and A. Rubin. Risk of the passport single signon protocol. In *Computer Networks, Elsevier Science Press*, volume 33, 2000.

[19] E. Maler, P. Hallam-Baker, et al. *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) 1.0.* OASIS, May 2002.

[20] Microsoft Corporation. *Microsoft .NET Passport - Technical Overview*, 2001.

[21] Microsoft Corporation. *Microsoft Passport Software Development Kit Documentation*, 2002. (http://msdn.microsoft.com).

[22] P. Mishra et al. *The SOAP Profile of the OASIS Security Assertion Markup Language (SAML) 1.0.* OASIS, March 2002.

[23] P. Mishra et al. *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) 1.0.* OASIS, May 2002.

[24] Open Group. *Introduction to Single Sign-On*, 2002. (http://www.opengroup.org/security/sso/).

[25] J. Rouault et al. *Liberty Bindings and Profiles Specification 1.0.* Liberty Alliance, 2002.

[26] S. Shin. *SOAP (presentation at the SUN TECH DAYS 2001/2002).* SUN Microsystems.

[27] M. Slemko. *Microsoft Passport to Trouble*, 2001. (http://alive.znep.com/ marcs/passport/).

[28] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open networks systems. In *Usenix Conference Proceedings*, 1988.

[29] SUN Microsystems. *Java Web Services Developer Pack*, 2002. (http://java.sun.com/webservices/webservicespack.html).

[30] B. Tung. *The Moron's Guide to Kerberos*, 1996. (http://www.isi.edu/gost/brian/security/kerberos.html).