View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Theseus

ARCADA

Authenticating users through a Security Token Service

Translation of user credentials

Carolina Lindqvist

Degree Thesis

Information and Media Technology

2013

DEGREE THESIS	
Arcada	
Degree Programme:	Information and Media Technology
Identification number:	4054
Author:	Carolina Lindqvist
Title:	Authenticating users through a Security Token Service - Translation of user credentials
Supervisor (Arcada):	Dr. Tech Göran Pulkkis
Commissioned by:	Helsinki Institute of Physics (HIP)
Supervisor (HIP):	M.Sc. (Tech) Henri Mikkonen

Abstract:
The motivation for this thesis is to inspect the process of translating a type of credential to another. The main focus lies on converting a username and password combination to an X.509 certificate. The components involved in this process, a client, and a Security Token Service that is responsible for the translation, are presented thoroughly. The scope of this thesis also includes development of the client software. The development of the client began in the summer 2012, as part of the author's summer student work for Helsinki Institute of Physics at the European Organization for Nuclear Research (CERN). The work is based on extensive studies of modern standards used in web services and current research.

This thesis work shows how the process of token translation can be accomplished. The path of theory that leads to the end result is also described. The code library that constitutes the client software, was extended with additional functionality, and parts of it was released to the community. The conclusions are, that translation of security tokens can be used as a tool to improve the usability of web services, while still taking into account various security aspects. Token translation can also be of use especially in large organizations, to authenticate users from other trusted organizations, and in this way allow for collaboration and sharing of resources between different parties.

Keywords	Web Service, Authentication, Security Token Service, Token Translation, Security Assertion Markup Language v.2.0, SAML, OpenSAML v.3.0, Enhanced Client-Proxy, Helsinki Institute of Physics, HIP
Number of pages	62
Language	English
Date of acceptance	29.01.2013

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	4054
Författare:	Carolina Lindqvist
Arbetets namn:	Certifikattjänst för användarautentisering - överföring av kreditiv
Handledare (Arcada):	Tekn. dr. Göran Pulkkis
Uppdragsgivare:	Helsinki Institute of Physics (HIP)
Handledare (HIP):	DI Henri Mikkonen

Sammanfattning:
Avsikten med detta examensarbete är att undersöka processen av att omvandla en typ av autentiseringsinformation till en annan. Huvudsakligen behandlas fallet där en kombination av ett användarnamn och ett lösenord omvandlas till ett X.509-certifikat. Programvaran som är involverad i detta, en klient samt tjänsten som utfärdar konverterade identitetsuppgifter (Security Token Service), presenteras noggrant. I detta slutarbete ingår även vidareutveckling av programvaran för klienten. Utvecklandet av klienten påbörjades sommaren 2012, som en del av författarens arbetspraktik för Helsinki Institute of Physics vid European Organization for Nuclear Research (CERN). Teorin baserar sig på studier av de moderna standarder som används i webbtjänster samt på aktuell forskning.

Slutarbetet visar hur autentiseringsinformation kan omvandlas från en form till en annan. I detta ingår en omfattande presentation av teorin som lägger grunden till denna process. Som resultat av detta har även det programbibliotek, som ligger till grund för klienten, utökats med stöd för en annan typ av autentiseringsinformation. Delar av detta bibliotek har även gjorts tillgängliga för allmänheten. Slutsatserna är att omvandling av autentiseringsinformation kan användas som ett verktyg för att förbättra webbtjänsters användarvänlighet, även så att säkerheten beaktas. Inom stora organisationer är det även möjligt att autentisera användare från främmande, förtrogna organisationer genom denna teknik och på så vis främja samarbete samt gemensam användning av resurser.

Nyckelord	Webbtjänst, autentisering, Security Token Service, omvandling av autentiseringsinformation, Security Assertion Markup Language v.2.0, SAML, OpenSAML v.3.0, Enhanced Client-Proxy, Helsinki Institute of Physics, HIP
Sidantal	62
Språk	Engelska
Datum för godkännande	29.01.2013

OPINNÄYTE	
Arcada	
Koulutusohjelma:	Informaatio- ja mediateknologia
Tunnistenumero:	4054
Tekijä:	Carolina Lindqvist
Työn nimi:	Käyttäjätunnistus verkkopalvelulla - kirjautumistietojen siirtäminen
Työn ohjaaja (Arcada):	TkT Göran Pulkkis
Toimeksiantaja:	Helsinki Institute of Physics (HIP)
Työn ohjaaja (HIP):	DI Henri Mikkonen

Tiivistelmä:
Opinnäytteen tarkoitus on tutkia käyttäjätietojen muuntamista muodosta toiseen verkkopalvelun avulla. Pääaiheena on käyttäjätunnuksen ja salasanan kääntäminen X.509-varmennemuotoon. Ohjelmisto jonka avulla prosessi toteutetaan koostuu asiakasohjelmasta sekä (muunnettuja) käyttäjätietoja myöntävästä verkkopalvelusta (Security Token Service) joita työssä kuvataan tarkasti. Asiakasohjelmaa laajennettiin lopputyön osana. Asiakasohjelmiston kehitys aloitettiin kesällä 2012, osana kirjoittajan työharjoittelua Helsinki Institute of Physicsin teknologiaohjelmassa European Organization for Nuclear Researchissa (CERN). Teoriapuoli perustuu standardeihin joita käytetään verkkopalveluissa sekä ajankohtaiseen tutkimukseen.

Työ näyttää miten tunnistautumistietoja voi muuttaa yhdestä muodosta toiseen. Tähän sisältyy kattava esittely teoriasta johon muuntamisprosessi perustuu. Asiakasohjelma on myös laajennettu tukemaan toisentyyppistä tunnistatumistieoa. Osia ohjelmakirjastosta on julkaistu avoimelle yhteisölle. Lopputulos ovat, että käyttäjätietojen muuntamista voi käyttää työkaluna parantamaan verkkopalvelujen käyttäjäystävällisyyttä, myös niin, että tietoturva huomioidaan. Suurten organisaatioiden sisällä on tämän tekniikan avulla myös mahdollista tunnistaa käyttäjiä vieraista, luotettavista organisaatioista. Näin yhteistyö ja resurssien yhteiskäyttö helpottuu.

Avainsanat	Verkkopalvelu, autentikointi, Security Token Service, kirjautumistietojen siirtäminen, Security Assertion Markup Language v.2.0, SAML, OpenSAML v.3.0, Enhanced Client-Proxy, Helsinki Institute of Physics, HIP
Sivumäärä	62
Kieli	Englanti
Hyväksymispäivämäärä	29.01.2013

Contents

List of Figures

Concepts and abbreviations

Assertion (SAML) An XML-encoded message that contains security statements about a subject, e.g. authentication information.

AuthnRequest (SAML) An XML-encoded message that contains a request for an entity to identify itself.

CA Certificate Authority, an entity that issues digital certificates.

ECP Enhanced Client-Proxy

IDE Integrated Development Environment

IDP Identity Provider, a web service that is used to authenticate a user.

Maven A project management tool.

Metadata (SAML) XML-encoded information that describes an entity, for example an SP or an IDP.

OASIS Organization for the Advancement of Structured Information Standards

OpenSAML A code library that implements the SAML standards.

PAOS A protocol for web services, very similar to SOAP, but in reverse.

PKI Public Key Infrastructure.

REST REpresentational State Transfer, a set of principles that define a simple and stateless design for web services.

RFC Request For Comments, a technical document that describes an Internet standard.

SAML Security Assertion Markup Language, an XML based notation for security assertions. In this thesis, the use of this abbreviation refers to the SAML version 2.0 unless otherwise is stated.

Security Token A digital token that can be used to identify a user.

SOAP A protocol for web services to exchange XML-encoded information.

SP Service Provider, a web service that provides some resources or services.

SSO Single Sign-On, to simultaneously log in to several different services.

STS Security Token Service, a web service capable of issuing security tokens.

UML Unified Modeling Language

WS-* Web Services-*, an umbrella term that covers standards related to web services.

XML Extensible Markup Language, a meta-language used to define the syntax of a document.

1. Introduction

In the summer 2012 a client and a service provider prototype that would support the SAML 2.0 ECP Profile (Security Assertion Markup Language 2.0 Enhanced Client-Proxy Profile) were developed as part of the author's tasks as a summer student of Helsinki Institute of Physics (HIP) posted at CERN. After completion of this task, the main focus turned to equipping the client with the capability of requesting security tokens from a web service. The summer job later evolved into this thesis work - an extensive study of the techniques and standards that the translation process builds upon as well as added functionality to the client.

1.1 Background

The process of authenticating a user can be carried out in different ways. The user has to transfer some type of credentials to the authenticating entity. A very common method of creating a credential is to ask for a username and a related password. Another type of credential is a digital certificate, whose functionality can be compared to that of an identity card, that has been issued by a trusted authority. These credentials can also be turned into security tokens. Security tokens are digital items that contain enough information for a user to be identified or authorized to perform some action or to access a resource. Some tokens are more common and easier to use. Other token types burden the user with knowledge of unnecessary technical details and take a long time to acquire. The capability of translating a token from one type to another, allows for the use of various authentication solutions.

1.2 Objective

This thesis examines the possibilities that translation of security tokens bring along. Are all authentication types of equal value and strength? What problems rise from translating a security token from a format to another? Which type of security token is the best for a certain application? How does the token translation process work and why? These are questions this thesis aims to answer and explain. Another goal of this thesis work is to present the components that are involved in translating security tokens, and to improve and enrich the client software.

1.3 Theory

The theory behind this thesis lies in identity management, a subtopic of computer security. Several standards related to web services are needed and relied on. They are mentioned in their proper context. The software components exchange messages that contain elements with information related to the authentication process. It is strongly recommended that the reader takes time to become familiar with these elements, and also with the messages they are contained in while reading this thesis work.

1.4 Implementation

Most of the software development was done in the summer 2012, when the client and a service provider prototype were created. Later, in the fall 2012, the client's functionality was extended. Support for an additional type of authentication token was added. Java was chosen as development language. The benefits were a high degree of platform independence and that the OpenSAML library easily could be used for building the software. The implementation was done based on the SAML 2.0 specification documents which describe the exchange of messages between the client, the Identity Provider (IDP), and the service provider (SP). Later, WS-* specifications were introduced to describe the exchange of messages between the security token service and the client.

1.5 Methods

The prototypes were created using the Eclipse IDE (Indigo) with the m2e Maven plugin. Maven (Apache, 2012) is a powerful project management tool whose main goals are to simplify the build process and to keep the project's quality standards on a high level. Self-signed X.509 certificates were used for authentication and to create a circle of trust between the components for test purposes. The code was tested with unit tests. These tests were found to be very helpful during the development process. Offline testing as well as testing that is independent of other necessary components can be achieved with unit tests. A large range of literature studies, mostly studies of specifications and documentation of the used libraries, were necessary to finish the assignment.

1.6 Scope of the Thesis

Only the relevant parts of the SAML 2.0 (Security Assertion Markup Language) specifications are described in the theory part. This means that many other applications of this markup language are left out. The reader is recommended to consult the specifications, (Cantor et al., 2005a), for further information on how SAML 2.0 can be used.

The client is only capable of accessing plain text resources. Only the client version with the STS extension is capable of translating a SAML assertion into an X.509 Certificate. Here, the main goals of the development of the client are to create a prototype, to be able to use it for test and demonstration purposes. The client is based on a code library, and this library is extended during the development phase.

2. Digital Identity Management

Digital Identity Management concerns handling digital identities in a safe and efficient way. Issues that this field of security addresses are, among others; the identification of a subject, secure authentication methods and a subject's privacy (Stallings, 2011). In the following sections some applications related to digital identity management are presented. The intention is to give the reader a picture of the parts of digital identity management that are related to the domain this thesis belongs to.

2.1 Federated Identity Management

Federated Identity Management deals with sharing and using data related to identities among different entities that are operating together in a federation (Camenish et al., 2007). In a typical use case there are thousands of users, from different organizations, with needs to use a variety of applications. These organizations can form a federation, a union. In this union, Identity Providers (IDPs) handle the user authentication for every user in the federation. Authentication information can then be shared between organizations through the IDP. Users can use their home organization's credentials to log onto any service in the federation. Minutes later, if the user wishes to access a service provided by another member in the federation, the user does not have to perform another login action, since the authentication was done when the user logged on to the first service. This method is called Single Sign-On (SSO). (Stallings, 2011)

The information that is shared between organizations; credentials, attributes, business information needs to be transferred in a secure way that does not compromise the subject's privacy. As its best, no other information is shared, except the fact that the user has been authenticated. The IDP still stores enough information related to the authentication event, to ensure that actions performed by the user can be traced afterwards, if needed. At the moment, various protocols, standards and architectures exist, whose main purpose is to solve these problems.

Shibboleth (Shibboleth, 2012b) is a widely used software collection that allows users from one home organization to access a service of another organization, with the same credentials that are used to identify the user at its home organization. The prerequisite is, that the two organizations are collaborating, trusting each other. This technology is most notably used by universities, to provide students with access to online learning platforms (Moodle, 2012) or (itslearning, 2012). In Finland, the main identifying organization, HAKA (CSC, 2012), allows a user registered at a member university to log on and use online resources, i.e. an online article database or another online resource using the same credentials and authenticating only once.

To further help the reader relate to the subject, some examples of software associated with federated identity management are listed:

- IBM Idemix (IBM, 2012) is a cryptographic program library that enables the use of anonymous user credentials. This library can be used to solve issues related to the subject's privacy in online transactions and activities where anonymity is to be preserved.
- Eduroam (eduroam, 2011) provides global access to the Internet from certain sites that offer this service through a service provider. The user is identified with its own credentials that are issued from the user's home organization. For example, a user with credentials from a university in Finland could log on and use the eduroam service at a university in Germany.
- Project Moonshot (Smith, 2012) is a set of technologies that can be used in federated authentication. Moonshot can be used for authentication in cloud services, supports a wide range of user credentials, and builds upon older technology that most organizations have installed.

2.2 PKI

Public Key Infrastructure (PKI) is a concept for creating a digital environment with trusted and secure communication channels between the entities that take part of it (van Tilborg et al., 2011). It is a model for how certificates are handled and used to build up a network of trust. A certificate can be compared to a digital version of an identity card. PKI explains how the certificates should be distributed and used, what kind of components the infrastructure consists of, what kind of scenarios this technique can be used for, which problems PKI can solve.

The PKI concept is based on a asymmetric key pairs that consist of a public key and a private key. The public keys can be bound to certificates that can be used for authenticating a subject. Because of the chain of trust, illustrated in Fig. 1, that lies behind a certificate, a party can decide whether to trust a subject or not. Every certificate that is issued contains the issuer's name and is digitally signed by the issuer. With these facts, the authenticity of a certificate can be proved, if the chain is followed from a node upwards from the owner to the issuer. PKI makes it easier for a subject to decide whether to trust another peer or not. This knowledge can be used to secure a transaction by authenticating both parts, to create a secure communication channel, and also to verify a signature. Another application of PKI is SPKI, described in RFC 2693. SPKI certificates are used to give access rights or permission to perform an action. It contains an issuer that grants the right and a subject that is authorized.

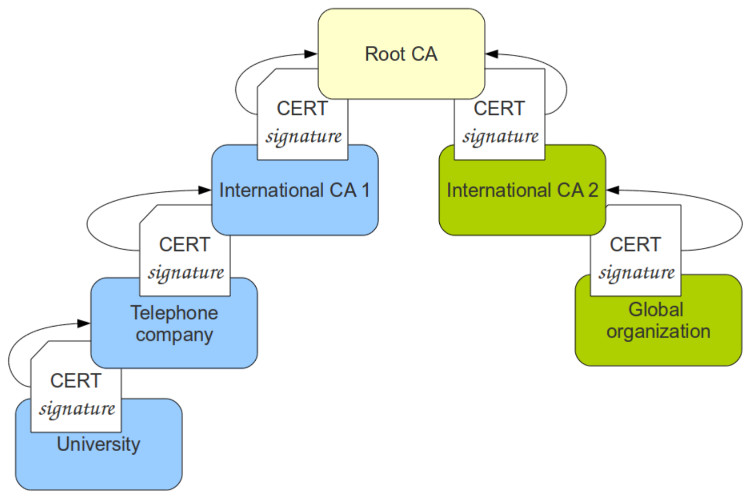

Figure 1: An example of a certificate's chain of trust.

The public key is part of a key pair, of which the other part is the private key. The public key is made available to anyone, and can be used for encrypting a message. The private key, is held private by the owner, the subject. A message that is encrypted with the public key can only be decrypted with the private key that corresponds to the public key. The private key can also be used to digitally sign a message or a document. In that case the corresponding public key can be used to verify the signature. The signature can be verified if the signed document has not changed since it was signed, and the public key that is used for verification is the pair of the private key that was used for signing.

Public keys can also be distributed without a CA, that is, from user to user. PGP/GPG is based on a web of users that trust each other (Alfarez, 1997). Each user can sign a key that is trusted to belong to the said owner. Let us say that user A trusts user B to a hundred percent. Then A receives a key, owned by user C and signed by user B. According to the rules of PGP/GPG the user A then trusts that this key is the property of C, because of B's signature. Partial trust would mean that A would need more than one trusted signee to have signed the key, before A would trust that the key belongs to its owner.

2.3 X.509 Certificates

An X.509 certificate is a digital document that combines the identity of a user with a public key. The authenticity of the information that the certificate contains, is confirmed by a digital signature. The signature is signed using the private key of a certificate authority (CA). A CA is often a global or national organization whose main purpose is to issue trustworthy certificates. If the receiver of the certificate trusts the CA, it is an indication that the certificate also can be trusted. If a certificate is misused or lies under suspicion to be untrustworthy, it can be revoked and placed on a certificate revocation list (CRL). This list will updated by the CA, from which a user can check the validity of a received certificate. Different CAs can also verify the identity of each other and create a network of trust that strengthens the authenticity of a certificate. A less resource-consuming solution for confirming the validity of a certificate is to use OCSP (RFC 2560). It is a protocol that is used to send information that identifies the certificate to a OCSP server. This server performs a lookup and confirms or denies the validity of the certificate. There is no need to transfer the whole list of revoked certificates from a server to a client performing a request, which makes OCSP more efficient.

In general, a certificate contains the following fields: Version, Serial number, Signature algorithm identifier, Issuer name, Period of validity, Subject name, Subject's public key information, Issuer unique identifier, Subject unique identifier, Extensions and a Signature element (Stallings, 2011). The serial number is a unique identifier for the certificate, this number is bound to the issuer, the CA. The subject name field contains the name of the entity that the certificate identifies.

Fig. 2 shows what a certificate might look like. The abbreviations in the "Issuer" and "Subject" fields are the following: C = country, ST = state, O = organization, CN = common name. Of these, the most important is the CN field, the name of the entity. If the certificate is used for establishing a secure connection between two parties, e.g. a client and a homepage, the CN field must be the same as the home page's DNS name or IP address.

The higher up in the hierarchy a CA is, the more authority it has. This brings forth the problem, who can prove the identity of the CA that is placed at the top, the root CA? If there is only one CA, it can issue a certificate for itself. This is a special, self-signed certificate. Self signed certificates are also used as test certificates, but since there is no chain of trust that they can rely on, most applications do not accept this kind of certificate, unless users add it by themselves to the application's trust store. The trust store contains the certificates. Public and private keys are stored in a key store.

```
Certificate:
Data:
    Version: 1 (0x0)
    Serial Number:
        94:d0:e9:00:aa:36:91:1c
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd, CN=localhost
    Validity
        Not Before: Jul  1 11:51:04 2012 GMT
        Not After : Oct  9 11:51:04 2012 GMT
    Subject: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd, CN=localhost
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (1024 bit)
            Modulus (1024 bit):
                00:b3:9d:1b:03:c4:82:91:cc:c2:60:74:14:30:d4:
                66:81:8d:9e:04:2a:7d:81:40:cc:89:03:1f:2e:71:
                5b:0c:d5:17:34:31:44:24:d0:00:21:59:70:3c:38:
                03:da:6a:cd:56:91:24:f3:29:90:4e:1b:22:a7:99:
                9b:e1:60:9b:c9:6f:16:18:9e:d2:85:7a:e7:25:81:
                c7:75:c0:f4:5d:3b:87:bf:83:51:7b:46:df:8b:82:
                e5:f6:39:0d:da:cc:8a:e6:f3:82:48:53:a0:9f:93:
                b2:cc:34:36:7b:57:c8:9b:57:12:33:37:77:3a:29:
                70:6a:4d:44:bf:50:c8:a6:21
            Exponent: 65537 (0x10001)
Signature Algorithm: sha1WithRSAEncryption
    9f:0d:02:6b:27:b5:d8:3d:a5:c1:9b:0c:b7:ba:67:df:c5:e0:
    e5:df:ce:73:7a:1e:1a:78:41:eb:70:4d:23:2a:e4:04:38:34:
    0e:a2:4a:56:f4:22:e0:b9:7d:9a:55:6e:fa:7f:d6:d4:00:c8:
    7e:f6:5e:d7:b5:56:6b:32:a2:d4:14:1f:11:a4:79:fc:2c:1e:
    ea:94:c6:17:b0:8c:a2:be:11:e3:fd:a1:8a:79:66:ce:63:b3:
    b9:5f:70:37:e6:58:c5:be:1c:86:7e:45:f6:57:00:0d:a1:21:
    32:b7:a8:76:f1:0e:8e:99:9f:fb:0e:4e:73:38:fa:22:87:31:
    df:b5
```

Figure 2: A self-signed certificate issued to the entity 'localhost' by itself

2.4 XML Security

Extensible markup language (XML) is a language that is used to encode data, i.e. in order to facilitate the transport of data between applications and ensure interoperability. The structure of an XML document is defined in a schema, written with XML. The schema shows the structure of the document. An XML document is divided in parts, elements, which consist of nodes. In addition, attributes can be added to both the elements and the nodes. In Fig. 3 a sample XML document is presented. This XML document shows a node that is contained in an element. The node is the child of the element, conversely the element is the parent of the node. The `xmlns:ns` attribute defines the namespace. A namespace is a unique name that is used to tell apart similar XML elements that hold different content.

```
<ns:Element xmlns:ns="my-namespace-name" >
   <ns:Node some-attribute="another value">
      The value of the node.
   </Node>
</Element>
```

Figure 3: A sample XML document.

XML security is concerned with protecting XML documents and guaranteeing the integrity and authenticity of the document. Many online services (web services) use XML encoded messages when communicating, for example when the SOAP protocol (described in subsection "SOAP") is used. These messages can be protected with a digital signature, encryption or both. If any part of the document is changed, the signature will be invalidated. Therefore, a signature protects the content of a message. The signer cannot deny that they have signed the document (non-repudiation) and the contents cannot be forged. (Ardagna et al., 2007)

An XML signature is a digital signature that can be signed over the whole document or only parts of it, i.e. some specific elements. Encryption can also be applied to a whole signed document, or only to parts of it. The XML signature is placed in a separate element that is added to the rest of the document. A document can also hold many different signatures. This possibility can be used if many parties are to add their own signatures to a document, i.e. when signing an agreement. The signature is created by calculating a message digest, a short representation of the message, for example using a hash function. The signature element is created by signing the digest and adding the signature to a signature element. In Fig. 4 a sample XML signature is shown. (Ardagna et al., 2007)

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#_a4705b56414395f000d8a0f06b7bb737">
         <ds:Transforms>
            ...
         </ds:Transforms>
         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <ds:DigestValue>9+JdtMdLoII7AcEyl/t8E2dIW+Y=</ds:DigestValue>
      </ds:Reference>
   </ds:SignedInfo>
   <ds:SignatureValue>
     mekYjQGYv3mkoVN0XJyI7Xio2qR5EjmVrtiweKdSc+FtEh8TaxUwg22rvuUaVLfSboMNs
     /vNZEokYYBX4UU+qoNNheQG9fb0zrpyUaR6uJKASeAO+2ntaj2owVoL5Qud3YoWZrvRdi
     XNJHxveLmHDe+aW/vnLs4nIgM6g1IkDjolKKcbdEL3hqTP8lMYT+ZvfFgBKK0xsELAILv
     rFsQszsIKgl/LZ1dbHE07WzF39EVkTFmfvznNfVMzLYWugoCXAt3F9K1E32UFUw5eewWg
     TTvVy4Mw0r4IBvPdq2E8U4hnNoSZLPFpFtFcYRAVUtrjcUo+VBRX+ID3emnTH14n7Q==
   </ds:SignatureValue>
   <ds:KeyInfo>
     ....
   </ds:KeyInfo>
</ds:Signature>
```

Figure 4: A sample XML signature.

The `Signature` element in Fig. 4 contains information about

- the algorithm that was used for encryption, `<SignatureMethod>`.
- the public key that can be used to verify the signature, `<KeyInfo>`.
- a reference that points to the element that was signed, `<Reference URI=...>`.
- the actual signature, `<SignatureValue>` .

If the content has to be signed and encrypted, the order in which these operations are carried out is important. The XML Encryption Requirements standard (Reagle, 2002) mentions some security issues related to this. If something encrypted is signed, the non-repudiation property might not be valid anymore, since the signer can refuse to acknowledge that he or she knew what was signed. If again, unencrypted information is signed, and only the signed information is encrypted, but the signature is left unencrypted, then the signature might contain information that can reveal the original data.

3. Web Services

The Web Services Architecture (Booth, 2004) defines the term web service as a "... software system designed to support interoperable machine-to-machine interaction over a network ...". A web service has an interface described with WSDL (Web Service Description Language) through which clients can use the web service. This is achieved "...using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" (Booth, 2004). Other definitions might omit the use of SOAP messages, since there are applications that can be defined as web services which do not use SOAP messages as a mean of communication. REST (Representional State Transfer) services are examples of web services which are supposed to use only basic HTTP methods and these services can be implemented without SOAP.

3.1 SOAP

SOAP (Gudgin et al., 2007) is a standard for exchanging messages between online services. It can also be seen as a method for invoking remote services. Before the current version, SOAP was an acronym (Simple Object Access Protocol), but this has been removed from the specification. The message consists of a SOAP envelope, a sample envelope is shown in Fig. 5. This `Envelope` element contains a `Header` and a `Body` element. The content in both elements is application specific - the body can for example contain a binary encoded image, XML or plaintext. The header block can contain additional information related to the message body, for examle information on how to process the message. The header can also be absent. If a recepient fails to process a SOAP message, the recepient should send a SOAP Fault message as a response to the sender. This fault message contains information about and possible causes to the error.

```
<soap11:Envelope>
    <soap11:Header/>
    <soap11:Body>
          <soap11:Fault>
              <faultcode>soap11:Server</faultcode>
              <faultstring>Internal server error</faultstring>
          </soap11:Fault>
    </soap11:Body>
  </soap11:Envelope>
```

Figure 5: A sample SOAP v. 1.1 Fault envelope.

The SOAP recommendation (Gudgin et al., 2007) states that SOAP envelopes can be transported over a wide range of protocols. The use of HTTP (Hypertext Transfer Protocol) with SOAP, a common method, is defined in the recommendation by the HTTP binding. Bindings are descriptions of how a message should be transported and handled. SOAP over HTTP is widely used in web applications like reservation systems, online banking systems, weather services... Virtually any web service can be set up to use SOAP. Still, the difference between using SOAP over HTTP and using application specific XML over HTTP is small. SOAP can be used to add an outer layer to application specific XML by wrapping this information in the message body that is sent in an envelope. The advantage of using SOAP is, that it can simplify the processing of a message at intermediate nodes and that this provides a standard way of transporting and processing information compared to using only application specific XML.

SOAP messages are described as SOAP requests and SOAP responses. A client can initiate communication by sending a SOAP request to a web service and receive a SOAP response in return. An example of message exchange is shown in Fig. 6.

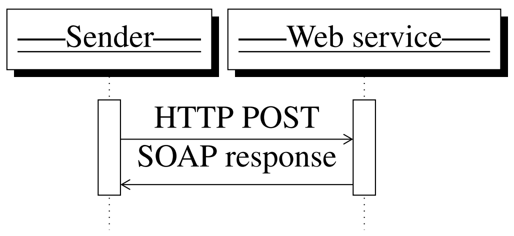

Figure 6: SOAP message exchange

3.1.1 PAOS

PAOS (Aarts 2006) (reverse SOAP) is a protocol, similar to the SOAP HTTP binding, but in reverse. It uses SOAP envelopes and HTTP requests to exchange information. The typically used term, "reverse SOAP" refers to how the protocol operates. The difference between a SOAP and PAOS request-response message exchange is, that PAOS requires two pairs of message exchanges in a scenario where SOAP requires one pair of messages. As illustrated in Fig. 7, a typical use case is when a client requests a resource from a web service with a simple HTTP GET request. The web service answers with a PAOS Request, that the client responds to with a PAOS Response that contains the information that the web service requested. Then the web service returns a resource, for example a home page. The big difference between PAOS and SOAP is, that with PAOS the server side sends the PAOS request message and the client answers with a PAOS response message. These messages are bound to their corresponding HTTP requests as has been described previously.

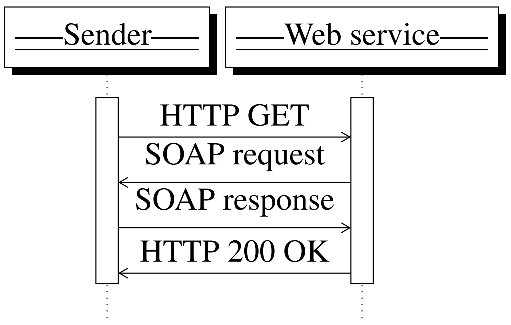

Figure 7: PAOS message exchange

3.2 WS-Security

The WS-Security specification (Nadalin et al., 2012a) defines methods for how the security of a web service that communicates using SOAP messages can be improved. The main means are to encrypt and/or sign SOAP messages. The specification also contains descriptions of how security tokens, a kind of credential, can be added to SOAP messages and sent securely between entities.

WS-Security does not in itself offer any guarantees that information can be transported securely, it can be used with other technologies (SSL/TLS, Kerberos...) to build a safe environment for communication. An envelope that contains a security token, that is signed by some authority, can be used by a subject to prove its identity to any entity that trusts the signer, the authority. A SOAP envelope can also be signed and encrypted if the sender needs to show knowledge of a key or confirm the authenticity of the security token.

3.2.1 Token types

A security token is a type of credential that contains information related to the subject of the credential. In the WS-Security specification (Nadalin et al., 2012a) a token is described as "a collection ... of claims". "Claim" refers to some property the subject concerned claims to have, for example a name or a role. There are different kinds of security tokens: X509 tokens (a certificate), UsernameTokens (a username), XML Tokens (custom tokens), Kerberos tickets ... (Nadalin et al., 2012a). Each of these tokens can be added to a SOAP message and sent to an endpoint, where the token can be used to prove the identity of a user. In SOAP messages, security tokens are included in the `<wsse:Security>` header block that is placed in the SOAP envelope header. Other parts of the message can refer to the token using a reference to the token's `wsu:Id` element. Tokens can also be sent as encrypted data.

Compared to each other, the token types contain different types of information. A username (with a password) can be used for authentication purposes in several types of web services. A certificate can be used for verifying an identity or a signature. The public key from the certificate can be used for encrypting a message. A SAML token can contain more specific information, since the content is a SAML assertion (described in section "SAML"). This information can be personal (a name, profession, an address ...) or technical (access rights or a role). Therefore, the SAML token can be used to transport information about a subject, or the information can be used in a suitable context. However, through extensions, a certificate can be made to contain similar facts.

The X509Token

An X.509 Token is a token type that contains an X.509 certificate. The X.509 Token Profile (Nadalin et al., 2012c) contains the specification for how X.509 certificates are used in conjuction with the WS-Security specification (Nadalin et al., 2012a). When an X.509 token is sent in a SOAP envelope, it can be wrapped in a `<wsse:BinarySecurityToken>` element that is placed inside the `<Security>` element in the SOAP envelope header. Then the X.509 token can be referenced to by a URI. This is illustrated in Fig. 8 . The reference, "#X509SecurityToken" can be used, for example in an XML signature to mark that the token has been processed as part of the signature. X.509 tokens are an example of how certificates can be transported between entities in a network and used for authentication purposes.

```
<soap11:Envelope>
     <soap11:Header>
        <wsse:Security>
           <wsse:BinarySecurityToken
           EncodingType="http://docs.oasis-open.org/wss/2004/01/
           oasis-200401-wss-x509-token-profile-1.0#Base64Binary"
           wsu:Id="X509SecurityToken">
              .... X.509 Certificate ....
              MIIEoYFpcmsDCVrlvKDowEqMvv ...
           </wsse:BinarySecurityToken>
        </wsse:Security>
     </soap11:Header>
     <soap11:Body>
     </soap11:Body>
   </soap11:Envelope>
```

Figure 8: A SOAP message that contains an X.509 token.

The UsernameToken

A UsernameToken is a very simple credential. The main content is a username and a password. With this information, a UsernameToken can be used as a login credential. The password can be transferred in plaintext or transformed into a Base64 encoded SHA-1 hash. Preferably, the token should be encrypted, but the password should at least be transported over a secure connection. For additional security, to prevent replay attacks, a `UsernameToken` can contain a timestamp and a nonce value. If any of these are present, they are concatenated to the password whereafter the Base64 encoded SHA-1 hash value is calculated. The UsernameToken Profile (Nadalin et al., 2012b) recommends the use of both timestamps and nonces. In (Peng et al. 2008) an extended version of a username token is also proposed as an alternative to the HTTP Basic- and HTTP Digest Authentication methods in web services. The additional fields in the extended username token (the timestamp and the nonce) can for example be used to reflect a replay attack, where an old login message is sent to a service in order to acquire unauthorized access to it.

```
<soap11:Envelope>

   <soap11:Header>
      <wsse:Security>
         <wsse:UsernameToken wsu:Id="token-ID">
            <wsse:Username>username</wsse:Username>
            <wsse:Password">password</wsse:Password>
         </wsse:UsernameToken>
      </wsse:Security>
   </soap11:Header>

   <soap11:Body>
      <info:subjectAge>31</info:subjectAge>
   </soap11:Body>

</soap11:Envelope>
```

Figure 9: A simplified example of a UsernameToken

The SAMLToken

A SAML Token, as specified in the SAML Token profile (Monzillo et al., 2006a) is a security token that contains a SAML assertion. SAML assertions are described in section "SAML". This type of security token contains information, claims, that are related to an identified subject. A SAML Token can contain a lot of information about a subject through the SAML assertion, for example, at which point in time the authentication happened, for how long the assertion is valid, to whom the assertion is sent or very specific, personal information about the authenticated subject.

3.3 WS-Trust

WS-Trust (Nadalin et al., 2009) is an expansion to the WS-Security(Nadalin et al., 2012a) specifications. The use of the different token types, that are defined in WS-Security, is described in larger detail. Another significant part of WS-Trust is the trust between entities in relation to each other. The presented trust model also defines how a Security Token Service (STS) works.

4. SAML

SAML (Security Assertion Markup Language) (Cantor et al., 2005b) is an XML based specification for messages that contain statements about a subject (assertions), specifications for messages that are sent between entities in order to create an assertion, and specifications for how to process these messages and what information they should contain. The SAML specifications also define different profiles to use with different implementations. The Web Browser Single Sign-On (SSO) profile (Cantor et al., 2005a) is the most commonly used profile. It allows a user to sign in to different services through a single authentication event. There are different versions of SAML, the latest being SAML 2.0. The earlier version that is still in use is the SAML 1.1 version.

A SAML Assertion is issued by a SAML authority and used by a relying party (Cantor et al., 2005b). The commonly used SAML authorities are IDPs and the relying parties are SPs. An SP can also be named "assertion consumer". This name is given since the SP is the component that uses (consumes) the assertion, i.e. to grant access to a service. An assertion contains the issuer's name, an issuing timestamp, an identifier and the SAML version. It can optionally be encrypted and/or digitally signed if message integrity and confidentiality are required. If there are signatures present, the receiver has to verify these before relying upon the statements from the assertion. The statements are made about a subject entity, and they can be used to authorize the subject to take an action, authenticate them or to connect attributes to them. An assertion can have a time of validity, a reference to the message it is a response to or a recepient address. These fields provide different means for validating the authenticity of the assertion message. Assertions are delivered in `Response` messages, these are usually wrapped in a SOAP envelope. An assertion can also be encrypted, in which case it is stored in an `EncryptedAssertion` element.

SAML elements, like assertions, are messages delivered in `Response` and `Request` messages. There are elements specified for requesting the authentication of a subject (`AuthnRequest`), logging out from a service provider (`LogoutRequest`) or to resolve a previously sent message (`ArtifactResolve`).These message specifications exist in conjunction with their own protocols, as stated in the specifications.

4.1 Components

In this subsection, some main components from federated identity managment are described in the context of how they are used with SAML 2.0. These are the components responsible for requesting and issuing SAML messages, the applications. The main focus lies on presenting the components that are important in the scope of this thesis, and how they are used for the specific purpose this thesis presents. Most of the information is still generally applicable to any SAML 2.0 scenario where the components are used, and the general idea is applicable to any federated identity management scenario.

4.1.1 Service Provider

The Service Provider (SP) is an entity that protects a resource. It is a software component that acts as the server side of an online service. If a client, that has not yet been authenticated, tries to access a resource at the SP, it will be denied access. An `AuthnRequest` is issued by the SP, and the SP will wait for a Response message containing a SAML assertion with statements about the client's identity, that will be received when the client has been authenticated. A sample `AuthnRequest` is presented in Fig. 10. When the SP has received and verified the assertion, the client will be logged on to the service. Thereafter, the SP will decide whether the client has a right to access that resource or not. If the access rights are considered to be sufficient, the client will be redirected to the originally requested resource. Otherwise it will be denied access, but is still logged on. More details on the process can be found in section "3.4 Authentication Request protocol" in (Cantor et al., 2005b).

Noteworthy fields in the `AuthnRequest` are:

- `AssertionConsumerURL` - The final destination for the assertion that is received as a response to this request.
- `ID` - A field that is used to combine an `AuthnRequest` with its proper response that contains this ID.
- `ProtocolBinding` - A field that indicates the protocol that is used, in this case PAOS.
- `Issuer` - The identity of the issuer. The entity that receives this request must know the sender.
- `Signature` - The integrity of the message should be protected with a signature.

```
<saml2p:AuthnRequest
AssertionConsumerServiceURL="/someURL"
ID="cb1bka1mlw561c5apn22dps4bf2a3c89"
IssueInstant="2012-10-02T08:46:16.297Z"
ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:PAOS" Version="2.0">
<saml2:Issuer">https://www.myserviceprovider.org/saml</saml2:Issuer>
<ds:Signature>
   ... An XML signature ...
</ds:Signature>
</saml2p:AuthnRequest>
```

Figure 10: An AuthnRequest issued by a Service Provider.

4.1.2 IDP

The IDP is the software component, installed on a server, where the login and authentication happens. An IDP can issue SAML assertions with statements that concern the identified subject. A simplified version of a SAML assertion is presented in Fig. 11. These statements, or claims can be access rights, usernames, roles or similar information. The authentication can be done in many ways, of which HTTP Basic is a very common solution. How the authentication is done, the method, is out of the scope of the SAML 2.0 specifications.

A single logout is also defined in the SAML 2.0 specifications, the single logout profile (Cantor et al., 2005a). When a client tries to log out, the service provider will request a logout from the IDP. The IDP will distribute the logout request to other service providers involved in this context, if there are any. All of these return a response message and finally a logout response is returned to the original service and the client.

```
<saml2:Assertion ID="_dkgk39g6slo7518eedgk295hdws0nd0f3"
   IssueInstant="2012-10-01T12:23:47.315Z" Version="2.0">

  <saml2:Issuer ... >some-identity-provider-name</saml2:Issuer>

  <ds:Signature> ... An XML signature ...</ds:Signature>

  <saml2:Subject>
    <saml2:SubjectConfirmationData
      Address="88.105.239.1"
      InResponseTo="cb1bka1mlw561c5apn22dps4bf2a3c89"
      NotOnOrAfter="2012-10-01T12:28:47.315Z"
      Recipient="/someURL"/>
    </saml2:SubjectConfirmation>
  </saml2:Subject>

  <saml2:Conditions NotBefore="2012-10-01T12:23:47.315Z"
    NotOnOrAfter="2012-10-01T12:28:47.315Z">
    <saml2:AudienceRestriction>
       <saml2:Audience>https://www.myserviceprovider.org/saml</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>

  <saml2:AuthnStatement AuthnInstant="2012-10-01T12:23:47.314Z">
     ...
    <saml2:AttributeStatement>
      <saml2:Attribute FriendlyName="username" ... >
        <saml2:AttributeValue xsi:type="xs:string">Joe User</saml2:AttributeValue>
      </saml2:Attribute>
    </saml2:AttributeStatement>
</saml2:Assertion>
```

Figure 11: A simplified SAML assertion issued by an IDP.

Noteworthy fields in a SAML assertion are:

- The various timestamps which indicate validity, i.e. `NotOnOrAfter` or the `IssueInstant`.
- `InResponseTo` - This field contains the ID of the AuthnRequest that the assertion is a response to.
- `Recipient` - This field contains the `AssertionConsumerURL` field from the `AuthnRequest`.
- `Audience` - The URL of the Service Provider

- `AttributeStatement` - Holds information regarding the identified subject.

4.1.3 Client

Depending on the profile that is used, in this context, the client is a web browser or an application that acts in a way similar to the web browser. The client requests resources from a service provider. The requests are dealt with as described in the section "Service Provider". When the SP receives the assertion that the IDP issued, the SP decides whether the received assertion is valid or not. In the case that the client has sufficient access rights, the client receives the resource it tried to access from the start. The client and the SP can also establish a session, i.e. using a cookie during this last step.

4.2 Profiles

SAML 2.0 provides different specifications for different use cases. These specifications are called profiles. The most important role of the specifications and profiles is to ensure interoperability between different applications. A commonly used profile is the Single Sign-on profile described in (Cantor et al., 2005a), that enables a user to simultaneously log in to several services using a single password. The profiles describe in detail how SAML assertions are handled, the structure of the messages that are used for communication between the components (IDP, SP, client, intermediaries..), and how different protocols are used to convey a message. Each profile has a URI (Uniform Resource Identifier) that is used during the exchange of messages between entities to ensure that the messages are processed properly.

4.2.1 The SSO Profile

As described in "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0" (Cantor et al., 2005a), there are a few different profiles which support SSO. The Web Browser SSO profile demonstrates the main feature of SSO, to allow the user to log on to several services inside the same organization. The other profiles define the single logout function, the use of an enhanced client (ECP) and a profile for discovering IDPs.

4.2.2 The ECP Profile

The ECP Profile (Enhanced Client or Proxy) is a profile that enables a client to authenticate itself to a service in a non-browser context. It is a more specific version of the Web Browser SSO profile. The main difference is, that the communication between the components uses the PAOS (reverse SOAP) protocol. A web browser without ECP capabilities uses HTTP methods only during the communication process.

In the Shibboleth wiki space, several implementations of ECP clients are listed (Shibboleth, 2012e). These are mainly there for test use. Real implementations of ECP clients are said to be in use, but practical examples are not easily available (Shibboleth, 2012e). Some applications for the ECP profile are; to use services that are not web based in a federation as in (Köhler et al., 2012), or for home automation using SAML as in (Jung et al., 2011).

4.3 OpenSAML

OpenSAML (Shibboleth, 2012c) is a set of libraries that are used for implementing SAML based web services. It is developed by several groups and the code is released as open source. The current version is 2, the 3.0 version is under development (December 2012). Java and C++ versions of the library are available. Well-known software that uses the OpenSAML library are the Shibboleth products (Shibboleth, 2012b).

OpenSAML contains functionalities which allow the developer to create SOAP messages that are relevant for the SAML 2.0 profiles, as well as some functionality for adding elements from the WS-* specifications to the messages. Apart from this, the library also provides functionality for encrypting, decrypting, signing and validating various parts of the messages. It can be used for implementing a service provider, a client or an IDP if an environment specific component is needed.

The 3.0 version (Shibboleth, 2012d) is the newest version of the OpenSAML library. This version is still under development. Some differences between the 3.0 version and earlier versions are added support for different SAML 2.0 profiles, added support for the WS-* specifications, and changes in the configuration of the library.

4.4 Shibboleth

Shibboleth (Shibboleth, 2012b) is a project whose main activity is to develop open source software that enables SSO functionality inside a federation. The earliest work started in year 2000. Among the available components, the IDP and the SP can be considered to be the most important. The Shibboleth consortium is also responsible for the development of the OpenSAML library, on which the products are based on. At the wiki space, `https://wiki.shibboleth.net` , documentation, installation and configuration information related to the products and to the OpenSAML library can be found. Several universities use Shibboleth software to provide access to partners' resources inside a federation. In Finland, the HAKA (CSC, 2012) federation offers SSO services based on Shibboleth products to the federation's member universities.

5. STS - Security Token Service

A Security Token Service, as described in the WS-Trust specifications, (Nadalin et al., 2009), is a type of Web service that is capable of issuing a security token. The tokens are issued from the STS based on information that is sent to it. This information can for example be a username and a password, a certificate, a SAML assertion or a custom type of identification. The requester can also encrypt the information that it sends, in a way that only the STS can decrypt or sign the message with its own key. The STS evaluates whether it trusts the information it received or not, and subsequently returns the requested security token, if the given information was considered valid. If the STS doubts the information it received an error is returned instead. An STS can also be used to translate a token from one format to another.

5.1 The messages

The requester sends a SOAP envelope that contains a `RequestSecurityToken` element in the message body. An example of this element is shown in Fig. 13. This figure contains a SOAP message body with a request for an X.509 token (`TokenType`). If the token request is signed, the public key that verifies the signature may be added to the message. Noteworthy parts of the request message are:

- `RequestType` - it tells the STS what it is supposed to do, in this case, issue a token.
- `TokenType` - it contains the type of token that is requested, an X.509 token.
- `Claims` - inside it is a reference to the token/credential that was placed in the message header and a description of what kind of token it is. The description lies in the `Dialect` attribute, that in this case refers to a SAML token, since the attached token is a SAML assertion.

5.1.1 Key exchange

Occasionally, before a token is acquired, a key exchange process takes place. For example, if an X.509 token is requested, either the client (requester) or the STS (issuer) needs to transfer at least the public key to the other party, since the certificate contains the public key. Keys can also be sent to third parties, for example if the token and key are to be shared with another entity, or if the usage right is forwarded (delegated) to another entity. To ensure that the transfer turns out to be safe, encryption and signatures are used if needed.

One scenario explained in (Nadalin et al., 2009), appendix "A.5.3 Delegated Key Transfer" describes how a key, with permission, can be used by an entity that is not the owner of the key. The scenario is described as follows: "In this example a custom token is issued from party A to party B. The token indicates that B (specifically B's key) has the right to submit purchase orders. The token is signed using a secret key known to the target service T and party A (the key used to ultimately authorize the requests that B makes to T), and a new session key that is encrypted for T. A proof-of-possession token is included that contains the session key encrypted for B. As a result, B is effectively using A's key, but doesn't actually know the key."

The specification (Nadalin et al., 2009) also provides an example of the message that is the result of this transaction, shown in Fig. 12.

```
<wst:RequestSecurityTokenResponseCollection xmlns:wst="...">
     <wst:RequestSecurityTokenResponse>
        <wst:RequestedSecurityToken>
          <xyz:CustomToken xmlns:xyz="...">
               ...
            <xyz:DelegateTo>B</xyz:DelegateTo>
            <xyz:DelegateRights>
                SubmitPurchaseOrder
            </xyz:DelegateRights>
            <xenc:EncryptedKey xmlns:xenc="...">
               ...
            </xenc:EncryptedKey>
            <ds:Signature xmlns:ds="...">...</ds:Signature>
               ...
           </xyz:CustomToken>
        </wst:RequestedSecurityToken>

        <wst:RequestedProofToken>
          <xenc:EncryptedKey xmlns:xenc="..." Id="newProof">
               ...
          </xenc:EncryptedKey>
        </wst:RequestedProofToken>
     </wst:RequestSecurityTokenResponse>
</wst:RequestSecurityTokenResponseCollection>
```

Figure 12: A token message that contains a token, as in (Nadalin et al., 2009).

In other scenarios, the signature can act as a proof-of-possession, that is, a proof that the sender has both the public and the private keys.

```
<soap11:Body>
   <wst:RequestSecurityToken>
      <wst:RequestType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue </wst:RequestType>
      <wst:TokenType>
      http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-tokenprofile-1.0#X509v3
      </wst:TokenType>
      <wst:Claims Dialect="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
         <wsse:SecurityTokenReference>
            <wsse:Reference URI="#_7417184ed45af9d803a0f0d5cf2eadb4"/>
         </wsse:SecurityTokenReference>
      </wst:Claims>
      <wst:UseKey>
         <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:KeyValue>
               <ds:RSAKeyValue>
                  ... A public key ...
               </ds:RSAKeyValue>
            </ds:KeyValue>
         </ds:KeyInfo>
      </wst:UseKey>
   </wst:RequestSecurityToken>
</soap11:Body>
```

Figure 13: The body of a token request message.

A response message that is sent from the STS contains a `RequestSecurityTokenResponse` with the requested security token, as illustrated in Fig. 14. The STS can also respond with an error message if something is wrong, or request more information from a client. The latter is achieved through a challenge, sent in an `InteractiveChallenge` element, for example a PIN code that needs to be entered in addition to the information that the request message contained. Due to the flexible nature of WS-Trust (Nadalin et al., 2009), it is also possible to extend the basic messaging sequences and add additional, custom XML elements with additional information to the messages.

```
<soap11:Envelope>
 <soap11:Header>
  <wsse:Security>
    <wsu:Timestamp>
       <wsu:Created>2011-10-05T08:46:00.601Z</wsu:Created>
    </wsu:Timestamp>
    <wsse:BinarySecurityToken EncodingType=
     "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#Base64Binary"
      wsu:Id="X509SecurityToken">
          ... An X.509 certificate ...
    </wsse:BinarySecurityToken>
  </wsse:Security>
 </soap11:Header>

 <soap11:Body xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/">
   <wst:RequestSecurityTokenResponseCollection>
      <wst:RequestSecurityTokenResponse>
         <wst:RequestedSecurityToken>
            <wsse:SecurityTokenReference>
               <wsse:Reference URI="#X509SecurityToken"/>
            </wsse:SecurityTokenReference>
         </wst:RequestedSecurityToken>
      </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
 </soap11:Body>
</soap11:Envelope>
```

Figure 14: A response from an STS.

5.2 The STS in this thesis

The STS that is used as part of this thesis work is capable of translating credentials into X.509 certificates. Supported credentials are SAML assertions and username-password combinations. The certificates are issued as described in Fig. 13. If the SAML assertion alternative is used, the STS works as a service provider by issuing an `AuthnRequest` when the certificate is asked for by a client. Then the client returns a valid SAML assertion in response to the authentication request. The assertion is wrapped in a `RequestSecurityToken` message. This message also contains the user's own public key. After validating the credentials, the STS responds with a message that contains an X.509 certificate. This certificate binds the user's identity to the provided key. If the username-password combination is chosen, the STS receives a username and a password that it validates as well as a public key that is claimed to belong to the user(name). If these credentials are valid, the STS returns an X.509 certificate.

6. The ECP Client

An ECP client as a concept refers to the Java client prototype, that was created by the author of this thesis during 2012. The initial version of the client could only follow the scenario that is described in the SAML ECP profile specifications (Cantor et al., 2012) and is illustrated in this chapter. Note again, that the main purpose of the ECP profile is to be a more specific version of the Web-SSO profile. The difference between these two is, that the ECP profile operates in a non-browser context. This means, that the client between the SP and the IDP is something (anything) else than a web browser.

To be able to test the scenario, a very simple SP with support for the ECP profile was created by the author. This SP's main function is to issue an `AuthnRequest` and be able to receive and partially validate a SAML assertion. Encrypted assertions are also supported. In the development process, the SP was used for testing the client's functionality. Some minor parts of this SP were also of use for the development of the STS. The client and this SP follow the scenario that is described in the subsection "6.2.1 The whole process - retrieving a resource".

As additions to the client that was created in the summer 2012, some changes were made in late 2012. The command line interface was changed, support for WS-* elements was added and some minor changes to the code were made. In the present version, the command line parameters are processed with the cli library (Commons Cli, 2012). The client was made to communicate with an STS instead of an SP. This is described in the subsection "6.3.1 How the client is used with an STS as an SP". Another extension was the possibility for the client to omit the IDP from this scenario and send the client's credentials straight to the STS.

6.1 Usage instructions

The client is used from the command line. Given parameters are the endpoint URL of the resource the user wishes to access and the (registered) IDP's ID. The available options for the STS client can be seen in Fig. 15. In short, this client can be used for logging on to an IDP, fetching an assertion, and delivering it to an SP endpoint. The resource that is returned can only be in plain text.

```
usage: java -jar client.jar <SP endpoint> [options]
 -certfile <arg>     Define where the certificate will be stored. <arg> is
                     the file name.
 -encrypted <arg>    Option for encrypting a generated private key. <arg>
                     is the key size (1024 || 2048)
 -h,--help           Prints a help message.
 -idp <arg>          The IDP ID.
 -keyfile <arg>      Define where the private key will be stored. <arg> is
                     the file name.
 -nc                 Use nonce- and created fields in the UsernameToken.
 -sts_login          Login and authenticate at the STS.
 -v,--verbose        Prints the messages sent between the client, SP and
                     IDP.
```

Figure 15: Printout from the client's help instructions.

6.2 The architecture

Some UML diagrams (class and sequence) which describe the main classes and the architecture of the current version of the ECP client can be found in Appendix 4. The main parts of the architecture are discussed in detail here. It is advised that the readers familiarize themselves with the UML diagrams while studying this part of the text. A class diagram with the most important classes is presented in Fig. 16. The parts marked in green are the most important changes that were done as part of this thesis.

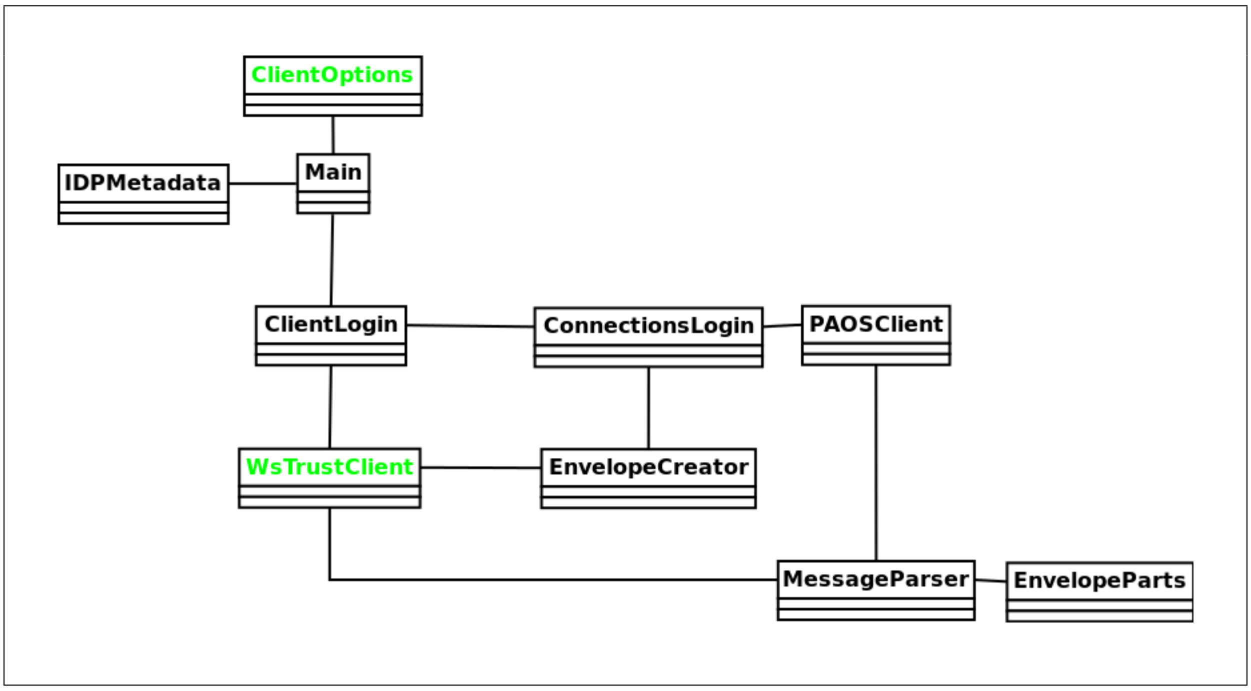

Figure 16: The most important classes in the ECP client architecture.

What follows, is a verbose explanation that can also be read from the sequence diagrams in Appendix 4. When a user starts the application, the given parameters are processed in the `ClientOptions` class and forwarded through the `Main` class to other parts of the application. Information related to the given IDP-ID (its URL) is loaded from a metadata file. Then the client retrieves the SAML assertion through the `PAOSClient` . The `ClientLogin` class is intended to be an intermediary between the `PAOSClient` class and the `WsTrustClient` class. The `MessageParser` part is responsible for reading the incoming bytestream and parsing it into envelopes and envelope parts (headers and bodies). Similar functionality for whole envelopes is available in the OpenSAML library, but it was not used, since the OpenSAML library uses another HTTP client. It was necessary to split the envelopes, since the messages have to be modified (i.e. the headers have to be removed) before an envelope is forwarded to another component.

When the message exchange between the SP, IDP and client has taken place (at `PAOSClient`), the SAML assertion is extracted from the IDP `Response` message and returned to the `ClientLogin` class. This was a solution that aims to make the client more extendable. By adding a class (like the `WsTrustClient` class) to which the SAML assertion can be passed, the client can be extended to forward SAML assertions packed in other message formats to other endpoints quite easily. What happens in the `WsTrustClient` class is, that the SAML assertion is inserted into a SOAP message header. This envelope is sent to the STS, that answers with a SOAP envelope in which the X.509 certificate is included. The certificate and the private key are saved and the execution of the program stops.

6.2.1 The whole process - retrieving a resource

What the client does, is to act as an intermediary between the SP and the IDP. In this section the message flow is presented, as in (Lindqvist, 2012), based on (Cantor et al., 2012).

The client sends a HTTP GET request to the SP. This HTTP request contains two headers which show that the client supports the ECP profile. The `accept` header contains the notation `application/vnd.paos+xml`. In addition to this, a separate `PAOS` header is included. The `AuthnRequest` is also signed by the SP, so that the IDP can verify the identity of the issuer (the SP). The whole step is illustrated in Fig. 17.

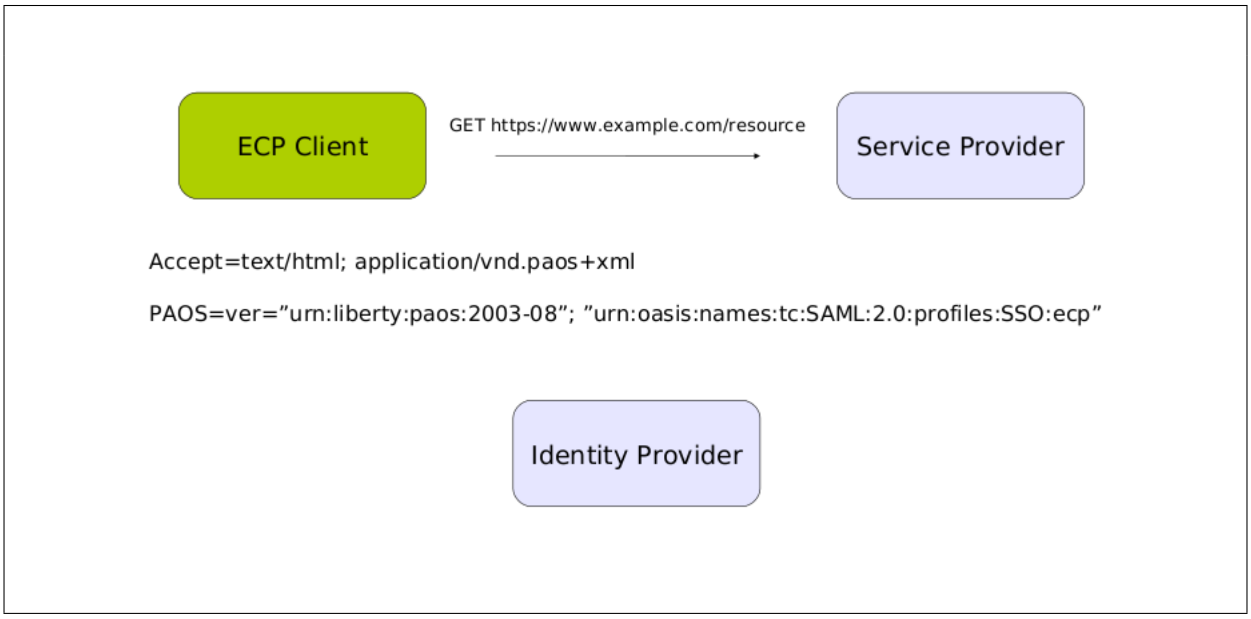
ECP Client
GET https://www.example.com/resource
Service Provider
Accept=text/html; application/vnd.paos+xml
PAOS=ver="urn:liberty:paos:2003-08"; "urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"
Identity Provider

Figure 17: The ECP client tries to access a resource at an SP.

The second step is for the SP to issue an `AuthnRequest` and send it packed in a SOAP envelope to the client. The SOAP header contains two important blocks; the PAOS and ECP Request header blocks. See Fig. 18.

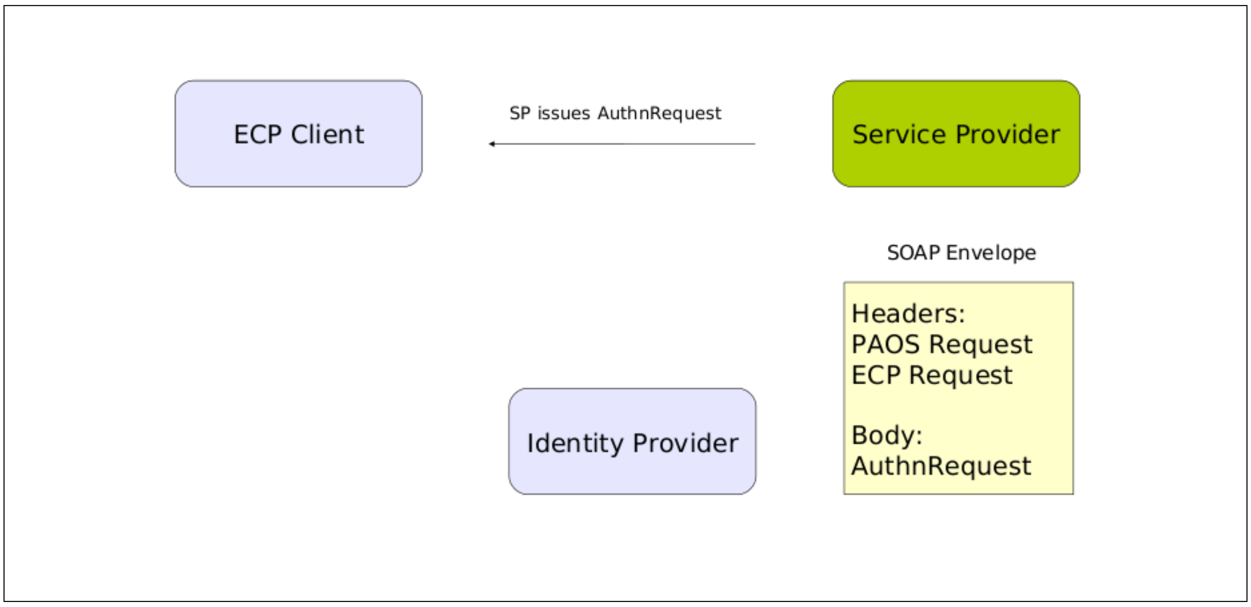

Figure 18: The SP issues an AuthnRequest.

In the client, the header blocks are removed from the envelope, before the client forwards the SOAP envelope with the `AuthnRequest` to the IDP. The information contained in these blocks is stored in the client. The PAOS header block can contain a list of IDPs that the SP wishes to use. The client determines an IDP and forwards the envelope to it. This is illustrated in Fig. 19.

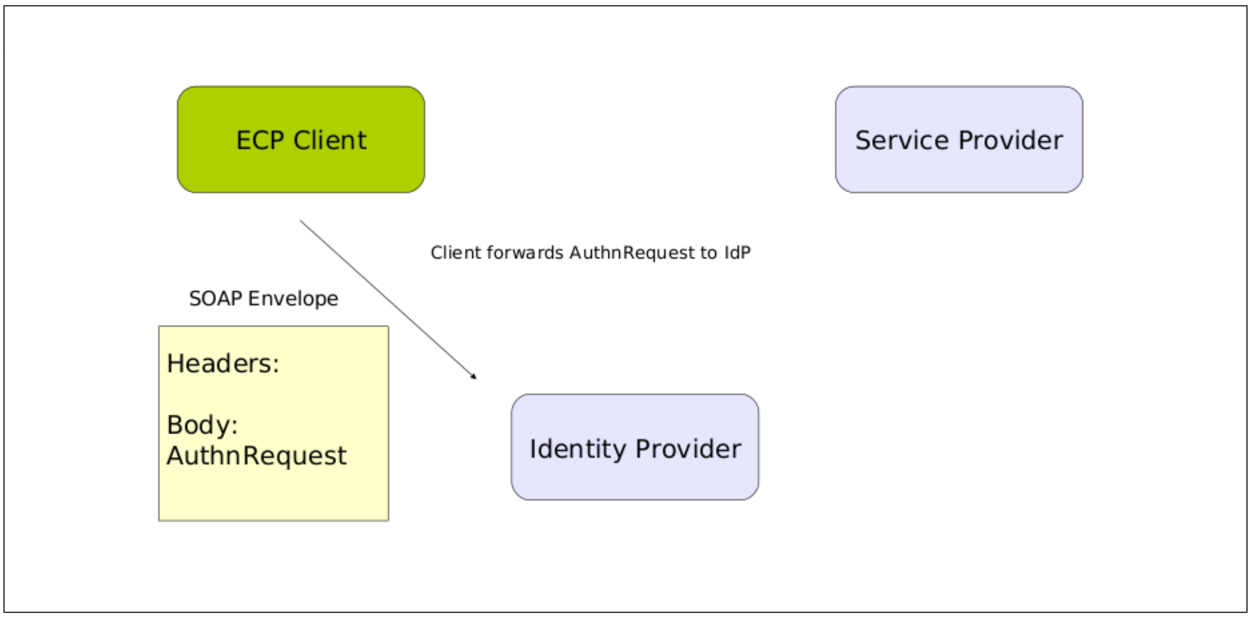

Figure 19: The client forwards the AuthnRequest.

In the next step, the IDP asks the client for authentication information, as illustrated in Fig. 20.

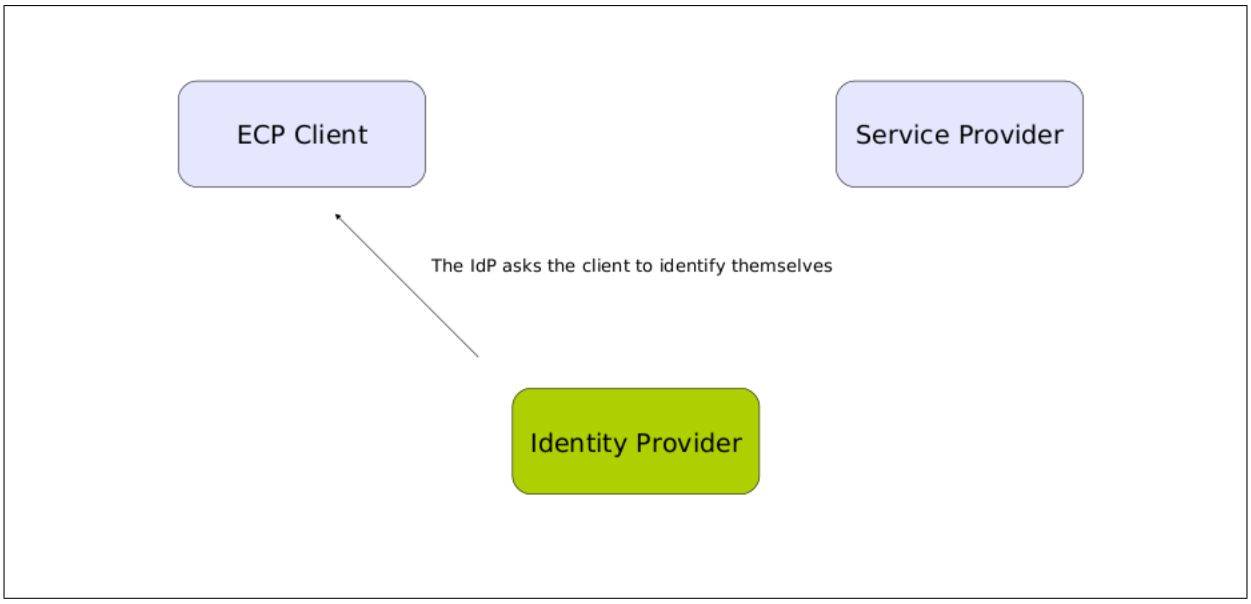

Figure 20: The IDP asks the client for a username and a password.

This client uses HTTP Basic authentication over a secured connection. The username and the password are sent to the IDP as is shown in Fig. 21.

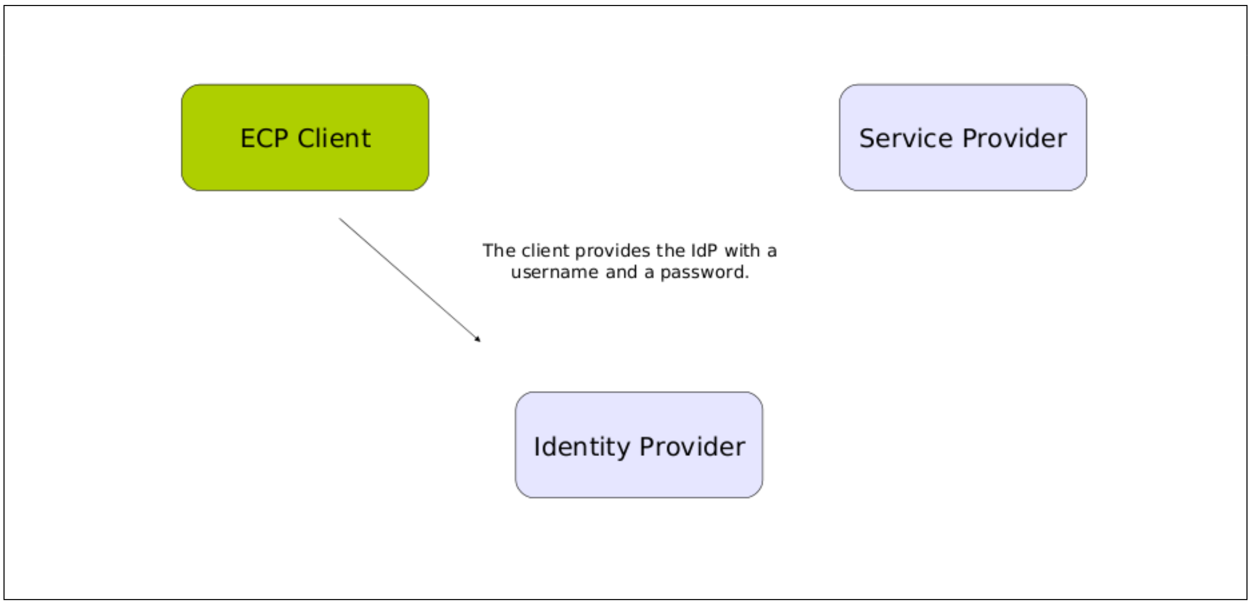

Figure 21: The client sends its credentials.

The IDP verifies that the user trying to log in has a role that is allowed to log in. After this, the IDP verifies the password. The `AuthnRequest` is validated against a schema and its content is checked. For example, the signature on the `AuthnRequest` is checked against the certificate that the IDP has stored in the SP's metadata file. If the process of identification succeeds, the IDP sends a SAML assertion to the client. The assertion is wrapped in a `Response` element, that is packed in a SOAP envelope. See Fig. 22. This envelope also contains an `ECP` header block.

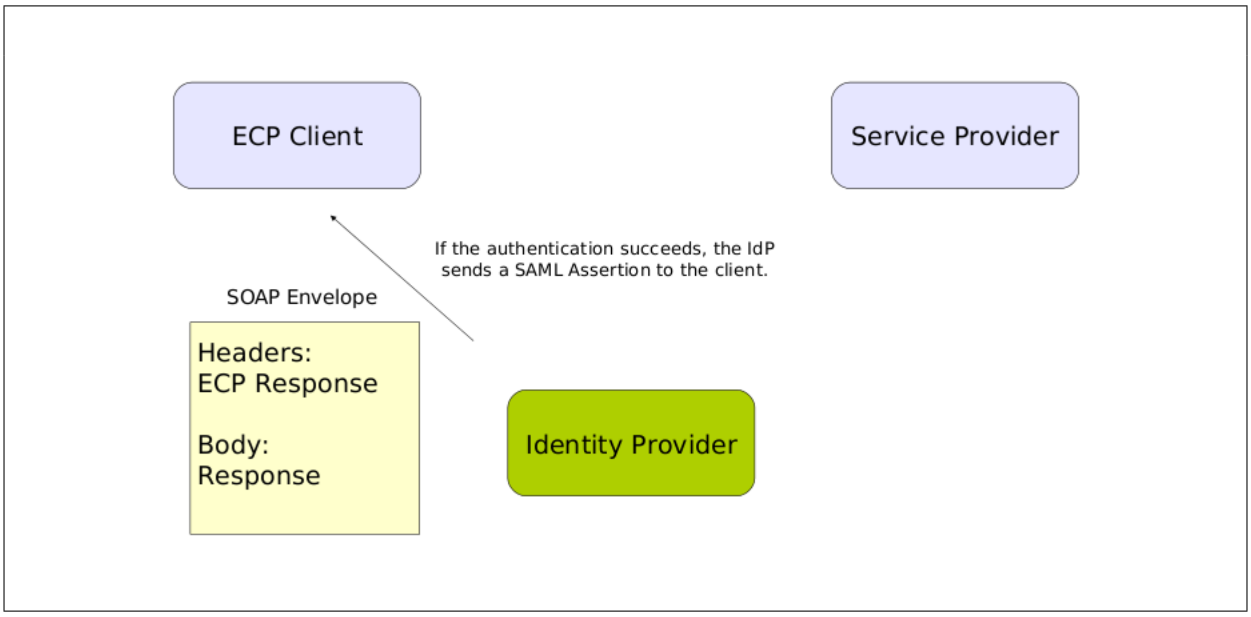

Figure 22: The IDP issues an assertion.

The client removes the ECP header block. At this point, information from the envelope that the SP initially sent is used. As the `AuthnRequest` contained an ID number that the client stored, this ID and an `InResponseTo` attribute from the ECP header block are compared. The client has to know that it forwards an assertion that belongs to an `AuthnRequest` . The client also determines to which address at the SP the assertion will be sent. This address is called the `AssertionConsumerURL` and is the location of where the SP will check and validate the assertion. It can be an address different from the one that initiated the ECP profile actions. This, and security reasons, are the reasons for why the `AssertionConsumerURL` needs to be checked. The `AuthnRequest` also contains an `AssertionConsumerURL`. The client compares these two values and forwards the envelope with the SAML assertion to this address (23) at the SP if the addresses are the same.

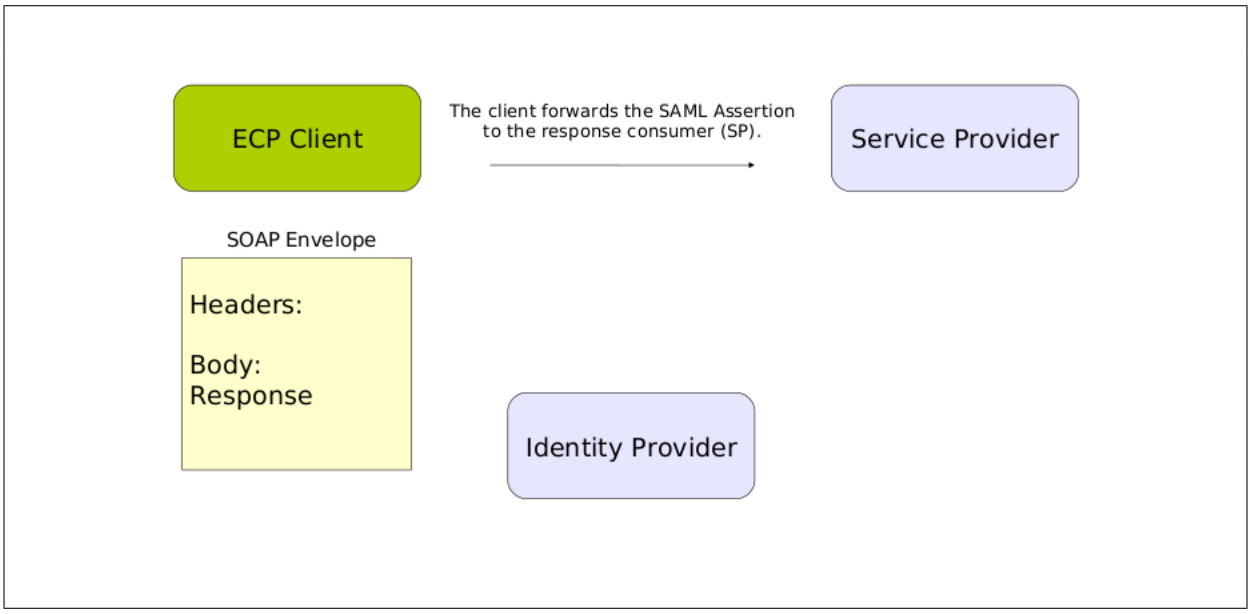

Figure 23: The client forwards the assertion.

The SP receives the SOAP envelope with the SAML assertion, after which the assertion is validated and checked by the SP. The SP can, for example, validate a signature that is possibly placed on the assertion. The validation is done with the IDP's certificate that the SP has stored in its IDP metadata. If the SP decides to trust the authentication information it received through the SAML assertion, and the client has sufficient access rights to the resource it wanted to access, the SP redirects the client to the resource. In this case, the client is only able to receive a resource that consists of plain text. This final step is illustrated in Fig. 24.

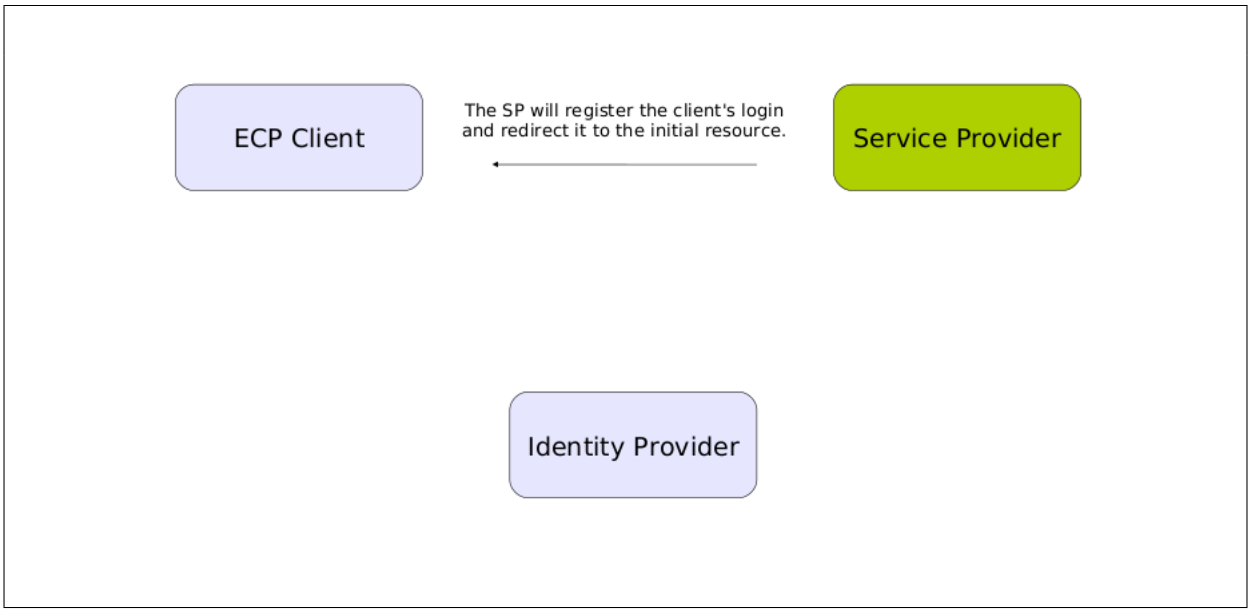
ECP Client
The SP will register the client's login and redirect it to the initial resource.
Service Provider
Identity Provider

Figure 24: The SP returns a resource.

6.3 The development process

The development work started from scratch and took about 2-3 months of time. Useful help was found from the OpenSAML library itself (the code and tests), the Shibboleth wiki and mailing list, several specifications, the Spring Security projects, several conversations from online forums, and last but not least my supervisor(s) and colleagues. Additional software that was of great help were the `netstat` and `tcpdump` tools. The TestNg library (TestNg, 2012) was used for creating unit tests. Self-signed certificates along with their corresponding keys were generated with `openssl`. The exact commands that were used are described in Appendix 3.

Maven was used to keep track of dependencies and the build cycle. The build cycle also takes unit testing into account and can report the test coverage of the code. In this project Maven is mainly used to handle the build process and to manage dependencies on external libraries. Maven works with a file named POM.xml that contains the configuration, plugins and dependencies.

In the short sample POM.xml file that is provided as Appendix 1, some properties are defined, a dependency and a plugin are added. It is worth noticing that one can define global properties for the project using the `<Properties>` tags and adding elements with arbitrarily chosen names. The dependencies always contain the `<groupId>` and `<artifactId>` fields.

For other developers, I would advice them to read the specifications and follow them as far as possible. Several of the problems I faced were related to a) not yet being able to navigate the OpenSAML library, b) not understanding some large part of the specification and c) technical problems that originated from the lack of documentation for OpenSAML or some other library I used. Most confusing was to discover, how much and how fast the code in a library that is under development can change or disappear.

The IDP that was used in the project this thesis relates to, is a Shibboleth IDP v. 2.3.6. (Shibboleth, 2012a), an implementation based on the OpenSAML 2.0 library. The configuration of the IDP was also crucial for the process to work. Appendix 2 contains instructions on how the ECP client option was activated at the IDP.

6.3.1 How the client is used with an STS as SP

When an STS is used as a service provider, the resource that the client retrieves is a certificate. The client can retrieve the SAML assertion as any normal ECP client would, then it generates a pair of keys, and sends the public key, in a SOAP message that is signed by the corresponding private key, to the STS. The SAML assertion is placed in the SOAP message header. The body of the same SOAP envelope contains a `RequestSecurityToken` element in which the request for an X.509 certificate and a public RSA key that the client has generated are inserted. With this information (the assertion and the public key) the STS can issue a certificate that is sent back to the client. The client stores the certificate and the private key for future use. The scenario is illustrated in Fig. 25.

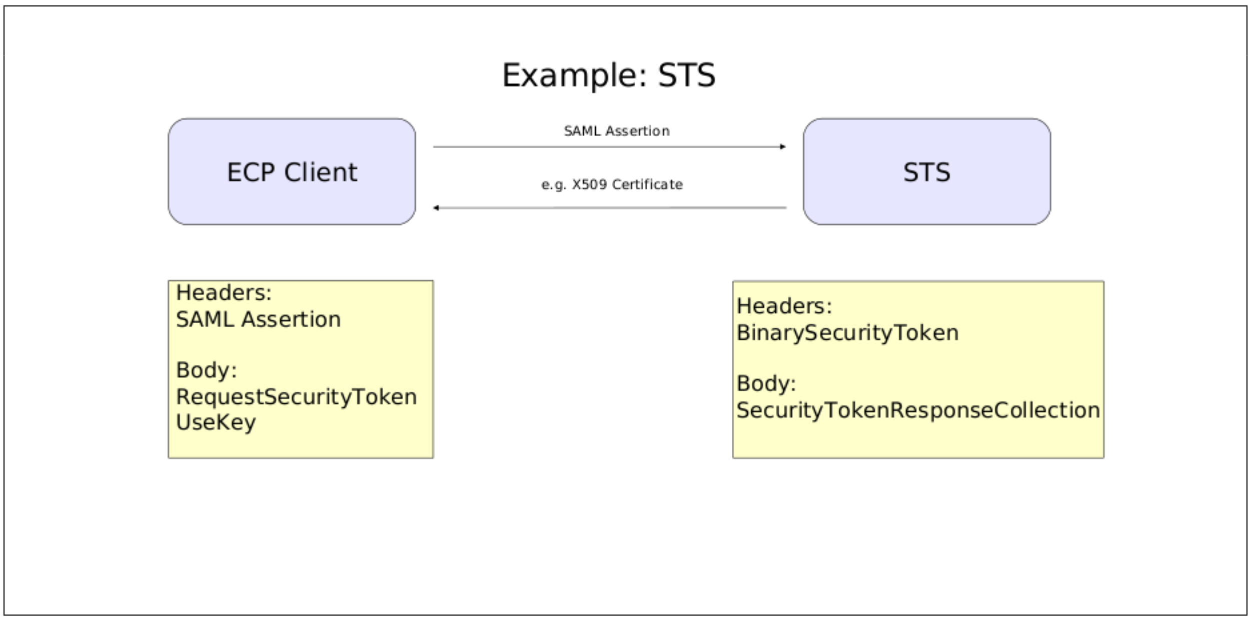

Figure 25: The client exchanges the assertion for an X.509 certificate.

The client can also send its credentials (a username and a password) packed in a `UsernameToken` that is placed in the SOAP envelope header, instead of the SAML assertion. In this case the IDP is not used at all. The client authentication happens at the STS instead.

6.4 Limitations

The session management, i.e. that the client stays logged in, is not fully implemented, since this was not needed for the STS scenario. The client has not been tested with other SP's than the SP that was developed for test use and the STS. The final product is still more of a code library and a prototype. It is not intended for use in an environment where security is an issue. Conducting a security analysis that would discover possible threats against the client was out of the scope for this thesis.

7. Translation of security tokens

As is described in section "3.2.1 Token types", WS-Security (Nadalin et al., 2012a) contains different token types. The types that are relevant for this thesis are the `UsernameToken`, the `X509Token` and the `SAMLToken`. The process of translation consists of turning one type of credentials into another type of credentials. This involves sending credentials to and fro different services, building a network of trustworthy members that use the credentials, and ensuring that every member follows the same standards.

7.1 Why translate a security token?

Among the multitude of applications and web services that are available, different types of credentials are used. Old services may not be interoperable with newer ones, some types of credentials may be difficult to generate for a new user, or the security requirements may be different from one service to another. The end user is the one who has to live with the choices that are made by the web service developers. The authentication method may be cumbersome, or the user might be required to authenticate several times when moving from one service to another. As a solution to the latter problem, single sign-on is used. Easier authentication methods are being researched all the time. For example, a modern approach as described in (Lee et al., 2011) is for a service to generate a 2D bar code (QR-code) on a screen, that the user photographs and sends to the service.

Translation of security tokens is needed to simplify the authentication phase, and to facilitate the transfer of a session from one service to another. For example, a user can authenticate using a username and a password, have this translated into a SAML assertion through an STS, and finally gain access to a SAML based web service. This scenario might be used if the user does not trust the SAML based web service with his or her credentials. The assertion is also valid for a limited period of time compared to the often longer time of validity that a username-password combination has. The central point of trust is the STS that issues the tokens and validates the claims that are sent to it. In this way, a system where credentials do not have to be sent to unknown third parties, can be constructed. This is but one example of why translated tokens are useful.

7.2 How can a token be translated?

The scenario that is visible to the user is, that he or she gives a certain kind of credential that causes the user to log in. For the convenience of the user, this process should last a few seconds. In this thesis the focus lies on using an STS to translate security tokens, namely a username and a password into an X.509 certificate. The message exchange follows the same steps as are described in the SAML 2.0 ECP profile, with the exception that the role of the SP is played by the STS, and the received resource is not a home page but a certificate. Fig. 26 provides a visual description of the process in general. Note that the client may recieve tokens from a third party (i.e. a SAML assertion issued by an IDP), and that the STS has to decide by itself whether the claims can be trusted or not.

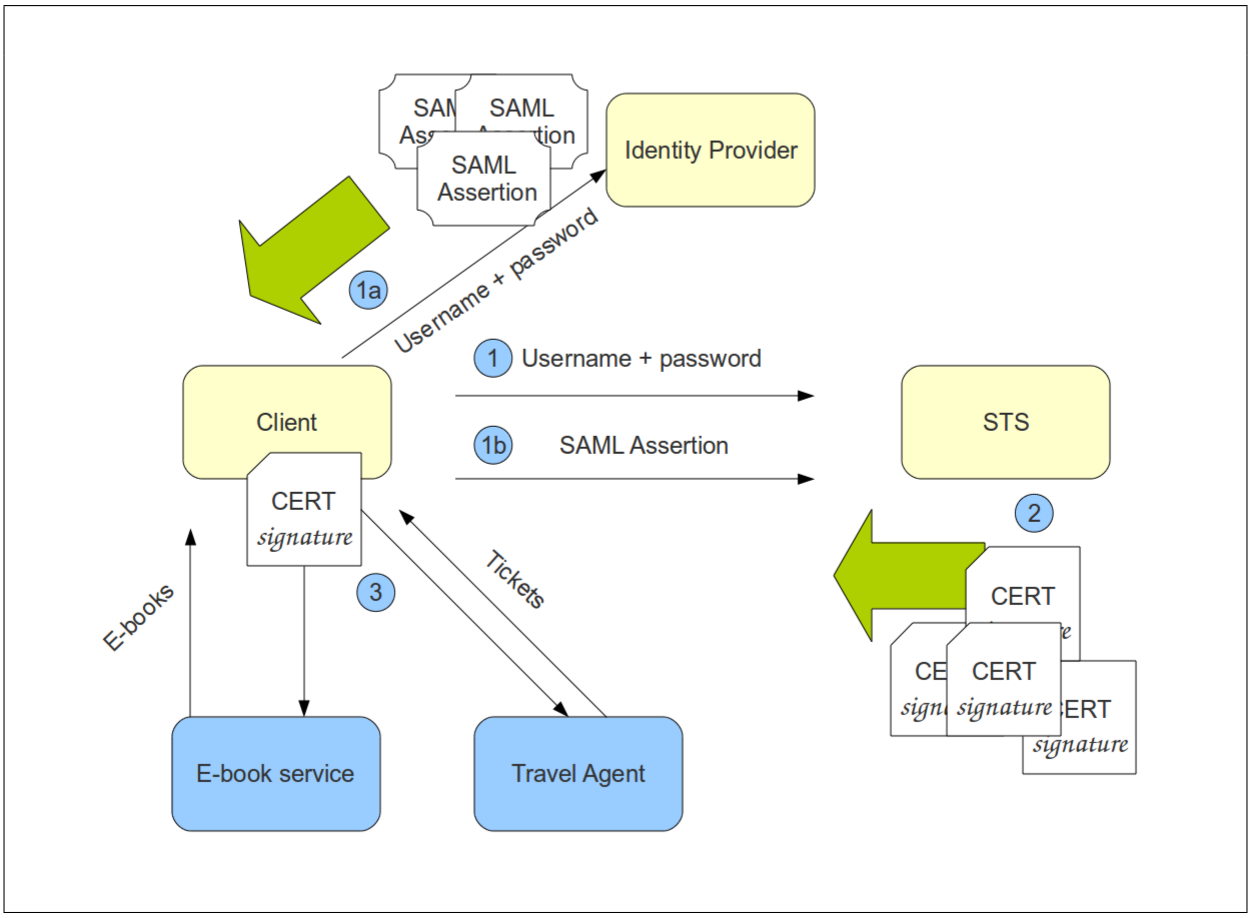

Figure 26: Presentation of the token translation process.

In Fig. 26 a client uses a SAML assertion (1a-1b) or a username-password combination (1) to acquire a certificate (2), that can be used for example (3) when booking a ticket or buying an e-book if these web services require a digital signature as verification of a transaction.

8. Conclusions

The final result of this thesis work is a brief study and explanation of how web services can be secured using modern standards and techniques. Focus lies on the authentication process and the different types of credentials that can be used. The further development of the ECP client software component renders it possible to translate different token types into other types. It can be used for demonstrating the concept of token translation as well as for testing web services. The code library that has been developed, and is to be released as open source code, can be used to develop similar services, to test existing services, and to gain a better understanding of the message exchange that takes place between the components that are involved in the scenario.

The need for creating an easier to use alternative to the direct use of certificates was the basis for this project. If users are faced with the challenge of authenticating themselves with a username and a password, their task is significantly easier than if they would be asked to acquire a certificate. Currently, in any situation where a user that is not familiar with the use of certificates needs one, additional training or some kind of tutorial is required. This takes time off other tasks and burdens the user with unnecessary technical details. Translating simpler types of credentials to a certificate, on behalf of the user, can save much of the time the user would spend acquiring a certificate. Even if a user has mastered the process of requesting a certificate, the delivery of a certificate lasts longer than a sign-on does.

Since the STS supports both username-password login and SAML assertions, this rises the question of which type is more suitable for a certain scenario? The user perceives both ways of interacting with the client software in mostly the same way. Information is entered and the result is a certificate. Using a `UsernameToken` is in a way less complicated, since it involves fewer components, fewer messages, and thus a lesser amount of sensitive information is sent. The client also has the choice of trusting its credentials to an IDP or to an STS. If the user does not wish to share the password with the STS an assertion can be used. The assertion can also be made more anonymous than a username and a password might be.

The extended version of the username token that was proposed in (Peng et al. 2008), as an alternative to the HTTP Basic and HTTP Digest Authentication methods in web services, is also useful. An STS could be used for translating credentials into this form of authentication token, which could replace the above mentioned HTTP authentication methods.

As a result of the translation process, complicated authentication mechanisms can be concealed from the end user. If a web service or other software component requires the use of a certain type of token, for example a certificate, the user could obtain this credential through another credential. The more common username-password combination can be used instead of a normal request for a certificate, and the same goal, gaining access to a service, can be achieved. This also happens faster than it would normally take to fill in a certificate request. Since the whole process is based on long XML messages that need to be parsed, validated and the information verified, the process that goes on in the software components is still slow and prone to errors compared to a simple HTTP Basic authentication for example.

What is gained by token translation is a higher degree of interoperability, since all components follow the same standards, as well as a higher degree of security through the framework that these standards provide. Since several parts of the software that is described herein are released as open source, this means that the implementations can be more cost-effective in use. The code and the standards that are freely available are also subject to review by a large public.

Bibliography

Aarts R., Liberty Reverse HTTP Binding for SOAP Specification v. 1.1, Liberty Alliance Project, 2006, Available: `http://www.projectliberty.org/liberty/content/download/1219/7957/file/liberty-paos-v1.1.pdf` Accessed: 15.10.2012

Alfarez, Abdul-Rahman, A distributed trust model, NSPW '97 Proceedings of the 1997 workshop on New security paradigms, ACM, New York NY USA, 1997.

The Apache Software Foundation, What is Maven? Available: `http://maven.apache.org/what-is-maven.html` Accessed: 12.11.2012

Ardagna, C. A., Damiani, E., De Capitani di Vimercati, S., Samarati, P., XML Security in Security, Privacy, and Trust in Modern Data Management Eds. Petkovic, M., Jonker, W., 2007, Springer Berlin Heidelberg.

Booth D. et al., Web Services Architecture, W3C, 2004, Available: `http://www.w3.org/TR/ws-arch/` Accessed: 26.11.2012

Camenisch, J., Pfitzmann, B., Federated Identity Management in Security, Privacy, and Trust in Modern Data Management Eds. Petkovic, M., Jonker, W., 2007, Springer Berlin Heidelberg.

Cantor S. et al., Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS standard, 2005a. Available: `http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf` Accessed: 15.10.2012

Cantor S. et al., Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS standard, 2005b. Available: `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf` Accessed: 15.10.2012

Cantor S. et al., SAML V2.0 Enhanced Client or Proxy Profile Version 2.0 Working Draft 06 OASIS standard, 2012. Available: `http://www.oasis-open.org/committees/download.php/47214/sstc-saml-ecp-v2.0-wd06.pdf` Accessed: 15.10.2012

Commons CLI, The Apache Software Foundation, 2012. Available: `http://commons.apache.org/cli/index.html` Accessed: 15.12.2012

CSC - IT Center for Science Ltd. Haka, Available: `http://www.csc.fi/english/institutions/haka/federation` Accessed: 25.11.2012

Eastlake D. et al., XML Signature Syntax and Processing, W3C, 2008 `http://www.w3.org/TR/xmldsig-core` Accessed: 12.11.2012 eduroam, eduroam Compliance Statement, 2011. Available: http://www.eduroam.org/downloads/docs/eduroam_Compliance_Statement_v1_0.pdf Accessed: 8.12.2012

Gudgin M. et al, SOAP Version 1.2 Part 1: Messaging Framework W3C, 2007 Available: http://www.w3.org/TR/soap12-part1/ Accessed: 25.11.2012

Hirsch F. et al., Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS standard, 2005. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf Accessed: 15.10.2012

IBM, IBM Identity Mixer. Available: http://www.zurich.ibm.com/idemix/details.html Accessed: 25.11.2012

itslearning, Itslearning, Available: https://www.itslearning.com/welcome.aspx Accessed: 25.11.2012

Jung M. et al. Privacy enabled Web service access control using SAML and XACML for home automation gateways, 6th International Conference on Internet Technology and Secured Transactions, 11-14 December 2011, Abu Dhabi, United Arab Emirates. 2011.

Köhler J et al. bw-IDM: Federated Identity Management for the state of Baden-Würtemberg (Germany). Poster presented at EGI Technical Forum 2012, 2012 Sept 17 - 21, Prague, Czech Republic. Project homepage: http://www.bw-grid.de/bwservices/bwidm/ Accessed: 12.11.2012

Lee, Mun-Kyu; Ku, Bo Kyoung; Kim, Jin Bok, Easy Authentication Using Smart Phones and 2-D Barcodes, 2011 IEEE International Conference on Consumer Electronics (ICCE), 2011.

Lindqvist, C., Exploring the SAML 2.0 ECP-profile, Presentation at EGI Technical Forum 2012, 2012 Sept 17 - 21, Prague, Czech Republic. Available: https://indico.egi.eu/indico/contributionDisplay.py?contribId=262&sessionId=46&confId=1019 Accessed: 22.11.2012

Monzillo R. et al., Web Services Security SAML Token Profile 1.1, OASIS Standard, 2006 Available: http://docs.oasis-open.org/wss-m/wss/v1.1.1/os/wss-SAMLTokenProfile-v1.1.1-os.pdf Accessed: 15.10.2012

Moodle Trust, Moodle, Available: https://moodle.org/ Accessed: 25.11.2012

Nadalin A. et al. WS-Trust 1.4, OASIS Standard, 2009 Available: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.doc Accessed: 15.10.2012

Nadalin A. et al., Web Services Security: SOAP Message Security Version 1.1.1, OASIS Standard, 2012a Available: http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SOAPMessageSecurity-v1.1.1.pdf Accessed: 15.10.2012

Nadalin A. et al., Web Services Security Username Token Profile Version 1.1.1, OASIS Standard, 2012b Available: http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SOAPMessageSecurity-v1.1.1.pdf Accessed: 15.10.2012

Nadalin A. et al., Web Services Security X.509 Certificate Token Profile Version 1.1.1, OASIS Standard, 2012c Available: http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-x509TokenProfile-v1.1.1.pdf Accessed: 15.10.2012

Peng D., Li C., Huo H., An Extended UsernameToken-based Approach for REST-style Web Service Security Authentication, 2009 2nd IEEE International Conference on Computer Science and Information Technology, 2008.

Perlman R., Kaufman C., User-centric PKI, Proceedings of the 7th symposium on Identity and trust on the Internet, IDtrust '08, p. 59-71, 2008, ACM.

Prateek M. et al. Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2.0 OASIS standard, 2005 Available: http://docs.oasis-open.org/security/saml/v2.0/saml-conformance-2.0-os.pdf Accessed: 3.12.2012

Reagle J. XML Encryption Requirements W3C, http://www.w3.org/TR/xml-encryption-req Accessed: 25.11.2012

Shibboleth consortium a, Shibboleth IDP 2.3.6, Available: http://shibboleth.net/downloads/identity-provider/2.3.6/ Accessed: 22.11.2012

Shibboleth consortium b, Shibboleth home page, Available: http://shibboleth.net/ Accessed: 25.11.2012

Shibboleth consortium c, OpenSAML home page, Available: https://wiki.shibboleth.net/confluence/display/OpenSAML/Home Accessed: 4.12.2012

Shibboleth consortium d, OpenSAML 3.0 home page, Available: https://wiki.shibboleth.net/confluence/display/OS30/Home Accessed: 4.12.2012

Shibboleth consortium e, ECP, Available: https://wiki.shibboleth.net/confluence/display/SHIB2/ECP Accessed: 4.12.2012

Smith, R., Moonshot, Online seminar presented 14.11.2012. Available: http://webmedia.company.ja.net/content/presentations/shared/moonshotob/ Accessed: 22.11.2012

Stallings W., Cryptography and Network Security: Principles and Practice, 2011, 5th edition, Pearson Prentice Hall, Upper Saddle River, NJ, USA.

TestNG, TestNg home page. Available: `http://testng.org/` Accessed: 25.11.2012

Eds. van Tilborg, H., Jajodia S., Encyclopedia of Cryptography and Security, 2nd ed. 2011, Springer Berlin Heidelberg.

Appendices

Appendix 1 - A sample POM.xml file

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>Service</groupId>
    <artifactId>Web Service Servlet</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <!-- Properties -->
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <openSamlVersion>3.0-SNAPSHOT</openSamlVersion>
    </properties>

    <!-- Dependencies -->
    <dependencies>
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-saml-api</artifactId>
            <version>${openSamlVersion}</version>
        </dependency>
    </dependencies>

    <!-- Plugins -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <finalName>my-jar-file-name</finalName>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Appendix 2 - Activating the ECP profile for a Shibboleth IDP v.2.3.6

This is a brief review of how a Shibboleth IDP v 2.3.6 can be configured to support the ECP profile. In some of the files it is enough to comment and/or uncomment a setting, but some files need manually added information. For example, the IDP needs to maintain a list of the SP's that are trusted. This is achieved by adding metadata, information about the SP into a configuration file, i.e. an ID and a certificate. With the certificate, the IDP is able to validate signed messages from the SP and encrypt outgoing messages. Users that are allowed to log on to the IDP are also managed, i. e. from a database. The IDP itself runs in a servlet container, Tomcat 6 was used in this project. In this case, enabling the possibility for the client to log on to the IDP also involves editing Tomcat 6 files. The example here shows how HTTP Basic login can be used as an authentication method.

First, roles and users need to be defined in the `tomcat6/conf/tomcat-users.xml` file. An example is shown in Fig. 2.1.

```
<role rolename ="ECPLogin-role"/>
<user username ="jimbo" password="examplePassword" roles="ECPLogin-role"/>
```

Figure 2.1: Defining a role and a user in tomcat-users.xml .

The next step is the `shibboleth-idp/war/WEB-INF/web.xml` file. Here, the HTTP-Basic part should be uncommented, and the ECP profile should be configured to use HTTP Basic. The example XML is shown in Fig. 2.2. Make sure the `<role>` tag has the same content as was written in the `tomcat-users.xml` file.

In the `shibboleth-idp/conf/attribute-resolver.xml` file, the attributes that will be inserted into the issued assertion can be edited.

```
<!-- Uncomment to use container managed authentication -->
 <security-constraint>
     <display-name>Shibboleth IdP</display-name>
     <web-resource-collection>
         <web-resource-name>ECP</web-resource-name>
         <url-pattern>/profile/SAML2/SOAP/ECP</url-pattern>
         <http-method>GET</http-method>
         <http-method>POST</http-method>
     </web-resource-collection>
     <auth-constraint>
        <role-name>ECPLogin-role</role-name>
     </auth-constraint>
     <user-data-constraint>
           <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
   </security-constraint>

  <!-- Uncomment if you want BASIC auth managed by the container -->
  <login-config>
     <auth-method>BASIC</auth-method>
     <realm-name>IdP Password Authentication</realm-name>
  </login-config>
```

Figure 2.2: Enabling HTTP Basic authentication for the ECP profile on an IDP.

In `/shibboleth-idp/conf/relying-party.xml` the indication that the SP supports the ECP profile and can be relied upon is added, by adding information about the SP as a `RelyingParty` with the `entityId` of the SP and some IDP configuration information as is shown in Fig. 2.3.

```
<!-- A Service Provider that understands the ECP profile  -->
     <rp:RelyingParty id="the-SP's-entity-id" provider="the-IdP's-ID"
      defaultSigningCredentialRef="IdPCredential">

      <rp:ProfileConfiguration xsi:type="saml:SAML2ECPProfile"
                includeAttributeStatement="true"
                assertionLifetime="PT5M" assertionProxyCount="0"
                signResponses="never" signAssertions="always"
                encryptAssertions="conditional" encryptNameIds="never"/>
      </rp:RelyingParty>
```

Figure 2.3: Configuring the ECP profile and indicating that the SP supports it.

In the same `/shibboleth-idp/conf/relying-party.xml` file, the source from which the SP's metadata will be read is configured, an example of this is provided in Fig. 2.4.

```
        <!-- Load new sp-metadata. -->
        <metadata:MetadataProvider
        id="serviceProvidermetadataId"
        xsi:type="metadata:FilesystemMetadataProvider"
        metadataFile="/shibboleth-idp/metadata/my-sp-metadata-file.xml"/>
```

Figure 2.4: Adding the location of the SP's metadata.

The contents of the metadata file, whose location was specified in Fig. 2.4 contains the `entityId` of the SP, indicates that the SP supports the ECP profile and also contains the SP's certificate. The `entityId` holds the same value as the id attribute in the `RelyingParty` element in Fig. 2.3. The `AssertionConsumerService` element's `location` attribute, is the final URL (or URI) at the SP, to which the issued SAML Assertion will be sent.

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
  entityID="the-SP's-entity-id">
   <md:SPSSODescriptor
      protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
      <md:KeyDescriptor>
         <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:X509Data>
               <ds:X509Certificate>
                    ... X.509 Certificate  ...
               </ds:X509Certificate>
            </ds:X509Data>
         </ds:KeyInfo>
      </md:KeyDescriptor>
      <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:PAOS"
      Location="the-assertion-consumer-url-of-the-SP" index="0" isDefault="true"/>
   </md:SPSSODescriptor>
</md:EntityDescriptor>
```

Figure 2.5: An example of a file that contains metadata of an SP.

Appendix 3 - Useful openssl commands

The following list contains commands for the openssl tool that can be used to create self-signed certificates. In order to use the generated certificate and the key in a Java application, it is necessary to convert the format from PEM format to DER format. If the sequence listed below is followed, the user will end up with a key file (key.der) and a certificate (cert.der) that can be loaded into a Java application.

- Create a 2048 bit RSA key:

  ```
  openssl genrsa -des3 -out key.pem 2048
  ```

- Create a certificate signing request:

  ```
  openssl req -new -key key.pem -out client.csr
  ```

- Create a self-signed certificate, valid for 100 days:

  ```
  openssl x509 -req -days 100 -in client.csr -signkey  key.pem
  -out cert.pem
  ```

- Convert the certificate to DER format:

  ```
  openssl x509 -inform pem -in cert.pem -out cert.der -outform der
  ```

- Convert the key to DER format:

  ```
  openssl rsa -in key.pem -inform pem -out key.der -outform der
  ```

Appendix 4 - UML diagrams (class- and sequence-) related to the ECP client

Fig. 4.1 shows a more detailled view of the classes and the functionality that were added to the client component as part of this thesis work.

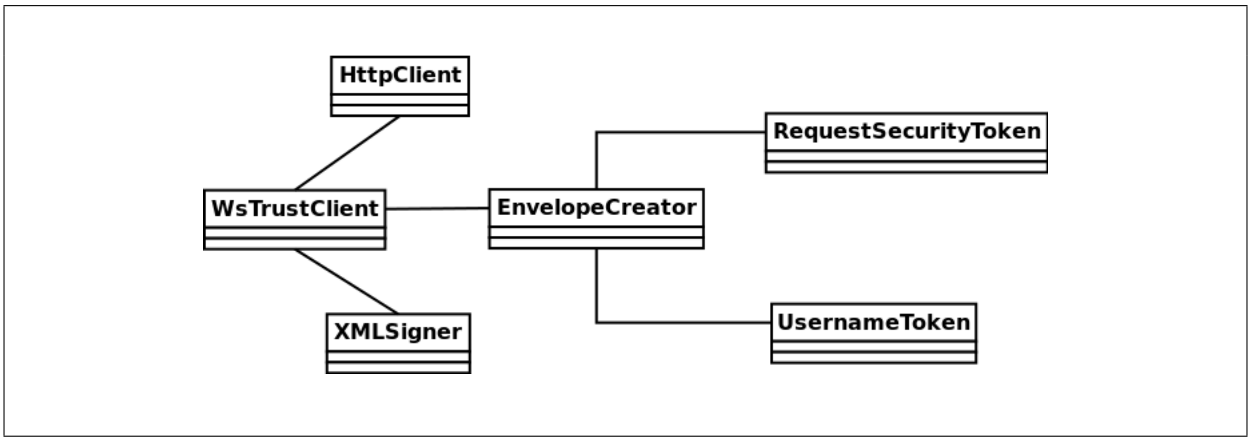

Figure 4.1: Detailed view of the added functionality.

The following sequence diagrams, Fig. 4.2 and Fig. 4.3 depict the functionality of the client. In Fig. 4.2 the basic ECP client and the added functionality are shown. The red line indicates when the client has received a SAML assertion. The WSTrustClient part of the diagram describes how the client requests a certificate and handles the response. Fig. 4.3 aims to explain how the client acquires an assertion.

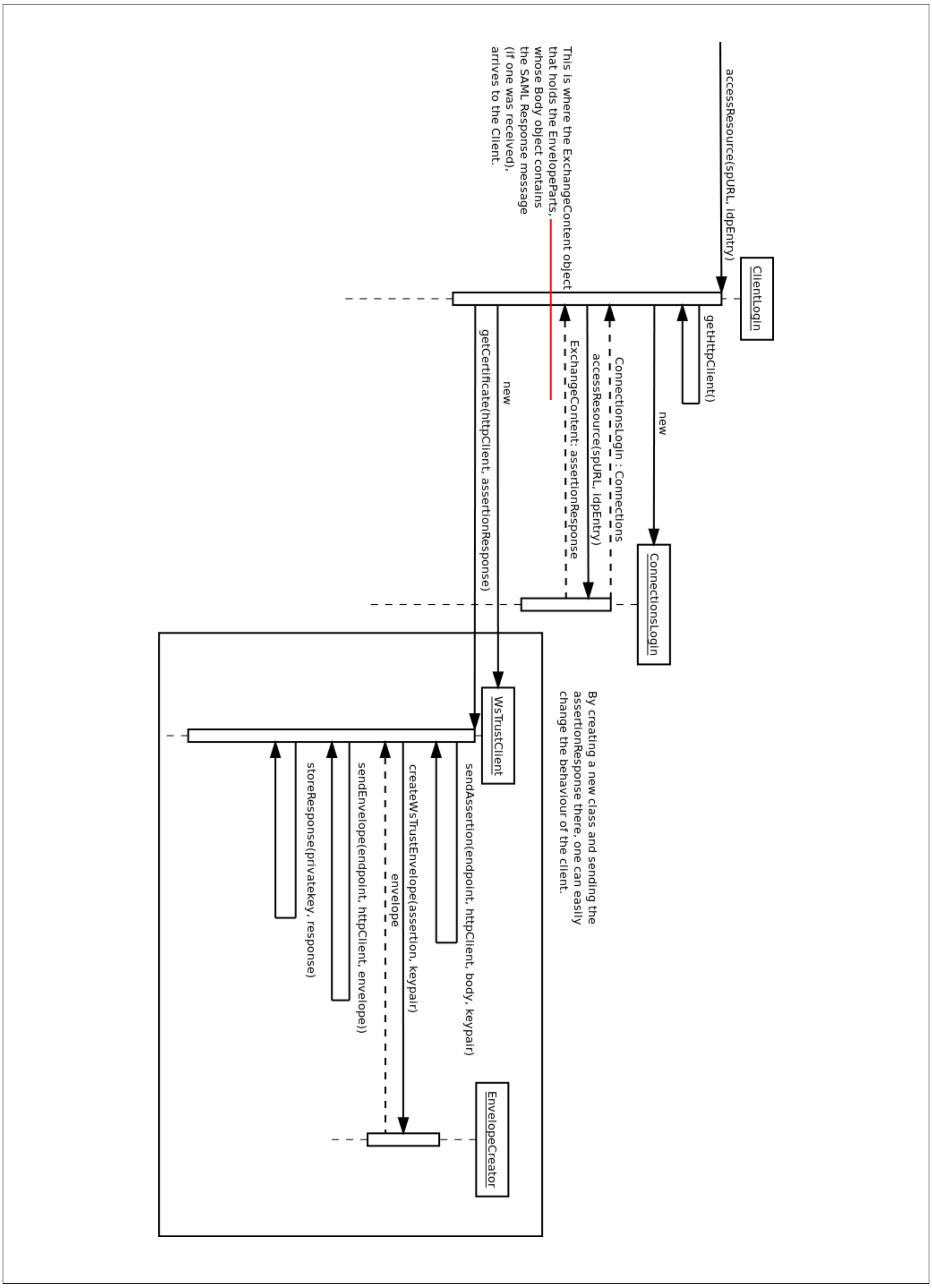
ClientLogin
accessResource(spURL, idpEntry)
getHttpClient()
new
ConnectionsLogin
ConnectionsLogin : Connections
accessResource(spURL, idpEntry)
ExchangeContent: assertionResponse
This is where the ExchangeContent object that holds the EnvelopeParts, whose Body object contains the SAML Response message (if one was received), arrives to the Client.
new
getCertificate(httpClient, assertionResponse)
WsTrustClient
By creating a new class and sending the assertionResponse there, one can easily change the behaviour of the client.
sendAssertion(endpoint, httpClient, body, keypair)
createWsTrustEnvelope(assertion, keypair)
envelope
sendEnvelope(endpoint, httpClient, envelope)
storeResponse(privatekey, response)
EnvelopeCreator

Figure 4.2: Sequence diagram of the certificate acquisition process using a SAML assertion.

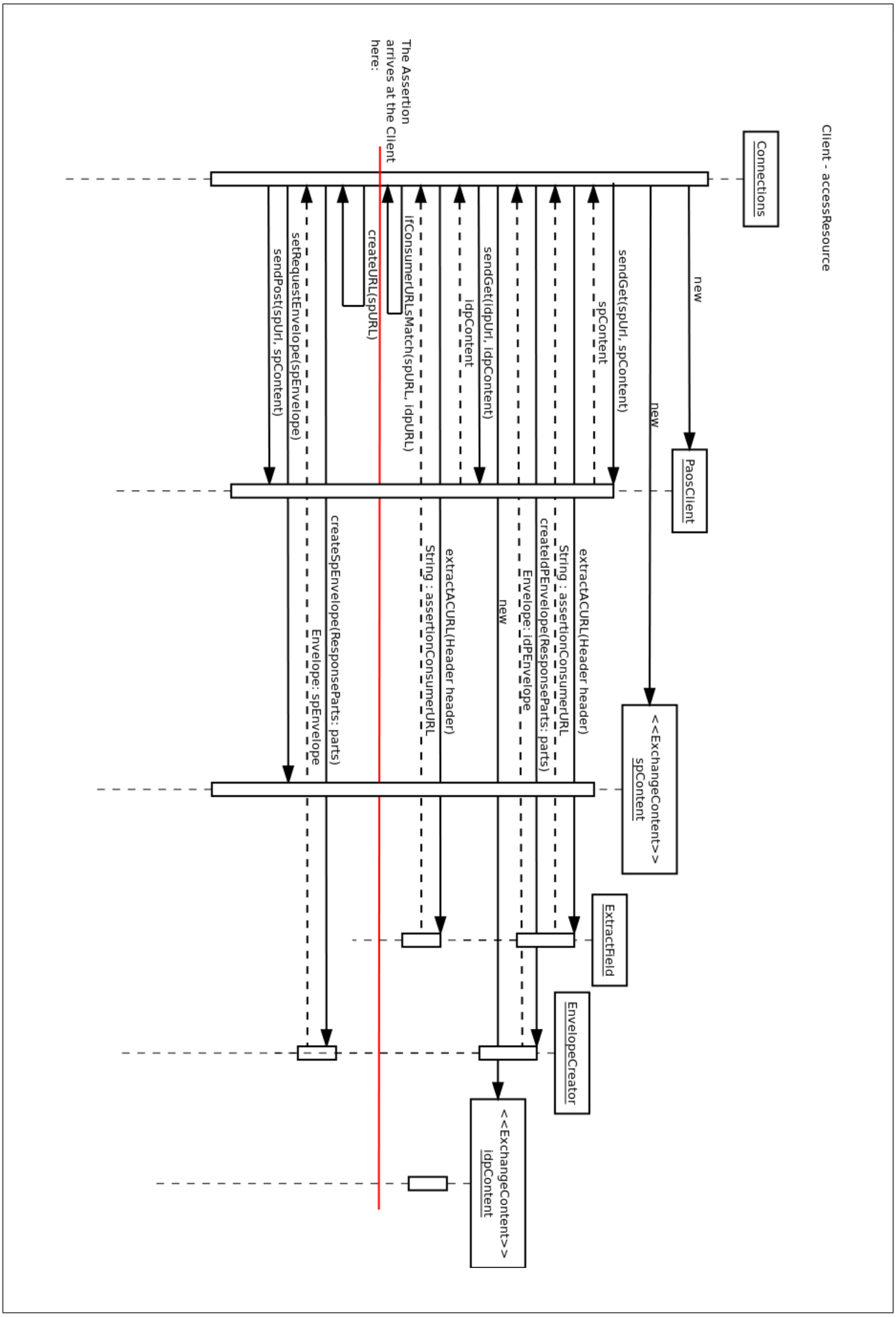
Client - accessResource
Connections
PaosClient
<<ExchangeContent>> spContent
ExtractField
EnvelopeCreator
<<ExchangeContent>> idpContent
new
new
sendGet(spUrl, spContent)
spContent
sendGet(idpUrl, idpContent)
idpContent
ifConsumerURLsMatch(spURL, idpURL)
createURL(spURL)
setRequestEnvelope(spEnvelope)
sendPost(spUrl, spContent)
extractACURL(Header header)
String : assertionConsumerURL
createIdPEnvelope(ResponseParts: parts)
Envelope: idPEnvelope
new
extractACURL(Header header)
String : assertionConsumerURL
createSpEnvelope(ResponseParts: parts)
Envelope: spEnvelope
The Assertion arrives at the Client here:

Figure 4.3: Sequence diagram of the assertion acquisition process.