

**The Open University of Israel**  
**Department of Mathematics and Computer Science**

# **Web Services Security**

## **- Focus on SAML and XACML**

Final Paper submitted as partial fulfillment of the requirements  
towards an M.Sc. degree in Computer Science

The Open University of Israel  
Computer Science Division

By  
Andrey Kogan

Prepared under the supervision of Prof. Ehud Gudes

February 2015

## Abstract

This paper focuses on web service authorization using SAML (Security Assertion Markup Language) and XACML (eXtensible Access Control Markup Language). The paper defines requirements for authorization of web services and investigates existing authorization solutions concerning these requirements. It shows the importance of web service authorization.

The paper overviews web services, concepts and technologies behind them, and web services security. It shows the issues of web service security: confidentiality, authentication, and network security, and provides solutions for authentication and authorization issues. It discusses the SAML specification, which defines the syntax and semantics of statements for security message exchange and provides high-level SAML use cases. It introduces XACML, which describes how to evaluate authorization request according to policy rules. It shows the XACML architecture and examples of XACML rules.

First, the paper reviews articles that describe web services security. The articles present simple and complex solutions for web service authorization. Some papers introduce implementation of web service authorization networks and describe experiences with the implementation. These works provide security for cooperative business processes using web services in an open environment. The main contribution of these papers is in introducing the basic concepts for securing web services and proposing system architectures for web services. The authors show the importance of authorization policies in web services composition. The referenced papers describe concepts and design of the proposed frameworks and overviews selected implementation aspects such as authorization data access.

Second, the paper presents an application using the Microsoft framework called WIF. The paper provides a high level overview about WIF (Windows Identity Foundation), a framework for implementing claim-based identity in applications. It shows a basic scenario with web services and a basic scenario with a web browser. The paper presents a solution that uses SAML tokens with ASP.NET Web API technology. The solution includes three applications: Client Web Application that accesses the web service, STS (Security Token Service) that issues SAML tokens, and SP (Service Provider), the web service that requires authentication of a client prior to authorizing the client. The Web client sends the request to the STS. The STS determines that the client's credentials are valid, and sends the SAML token to the web client. The web client caches the SAML in a cookie and sends a request message to the service provider; the request message includes the SAML token. Implementation details and demonstration of the results are presented.

## Table of Contents

1.	Introduction to Web Services Security .....	5
1.1.	Structure of this work .....	7
2.	Review of Web Services and Web Services Security .....	8
2.1.	Web Services Security Requirements .....	9
2.2.	Authentication .....	10
2.3.	Authorization .....	11
2.4.	Web Service Example 1 .....	12
2.5.	Web Service Example 2 .....	14
2.5.1.	Authentication Requirements.....	16
2.5.2.	Data protection .....	16
2.5.3.	Authorization .....	17
3.	SAML .....	18
3.1.	OASIS .....	18
3.2.	What is SAML? .....	18
3.3.	The Scope of SAML.....	20
3.4.	SAML Assertions.....	22
3.5.	Authentication statement.....	24
3.6.	Attribute statement .....	24
3.7.	Authorization statement.....	25
3.8.	SAML Request/Response .....	26
3.9.	SAML Request .....	26
3.10.	SAML Response .....	27
3.11.	Bindings.....	29
3.12.	SOAP Binding.....	29
3.13.	Profiles .....	29
3.14.	SAML Artifact .....	30
3.15.	SAML POST .....	31
4.	High-Level SAML Use Cases .....	32
4.1.	Web Single Sign-On Use Case .....	32
4.2.	Identity Federation Use Case .....	35
5.	XACML.....	37

5.1.	Introduction to XACML .....	37
5.2.	XACML Architecture Diagram .....	38
5.3.	Policy in XACML.....	40
5.4.	XACML Rule Examples.....	41
6.	Windows Identity Foundation .....	44
7.	Papers Survey.....	47
7.1.	Web Service Authorization Framework by Thomas Ziebermayr and Stefan Probst [1] .....	47
7.2.	Trust Management for Web Services by Weiliang Zhao and Vijay Varadharajan [2] .....	53
7.3.	Definition and Implementation of a SAML-XACML Profile for Authorization Interoperability across Grid Middleware in OSG and EGEE by Gabriele Garzoglio, Ian Alderman, Mine Altunay, and Rachana Ananthakrishnan [3].....	57
7.4.	Access Control Policies for Web Services in Medical Aid System by Li-Qun Kuang, Yuan Zhang, and Xie Han [4].....	59
7.5.	Short Papers Survey .....	61
7.5.1.	Specification and Verification of Authorization Policies for Web Services Composition by Mohsen Rouached and Claude Godart [5].....	61
7.5.2.	Towards Secure Execution Orders for Composite Web Services by Joachim Bickup, Barbara Carminati, Elena Ferrari, Frank Muller, and Sandra Wortmann [6].....	61
7.5.3.	Extending XACML Authorization Model to Support Policy Obligations Handling in Distributed Application by Yuri Demchenko and Cees de Laat [7] .....	62
7.5.4.	Authorization and Privacy for Semantic Web Services by Lalana Kagal, Tim Finin, Massimo Paolucci, Naveen Srinivasan, Katia Sycara, and Grit Denker [8] .....	63
8.	The Application .....	64
9.	Conclusion.....	71
10.	Summary .....	72
11.	References .....	73
	Appendix A. Definitions .....	75
	Appendix B. SAML Syntax Samples.....	80
	Appendix C. SAML Request / Response Samples.....	83
	Appendix D. XACML Syntax Samples .....	85
	Appendix E. Class Diagram of the Applications .....	91
	Appendix F. Encrypted SAML Data of the Applications .....	92

## **1. Introduction to Web Services Security**

Web services are open standards based web applications that interact with other web applications to exchange data. Web services help applications communicate with each other. There are no requirements as to how services are provided. Web services are stateless and follow a request/response model of interaction. Web services standards provide a way to locate interfaces and exchange data in an understandable way. Web services are the basis for service-oriented architectures.

Web services expand the Web from a user front end to an application service. With web service, the originator of web connection is no longer just the consumer or supplier of information. He can participate in a distributed application environment and make remote procedure calls to request services.

Why should we use web services? There are a few reasons to use it, listed below.

In the past, a purchasing agent sent a purchase order to a supplier via email. Now a company's purchasing application communicates via the Internet directly with the supplier's warehouse application. It can check price and availability, and submit purchasing order automatically, without any human intervention.

The benefits of web services are not limited to interaction between companies. Business units within the same enterprise can communicate with each other, using web services.

We can define a web service as XML-based messaging interface to computing resources that is accessible via Internet standard protocols. Web service is a managed code that can be invoked remotely using HTTP, so it can be activated using HTTP requests. We can expose functionality of existing code over the network.

We can also connect different applications. For example, our .NET application can talk to java web service and vice versa. Thus, web services make application technology independent.

Web services use standard protocols for communications. In most cases, web service uses HTTP (Hypertext Transfer Protocol), but it can also use FTP (File Transfer Protocol) or SMTP (Simple Mail Transfer Protocol). Web services use the following technologies: Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI).

For Intranet applications, SOAP messages may be received directly by application servers. For extranet applications, SOAP messages may be received by Web servers or integration servers that pass messages on to the appropriate applications (because of security problems).

Web Services Protocol Stack consists of 4 layers: a) Service Transport that is responsible for transporting messages between applications. It includes HTTP, SMTP, FTP, and other protocols. b) XML Messaging that is responsible for encoding messages in common XML format (XML-RPC – Remote Procedure Calls and SOAP). c) Service Description that is responsible for describing public interface to specific web service via WSDL. d) Service Discovery that is responsible for centralizing services into common registry and providing easy Publish functionality via UDDI.

Given the potential risks, security is critical for web services. Security risks can be described by the following terms: assets, threats, vulnerability, or attacks. An asset is something related to the application that is worth protecting. Sensitive data, intellectual property, and access to critical operations are all assets. For example, user credit card numbers are an asset worth protecting in applications. A threat is any potential occurrence that could harm an asset. Vulnerability is a weakness that makes a threat possible. This may be because of poor design, configuration mistakes, or insecure coding technology. Weak input validation is an example for an application layer vulnerability, which can result in input attack. An attack is an action that exploits vulnerability or enacts a threat. Examples of attacks include sending malicious input to an application, or flooding a network in an attempt to deny web service.

## **1.1. Structure of this work**

Chapter 2 overviews web services, concepts and technologies behind them, and web services security.

Chapter 3 discusses SAML, specification that defines syntax and semantics of statements supporting security message exchange.

Chapter 4 presents high-level SAML use cases.

Chapter 5 presents XACML, specification for expressing access control policies over internet.

Chapter 6 presents Microsoft Identity Foundation.

Chapter 7 reviews articles that describe web services security.

Chapter 8 presents application.

Chapter 9 contains conclusions of this paper.

Chapter 10 presents paper summary.

Chapter 11 contains references.

Appendix A overviews description of underlying technologies.

Appendix B contains samples of SAML syntax.

Appendix C contains samples of SAML Request / Response.

Appendix D contains samples of XACML syntax.

Appendix E contains class diagram of applications.

Appendix F contains encrypted SAML data of applications.

## 2. Review of Web Services and Web Services Security

The following issues are important for web services security: confidentiality, authentication, and network security [14]. When a client sends request to server, we need to ensure that communication will be confidential. When a client connects to web service, we need to identify the user and check if this user is authorized to use our service.

Network security includes the policies adopted by a network administrator to prevent and monitor unauthorized access, misuse, or modification of a computer network and network resources. Network security includes the authorization of access to network resources, which is controlled by the network administrator. Users can have an ID and password or different authenticating information that allows them access to programs within their authority. The most common and simple way of protecting a network resource is by assigning a unique ID and a corresponding password.

The solutions of confidentiality issue are the following: a) XML-RPC and SOAP run on top of HTTP; b) HTTP supports SSL (Secure Sockets Layer); c) Encrypt communication with SSL.

The solutions of authentication issue are the following:

1. HTTP has built-in support for Basic and Digest authentication, so web services can be protected in the same manner as HTML documents.
2. Digital Signature of SOAP Security Extensions: public key to digitally sign SOAP messages.
3. SAML (Security Assertion Markup Language) of OASIS (Organization for the Advancement of Structured Information Standards).

Network security has no easy solution. Networks are subject to attacks from malicious sources. There are two kinds of attacks: "Passive" when a network intruder intercepts data traveling through the network, and "Active" in which an intruder initiates commands to disrupt the network's normal operation. If we truly want to filter out SOAP or XML-RPC messages, one possibility is to filter out all HTTP POST requests that set their content type to text/xml. It is a requirement of both specifications. Another alternative is to filter the SOAPAction HTTP header attribute. The SOAPAction header is a server-specific URI used to indicate the intent of the request. This makes it possible to quickly determine the nature of the SOAP request, without actually examining the SOAP message payload. In practice, the header is frequently used by firewalls as a mechanism for blocking out SOAP requests or for quickly dispatching SOAP messages to specific SOAP servers. Firewall vendors are also currently developing tools explicitly designed to filter web service traffic [14].



We see that security is a very important part of any web service implementation. Here are the examples of business models that need security:

1. **E-commerce sites on the Internet.** Sites depend on credit card authorization services provided by outside company.
2. **Cross-selling and customer relationship management.** Customer information can be shared across many lines of business within enterprise company.
3. **Supply chain management.** Requires communication among all suppliers in the manufacturing chain to be sure that supply of parts is enough for demand.
4. **Bandwidth on demand.** Allows customers to make dynamic requests in order to increase quality of telecommunications service and get immediate results.



Figure 2.1 Basic requirements for a web service [10]

Figure 2.1 [10] demonstrates basic requirements of a web service. It shows online internet store ePortal. The visitors access site via browser using HTML. The store sells products to customers using shopping carts. Registered members have access to some deals that are not accessible to unregistered users (visitors). Visitors can browse ePortal. Staff can administrate the portal. All services provided by the portal are implemented by eBusiness. eBusiness stores products data and processes portal orders.

## 2.1. Web Services Security Requirements

Web services security requirements are the following [10]:

- **Authentication.** Verifies that users are who they claim to be (passwords, tokens, public key certificates, secret keys).
- **Authorization.** Grants permission to access resources, providing basis for access control.
- **Encryption.** Provides cryptographic algorithms and protocols to protect data and messages from disclosure and modification.
- **Accountability.** Ensures that users are accountable for their action.
- **Security administration.** Defines security policy for user profiles, authentication, authorization, etc.

Each web service transaction must be traceable from its origin to its fulfillment, maintaining consistent security across processing tiers and domains.

Web services security must be flexible enough to identify all participants in a transaction.

Additional benefit of seamless integration of security with web services is to provide single sign-on (SSO), even across organizations using different security technologies.

## **2.2. Authentication**

Authentication provides service that determines that requester is exactly who he claims to be. There are two categories of authentication – password-based and challenge-response based.

Password-based authentication has a few limitations. Challenge-response authentication provides higher security level, but it is more complex. Password-based authentication is more popular than challenge-response based authentication.

There are few different authentication systems:

1. Operating system-based authentication.

Microsoft IIS (Internet Information Server) uses this authentication system. It uses facilities of Windows operating system. IIS has the following methods for authentication:

a) HTTP basic authentication (using user name and password); b) Challenge-response based HTTP digest authentication; c) SSL authentication; 4) Kerberos authentication.

2. Web server-based authentication.

Web server can handle authentication requirements for HTTP: basic password-based authentication and HTTP digest authentication.

3. Token-based authentication.

User must have token card and know PIN. The token is stronger than password. Authentication server needs to verify the value of token. Response will be sent from the Authentication server to the workstation during the authentication protocol.

4. Web single sign-on.

We can save authentication information in web browser cache. In this case, user doesn't have to enter username and password multiple times. If there are several web servers in the cluster, there is a problem when we need the same authentication information for the different servers. Web SSO systems (for example, Microsoft Passport) can solve this problem. This system maintains session between HTTP requests to the same server in cluster. So user can log in only once and access any server in a cluster without logging in again. After successful authentication, the server creates cookie with username, IP address, browser, and expiration time. Most Web SSO systems have authentication mechanisms as password, RSA SecurID, Windows

NT domain login and public key certificate, and provide authorization services. Web SSO systems can be integrated with web servers.

## **2.3. Authorization**

Authorization means specifying and granting permissions to access web services resources. It provides basis for access control. Authentication supports authorization. Server makes decision whether to grant access to requested web service resource.

There are the following mechanisms for authorization in web services:

- Operating systems – it's possible to use operating system to authorize action and enforce authorization, if user is authorized by operating system.
- Web servers – web servers are designed to protect web pages.
- Application servers – application servers perform authorization based on authentication they perform or identity passed to them by front-end component.
- Application – applications can enforce authorization.

Authorization policies allow access to resources based on collections of authenticated users (groups, roles, privileges). The policies can restrict access to any collection of resources, for example, hosts, files, web pages, or database. Authorization policies may restrict access based on global context (such as day of week) or data values (such as trading of shares for minimal sum of 1000 dollars). Web servers support access control for directory/path level. Entire page may be protected, but not a part of it.

Security Assertion Markup Language (SAML) is an XML-based open standard data format for exchanging authentication and authorization data between an identity provider and a service provider. The most important requirement that SAML addresses is web browser single sign-on (SSO). The SAML specification defines three roles: the user, the identity provider (IdP), and the service provider (SP). The principal requests a service from the service provider. The service provider requests and obtains an identity assertion from the identity provider. On the basis of this assertion, the service provider can make an access control decision; the service provider can decide whether to perform some service for the connected user. Before delivering the identity assertion to the SP, the IdP may request some information from the principal (for example, a user name and password) in order to authenticate the principal. In SAML, one identity provider may provide SAML assertions to many service providers. Similarly, one SP may rely on and trust assertions from many independent IdPs.

Extensible Access Control Markup Language (XACML) is a standard that defines a declarative access control policy language implemented in XML and a processing model

describing how to evaluate access requests according to the rules defined in policies. One of the goals of XACML is to promote common terminology and interoperability between access control implementations by multiple vendors. The XACML model supports and encourages the separation of the access decision from the point of use. When the client is decoupled from the access decision, authorization policies can be updated on the fly and affect all clients immediately.

## 2.4. Web Service Example 1

Figure 2.2 [10] shows a web service implementation using Microsoft technology ASP.NET. ePortal is running on IIS web server. If ePortal requests appropriate eBusiness service (products list, price or order), IIS web server sends SOAP/HTTP request to SOAP server named StoreFrontService that is running on eBusiness. The SOAP/HTTP server connects the web service and COM+ implementation of business logic.

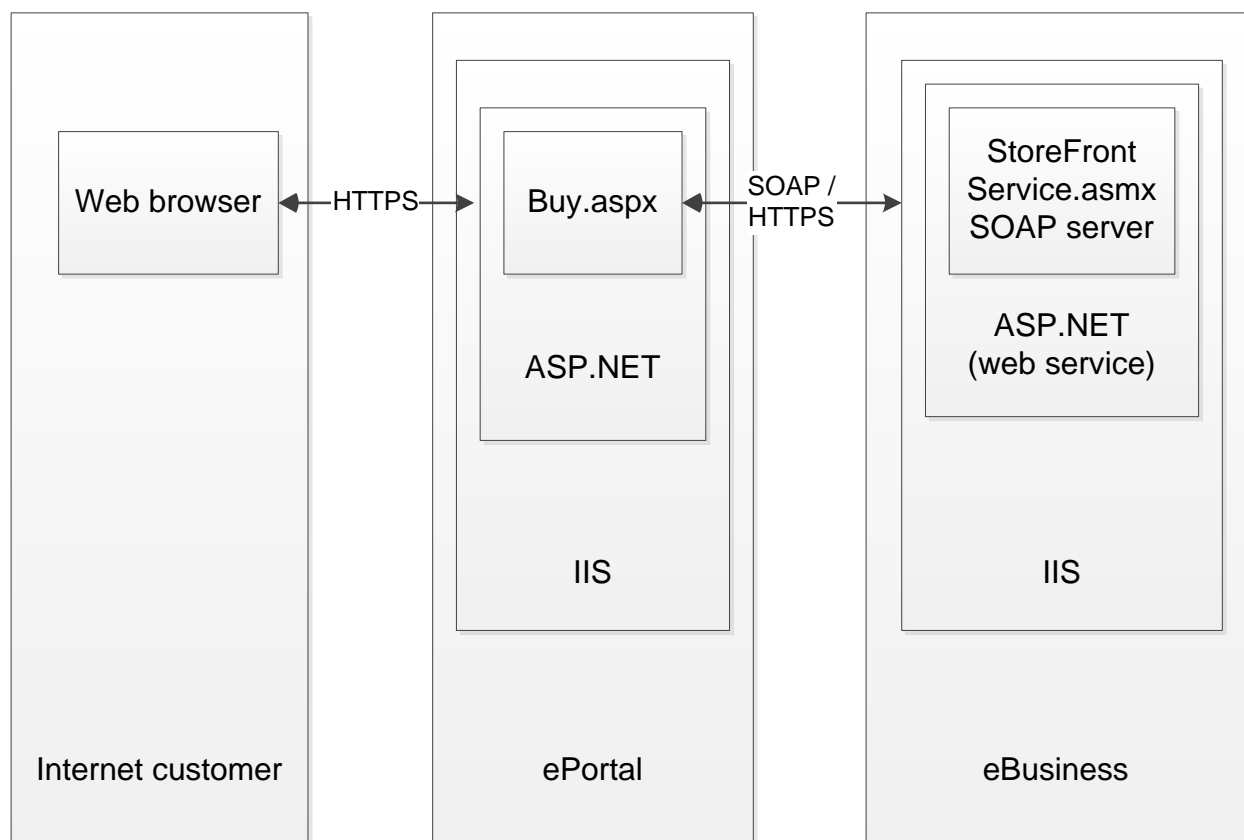


Figure 2.2 Implementation of a Web service using ASP.NET [10]

The basic security requirements are:

- 1) Encryption. All traffic between client browser and ePortal and between ePortal and eBusiness must be encrypted because of customer privacy and storing credit cards data confidentiality.
- 2) Authentication. Visitors, members and staff must be authenticated, because they have different roles.
- 3) Authorization. Visitors, members and staff must be permitted to access specific pages depending on their role.

We have two security requirements:

- 1) HTTP connection between client browser and ePortal.
- 2) SOAP/HTTP connection between ePortal and eBusiness. ePortal sends credential information to eBusiness.

Existing Microsoft security technique takes care of all security issues. This technique provides the following security features:

1. Encryption.

SSL protects all traffic between browser and portal and between ePortal and eBusiness. Customer accesses web server via HTTPS that encrypts traffic. For this purpose, ePortal and eBusiness are configured to require secure channel. SSL connection guaranties message confidentiality and integrity.

2. Authentication

HTTP basic authentication is part of Microsoft environment. Visitors, members and staff have username and password. Each person needs to enter this data to access resources. Each user is mapped to Windows user. Visitors have access to unprotected resources as anonymous user, so they do not need log in. Basic authentication and SSL guarantee that passwords cannot be stolen as network traffic. ePortal IIS does not perform password authentication check. It impersonates user and forwards his username and password to eBusiness. ASP.NET configuration file of eBusiness uses Windows authentication mode, so ASP.NET web service uses authentication performed by IIS web server of eBusiness.

3. Authorization

When a client requests access to a web service method (for example, he is asking for the price of a product), eBusiness Windows uses authenticated identity provided by IIS and checks if this user is permitted to access the service according to his role.

This example has a few limitations.

I described Microsoft technology only. It's easy to implement and deploy application using single technology. But primary advantage of web service is support of cross-platform

applications (for example, connecting ASP.NET and Java applications). Using single technology terminates this advantage of web service.

The second limitation is that I described using IIS security mechanism to authenticate users and protect network traffic. It's better to provide additional protection layers because of IIS security weakness. In example 1 there is no accountability service that records accesses in audit log. It's very important to trace sources to detect intruder attacks.

Password-based authentication has a weak security. It's very risky to use it for high-value transactions. Simple passwords are easy to guess. Complex passwords are easy to steal, because people write it on notes. The example only guaranties that password is not exposed on Internet.

The example uses impersonation model. Client logs in into ePortal using user name and password, and ePortal forwards username and password to eBusiness, and impersonates user making SOAP request to eBusiness. eBusiness thinks that it receives request directly from user. If StoreFrontService of eBusiness needs to access other applications, it again forwards same user name and password and impersonates user. In this case, user name and password must go to different servers. If attacker cracks one of the servers, he will get all passwords, and it will damage all servers.

It's better to use different authentication mechanisms and databases. If ePortal and eBusiness are different companies, they will not share user names and passwords in their databases. Web service that has many users should not use Windows-based authentication, it will use Web SSO authentication system. In a typical scenario ePortal authenticates user with own database and tells to eBusiness that user is authenticated. ePortal tracks visitor, member and staff role membership and sends user's identity and role to eBusiness. So eBusiness does not need to authenticate user again, because it gets user and role information from ePortal.

## **2.5. Web Service Example 2**

Figure 2.3 [10] illustrates an example where an eBuyer user accesses ePortal via a web service purchasing system using the browser. There are two service subscribers: eBuyer and ePortal. Service subscriber is business or business unit. eBuyer purchasing system is a service subscriber to ePortal. Users of eBuyer can be initiators of transactions at ePortal. ePortal is subscriber to services provided by eBusiness. eBusiness responds to product and pricing requests, and purchasing requests from ePortal.

ePortal takes purchasing requirements of the buyer and presents an offer that matches the purchasing criteria. eBusiness receives requests from ePortal about products, prices, and availability. eBusiness also receives purchase orders from ePortal.

Intermediary that handles web service message can change content of the message or process header and leave body without changes. ePortal has an accounting system. Accounting system is intermediary and receives buy requests on the way to eBusiness, and records transactions. Funds are collected from eBuyer and paid to eBusiness. Accounting system does not change the message body.

ePortal extracts part of messages received from eBusiness and eBuyer. It creates new messages. It is another kind of intermediary interaction.

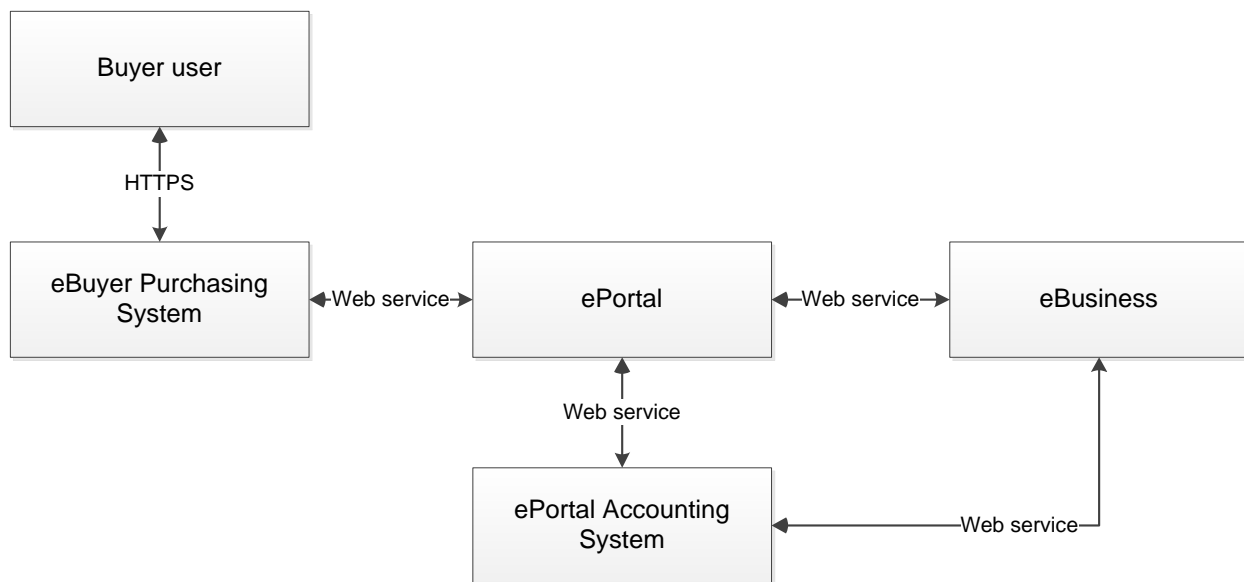


Figure 2.3 Web service purchasing example [10]

### **2.5.1. Authentication Requirements**

ePortal needs to verify identity of customer John before request initiating. Customer wants to know price of product. Before ePortal processes request, it makes sure who made request. ePortal wants to decide if request is legitimate. John uses browser to send request. Customer can use non-browser client or make purchase order by email. ePortal must verify the sender.

When ePortal determines that it must request web service from eBusiness, eBusiness needs two authenticated identities: ePortal and Buyer. First, it needs to verify who sent the request. It is important because eBusiness' business relationship is with ePortal, and only companies with which it has business relationships can make requests. eBusiness also wants to make sure that it will be paid for providing web service, and ePortal is responsible for payment. Second, although there are other options we demand that eBusiness knows who initiated the request. It is important when eBusiness must take action on behalf of the initiator. For instance, eBusiness may not mind responding with pricing information and sharing it with Joe.

eBusiness doesn't want to ship product unless it knows that Joe is a legitimate buyer. In this example eBusiness cannot directly authenticate John. eBusiness accepts customer authentication of ePortal.

There are four requirements for authenticated identity during interaction between eBusiness and ePortal:

- ePortal must know who is the initiator.
- eBusiness must be sure that it received SOAP request from ePortal.
- eBusiness must know exactly who is the initiator.
- ePortal must be sure that eBusiness sent SOAP response.

There are two categories of authentication – connection-oriented and document-oriented. Connection-oriented authentication system identifies who is at the other end of connection. Document-oriented authentication system attaches or embeds authentication token with message. Message and authentication token can be transported using HTTP, SMTP, or FTP protocols. Document-oriented authentication system embeds information about entity in the body of document.

### **2.5.2. Data protection**

Information in the message between eBusiness and ePortal is not for general use. We need to protect it from stealing and modifying.



Private information is exchanged between eBuyer and ePortal, and between ePortal and eBusiness. ePortal receives request information from eBuyer and sends request to eBusiness. eBusiness sends product information, price and payment instruction to ePortal. ePortal receives this offer and sends offer that includes payment instruction to eBuyer. Neither ePortal nor eBuyer can be allowed to understand payment instruction.

In opposite direction eBuyer sends accounts payable information in case of acceptance of eBusiness' offer. ePortal receives acceptance and sends it to eBusiness. This message passes through accounting system of ePortal. Only ePortal accounting system (not ePortal and not eBusiness) must be able to understand this data. Accounting system must be sure that money is collected from eBuyer and paid to eBusiness. SOAP header instructs accounting system to record content of the message and makes entries into accounts payable and accounts receivable systems. After that, when header is no longer needed, audit system passes SOAP message without audit header. SOAP message can be created from other received SOAP messages.

### **2.5.3. Authorization**

There are three critical aspects of securing web services: authentication, message protection, and authorization.

There are two aspects of authorization. The first aspect is that eBusiness needs to ensure that it provides to ePortal only services allowed for ePortal. eBusiness can provide services, and ePortal is not contracted to use some of them. So eBusiness needs to be sure that it restricts ePortal to such services. The second aspect is that eBusiness wants to be sure that initiator is entitled to request service on its behalf. So if initiator is John, transaction account will be one that John (and not Sue, for example) is entitled to use.

There are the following requirements for eBusiness:

- eBusiness needs to be sure that ePortal is entitled to use the services it requested.
- eBusiness needs to be sure that initiator John has authority to buy the product.

We can use SAML assertion. It includes information about John and allows ePortal to include attributes about user. When eBusiness verifies signature of ePortal, it can be sure that subject of SAML assertion is the initiator of transaction. eBusiness needs to be sure that ePortal authenticated the initiator.

## 3. SAML

### 3.1. OASIS

OASIS (Organization for Advancement of Structured Information Standards) is a global consortium that drives development and adoption of e-business and web service standards. One of the purposes of OASIS is create standards based on XML. OASIS developed SAML, standard XML-based framework for secure exchange of authentication and authorization information. OASIS also released XACML, standard XML-based protocol for access control policies.

### 3.2. What is SAML?

There are two main problems in web services security.

The first problem is how to interpret security context information when data is transferred from source application to target application. If source and target are the same division of the same company, we can use the same technology, so it's not difficult to solve this problem. The real problem is when we have two different companies with different security policies.

The second problem is single sign-on capability (SSO). People want to log on to system only once to have access to different web service applications without additional login.

Solution of these problems is standard representation of security data. In this case, different security services can recognize security data regardless of security policy and technology they are using. OASIS Security Services Technical Committee released SAML specification.

Security Assertion Markup Language (SAML) is XML-based framework to exchange security related information between service consumers, identity providers and service providers. The security information is expressed in terms of assertions, statements about a subject or a user in the form of the SAML token.

SAML solves the following security problems:

- *Interoperability* – ability to communicate between security systems within different tiers.
- *Privacy* – ability to protect identity and personal data of user (for example, credit card number).
- *Federation* – ability to securely exchange data between different companies.
- *SSO (single sign-on)* – ability to log in once to access different applications.

SAML solves problem of secure interoperability of web services. SAML has an ability to define standard format for security information and standard way to transmit security data among different applications.

Identity provider provides local authentication services to the principal (user). SAML doesn't specify implementation of local services. SAML does not care how local authentication service was implemented. When principal sends request, identity provider passes SAML assertion to service provider. After that, service provider makes access control decision.

The main part of SAML specification is XML Schema. XML Schema defines representation of security data. This representation is assertion when trusted security service says that authentication or authorization is correct. For example, if authentication assertion is representation by third party, the principal is authenticated. If target trusts third party, it accepts assertion as true and accepts the principal by authentication assertion as authenticated.

Third-party services are trusted sources when they assert correctness of authentication, attribute activity, and authorization. They return SAML assertions to requesting party. If client tries to access target, it will pass along SAML assertion from third party. Target verifies that assertion is from the third party it trusts. If it trusts, it uses this information and knows that user is authenticated correctly. Now target has accepted SAML authentication assertion. Target goes to another third-party service; it requests attribute assertion for this authenticated user and passes authenticated assertion. Third-party service returns to target attribute assertion that contains attributes for this user. Third-party attribute service guarantees that attributes are correct. And finally, target sends request to third-party authorization service, passes attribute or authentication assertion and asks if user is permitted to perform activity on target resource. Authorization can be more complex than authentication and attribute retrieval.

SAML is written in XML. So it uses such advantages of XML as platform and language independence. SAML solves the problem of Web browser single sign-on (SSO).

SAML is designed to work with other specifications. SAML uses the following standards:

- XML (Extensible Markup Language). XML is root for SAML.
- XML Schema. SAML assertions and protocols are specified using XML Schema.
- XML Signature. SAML uses digital signature based on XML Signature for authentication and message integrity.
- XML Encryption. SAML uses XML Encryption for providing elements for encrypted name identifiers, encrypted attributes, and encrypted assertions.
- HTTP (Hypertext Transfer Protocol). SAML uses HTTP as communication protocol.
- SOAP (Simple Object Access Protocol). SAML specifies the use of SOAP.

- SAML 1.0 was adopted as OASIS standard in November 2003. SAML 1.1 was ratified by SSTC (OASIS Security Services Technical Committee) as OASIS standard in September 2003. SAML 2.0 became OASIS standard in March 2005.

### 3.3. The Scope of SAML

The main goal of SAML is to define security documents. Specification defines how to talk to the SAML services. It uses the term *bindings* to describe a standard way to request authentication, attributes, or authorization decisions from the appropriate SAML service and for authority to respond to the requestor. *Profiles* are descriptions of how protocols transfer SAML assertions from one application to another.

Let's describe some terms and concepts. Entity is an active element of a network system. Principal is an entity whose identity can be authenticated. Subject is a principal in the context of a security domain.

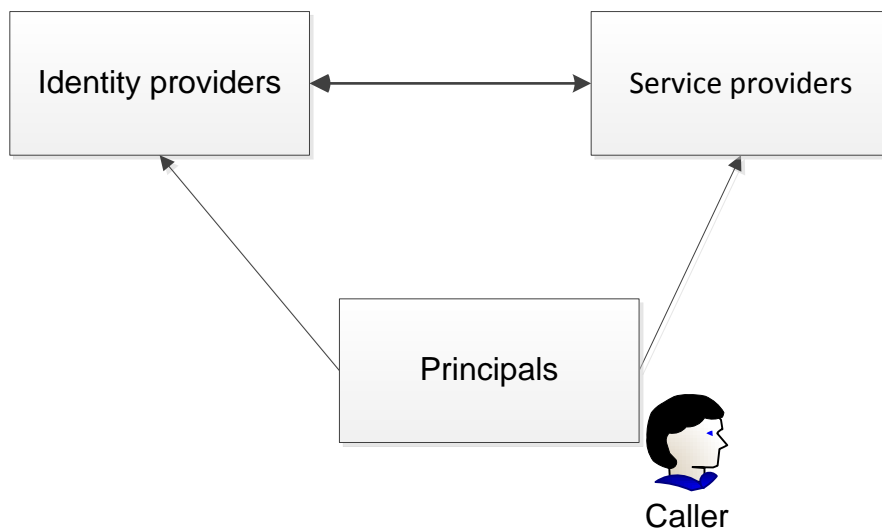


Figure 3.1 Identity providers and service providers

Asserting party (SAML authority) is an entity that produces SAML assertions. Service provider (SP) is an entity that decides to take an action based on information from another system entity. Identity provider (IdP) is an entity that creates, maintains, and manages identity information for principals and provides principal authentication to other service providers. In the WS-Federation Model, an identity provider is a Security Token Service (STS). Service Providers depend on an Identity Provider or Security Token Service to do the user authentication. Service

provider is an entity that provides services to principals or other entities. A service provider relies on a trusted Identity Provider or Security Token Service (STS) for authentication and authorization. SAML's service provider depends on receiving assertions from a SAML authority or asserting party, a SAML Identity Provider.

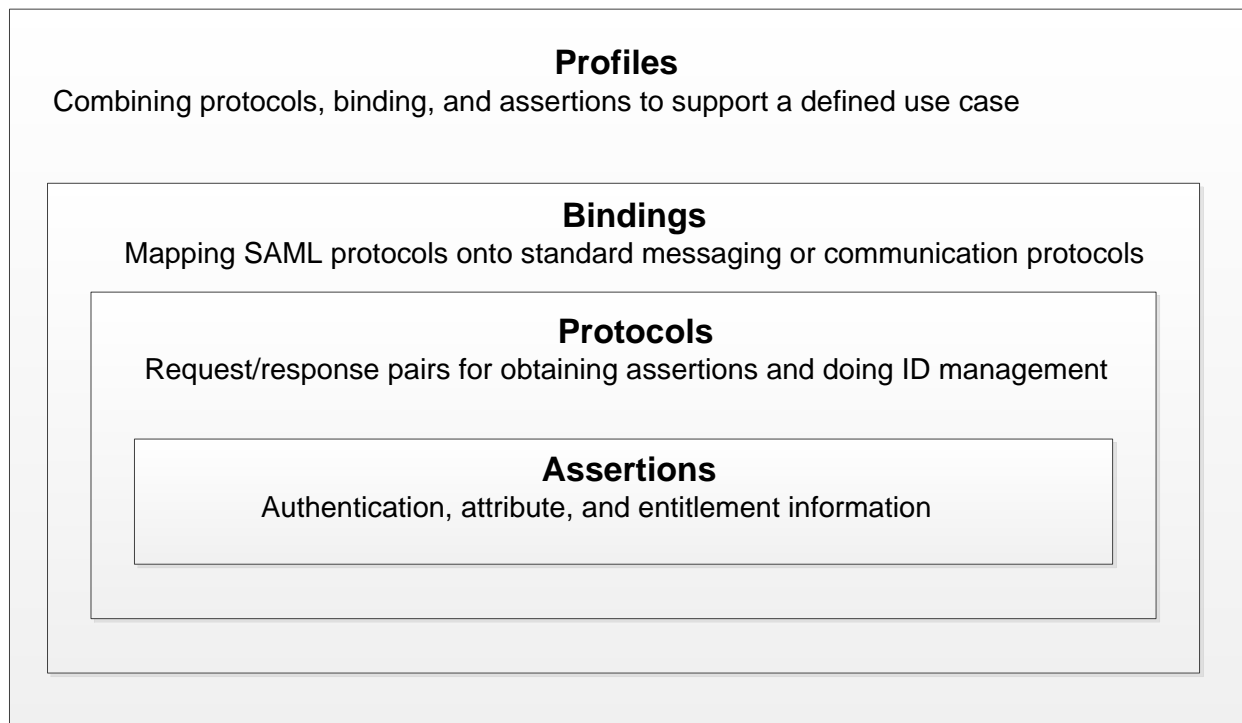


Figure 3.2 SAML concepts [15]

Client sends request to target organization that has SAML authentication assertion. Client retrieved an authentication assertion from authentication authority when he was authenticated. Target organization receives request. Organization examines authentication assertion, and if assertion is correct and it is from trusted authority, the organization is satisfied. After that target organization goes to SAML attribute authority, passes authentication assertion to attribute authority and requests SAML attribute assertion. Attribute authority returns attribute assertion with attributes of the customer to the organization, or the client can get attribute assertion also from the attribute authority and give it to the target organization. In any case, the target organization sends the SAML authorization request to the authorization authority with the name and ID of the resource that the client needs to access. Authorization request contains the assertion that has an authentication statement and/or the attributes statement. Authorization authority makes grant or deny decision and returns it.

### 3.4. SAML Assertions

Each assertion has a set of data that is common to all assertions.

We can divide SAML assertions into two general areas. The first area is common for all SAML assertions and contains things like version number, security principal, and conditions. Namespaces define originating organization and which specifications this assertion uses. The second contains actual statements about authentication, attributes, or authorization.

*Subject* of SAML contains identity of security principal of this assertion. *Subject* contains domain and name. *Subject confirmation* is an alternative way of identifying the subject. The XML schema below [10] shows *Subject* and *Subject confirmation*.

```
<element name="Subject" type="saml:SubjectType"/>
<complexType name="SubjectType">
  <choice>
    <sequence>
      <element ref="saml:NameIdentifier"/>
      <element ref="saml:SubjectConfirmation" minOccurs="0"/>
    </sequence>
    <element ref="saml:SubjectConfirmation"/>
  </choice>
</complexType>
```

The first line defines an element, which is a basic type in XML, whose name is *Subject*. The *Subject* is of type *saml:SubjectType*. The *saml:* means that the *SubjectType* is defined by SAML. On the second line, we see the beginning of the definition of *SubjectType*. It is a *complexType*, so it is composed of a number of other definitions. A simple type (in contrast to a complex type) can consist of only one element such as a string or an integer. The third line says that the things that compose the *SubjectType* are a choice of two elements. The first choice is a sequence of a *NameIdentifier* followed by a *SubjectConfirmation*. The *SubjectConfirmation* is followed by a *minOccurs="0"*. The *SubjectConfirmation* is optional. There is also a term *maxOccurs* that tell how many times the element can occur. The default value of both *minOccurs* and *maxOccurs* is 1.

In the example below we can see a SAML assertion [11].

```
<Assertion Version="2.0"
  ID="18d9be83-4e26-4c47-825a-5df08b0ebdf8"
  IssueInstant="2014-08-18T02:00:00.00Z"
  xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
  <Issuer>idp.sample.com</Issuer>
  <Subject>
```

```

<NameID NameQualifier="sample.com">b75e15a6-0c2b-4ff6-84a3-94428dbb42b2</NameID>
<SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
  <SubjectConfirmationData
    NotOnOrAfter="2014-08-18T02:05:00.00Z "
    Recipient="http://sp.sample.com/SSO.asmx" />
  </SubjectConfirmation>
</Subject>
<Conditions NotBefore="2014-08-18T02:00:00.00Z "
  NotOnOrAfter="2014-08-18T02:05:00.00Z ">
  <AudienceRestriction>
    <Audience>davidsps8.com</Audience>
  </AudienceRestriction>
</Conditions>
<AuthnStatement AuthnInstant="2014-08-18T02:00:00.00Z ">
  <AuthnContext>
    <AuthnContextClassRef>AuthnContextClassRef</AuthnContextClassRef>
  </AuthnContext>
</AuthnStatement>
<AttributeStatement>
  <Attribute Name="email"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
    <AttributeValue xsi:type="xsd:string">andrey@gmail.com</AttributeValue>
  </Attribute>
</AttributeStatement>
</Assertion>

```

The example above shows an assertion issued by an identity provider (<http://idp.sample.com>) to a service provider (<http://sp.sample.com>). The assertion includes both an Authentication Assertion <saml:AuthnStatement> and an Attribute Assertion <saml:AttributeStatement>, which the service provider uses to make an access control decision. The assertion encodes the following information: The assertion ("18d9be83-4e26-4c47-825a-5df08b0ebdf8") was issued at time "2014-08-18T02:00:00.00Z" by identity provider (<http://idp.sample.com>) regarding subject (b75e15a6-0c2b-4ff6-84a3-94428dbb42b2) exclusively for service provider (<http://www.sample.com/SSO.asmx>). The authentication statement asserts the following: The principal identified in the <saml:Subject> element was authenticated at the time "2014-08-18T02:00:00.00Z" by means of a password sent using a protected channel.

The top-level statement portion of assertion is abstract element. Valid assertion must contain one of three possible statements defined by SAML: authentication, authorization, or attribute.

### 3.5. Authentication statement

Authentication statement is derived from abstract *SubjectStatementAbstractType*. *AuthenticationMethod* element identifies the type of authentication [10], *AuthenticationInstant* contains time of authentication.

The following fragment shows authentication statement:

```
<sequence>
  <element ref="saml:SubjectLocality" minOccurs="0"/>
  <element ref="saml:AuthorityBinding" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
<attribute name="AuthenticationMethod" type="anyURI" use="required"/>
<attribute name="AuthenticationInstant" type="dateTime" use="required"/>
```

### 3.6. Attribute statement

The following fragment shows attribute statement:

```
<sequence>
  <element ref="saml:Attribute" maxOccurs="unbounded"/>
</sequence>
```

The following fragment shows attribute statement, when attribute element contains *AttributeValue* element:

```
<sequence>
  <element ref="saml:AttributeValue" maxOccurs="unbounded"/>
</sequence>
```

*AttributeType* inherits from *AttributeStatementType*. The following fragment shows *AttributeDesignatorType*:

```
<complexType name="AttributeDesignatorType">
  <attribute name="AttributeName" type="string" use="required"/>
  <attribute name="AttributeNamespace" type="anyURI" use="required"/>
</complexType>
```

Below I give an example of the *AttributeQuery* [11]:



```

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap/envelope/">
  <env:Body>
    <samlp:AttributeQuery xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
      xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
      ID="aaf23196-1773-2113-474a-fe114412ab72"
      Version="2.0"
      IssueInstant="2006-07-17T20:31:40Z">
      <saml:Issuer>http://example.sp.com</saml:Issuer>
      <saml:Subject>
        <saml:NameID
          Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">
          C=US, O=NCSA-TEST, OU=User, CN=trscavo@uiuc.edu
        </saml:NameID>
      </saml:Subject>
      <saml:Attribute
        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
        Name="urn:oid:2.5.4.42"
        FriendlyName="givenName">
      </saml:Attribute>
    </samlp:AttributeQuery>
  </env:Body>
</env:Envelope>

```

The example shows an XML document containing an example SAML attribute query message being transported within a SOAP envelope. The SOAP envelope starts on line 2. The SAML attribute query starts on line 5. It is embedded in a SOAP body element starting on line 4. The attribute query contains various required and optional XML attributes including declarations of the SAML assertion and protocol namespaces, and the message ID. The request specifies a number of optional elements. This includes, for example, the requested attribute (givenName).

### 3.7. Authorization statement

In case of authorization, schema has decision element to indicate, if authorization is granted or not. *Decision* element returns result of decision. *Decision* element can take the following values: Permit, Deny, and Indeterminate.

The following fragment shows an authorization statement [10]:

```

<complexContent>
  <extension base="saml:SubjectStatementAbstractType">
    <sequence>
      <element ref="saml:Action" "maxOccurs = 'unbounded'"/>
      <element ref="saml:Evidence" minOccurs="0"/>
    </sequence>
    <attribute name="Resource" type="anyURI" use="required"/>
    <attribute name="Decision" type="saml:DecisionType" "use="required"/>
  </extension>
</complexContent>

```

The *Decision* returns the results of a decision. A *Decision* element may take one of the following values: Permit, Deny, or Indeterminate.

### 3.8. SAML Request/Response

Specification defines protocol for requesting assertions from third parties and returning response with completed assertion.

There are three options of SAML request/response: for authentication, for attributes, and for authorization. Request and response are XML documents. Header of request/response has definition of namespaces and import of other specification schemas. The imported schemas are digital signature specification, assertion schema and XML schema itself.

### 3.9. SAML Request

SAML request is a format to ask questions about authentication, attribute, and authorization authority. In case of authentication authority, SAML request asks if the subject is authenticated. Response will be SAML authentication assertion. In case of attribute authority, SAML request asks: what are the attributes for this authenticated subject? Response will be SAML attribute assertion. In case of authorization authority, SAML request asks if subject can do specific action on specific resource. Response will be SAML authorization assertion.

Request portion in SAML protocol starts with abstract type *RequestAbstractType*. After that, we can see the version of specification and time of request. The type of response is defined in *RespondWith* element with *AuthenticationStatement*, *AttributeStatement*, and *AuthorizationStatement*.

Next is the part of the schema of a SAML request element [10]:

```
<element name="Request" type="samlp:RequestType"/>
<complexType name="RequestType">
  <complexContent>
    <extension base="samlp:RequestAbstractType">
      <choice>
        <element ref="samlp:Query"/>
        <element ref="samlp:SubjectQuery"/>
        <element ref="samlp:AuthenticationQuery"/>
        <element ref="samlp:AttributeQuery"/>
        <element ref="samlp:AuthorizationDecisionQuery"/>
        <element ref="saml:AssertionIDReference" maxOccurs="unbounded"/>
        <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

Request type can be: Query, SubjectQuery, AuthenticationQuery, AttributeQuery, AuthorizationQuery, AssertionIDReference, or AssertionArtifact.

*AuthenticationQuery* element asks: what authentication assertions are available for subject? *AttributeQuery* extends *SubjectQueryAbstractType*. Here is the sample of *AttributeQuery*:

```
<extension base="samlp:SubjectQueryAbstractType">
  <sequence>
    <element ref="saml:AttributeDesignator" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Resource" type="URI reference" use="optional"/>
</extension>
```

*AuthorizationQuery* is used to request that authorization authority answer the question, if subject is permitted to perform stated action on given resource. The elements of *AuthorizationQuery* are name of resource (defined by URI), actions and evidence (optional). *Actions* and *Evidence* are defined in SAML assertions.

Here is the schema of *AuthorizationDecisionQuery*:

```
<extension base="samlp:SubjectQueryAbstractType">
  <sequence>
    <element ref="saml:Action" maxOccurs="unbounded"/>
    <element ref="saml:Evidence" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <attribute name="Resource" type="anyURI" use="required"/>
</extension>
```

### 3.10. SAML Response

Response, as request, starts with common portion that contains version number, *Subject*, etc. The new attributes are *ResponseID* and *InResponseTo*. *InResponseTo* connects response to corresponding request. *InResponseTo* is *RequestID* of corresponding request. *Response* element has structure similar to request, derived from *ResponseAbstractType*. The main element of response is assertion or assertions.

The status of response can be *Success*, *VersionMismatch*, *Requestor*, or *Responder*. Element *StatusMessage* returns explanatory message.

An example of a sample SAML Response is below [10]:

```
<samlp:Response
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="identifier_2"
  InResponseTo="identifier_1"
```

```

Version="2.0"
IssueInstant="2014-08-18T09:00:00"
Destination="https://sp.sample.com/SAML2/SSO/POST">
<saml:Issuer>https://idp.sample.com/SAML2</saml:Issuer>
<samlp:Status>
  <samlp:StatusCode
    Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="identifier_3"
  Version="2.0"
  IssueInstant="2014-08-18T09:00:00">
<saml:Issuer>https://idp.sample.com/SAML2</saml:Issuer>
<!-- a POSTed assertion MUST be signed -->
<ds:Signature
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
<saml:Subject>
  <saml:NameID
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
    6fa71759-f712-4632-96e6-6841ddd996ae
  </saml:NameID>
  <saml:SubjectConfirmation
    Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <saml:SubjectConfirmationData
      InResponseTo="identifier_1"
      Recipient="https://sp.sample.com/SAML2/SSO/POST"
      NotOnOrAfter="2014-08-18T09:05:00"/>
    </saml:SubjectConfirmation>
  </saml:Subject>
  <saml:Conditions
    NotBefore="2014-08-18T09:05:00"
    NotOnOrAfter="2014-08-18T09:15:00">
    <saml:AudienceRestriction>
      <saml:Audience>https://sp.sample.com/SAML2</saml:Audience>
    </saml:AudienceRestriction>
    </saml:Conditions>
  <saml:AuthnStatement
    AuthnInstant="2014-08-18T09:00:00"
    SessionIndex="identifier_3">
    <saml:AuthnContext>
      <saml:AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
      </saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
</saml:Assertion>
</samlp:Response>

```

### 3.11. Bindings

*Binding* is the term that describes standard way to request authentication, attributes, and authorization from SAML service. Bindings define how to pass assertion in messaging protocol. Two bindings defined in initial specification of SAML 1.0 are SOAP and HTTP POST protocol.

SAML 2.0 defines the following bindings:

- SAML SOAP Binding;
- Reverse SOAP Binding;
- HTTP Redirect (GET) Binding;
- HTTP POST Binding;
- HTTP Artifact Binding;
- SAML URI Binding.

### 3.12. SOAP Binding

Security information in SOAP message is contained in SOAP header. The purpose of SOAP binding is not to secure message, but request and receive security information, specifically SAML assertions. SAML information is the message and is contained in SOAP body. Message body can contain only SAML request or reply when it is used to send request or response for SAML assertions to or from SAML authority. SOAP message cannot contain more than one request or response. Response is a SOAP message that contains the SAML response in the SOAP body or SOAP fault code.

### 3.13. Profiles

*Profiles* are descriptions how protocol transfers SAML assertions from one application to another.

*Artifact* is a token that is passed from web browser to web server that allows web server to get assertions from trusted third party.

Browser POST profile describes how to pass assertions in HTTP POST request securely.

SAML 2.0 has the following profiles:

- SSO Profiles
  - Web Browser SSO Profile;
  - ECP (Enhanced Client or Proxy) Profile;
  - Identity Provider Discovery Profile;
  - Single Logout Profile;
  - Name Identifier Management Profile;
- Artifact Resolution Profile;
- Assertion Query / Request Profile;
- Name Identifier Mapping Profile;
- SAML Attribute Profile.

### 3.14. SAML Artifact

*Artifact* is a small identifier that the browser first obtains from the source site. SAML assertions may fit into HTTP URL from the web browser. Typically, the browser passes security information in cookies. But it's impossible to use cookies in cross-domain applications, so it's possible to use URL instead, to pass security data.

We could share a cookie between `foo.example.com` and `new.example.com` but never between `example.com` and `example2.com` and that's for security reasons. In order to pass cookie credentials we need to include "Access-Control-Allow-Credentials:true" header. When a request is made for a resource with credentials, if this header is not returned with the resource, the response is ignored by the browser and is not returned with web content. "Access-Control-Allow-Origin" must not use wildcard (\*). The origin is a URI, indicating the server from which the request initiated. It does not include any path information, but only the server name; we must specify the exact domain. Wildcard (\*) is too permissive and would defeat the use of credentials. So we have to use header "Access-Control-Allow-Origin: `http://foo.example.com`".

The artifact structure consists of the two-byte field with additional optional data. *SourceID* is used to identify the source site. *AssertionHandle* is used to locate assertion for browser user.

There are two cases how Browser Artifact Protocol can initiate SSO connection:

1. Browser tries to access target without artifact, target causes browser request to redirect to third-party authority, called source site in protocol.
2. Browser accesses source site on its own and requests artifact.

In all cases, browser must authenticate itself with the source site. It can be done before request is sent to target, or target may force authentication in first method.

In first case, browser reaches source site by redirection and requests artifact by presenting information on target. If browser is not authenticated to source site, browser is required to log in. It will be single login request for this session with the source site. Source site constructs artifact and returns it to browser, and redirects browser to the target.

In second case, browser accesses source site, having logged previously or at time of access, and requests artifact for target.

Browser presents artifact or artifacts to the target. Target determines location of source site, makes SAML request to source site, sending artifact(s) that it received from the browser. Target must use SAML binding protocol in accessing source site. Request is SAML request that is asking for assertion(s) associated with artifact(s). Source site inspects artifact(s), creates or retrieves proper assertion(s) and returns it to the target in SAML response.

### **3.15. SAML POST**

HTTP POST protocol transmits assertions from browser to the target. Browser makes request to source site, passing requested target site the browser wants to access. Source site builds assertion(s) and puts it in HTTP form. Source site returns form that contains assertion(s) to browser with redirect to target. Assertions must be digitally signed.

Section 4 below contains an extended SAML sample.

## 4. High-Level SAML Use Cases

### 4.1. Web Single Sign-On Use Case

Web browser single sign-on (SSO) is the primary SAML use case. User (web browser) requests web resource protected by SAML service provider. Service provider needs to know identity of requesting web browser. Service provider sends authentication request to SAML identity provider through user agent.

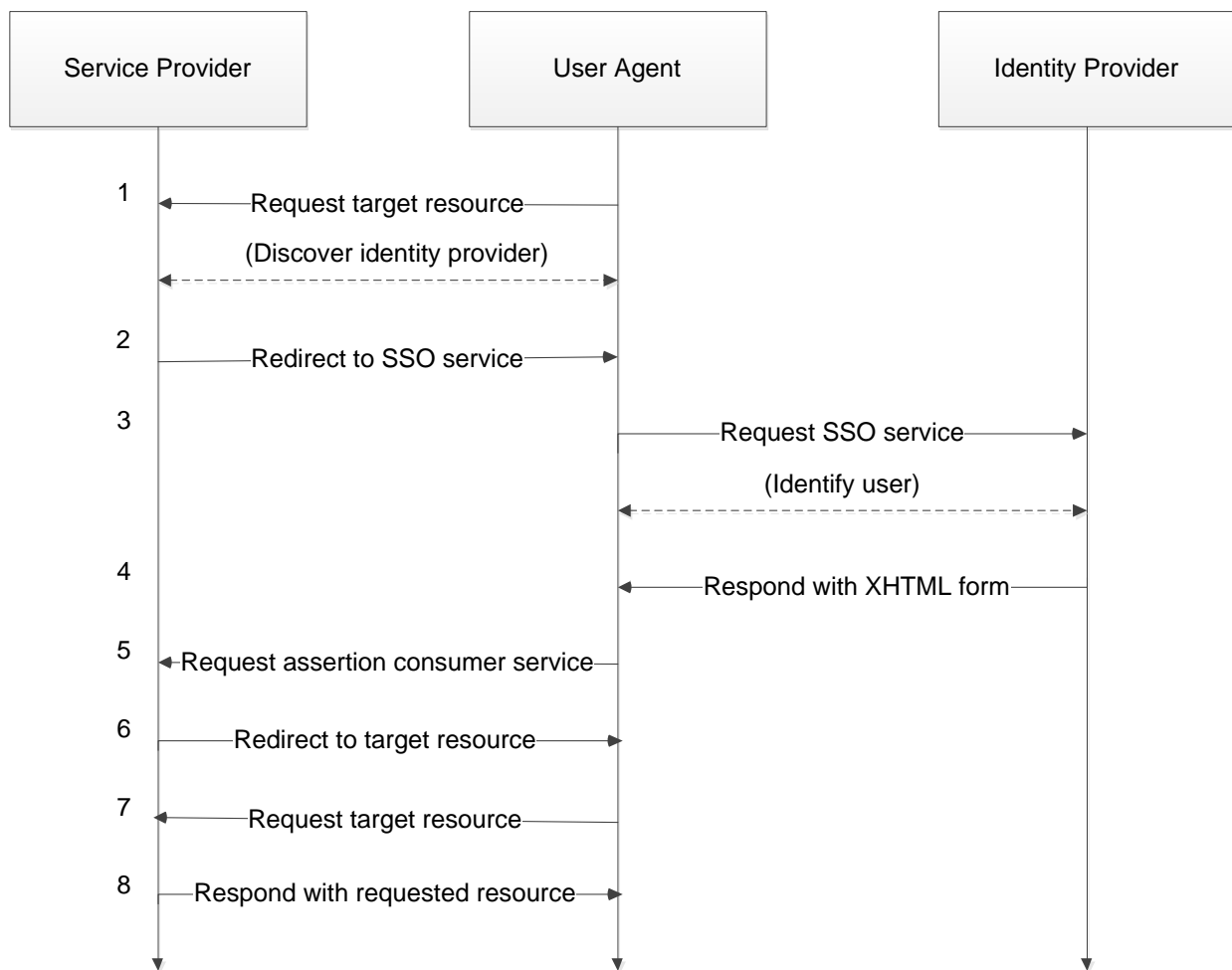


Figure 4.1 SAML web browser SSO [13]



1. Request target resource at Service Provider.  
Principal requests target resource at service provider via HTTP user agent. Service provider does security check on behalf of target resource. If valid security context already exists at service provider, we skip steps 2-7.  
Example: <https://www.news.com/resource1>
2. Redirect to SSO service at Identity Provider.  
Service provider determines user's identity provider and redirects user agent to SSO service at identity provider.  
Example: <https://www.news.com/SAML/SSO/Redirect?SAMLRequest=myrequest>
3. Request SSO service at Identity Provider.  
User agent performs GET request to SSO service at identity provider. SSO service performs security check. If user doesn't have valid security context, identity provider identifies user.
4. Respond with XHTML form.  
SSO service validates request and responds with document containing XHTML form.
5. Request assertion consumer service at Service Provider.  
User agent performs POST request to assertion consumer service at service provider.
6. Redirect to target resource.  
Assertion consumer service processes response, creates security context at service provider and redirects user agent to target resource.
7. Request target resource at Service Provider again.  
User agent requests target resource at Service Provider again.
8. Respond with requested resource.  
Security context exists, so Service Provider returns resource to user agent.

In this use case, a login has a login session (security context) on a web site (Airlines.com) and is accessing resources on that site. At some point, either explicitly or transparently, user is directed over to a partner's web site Cars.com. We assume that a federated identity for the user has been previously established between Airlines.com and Cars.com based on a business agreement between them. The identity provider site Airlines.com asserts to the service provider site Cars.com that the user is known (by referring to the user by their federated identity), was authenticated, and has certain identity attributes (for example, has a "Gold membership"). Since Cars.com trusts Airlines.com, it trusts that the user is valid and properly authenticated and thus creates a local session for the user. This use case is shown in Figure 4.2, which illustrates the fact that the user is not required to be re-authenticated when directed over to the Cars.com site.

User was first authenticated at the identity provider before accessing a protected resource at the service provider. This scenario is commonly referred to as an IdP-initiated web SSO scenario. While IdP-initiated SSO is useful in certain cases, a more common scenario starts with a user visiting a service provider site through a browser bookmark, possibly first accessing

resources that require no special authentication or authorization. When user attempts to access a protected resource at the service provider, the service provider will send the user to the identity provider with an authentication request in order to have the user log in. Thus this scenario is referred to as SP-initiated web SSO. Once logged in, the identity provider can produce an assertion that can be used by the service provider to validate the user's access rights to the protected resource. SAML 2.0 supports both the IdP-initiated and SP-initiated flows.

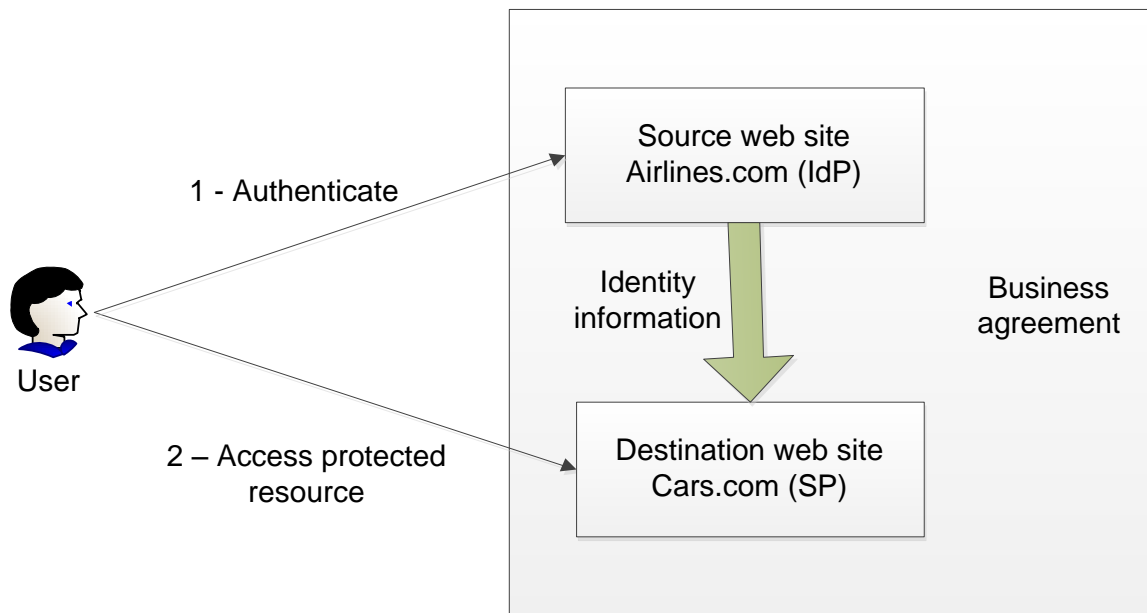


Figure 4.2 General single sign-on use case [11]

## 4.2. Identity Federation Use Case

User's identity is federated between a set of providers when there is an agreement between the providers on a set of identifiers and/or identity attributes by which the sites will refer to the user.

Figure 4.3 shows how, during web SSO, the sites can dynamically establish the federated name identifiers used in the account linking process. Airlines.com is identity provider, and Cars.com and Hotels.com are service providers. Cars.com offers car rentals, and Hotels.com offers hotel booking. Consider that a user is registered on all three providers. They have pre-existing local login accounts, but the local accounts all have different account identifiers. User Sam Foster is registered at Airlines.com as **samfoster**, on Cars.com his account is **sfoster**, and on Hotels.com it is **samf**. The sites have established an agreement to use persistent SAML privacy-preserving pseudonyms for the user's federated name identifiers. Sam has not previously federated his identities between these sites.

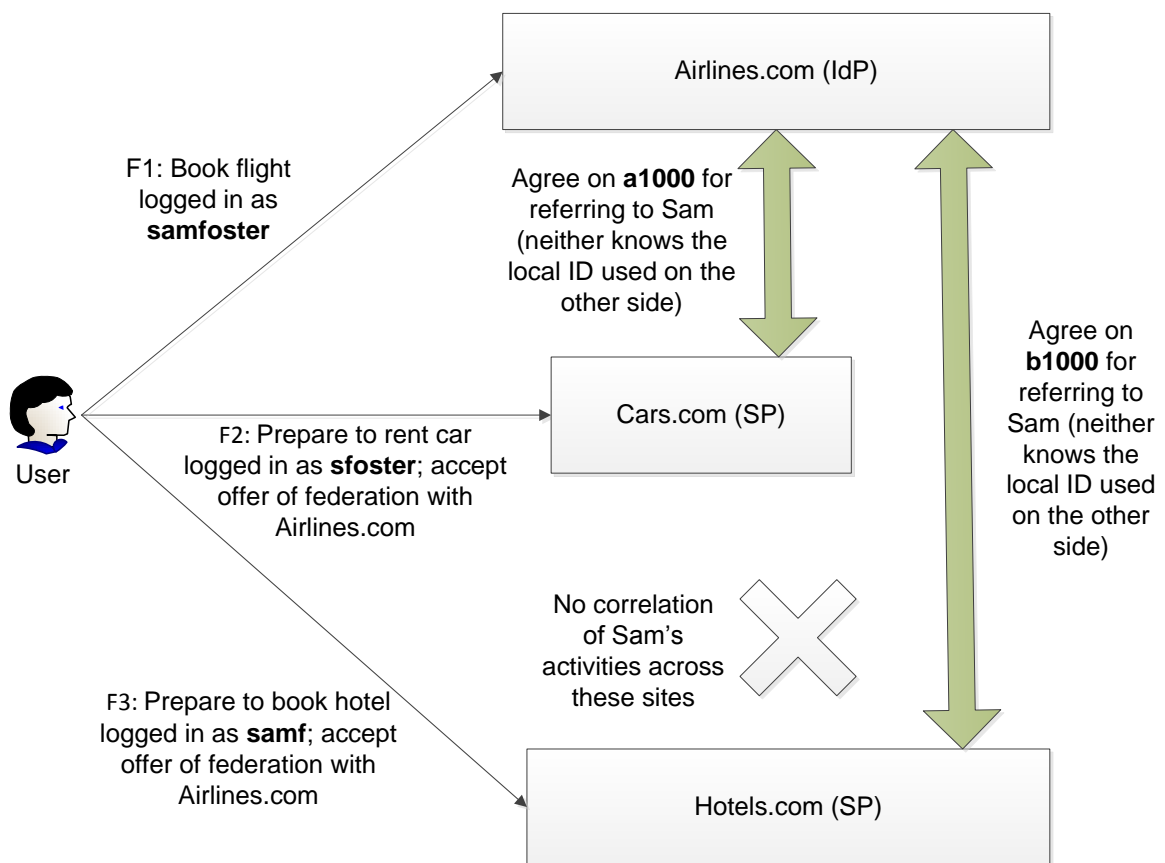


Figure 4.3 General identity federation use case [11]

The processing sequence is as follows:

1. Sam books a flight at Airlines.com using user account **samfoster** (F1).
2. Sam wants to reserve a car. He clicks on a link to visit Cars.com. This site sees that the browser user is not logged in locally but that he has previously visited their identity provider partner site Airlines.com. So Cars.com asks Sam if he would like to agree to federate his local Cars.com identity with Airlines.com (F2).
3. Sam agrees to the federation and his browser is redirected back to Airlines.com where the site creates a new pseudonym, **a1000** for Sam's use when he visits Cars.com. The pseudonym is linked to his **samfoster** account. Both providers agree to use this identifier to refer to Sam in subsequent transactions.
4. Then Sam is redirected back to Cars.com with a SAML assertion indicating that the user represented by the federated persistent identifier **a1000** is logged in at the identity provider. Since this is the first time that Cars.com has seen this identifier, it does not know the local user account to which it applies.
5. So, Sam must log in at Cars.com using his **sfoster** account. Then Cars.com attaches the identity **a1000** to the local **sfoster** account for future use with the identity provider Airlines.com. The user accounts at the identity provider and this service provider are now linked using federated name identifier **a1000**.
6. After reserving a car, Sam clicks on a link to visit Hotels.com in order to book a hotel room (F3).
7. The federation process is repeated with the identity provider Airlines.com, creating a new pseudonym, **b1000**, for IdP user **samfoster** that will be used when visiting Hotels.com.
8. Sam is then redirected back to the service provider Hotels.com with a new SAML assertion. The service provider requires Sam to log in his local **samf** user account and adds the pseudonym as the federated name identifier for future use with the IdP Airlines.com. The user accounts at the identity provider and this service provider are now linked using the federated name identifier **b1000**.

In the future, whenever Sam needs to book flight, car, or hotel, he will only need to log in once to Airlines.com before visiting Cars.com and Hotels.com. The identity provider Airlines.com will identify Sam as **a1000** to Cars.com and **b1000** to Hotels.com. Each service provider will locate Sam's local user account through the linked persistent pseudonyms and allow Sam to conduct business after the SSO exchange.

Appendix C contains SAML request/response samples.

## 5. XACML

### 5.1. Introduction to XACML

XACML (*eXtensible Access Control Markup Language*) is a standard that defines XML-based language for declarative access control policy. XACML describes how to evaluate authorization request according to policy rules. XAML model supports separation of authorization decision and point of use. It's possible to update authorization policy on the fly, and it immediately affects all clients.

XACML defines rules to allow access to resources (read / write / execute) based on characteristics of requester, characteristics of the request protocol, and authentication context. XACML defines conditions for creating rules, how rules may be combined, and how rules are processed to perform decisions. A XACML rule is defined as “Target, effect, and set of conditions” [9]. Target is decision request that refers to resources (documents, services) by subject (users or computers). For example, target may be “Write access to documents in drive D for managers.” Collecting rules together creates policy statements.

Examples:

- Only members of Accounting Department can access specific document.
- To access specific document, SSL should be used.
- Digital certificate must be used for authentication to read the document.

XACML uses *obligation* that is directive from PDP (Policy Decision Point) to PEP (Policy Enforcement Point). Directive must be performed before or after access is granted. If PEP cannot perform directive, granted access may or must not be realized. This obligation removes gap between formal requirement and policy execution. Obligation is effective way for formal requirements that are hard to implement as access control rules.

Documents accessed using policies expressed in XACML don't need to be XML documents. XACML can provide complex scenarios for access to different resources by users or computers depending not only on requestor or resource, but on other conditions. *Predicate* of rule is ability to query attribute.

Although XACML defines XACML request/response messages format, it does not provide any suggestions about using one or another transport container or protocol. Using XACML messages directly as authorization assertions impose some security problems because they do not have mechanisms to bind authority (trust) or imply security restrictions as they are provided by such SAML elements as *Issuer* or *Conditions*.

SAML 2.0 profile of XACML combines SAML security assertions format and rich functionality of the XACML policy format. This solution provides a good combination between XACML policy expression and evaluation functionality and SAML security assertion management functionality.

XACML is designed to be integrated with the SAML for exchanging security assertions and providing protocol mechanisms. SAML defines schemas intended for use in requesting/responding with various types of security assertions. The SAML schemas include information needed to identify and validate the contents of the assertions, such as identity of the assertion issuer, the validity period of the assertion, and the digital signature of the assertion. The SAML specification describes how these elements are to be used. SAML-XACML profile defines how to use SAML 2.0 to protect, transport, and request XACML schema instances and other information needed by an XACML implementation. SAML-XACML profile is supported by the popular open-source SAML implementation OpenSAML2.

XACML is structured into three levels of elements: *PolicySet*, *Policy*, and *Rule*. All these elements can contain *Target* elements. *PolicySet* can contain any number of *Policy* elements and *PolicySet* elements. *Policy* element can contain any number of *Rule* elements.

The latest version XACML 3.0 was ratified in 2010.

## 5.2. XACML Architecture Diagram

Figure 5.1 [9] shows a XACML architecture diagram and a SOAP request received and intercepted by the PEP (Policy Enforcement Point). Allow/deny decision includes collaboration of other XACML entities. Response of web service depends on results of XACML policy effect.

In step 1, the PEP intercepts information from incoming SOAP request. In step 2, the PEP constructs SAML authorization decision query. Results of the query determine if the request has granted access to the web service. An authorization decision query includes details of the requested resource. It is sent to the PDP (Policy Decision Point) with details of the requestor identity. This identity information is taken from the web service request. The PDP receives all information it needs to make a decision. The PDP evaluates rule and returns SAML authorization decision assertion if access is granted. The PEP can be a web service that receives request for the resources and connects to the PDP to query authorization decision for resources.

In steps 3 and 5, PDP retrieves policy from the PRP (Policy Retrieval Point). Usually the PRP and the PDP are located on the same machine. The PDP can cache policy information.

If policy information for the resource is not available at the PRP, it can be retrieved from the policy store. The policy store should be able to import and export XACML. Administrator combines rules into policies using the PAP (Policy Administration Point).

In step 6, the PDP sends information request to the PIP (Policy Information Point). The PIP calculates the predicate of the rule and returns the attribute information as a SAML attribute assertion in step 7. Now the PDP has all the information it needs to make a decision. It evaluates the rule and in step 8 the PDP returns a SAML authorization decision assertion to the PEP. In

step 9, the SAML authorization assertion can be inserted into the SOAP message and forwarded to the target web service.

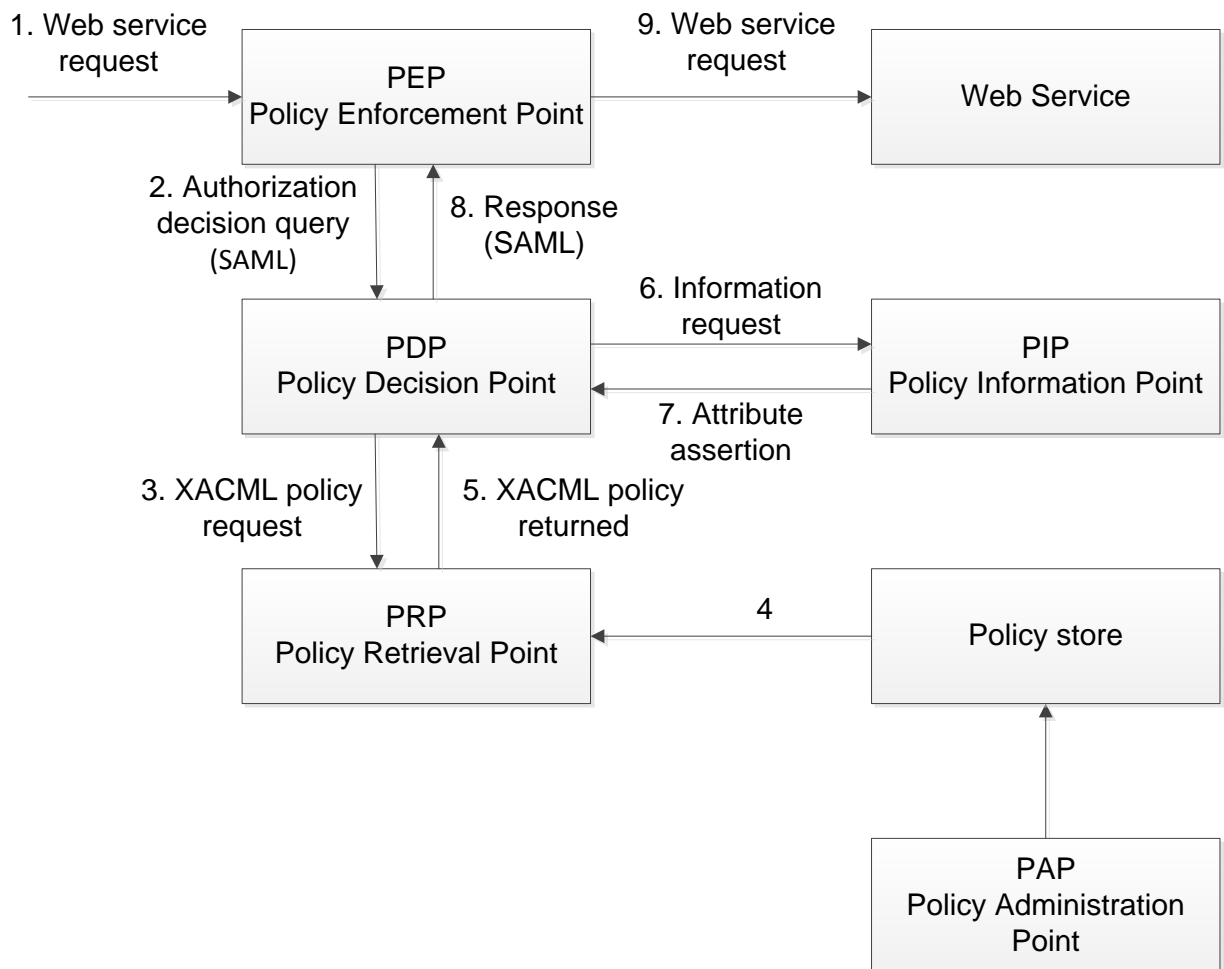


Figure 5.1 XACML architecture [9]

### 5.3. Policy in XACML

Policy is a very important aspect of XACML. Company can use single policy to provide access to different systems. XACML has *PolicyStatement* element to hold information about policy in XML format.

*Statement* element contains the following content: PolicyID, MetaPolicy, Effect, Target, Rules, Rule-combining algorithm, and Obligations. Each policy refers to one or more rules, linked together using a rule-combining algorithm.

Policy Decision Point uses target of policy statement to determine if policy is applicable for request. If target is already identified in rules, there is no need to include it in the policy statement.

```
<Policy PolicyId="1" RuleCombiningAlgId="deny-overrides">
  <Target>
    <Subjects> <AnySubject/> </Subjects>
    <Resources>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">SampleServer</AttributeValue>
        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
    </Resources>
    <Actions> <AnyAction/> </Actions>
  </Target>
  <Rule RuleId="1" Effect="Permit">
    <Target>
      <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
          <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
            AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
          <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
            AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
        </Apply>
      </Condition>
    </Target>
  </Rule>
</Policy>
```



Above is an example of XACML policy [12]. The Target of the example above says that the Policy only applies to requests for the server called "SampleServer". The Policy has a Rule with a Target that requires an action of "login" and a Condition that applies only if the Subject is trying to log in between 9:00 and 17:00.

## 5.4. XACML Rule Examples

The following rule grants read access to patient medical records of the site only if the requestor is the patient.

```
<Rule RuleId="1" Effect="Permit">
  <Target>
    <Actions MatchId="function:subset" DataType="xs:boolean">
      <Attribute DataType="xs:string">read</Attribute>
    </Actions>
  </Target>
  <Condition FunctionId="function:string-equal" DataType="xs:boolean">
    <AttributeDesignator Designator="SubjectId" DataType="xs:string"/>
    <AttributeDesignator Designator="patientName" DataType="xs:string"/>
  </Condition>
</Rule>
```

We see that it has a target, effect, and conditions, which is the definition of a rule in XACML. The effect of rule is “permit”, so the rule is granted, and access is permitted. The other type of effect would be “deny”.

XACML can use mathematic statements for rules processing. XACML includes definition of arithmetical operations (add, subtract, multiply, and divide) on integers, decimal numbers, dates, rounding, greater than/less than, and matching. It allows constructing complex rules.

Here is the example of “Deny” rule. The target of rule is administrator read/write access to patient records. Target includes subject’s role (administrator), resource, and action (read or write):

```
<Attribute DataType="xs:string">administrator</Attribute> </Subjects>

  <Resources MatchId="function:anyURI-equal" DataType="xs:boolean"> <AttributeDesignator
Designator= "//xacmlContext/Request/Resource/@ResourceURI" DataType="xs:anyURI"/>

<Attribute DataType="xs:string">read write</Attribute>
```

Next I will define briefly four separate rules [17]. Appendix D contains full text of the rule examples.

Rule 1. A person can read any record for which he is the designated patient.

It is a simple rule with a single condition. The following fragments show XACML rule that expresses Rule 1.

```
<rule ruleId="1" effect="Permit"
<resources>
  <saml:Attribute AttributeName="documentURI" >
    <saml:AttributeValue>//medico.com/record.*</saml:AttributeValue>
  </saml:Attribute>
</resources>
<actions>
  <saml:Action>read</saml:Action>
</actions>
```

Rule 2. A person can read any record for which he is the designated parent or guardian, and if the patient is under 16 years old.

This rule uses mathematical function, *minus* function to calculate age. It also uses predicate expressions, with *and* and *not* elements.

```
<rule ruleId="2" effect="Permit"
<condition>
  <and>
    <equal>
      <saml:AttributeDesignator AttributeName="requestor" />
      <saml:AttributeDesignator AttributeName="parentGuardianName" />
    </equal>
    <not>
      <greaterOrEqual>
        <saml:Attribute AttributeName="ageOfConsent">
          <saml:AttributeValue>16-0-0</saml:AttributeValue>
        </saml:Attribute>
      </greaterOrEqual>
    </not>
  </and>
</condition>
```

Rule 3. A physician can write any medical element for which he is the designated primary care physician.

```
<subjects>
  <saml:Attribute AttributeName="role">
    <saml:AttributeValue>physician</saml:AttributeValue>
  </saml:Attribute>
</subjects>
<resources>
  <saml:Attribute AttributeName="documentURI" ></saml:Attribute>
</resources>
<actions>
  <saml:Action>write</saml:Action>
</actions>
```

Rule 4. An administrator is not permitted to read or write any medical element.

```
<rule ruleId="4" effect="Deny">
  <target>
    <subjects>
      <saml:Attribute AttributeName="role">
        <saml:AttributeValue>administrator</saml:AttributeValue>
      </saml:Attribute>
    </subjects>
    <actions>
      <saml:Action>read write</saml:Action>
    </actions>
  </target>
</rule>
```

## 6. Windows Identity Foundation

Windows Identity Foundation (WIF) is a set of .NET Framework classes that uses the SAML and XACML concepts. It is a framework for implementing claim-based identity in our applications. When we build claim-aware applications, the user presents his identity to our application as a set of claims. One claim may be the user's name; another may be his email address, age, membership in the Sales role, and so on. An external identity system is configured to give our application everything it needs to know about the user with each request he makes, along with cryptographic guaranty that the identity data we receive comes from a trusted source. Under this model, single sign-on is much easier to achieve, and our application is no longer responsible for authenticating users, storing user accounts and passwords, and calling to enterprise directories to look up user identity details. Our application makes identity-related decisions based on claims supplied by the user.



Figure 6.1 User Presents Claims [18]

Any company that has more than one web application or web service can benefit by starting with a claims-based model for identity. One of the major benefits is to have application developers focus only on application business logic while the identity related requirements can be handled by external issuing authorities like Active Directory Federation Services ADFS 2.0.

In the claim-based identity model our application does not look up user attributes in a directory. Instead, the user delivers claims to our application, and we are going to examine them. Each claim is made by an issuer, and we'll trust the claim only as much as we trust the issuer. The Claim class in WIF has an Issuer property that allows us to find out who issued the claim.

The user delivers a set of claims to our application. In a web service, these claims are carried in the security header of the SOAP envelope. In a browser-based web application, the claims arrive via an HTTP POST from the user's browser, and may later be cached in a cookie if a session is acceptable. Regardless of how claims arrive, they must be serialized somehow, and this is where SAML tokens come in. A SAML token is a serialized set of claims that is digitally signed by the issuing authority. The issuing authority is a web service or web application that knows how to issue security tokens. The signature is important because it gives us assurance that

the user did not just make up a bunch of claims and send them to us. One of the core features in WIF is the ability to create and read security tokens.

A security token service (STS) is the service that builds, signs, and issues security tokens according to the appropriate protocols. There is a lot of work that goes into implementing these protocols, but WIF does all of this for us.

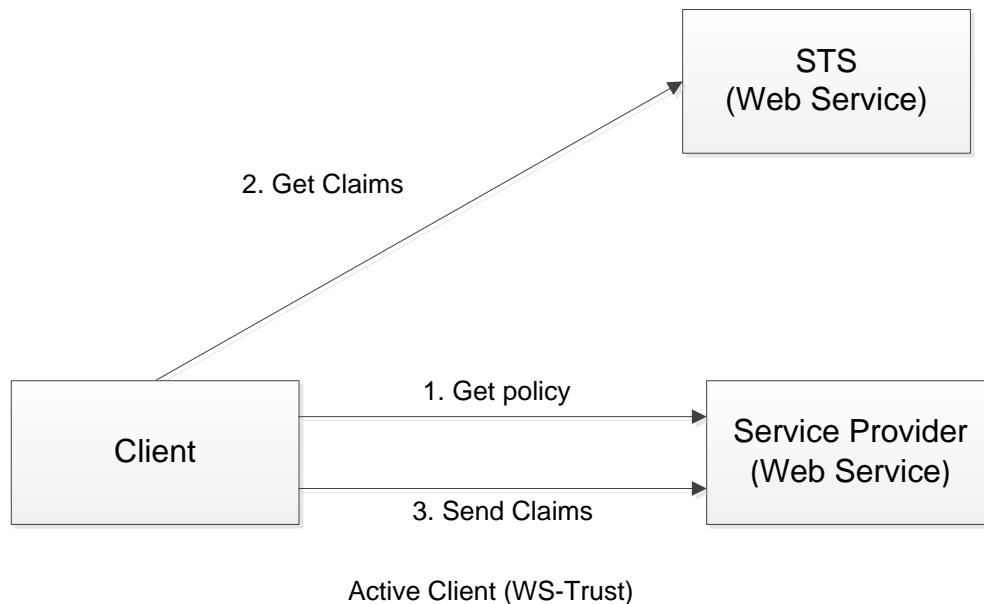


Figure 6.2 Active Basic Scenario with Web Services in WIF [18]

Here is an example of a claims-based system in action. Figure 6.2 shows a claim-aware web service (service provider) and a client that wants to use that service. The service provider exposes policy that describes its addresses, binding, and contracts. But the policy also includes a list of claims that the service provider needs, for example, user name, email address, and role membership. The policy also tells the client the address of the STS where it should retrieve these claims. After retrieving this policy (1), the client now knows where to go to authenticate: the STS. The client makes a web service request (2) to the STS, requesting the claims that the service provider asked for through its policy. The STS authenticates the user and returns a SAML token that gives the service provider all of the claims it needs. The client then makes its request to the service provider (3), sending the SAML token along in the security SOAP header. The service provider now receives claims with each request, and simply rejects any requests that do not include a SAML token from the STS that it trusts.

Policy is retrieved using HTTP GET and the policy itself is structured according to the WS-Policy specification. The STS exposes endpoints that implement the WS-Trust specification, which describes how to request and receive SAML tokens.

Browser-based applications can also participate in the world of claim-based identity. Figure 6.3 shows how this works. The user points his browser at a claim-aware web application (service provider). The web application redirects the browser to the STS so the user can be authenticated. The STS is wrapped by a simple web application that reads the incoming request, authenticates the user via standard HTTP mechanisms, and then creates a SAML token and emits a bit of JavaScript that causes the browser to initiate an HTTP POST that sends the SAML token back to the service provider. The SAML token in the POST body contains the claims that the service provider requested. It is common for the service provider to package the claims into a cookie so that the user does not have to be redirected for each request.

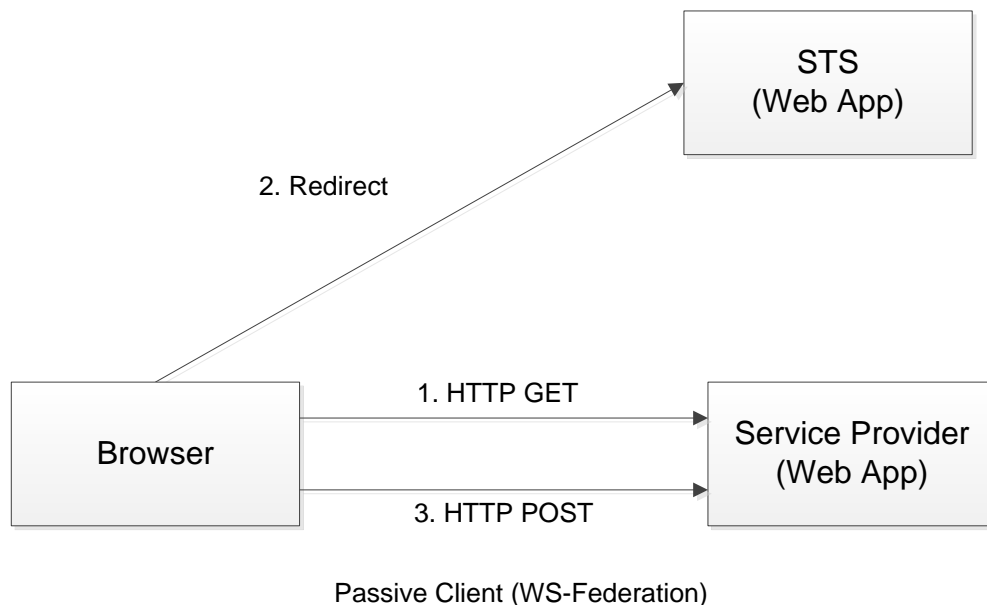


Figure 6.3 Passive Basic Scenario with a Web Browser [18]

## 7. Papers Survey

### 7.1. Web Service Authorization Framework by Thomas Ziebermayr and Stefan Probst [1]

The paper defines requirements for authorization of web services and investigates existing authorization solutions concerning these requirements. Web Service Authorization framework is based on existing authorization solutions and defined requirements. The paper describes concepts and design of proposed framework and overviews selected implementation aspects such as authorization data access and descriptive deployment.

Security model includes the following mechanisms to enforce security policy:

- *Authentication* that establishes identity of one party to another party.
- *Access Control* that determines if user is allowed to access object.
- *Auditing* that gathers data about activity in system and analyzes it to discover security violation.

The paper focuses on web service authorization. There are the following requirements of web service authorization:

- Independent from web service carrier protocol;
- Conforms to web service protocol;
- Service, not object authorization;
- Abstract solution deployable to various web service scenarios.

Administration of authorization data should be done in parallel to the use of web services. Authorization used for web services should be applicable to other similar web services. Thus, there are additional requirements:

- Fine grained authorization at parameter level;
- Efficient access to authorization data;
- Descriptive tailoring of authorization;
- Easy use and deployment;
- Applicable to existing web service frameworks;
- Multi user access to authorization data.

Authorization is giving the subject the right to access object. This decision is based on rules describing who is allowed to access object.

Authorization and Access Control	
Authorization (Password, Challenge-Response, Biometrics, Kerberos)	Auditing
Communication Security (VPN, SSL/TLS, Firewalls)	
Cryptography (Hashing, Cryptography, Digital Signatures, Certificates)	

Figure 7.1 Levels of security mechanisms

Figure 7.1 from [1] shows that security consists of several levels. Low-level security addresses the secure transmission of data via an untrusted network, using communication security mechanisms (VPN, IPsec, SSL/TLS, S/MIME, firewalls) and cryptography (hashing, cryptography, digital signatures, certificates). High-level security protects the application itself by defining a security model. A security model includes the following mechanisms to provide required security policy: authentication, access control, and auditing. Authentication establishes the identity of the party. Access control determines whether a user is allowed to access an object or not. Auditing gathers data about the activity in the system and analyzes it to discover security violations.



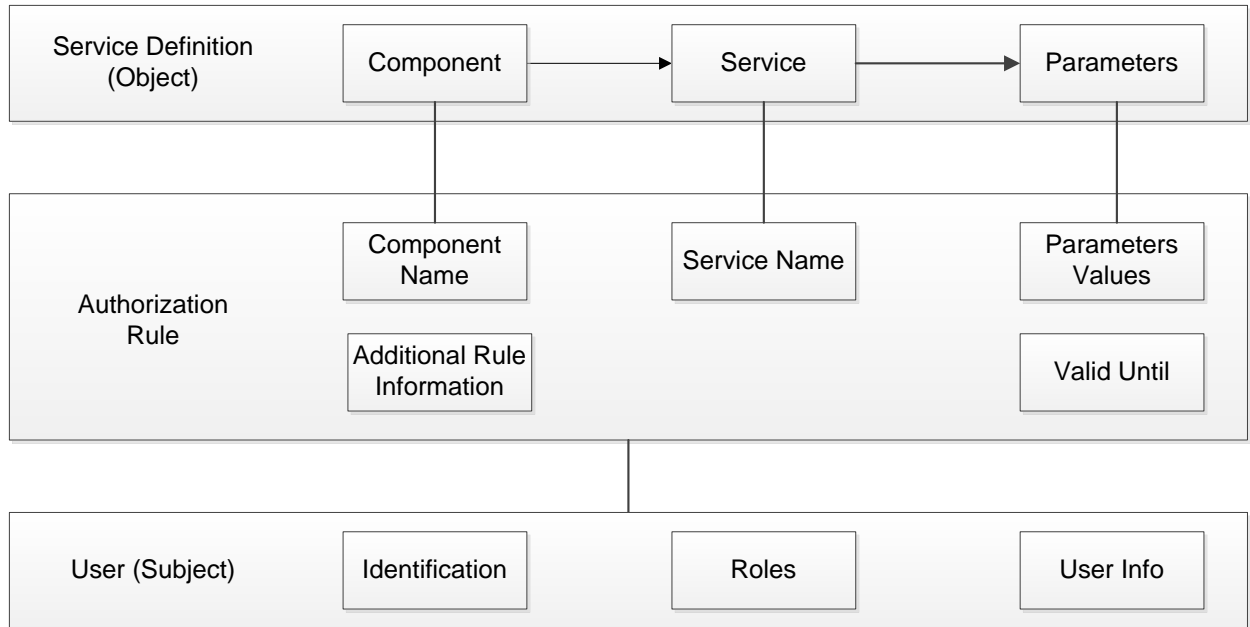


Figure 7.2 User-service relations

Figure 7.2 from [1] shows how authorization rule connects subject and object. A service description consists of a component name that offers services, a service name and the names of the parameters of the service. This simple definition is sufficient for the definition of the service and matches an actual service call to the description in the rule base. The user has identification, roles, and user info. It is necessary to define connections between objects and subjects. The connections define who is allowed to use each service. An access control rule consists of a reference to a service definition, a reference to a user and additional rule information. Additionally for all parameters that should be used in access control, a value (or value range) is defined. Additionally the rule contains metadata for access control like a “valid until” date.

Access control rules include reference to service definition, reference to the user and additional information. ID of subject (user) defines connection from subject to the rule. For example, subject uses service to retrieve information about weather. This user is already registered at service provider for using this service. User pays only for European weather service, not worldwide. So, access control needs to check if user is allowed access to service and region.

When user sends service request, access control searches for user and his rights. First of all, user must exist in the rule base. If the user does not exist, authentication will not succeed, and system error occurs. First, authorization service searches for exact match of service request to service rights of user. When user is allowed to use service, access control checks if service is protected by parameter restrictions. If the user has rights to use service without restrictions, access control task is finished and access allowed. If the user has parameter restrictions, access

control compares parameter of service request with defined restrictions. If parameter check succeeds, service call is allowed. Figure 7.3 shows order of rules of secured application.

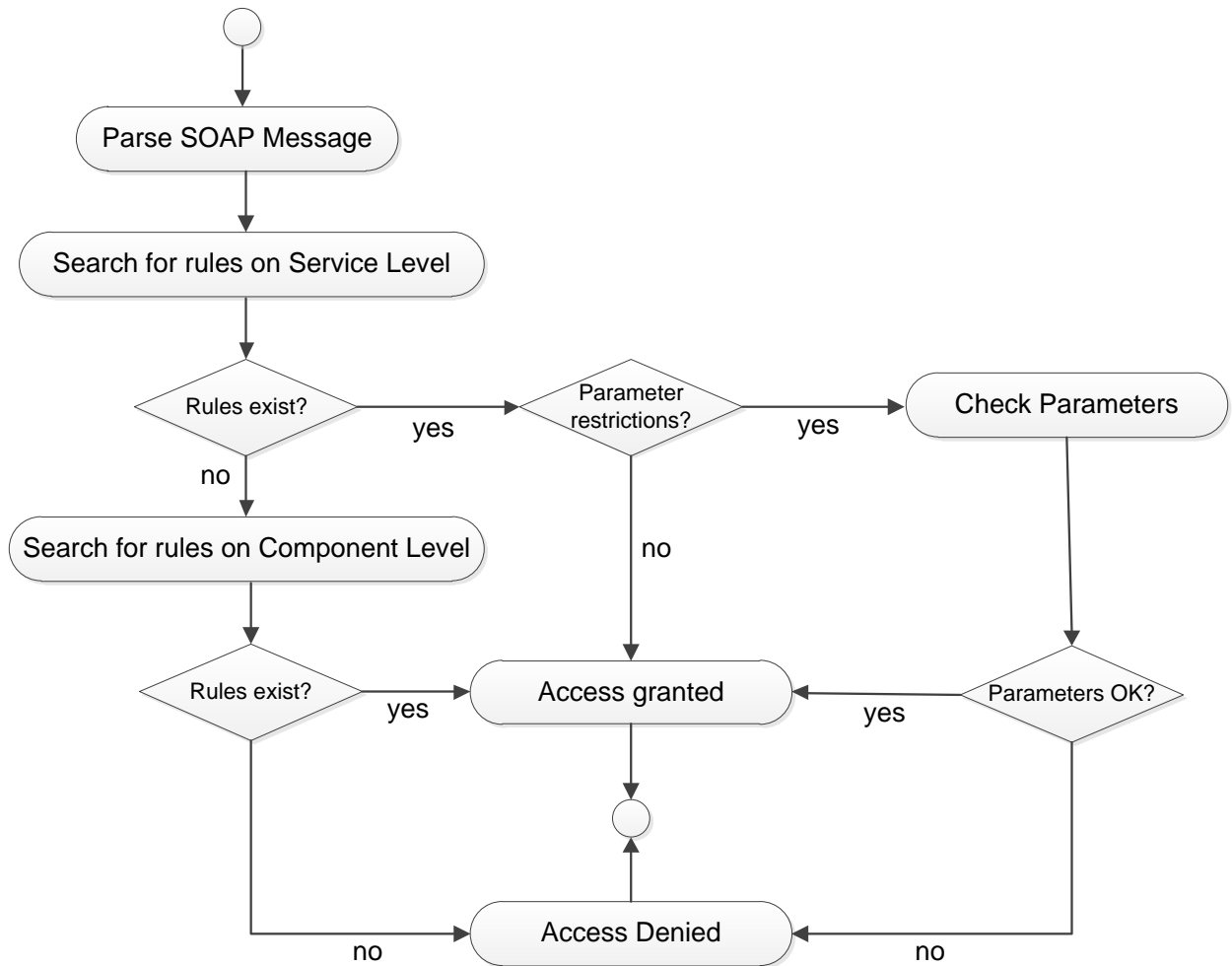


Figure 7.3 Authorization activities.

If there is no rule found for user, access control logic searches rule base for matches at component level. If no rule can be found for user that matches requested service at component level, access is not allowed. Granularity of rule defines which services are allowed to access and if it is restricted by parameter definitions. Usually, rules are defined at service level. Authorization rules at component level are high-level rules that allow a lot of rights to the user.

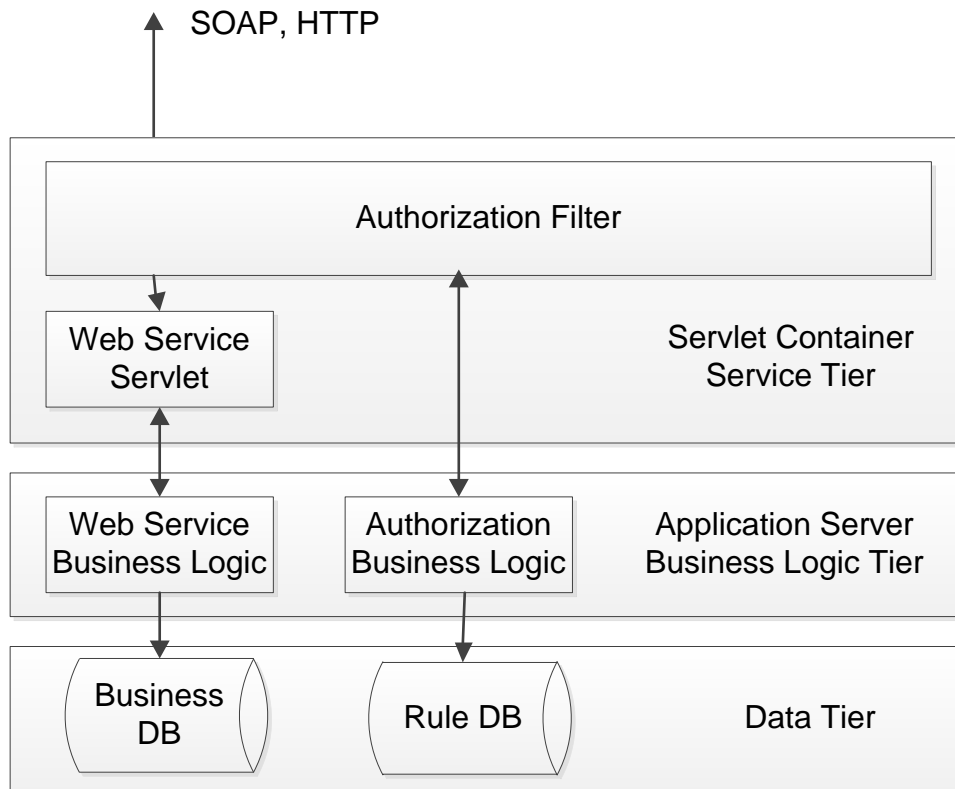


Figure 7.4 Framework architecture

The paper introduces WSAF (Web Service Authorization Framework). The framework is based on Java Web Technology (Servlet). Web service is published using Apache Axis framework. The framework follows n-tier architecture (Figure 7.4). Framework data (service definition, user definition, access rules) is stored in database layer (relational database). The logic for interpretation of authorization rules is implemented in the logic layer. This logic decides if user is allowed to call service or not. Communication between authorization filter and authorization logic is not web service based.

Servlet filter implements authorization client. Servlet parses web service request and uses authorization components to decide if access is allowed or not allowed. If user is not authorized, request is rejected. If user is authorized, call is forwarded to web service. Information in SOAP request helps to make access control decision. SOAP request does not contain information about component, which there is a need to authorize.

We will consider the technologies that solve web service authorization problem.

XACML (Extensible Access Control Markup Language) is an OASIS standard that describes a policy language and provides access control decision service interface. Although XACML does

provide a standardized way to describe access policies and standardized interface to decision service, it does not fit the requirements for authorization that the authors define in the article. It is necessary to provide additional environment that supports transactional access from multiple users and provides efficient access to the rule data. The important feature of XACML is the possibility to distribute the access control decision services. The definition of the access control decision service as web service leads to the question how to secure his service. XACML tries to provide a general solution to web service access control, and it leads to complex solution.

Java provides different libraries addressing security issues. Java has support for server applications by the Java Enterprise Edition (J2EE). J2EE security does not provide appropriate mechanisms to define fine granular access control. For example, J2EE only allows the assignment of access rules to a role on method level. Thus access restrictions of a single user at parameter level cannot be defined with J2EE security mechanisms. The authorization itself must be done programmatically, so the code must be changed every time the security policy changes. It has negative impact on reusability.

Microsoft's .NET platform provides a wide variety of security features, which enable the implementation and integration of high-level security models into applications. Unfortunately, .NET framework does not provide declarative security, which means that security must be addressed directly in the code by providing appropriate statements. The .NET uses a role-based access control model, allowing to state that specific code pieces can only be executed by specific roles. Nevertheless, this model can be extended to provide a finer granularity in access control. It requires some additional efforts from the programmer.

Many technologies are available for security of distributed applications. The paper presents simple solution for web service authorization. The framework does not address all existing web service scenarios, but the most common. The paper introduces implementation of web service authorization framework and describes experiences with implementation.

WSAF has many benefits. It is independent from web service carrier protocol, but it is easy to provide additional authorization clients for other protocols. WSAF uses abstract service and user definition, so it is applicable to other web service scenarios. The framework data (service definition, user definition, access rules) is stored in relational database. WSAF bases on proved database technology, it allows efficient data access and ensures transactional security. WSAF is applicable to existing web service frameworks.

## **7.2. Trust Management for Web Services by Weiliang Zhao and Vijay Varadharajan [2]**

The paper proposes complex trust management approach for web services. This approach covers analysis and modeling of trust relationships and development of trust management layer. The article introduces separated trust management layer for web services that holds computing components for trust management tasks.

The paper provides high level overview about taxonomy framework of trust. Taxonomy framework of trust provides technologies and enables tools for analysis and modeling of trust relationships in distributed information systems. Taxonomy framework of trust is based on formal definition of trust relationships.

The authors consider some generic requirements for trust relationships with web services. There are some specific trust relationships that are related to access control of web services or web service methods. The web services or methods are the resources to be protected. There are the following problems in the access control for web services. Firstly, the trustor set is normally modeled as a set of web services or a set of web service methods. Secondly, the trustee set includes the entities that request the web services or web service methods in the trustor set. Thirdly, the condition set may require the trust mechanisms such as credentials, reputation, data storage, and environment parameters. Fourthly, the property set may include the basic operations on web services or web services methods such as (a) execute: access and execute the web services or web service methods; (b) update: update web services or web service methods; (c) find: provide search capabilities for properties and metadata associated with web services or web service methods.

Security tokens in SOAP messages defined in WS-Trust provide evidences for some conditions in the condition set. WS-Trust is a standard trust mechanism at messaging level.

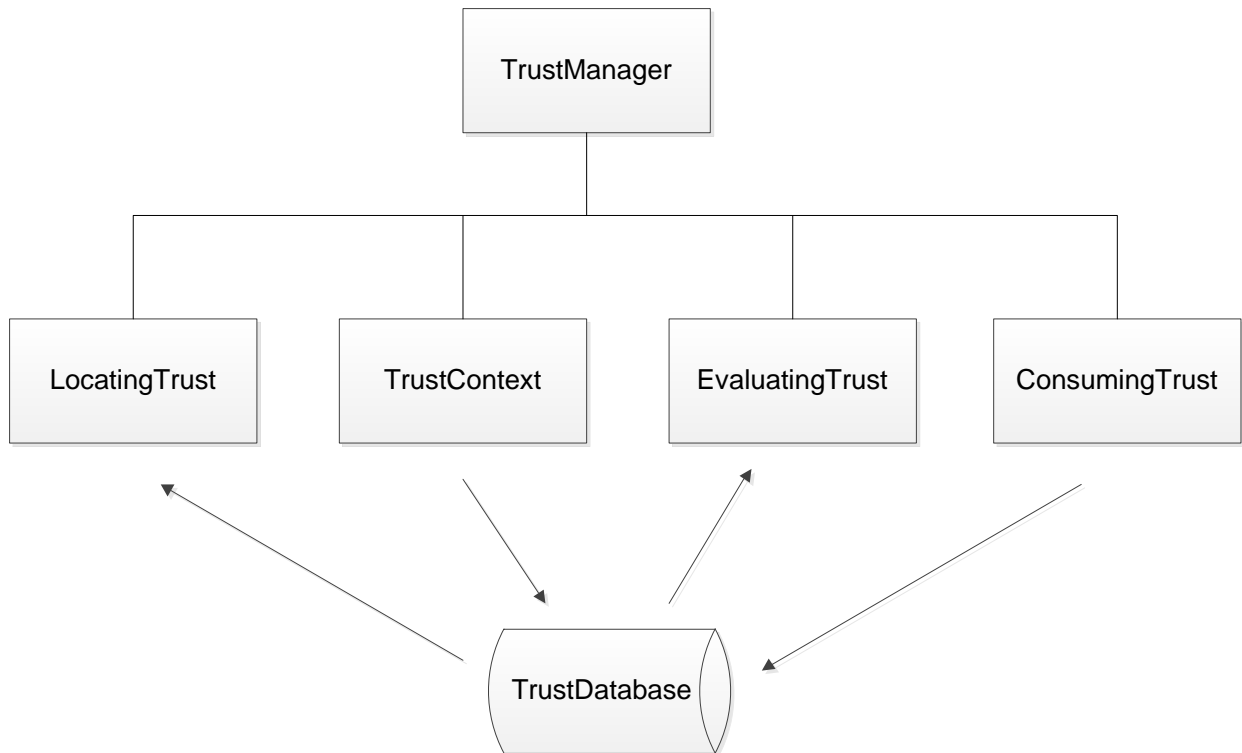


Figure 7.5 Trust Management architecture for web services

The architecture contains the following components:

- *TrustDatabase* that provides permanent storage mechanism for storing and retrieving trust related information including trust relationships and trust parameters for web services.
- *TrustManager* that manages trust management layer for web services.
- *LocatingTrust* that is a package for finding trust relationship and associated processing information based on request.
- *TrustContext* that contains computing components for management and processing of contextual information related to trust management of web services.
- *EvaluatingTrust* that contains computing components required for evaluation of trust relationships.
- *ConsumingTrust* that contains computing components for control and management of trust consuming.

The package TrustContext has the following components:

- *TrustContext Controller* that is a manager that assigns tasks to “WS-Trust Message Handler” and “Context Information Handler”.
- *WS-Trust Message Handler* that adapts incoming WS-Trust SOAP messages and generates outgoing WS-Trust SOAP messages.
- *Context Information Handler* that takes care of management of trust-relevant contextual information for trust evaluation and trust consuming.

The EvaluatingTrust has the following components:

- *Trust Evaluation Controller* that is the computing component that assigns evaluation tasks to other functional components in the package.
- *Credential Evaluation*.
- *Reputation Evaluation*.
- *Stored Data Evaluation*.
- *Environment Evaluation*.

These components should be the implementation of corresponding generic components of general trust management architecture for distributed information systems.

The ConsumingTrust has the following components:

- *Consuming Controller*, which is the manager of trust consuming.
- *Direct Trust Consuming Controller*.
- *Credential Generator Consuming*.
- *Data Storage Consuming*.
- *System Consuming*.

These components are the implementation of corresponding generic components of general trust management architecture for distributed information systems. For “Direct Trust Consuming Controller” and “Credential Generator Consuming”, it is possible to require “WS-Trust Message Handler” and “Context Information Handler” for cooperation in producing/delivering messages to the involved parties.

The handling of WS-Trust SOAP messages that include security tokens and other trust-relevant information is a specific task in web services paradigm. “WS-Trust Message Handler” is the component that adapts the incoming WS-Trust SOAP messages and generates the outgoing WS-Trust SOAP messages.

Web services coordination layer is on the top of the trust management layer. The trust management layer serves WS-Coordination and WS-Transaction layers.

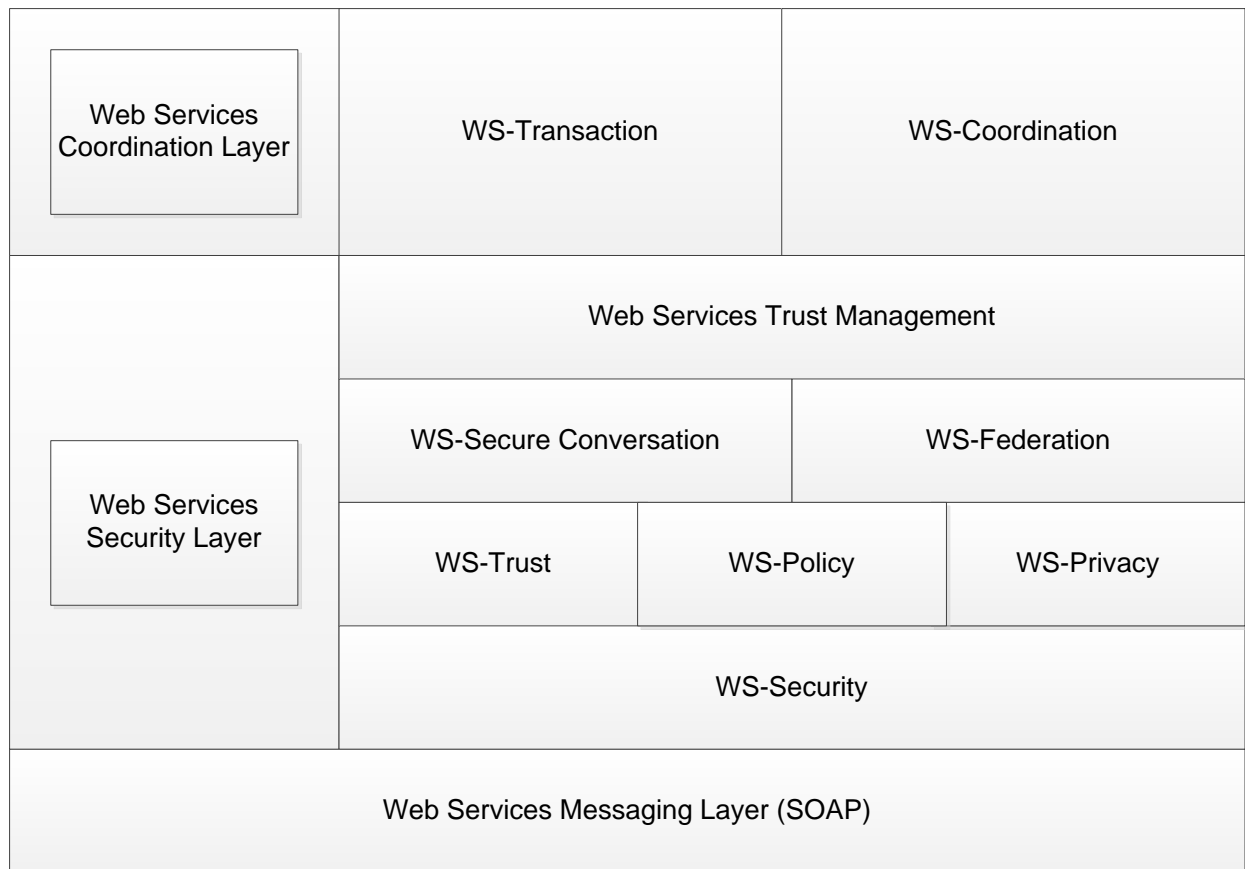


Figure 7.6 Web services architecture layers

The paper proposes trust management architecture for web services for creating trust management layer. This trust management architecture deals with trust relationships, trust evaluations and trust consumption in web services.

The proposed architecture provides high level design. It can be used for development of trust management layer for web services, including specification of requirements, design, deployment, and maintenance. Trust management layer for web services provides facilities for all trust management tasks related to trust relationships.

The paper presents complex trust management approach for web services that deals with analysis and modeling of trust relationships and development of trust management layer. It provides starting point and useful tools for development of trust management solution. The trust management layer extends WS-Trust and covers existing trust mechanisms including credentials, reputation, data storage, and environment parameters.



### **7.3. Definition and Implementation of a SAML-XACML Profile for Authorization Interoperability across Grid Middleware in OSG and EGEE by Gabriele Garzoglio, Ian Alderman, Mine Altunay, and Rachana Ananthakrishnan [3]**

The paper describes OSG and EGEE security models. OSG (Open Science Grid) and EGEE (Enabling Grids for E-scienceE) are projects that promote science via use of distributing Grid computing. These projects are independent and operate within hardware resources in different countries. Both Grids extended infrastructures to include role-based access to resources based on VO (Virtual Organization) membership of the user.

Paper summarizes principal elements of common Authorization Interoperability profile. Authorization Interoperability activity was formed in 2006. Collaboration defined common protocol for use by OSG/EGEE and identity attribute profile for authorization to policy decision services.

VOs (Virtual Organizations) are resources that are available on Grid for user communities. Access to resources is granted to users on basis of their membership in VOs. Group members have special roles for their group. Virtual Organization Management (VOMS) manages and publishes this structure and relative membership of each member.

Each Grid organization makes available to its member sites lists of member VOs and VO preferred resource access policies. Users interact with resource gateways on behalf of VOs and VO groups with certain group role. Every user is responsible for including information with his credentials.

Authorization Interoperability activity is focused on standardizing protocol for PEP to PDP communication. Authorization is based on user X509 identity attributes and VO membership attributes.

SAML profile of XACML transfers detailed request-reply message information about subject, resource and action from PEP (service gateway) to PDP (centralized authorization service) and transfers back authorization decision. Request is a message that is sent from PEP to PDP. Response is a message going from PDP to PEP. PEP sends XACML request with authorization query to PDP. Query contains user or service allowed to execute “action” on “resource” controlled by PEP. After processing XACML request of PEP for access authorization, PDP returns XACML response to PEP.

Request contains four attribute sections or contexts:

- *Subject*. Subject declares entity for which authorization decision is requested. The attributes in Subject help to make authorization decision.
- *Resource*. The attributes in Resource context describe resource targeted for authorization request.
- *Action*. The attributes in Action context describe action that subject wants to perform on specified resource.
- *Environment*. The attributes in Environment context transfer additional parameters in authorization request of subject to perform action on specified resource.

Authorization Decision Statement is the main element of XACML response that contains actual decision. It can be “Permit”, “Deny”, “Indeterminate”, or “Not Applicable”. PDP can return “Permit” with conditions under which access will be granted.

Authorization Interoperability activity is a set of libraries that implement Authorization Interoperability protocol. These libraries are used for implementations of PEPs and PDPs in OSG and EGEE. Authorization query is SAML/XACML messages that are sent as SOAP messages over TLS transportation layer. This protocol is implemented as web service interface.

The main goal of Authorization Interoperability activity is to ensure interoperability between middleware and authorization infrastructures used in OSG and EGEE projects. Both Grids have a common security model, so users are pushed to resources identity and attributes assertions, based on X509 certificates and VOMS attribute certificates. Authorization interoperability is achieved by defining common authorization decision query protocol with associated profile.

The main contribution of this paper is in presenting a summary of the agreed-upon protocol, profile and the software components involved. The authors use only standard authorization query interface as specified by the SAML-XACML profile, and don’t mandate any use of XACML-compliant policy evaluators at the PDPs. Despite the commonality of the security model, the Authorization Interoperability activity is a fundamental step to allow the deployment of resource gateway implementations on OSG or EGEE, without the need for Grid-specific authorization plug-ins. The common protocol allows Grid developer groups associated with OSG or EGEE to reuse a common implementation of the security call-out libraries, thus reducing maintenance and eliminating duplication of work. The definition of the Authorization Interoperability protocol is a significant step forward for OSG and EGEE, as it enables better interoperability of services as well as providing software reuse opportunities across the projects. The results of authors’ activity are generally available via the Internet2 and Globus open source projects.

#### 7.4. Access Control Policies for Web Services in Medical Aid System by Li-Qun Kuang, Yuan Zhang, and Xie Han [4]

The article introduces basic concepts for securing web services and proposes certain system architecture. XML based web services are used in hospital information systems and medical aid systems. The article describes automation of access control for each participant in medical aid system. The paper provides security for cooperative business processes using web services in the open environment.

WS-Policy (web services policy) is an extension of WS-Security specification. WS-Policy describes how sender and receiver specify their requirements and abilities. WS-Policy provides mechanisms that help web service application to specify policy information. WS-Policy specification defines the following:

- XML Infoset (policy expression) that contains domain-specific web service policy information.
- Set of constructs to indicate how choices and combination of domain-specific policy assertions apply in web services environment.

The architecture for enforcing access control policies is shown in Figure 7.7.

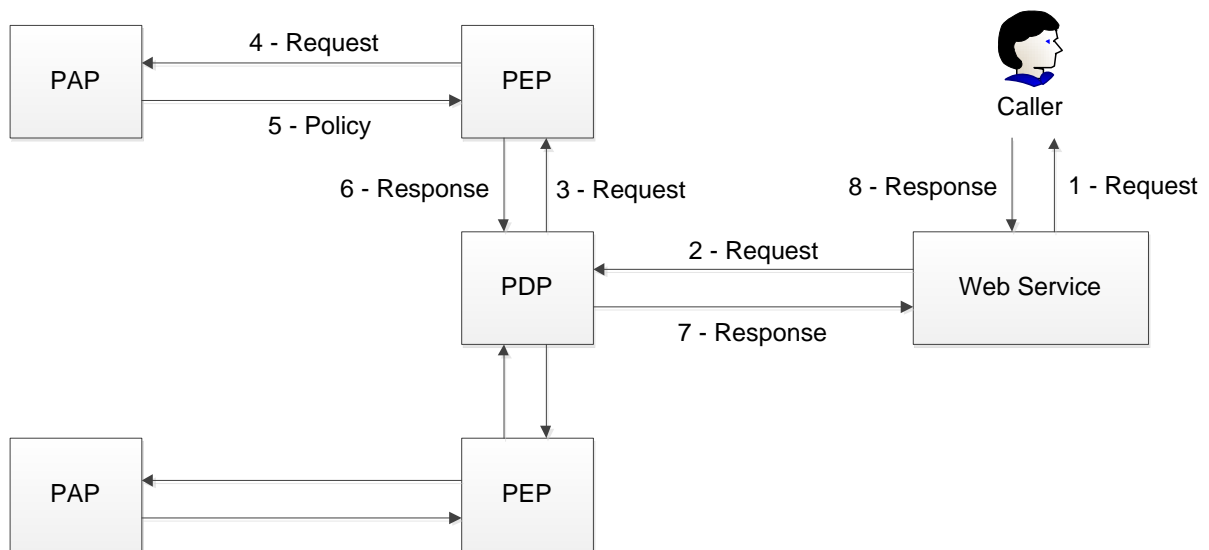


Figure 7.7 WS-Policy system architecture

The architecture contains the following main components:

- Web service. Web service enables each participant to provide access to his internal functionality and data to other participants.
- PDP (Policy decision point). PDP receives caller's identity, attributes and web service call details from PEP. Then PDP evaluates stored policies and returns grant or deny decision to PEP.
- PEP (Policy enforcement point). PEP passes requests to access protected resources to PDP. PDP consults relevant access control policy and returns decision to PEP. PEP uses this decision; it allows traffic to pass through PEP or stops traffic at that point.
- PAP (Policy administration point). PAP is a policy repository that allows inserting, updating and deleting policies. It prevents inserting invalid policies and executing updates that generate invalid policies.

If PAP requires authentication of PEP module, PAP creates new PDP and at least a pair of PEP-PAP.

PDP-PEP-PAP architecture has several advantages:

- Passwords and policies don't need to be stored on DMZ servers that are untrusted. DMZ is a network that exposes external service to untrusted network (internet).
- Policy changes made at PAP are immediately used by PEP.

The main contribution of this paper is in introducing the basic concepts for securing web services and proposing the system architecture for medical web services. The paper presents a using of web services in hospital information systems and medical aid systems. The authors show us the importance of access control for each participant in the medical aid systems, such that a sending participant can verify whether a requesting participant has the privilege to use a web service. The paper describes the architecture for enforcing access control policies. One of the most important features of this architecture is that it can prevent from inserting invalid policies and from executing update operations that generate invalid policies. The paper introduces medical services domain and defines requirements for access control of designing medical web services.

## **7.5. Short Papers Survey**

This section contains a brief review of some additional relevant papers.

### **7.5.1. Specification and Verification of Authorization Policies for Web Services Composition by Mohsen Rouached and Claude Godart [5]**

The article proposes logic based approach using specifying authorization policies in web services composition. The paper presents how to specify authorization policies using EC (event calculus logic). Authorization policies should be defined in flexible language to allow specification of conditions that will include multiple triggering events. EC language is good for it. The authors adapt simple classical logic form of EC. EC language consists of set of time-points, set of time-varying properties, and set of event types or actions.

The paper presents a new framework for managing authorization policies for web services compositions. It describes use of EC logic for developing language that supports specification and analysis of authorization policies for web service composition. The paper shows two scenarios for enforcement model. The described formal policy model considers that web service is both object and subject. The authors use functions as parameters in basic predicate symbols of EC. They define these functions as events that can occur during composition execution. The framework supports compositions of web services that are compatible with authorization requirements.

### **7.5.2. Towards Secure Execution Orders for Composite Web Services by Joachim Bickup, Barbara Carminati, Elena Ferrari, Frank Muller, and Sandra Wortmann [6]**

The article proposes framework for decentralized execution of composite web services that can ensure correctness and security of execution. BPEL (Business Process Execution Language) is OASIS standard executable language for specifying actions within business processes with web services. Paper describes four main phases of secure execution order: offer phase, location phase, container generation phase, and execution phase.

The framework relies on data structure, called *container*. Container stores information about web service composition that needs to be executed, and user credentials. Container is passed among participating web services.

The paper presents decentralized mechanism and framework that ensures secure execution of composite web services. Proposed framework delegates execution to participating web services as much as possible. At the same time framework ensures correctness of control flow and main security requirements. The main benefit of proposed solution is that this solution does not need a dedicated web service for WS-BPEL, but at the same time provides security requirements for composition execution.

### **7.5.3. Extending XACML Authorization Model to Support Policy Obligations Handling in Distributed Application by Yuri Demchenko and Cees de Laat [7]**

The article summarizes recent developments and discussions in networking security community to build scalable authorization infrastructure for distributed applications. The paper analyses basic use cases for obligations in computer Grids and on-demand network resource provisioning. Paper refers to implementations of policy obligations interoperability and handling framework. The paper provides short overview of policy obligations definition in XACML. It summarizes common requirements to AuthZ service infrastructure to support multidomain CRP.

The paper introduces OHRM (Obligation Handling Reference Model), that extends XACML and generic authorization models to support obligations handling in distributed AuthZ infrastructure. OHRM extends XACML and generic authorization models to support obligations handling in the distributed AuthZ infrastructure. Obligations are included into the policy definition and returned by PDP to PEP which in its turn should take actions as prescribed in the obligation instructions or statements.

The paper provides implementation suggestions for OHRM as component of AuthZ infrastructure in Grid based applications.

The paper provides basis for interoperability and further discussion on general obligations definition and handling framework. It fills the gap between general concept and many implementation details that play important role in building AuthZ (authorization) infrastructure solutions.

#### **7.5.4. Authorization and Privacy for Semantic Web Services by Lalana Kagal, Tim Finin, Massimo Paolucci, Naveen Srinivasan, Katia Sycara, and Grit Denker [8]**

The Web Ontology Language (OWL) is a family of knowledge representation languages or ontology languages for authoring ontologies or knowledge bases. The languages are characterized by formal semantics and XML-based serializations. OWL is endorsed by the World Wide Web Consortium (W3C) and has attracted academic, medical and commercial interest.

OWL-S is an ontology built on top of OWL, for describing web services. OWL-S describes web services with the help of three methods: a profile that provides a general description of the web service, a process model that describes how the web service performs its tasks, and the grounding that specifies how the atomic processes in the process model map onto WSDL (Web Services Description Language) representations.

The article proposes security markup of service parameters in OWL-S. It provides security and policy annotations for OWL-S service descriptions and enforcements. The paper extends OWL-S Matchmaker for policy matching and OWL-S Virtual Machine with policy enforcement and security mechanisms. The paper addresses two types of policies: privacy and authorization.

The paper uses Rei, language for policy specification. It's a Prolog-like language. Rei provides metapolicy prioritization mechanism to resolve policy conflicts. For example, Rei allows defining authorization policy such as "Permit everyone access service who is in the same group as provider of this service."

The paper extends OWL-S VM (Virtual Machine) to enforce authorization and privacy policies. The authors implement required security transformation on input/output parameters in OWL-S VM. They use SOAP security annotations to implement message encryption or signing.

The additional references contain technical information which was used by us and is not further discussed here.

## 8. The Application

SAML tokens are standards-based XML tokens that are used to exchange security information, including attribute statements, authentication decision statements, and authorization decision statements. SAML tokens are also extendible; we can extend the schema of the token to meet additional requirements. SAML tokens are important for web service security because they provide cross-platform interoperability and a means of exchanging security information between clients and services that don't reside within a single security document.

SAML tokens are XML representations of claims. SAML tokens carry statements that are sets of claims made by one entity about another entity. Claims can be anything you can imagine, but there are some very common claims offered by most issuers. They are simple, commonly available pieces of information, such as first name, last name, email, groups, roles, and so on. Each issuer can be configured to offer different claims.

For example, in federated security scenarios, the statements are made by a security token service about certain user in the system. The security token service signs the SAML token to indicate the reliability of the statements contained in the token. Moreover, the SAML token is associated with cryptographic key material that the user of the SAML token proves knowledge of. This proof satisfies the service provider that the SAML token was issued to that user.

Below is the typical scenario [16]:

1. A client requests a SAML token from a security token service, authenticating to that security token service by using Windows credentials.
2. The security token service sends a SAML token to the client. The SAML token is signed with a certificate associated with the security token service and contains a proof key encrypted for the target service.
3. The client also receives a copy of the proof key. Then the client presents the SAML token to the service provider and signs the message with that proof key.
4. The signature over the SAML token tells the service provider that the security token service issued the token. The message signature created with the proof key tells the application service that the token was sent to the client.

The main downside of using SAML tokens with WS-Trust infrastructure is that this step forces to go in the opposite direction of embracing a technology such as HTTP, which finds passport in all platforms. SAML is a standard and so is WS-Trust, which is the typical protocol, used for requesting SAML tokens. Forcing clients to talk to WS-Trust endpoints to get a SAML token reduces the reach of the web API. A client that does not have the ability to talk WS-Trust protocol will not be able to use that web API. Reusing the existing WS-Trust endpoint is a main motivation for using SAML tokens. Also, SAML is XML; hence SAML tokens tend to get heavier.



The main advantage of using SAML token to secure ASP.NET Web API is that it provides opportunity to reuse existing token issuance infrastructure. ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. With Web API content negotiation, we can return data based on the client requests. What I mean is, if the client is requesting the data to be returned as JSON or XML, the Web API framework deals with the request type and returns the data appropriately based on the media type. By default, Web API provides JSON and XML based responses.

SAML tokens play a major role in identity management as they are the device of choice for authenticating and authorizing a user's identity or "digital identity". A Security Token Service (STS) is a web service that issues security tokens. The concept of STS is defined in a web service specification called WS-Trust, which specifies how a security token must be requested and issued. A SAML assertion in WS-Trust is a kind of security token. Windows Identity Foundation (WIF), a framework from Microsoft presents a nice API surface to build an STS.

WIF is Microsoft software framework for building identity-aware applications. It provides APIs for building ASP.NET and WCF based security token services as well as tools for building claim-aware and federation capable applications. WIF classes support getting SAML token through WS-Trust, parsing and validating SAML tokens, all out of the box. WIF classes have been absorbed into the core namespaces. WIF is the part of Microsoft .NET Framework 4.5.1.

The solution uses brokered authentication with a SAML token issued by a Security Token Service (STS). The STS is trusted by both the client and the web service (service provider) to provide interoperable SAML tokens.

The client sends an authentication request, with accompanying credentials, to the STS. The STS verifies the credentials presented by the client, and then in response, it issues a SAML token that provides proof that the client authenticated with the STS. The client presents the SAML token to the service provider, which proves that the client has successfully authenticated with the STS.

The protocol used for issuing SAML tokens is based on WS-Trust. WS-Trust is a web service specification that builds on WS-Security. It describes a protocol used for issuance, exchange, and validation of security tokens. In WS-Trust, the type of message sent to an STS to request issuance of a security token is known as a Request Security Token (RST) message. The RST message contains credentials for the client to be authenticated, such as the user ID and password contained in a UsernameToken. The response message from the STS is known as a Request Security Token Response (RSTR) message. The RSTR contains a SAML token.

The solution includes the following applications:

1. Client. Client Web Application **SamlWebClient** accesses the web service. The client provides the credentials for authentication during the request to the web service. I built a

client web application that makes RST request to STS and gets SAML token from RSTR response. I put SAML token as base64-encoded string in Authorization request header, using custom scheme “Saml”. I use proof key to sign SAML token and send signature separately in another header **X-ProofSignature**. After that, **SamlWebClient** calls **SamlWebApi** application. I use HttpClient to make HTTP GET to ASP.NET Web Api application **SamlWebApi**.

2. STS. The application **SamlSts** is the web service that authenticates clients by validating credentials that are provided by a client. The STS issues to a client a SAML token for a successfully authenticated client. **SamlSts** is a custom STS application with WS-Trust endpoint that creates SAML tokens. I get SAML token from STS and use it from ASP.NET Web API application.
3. Service provider. ASP.NET Web API application **SamlWebApi** is the web service that requires authentication of a client prior to authorizing the client. I use message handler to read, validate token, extract claims and use the same message handler to build principal. If there is no authorization header or the scheme does not match name “Saml”, I don’t send unauthorized response. I specify X.509 certificate used by STS as encrypting credentials, so token handler can decrypt it correctly. I use certificate CN=PR. On computer running STS, this certificate contains only the public key. On computer running Web API, there is a certificate with the private key. To check token ownership using proof key, I compute HMAX-SHA256 and compare HMAC computed with HMAX that client sends. If they match, the client is the owner of the token.

For development I used .NET Framework 4.5.1 in Visual Studio 2013.

As illustrated in Figure 8.1, the following steps describe STS token issuance and request/response process:

1. The web client initializes and sends authentication request to the STS. The authentication request to the STS is in form of an RST message. The RST message contains a SAML token that holds the client credentials, which are required to authenticate the client. Claims in the client’s credentials, such a password, may be sensitive in nature, so it’s very important to secure the RST. The client and STS use X.509 certificates to sign and encrypt messages sent between them.
2. The STS validates the client’s credentials. After the STS determines that the client’s credentials are valid, it may also decide whether to issue a SAML token to the authenticated client.
3. The STS issues a security token to the client. If the client’s credentials are successfully validated, the STS issues a SAML token in an RSTR message to the client. The SAML token contains claims related to the client. The security token is signed by the STS; the service can confirm that the token was issued by the STS and that the SAML token was not tampered with after it was issued.

4. The client initializes and sends a request message to the service provider. A request message includes the issued token.
5. The service provider validates the SAML token and processes the request. The SAML token is validated by the service provider to verify that the token was issued by the trusted STS and that the token was not tampered with after it was issued. After the token is validated by the service, it is used to establish security context for the client, so the service can make authorization decisions or audit activity.
6. The service provider initializes and sends a response message to the client. The response message contains certain flights information.

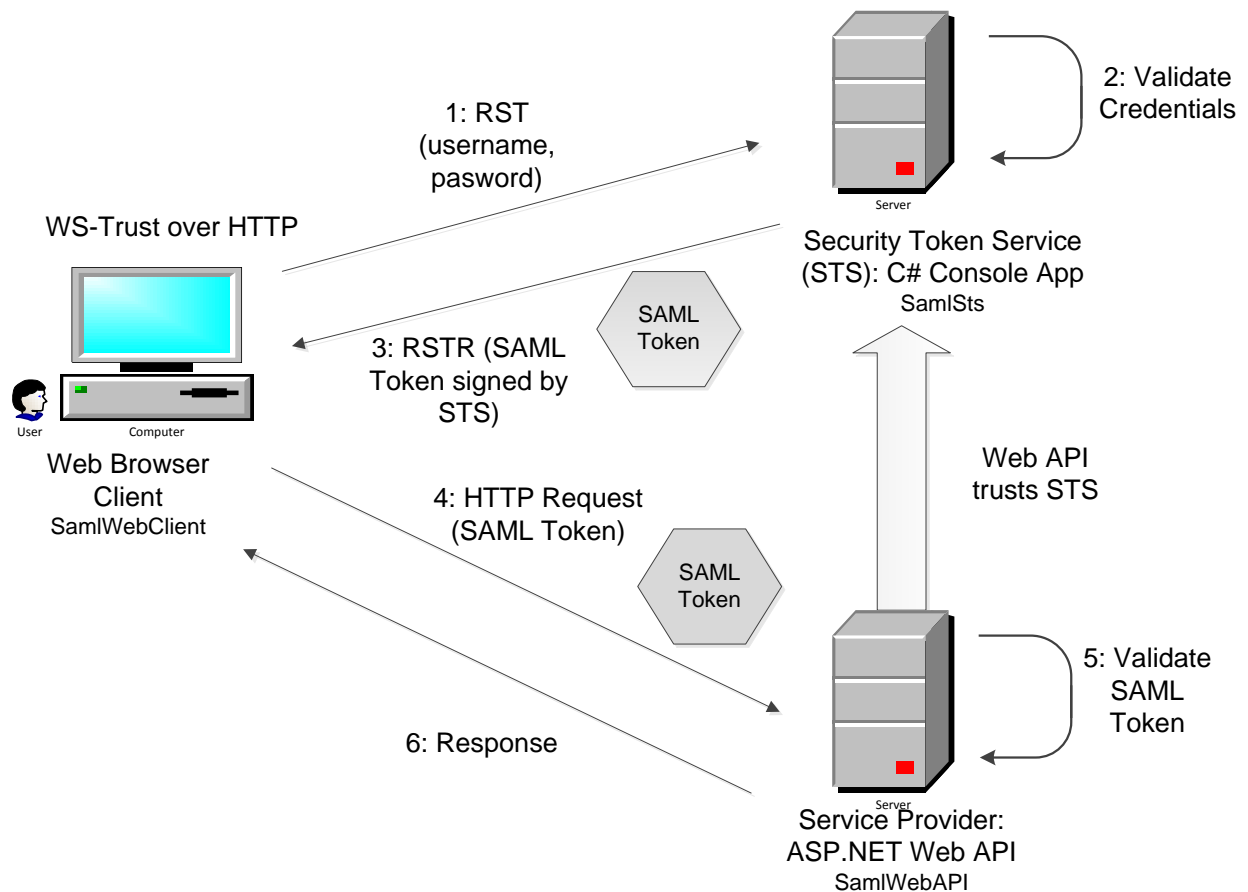
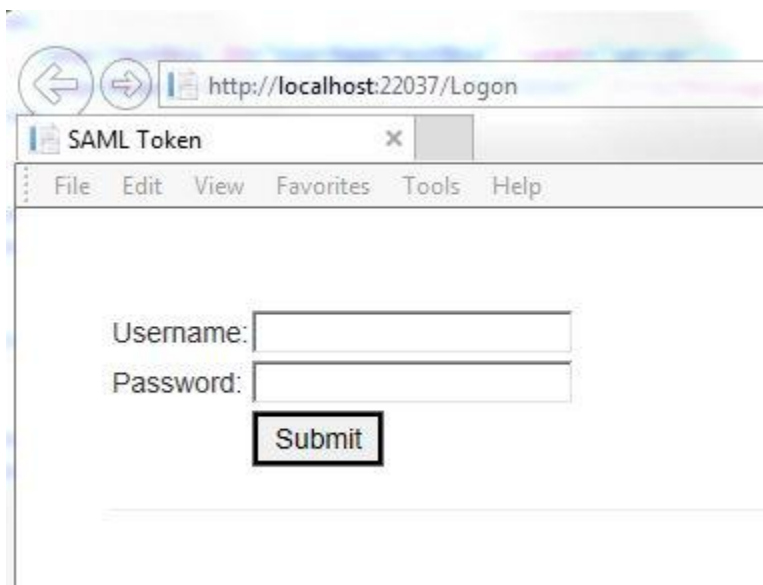


Figure 8.1 Using SAML token with ASP.NET Web API [16]

Figure 8.1 shows overall setup. STS exposes WS-Trust endpoint that creates token in RSTR (request for security token response) response for incoming RST (request for security token) request. STS supports WS-Federation for passive clients and WS-Trust directly for active clients.

The user enters his name and password on the page “Logon.aspx” (Figure 8.2). The Web client sends the request to the STS. The STS determines that the client’s credentials are valid, and sends the SAML token to the web client (Figure 8.3). The web client caches the SAML in a cookie and sends a request message to the service provider; a request message includes the SAML token (Figure 8.4). The service provider validates the SAML token and processes the request. Figure 8.5 shows final data received from the service provider.



The image shows a web browser window with the address bar set to `http://localhost:22037/Logon`. The browser's title bar reads "SAML Token". The page content includes a "Username:" label next to a text input field, a "Password:" label next to another text input field, and a "Submit" button positioned below the password field.

Figure 8.2. Logon page

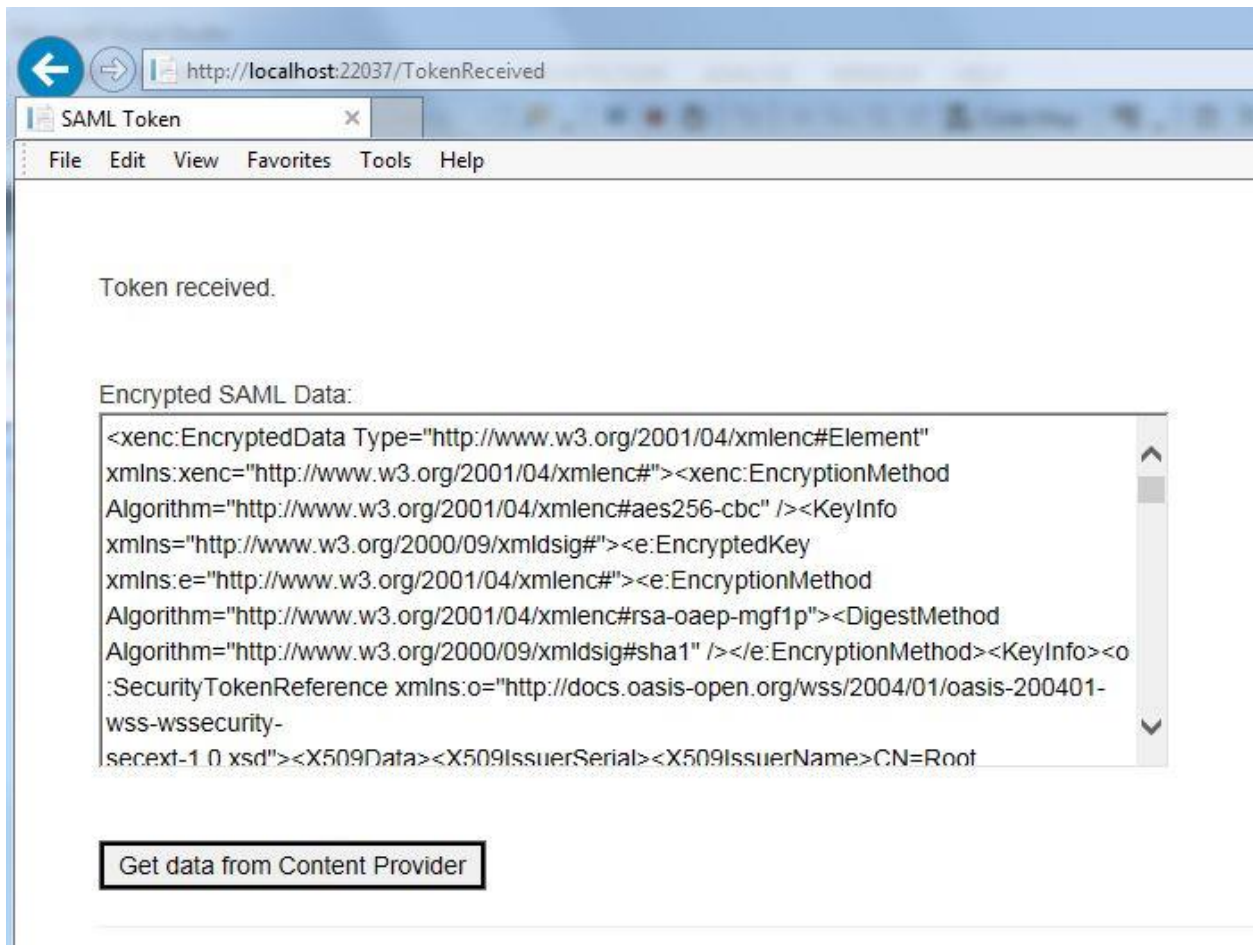


Figure 8.3. Token received

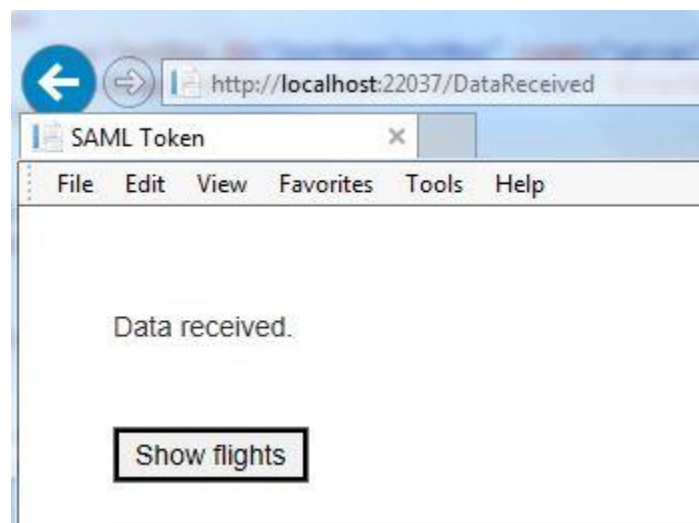
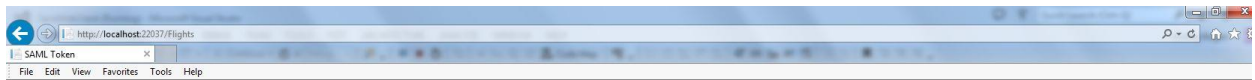


Figure 8.4. Data received



BELTS 7-10		
FLIGHT	FROM	BELT
LY028 09:20	NEW-YORK	1 ▶
OK286 09:40	PRAGUE	2 ▶
VY7844 10:20	BARCELONA	3 ▶
Unusual Baggage		

Figure 8.5. Show final flights data

In order to create certificates, I run the following script in Developer Command Prompt for Visual Studio 2013 (users of Windows 7 need to run it under Administrator permissions):

```
makecert.exe -sr LocalMachine -ss My -a sha1 -n CN=MySTS -sky exchange -pe
```

```
makecert.exe -sr LocalMachine -ss My -a sha1 -n CN=RP -sky exchange -pe
```

After running, we see “Succeeded” message.

Appendix E contains class diagrams of the applications. Appendix F contains encrypted SAML data.

## 9. Conclusion

SAML can be extremely powerful. It has rich benefits. Companies can easily share identity information, and security is improved by eliminating shared accounts. Companies must have documentation available to exchange when setting up SAML associates. Each SAML use case can be customized according to individual business needs.

SAML solves the problem of secure interoperability of web services. Service providers can use different security protocols.

Using SAML metadata is very helpful because it eliminates typos and errors when setting up partner entity. Metadata can help identity provider to understand what exactly service provider needs in SAML assertion.

SAML helps to solve the following security problems:

- Interoperability, the ability to communicate between security systems within different tiers.
- Privacy, the ability to protect users' identity and personal data (for example, credit card numbers) from disclosure.
- Federation, the ability of different companies to exchange requests and data with each other.
- SSO, the ability to log in once and use this login to access applications.

We need open standards like SAML, because the model must be publicly available and must be analyzed very well. When full model is available to all, many people are trying to find security holes, so all security problems have to be solved.

## 10. Summary

The paper overviews web services and technologies behind them, and web service security. The paper focuses on web service authorization using SAML and XACML. It defines requirements for authorization of web services and investigates existing authorization solutions.

The paper shows the importance of web service authorization. It provides a solution for authentication and authorization issues. The paper discusses SAML, which defines syntax and semantics for security message exchanges. It then introduces XACML, which describes how to evaluate authorization requests according to policy rules.

The paper reviews several academic articles about web service security. The articles show simple and complex solutions for web service authorization. The papers introduce the basic concepts for securing web services and propose the system architectures for web services. The articles show the importance of authorization policies in web service composition. The papers show concepts and design of the frameworks and introduce selected implementation aspects.

The last part of the paper presents an application using the Microsoft framework WIF (Windows Identity Foundation). WIF is a framework for implementing claim-based scenario with web browsers. After presenting the high level overview of WIF, the paper presents a solution that uses SAML tokens. The solution was implemented using ASP.NET Web API technology and sample GUI is presented to demonstrate it.



## 11. References

- [1] Thomas Ziebermayr, Stefan Probst: *Web Service Authorization Framework*. Proceedings of the IEEE International Conference on Web Services (ICWS'04), San Diego, California, June 06-June 09, 2004. ISBN: 0-7695-2167-3.
- [2] Weiliang Zhao, Vijay Varadharajan: *Trust Management for Web Services*, 2008 IEEE International Conference on Web Services, pages 818-821, 2008.
- [3] Gabriele Garzoglio, Ian Alderman, Mine Altunay, Rachana Ananthakrishnan: *Definition and Implementation of a SAML-XACML Profile for Authorization Interoperability Across Grid Middleware in OSG and EGEE*, Journal of Grid Computing Volume 7, Number 3, pages 297-307, 23 April 2009.
- [4] Li-Qun Kuang, Yuan Zhang, and Xie Han: *Access Control Policies for Web Services in Medical Aid System*, ICIII '09 Proceedings of the 2009 International Conference on Information Management, Innovation Management and Industrial Engineering - Volume 02: pages 167-170, 2009.
- [5] Mohsen Rouached and Claude Godart: *Specification and Verification of Authorization Policies for Web Services Composition*. CAiSE Forum, pages 33-37, 2007.
- [6] Joachim Biskup, Barbara Carminati, Elena Ferrari, Frank Muller, and Sandra Wortmann: *Towards Secure Execution Orders for Composite Web Services*, ICWS 2007: pages 489-496, 2007.
- [7] Yuri Demchenko, Cees de Laat: *Extending XACML Authorisation Model to Support Policy Obligations Handling in Distributed Application*. MGC'08, December 1-5, 2008 Leuven, Belgium, 2008.
- [8] Lalana Kagal, Tim Finin, Massimo Paolucci, Naveen Srinivasan, Katia Sycara, and Grit Denker: *Authorization and Privacy for Semantic Web Services*, IEEE Intelligent Systems 19(4): pages 50-56, 2004.
- [9] Mark O'Neill: *Web Services Security*. McGraw-Hill/Osborne, 2003.
- [10] Bret Hartman, Donald J. Flinn, Konstantin Beznosov, Shirley Kawamoto: *Mastering Web Services Security*. Wiley Publishing, 2003.
- [11] OASIS: Security Assertion Markup Language (SAML) V2.0 Technical Overview. OASIS, 2008.
- [12] OASIS: A Brief Introduction to XACML. OASIS, 2003.
- [13] OASIS: Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS, 2005.

- [14] Ethan Cerami: Web Services Essentials, 2002
- [15] Jothy Rosenberg, David Remy: *Securing Web Services with WS-Security*. SAMS Publishing, 2004.
- [16] Badrinarayanan Lakshmiraghavan: Pro ASP.NET Web API Security, 2013
- [17] OASIS: eXtensible Access Control Markup Language (XACML) .OASIS, 2005.
- [18] Keith Brown, Sesha Mani: Microsoft Windows Identity Foundation (WIF) Whitepaper for Developers, 2009

## **Appendix A. Definitions**

### ACL

ACL (Access Control List) is part of operating system that determines who can access, what and with which privileges.

There are two types of ACL: DACL (Discretionary Access Control List) that determines access rights to shared objects, and SACL (System Access Control List) that determines audit policy for secured resources.

ACL can secure the following resources: files and directories, shares, registry keys, job objects, printers, Active Directory objects.

### Active Directory

Active Directory is directory service that stores information about resources on network and exposes this information to users and administrators. Active Directory gives users access to permitted resources on network using single logon.

### HTTPS

Combination of HTTP and SSL/TLS that provides encrypted communication between browser and web server.

### IIS

IIS (Internet Information Server) is a web server application and set of feature extension modules created by Microsoft to use with Microsoft Windows.

IIS supports HTTP, HTTPS, FTP, FTPS, and NNTP.

### OASIS

Organization for the Advancement of Structured Information Standards.

### PDP, PEP, PRP, PAP, PIP

All policies stored at PDP (Policy Decision Point). When PEP (Policy Enforcement Point) needs to make a decision, it sends relevant information to PDP. PDP analyzes this information, makes decision and passes it to PEP. Then PEP performs this decision. PRP is Policy Retrieval Point. PAP is Policy Administration Point. PIP is Policy Information Point.

### SAML

The SAML (Security Assertion Markup Language) is XML framework for exchanging authentication and authorization information. The information is expressed in the form of assertions about subjects. Subject is an entity that has identity within security domain. It can be people or computer processes. The protocol defines how client can request assertions from authorities. Server listens for incoming SOAP calls, decodes XML business logic, and applies it to relevant application. Client sends SAML requests to authority. Authority sends SAML assertion responses. Applications can use request/response protocol.

### SOAP

SOAP (Simple Object Access Protocol) transports XML from one computer to another via standard transport protocols. HTTP is the most common transport protocol. SOAP itself is defined using XML. SOAP needs to be secured.

### SP

Service Provider.

### SSO

SSO (single sign-on) allows user log in once and gain access to all systems without logging in again into each of the systems.

### STS

Security Token Service

## UDDI

UDDI (Universal Description, Discovery, and Integration) is a repository with the location of web service and WSDL that describes it. Client sends correctly formatted SOAP document to web service, and web service returns another SOAP document containing response. The response is XML document that contains required data. If we need to change technology, we don't need to change web service itself. We just need to develop new agent that implements the same service. So clients do not need to know if something changed.

## W3C

W3C is World Wide Web Consortium, main international standards organization for WWW (World Wide Web).

## WS-Policy

WS-Policy (Web Services Policy Framework) provides model and syntax to describe and communicate policies of web service. WS-Policy defines policy in terms of assertions. Policy can be a single assertion or collection of assertions.

## WS-Security

WS-Security (Web Services Security) defines enhancements to SOAP specification. It provides quality of protection through message integrity, confidentiality, and authentication. WS-Security provides mechanism for associating security tokens with SOAP messages. The base methods it supports are X.509 and Kerberos.

## WIF

WIF (Windows Identity Foundation) is Microsoft software framework for building identity-aware applications.

## WSDL

Web service exposes interface, described by WSDL document. WSDL (Web Services Description Language) is a language that defines interfaces and returned data types to and from web service. WSDL defines set of operations that web service provides and structure of SOAP messages.

## XML

XML is structured self-describing way to represent data. This data is independent of application, protocol, operating system, and programming language. XML allows sharing data among different applications, so they can easily talk to one another.

## XML Encryption

XML Encryption is a process for encrypting and decrypting of digital content, including entire XML documents or their parts. XML-based syntax represents encrypted content and information that enables to decrypt it. XML Encryption allows encryption of digital content (for example, images) or XML fragments. The disadvantage of encrypting entire document is that you cannot search non-sensitive information.

## XML Schema

XML Schema describes rules for XML document.

## XML Signature

XML Signature is foundational technology for standard WS-Security and for web services security in general. Digital signature provides mechanism for message integrity. The purpose of XML Signature specification is to have XML-compliant syntax to represent signature of XML documents. XML Signature is based upon public key cryptography. Each participant has public key and private key. He makes his public key available to the public and keeps his private key secret. It is impossible to guess private key of participant via his public key.

## XPath

Xpath (XML Path Language) is a query language for selecting parts of XML document. The version 2.0 is the current version of XPath defined by W3C. XML is a tree of nodes. XPath allows selecting nodes by navigation path in the document tree.

XPath 2.0 is sublanguage of XSLT 2.0 and subset of XQuery 1.0. All three languages have the same data model, type system and functions library.

## Appendix B. SAML Syntax Samples

### Assertions

```
<element name="Subject" type="saml:SubjectType"/>
<complexType name="SubjectType">
  <choice>
    <sequence>
      <element ref="saml:NameIdentifier"/>
      <element ref="saml:SubjectConfirmation" minOccurs="0"/>
    </sequence>
    <element ref="saml:SubjectConfirmation"/>
  </choice>
</complexType>
```

The top-level statement portion of assertion is abstract element. Valid assertion must contain one of three possible statements defined by SAML – authentication, authorization, or attribute.

### Authentication statement

```
<element name="AuthenticationStatement" type="saml:AuthenticationStatementType"/>
<complexType name="AuthenticationStatementType">
  <complexContent>
    <extension base="saml:SubjectStatementAbstractType">
      <sequence>
        <element ref="saml:SubjectLocality" minOccurs="0"/>
        <element ref="saml:AuthorityBinding" minOccurs="0"
          maxOccurs="unbounded"/>
      </sequence>
      <attribute name="AuthenticationMethod" type="anyURI" use="required"/>
      <attribute name="AuthenticationInstant" type="dateTime" use="required"/>
    </extension>
  </complexContent>
</complexType>
```



## Attribute statement

```
<element name="AttributeStatement" type="saml:AttributeStatementType"/>
<complexType name="AttributeStatementType">
  <complexContent>
    <extension base="saml:SubjectStatementAbstractType">
      <sequence>
        <element ref="saml:Attribute" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
<element name="Attribute" type="saml:AttributeType"/>
<complexType name="AttributeType">
  <complexContent>
    <extension base="saml:AttributeDesignatorType">
      <sequence>
        <element ref="saml:AttributeValue" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
<element name="AttributeDesignator" type="saml:AttributeDesignatorType"/>
<complexType name="AttributeDesignatorType">
  <attribute name="AttributeName" type="string" use="required"/>
  <attribute name="AttributeNamespace" type="anyURI" use="required"/>
</complexType>
```

## Authorization statement

```
<element name="AuthorizationDecisionStatement" type="saml:AuthorizationDecisionStatementType"/>
<complexType name="AuthorizationDecisionStatementType">
  <complexContent>
    <extension base="saml:SubjectStatementAbstractType">
      <sequence>
        <element ref="saml:Action" maxOccurs="unbounded"/>
        <element ref="saml:Evidence" minOccurs="0"/>
      </sequence>
      <attribute name="Resource" type="anyURI" use="required"/>
      <attribute name="Decision" type="saml:DecisionType" use="required"/>
    </extension>
  </complexContent>
</complexType>
```

## Assertion

```
<saml:AttributeStatement>
  <saml:Subject>
    <saml:NameIdentifier
      NameQualifier="www.test.com"
      Format="urn:oasis:names:tc:SAML:1.0:assertion
#WindowsDomainQualifiedName">ne\testuser</saml:NameIdentifier>
    </saml:Subject>
    <saml:Attribute AttributeName="role" AttributeNamespace="http://www.test.com/t">
      <saml:AttributeValue>
        Administrator
      </saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
```

## SAML Request

```
<element name="Request" type="samlp:RequestType"/>
<complexType name="RequestType">
  <complexContent>
    <extension base="samlp:RequestAbstractType">
      <choice>
        <element ref="samlp:Query"/>
        <element ref="samlp:SubjectQuery"/>
        <element ref="samlp:AuthenticationQuery"/>
        <element ref="samlp:AttributeQuery"/>
        <element ref="samlp:AuthorizationDecisionQuery"/>
        <element ref="saml:AssertionIDReference" maxOccurs="unbounded"/>
        <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

## Attribute Query

```
<element name="AttributeQuery" type="samlp:AttributeQueryType"/>
<complexType name="AttributeQueryType">
  <complexContent>
    <extension base="samlp:SubjectQueryAbstractType">
      <sequence>
        <element
          ref="saml:AttributeDesignator"
          minOccurs="0"
          maxOccurs="unbounded"/>
      </sequence>
      <attribute name="Resource" type="URI reference" use="optional"/>
    </extension>
  </complexContent>
</complexType>
```

## Authorization Query

```
<element name="AuthorizationDecisionQuery" type="samlp:AuthorizationDecisionQueryType"/>
<complexType name="AuthorizationDecisionQueryType">
  <complexContent>
    <extension base="samlp:SubjectQueryAbstractType">
      <sequence>
        <element ref="saml:Action" maxOccurs="unbounded"/>
        <element ref="saml:Evidence" minOccurs="0" maxOccurs="1"/>
      </sequence>
      <attribute name="Resource" type="anyURI" use="required"/>
    </extension>
  </complexContent>
</complexType>
```

## Appendix C. SAML Request / Response Samples

### SAML Request:

```
<samlp:AuthnRequest
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="identifier_1"
  Version="2.0"
  IssueInstant="2014-12-05T09:00:00Z"
  AssertionConsumerServiceIndex="1">
  <saml:Issuer>https://cars.com/SAML2</saml:Issuer>
  <samlp:NameIDPolicy AllowCreate="true"
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
</samlp:AuthnRequest>
```

## SAML Response:

```
<samlp:Response
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="identifier_2"
InResponseTo="identifier_1"
Version="2.0"
IssueInstant="2014-12-05T09:00:01Z"
Destination="https://cars.com/SAML2/SSO/POST">
<saml:Issuer>https://airlines.com/SAML2</saml:Issuer>
<samlp:Status>
<samlp:StatusCode
Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="identifier_3"
Version="2.0"
IssueInstant="2014-12-05T09:00:05Z">
<saml:Issuer>https://airlines.com /SAML2</saml:Issuer>
<ds:Signature
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
<saml:Subject>
<saml:NameID
Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
3f7b3dcf-1674-4ecd-92c8-1544f346baf8
</saml:NameID>
<saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
<saml:SubjectConfirmationData
InResponseTo="identifier_1"
Recipient="https:// cars.com /SAML2/SSO/POST"
NotOnOrAfter="2004-12-05T09:27:05Z"/>
</saml:SubjectConfirmation>
</saml:Subject>
<saml:Conditions
NotBefore="2014-12-05T09:00:00Z"
NotOnOrAfter="2014-12-05T09:10:00Z">
<saml:AudienceRestriction>
<saml:Audience>https:// cars.com /SAML2</saml:Audience>
</saml:AudienceRestriction>
</saml:Conditions>
<saml:AuthnStatement
AuthnInstant="2014-12-05T00:00:00Z"
SessionIndex="identifier_3">
<saml:AuthnContext>
<saml:AuthnContextClassRef>
urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
</saml:AuthnContextClassRef>
</saml:AuthnContext>
</saml:AuthnStatement>
</saml:Assertion>
</samlp:Response>
```

## Appendix D. XACML Syntax Samples

XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<record xmlns="medico.com/records.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="medico.com/records.xsd
D:\MYDOCU~1\Standards\XACML\record.xsd">
  <patient>
    <patientName>
      <first>Bartholomew</first>
      <last>Simpson</last>
    </patientName>
    <patientContact>
      <street>27 Shelbyville Road</street>
      <city>Springfield</city>
      <state>MA</state>
      <zip>12345</zip>
      <phone>555.123.4567</233 phone>
      <fax/>
      <email/>
    </patientContact>
    <patientDoB xsi:type="date">1992-03-21</patientDoB>
    <patientGender xsi:type="string">male</patientGender>
    <policyNumber xsi:type="string">555555</policyNumber>
  </patient>
  <parentGuardian>
    <parentGuardianName>
      <first>Homer</first>
      <last>Simpson</last>
    </parentGuardianName>
    <parentGuardianContact>
      <street>27 Shelbyville Road</street>
      <city>Springfield</city>
      <state>MA</state>
      <zip>12345</zip>
      <phone>555.123.4567</phone>
      <fax/>
      <email>homers@aol.com</email>
    </parentGuardianContact>
  </parentGuardian>
  <primaryCarePhysician>
    <physicianName>
      <first>Julius</first>
      <last>Hibbert</last>
    </physicianName>
    <physicianContact>
      <street>1 First St</street>
      <city>Springfield</city>
      <state>MA</state>
      <zip>12345</zip>
      <phone>555.123.9012</phone>
```

```

<fax>555.123.9013</fax>
<email/>
</physicianContact>
<registrationID>ABC123</registrationID>
</primaryCarePhysician>
<insurer>
<name>Blue Cross</name>
<street>1234 Main St</street>
<city>Springfield</city>
<state>MA</state>
<zip>12345</zip>
<phone>555.123.5678</phone>
<fax>555.123.5679</fax>
<email/>
</insurer>
<medical>
<treatment>
<drug>
<name>methylphenidate hydrochloride</name>
<dailyDosage>30mgs</dailyDosage>
<startDate>1999-01-12</startDate>
</drug>
<comment>patient exhibits side-effects of skin coloration and carpal degeneration</comment>
</treatment>
<result>
<test>blood pressure</test>
<value>120/80</value>
<date>2001-06-09</date>
<performedBy>Nurse Betty</performedBy>
</result>
</medical>
</record>

```

## Rule 1

```

<?xml version="1.0" encoding="UTF-8"?>
<rule ruleId="//medico.com/rules/rule1" effect="Permit"
xmlns="http://www.oasis-open.org/committees/xacml/docs/draft-xacml-schema-policy-12.xsd"
xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-28.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-open.org/committees/xacml/docs/draft-xacml-schema-policy-12.xsd
D:\MYDOCU~1\Standards\XACML\V12SCH~1\XACMLV~3.XSD">
<description>A person may read any record for which he or she is the designated patient</description>
<target>
<subjects>
<saml:Attribute AttributeName="RFC822Name" AttributeNamespace="//medico.com">
<saml:AttributeValue>*</saml:AttributeValue>
</saml:Attribute>
</subjects>
<resources>
<saml:Attribute AttributeName="documentURI" AttributeNamespace="//medico.com">
<saml:AttributeValue>//medico.com/record.*</saml:AttributeValue>
</saml:Attribute>

```

```

</resources>
<actions>
<saml:Action>read</saml:Action>
</actions>
</target>
<condition>
<equal>
<saml:AttributeDesignator AttributeName="requestor" AttributeNamespace="//oasis-
open.org/committees/xacml/docs/identifiers"/>
<saml:AttributeDesignator AttributeName="patientName"
AttributeNamespace="//medico.com/record/patient"/>
</equal>
</condition>
</rule>

```

## Rule 2

```

<?xml version="1.0" encoding="UTF-8"?>
<rule ruleId="//medico.com/rules/rule2" effect="Permit"
xmlns="http://www.oasis-open.org/committees/xacml/docs/draft-xacml-schema-policy-12.xsd"
xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-28.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-open.org/committees/xacml/docs/draft-xacml-schema-policy-12.xsd
D:\MYDOCU~1\Standards\XACML\V12SCH~1\XACMLV~3.XSD">
<description>A person may read any record for which he or she is the designated parent or guardian, and for
which the patient is under 16 years of age</description>
<target>
<subjects>
<saml:Attribute AttributeName="RFC822Name" AttributeNamespace="//medico.com">
<saml:AttributeValue>*</saml:AttributeValue>
</saml:Attribute>
</subjects>
<resources>
<saml:Attribute AttributeName="documentURI" AttributeNamespace="//medico.com">
<saml:AttributeValue>//medico.com/record.*</saml:AttributeValue>
</saml:Attribute>
</resources>
<actions>
<saml:Action>read</saml:Action>
</actions>
</target>
<condition>
<and>
<equal>
<saml:AttributeDesignator AttributeName="requestor" AttributeNamespace="//oasis-
open.org/committees/xacml/docs/identifiers"/>
<saml:AttributeDesignator AttributeName="parentGuardianName"
AttributeNamespace="//medico.com/record/parentGuardian"/>
</equal>
<not>
<greaterOrEqual>
<minus>
<saml:AttributeDesignator AttributeName="today'sDate" AttributeNamespace="//medico.com"/>

```

```

    <saml:AttributeDesignator AttributeName="patientDoB"
AttributeNamespace="//medico.com/record/patient"/>
    </minus>
    <saml:Attribute AttributeName="ageOfConsent" AttributeNamespace="//medico.com">
    <saml:AttributeValue>16-0-0</saml:AttributeValue>
    </saml:Attribute>
    </greaterOrEqual>
    </not>
    </and>
    </condition>
    </rule>

```

### Rule 3

```

<?xml version="1.0" encoding="UTF-8"?>
<saml:Assertion MajorVersion="0" MinorVersion="24" AssertionID="A7F34K90" Issuer="medico.com"
IssueInstant="2002-03-22T08:23:47-05:00">
    <policyStatement policyId="//medico.com/rules/policy3" ruleCombiningAlgId="//www.oasis-
open.org/committees/xacml/docs/ruleCombiningAlgorithms/denyOverrides" xmlns="http://www.oasis-
open.org/committees/xacml/docs/draft-xacml-schema490
    policy-12.xsd" xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-
28.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.oasis-open.org/committees/xacml/docs/draft-xacml-schema-policy-12.xsd
D:\MYDOCU~1\Standards\XACML\V12SCH~1\XACMLV~3.XSD">
    <description>A physician may write any medical element for which he or she is the designated primary care
physician, provided an email is sent to the patient</description>
    <target>
    <subjects>
    <saml:Attribute AttributeName="role"
AttributeNamespace="//medico.com">
    <saml:AttributeValue>physician</saml:AttributeValue>
    </saml:Attribute>
    </subjects>
    <resources>
    <saml:Attribute AttributeName="documentURI" AttributeNamespace="//medico.com">
    <saml:AttributeValue>//medico.com/record/medical.*</saml:AttributeValue>
    </saml:Attribute>
    </resources>
    <actions>
    <saml:Action>write</saml:Action>
    </actions>
    </target>
    <ruleSet>
    <rule ruleId="//medico.com/rules/rule3" effect="Permit">
    <description>A physician may write any medical element for which he or she is the designated primary care
physician</description>
    <target>
    <subjects>
    <saml:Attribute AttributeName="role" AttributeNamespace="//medico.com">
    <saml:AttributeValue>physician</saml:AttributeValue>
    </saml:Attribute>
    </subjects>
    <resources>

```



```

<saml:Attribute AttributeName="documentURI" AttributeNamespace="//medico.com">
<saml:AttributeValue>//medico.com/record/medical.*</saml:AttributeValue>
</saml:Attribute>
</resources>
<actions>
<saml:Action>write</saml:Action>
</actions>
</target>
<condition>
<and>
<equal>
<saml:AttributeDesignator AttributeName="requestor" AttributeNamespace="//oasis-
open.org/committees/xacml/docs/identifiers"/>
<saml:AttributeDesignator AttributeName="physicianName"
AttributeNamespace="//medico.com/record/primaryCarePhysician"/>
</equal>
<present>
<saml:AttributeDesignator AttributeName="patient/email" AttributeNamespace="//medico.com/record"/>
</present>
</and>
</condition>
</rule>
</ruleSet>
<obligations>
<obligation ObligationId="//medico.com/emailer" fulfilOn="Permit">
<saml:AttributeDesignator AttributeName="patient/email" AttributeNamespace="//medico.com/record"/>
<saml:Attribute AttributeName="messageText" AttributeNamespace="//medico.com/">
<saml:AttributeValue>Your medical record has been accessed by:</saml:AttributeValue>
</saml:Attribute>
<saml:AttributeDesignator AttributeName="requestor" AttributeNamespace="//oasis-
open.org/committees/xacml/docs/identifiers"/>
</obligation>
</obligations>
</policyStatement>
</saml:Assertion>

```

#### Rule 4.

```

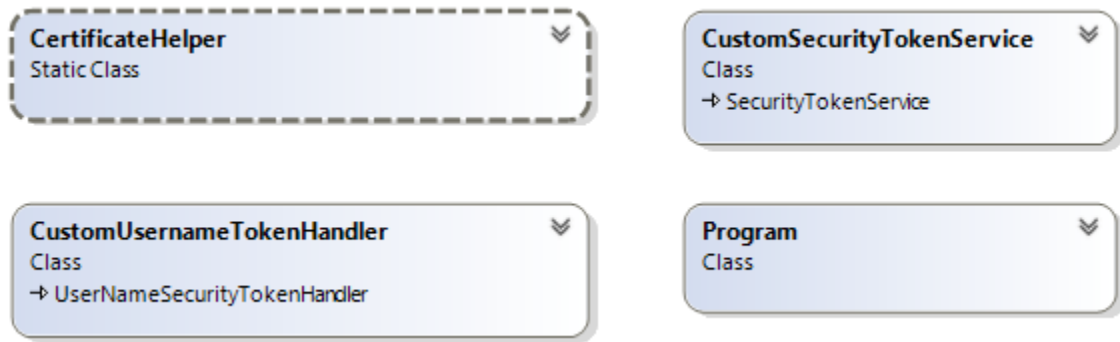
<?xml version="1.0" encoding="UTF-8"?>
<rule ruleId="//medico.com/rules/rule4" effect="Deny"
xmlns="http://www.oasis-open.org/committees/xacml/docs/draft-xacml-schema-policy-12.xsd"
xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-28.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-open.org/committees/xacml/docs/draft-xacml-schema-policy-12.xsd
D:\MYDOCU~1\Standards\XACML\VI2SCH~1\XACMLV~3.XSD">
<description>An administrator shall not be permitted to read or write medical elements of a patient
record</description>
<target>
<subjects>
<saml:Attribute AttributeName="role" AttributeNamespace="//medico.com">

```

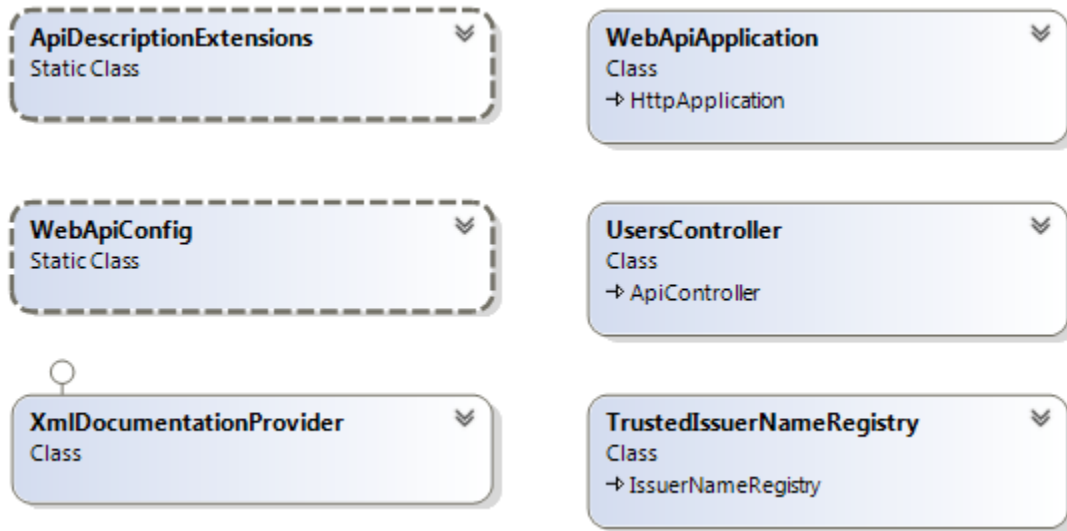
```
<saml:AttributeValue>administrator</saml:AttributeValue>
</saml:Attribute>
</subjects>
<resources>
<saml:Attribute AttributeName="documentURI" AttributeNamespace="//medico.com">
<saml:AttributeValue>//medico.com/record/medical.*</saml:AttributeValue>
</saml:Attribute>
</resources>
<actions>
<saml:Action>read write</saml:Action>
</actions>
</target>
</rule>
```

## Appendix E. Class Diagram of the Applications

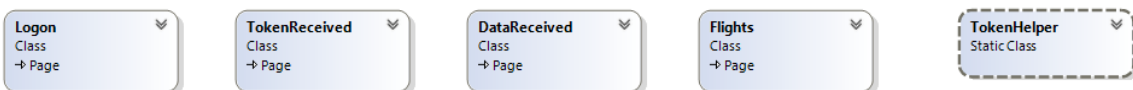
### SamlSts



### SamlWebApi



### SamlWebClient



## Appendix F. Encrypted SAML Data of the Applications

```
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmenc#aes256-cbc" /><KeyInfo
xmlns="http://www.w3.org/2000/09/xmldsig#"><e:EncryptedKey
xmlns:e="http://www.w3.org/2001/04/xmenc#"><e:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmenc#rsa-oaep-mgf1p"><DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
/></e:EncryptionMethod><KeyInfo><o:SecurityTokenReference xmlns:o="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd"><X509Data><X509IssuerSerial><X509IssuerName>CN=Root
Agency</X509IssuerName><X509SerialNumber>168033855021051652025182686688371576072</X50
9SerialNumber></X509IssuerSerial></X509Data></o:SecurityTokenReference></KeyInfo><e:CipherDat
a><e:CipherValue>wxkZ1gllP/oZFNtzbPIH0AI267qAS7+M7x50ThGBC7Kc0i7MXyczBG6zUQqKdN8uT
ShqUwC/laYcsX0uKcTSu9YJpZLWYEGHhXeN9eJ7EcrhuaY7qcXt55WFyFaWY6zXuA+ejqpZCZ2ujx9Pa
YlapF6uDvDsWKpe2htlWqx/6vPcBP2TNkHwVdHr4skciVTqE2DIHynbRmAq2fi49JuSxaVR/A12tpHgkX0J
MzfVMzSmyiB6rQEOPHa5yDEAlJyNTdxaRuJO9VX0YaA9X7ATX0W25LHNISdZt99MArrq2W+weqgrloTr
AvelbG5wV0/X8Peinws+pY52ZYYOyuevg==</e:CipherValue></e:CipherData></e:EncryptedKey></KeyI
nfo><xenc:CipherData><xenc:CipherValue>4vG+VlloqcbYiWAPyUs/n5lVZcug4qXRO0TfZqfOefO+ZirYa
mzdsavvTYW/it4PteXLGLLkI+cBX6XXyMEPLCexnB7G7le5zSI0pYcvFV791lnRZF//m/soURdguYITIHN
Wwu9dE2tde+KhC/YBYSQE+EflIq5K290QjsJ+uOatTASEXJo/QtwvqhTIIIMCxyijcfJ9RWgrUAZ4NHQmg6
dwFyMzL8pA5A7CZ3hPjEMRWXV+2a51BgDL62qhxtJehDmdEZhWzjjGeKLAq1Sa9Kunl2hedm4MZqoh
h69Y6xVF11zQPDtODF3htprtCXqUDQK2vQmAvJDflzypWT77ceQg46PdnlDIAj+0MKCKpo5jJjDj7w8dW
TMTogo/y58ita8/H8YPxQcrKf8yheIIDzC2VGuKR9uufuF0JdOhkuiYNuqIT+PCRahb8sn7xYyd96+DQT7T
RtyPB54XcqIVZHD+1K2tSt9FXkDHGJixgsy4pYOB6iVOceLex+1X0rJLWa7j7WBtqGL9kHDTIn+kVMTI
eAxfYUrBnZsGuJPXZ1DQV8wVPwNNBL/RlkSKHwezFhhlwPfEV/xhz1bEbzK0hIA+5DyhzSSp5hZB52+z
zcBZRjzTmzKhqUW3fd/hoSCReF25egj6EQP/Bzx1Rqw3LczsNGPX0F5IWmHghFGEsCmWE7hybFn5Yy
H75xFVigw3z27995D/RSp2ND/PtTM2QUxK1YaPN8nGUrV96RVuipizF5OWqqua2lzi8EvSjQnEZ62PdRO
m0ZwA8zQG5JyA5hyxk1PB15olr8TvfNYmAAOGEMR5pHvsdygp7nkuF2ikdFawOdZiYwvpv1t/RxvLY/ExZ
X3DxK1xouC2dNTjqso+IDsWe0npesSo+kxbn8iFkgPeRYL9Yh6WMrWARyFhV3GeXWor484+TazQyD5i
OTa+HZy7lRo3Jl7TGGdNNlvKhYDmOWliro3bqbN8qfBz7klyzI0onhvVePihCE6Wbl7CT9b3qL37qNzMzc
/BQtuf3s978vB0tiPsa9vCfR6g7/VTLsZ6NPO5aSB8qWihGJ7uB9evTiHAJikLju3jiZ60uyutfk0J4ro8GGkA
8WMOE8l0VqxZemcw6+CCYr4YGHcccyKmYjiATPdrfpCM918QsBXdfV3JvMkDc2RKZ6bpe10H5wF09g
kqYTcqHy4yes+JaGZmnmxVAmKD6K0krFmQy3pLxIrC/HQzb5GdiLELrl+081vhVnZ5jQ05Kc0Arzgf/c
8UQgUkMY+ifCLyBENW75VFzCJ3FxtMqAYkAGE0zVpM0kov3xWkuhEDJhwc3F9oZ56bSsPP+EV6np
mXqffKp93xoAu9k3uLXfzE+/x1w9EfAtbyisJrNyiNJXn8UPQID3/1ma5Snllf8nFw+mTB66iN3rS6qHXQ5F
QcAk8a4vIFwSJRQLT3BhbH4glEuTfkkGpyYxTDLXbj6r+7lBuPPLHKGIMCYk9P/qOWQXPMmZ72muD+R
m987zg2LsvyA8GdfFot1S9mN28MrDvuhsdqf1c17npyyi5Du8rHczgJPZ0xNBrhUx0Q74alsU2mmaVax8CtL
qwrwFEzidXrYW6hknAgNQGWuCov9H+HO0rT+IXf/xe2BCC5i4LoN56HOuyXPXu0kGtpjl+Ssv6o3CCs51i
FqKDjw9OJAXurh+yDqklUJP0TNXKC6AtWOKItjNoFRZK4xuxRqsYgEEy4mxwY03p0PNDMBBXd3zoqR
zMvSaEjCWUGLEl4gvyLfk2B5vHsbaCmBCsNauoi94l9FIZs6ZyGlbzAgyeiULCDxMAE5A8ESxIZUbkbZ
WFsFU/kEuWZ7iw+mCXKgNKL/pu0STlwJ+Ce25blcwDGTcrekRvXiRHysSdcBvZTDxPLssemZ/5v1CYZ/
v7V+sPPuvBUwch22l7xVJtydIAqUgNg22xSzXY4to9j862QHqVeZc2ML/dkiPc08BtCJiauDA5kGty/sls42u
wfVo+7nFlwVALc+yip+JBgY87zO/wnuFt2yzAd4X1TJXv5SqOCPfdq1MFnwEaH0yAT4/eOpNkm6GHEZz
pFKLZpBN4t67F5KwBtASvAJSYHviVcMoVzbHq2rJJN4MPwcU+jwNoCva1pTeBNyhovDdAdg7FC14GK
ME8jaJhlHX8F00AckBlxHq9e+Cbt9WKfG4LHfNkmZ1FRDAo1Bl6iwx8+3A6UvGoehHA2cjVIIPOJhwOff
9AYVfb1kqdtSqeMCiu1RmmwYGTOINO3MWBp2hT7/nwO1eU/SLqiSSaTWvw10ZzILM0l269tt1HtklImm
TjMz/do5wyPkKJ2JnCOzznVn0CpnOyJkKBMkZETX21RUvRq4X7L3pE2eEJyeuEZGCxVWe/wKtiEGDN
8l1vv9moZFLfGM0y7KO9OM3DwDHZ2vzD2O60lb3ZfBZpae547q8mjHCKYlJAu+JaX15c20XoVRGy1O
zM5c7kfdFIRHHpdmu4x5MYr6o3u4Y0duxG0fZk6AdTkM6aB1sTDEoj57DX+eomD/Uo/O7lBzRJfotCa/eD
KZqVvfSDfh60ejZNLJl307sKM2K2YfVbjZou99Qv/Z1oolejvdoCUuG+kE/kaHqFPpdl6pgXdzJ21xZ//HA79
Dw4qf3ZJmLyh/Uvk70kc1uuQKGx1DRScwSQYDqeve7GcCpEHRUAP4JVDJoJ5SR9OPs90bQppOsaiaZ9
DB6Fe6PqgMJ6X53IPvj8pQj+/C6Lrtm1gzQQiLUkJOOctbpmcRVj1u3Q4o9tRtL5MxcCMWLfKcObCCe
sCX3q0W9JaQB3z6M6SdtD4ukiok2fat7H3fHJxDxOL+MP2jFG/VTDAxsrjgHOLVGj8fKWQQ3ycobq6xXH
```

KljY4vIG9VPSbhivS4UPJBRGPndEGnhv+vv+SxqS7IR37EME1Adfi23xoYQ3Kof8T6hOBnJoESr4WNN4  
K8E0YgK3uyE23SVz4BuWdluwBUskVPObA7q5Ls9A4H/oGURsKdBeXgx+hXbkqUqtRn3mLED/PLwD4  
5TfiYma1VIMuzYSTQGviVJyrlTz6wGdB0nMKV8avDQgwk/cVNpgNA/iWq74+9Yigox3oo0xT3PNou4tnQ  
Mn1Gd0k2LXj7oOctlw7F2MqVcQTE/cxYdbln6UGsAATtvv1EgXkixFJZyHdyYp0eNGsD+U2xfewBzQXUH  
1TJENHhwU/n3igDtKhuHsdOyRUSmdc6oqe4G3IYu8wv4rHH6rHKc9J7D3YPWiZLwBZTJTIXIGAQ+bscc  
FdR6QFB26GvtLkL9Dbbn1V1f1RpHeqefS4JgCA155o6Er+K1EqRbzt9ZLvGaGHIb9KR0C1dfEnKd6p3vX  
AdJWEWKGSnumDQzCuFhWL4yefRXjXS2zPocDMv24rwkI9zoPQsJ/VXCEIQGZVual7rybfwjHqFak0vyv  
vq+EWq9unUXcxNbH7jJ7FD1wKcXRBgRdqrknOUiaExmNNDdORVbm+i3h0cYDi68ClmJCGeqv0HtkP  
TBkHk5uLYZmN8Vv/ZL2islu5JbeFfU4GK6DaKj/E3jeAssj+MhaTBOUCq92A+7DttSi9DZFG9dtuuhjFcaGl  
n56tldFxUxIbyoZ7MNEigisNntAZe3me+jiWZPexzTtYOxdHZ46fgr/oSyPk3xyM56n3/fbsZYIub9k3RxyFfBT  
WkvlwX4cYY38TM/4Yi5IPylg3oi8u928Fpv3/SIYBHFXFUZ2BHL7libBpZvbwT0H50Hm5dyKxUjue14tG0P  
gHXyYqvwMHBHszPxT5ijzvYbSzI9h3I4luXKjtLdYqEfZVsddeI2fU2gjdJLhEhqbWr6m3ebD9cl6vX72pHV7  
3Ez0taYlfbW4KTTf62A+X1IGrYWdlB9nFM/gzEHYt2QzDZi1UFw0d+8+VUFA3gZf8hq3hdv49r3tvxvIzBc  
OEZ+ufWorHFJ77sdN0OMfE6uftjKIEfDzGFWAsbfT3r9X33YUQDqSnAr9uv262mhudlm7wDuMH0PPEjx9  
7cgjZs9q9hQhBkKEeXoxrFdr5LCy7YfaJKC5G9v8ZLw+PdsgFU9kYjh89vj5XvgzBeek/ZQrvjnN5z2uL6aA  
3N5wrF4RWe+p1ZHgfYXnzpQIEyfrUrpXJQzLDcL6Y7Ve4KPz3rdROfzQtmhRCgJk04+6Cgqn2324zkZdd  
QqogzsJQRu14B1AiH8gAUcmkiQR0TyFnQEo5vPtsTYUDLfFd+AasTenMts//RjCu4ynMOPE/3mw7NgCsL  
z8C8/gNPzKuUMBCYhnDi/IDlp5bi8evgiupkvPVAUgOwB9dma9wsaFjpONBX/uuxWO/k38ZI0fuRTmvaZ  
wpoekGRWCNY+ompXN+XeoVzznRsgAbK+Rq6q2d2BsJq72rWUNbDRQQR89IRdViLsaTxXFKO4of4M  
5FXMG3nkwudADQxy2bRVj6fUcA3IO4YQ75CdQxBf0uMFBxkseXfSz98cvs9CHtxwiKWZZVdxttegMj7yrb  
nRCo0d6M6V0ma/E+J3Bx7+Auzqw4daAmMERXJmDxS8uP22kDmjX7zMzolqEAsMFQ1O25woZNV4n26  
NzCxT4EiB368ji8ZTfLekQyQb6o1MDm4aRJeB/dXuChgwrOww1ODLUI4bukSEkcooQA+aG9iFmhWHER/  
KmrJsoXEYidmCUQ+RGv6KJbRaxMjSW+/kB0gnkymF7JAnMu9wwT107TWeq9S6btuzTKUEip2mzInAL  
cN+NcRYQVMj/Z8CBXniBDFNRt20nkXf7wTJ7QKae9V6OFzdk6b/igQxsR4E4cOeV4oJbCbzRAa5HAQ2  
HoGDHp6OA509vi93yPK1UwTGUXt+dHJ8BPtetlwNvk1ljQ9mkBeHIE5kSmrTXwvR5YuqfdQwYWJlJ0ixv  
Kj73U5vbCr09mz5h6M/Cgx80GU9wzgN+bFujpB9m7vND1vsu+aC4vlgfiZlXv2kugYiCYtO5iMaz1MQfXy7a  
G4z6IYSUnLyYpyP9qK9T3/aa4gNW8fVluD+L2OzIIWxsGsUiprfwjS1XgSGDQz6iHRPKDMyO4Gd+xY1IW  
WPqwgltf9R/RHWuEjWtAlQLJG0QxLrXSZdwFWR+NC6te9coPHIKO8ILQH+Aj1xIC+nnXnHGAnpCJSh  
Yaexw8Kkycezfxyf+1rpF6TC14FlNyi9SxRGXvzoLCUdE7q7PPLTcEgg4dA3N7ZuNUS0Qnzc0WBZSxU  
ssurqy+SnE/SdUIFNu6C/PFfkPvJtTBDzFAehweyDX0iyzKI5i5+oR9erXbD7JYf3P2ZblxfDAA</xenc:Ciphe  
rValue></xenc:CipherData></xenc:EncryptedData>

## תוכן עניינים

5.....	1. מבוא לאבטחת מידע של שירותי האינטרנט	1
7.....	1.1 מבנה העבודה	1.1
8.....	2. סקירה של שירותים לאבטחת מידע של שירותי האינטרנט	2
9.....	2.1 דרישות לאבטחת מידע של שירותי האינטרנט	2.1
10.....	2.2 אימות	2.2
11.....	2.3 הרשאות	2.3
12.....	2.4 דוגמא של שירות האינטרנט 1	2.4
14.....	2.5 דוגמא של שירות האינטרנט 2	2.5
15.....	2.5.1 דרישות לאימות	2.5.1
16.....	2.5.2 הגנה על נתונים	2.5.2
17.....	2.5.3 הרשאות	2.5.3
18.....	3. SAML	3
18.....	3.1 OASIS	3.1
18.....	3.2 מה זה SAML?	3.2
20.....	3.3 היקף של SAML	3.3
22.....	3.4 טענות של SAML	3.4
24.....	3.5 הצהרה של אימות	3.5
24.....	3.6 הצהרה של תכונה	3.6
25.....	3.7 הצהרה של הרשאות	3.7
26.....	3.8 בקשה / תגובה של SAML	3.8
26.....	3.9 בקשה של SAML	3.9
27.....	3.10 תגובה של SAML	3.10
29.....	3.11 Bindings	3.11
29.....	3.12 SOAP Binding	3.12
29.....	3.13 פרופילים	3.13
30.....	3.14 SAML Artifact	3.14
31.....	3.15 SAML POST	3.15
32.....	4. תרחישים לשימוש ב SAML	4
32.....	4.1 תרחישים של Web Single Sign-On	4.1
35.....	4.2 תרחישים של Identity Federation	4.2
37.....	5. XACML	5
37.....	5.1 מבוא ל XACML	5.1
38.....	5.2 תרשים של ארכיטקטורה של XACML	5.2
40.....	5.3 מדיניות של XACML	5.3
41.....	5.4 דוגמאות של כללים של XACML	5.4
44.....	6. טכנולוגיית WIF	6
47.....	7. סיכום של מאמרים	7
47.....	7.1 תשתית של הרשאות בשירותי האינטרנט [1]	7.1
53.....	7.2 ניהול נאמנויות בשירותי האינטרנט [2]	7.2
53.....	7.3 הגדרה ו יישום של פרופיל של SAML-XACML לאינטראקציה של הרשאות על פני הרשת	7.3
57.....	7.4 Middleware ב OSG ו EGEE [3]	7.4
59.....	7.5 מדיניות של בקרת גישה בשירותי האינטרנט במערכת סיוע רפואי [4]	7.5
61.....	7.5.1 סקירה קצרה של מאמרים	7.5.1
61.....	7.5.2 מפרט ואימות של מדיניות הרשאות לקומפוזיציה של שירותי [5]	7.5.2
61.....	7.5.2 אבטחת מידע של הזמנות ביצוע לקומפוזיציה של שירותי האינטרנט [6]	7.5.2

7.5.3.	הארכה של מודל של הרשאות של XACML לתמיכה של טיפול בהתחייבויות של מדיניות ביישומים מופצים [7]	62
7.5.4.	הרשאות ופרטיות לשירותי האינטרנט סמנטיים [8]	63
8.	היישום שלי	64
9.	מסקנות	71
10.	סיכום	72
11.	קישורים	73
75.	נספח A. הגדרות	75
80.	נספח B. דוגמאות של תחביר של SAML	80
83.	נספח C. דוגמאות בקשה / תגובה של SAML	83
85.	נספח D. דוגמאות של תחביר של XACML	85
91.	נספח E. תרשים של היישום	91
92.	נספח F. נתונים מוצפנים של SAML ליישום	92

## תקציר

העבודה מתמקדת באימות של שירותי רשת באמצעות SAML ו XACML. המסמך מפרט דרישות לבדיקת אימות והרשאות של שירותי רשת וחוקר פתרונות קיימים של אימות לפי דרישות שהוגדרו. הוא מראה את חשיבות של אימות והרשאות בשירותי האינטרנט.

העבודה סוקרת את שירותי האינטרנט, מושגים וטכנולוגיות שמאחוריהם, ואבטחת מידע של שירותי אינטרנט. היא מציגה את הבעיות של אבטחת מידע ברשת האינטרנט: סודיות, אימות, אבטחת מידע ברשת, ומספק את הפתרונות לבעיות של אימות. המסמך מסביר מהו SAML, שהוא מפרט שמגדיר תחביר וסמנטיקה של הצהרות להעברה של הודעות מאובטחות. המסמך מספק תרחישים לשימוש ב SAML. המסמך מציג את XACML שמתאר כיצד להעריך את בקשת אימות והרשאות על פי כללי מדיניות. הוא מראה ארכיטקטורה של XACML ודוגמאות של כללים של XACML.

ראשית, המסמך סוקר מאמרים המתארים את אבטחת המידע של שירותי אינטרנט. המאמרים מציגים פתרונות פשוטים ומורכבים לבעיות של אימות והרשאות של שירותי האינטרנט. מאמרים מסוימים מראים את היישום של רשתות אימות של שירותי האינטרנט ומתארים ניסיון של יישומן. העבודות מספקות אבטחת מידע לתהליכים עסקיים שיתופיים באמצעות שירותי אינטרנט בסביבה הפתוחה. התרומה העיקרית של מאמרים אלה היא הצגה של המושגים הבסיסיים לאבטחת שירותי אינטרנט והצעה לארכיטקטורות עבור שירותי אינטרנט. המחברים מראים את חשיבותה של מדיניות של אימות והרשאות בשירותי אינטרנט. המאמרים מתארים מושגים וארכיטקטורה של התשתיות המוצעות וסוקרים היבטי יישום כגון אימות נתונים.

שנית המסמך מתאר מימוש של יישום בשימוש במסגרת של מיקרוסופט בשם WIF. המסמך מספק סקירה כללית על WIF, תשתית ליישום claim-based identity באפליקציות. הוא מראה תרחיש בסיסי עם שירותי אינטרנט ותרחיש בסיסי עם דפדפן אינטרנט. המסמך מציג פתרון שמשמש באסימון SAML עם טכנולוגיה Web API ASP.NET. הפתרון כולל שלושה יישומים: Web Client, שמתחבר לשירות האינטרנט, STS שמנפיק אסימוני SAML, SPI, שירות האינטרנט הדורש הזדהות של המשתמש לפני אימות המשתמש. צד לקוח שולח את הבקשה ל STS, STS קובע שתעודה של לקוח תקפה, ושולח את אסימון SAML ללקוח האינטרנט. הלקוח האינטרנט שומר SAML ב cookies ושולח הודעת בקשה ל ספק שירות; הודעת בקשה כוללת את האסימון SAML. ספק השירות נותן תוקף ל אסימון SAML ומעבד את הבקשה.

לצורך מימוש האפליקציה השתמשתי ב Microsoft Visual Studio 2013, Microsoft .NET 4.5.1 Framework.



האוניברסיטה הפתוחה  
המחלקה למתמטיקה ולמדעי המחשב

**אבטחה בשרותי אינטרנט -  
מיקוד ב SAML ו XACML**

עבודה מסכמת זו הוגשה כחלק מהדרישות לקבלת תואר  
M.Sc. "מוסמך למדעים" במדעי המחשב  
באוניברסיטה הפתוחה  
המחלקה למתמטיקה ומדעי המחשב

על-ידי  
**אנדריי קוגן**  
ת.ז. 314600230

העבודה הוכנה בהדרכתו של פרופ' אהוד גודס

פברואר 2015