Privacy enabled Web service access control using SAML and XACML for home automation gateways

Markus Jung
Institute of Computer Aided Automation
Vienna University of Technology
Email: mjung@auto.tuwien.ac.at

Georg Kienesberger
Institute of Computer Technology
Vienna University of Technology
Email: kienesberger@ict.tuwien.ac.at

Wolfgang Granzer
Institute of Computer Aided Automation
Vienna University of Technology
Email: wgranzer@auto.tuwien.ac.at

Martin Unger
Institute of Computer Aided Automation
Vienna University of Technology
Email: e0726109@student.tuwien.ac.at

Wolfgang Kastner
Institute of Computer Aided Automation
Vienna University of Technology
Email: k@auto.tuwien.ac.at

***Abstract*—A recent trend in home automation are gateways that offer a Web service based Application Programming Interface (API) to access an underlying home automation system. Due to the ease of use and the interoperability of Web services numerous use cases can be found for third party applications using such APIs. Smart homes allow to control nearly every aspect of living within a building, which also imposes great security and privacy concerns. Therefore this paper contributes a generic access control concept for Web service based APIs using the Security Assertion Markup Language and the Extensible Access Control Markup Language. This concept allows a user to securely authorize the access of third party applications to the home automation system in order to protect privacy and to ensure security. The access control concept is generic since no API change is required leaving the service provider and service consumer untouched.**

***Index Terms*—Home automation, Web services, access control.**

I. Introduction

Home automation gateways try to overcome the heterogeneity of devices found in today's dwellings. Providing access through the Internet to the home automation system, or interworking between different building automation technologies [1][2] like KNX, LonWorks, BACnet or ZigBee, may fall into the responsibility of a home automation gateway. The details of the home automation system are hidden by a Web service based API that offers a well-defined interface to create applications. Web services are a reasonable choice for such an API, due to interoperability to any target application platform and the industry-wide support that already exists for Web services. These APIs allow third-parties to provide applications for the different aspects of home automation like HVAC, lighting, shading, security, safety, entertainment and household devices. The applications can be custom made or offered through a marketplace concept to the customer. Figure 1 illustrates an overview of such a scenario.

Since these applications have a different purpose and scope, they only need access to a certain functionality of the home automation system. There are several use cases for applications that use such a Web service interface, like

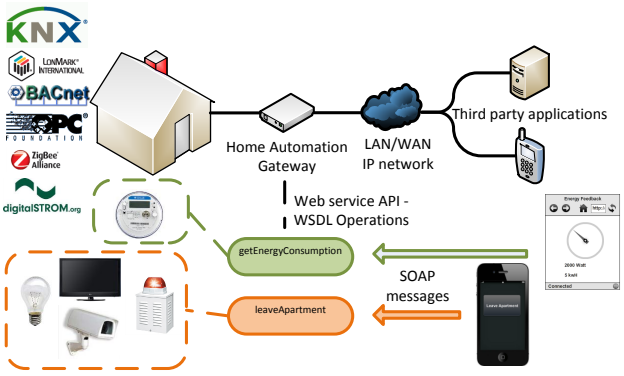

Fig. 1. Smart Home Use Case

- *Remote home control:* A mobile phone application that uses a wireless local area network and provides functionality to remotely control your home. This application could let you to switch lights and devices on and off and also provide other useful functions, such as a leaving appartment feature, that makes sure that all devices are switched off if you leave your home.
- *Smart grid demand side management:* You may grant access to your home automation system to your energy utility if you participate in a smart grid demand side management program. This includes access to smart meters or to your climate control or devices with a high demand side management potential [3].
- *Energy feedback:* An energy advisory service may be provided as a Web application hosted by your energy utility. In this case you just want to grant access to meter readings of your current energy consumption.

These examples illustrate that there are different use cases of such a Web service API used by third party applications with different security and privacy requirements. For all use cases the home automation gateway is the service provider

but the service consumer may range from a local smart phone application to a remote server based service consumer. The customer needs to control which service consumer is allowed to access certain functionalities. This problem is not only specific for the home automation area and there are already different approaches to solve it. This paper contributes a generic concept that allows a customer to control the access of each service consumer in a standardized way. The access control concept uses a generic policy that works on the structure of SOAP messages and on the well-defined Web service interface and is based on XACML [4]. A SOAP intermediary operates between client applications and the home automation gateway and enforces the generic policy, which uses XACML attributes carried with each Web service within the SAML [5] based access token. A simple example for a Web service based API in home automation is provided in Section II. Section III examines the privacy and security requirements that outline the need for access control in Web service based APIs. Related research work is provided in Section IV. The generic and reusable Web service access control concept is presented in Section V. A proof of concept implementation is the topic of Section VI. Finally, the results are summarized and a conclusion and outlook for further work are given in Section VII.

II. Web service based APIs in home automation

Web service based APIs hide the technical complexity of home automation and its technologies. The purpose of an API is to provide an easy to use and interoperable interface, which can be used to build applications. An API defines functions and data structures that allow to access an underlying system. With the rising popularity of Web services the term API can also be used to address Web service endpoints. First, the so called Web APIs describe how to interact with the service provider and which functionality is offered by Web service operations. Second, a definition for the exchanged data structures is given. Web services may be offered in two different architectural styles. The first option is to use SOAP (Simple Object Access Protocol) [6] for message exchange and the Extensible Markup Language (XML) to serialize the messages. SOAP based Web services allow the publishing of an interface description by using the Web Service Description Language (WSDL) [7]. As transport layer for SOAP messages the Hypertext Transfer Protocol (HTTP) can be taken, but other transports may also be possible. Another architectural approach are Web services that are based on the Representational State Transfer (REST) architecture. Central concepts of the RESTful approach are resources that share a uniform interface and a highly restricted set of operations (GET, PUT and POST) on these resources. [8] provides a detailed comparison between these two architectural approaches. The access control concept presented in this paper works for SOAP based Web services. Let us illustrate a home automation Web API by providing a WSDL description that offers two service operations. The first operation returns the current energy consumption of a complete household. The second operation deactivates light and consumer electronic devices and activates the alarm systems. The simple applications, sketched in Figure 1, that can be realized with these services, are either an energy feedback application that delivers information about the current energy consumption, or a "leave apartment" application. An excerpt of the WSDL document describing the interface is given in Listing 1. As a real world example for such a home automation gateway interface description the WSDL document of the open-source Digitalstrom server can be taken. Since the software server part of the PLC based home automation system [9] is open-source, it is freely accessible[1].

Listing 1. Home Automation API - Service Description (WSDL)

```
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/
    soap/" xmlns:tns="http://gateway.homeautomation.tuwien.
    ac.at/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://gateway.homeautomation.tuwien.
    ac.at/" name="APIService">
<types>
  <xsd:schema>
    <xsd:import namespace="http://gateway.homeautomation.
        tuwien.ac.at/" schemaLocation="http://localhost
        :8080/homeautomation?xsd=1"/>
  </xsd:schema>
</types>
<message name="leaveApartment">...</message>
<message name="leaveApartmentResponse">...</message>
<message name="getEnergyConsumption">...</message>
<message name="getEnergyConsumptionResponse">...</message>
<portType name="API">
  <operation name="leaveApartment">
    <input message="tns:leaveApartment"/>
    <output message="tns:leaveApartmentResponse"/>
  </operation>
  <operation name="getEnergyConsumption">
    <input message="tns:getEnergyConsumption"/>
    <output message="tns:getEnergyConsumptionResponse"/>
  </operation>
</portType>
<binding name="APIPortBinding" type="tns:API">...</binding>
<service name="APIService">...</service>
</definitions>
```

For the encoding *document/literal* is used, therefore the input and output messages of the defined operations are represented through single XML fragments that wrap up all parameters. The schema for the used types is kept simple for this example and given in Listing 2.

Listing 2. Home Automation API - Types

```
<xs:schema xmlns:tns="http://gateway.homeautomation.tuwien.
    ac.at/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0" targetNamespace="http://gateway.
    homeautomation.tuwien.ac.at/">
  <xs:element name="getEnergyConsumption" type="tns:
      getEnergyConsumption"/>
  <xs:element name="getEnergyConsumptionResponse" type="tns:
      getEnergyConsumptionResponse"/>
  <xs:element name="leaveApartment" type="tns:leaveApartment
      "/>
  <xs:element name="leaveApartmentResponse" type="tns:
      leaveApartmentResponse"/>
  <xs:complexType name="getEnergyConsumption">
    <xs:sequence/>
  </xs:complexType>
  <xs:complexType name="getEnergyConsumptionResponse">
    <xs:sequence>
      <xs:element name="return" type="tns:energyConsumption"
          minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
```

[1] http://developer.digitalstrom.org/download/dss/

```
  <xs:complexType name="energyConsumption">
    <xs:sequence>
      <xs:element name="energy" type="xs:double"/>
      <xs:element name="power" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="leaveApartment">
    <xs:sequence/>
  </xs:complexType>
  <xs:complexType name="leaveApartmentResponse">
    <xs:sequence/>
  </xs:complexType>
</xs:schema>
```

The SOAP message to retrieve the current energy consumption is shown in Listing 3. By using the *document/literal* encoding, the message can be directly assigned to an operation due to the usage of the type *getEnergyConsumption* within the SOAP body.

Listing 3. Energy consumption - SOAP message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
    soap/envelope/" xmlns:gat="http://gateway.
    homeautomation.tuwien.ac.at/">
  <soapenv:Header/>
  <soapenv:Body>
    <gat:getEnergyConsumption/>
  </soapenv:Body>
</soapenv:Envelope>
```

III. Privacy and Security Requirements

As already mentioned, in a smart home hardly any aspect of living is left completely untouched. The numerous deployed sensors and actuators range from simple temperature sensors and light switches to full-blown surveillance cameras and automatic door locks. Nevertheless, it is still a home and therefore a private space, where people expect privacy.

The taxonomy presented in [10] has four general categories of activities where privacy violations might occur: the collection, processing, and dissemination of information related to a person, as well as directly affecting somebody's life.

When connecting a home automation gateway to external applications, obviously, information on many different aspects of a person may potentially be collected, processed, and in consequence also disseminated. Additionally, a person's personal space may also be directly intruded by using the actuators deployed within the building. Since people do not only have different conceptions of their private sphere, but also varying needs which they have to balance against privacy considerations, the same specific activity that is still acceptable for one person, might be totally unacceptable for another. Hence, there cannot be a single privacy policy which is suitable for everyone. Instead it has to be possible for each user to define his own privacy policy.

This implies to be able to control, as fine-grained as possible, not only what data or functionality of the home automation gateway may be used by external applications, but also which specific application is allowed to access it. Taking the use cases presented in Section I, one may allow an application working locally on a smart phone to control all devices as long as the information gathered from sensors is kept local and no external control of devices is possible. In contrast, an energy advisory application, running externally and accessing your home automation sensors through the internet, should only have read access to the meter values gathered by the smart meter. The application must not have access to any actuators or other sensors.

When it comes to security, requirements emerging directly from the home automation system have to be taken into account first. Home automation devices with restricted access, for instance, those related to physical security, naturally also must have access restrictions regarding external applications. However, this again boils down to the requirement that it must be possible to restrict access in a fine-grained manner, as it is the case with the privacy aspects.

Second, the need to enforce the user's access policy and prevent unauthorized or malicious access, leads to indirect security requirements. Without secure authentication between the application and the home automation gateway, it cannot be made sure that the application requesting access is indeed the one it claims to be, effectively rendering any access policy effectless. Moreover, the exchanged messages' integrity has to be ensured to prevent alteration, and finally, the secrecy of the transmitted information has to be guaranteed in order to ensure privacy.

IV. Related Work

The Security Markup Language (SAML) is an XML based framework for exchanging security information between online-business partners [11]. The main use cases for SAML are single sign-on, federated identity and the use within Web services and other industry standards. The core elements of SAML are assertions which are built of authentication statements, attribute statements and authorization decision statements. The participants involved in a SAML interaction are a SAML asserting party and a SAML relying party. A SAML assertion issued by the asserting party targets a subject that could be any kind of entity, for example a human being, a computer or an application. The relying party that receives the assertion within an interaction uses the asserted information for access decisions. An assertion contains the information of how a subject has been authenticated and additional information about the authentication method. Furthermore, attributes and authorization decision statements can be added. The authorization decision statements define what the subject is allowed to access. The SAML standard has frozen this element in the latest specification and suggests the use of Extensible Access Control Language (XACML) instead.

XACML is an access control language based on XML. It defines a policy language and furthermore provides a request/response model for access decisions. The policy language model consists of the elements rule, policy and policy sets, which contain further sub-elements to provide an expressive language to specify policies. The other part of XACML is the data-flow model, which describes how policy enforcement points can issue a policy decision request to the policy decision point using the context handler that transforms native decision requests to the XACML canonical form.

In [12], the first proposal of a fine-grained access control concept that leverages the structure of SOAP messages is presented. It operates on the encoding style *rpc/encoded* and defines custom XML structures to identify subjects and policies. XML structures are attached to the SOAP header that act as access tokens. This information is used to identify certain subjects characterized by the fixed set of attributes *user-id*, *IP-address*, *sym-address* and *role-id*. This subject information is then used by a SOAP gateway to make access decisions by using a policy that is specified in a custom defined XML language.

Yin *et al.* [13] present a SAML/XACML access control between Portal and Web services. A SAML assertion is used to identify a subject and transferred by using WS-Security within the SOAP header. XACML is used to make an access decision based on the authorization of the subject stored in a central policy decision point. In the concept presented in this paper the use of a central repository is avoided and all required information to authorize a request, is carried within the SOAP request itself.

An access control framework based on SAML and XACML is described in [14]. It takes context-awareness into account using a role based access control concept. The framework follows the complete XACML data flow model to decide upon an authorization request. The work is based on the models presented in [15] and [16] and targets other use-cases rather than our solution.

OAuth 2.0 [17], which is under standardization at the moment, is an authorization protocol that enables a third-party limited access to an API that is based on HTTP on behalf of a resource owner. Different so-called flow models are part of the specification that reflect the different client types that may interact within the authorization protocol. OAuth 2.0 only targets the HTTP protocol for carrying an access token. In our approach we want to stick to a pure SOAP approach to keep our concept independent of the transport layer. How the access token is serialized and what information is carried in it is also not in the scope of OAuth 2.0.

V. Privacy Enabled Web Service Access Control

The contributions of this paper are:

- A privacy preserving API authorization mechanism for home automation gateways.
- Access tokens containing the enabled SOAP operations according to the WSDL document of the Web service API. Carrying the enabled SOAP operations within the access token allows to decide an access request based on the access token and the current SOAP request without the need for a central repository.
- A standardized way of building access tokens using SAML and XACML and integration into the SOAP header.

These access tokens can be used by the service consumer of a Web service and checked by a generic access control component that is realized as SOAP intermediary. The concept is built of two major parts which are independent of each other. First, there is the token acquisition part which defines how an access token is created by the user and how the service consumer obtains the access token. Second, there is the access control enforcement in which the access token is used by the service consumer and provided for each SOAP request within the SOAP header. A SOAP intermediary can then use this token to decide if the request is authorized or not. The access control enforcement does not depend on the way in which the token is acquired. While the token acquisition addresses the privacy requirement of our concept, the access control enforcement addresses the security requirements. The whole concept is secured by using secured channels for communication (e.g. HTTPS or WS-Security) and using a Web of trust based on public and private key based encryption for mutual authentication and to sign exchanged messages. An overview of this concept is given in Figure 2. The interaction between the different components and a detailed description of each component is given throughout the rest of this section.

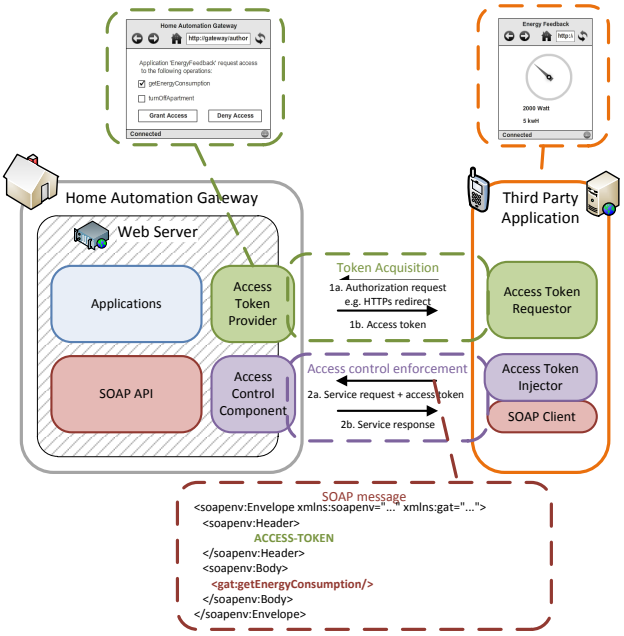

Fig. 2. Architecture Overview

A. Interaction overview

With the first usage of a third-party application, an access token has to be obtained. Take for example a browser based energy advisory application that provides energy feedback to a user and runs on a Web server at the energy utility's site. The Web server acts as a service consumer and wants to access the Web service API provided by the home automation gateway through the Internet. The service consumer sends an authorization request to the home automation gateway Web interface by doing a browser redirect containing the desired SOAP operations that should be enabled (1a.). The browser of the user is redirected by using a secure channel, e.g. a

HTTPS redirect. The user is then authenticated at the access token provider component and authorizes the requested SOAP operations. In the example provided in Figure 2 the *getEnergyConsumption* operation is requested. The user chooses the data and functionality that should be enabled for the third-party application and an access token is created based on the decision and signed with the public key of the home automation gateway. The browser is then redirected back with the access token as payload to the third-party application (1b.). This is a possible data flow for Web based applications. In the case of a native client application, for example a smart phone application, a platform specific secure development kit would be required (provided by the home automation gateway provider), since the user authentication and the service consumer authorization have to happen in a secure environment. This access token is then used for all further SOAP requests to the API provided by the home automation gateway acting as service provider (2a. and 2b.).

B. Access Token Provider

The access token provider component offers a Web based user interface to specify an access policy for an application. A generic user interface could work directly on the operations of a Web service defined within a WSDL document. The meanings of the SOAP operations in the provided example in Listing 1 are obvious, but the exposure of technical details should not overwhelm the customer. For a complex API with manifold operations a high-level mapping to domain specific terms that group multiple operations is necessary.

C. Access token

The access token is provided as SAML assertion containing attributes according to the SAML profile of XACML [18] identifying the enabled SOAP operations. The subject of this token is the application, especially a unique identifier specific to the application that is used by the user. Listing 4 provides an example of such an access token. The *saml:Subject* contains the hash of the public key of the client application, which is used to uniquely identify an application. The *saml:AttributeStatement* contains a single *saml:Attribute* allowing the SOAP operation *getEnergyConsumption* for the *energyFeedbackApplication*. With this access token the client application is only allowed to use this single SOAP operation. If more operations are allowed multiple *saml:Attribute* elements would be listed.

Listing 4. Access token

```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:
    assertion"
     ID="Assertion-1" IssueInstant="2011-10-12T12:56:55.813
          Z" Version="2.0">
      <saml:Issuer Format="urn:oasis:names:tc:SAML:1.1:
          nameid-format:X509SubjectName">
           homeAutomationGateway@localdomain
      </saml:Issuer>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/
          xmldsig#">
      <ds:SignedInfo>
      <ds:CanonicalizationMethod
          Algorithm="http://www.w3.org/2001/10/xml-exc-
               c14n#" />
          <ds:SignatureMethod
               Algorithm="http://www.w3.org/2000/09/
                    xmldsig#dsa-sha1" />
          <ds:Reference URI="#Assertion-1">
          <ds:Transforms>
          <ds:Transform
               Algorithm="http://www.w3.org/2000/09/
                    xmldsig#enveloped-signature" />
          <ds:Transform Algorithm="http://www.w3.org
               /2001/10/xml-exc-c14n#" />
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org
               /2000/09/xmldsig#sha1" />
          <ds:DigestValue>2v+QxmwdoCR4IrujyYbxe6lw6mQ=</ds
               :DigestValue>
          </ds:Reference>
          </ds:SignedInfo>
          <ds:SignatureValue>dXn5xvrSwv/
               MCeecrfB7iR6tpAdudpQKTn6E3bzvhYK8GniBRbBuWQ
               ==</ds:SignatureValue>
          <ds:KeyInfo />
      </ds:Signature>
      <saml:Subject>
          <saml:NameID
               Format="urn:oasis:names:tc:SAML:1.1:nameid
                    -format:unspecified">
               a61881c846de353b2c1b8243dbf01f2f623d3ba3
          </saml:NameID>
      </saml:Subject>
      <saml:AttributeStatement>
          <saml:Attribute
               xmlns:xacmlprof="urn:oasis:names:tc:SAML
                    :2.0:profiles:attribute:XACML"
               xacmlprof:DataType="http://www.w3.org
                    /2001/XMLSchema#string"
               NameFormat="http://tuwien.ac.at/
                    accessControlComponent/attr-formats"
               Name="EnabledSoapOperation" FriendlyName="
                    EnabledSoapOperation">
               <saml:AttributeValue xsi:type="xs:string">
                    getEnergyConsumption</saml:
                    AttributeValue>
          </saml:Attribute>
      </saml:AttributeStatement>
</saml:Assertion>
```

To protect the token from modifications it is signed with the private key of the home automation system according to the XML signature standard [19] in the *ds:Signature* element. The signature preserves the integrity of the token. Inception or replay attacks need to be prevented which is done by signing each SOAP message sent by the service consumer with its private key or using secure communication channels, like those provided through WS-Security. Furthermore, to enhance security the validity of the token can be restricted to a certain amount of time. See [20] for a complete analysis of security threats to SAML assertions and Section VI for a security evaluation of our concept.

D. Access token injector

The SOAP client acting as a service consumer must attach the access token to each service request. A generic access token injector component can handle this by adding the token to the SOAP header. By using the SAML token profile of the WS-Security standard, the access token can be carried in a standardized way. The access token injector can be registered as a handler in the SOAP handler chain of the used Web service framework. In that way an already existing SOAP client can be left unchanged.

E. Access control component

The access control component acts as a SOAP intermediary and processes all SOAP requests to the home automation gateway. The access control component performs multiple roles from the viewport of XACML as visualized in Figure 3. It acts as a policy enforcement point (PEP), context handler, policy decision point (PDP) and policy administration point (PAP). A generic XACML policy can be re-used for any kind of SOAP based Web service as all the information required to decide the access requests is available within the SOAP request. No domain specific attributes or resources exist, so the policy can be reused for different use cases. Listing 5, again outlines the fact that all information required to decide whether the access is allowed or not is represented in the header of the according SOAP message. The SAML attributes that are mapped to XACML list the enabled SOAP operations.

Listing 5. Energy consumption - SOAP message including access token

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
    soap/envelope/" xmlns:gat="http://gateway.
    homeautomation.tuwien.ac.at/">
  <soapenv:Header>
    <wsse:Security>
     <saml:Assertion xmlns:saml=urn:oasis:names:tc:SAML
          :2.0:assertion Version="2.0" IssueInstant="
          2011-08-22T12:00:00Z">
          ...
          <saml:AttributeValue xsi:type="xs:string">
               getEnergyConsumption</saml:AttributeValue>
          ...
      </saml:Assertion>
    </wsse:Security>
  <soapenv:Body>
    <gat:getEnergyConsumption/>
  </soapenv:Body>
</soapenv:Envelope>
```

The context handler performs a major role since it creates a XACML decision request based on the native access request. A XACML policy information point is not required since all required attributes are contained in the SAML assertion available in the SOAP header of the request. The resource is represented through the SOAP operation that should be performed.

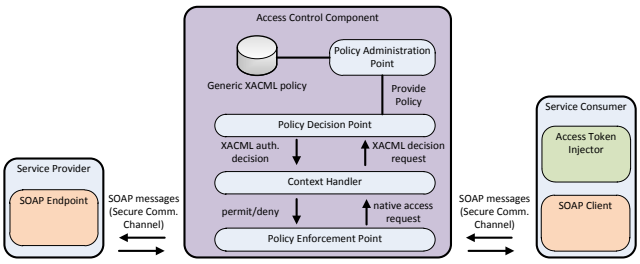

Fig. 3. Access Control Component

The PEP first validates the integrity of the request, which is possible since the SAML assertion is signed by the home automation gateway. Since the access control component and the access token provider reside on the same home automation gateway they share the same public/private key pair. To ensure that the access token is not used by another client application the request shall also be signed with the client application's private key. After the integrity of the message is ensured, the XACML policy decision request is created by taking the SOAP operation as a resource, the client application as subject, and the enabled SOAP operation as attributes for the subject. Listing 6 shows an example XACML decision request that is passed by the PEP to the PDP. Since both components are running in the same process, no inter-process communication is required.

Listing 6. XACML decision request

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:
    os">
   <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:SAML:1.1:
          nameid-format:unspecified" DataType="http://www.
          w3.org/2001/XMLSchema#string">
         <AttributeValue>
             a61881c846de353b2c1b8243dbf01f2f623d3ba3</
             AttributeValue>
      </Attribute>
      <Attribute AttributeId="EnabledSoapOperation" DataType
          ="http://www.w3.org/2001/XMLSchema#string">
         <AttributeValue>getEnergyConsumption</
             AttributeValue>
         <AttributeValue>leaveApartment</AttributeValue>
      </Attribute>
   </Subject>
   <Resource>
      <Attribute AttributeId="RequestedSoapOperation"
          DataType="http://www.w3.org/2001/XMLSchema#string
          ">
         <AttributeValue>getEnergyConsumption</
             AttributeValue>
      </Attribute>
   </Resource>
   <Action>
      <Attribute AttributeId="SoapAction" DataType="http://
          www.w3.org/2001/XMLSchema#string">
         <AttributeValue>Execute</AttributeValue>
      </Attribute>
   </Action>
   <Environment/>
</Request>
```

The PDP then uses the generic policy given in Listing 7 to decide whether to permit or deny the access request.

Listing 7. Generic XACML policy

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
    xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy
        :schema:os
    http://docs.oasis-open.org/xacml/2.0/access_control-
        xacml-2.0-context-schema-os.xsd"
    PolicyId="urn:oasis:names:tc:xacml:2.0:conformance-test
        :IIA1:policy"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
        combining-algorithm:first-applicable">
   <Description>
      Generic policy that evaluates the access to a SOAP
          based Web service API using the XACML attributes
          that specify that enabled operations.
   </Description>
   <Target/>
   <Rule
       RuleId="EnabledOperation"
       Effect="Permit">
      <Description>
        The requested SOAP operation is in the list of
            enabled SOAP operations.
      </Description>
      <Condition>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
            function:string-is-in">
```

```
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
            function:string-one-and-only">
          <ResourceAttributeDesignator AttributeId="
              RequestedSoapOperation" DataType="http://
              www.w3.org/2001/XMLSchema#string" />
        </Apply>
        <SubjectAttributeDesignator AttributeId="
            EnabledSoapOperation" DataType="http://www.w3
            .org/2001/XMLSchema#string" />
      </Apply>
    </Condition>
  </Rule>
  <Rule
      RuleId="NotEnabledOperation"
      Effect="Deny">
    <Description>
      SOAP operation is not enabeld for the service
          consumer.
    </Description>
  </Rule>
</Policy>
```

F. Service provider

The service provider and the SOAP services are not affected by this generic access protection. Since the access control component acts as a SOAP intermediary, the complete concept is transparent to the service provider. Additionally the business logic offered by the service does not need to be bloated with a token parameter which has to be passed with every request.

VI. Proof of Concept

A. Implementation

The implementation of the generic access control concept as a proof of concept is done in Java using the Metro Web service stack[2]. As a reference Web service the SOAP API definition provided by the open-source Digitalstrom server[3] is taken, which acts as a home automation gateway for the Digitalstrom home automation system [9] as mentioned in Section I. For the implementation we focused on the access control component by using a mockup for the service provider and the service consumer. The access control component runs on a SOAP intermediary powered by the Membrane framework[4] for Java. In a real world scenario the access control component resides directly on the home automation gateway. For the policy decision point we relied on a third-party open-source implementation[5] to prove the standard conformance of our policy and policy decision request. The implementation successfully proved the access control functionality of the presented concept and the implementation will be further enhanced to also include the generic access policy creation component that leverages the privacy experience of the home automation system.

B. Packet size evaluation

Figure 4 shows the impact of the attached SAML access token regarding the SOAP payload of a request to the different SOAP operations provided by the home automation gateway. In the average case the message size per call increases from 750 Bytes to 14 KBytes. For the evaluation a mocked Web service API based on the open-source Digitalstrom WSDL document has been used.

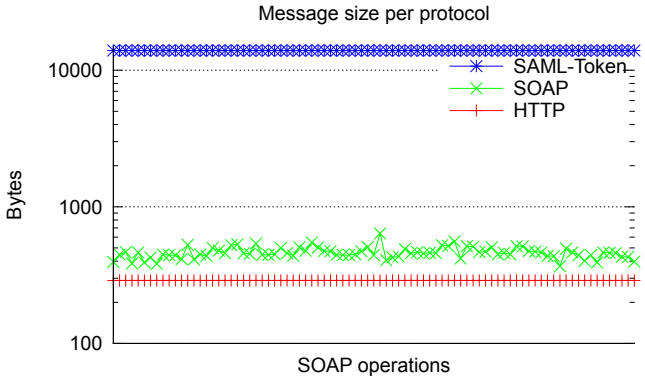

Fig. 4. Packet Size

C. Security evaluation

The following list evaluates the concept against various security requirements:

- **Authentication:** The user establishes the trust to a third party application. The creation of an access token that is used further for authentication and authorization by the service consumer, is triggered by the user. So at that point a **Web of trust** is created since the private key of the home automation gateway signs the access token that contains the hash of the public key of the service consumer. All SOAP requests of the service consumer are signed with its private key and its public key is also attached to each request. Replay attacks can be prevented by adding a unique message identifier to each message.
- **Confidentiality:** Confidentiality is not provided by the concept, but could easily be extended if the whole access token is encrypted using the private key of the home automation gateway. Furthermore the usage of a secure communication channel can also provide confidentiality.
- **Data integrity:** Data integrity is ensured for the access token, since the home automation gateway signs the token using its private key. The data integrity for the SOAP messages using this access token is provided by the fact that the service consumer signs each message with its private key.

D. Lessons learned and outlook

The implementation of the concept shows that there is a great lack of industry-ready implementations of the standards used in this concept. While there is a Java open-source implementation for SAML available, the open-source project for XACML lacks a big community. The packet size evaluation of the performed experiments shows that the mechanism is quite costly and further proof is needed to see if it is possible to deploy it on an embedded home automation system. The next step will be to extend our proof of concept with the

[2]http://metro.java.net
[3]http://developer.digitalstrom.org/download/dss
[4]http://www.membrane-soa.org/soap-monitor
[5]http://code.google.com/p/enterprise-java-xacml/

token acquisition functionality and to deploy it on an actual home automation gateway, to get some real world performance results. However, since the access control concept is generic, it can also be applied in non-embedded areas to provide a fine-grained access control for SOAP based Web service APIs.

VII. Conclusion

This paper presented a generic and reusable access control concept for SOAP based APIs that leverages privacy and security for home automation gateways. Through the application of our concept, the privacy of a user is improved since there is full control over the authorization of access to data and functionality provided through the home automation systems. The same is true for security concerns. By using the fine-grained access control mechanism, it is possible to reduce the risk of the access of third party applications with malicious intent. Therefore the security and privacy requirements for the use cases *remote home control*, *smart grid demand side management* and *energy feedback* can be fulfilled. The access control concept is generic and reusable since it grants access to SOAP operations to subjects and the policy can be re-used for any kind of SOAP based APIs. Our implementation of the access control component can be placed in front of every kind of SOAP Web service as a SOAP intermediary. This opens the application to other use cases where only security or business concerns are an issue. As further work we consider:

- The proof of concept is extended to operate on a real world home automation gateway to allow a complete concept evaluation and performance measurements.
- The extension of the generic access token to reference the information model used by the Web service should allow the protection of data-centric Web services that use generic SOAP operations.
- Security and privacy for Web service based data exchange in smart grids can be improved by using the generic access control concept [21].

Acknowledgment

Specials thanks to Martin Unger who implemented most parts of the presented concept as a proof of concept and as part of his bachelor thesis. This work has been partially funded by the *Austrian Climate and Energy Fund* within the programme *New Energies 2020* and the project *Smart Web Grid (P 829 902)*.

References

[1] W. Kastner, G. Neugschwandtner, S. Soucek, and H. Newmann, "Communication systems for building automation and control," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, 2005.

[2] W. Granzer, W. Kastner, and C. Reinisch, "Gateway-free Integration of BACnet and KNX using Multi-Protocol Devices," in *Proc. 6th IEEE International Conference on Industrial Informatics (INDIN '08)*, 2008, pp. 973–978.

[3] K. Wacks, "Utility load management using home automation," *Consumer Electronics, IEEE Transactions on*, vol. 37, no. 2, pp. 168–174, 1991.

[4] T. Moses *et al.*, "Extensible access control markup language (XACML) version 2.0," *Oasis Standard*, vol. 200502, 2005.

[5] S. Cantor, J. Kemp, R. Philpott, and E. Maler, "Assertions and protocols for the oasis security assertion markup language (SAML) v2. 0. oasis standard saml-core-2.0-os," 2005.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1, 2000," *W3C Note*.

[7] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana *et al.*, "Web services description language (WSDL) 1.1," 2001.

[8] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services: making the right architectural decision," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 805–814. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367606

[9] G. Dickmann, "Digitalstrom: A centralized PLC topology for home automation and energy management," in *Power Line Communications and Its Applications (ISPLC), 2011 IEEE International Symposium on*, 2011, pp. 352–357.

[10] D. J. Solove, "A taxonomy of privacy," *University of Pennsylvania Law Review, Vol. 154, No. 3, p. 477*, 2006.

[11] J. Hughes and E. Maler, "Security assertion markup language (SAML) v2. 0 technical overview," *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, 2005.

[12] E. Damiani, S. di Vimercati, S. Paraboschi, and P. Samarati, "Fine grained access control for SOAP e-services," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 504–513.

[13] H. Yin, J. Zhou, H. Wu, and L. Yu, "A SAML/XACML based access control between portal and web services," in *Data, Privacy, and E-Commerce, 2007. ISDPE 2007. The First International Symposium on*. IEEE, 2007, pp. 356–360.

[14] M. Sharifi, H. Movahednejad, S. G. H. Tabatabei, and S. Ibrahim, "An effective access control approach to support web service security," in *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, 2009, pp. 529–535.

[15] C. Shang, Z. Yang, Q. Liu, and C. Zhao, "A context based dynamic access control model for Web service," in *Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on*, vol. 2. IEEE, 2008, pp. 339–343.

[16] H. Tao, "A XACML-based access control model for web service," in *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, vol. 2. IEEE, 2005, pp. 1140–1144.

[17] E. Hammer-Lahav, D. Recordon, and D. Hardt, "The OAuth 2.0 authorization protocol," *IETF Draft. February*, 2011.

[18] A. Anderson and H. Lockhart, "SAML 2.0 profile of XACML," *OASIS Committee Draft*, vol. 2, no. 11, 2004.

[19] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, "XML-signature syntax and processing," *W3C recommendation*, vol. 12, p. 2002, 2002.

[20] E. Maler, C. Cahill, A. Hughes, A. Origin, M. Beach, B. Metz, B. Hamilton, R. Randall, T. Wisniewski, E. Reid *et al.*, "Security and privacy considerations for the oasis security assertion markup language (saml) v2. 0," *Language (SAML)*, vol. 2, 2005.

[21] G. Kienesberger, M. Meisel, and A. Adegbite, "A comprehensive information platform for the smart grid," in *Proceedings of IEEE AFRICON 2011*, to appear.