*NORDUGRID*

---

NORDUGRID-TECH-16
6/8/08

# SECURITY FRAMEWORK OF ARC1

W.Qiang, A.Konstantinov

*Abstract*

This document is about security design concerns and ideas, as well as security framework implementation in ARC1 middleware.

# Table of Contents


1. Introduction....3
2. Security architecture in HED. SecHandler and PDP....3
    2.1. Structure of SecHandler and PDP....3
    2.2. Interface of SecHandler....5
    2.3. Interface of PDP....5
3. Policy Evaluation Engine....6
    3.1. Design of policy evaluation engine....6
    3.2. Schemas for policy evaluation engine....7
    3.3. Interface for using the policy evaluation engine....7
4. Policy Decision Service....7
5. Security Attributes....8
    5.1. Infrastructure....8
    5.2. Available collectors....8
        5.2.1. TCP....8
        5.2.2. TLS....8
        5.2.3. HTTP....9
        5.2.4. SOAP....9
6. Delegation Restrictions....9
    6.1. Delegation Architecture....9
    6.2. Delegation Collector....9
    6.3. Delegation PDP....10
    6.4. Delegation interface....10
7. Schemas, descriptions and examples....10
    7.1. Authorization Policy....10
    7.2. Authorization Request....10
    7.3. Authorization Response....10
    7.4. Interface of policy decision service....10
    7.5. Configuration of PDP service....10
    7.6. SimpleListPDP configuration and Policy Example....11
    7.7. ArcPDP configuration and Policy Example....11
    7.8. PDPServiceInvoker configuration....13
    7.9. DelegationPDP configuration....13
    7.10. Delegation SecHandler Configuration....13
    7.11. UsernameToken SecHandler configuration....14
8. User Manual....14
    8.1. Authorization SecHandler and PDPs....14
    8.2. Delegation SecHandler, DelegationPDP and Proxy Certificate Generation....15
    8.3. UsernameToken SecHandler....15

# 1. Introduction

The security framework of ARC1 includes two parts of capabilities: security capability embedded in hosting environment, and security capability implemented as plug-ins with well-defined interfaces which can be accessed by hosting environment and applications. The following design concerns were employed when designing:

- Interoperability and standardization

In consistent with the main design concern of ARC1, interoperability and standardization is considered in security framework. For example, in terms of authentication, PKI infrastructure and proxy certificate (RFC3820 14) is used as most of the other grid middle-wares do. Since supporting of standardization is a way for implementing interoperability, some standard specifications have been implemented as prototype and tested, such as SAML specification.

- Modularity and extensibility

Besides the security functionality which is embedded in hosting environment, the other security functionality is implemented as plug-ins which has well-defined interfaces, and is configurable and dynamically loadable. Since the interoperation interface between security plug-in and hosting environment or applications is predefined, it is easy to extend the security functionality in order to support some other security capability by implementing the interface.

- Backward compatibility

# 2. Security architecture in HED. SecHandler and PDP

## 2.1. Structure of SecHandler and PDP

In the implementation of ARC1, there is a service Container – the Hosting Environment Daemon (HED) (D1.2-2) which provides a hosting place for various services in application level, as well as a flexible and efficient communication mechanism for services which reside in the hosting environment.

HED contains a framework for implementing and enforcing authentication and authorization. Each message chain component (MCC) or service has a common interface which is in responsible for calling various authentication and authorization functionality. This interface is functional in some plug-ins called SecHandler. Each component (MCC or service) could have two queues of SecHandlers independently for incoming message and outgoing message, both of which will be sequentially executed and return the result.

Besides the pluggable characteristic of message chain components of HED or service, the SecHandler is pluggable and can be configured by using configuration file as well. Under the SecHandler there could be some pluggable and configurable sub-modules which specifically handle various security functionalities, such as authorization, authentication, etc. The current implemented sub-modules under SecHandler are mostly for authorization, such as arcpdp which is a policy decision point for ARC specific request and policy schema; but there is a sub-module coming with message level authentication based on WS-Security Username-Token profile. Figure 1 gives the structure of a MCC/Service, and the message sequence inside it.

Figure 2 shows the configuration about SecHandler for an example service called “Echo” service.

```
<Service name="echo" id="echo">
  <SecHandler name="identity.map" id="map" event="incoming">
    <PDP name="allow.pdp"><LocalName>test</LocalName></PDP>
  </SecHandler>
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="arc.pdp">
      <PolicyStore>
        <Location type="file">policy.xml</Location>
        <!-- other policy location-->
      </PolicyStore>
    </PDP>
    <PDP name="simplelist.pdp" location="pemittedlist.txt"/>
  </SecHandler>
```

*Figure 1. There are two chains of security handler inside the MCC or service. Each security handler will parse the security attributes which are generated by the upstream MCC/services or probably upstream security handlers in the same MCC/Service, and or authenticate or authorize the incoming/outgoing target based on the security attributes. The security handler could also change the payload itself, for example, the username token handler will insert the username token into header part of SOAP which is the payload of SOAP MCC. Policy decision point (PDP) is called by security handler and is supposed to make authorization decision.*

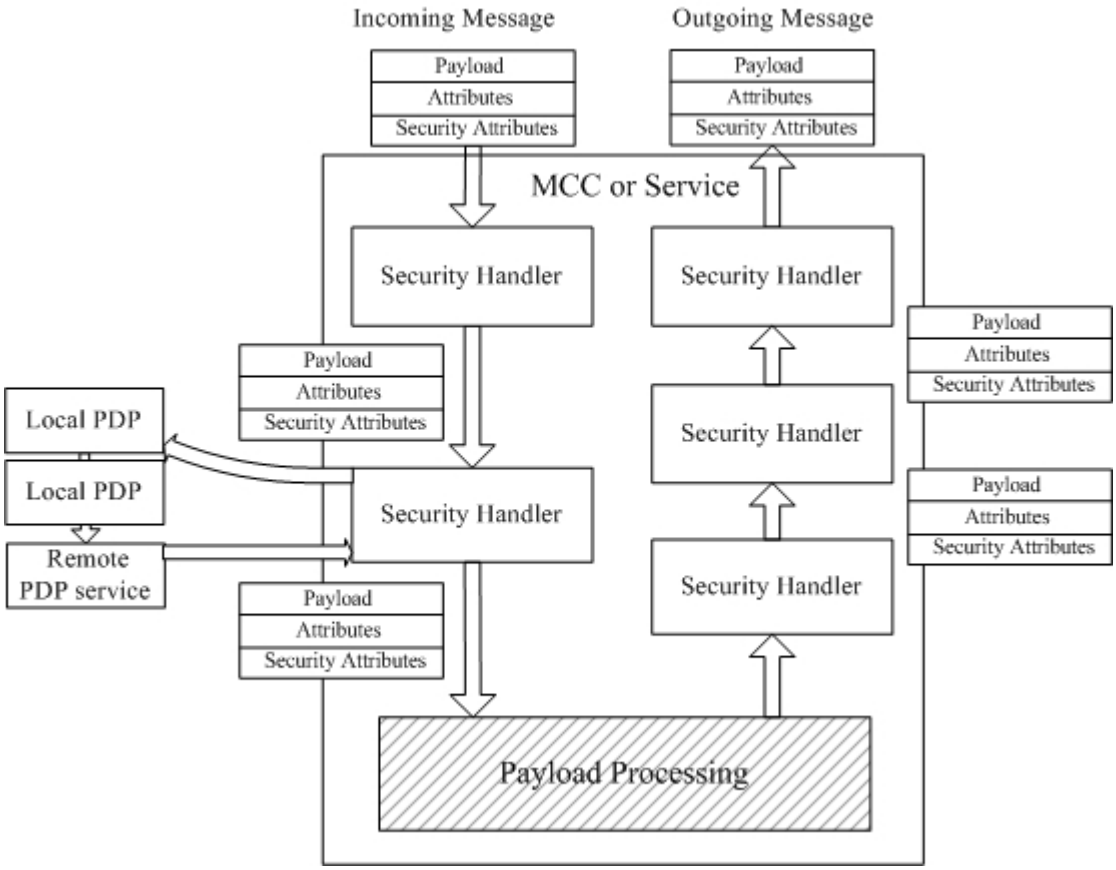


*Figure 2. Echo service includes two SecHandlers, both of which are responsible for authorization. One SecHandler includes simple gridmapfile-like policy decision point (PDP) which will map the identity in the incoming message into local identity like local linux user name. The other includes two PDPs: one will compose ARC specific request based on the security attribute parsed from "incoming" message and evaluate against the ARC specific authorization policy which is specified as "policy.xml"; the other will compare the identity in the incoming message against an identity list which includes the permitted identities.*

### 2.2. Interface of SecHandler

When one component (MCC or service) is loaded according to the configuration information, the SecHandler under the component and the plug-ins like PDP which are attached to the SecHandler will be loaded as well.

There is one simple interface (see Figure 3) defined in class SecHandler, which will be called by the containing MCC/Service once there is message (incoming or outgoing) need to be processed.

```
class SecHandler {
 public:
  SecHandler(Arc::Config*) {};
  virtual ~SecHandler() {};
  virtual bool Handle(Arc::Message *msg) = 0;
};
```

*Figure 3. class SecHandler is an abstract class which includes a general interface called Handle which uses Message object as argument. Any security handler implementation should inherit class SenHandler and implement the interface according to the actual functionality. The interface only return simple Boolean value, and any useful information generated during the calling of this interface should be put into the security attribute of the message, or put into the payload itself.*

Currently, ARC1 comes with the following four security handler implemented:

- *arc.authz* – Authorization SecHandler

The *arc.authz* is responsible for calling the interface of policy decision point and getting back the authorization result. There is one simple interface (see Figure 4) defined in PDP, which will be called by *arc.authz* if configured inside once there is message (incoming or outgoing) need to be processed.

- *identity.map* – Identity Mapping SecHandler

The *identity.map* is a specific authorization oriented security handler. It will map the global identity in the message into local identity like system username based on the result returned by Policy Decision Point components.

- *delegation.collector* – Delegation SecHandler

The *delegation.collector* is responsible for collecting the delegation policy information from the remote proxy credential (proxy certificate is compatible to RFC3820) inside the message, and putting this policy into message's security attribute for the usage of other components, such as *delegation.pdp*.

- *usernametoken.handler* – UseranemToken SecHandler

The task of the *usernametoken.handler* is to generate the WS-Security Username-Token and add it into header of SOAP message which is the payload of outgoing message. It can also extract the WS-Security Username-Token from the header of SOAP message which is the payload of incoming message.

### 2.3. Interface of PDP

Figure 4 shows the definition of abstract class PDP. The implementation could be some function which implements the interface by composing the policy evaluation request, evaluating this request against some policy, and returning the evaluation result, or just by composing the policy evaluation request, invoking some remote policy decision web service and getting back the evaluation result.

```
class PDP {
 public:
  PDP(Arc::Config* cfg) { };
  virtual ~PDP() {};
  virtual bool isPermitted(Arc::Message *msg) = 0;
};
```

*Figure 4. class PDP is an abstract class which includes a general interface called isPermitted which uses Message object as argument. Any policy decision point implementation should inherit class PDP and implement the interface according to the actual functionality. The interface only return simple Boolean value, and any useful information generated during the calling of this interface should be put into the security attribute of the message, or put into the payload itself.*

Currently, ARC1 comes with the following four policy decision point implementation:

- *arc.pdp* – Arc PDP

The Arc PDP will organize the security attributes into the ARC specific authorization request, call the policy evaluator to evaluate the request against the policy (which is in ARC specific format) repository, and get back the evaluation result. See paragraph 3 for detail information about request schema and policy schema.

- *delegtion.pdp* – Delegation PDP

The Delegation PDP is basically similar to Arc PDP, except it uses the delegation policy parsed from remote proxy credential by *delegation.collector*, and evaluates the request against delegation policy. See section 10 for the design idea and use case of delegation policy in fine-grained identity delegation.

- *simplelist.pdp* – Simplelist PDP

The SimplelistPDP is a simplest implementation of policy decision point. It will match the identity extracted from the remote credential (or proxy credential) with local list of permitted identities.

- *pdpservice.invoker* – PDP Service Invoker

The PDP Service Invoker is a client which can be used to invoke the PDP service which implements the same functionality as Arc PDP, except that the evaluation request and response are carried by SOAP message. The benefit of implementing PPD service and PDP Service Invoker is that the policy evaluation engine can be accessed remotely and maintained centrally.

## 3. Policy Evaluation Engine

### 3.1. Design of policy evaluation engine

ARC1 defines specific evaluation request and policy schema. Based on the schema definition, one policy evaluation engine is implemented. The design principal of policy evaluation engine is generality by which the implementation of the policy evaluation engine can be easily extended to adopt some other policy schema, such as XACML policy schema.

Figure 5 shows the UML class diagram about the policy evaluation engine. It shows the all classes and relations simultaneously for getting the overall picture.

The classes Evaluator is the key class for policy evaluation, which will accept request, evaluate against local policy repository, and return evaluation response.

Three abstract factories (FnFactory, AlgFactory, AttributeFactory) are responsible for creating the Function, CombiningAlg and AttributeValue object. Class “CombiningAlg” is for processing different combining algorithm which constrains relation between <Rule/>. Class “AttributeValue” is for processing different types of <Attribute/>. Class “Function” is for processing different comparison between <Attribute/>.

Class Policy is responsible for parsing <Policy> or <Rule>, including creating CombingAlg object according to the <RuleCombiningAlg/> attribute of <Policy/>, creating Function object according to the <Function/> attribute of <Attribute/>, and creating AttributeValue object according to the <Type/>

attribute of <Attribute/>. Those objects will be used when evaluating the request.

Class Request is responsible for parsing <Request>, including creating AttributeValue object according to the <Type/> attribute of <Attribute/>. When evaluating, each <Attribute/> in request will be evaluated against corresponding <Attribute/> in the policy side by using relevant <Function/>.

*Figure 5. The UML class diagram of the classes inside policy evaluation engine.*

### 3.2. Schemas for policy evaluation engine

Policy Schema:

http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Policy.xsd

Request Schema:

http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Request.xsd

Response Schema

http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Response.xsd

### 3.3. Interface for using the policy evaluation engine

There is some interface which is implemented for conveniently using the policy evaluation engine.

a) Create the policy evaluation object:

```
ArcSec::EvaluatorLoader eval_loader;
//Load the Evaluator
ArcSec::Evaluator* eval = NULL;
std::string evaluator = "arc.evaluator";
eval = eval_loader.getEvaluator(evaluator);
```

b) Create the policy object:

```
ArcSec::Policy* policy = NULL;
std::string policyclassname = "arc.policy";
ArcSec::SourceFile policy_source("Policy_Example.xml");
policy = eval_loader.getPolicy(policyclassname, policy_source);
```

c) Create the rule object:

```
ArcSec::Request* request = NULL;
std::string requestclassname = "arc.request";
ArcSec::SourceFile request_source("Request.xml");
request = eval_loader.getRequest(requestclassname, request_source);
```

d) Add the policy into evaluator object:

```
eval->addPolicy(policy);
```

e) Evaluate the request object:

```
ArcSec::Response *resp = NULL;
resp = eval->evaluate(request);
```

The steps d) and e) can also be replaced by:

```
resp = eval->evaluate(request, policy);
```

The “evaluate” method can also be feed up with file path, string, or XMLNode directly. See example code at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/testinterface.cpp for more details about usage of the interface.

## 4. Policy Decision Service

Policy decision service is a service implementation which contains the functionality of ArcPDP. It will accept the soap request containing policy decision request and return soap response containing policy decision response.

The WSDL description for policy decision service is presented in Section 7.5.

# 5. Security Attributes

## 5.1. Infrastructure

Security Attributes represent security related information inside HED framework and store information representing various aspects needed to perform authorization decison - identity of client, requested action, targeted resource, constraint policies.

Each kind of Security Attribute is represented by own class inherited from parent SecAttr class <arc/message/SecAttr.h>. Each Security Attribute stores it's information in internal format and is capable to export it to one of predefined formats using Export() method. Currently only supported format is ARC Policy/Request XML document described in Section 7.

Collectors of Security Attributes instantiate corresponding classes and link them to Secuirity Attributes containers - MessageAuth <arc/message/MessageAuth.h> and MessageAuthContext <arc/message/Message.h> storing collected attributes per request and per session correspondingly. Each attribute is assigned a name. Current implementations of Security Attributes Collectors are either integrated into existing MCCs or implemented as separate SecHandler plugins. See Section 9 for available Collectors and corresponding Security Attributes.

Note for service developers: Services may implement own authorization algorithms. But they may use Security Atributes as well by providing instances of classes inherited from SecAttr and running them through either configured or hardcoded processors/PDPs.

Processors of Security Attributes are implemented as Policy Decision Point components. Currently there are 2 PDP components available:

- ArcPDP makes use of Security Attributes containing identities of client, resource and requested action. It evaluates either all or selected set of attributes against specified Policy documents thus making it possible to enforce policies defined/selected by service providers.
- DelegationPDP is described below in Section 10.

## 5.2. Available collectors

Here Security Attribute collectors distributed as part of ARC1 are described except those used for Delegation Restrictions. Those are described in Section 10.

### *5.2.1. TCP*

Information is collected inside TCP MCC. Security Attribute is stored under name 'TCP' and exports ARC Request with following attributes:

| ***Element*** | ***AttributeId*** | ***Content*** |
|---|---|---|
| Resource | http://www.nordugrid.org/schemas/policy-arc/types/localendpoint | service_ip[:service_port] |
| SubjectAttribute | http://www.nordugrid.org/schemas/policy-arc/types/remoteendpoint | client_ip[:client_port] |

*Table 1. Security Attributes collected at TCP MCC*

### *5.2.2. TLS*

Information is collected inside TLS MCC. Generated Security Attribute class exports ARC Request with following attributes:

| ***Element*** | ***AttributeId*** | ***Content*** |
|---|---|---|
| SubjectAttribute | http://www.nordugrid.org/schemas/policy-arc/types/tls/ca | signer of first certificate in client's chain |
| SubjectAttribute | http://www.nordugrid.org/schemas/policy-arc/types/tls/chain | Subject of certificate in client's chain - multiple items |
| SubjectAttribute | http://www.nordugrid.org/schemas/policy- | Subject of last certificate in |

| *Element* | *AttributeId* | *Content* |
|---|---|---|
| | arc/types/tls/subject | client's chain |
| SubjectAttribute | http://www.nordugrid.org/schemas/policy-arc/types/tls/identity | Subject of last non-proxy certificate in client's chain |

*Table 2. Security Attributes collected at TLS MCC*

#### *5.2.3. HTTP*

Information is collected inside HTTP MCC. Security Attribute is stored under name 'HTTP' and exports ARC Request with following attributes:

| *Element* | *AttributeId* | *Content* |
|---|---|---|
| Resource | http://www.nordugrid.org/schemas/policy-arc/types/http/path | HTTP path without host and port part |
| Action | http://www.nordugrid.org/schemas/policy-arc/types/http/method | HTTP method |

*Table 3. Security Attributes collected at HTTP MCC*

#### *5.2.4. SOAP*

Information is collected inside SOAP MCC. Security Attribute is stored under name 'HTTP' and exports ARC Request with following attributes:

| *Element* | *AttributeId* | *Content* |
|---|---|---|
| Resource | http://www.nordugrid.org/schemas/policy-arc/types/soap/endpoint | To element of WS-Addressing structure |
| Action | http://www.nordugrid.org/schemas/policy-arc/types/soap/operation | SOAP top level element name without namespace prefix |
| Context | http://www.nordugrid.org/schemas/policy-arc/types/soap/namespace | Namespace of SOAP top level element |

*Table 4. Security Attributes collected at SOAP MCC*

## 6. Delegation Restrictions

### 6.1. Delegation Architecture

In current implementation delegation is achieved through Identity Delegation implemented using X509 Proxy Certificates as defined in RFC 3820. Client wishing to allow service to act on it's behalf provides Proxy Certificate to the service using Web Service based Delegation interface described in Section 10.

For limitinng the scope of delegated credentials along with usually used time constraints it is possible to attach Policy document to Proxy Certificate. According to RFC 3820 Policy is stored in ProxyPolicy extension. In order not to introduce new type of object Policy is assigned id-ppl-anyLanguage identifier. RFC 3820 allows any octet string associated with such object. We are using textual representation of ARC Policy XML document.

Each deployment implementing Delegation Restrictions must use dedicated Security Handler plugin (see section 8) to collect all Policy documents from Proxy Certificates used for establishing secure connection. Then those documents must be processed by dedicated Policy Decision Point plugin (see section 6) to make a final decision based on collected Policies and various information about client's identity and requested operation. Service or MCC chain supporting Delegation Restrictions must accept negative decision of this PDP as final and do not override it with any other decision based on other policies.

### 6.2. Delegation Collector

This Security Attribute is collected by dedicated Security Handler plugin named "delegation.pdp" avaialble in arcpdc plugins' library. It extracts policy document stored inside X509 certificate proxy extension as defined in RFC3820 and described in Section 10. All proxy certificates in a chain provided by client are examined and all available policies are extracted.

Extracted content is converted into XML document. Then document is checked to be of ARC Policy kind. If policy is not recognized as ARC Policy procedure fails and that causes failure of comunication.

Proxy certificates with id-ppl-inheritAll property are passed through and no policy document is generated for them. Proxies with other type of policies including id-ppl-independent are not accepted and generate immediate failure.

### 6.3. Delegation PDP

DelegationPDP is similar to ArcPDP described above except that it takes it's Policy documents directly from Security Attributes. Differently from Arc PDP it is meant to be used for enforcing policies defined by client.

### 6.4. Delegation interface

Delegation interface in ARC1 is implemented using Web Service approach. Each ARC1 service wishing to accept delegated credentials implements this interface. Here is how delegation procedure works:

- Step 1
  - Client contacts service requesting operation DelegateCredentialsInit. This operation has no arguments.
  - Service responds with DelegateCredentialsInitResponse message with element TokenRequest. That element contains credentials request generated by service in Value. Type of request is defined by attribute Format. Currently only supported format is x509. Along with Value service provides identifier Id which is used in second step.
- Step 2
  - Client requests UpdateCredentials operation with DelegatedToken argument. This element contains Value with serialized delegated credentials and Id which links it to first step. Delegated token element may also contain multiple Reference elements. Reference refers to the object which these credentials should be applied to in a way specific to the service. The DelegatedToken element must also be used for delegating credentials when Step 2 is combined with other operations on service.
  - **Service responds with empty UpdateCredentialsResponse message.**

## 7. Schemas, descriptions and examples

### 7.1. Authorization Policy

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Policy.xsd .

### 7.2. Authorization Request

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Request.xsd .

### 7.3. Authorization Response

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Response.xsd .

### 7.4. Interface of policy decision service

WSDL with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/services/pdp/pdp.wsdl .

### 7.5. Configuration of PDP service

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/services/pdp/pdp.xsd .

Below is an example configuration of PDP service which can evaluate ARC Request against ARC Policy stored in local file.

```
<Service name="pdp.service" id="pdp_service">
<!--The element <Evaluator/>, <Policy/> and <Request/> configuration
```

```
        are supposed to be used to load object; element <PolicyStore/> is
        supposed to be used to get the location of policy-->
    <pdp:PDPConfig>
        <pdp:PolicyStore>
            <Location Type="file">Policy_Example.xml</Location>
            <!-- other policy location-->
        </pdp:PolicyStore>
        <pdp:Evaluator name="arc.evaluator" />
        <pdp:Policy name="arc.policy" />
        <pdp:Request name="arc.request" />
    </pdp:PDPConfig>
</Service>
```

See Section7.7 for the explanation about policy.

### 7.6. SimpleListPDP configuration and Policy Example

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/simplelistpdp/SimpleListPDP.xsd .

Below is an example configuration of SimpleListPDP inside "echo" service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="simplelist.pdp" location="simplelist"/>
  </SecHandler>
  <next id="echo"/>
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

The attribute "name" of <PDP/> is critical for loading the object. Specifically, the name "simplelist.pdp" is for loading the SimpleListPDP object.

The policy file "simplelist" should a local file which contains the identity list of authorized entities. It the peer certificate is proxy certificate, the identity in this list should not include the random number part of the proxy certificate' DN, only the original identity.

simplelist should be like this:

/C=NO/O=UiO/CN=test1

/C=NO/O=UiO/CN=test2

### 7.7. Arc PDP configuration and Policy Example

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/ArcPDP.xsd .

Below is an example of configuration of ArcPDP inside "echo" service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="arc.pdp">
      <PolicyStore>
        <Location type="file">Policy_Example.xml</Location>
        <!--other policy location-->
      </PolicyStore>
    </PDP>
  </SecHandler>
  <next id="echo"/>
```

```
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

The name “arc.pdp” is for loading the ArcPDP object.

There could be a few policy files under <PolicyStore/>. The request will be checked against all of the policies.

The following is an example policy for echo service. See Section 7.1 for the policy schema. The explanation about the example policy is as follows:

1. The http path to “/Echo” is protected, which is as one kind of “Resource”

2. The “Action” to this resource is “POST”, “GET” and “echo”. Of cause, the “echo” action enforced on “/Echo” resource sounds not reasonable. In this case, you can define a separated <Rule/> for containing the “echo” action enforced on some other reasonable resource.

3. The requester should process two <Attribute/>: has specific identity, and is signed by specific CA.

4. No <Conditions/> defined.

5. If and only if all of the above constraints have been satisfied by requester, the <Rule/> can be passed.

About the attribute collecting, for arc.pdp, all of the container-specific attributes about the requester are collected by different MCCs. It is possible for service to collect some application-specific attribute by implementing the SecAttr interface, and it should be the task of application developer.

User can configure SecHandler for each MCC and Service, and define reasonable and meaningful policy. On point when defining policy is user must be sure that the attributes defined in the policy should have already been collected by components. For instance, policy with AttributeId “http://www.nordugrid.org/schemas/policy-arc/types/http/path” should not be configured inside SecHandler under MCCTLS.

*Policy.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy        xmlns="http://www.nordugrid.org/schemas/policy-arc"        PolicyId="sm-example:arcpdppolicy" CombiningAlg="Deny-Overrides">
   <Rule Effect="Permit">
     <Description>
       Example policy for echo service
     </Description>
      <Subjects>
         <Subject>
                      <Attribute  AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/tls/ca" Type="string">/C=NO/ST=Oslo/O=UiO/CN=CA</Attribute>
                      <Attribute  AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/tls/identity" Type="string">/C=NO/ST=Oslo/O=UiO/CN=test</Attribute>
         </Subject>
      </Subjects>
      <Resources>
                       <Resource   AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/http/path" Type="string">/Echo</Resource>
      </Resources>
      <Actions>
                         <Action   AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/http/method" Type="string">POST</Action>
                         <Action   AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/http/method" Type="string">GET</Action>
                         <Action   AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/soap/operation" Type="string">echo</Action>
      </Actions>
      <Conditions/>
```

```
   </Rule>
</Policy>
```

### 7.8. PDP Service Invoker configuration

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/pdpserviceinvoker/ArcPDPServiceInvoker.xsd .

Below is an example of configuration of PDPServiceInvoker inside "echo" service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <!--Remote pdp service invoking-->
      <PDP name="pdpservice.invoker">
        <ServiceEndpoint>https://127.0.0.1:60001/pdp.service</ServiceEndpoint>
        <KeyPath>./key.pem</KeyPath>
        <CertificatePath>./cert.pem</CertificatePath>
        <CACertificatePath>./ca.pem</CACertificatePath>
      </PDP>
  </SecHandler>
  <next id="echo"/>
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

The name "pdpservice.invoker" is for loading the PDPServiceInvoker object.

The pdpserviceinvoker works as a client to pdpservice. The credential configuration for pdpserviceinvoker is independent with the credential configuration for the main chain.

### 7.9. Delegation PDP configuration

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/delegationsh/DelegationPDP.xsd .

Below is an example of configuration of DelegationPDP inside "echo" service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="delegation.pdp"/>
  </SecHandler>
  <next id="echo"/>
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

For Delegation PDP, no specific configuration is needed. We only need to switch on it by add <PDP name="delegation.pdp"/> under <SecHandler/>

### 7.10. Delegation SecHandler Configuration

Below is an example of configuration of Delegation SecHandler inside MCCTLS component.

```
        <Component name="tls.service" id="tls"> <next id="http"/>
           <tls:KeyPath>./key.pem</tls:KeyPath>
           <tls:CertificatePath>./cert.pem</tls:CertificatePath>
           <tls:CACertificatePath>./ca.pem</tls:CACertificatePath>
           <!--delegation.collector must be inside tls MCC-->
         <SecHandler name="delegation.collector"
```

```
            id="delegation" event="incoming"></SecHandler>
        </Component>
```

Delegation SecHandler must be configured under MCCTLS.

### 7.11. UsernameToken SecHandler configuration

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/usernametokensh/UsernameTokenSH.xsd .

Below is an example of configuration of UsernameToken SecHandler inside MCCSOAP component.

```
    <Component name="soap.service" id="soap">
        <next id="echo"/>
                    <SecHandler  name="usernametoken.handler"  id="usernametoken"
event="incoming">
            <Process>extract</Process>
            <PasswordSource>password.txt</PasswordSource>
        </SecHandler>
    </Component>
```

UsernameToken SecHandler must be configured under MCCSOAP.

## 8. User Manual

This section describes how to configure and use SecHandler and PDP elements included in ARC1 and provides few examples of the ARC Policy documents. The target readers are those users who will use the ARC1 middleware. Curently this section is very short on details. It is going to be continuously extended. Especially taking user feedback into account.

### 8.1. Authorization SecHandler and PDPs

There is a specific Authorization SecHandler (arc.authz) which is implemented for calling the Policy Decision Points (PDP) and serves as their container.

Usually the Authorization SecHandler and included PDPs are used on the service side of communication channel. Although it is also possible to use them on the client side. All possibilities are achieved by modifying the configuration file (hereafter mentioned as service.xml) and possibly providing the authorization policy in a separate file.

Here the “echo” test service is used to explaining the usage, but the explanation applies to other services as well.

The procedure for configuring Authorization SecHandler in service.xml is following:

1. Add the Authorization SecHandler as child element <SecHandler/> of <Service/> element.

   The “name” and “event” attribute of <SecHandler/> element are both important. The “name” attribute is used for distinguishing betwenn loaded SecHandler objects. The “event” attribute defines for which message authorization would be enforced. Usually and reasonably it is done for “incoming” messages. But some services and other Message Chain components may define other internal types of messages. For possible values please refer to documentation of particular Service or MCC. In our particular case “echo” service only supports “incoming” messages for this purpose.

2. Add the PDP configuration as child element <PDP/> under <SecHandler/>. Currently there are four usable PDPs distributed as part of ARC1 middleware:

   - simplelist.pdp – compares Subject of user's X509 certificate to those stored in a file.
   - arc.pdp – compares authorization related information parsed from message at various processing steps to Policy document specified in configuration of this PDP.
   - pdpservice.invoker - composes the ARC Request, puts request into SOAP message, and invokes the remote PDP service to get the response SOAP which includes authorization decision. The PDP service functionality is similar to arc.pdp.

- delegation.pdp – compares authorization related information parsed from message at various processing steps and Policy document embedded in proxy certificate used by remote side.

Default behavior of Authorization SecHandler is to execute all configured PDPs sequentially till eithe r one of them fails or all produced positive results. This behavior may be modified by attribute “action” of <PDP/> element.

The description of PDP configuration and ARC Policy example are available in Section 7.

### 8.2. Delegation SecHandler, Delegation PDP and Proxy Certificate Generation

Delegation SecHandler and Delegation PDP in their current state provide an infrastructure for limiting capabilities of delegated credentials. Their collect and process policies attached to X509 Proxy Certificates respectively. Hence to have delegation restriction working both must be enabled in configuration of service. Configuration of Delegation SecHandler is described in section 7.10.

The possible location for Delegation PDP is inside Authorization SecHandler (arc.authz). Depending on how fine grained policy of delegated credentials is supposed to be corresponding Authrorization SecHandler may be attached to different MCCs or directly to Service component. However, the precondition for using Delegation PDP is that there must be Delegation SecHandler instantiated earlier in chain.

On the client side, command line utility “approxy” utility can be used to generate Proxy Certificate with Delegation Policy embedded.

Normally approxy appears in $ARC_LOCATION/bin. The usage of approxy is like:

```
approxy -P proxy.pem -C cert.pem -K key.pem -c constraints
```

By using argument "-c", some constraints can be specified for proxy certificate. Currently, the life time can be limited by using "-c validityStart=..." and "-c validityEnd=...", "-c validityStart=..." and "-c validityPeriod=...". Like for example

```
-c validityStart=2008-05-29T10:20:30Z
```

```
-c validityEnd=2008-06-29T10:20:30Z
```

The Delegation Policy can be specified by using "-c proxyPolicyFile=..." or "-c proxyPolicy=...". Like

```
-c proxyPolicyFile=delegation_policy.xml
```

The Delegation Policy is the same as the ARC Policy explained in section 7.7. There is real life example below which renders delegated credentials usable only for contacting service attached to HTTP communication channel under path /arex and allows HTTP operation POST on it.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy        xmlns="http://www.nordugrid.org/schemas/policy-arc"        PolicyId="sm-example:policy1" CombiningAlg="Deny-Overrides">
   <Rule RuleId="rule1" Effect="Permit">
      <Resources>
         <Resource Type="string" AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/http/path">/arex</Resource>
      </Resources>
      <Actions>
          <Action Type="string" AttributeId="http://www.nordugrid.org/schemas/policy-arc/types/http/method">POST</Action>
      </Actions>
   </Rule>
</Policy>
```

### 8.3. UsernameToken SecHandler

The UsernameToken SecHandler is meant for processing - generating and extracting - WS-Security [2. OASIS Web Services Security (WSS) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss] UsernameToken from SOAP header. Hence it must be attached to SOAP MCC of service or/and client communication channel.

On the service side, the functionality of extracting UsernameToken may be configured as described in

section 7.11.

On the client side, the UsernameToken SecHandler may be configured either by using client specific methods (for example see test_clientinterface.cpp src/tests/echo directory of source tree) or through generic client configuration file as shown in example below. This example will generate token with username “user” and password “pass” inside any SOAP message sent by client tools of ARC1.

```
<ArcConfig>
 <Plugins overlay="add"><Name>arcpdc</Name></Plugins>
 <Chain>
  <Component name="soap.client">
   <SecHandler name="usernametoken.handler" event="incoming" overlay="add">
    <Process>generate</Process>
    <Username>user</Username>
    <Password>pass</Password>
    <PasswordEncoding>digest</PasswordEncoding>
   </SecHandler>
  </Component>
 </Chain>
</ArcConfig>
```

## REFERENCES

1. RFC3820, http://rfc.net/rfc3820.html

2. OASIS Web Services Security (WSS) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss