GFD-R.232
NSI-WG
nsi-wg@ogf.org

Hans Trompert, SURFnet
John MacAuley, ESnet
August 18, 2017

NSI Authentication and Authorization

Status of This Document
Grid Forum Document (GFD), Proposed Recommendation (R).

Abstract
This document outlines security requirements placed on Network Service Agents (NSA) when participating in the Network Services Interface (NSI) Connection Service (CS) protocol. It describes in detail how the NSI CS security attributes should be used to deliver integration with end-user authentication and authorization mechanisms..

Contents

1 Introduction

The Network Services Interface provides an API that allows applications to monitor, control, interrogate, and support network resources that are made available by the provider of the network. The NSI Connection Service deals specifically with the request and management of network Connections on transport networks. NSI is inherently agnostic to the technology used in the transport plane. This technology agnostic approach is built into the NSI topology representation and is supported through the use of Service Definitions.

A Connection Service can be requested by any application that has implemented an NSI CS Requester Agent (RA). Similarly, any network provider who has implemented an NSI Provider Agent (PA) can service the request. These are both examples of a Network Service Agent (NSA).

Each service is managed by an exchange of NSI messages between agents. These messages operate using a set of service primitives. Service primitives are the set of instructions that allow the requester to set up and manage a service. Each service request will result in the allocation of a service id for the new service instance.

This document describes how security is implemented in Network Service Agents when participating in the NSI CS protocol. It describes in detail how the NSI CS security attributes should be used to deliver integration with end-user authentication and authorization mechanisms.

This document should be read in conjunction with GFD-R.212 Network Service Interface Connection Service v2.0 [GFD.212], GFD-I.213, Network Services Framework v2.0 [GFD.213] and GFD-I.217 NSI Signaling and Path Finding [GFD.217].

2 Notational Conventions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in [RFC 2119]. Words defined in the glossary are capitalized (e.g. Connection). NSI protocol messages and their attributes are written in camel case and italics (e.g. *reserveConfirmed*)

3 Requirements

The NSI Connection Service v2.0 recommendation [GFD.212] states that NSI security is achieved using Transport Layer Security (TLS) between NSAs. The version of TLS utilized is deployment specific and fluid based on currently reported vulnerabilities in the TLS implementations. At the time of writing of this document, deployments of NSI are using TLS version 1.2. In addition, SAML attributes are provided to convey additional information regarding NSI request authentication and authorization. This OGF recommendation goes into further detail about how to apply security to the NSI protocol. The following security requirements have been derived from the experience gained in during NSI pilot deployments.

- The integrity and confidentiality of the messages between NSAs MUST be ensured.
- All access to the NSI Service Plane MUST be authorized by the ultimate Requester Agent (uRA).
- Access to a network’s Transport Plane resources MUST be authorized by the ultimate Provider Agent (uPA) representing that network.
- It MUST be possible to identify the Originating Entity of an NSI request.
- It MUST be possible to identify the uRA of an NSI request.

- End user authorization schemes are deployment specific, and in many cases site specific as well. Therefore, it MUST be possible within the NSI security framework to simultaneously support multiple authorization mechanisms.

4 Fundamental Principles of Security in NSI

An NSI Service Plane consists of a set of NSI Network Service Agents that are allowed to connect to each other through a prearranged administrative agreement; however, the process for determining this agreement is a deployment specific issue. This administrative agreement can be likened to the process by which peering is agreed between providers at layer 3.

In addition, NSAs authenticate pair wise, but not all NSAs authentication with each other (there is no requirement that there be a full mesh of NSI reachability between NSAs), so the resulting Service Plane graph can be sparsely connected.

To allow communication between NSA that are not directly peered, the NSI CS allows for message exchange between indirectly connected NSA using an intermediate aggregator NSA. The aggregator NSA will process an incoming protocol message from a peer NSA, determine the destination NSA of the request, and generate a new outgoing protocol message targeting a second peer along the service plane path, and in many cases without the second peer NSA having any knowledge of the first peer NSA. The second peer has to trust that the aggregator NSA has done due diligence on the first peer's request before passing the message on. As a consequence, NSI Service Plane security is based on transitive trust, i.e. I trust my neighbours and the neighbours they trust. Any administrative peering process should take this fact into consideration when adding new peers to an NSI Service Plane.

NSI uses Client Authenticated TLS as a transport protocol to ensure the integrity and confidentiality of the messages traveling through a trusted Service Plane. Client Authenticated TLS uses X.509 certificates as a mechanism to authenticate the identity of peer NSA during TLS session setup. This allows an NSA to validate that it is communicating with a trusted peer, determine the identity of the trusted peer through remote host name and certificate DistinguishedName, and that all communications with the peer NSA is being encrypted.

Peer NSA MUST authenticate each other using Client Authenticated TLS. [GFD.212]

All traffic between two peering NSAs MUST be encrypted using TLS while in transit. [GFD.212]

The mechanism used for NSAs to authenticate each other via X.509 certificates can differ from one peer to another. For example, one group of NSA administrators can agree on the use of a common trusted Certificate Authority, while other administrators will just exchange certificates on a per peer basis using secure external channels. These certificates are then directly provisioned on the peer NSA. An advantage of this second method is that it also allows for the secure exchange of self-signed certificates. For self-signed certificates, the peer's public certificate is provisioned directly on the target NSA as an authenticating CA, allowing for secure client authentication.

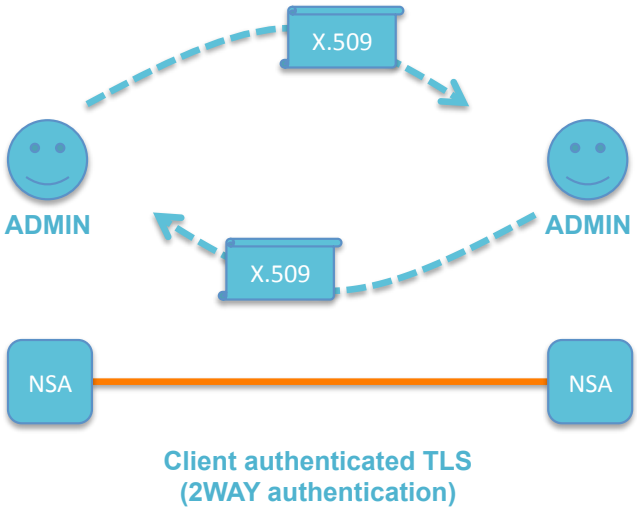

Figure 1 - 2-WAY TLS between peer NSA.

Additional certificate access control checks between peering NSAs can be implemented such as hostname verification, and subject *DistinguishedName* (DN) verification of the peer. In this case the Subject DN of the authenticated certificate is verified against Subject DN that was exchanged beforehand to uniquely identify the remote NSA and authorize the peering.

> *An NSA MUST authorize each peer individually before processing any NSI messages. (i.e is this NSA allowed to participate in the NSI CS with me?)*

In addition to Client Authenticated TLS, each NSA type has a specific security obligation to the Service Plane:

> *An Aggregator MUST process NSI messages from peers subject to NSI policy [NSI Policy], perform path computation if needed [GFD.213], and propagate messages to peers along a path to the target uPA or uRA depending on direction of message.*
>
> *A uRA MUST determine the identity of the requesting user and authorize that user's access to a trusted Service Plane. The uRA does not authorize a user's access to Transport Plane resources.*
>
> *A uPA MUST authorize a user's access to Transport Plane resources in its associated network.*

Figure 2 below illustrates these security concepts.

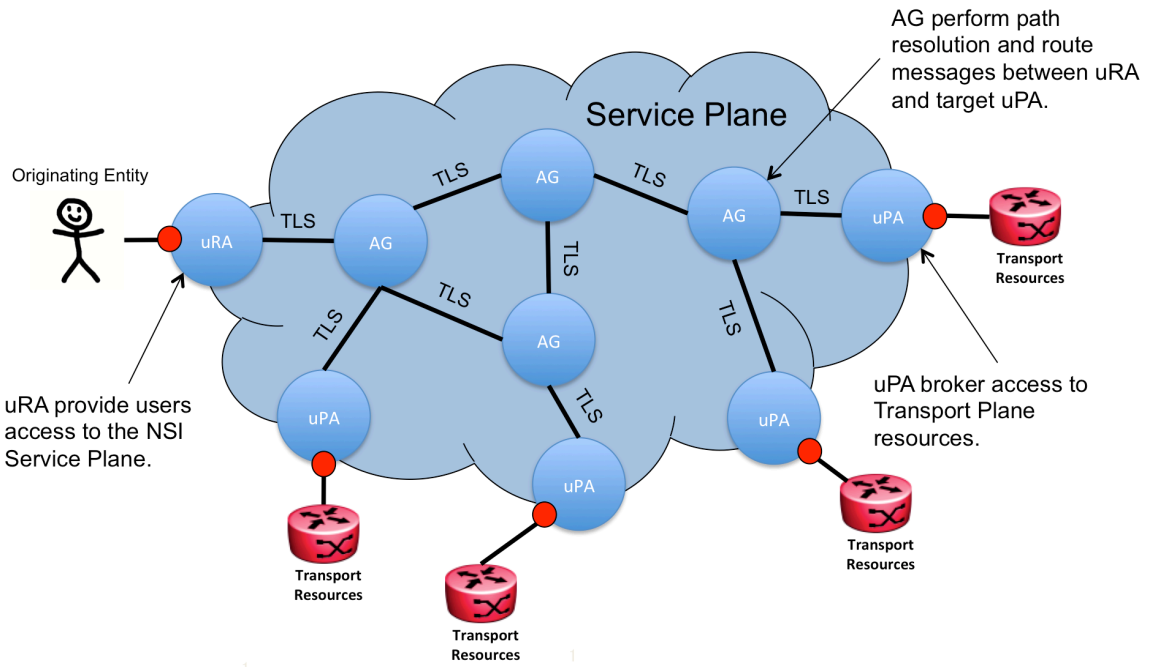

Figure 2 – Security in a Service Plane.

A group of NSAs that together form a trusted Service Plane will be self-regulating. NSA administrators are responsible for performing regulation through manual actions. Misbehaving NSAs MUST be called to account by the community, and in the worst case such a NSA will be removed from the Service Plane. There are no automated mechanisms for detecting or removing an NSA deemed to be "misbehaving".

A framework for the passing of security related attributes with the NSI messaging header is defined in [GFD.212]. This framework is based on flexible SAML attribute statements that are chosen for their ability to model generic security related attributes in a well-defined XML schema. However, [GFD.212] does not specify a formal use for the "*sessionSecurityAttr*" attribute, instead leaving it for further study. Within this document a formalized use of these SAML attribute statements is provided for modelling security related information.

The *sessionSecurityAttr* is used to implement two classes of security attribute defined to help deliver integration with end-user authentication, authorization, and policy mechanisms. The first class are considered mandatory and are used by NSAs within the Service Plane to perform functions such as user identity tracking for the purpose of auditing and troubleshooting. The second class of security attributes are those conveying external authorization information transparently through the Service Plane. This second type is typically populated by uRA (client NSA) for the communication of authorization information to uPA (NSA associated with resources). Section 7 will discuss the use of the *sessionSecurityAttr* in more detail.

The mechanism by which the uRA and uPA authorize a user access is a deployment decision and is out of scope of the NSI protocol.

5 Access to the Service Plane

An NSI Connection Service request is any RA to PA Connection Service message as listed in table 2 of the NSI Connection Service v2.0 [GFD.212]. A uRA is a Requester Agent that is the

originator of a Connection Service request, and responsible for providing users/applications access to NSI connection services. The uRA is the source of NSI Connection Service messages in a Service Plane, initiating messaging at the root of the tree or start of the chain, hence the designation of “Ultimate” requester agent.

The uRA is responsible for establishing the identity of the Originating Entity that has requested access to the NSI Connection Services. How this identity is established is a local matter (TLS client authentication, authentication through Identity Provider, local user accounts, access tokens, etc.).

The uRA MUST determine the identity of the Originating Entity.

The uRA is also responsible for authorizing the Originating Entity’s access to the Service Plane after having established its identity. How the uRA authorizes a user is a local matter, and may be something as simple as providing access if the identity of the Originating Entity can be established (open policy) or something more restrictive based on an authorization server (restrictive policy).

The uRA MUST authorize the Originating Entity's access to the Service Plane.

The uRA is also responsible for traceability of requests for the purpose of security auditing by other NSA within the network involved in a specific Connection Service instance. The Originating Entity’s identity information is added to the NSI message header, along with the NSA identifier of the uRA, and sent to all peer PA participating in the Connection Service request. The uRA must maintain a local audit log of the originating reference and the NSI message for future reference.

The uRA MUST populate each NSI Connection Service message with its unique NSA identifier.

The uRA MUST populate each NSI Connection Service message with the Originating Entity’s identity.

The uRA MAY choose to provide an obfuscated identifier to the to the Originating Entity’s identity instead of the identity itself for the purpose of privacy.

If an obfuscated identifier is used for the Originating Entity it MUST be possible for any NSA in the network to back trace this identity reference to the originating uRA of the Connection Service request, and resolve the reference to the identity of the Originating Entity.

For example, a uRA can authenticate a local Originating Entity as long as the uRA is a part of a trusted Service Plane as described earlier in this document. This includes authentication done by user applications that have an integrated uRA.

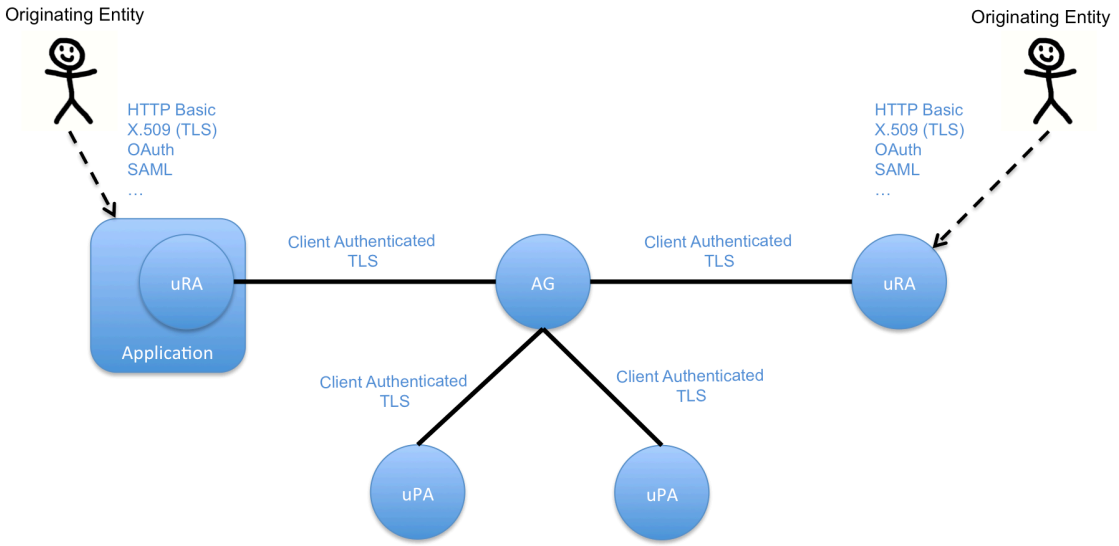

Figure 3 - Authenticated access to a Service Plane.

It is not required that every NSA along the reservation workflow be able to directly determine the Originating Entity's identity, however, it must be possible to trace the request back to the originating NSA, and from this NSA resolve the true identity of the Originating Entity. This will ensure that it is always be possible to reach the Originating Entity and hold it accountable even though that Originating Entity may not be identifiable at each NSA in the Service Plane.

If any NSA along the reservation workflow wants to hide the Originating Entity identifier found in the NSI message header, it is allowed to replace it with its own identity information and therewith take all responsibility for that message as it travels further through the Service Plane. If an NSA replaces an identity within the NSI message header is MUST maintain a record of the original Originating Entity so a reverse mapping can be performed for auditing purposes. In this case, the NSA will also rewrite the NSI message header to make it look like that NSA is the originating NSA. This act of anonymity is allowed for those organizations that do not wish to expose their end user's to other NSA within the Service Plane, but are willing take full responsibility for their actions.

> *An intermediate NSA in an NSI Connection Service message flow MAY replace the Originating Entity's identity reference with another identity reference. In this case this NSA MUST accept responsibility for the Connection Service request.*
>
> *An intermediate NSA in an NSI Connection Service message flow MAY replace the uRA's NSA identifier with its own only if it is willing to accept responsibility as the source of the Connection Service request, including all message audit requirements.*

6 Authorization

Every NSA must authorize NSI request messages and reject messages that do not comply to that NSA's policies. Authorization decisions are based on policies that are stored within a policy source. Such a policy source can either be local to the NSA or part of an authentication and authorization infrastructure where polices apply to a set of NSA. Depending on the deployment, a combination of local and/or remote policy sources can be used to authorize NSI requests. How authorization policies are administrated is deployment specific. In figure 4, NSA A is using a local

policy database as its policy source and NSA B, C and Z are using external AAI as a source for their policy.

All RA to PA Connection Service messages listed in table 2 of [GFD.212] must be authorized according to policy. There may be one policy for all messages, different policies for sets of messages, or even a per message policy. For example, this supports scenarios where a particular user is allowed to create a reservation, everybody that belongs to the same user group can query and modify but not terminate that reservation, and an administrator is allowed all actions including termination of the reservation. A policy such as 'allow everything' is a valid policy and can be adopted by providers wishing to leave usage unconstrained.

> *An NSA MUST enforce authorization when processing all Connection Service requests. (e.g is the Originating Entity allowed what they are requesting?)*

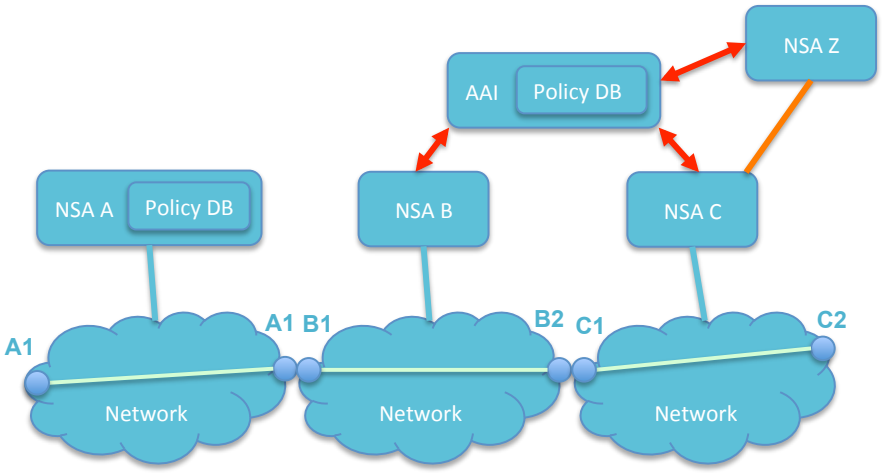

Figure 4. Policy source deployment example

Based on Figure 4, examples of authorization decisions that can be made by an NSA include:

- Is access to a specified endpoint STP allowed?
- Does the requested amount of bandwidth exceed the maximum amount allowed for that user (or user group, etc.)?
- Has the maximum number of reservations per day/week/year been exceeded?
- Does the local path segment involve a specific intermediate STP B2 that is part of SDP with Network C? (Transport Plane peering based authorization).
- Is the request received via the Service Plane from particular NSA Z? (*requesterNSA* attribute) (Service Plane peering based authorization).
- Use the default policy if no other policies are triggered.

The document [NSI Policy] captures a more detailed list of network policy requirements for enforcement by provider agents.

7 Security Attributes

As part of the definition of the NSI protocol message structure, a generic security attribute element called *sessionSecurityAttr* is defined. This attribute is a flexible container for transport of security related information. Zero or more of these *sessionSecurityAttr* elements can be populated in the *nsiHeader* element, which is itself carried in the SOAP envelope's *Header* element. The NSI Connection Services specification [GFD.212] Section 8.2.1 does not define the specific use of this *sessionSecurityAttr* element, instead leaving it for later definition and deployment specific use.

```
<soapenv:Header>
    <nsi_headers:nsiHeader xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
            xmlns:nsi_headers="http://schemas.ogf.org/nsi/2013/12/framework/headers"
            xmlns:nsi_ftypes="http://schemas.ogf.org/nsi/2013/12/framework/types">
        <protocolVersion>application/vnd.ogf.nsi.cs.v2.provider+soap</protocolVersion>
        <correlationId>urn:uuid:f123ef0a-a362-4524-b7ac-631cff3e7c66</correlationId>
        <requesterNSA>urn:ogf:network:example.net:2013:nsa:requester</requesterNSA>
        <providerNSA>urn:ogf:network:example.net:2013:nsa:aggregator</providerNSA>
        <replyTo>https://requester.example.net/requester/reply</replyTo>
        <sessionSecurityAttr type="urn:ogf:nsi:security:attr:example" name="example1">
            ...
        </sessionSecurityAttr>
        <sessionSecurityAttr type="urn:ogf:nsi:security:attr:example" name="example2">
            ...
        </sessionSecurityAttr>
    </nsi_headers:nsiHeader>
</soapenv:Header>
```

Figure 5 – The sessionSecurityAttr.

The *sessionSecurityAttr* element is defined using a standardized SAML *AtttributeStatementType* imported from the SAML namespace "*urn:oasis:names:tc:SAML:2.0:assertion"* with an NSI specific extension, adding a string based *type* and *name* attribute to this root element. This allows for multiple *sessionSecurityAttr* elements to be specified in the *nsiHeader* element, with each one identified for a specific use via the *type* and *name* attributes (for example, supplying user credentials per NSA domain).

The expected (default) behaviour is that a uRA will populate the security element based on information from/about the Originating Entity making the NSI request. Any NSA AG receiving these security elements will normally pass these on to all child NSAs, however, deployment specific behaviours may be introduced that change this default behaviour.

Other NSAs along a reservation workflow can add additional security attributes to a message; these are either new attributes that are deemed useful for NSAs downstream on the workflow, or modified attributes that are the result of evaluating existing message security attributes. Any NSA should be transparent to security attributes, meaning that all received attributes plus any potential new attributes are passed on to all downstream NSAs untouched.

> *An NSA SHOULD transparently pass all session security attributes from a received NSI request message through to all child NSAs receiving an NSI request message as part of the reservation.*
>
> *An NSA MAY add additional security attributes before sending a message on to a child NSA if that NSA has specific context information needed in the authorization flow of the message.*
>
> *An NSA MAY manipulate existing security attributes before sending on to a child NSA if the NSA has specific context information permitting this non-transparent manipulation.*

An NSA MAY delete security attributes before sending on to a child NSA if the NSA has specific context information requiring the removal of a specific attribute. Any deletion must be done with specific knowledge that the removed security attributes are not required by any other NSA within the Service Plane that will participate in that specific NSA message workflow. For tractability the NSA must maintain an audit record of any modification to or removal of security attributes from a message.

The context where a specific security attribute is to be evaluated is indicated by the *sessionSecurityAttr* element value itself. In this document we define two types of security elements:

1. A global standard security element with a defined *sessionSecurityAttr* element *type* attribute that all NSA understand and can utilize if required.
2. A realm specific element that is defined in the context of a group of NSAs considered part of a common authorization realm. In this case, the *sessionSecurityAttr* element *type* attribute identifies the element as domain specific and the name identifies the authorization realm itself. NSAs that are part of that authorization realm can identify the *sessionSecurityAttr* elements applicable to them by matching the element's *type/name* pair.

An NSA can be part of zero, one, or more authorization realms, and more than one NSA can be part of the same authorization realm.

7.1 Originating Entity Identifier

We introduce a specialized *sessionSecurityAttr* element called "*originatingId*" to address the uRA requirement to provide access to the Originating Entity's identity information, and the uRA's NSA identifier. A uRA populates the *nsiHeader* element of every NSI Connection Services request message with an *originatingId.* Response, Failed, Error, and Notification messages do not require an *originatingId* within the *nsiHeader*.

A uRA MUST populate an originatingId with its own NSA identifier and reference to the Originating Entity's identity.

The *originatingId* utilizes the *sessionSecurityAttr* element in the following way:

Parameter	Type	Mandatory	Description
name	Attr	True	The *sessionSecurityAttr.name* attribute contains the NSA identifier of the uRA issuing the request.
type	Attr	True	The *sessionSecurityAttr.type* attribute contains the NSI security attribute type identifier of "*urn:ogf:nsi:security:attr:originatingId*" following the SAML type identifier naming format.
Attribute	Elem	True	*The child SAML Attribute* element contains the reference to the Originating Entity's identity information as specified on the uRA.

7.1.1 Obfuscated Originating Entity identity reference

It is RECOMMENDED that an obfuscated identifier be used within the *originatingId* to provide confidentiality. The uRA is aware of the Originating Entity's true identity, while NSAs within the network have a reference to the entity that will allow them to contact the uRA for additional details, or to resolve a specific problem.

A uRA SHOULD populate the originatingId with an obfuscated reference to the requesting user's identity.

A *SAML Attribute* element of the *originatingId* is populated in the following way:

Parameter	Type	Mandatory	Description
Name	Attr	True	The *Attribute.Name* attribute contains the MACE identifier "*urn:mace:dir:attribute-def:eduPersonTargetedID*" indicating this Attribute is modelling a SAML/Shibboleth target identifier value.
NameFormat	Attr	True	The *Attribute.NameFormat* attribute contains the type identifier of "*urn:oasis:names:tc:SAML:2.0:attrname-format:uri*" indicating that the *Attribute.Name is a proper SAML URI*.
AttributeValue	Elem	True	*The child SAML AttributeValue* element contains the persistent reference to the user identity information as specified on the uRA. This *AttributeValue* element is populated with a SAML *NameID* element with the attribute *NameID.Format* set to "*urn:oasis:names:tc:SAML:2.0:nameid-format:persistent*", and a value of the persistent identifier.

The following is an example *originatingId* security attribute populated with an obfuscated identifier. In this example the originating uRA is identified as "`urn:ogf:network:example.net:2013:nsa:requester`" and the persistent identifier for the Originating Entity is "`c693b1c47a0da7de6518bc30a1bb8d2e44b56980`".

```
<sessionSecurityAttr type="urn:ogf:nsi:security:attr:originatingId"
        name="urn:ogf:network:example.net:2013:nsa:requester">
    <saml:Attribute Name="urn:mace:dir:attribute-def:eduPersonTargetedID"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
        <saml:AttributeValue>
            <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
                c693b1c47a0da7de6518bc30a1bb8d2e44b56980
            </saml:NameID>
        </saml:AttributeValue>
    </saml:Attribute>
</sessionSecurityAttr>
```

Figure 6 – *originatingId* with obfuscated entity identifier.

7.1.2 Direct user identity reference

An NSI deployment may decide not to use obfuscated identity in the *originatingId*, but instead a direct reference to the Originating Entity. The *SAML Attribute* element is flexible enough to handle these situations as well. For example, the *eduPersonPrincipalName* attribute is used by many organizations as part of their security federation, and is in the familiar form of "user@domain" that is typically assigned for authentication to network services within a security realm.

A uRA MAY populate the originatingId with a non-obfuscated reference to the requesting Originating Entity.

A *SAML Attribute* element of the *originatingId* is populated in the following way:

Parameter	Type	Mandatory	Description
Name	Attr	True	The *Attribute.Name* attribute contains the MACE identifier for the type of name being represented in the AttributeValue. Here we use "*urn:mace:dir:attribute-def:eduPersonPrincipalName*" indicating this Attribute is a scoped identifier for a person of the form user@domain.
NameFormat	Attr	True	The *Attribute.NameFormat* attribute contains the type identifier of "*urn:oasis:names:tc:SAML:2.0:attrname-format:uri*" indicating that the *Attribute.Name is a proper SAML URI*.
AttributeValue	Elem	True	*The child SAML AttributeValue* element is a string containing the scoped identifier for the user's identity information as specified on the uRA.

The following is an example *originatingId* security attribute populated with an *eduPersonPrincipalName* attribute identifier. In this example the originating uRA is identified as "`urn:ogf:network:example.net:2013:nsa:requester`" and the Originating Entity is "`bob@example.net`".

```
<sessionSecurityAttr type="urn:ogf:nsi:security:attr:originatingId"
        name="urn:ogf:network:example.net:2013:nsa:requester">
    <saml:Attribute Name="urn:mace:dir:attribute-def:eduPersonPrincipalName"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
        <saml:AttributeValue xsi:type="xsd:string">bob@example.net</saml:AttributeValue>
    </saml:Attribute>
</sessionSecurityAttr>
```

Figure 7 – *originatingId* with an *eduPersonPrincipalName* entity identifier.

In this example, an NSI deployment uses X.509 certificate authentication for all user entities accessing the network. For simplicity, the deployment utilizes the user's certificate subject DN as the unique identifier for the user within the *originatingId. For this case we have* originating uRA identified as "`urn:ogf:network:example.net:2013:nsa:requester`" and the Originating Entity as "`CN=bob@example.net,OU=User,O=Example Networks,C=US`". This uses the standard SAML *Subject* and *NameID* elements.

```
<sessionSecurityAttr type="urn:ogf:nsi:security:attr:originatingId"
        name="urn:ogf:network:example.net:2013:nsa:requester">
    <saml:Attribute Name="urn:oasis:names:tc:SAML:2.0:assertion:subject">
        <saml:AttributeValue>
            <saml:NameID
                    Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">
                CN=bob@example.net,OU=User,O=Example Networks,C=US
            </saml:NameID>
        </saml:AttributeValue>
    </saml:Attribute>
</sessionSecurityAttr>
```

Figure 8 – *originatingId* with X.509 subject name.

7.2 Authorization attributes

As discussed in section 6, NSI does not specify how a specific network deployment performs end user authorization. The final decision to approve an operation is left up to the uPA associated with the Network containing the requested resources. By making authorization a deployment time decision, NSI has provided the most flexibility for end networks, allowing each Network to decide on how they would like to authorize a user's access to their resources.

Similar to the mechanism used in section 7.1, “Originating Entity Identifier”, authorization information is passed from the uRA to the uPA using the flexible *sessionSecurityAttr* element for securely transporting security related information between NSA within the trusted Service Plane. As shown in Figure 9. , security related attributes introduced by the uRA are securely transported to all uPAs involved in the reservation through the secure Service Plane.

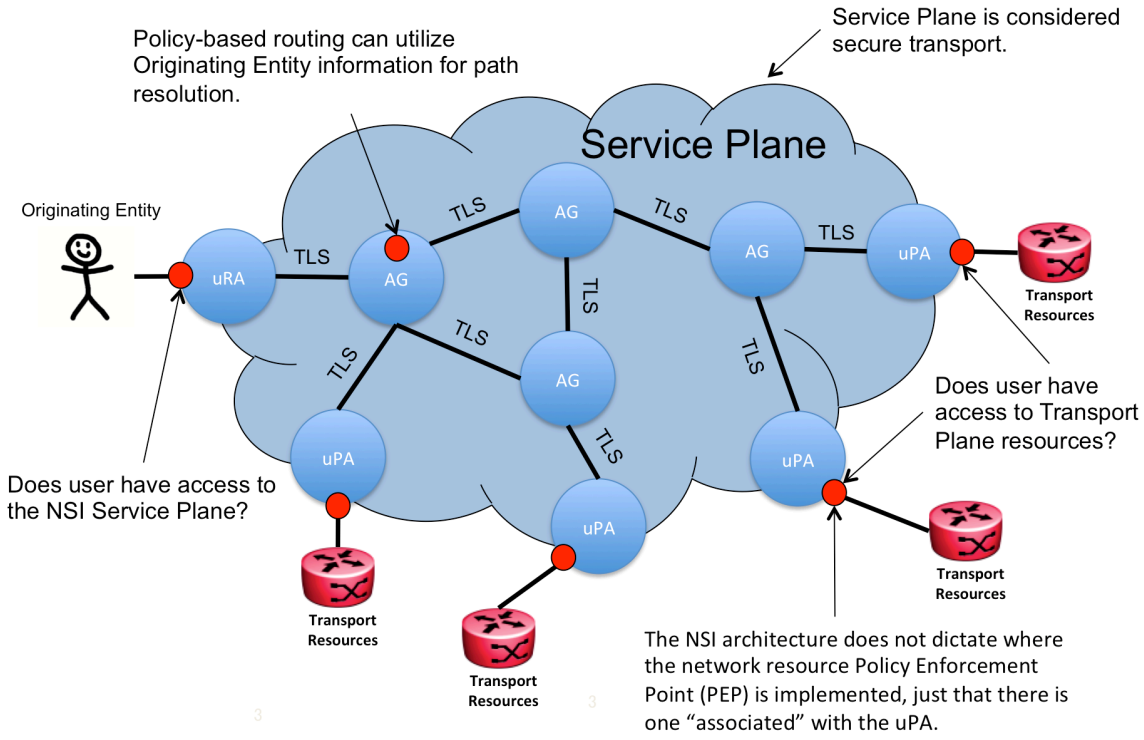

Figure 9. Service Plane security.

Authorization decisions are made based on attribute values that serve as input for policy rules that are either stored locally, or are fetched from one or more authorization policy sources, or both. Any NSI message attribute can be used as input for policy evaluation. Additional attributes needed for policy evaluation can be added to the NSI message header using the *sessionSecurityAttr* element. Examples of additional security attributes are:

- X.509 certificates
- OAuth access tokens
- Signed authorization certificates
- Group membership information

The uRA will be the primary source of security attributes within an NSI message, however, every NSA along the reservation workflow can add additional attributes to a message if needed. These are either new attributes that are deemed useful for NSAs downstream on the reservation workflow or modified attributes that are the result of evaluating existing message security attributes

The *sessionSecurityAttr* element is used to add additional security attributes to the NSI message header; it functions as a container for the individual attributes. The context where the

sessionSecurityAttr is evaluated is indicated by the *type* attribute. In the previous section the *urn:ogf:nsi:security:attr:originatingId* attribute type was defined with a specific behaviour that all NSAs can understand. In this section we define the *urn:ogf:nsi:security:attr:realm* attribute type that allows a *sessionSecurityAttr* element to be scoped within a specific authorization realm. NSAs that are members of an authorization realm will understand the contents of the element and use them appropriately. Those NSAs that are not a member can ignore the content, but should follow the transparency rules.

New *sessionSecurityAttr* element types can be defined and used as needed. With the existing transparency rules in place, these newly defined attributes will be seamlessly propagated to all NSAs participating in a specific reservation workflow. NSAs needing to interpret the new attributes can do so without impact to other NSAs in the Service Plane.

As an example, here is a *sessionSecurityAttr* element definition from an authorization realm "*http://idp.example.net*", with an *Attribute* element named *urn:mace:dir:attribute-def:eduPersonAffiliation*, and an *AttributeValue* of "*student*".

```
<sessionSecurityAttr type="urn:ogf:nsi:security:attr:realm" name="http://idp.example.net">
    <saml:Attribute Name="urn:mace:dir:attribute-def:eduPersonAffiliation"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
        <saml:AttributeValue xsi:type="xsd:string">student</saml:AttributeValue>
    </saml:Attribute>
</sessionSecurityAttr>
```

Figure 10 – example element *sessionSecurityAttr* element.

The following sections describe how to utilize the *sessionSecurityAttr* element to convey realm specific authorization information for the primary authorization use cases.

7.2.1 Authorization using OAuth

OAuth provides a method for clients to access a protected resource on behalf of a resource owner. Before a client can access a protected resource, it must first obtain an authorization grant from the resource owner, it can then be exchanged for an access token. This access token represents the grant's scope, duration, and other attributes associated with the authorization grant. A client then accesses the protected resource by presenting the access token to the resource server. See Figure 11 for one example of an OAuth abstract protocol flow, with more details available in [RFC6749].

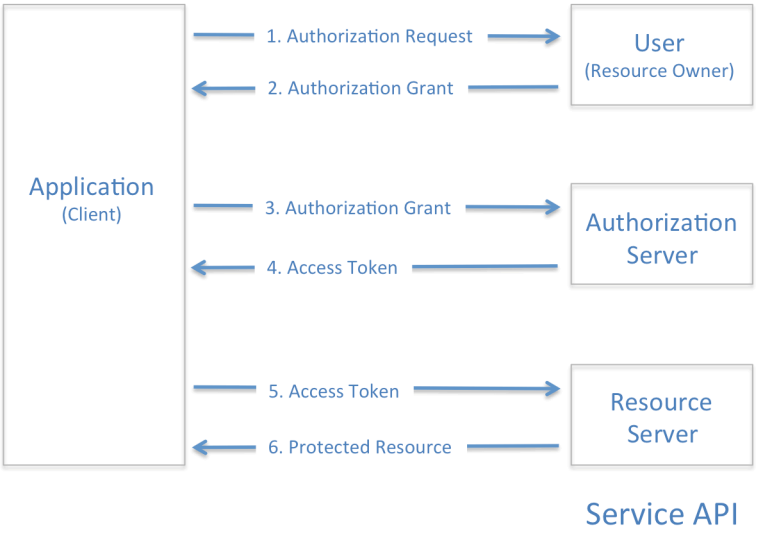

Figure 11 – OAuth 2.0 abstract protocol flow.

In some cases, a client can directly present its own credentials to an authorization server to obtain an access token without obtaining an authorization grant from a resource owner. An access token provides an abstraction, replacing different authorization constructs (e.g., username and password, assertion) for a single token understood by the resource server. This abstraction enables access tokens to be issued that are valid for a short time period, as well as removing the resource server's need to understand a wide range of authentication schemes.

OAuth assumes a point-to-point interaction model between an application (i.e. Originating Entity within NSI) and the actors within the protocol (i.e. resource owner, authorization server and resource server all OAuth components outside of NSI). An application uses SSL/TLS for secure communications with the authorization server and the resource server. An application is responsible for maintaining the context of the access token (i.e. it must know the resource server corresponding to the access token). An authorization server understands the concept of 'realm', so a single access token can grant access to resources on multiple resource servers if they were all part of the same realm.

An application client is responsible for maintaining the secrecy of the access token as an intercepted token can be used to gain access to resources. A limited lifetime is assigned to each access token to reduce the window of vulnerability for an intercepted token.

7.2.1.1 OAuth Attributes

NSI deployments can use OAuth as an authorization mechanism for granting access to network resources. In this case, the trusted Service Plane will provide secure transport between the Application and the Resource Server (Network Resource Manager or other service provider component). NSI does not participate in the protocol except for the transport of OAuth access tokens, and the return of any related OAuth error messages.

An Originating Entity issuing a reservation request to a uRA is responsible for obtaining any access tokens needed for resources associated with the reservation. However as the Originating Entity may not know which domain Resource Servers will be selected ahead of a path-computation action, the Originating Entity can effectively only provide access tokens (if needed) for resources it explicitly requests for. For instance, an Originating Entity may act as a third-party proxy for a circuit request, and may need access tokens from both the end-to-end source and

destination site Resource Servers in order for the circuit request to be successful. Another example might be that the Originating Entity was associated with a specific collaboration that could request for privileged resources along a path. In this case the Originating Entity could specify the path in the request using an Explicit Route Object, and provide the necessary access tokens for the corresponding Resource Servers. Figure 12 shows the abstract OAuth protocol flow using the NSI Service Plane as a secure transport between the Originating Entity and Resource Server associated with a network's uPA. Obtaining the tokens may require (1) communicating with multiple authorization servers depending on the nature of the reservation (endpoints used and authorization realms involved), returning possibly multiple (2) access tokens applying to different authorization domains. The Originating Entity (3) passes all access tokens associated with the request to the uRA that populates them in *sessionSecurityAttr* elements of the *nsiHeader* element. The access tokens are (4) passed down the reservation workflow in the NSI request to a uPA. The uPA (5) extracts the access tokens applicable to its associated realms, (6) queries the Authorization server to determine whether the token is valid and whether the Originating Entity has been granted access to the resources associated with the reservation. If the Authorization Server approves the use of the requested resources, and those resources are available for the reservation, the uPA holds the resources and (7) sends a confirmation back to the originating uRA as per the standard NSI CS reservation workflow.

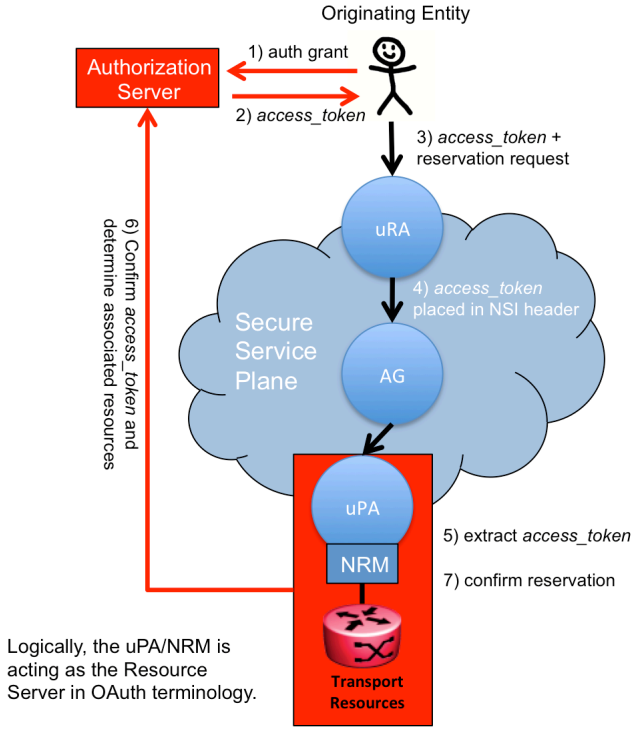

Figure 12 – OAuth protocol flow using NSI.

NSI has the flexibility to support an arbitrary number of Authorization Servers. Each Authorization Server is identified by a unique realm.

OAuth related tokens are included within the *nsiHeader* using the *sessionSecurityAttr element in the following way:*

Parameter	Type	Mandatory	Description
name	Attr	True	The *sessionSecurityAttr.name* attribute contains a unique OAuth provider 'realm' identifier.
type	Attr	True	The *sessionSecurityAttr.type* attribute contains an NSI security attribute type identifier of *urn:ogf:nsi:security:attr:realm* following the SAML type identifier naming format.
Attribute	Elem	True	The child SAML *Attribute* element contains an OAuth access_token(s) associated with the specified realm and needed to secure the target resources of the reservation. Other information that may be needed as part of the authorization access can be included in additional attributes.

The method by which an Originating Entity utilizes the access token to authenticate against a Resource Server (i.e. Network Resource Manager/Network Management System) depends on the type of access token issued by the Authorization Server. Specifications [RFC 6749] and [RFC 6750] describe this in additional detail.

At a minimum, the Originating Entity is required to include an OAuth *access_token* in the SAML *Attribute* element. An example of this is shown in the table below.

Parameter	Type	M/O	Description
Name	Attr	M	The *Attribute.Name* attribute contains the string "*access_token*" as defined in the OAuth specification [RFC 6749].
NameFormat	Attr	M	The *Attribute.NameFormat* attribute contains the type identifier *urn:oasis:names:tc:SAML:2.0:attrname-format:basic* indicating that the *Attribute.Name* is a basic name string.
AttributeValue	Elem	M	The SAML *AttributeValue* element contains an OAuth *access_token* value encoded as a string.

Any other OAuth related parameters can be included using a similar method. Additional OAuth tokens for different realms can be included in the *nsiHeader* by populating additional *sessionSecurityAttr* elements.

The following is an example of an OAuth *access_token* security attribute for the realm "`http://idp.example.net/oauth`" with a value of "`2YotnFZFEjr1zCsicMWpAA`".

```
<sessionSecurityAttr type="urn:ogf:nsi:security:attr:realm"
        name="http://idp.example.net/oauth">
    <saml:Attribute Name="access_token"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xsd:string">
            2YotnFZFEjr1zCsicMWpAA
        </saml:AttributeValue>
    </saml:Attribute>
</sessionSecurityAttr>
```

Figure 13 – OAuth *access_token* encoding in *sessionSecurityAttr* element.

NSI does not specify the form of the Originating Entity/uRA interface, and therefore, cannot specify how these OAuth access tokens are passed to a uRA. It is left up to the specific implementation of a uRA. The Originating Entity must be able to pass multiple *realm/access_token* pairs needed to utilize resources associated with the connection request.

7.2.1.2 OAuth Error Handling

[RFC 6749], Section 7 Accessing Protected Resources, describes the structure of error messages returned by a Resource Server in response to a failed access attempt. [RFC6750]

defines three specific authorization errors that can be returned from a Resource Server. There are three error fields associated with an error in the OAuth protocol:

error (REQUIRED)

- Is a single ASCII [USASCII] error code from the set defined in IETF RFC 6749 [RFC6749], and extended sets contained in IETF RFC 6750 [RFC6750]. For example, "invalid_request", "invalid_token" and "insufficient_scope".

error_description (OPTIONAL)

- Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred.

error_uri (OPTIONAL)

- A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error.

OAuth related authorization errors are populated in a *serviceException* element the following way:

Parameter	Type	Mandatory	Description
nsaId	Elem	True	The id of the NSA that generated the OAuth service exception.
connectionId	Elem	True	The *connectionId* associated with the reservation impacted by this error.
serviceType	Elem	False	The service type identifying the applicable service description in the context of the NSA generating the error.
errorId	Elem	True	The error code "00302" to indicate a security authorization issue.
text	Elem	True	The text error description "*AUTHORIZATION_FAILURE*" plus any addition descriptive text deemed useful by the generating NSA.
variables	Elem	True	Includes all fields associated with the OAuth error (error, error_description, and error_uri), as well as the original *realm* and *access_token* provided in the request to giving context to the authorization error.

The NSI protocol utilizes the operation specific failed response (i.e. *reserveFailed*) to communicate Resource Server error messages from the uPA to the Originating Entity (via the uRA) using the NSI *ServiceException* element. An NSI CS standard *00302 AUTHORIZATION_FAILURE* error code [GFD.212] is used for the OAuth type of *ServiceException*. Below is an example showing how the *variables* element is populated with the application OAuth error information.

```
<serviceException>
    <nsaId>urn:ogf:network:example.net:2013:nsa:provider</nsaId>
    <connectionId>urn:uuid:59d6c0b2-a8e0-4583-ae8a-0fc84eb89f07</connectionId>
    <serviceType>
        http://services.ogf.org/nsi/2013/12/descriptions/EVTS.A-GOLE
    </serviceType>
    <errorId>00302</errorId>
    <text>AUTHORIZATION_FAILURE</text>
    <variables>
        <variable type="urn:ogf:nsi:security:attr:realm">
            <value>http://idp.example.net/oauth</value>
        </variable>
        <variable type="access_token">
            <value>2YotnFZFEjr1zCsicMWpAA</value>
        </variable>
        <variable type="error">
```

```
            <value>invalid_token</value>
        </variable>
        <variable type="error_description">
            <value>Supplied token is invalid</value>
        </variable>
        <variable type="error_uri">
            <value>http://idp.example.net/oauth/errors/invalid_token.html</value>
        </variable>
    </variables>
</serviceException>
```

Figure 14 – *variables* element populated with the application OAuth error information.

7.2.2 Attribute Certificates

Authorization or Attribute certificates [RFC3281] are digital certificates containing signed attributes granted to the holder by the issuer of the certificate. The issuer (resource owner for example) creates the certificate with their private key, signing the attributes they would like to assign the holder (user/application). This certificate can then be verified by any Resource Server using the issuer's public key, instantly having access to the list of attributes associated with the user without needing to query an Authorization Server.

In contrast to OAuth, attribute certificates carry the authorization information in the certificate itself, whereas OAuth requires the access_token be used to lookup the user's authorization information. The user workflow for obtaining an authorization certificate can be considered similar to OAuth:

- The Originating Entity's identity is authenticated (typically using their X.509 certificate) by the Authorization Server (*Attribute Authority*).
- The Originating Entity requests an authorization grant for a set of resources, roles, etc. from the Authorization Server.
- The Authorization Server validates the Originating Entity's access, generates a certificate listing a set of attributes associated with the Originating Entity (access permissions expressed as attributes), and returns the generated certificate to the Originating Entity.
- The Originating Entity presents this attribute certificate to the Resource Server along with an access request.
- The Resource Server uses the Authorization Server's public key to verify that the presented attribute certificate was created by the Authorization Server, and utilizes the Originating Entity's public key to validate that the certificate corresponds to the requester. Once verified, the Resource Server grants access based on the attributes presented in the attribute certificate.

Attribute certificates can be populated in a *sessionSecurityAttr* element in the following way:

Parameter	Type	Mandatory	Description
Type	Attr	True	The *sessionSecurityAttr.type* attribute contains the NSI security attribute type identifier *urn:ogf:nsi:security:attr:realm*, following the naming format used in standard SAML type identifiers.
Name	Attr	True	The *sessionSecurityAttr.name* attribute should contain the DN of the issuing Attribute Authority to identify the security realm. This could be replaced with any string uniquely identifying the associated realm.
Attribute	Elem	True	The child SAML *Attribute* element contains the *base64Binary* encoded attribute certificate associated with the target resources of the reservation.

The SAML *Attribute* element would be populated with the attribute certificate as described in the table below.

Parameter	Type	M/O	Description
Name	Attr	M	The *Attribute.Name* attribute contains the string "*attributeCertificate*" indicating an attribute certificate [RFC 3281] is present in the *AttributeValue* member element.
NameFormat	Attr	M	The *Attribute.NameFormat* attribute contains the type identifier *urn:oasis:names:tc:SAML:2.0:attrname-format:basic* indicating that the *Attribute.Name* is a basic name string.
AttributeValue	Elem	M	The SAML *AttributeValue* element contains the *base64Binary* encoded attribute certificate.

The following example shows an attribute certificate included in the *sessionSecurityAttr* element using *base64Binary* encoding:

```
<sessionSecurityAttr type="urn:ogf:nsi:security:attr:realm"
        name="/C=US/O=EXAMPLE/OU=Grid Resources/CN=attributeauthority@example.net">
    <saml:Attribute Name="attributeCertificate"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xsd:base64Binary">
            MIICiDCCAXACCQDE+9eiWrm62jANBgkqhkiG9w0BAQQFADBFMQswCQYDVQQGEwJV
            UzESMBAGA1UEChMJTkNTQS1URVNUMQ0wCwYDVQQLEwRVc2VyMRMwEQYDVQQDEwpT
            UC1TZXJ2aWNlMB4XDTA2MDcxNzIwMjE0MVoXDTA2MDcxODIwMjE0MVowSzELMAkG
            A1UEBhMCVVMxEjAQBgNVBAoTCU5DU0EtVEVTVDENMAsGA1UECxMEVXNlcjEZMBcG
            A1UEAwwQdHJzY2F2b0B1aXVjLmVkdTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkC
            gYEAv9QMe4lRl3XbWPcflbCjGK9gty6zBJmp+tsaJINM0VaBaZ3t+tSXknelYife
            nCc2O3yaX76aq53QMXy+5wKQYe8Rzdw28Nv3a73wfjXJXoUhGkvERcscs9EfIWcC
            g2bHOg8uSh+Fbv3lHih4lBJ5MCS2buJfsR7dlr/xsadU2RcCAwEAATANBgkqhkiG
            9w0BAQQFAAOCAQEAdyIcMTob7TVkelfJ7+I1j0LO24UlKvbLzd2OPvcFTCv6fVHx
            Ejk0QxaZXJhreZ6+rIdiMXrEzlRdJEsNMxtDW8++sVp6avoB5EX1y3ez+CEAIL4g
            cjvKZUR4dMryWshWIBHKFFul+r7urUgvWI12KbMeE9KP+kiiiiTskLcKgFzngw1J
            selmHhTcTCrcDocn5yO2+d3dog52vSOtVFDBsBuvDixO2hv679JR6Hlqjtk4GExp
            E9iVI0wdPE038uQIJJTXlhsMMLvUGVh/c0ReJBn92Vj4dI/yy6PtY/8ncYLYNkjg
            oVN0J/ymOktn9lTlFyTiuY4OuJsZRO1+zWLy9g==
        </saml:AttributeValue>
    </saml:Attribute>
</sessionSecurityAttr>
```

Figure 15 – attribute certificate included in the *sessionSecurityAttr* element.

7.2.2.1 Attribute Certificate Error Handling

If a uPA determines the uRA has not presented a valid Attribute Certificate for the requested resources it should return a failed message with a *serviceException* element populated as follows:

Parameter	Type	Mandatory	Description
nsaId	Elem	True	The id of the NSA that generated the authroization service exception.
connectionId	Elem	True	The *connectionId* associated with the reservation impacted by this error.
serviceType	Elem	False	The service type identifying the applicable service description in the context of the NSA generating the error.
errorId	Elem	True	The error code "00302" to indicate a security authorization issue.
text	Elem	True	The text error description "*AUTHORIZATION_FAILURE*" plus

			any addition descriptive text deemed useful by the generating NSA.
variables	Elem	True	Include a list of zero or more security realms for which valid authorization credentials should be presented for access to requested resources.

The NSI protocol utilizes the operation specific failed response (i.e. *reserveFailed*) to communicate Resource Server error messages from the uPA to the Originating Entity (via the uRA) using the NSI *ServiceException* element. An NSI CS standard *00302 AUTHORIZATION_FAILURE* error code [GFD.212] is used for communicating this type of *ServiceException*. Below is an example of how the *variables* element is populated with the authorization error information.

```
<serviceException>
    <nsaId>urn:ogf:network:example.net:2013:nsa:provider</nsaId>
    <connectionId>urn:uuid:59d6c0b2-a8e0-4583-ae8a-0fc84eb89f07</connectionId>
    <serviceType>
        http://services.ogf.org/nsi/2013/12/descriptions/EVTS.A-GOLE
    </serviceType>
    <errorId>00302</errorId>
    <text>AUTHORIZATION_FAILURE</text>
    <variables>
        <variable type="urn:ogf:nsi:security:attr:realm">
            <value>
                /C=US/O=EXAMPLE/OU=Grid Resources/CN=idp@example.com
            </value>
        </variable>
    </variables>
</serviceException>
```

Figure 16 – *variables* element populated with the authorization error information.

8 Glossary

Aggregator NSA (AG)	The Aggregator NSA is a Provider Agent that acts as both a requester and provider NSA. It can service requests from other NSA, perform path finding, and distribute segment requests to child NSA for processing.
Client Authenticated TLS	Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are protocols that provide communication security. TLS is mandated in NSI for communication between NSAs.
Connection Service (CS)	The NSI Connection Service is a service that allows an RA to request and manage a Connection from a PA. See [OGF NSI-CS].
DistinguishedName (DN)	A Distinguished Name is a unique name for an entry in a Directory Service and is used within X.509 certificates to identify the subject (owner) of the certificate.
Network	A Network is an Inter-Network topology object that describes a set of STPs with a Transfer Function between STPs.
Network Service Agent (NSA)	The Network Service Agent is a concrete piece of software that sends and receives NSI Messages. The NSA includes a set of capabilities that allow Network Services to be delivered.
Network Service Interface (NSI)	The NSI is the interface between RAs and PAs. The NSI defines a set of interactions or transactions between these NSAs to realize a Network Service.
Requester/Provider Agent (RA/PA)	An NSA acts in one of two possible roles relative to a particular instance of an NSI. When an NSA requests a service, it is called a Requester Agent (RA). When an NSA realizes a service, it is called a Provider Agent (PA). A particular NSA may act in different roles at different interfaces.

Originating Entity	Any entity that originates a service request in to uPA. This could be a person, institution, software application, etc. This 'user' is not a formal part of the NSI protocol since NSI does not define the interface between the uRA and the Originating Entity.
Ultimate PA (uPA)	The ultimate PA is a Provider Agent that has an associated NRM.
Ultimate RA (uRA)	The Ultimate RA is a Requester Agent is the originator of a service request.
XML Schema Definition (XSD)	XSD is a schema language for XML. See [W3C XSD]
eXtensible Markup Language (XML)	XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

9 Contributors

Hans Trompert, SURFnet
John H. MacAuley, ESnet
Henrik Thostrup Jensen, NORDUnet
Guy Roberts, GÉANT
Chin Guok, ESnet

10 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights, which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

11 Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

12 Full Copyright Notice

approval of its board, may remove this restriction for inclusion of OGF document content for the purpose of producing standards in cooperation with other international standards bodies.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

13 References

[GFD.212] OGF GFD-I.212, Network Service Interface Connection Service, v2.0.

[GFD.213] OGF GFD-I.213, Network Services Framework v2.0.

[GFD.217] OGF GFD-I.217 NSI Signaling and Path Finding

[NSI Policy] OGF GFD-R (gfd-r-nsi-policy-v7), Network Service Interface Policy, NSI-WG 2015.

[RFC3281] IETF RFC 3281, An Internet Attribute Certificate Profile for Authorization, S. Farrell, R. Housley, April 2002.

[RFC6749] IETF RFC 6749, The OAuth 2.0 Authorization Framework, D. Hardt, Ed., October 2012.

[RFC6750] IETF RFC 6750, The OAuth 2.0 Authorization Framework: Bearer Token Usage, M. Jones, October 2012.