

NSI Authentication and Authorization

Status of This Document

Grid Forum Document (GFD), Recommendation (R).

This document should be read in conjunction with OGF GFD.212 Network Service Interface Connection Service, v2.0.

Copyright Notice

Copyright © Open Grid Forum (2008-2015). All Rights Reserved.

Trademark

OGSA is a registered trademark and service mark of the Open Grid Forum.

Abstract

This document outlines security requirements placed on Network Service Agents (NSA) when participating in the Network Services Interface (NSI) Connection Service protocol. Also discussed are the impacts of end-user authentication and authorization mechanism on the NSA and the NSI CS protocol through the use of the existing NSI security attributes.

Contents

Contents	1
1 Introduction.....	2
2 Notational Conventions	2
3 Requirements	2
4 Fundamental Principles of Security in NSI.....	3
5 Access to the Service Plane	4
6 Authorization Policies.....	6
7 Security Attributes	7
7.1 Originating Entity Identifier	9
7.2 Authorization	11
8 Security Proxy NSA.....	18
9 Glossary	18
10 Contributors.....	19
11 Intellectual Property Statement	19
12 Disclaimer.....	19
13 Full Copyright Notice	20
14 References	20

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.

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 applied to the NSI Connection Service and should be read in conjunction with GFD-R.212 Network Service Interface Connection Service version 2.0 [GFD.212], Open Grid forum GFD-I.213, Network Services Framework v2.0 [GFD.213] and OGF 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. 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 traveling through the NSI Service Plane MUST be ensured.
- It is REQUIRED that all access to the NSI Service Plane is authorized by the uRA.
- It is REQUIRED that access to a network’s Transport Plane resources is authorized by the uPA representing that network.
- It MUST be possible to identify the Originating Entity of an NSI request.
- It MUST be possible to identify the originating uRA of an NSI request.
- Authorization attributes MUST be transported transparently over the NSI Service Plane.

4 Fundamental Principles of Security in NSI

The 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.

NSI Service Plane security is based on transitive trust: I trust my neighbours and the neighbours they trust. As a result, any peering process should take this fact into consideration when adding new peers to the NSI Service Plane.

NSI uses Client Authenticated TLS as a transport protocol to ensure the integrity and confidentiality of the messages traveling through the 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 it is communicating with a trusted peer, and that all encrypted communications between the peer NSA and is in fact coming from the expected NSA.

Peer NSA MUST authenticate each other using Client Authenticated TLS.

All traffic between two peering NSAs MUST be encrypted using TLS while in transit.

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 NSAs can agree on the use of a common trusted Certificate Authority, while others will just exchange certificates on a per peer basis. An advantage of this second method is that it also allows for self-signed certificates.

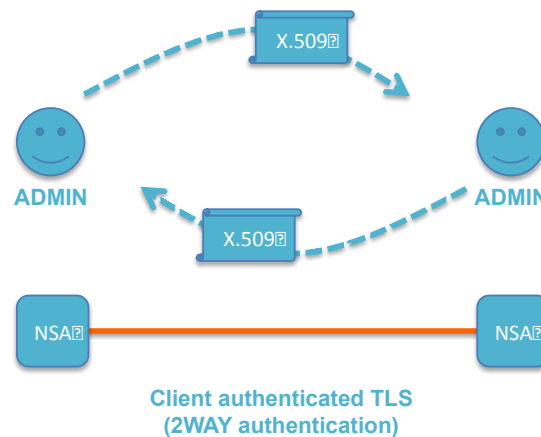


Figure 1 - 2-WAY TLS between peer NSA.

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 individually authorize a peer, before allowing them access to the Service Plane.

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

- An Aggregator is contractually obligated to accept 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 is contractually obligated to determine the identity of the requesting user and authorize that user's access to the Service Plane. The uRA does not authorize a user's access to network resources.
- A uPA is contractually obligated to authorize a user's access to Transport Plane resources in its associated network.

How the uRA and uPA authorize a user's access is a deployment decision and is out of scope of the NSI protocol. Figure 2 below illustrates these security concepts.

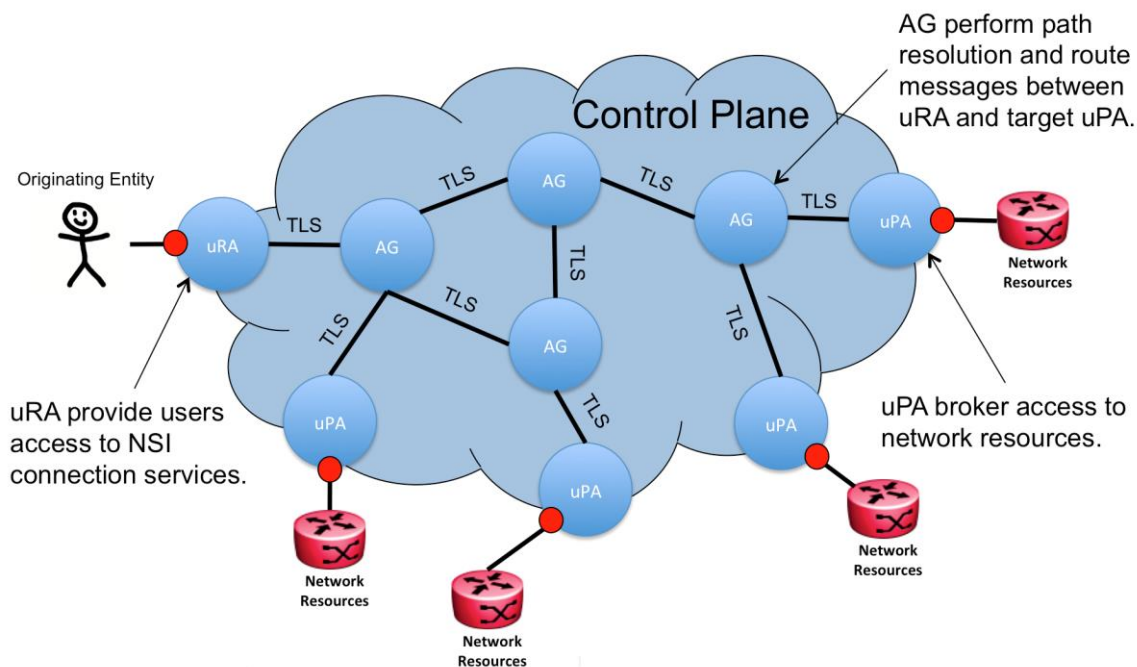


Figure 2 – Security in the Service Plane.

A group of NSAs that together form a Service Plane will be self-regulating. Misbehaving NSAs will 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 removing an NSA deemed to be “misbehaving”.

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]. An Ultimate Requester Agent (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 the 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 access to the Service Plane by the Originating Entity.

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 reference to 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 a reference to the Originating Entity's identity.

It MUST be possible for an 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.

Any NSA can authenticate an Originating Entity as long as this NSA is a part of a secure NSI Service Plane as describe earlier in this document. This includes authentication done by user applications that have an integrated NSA.

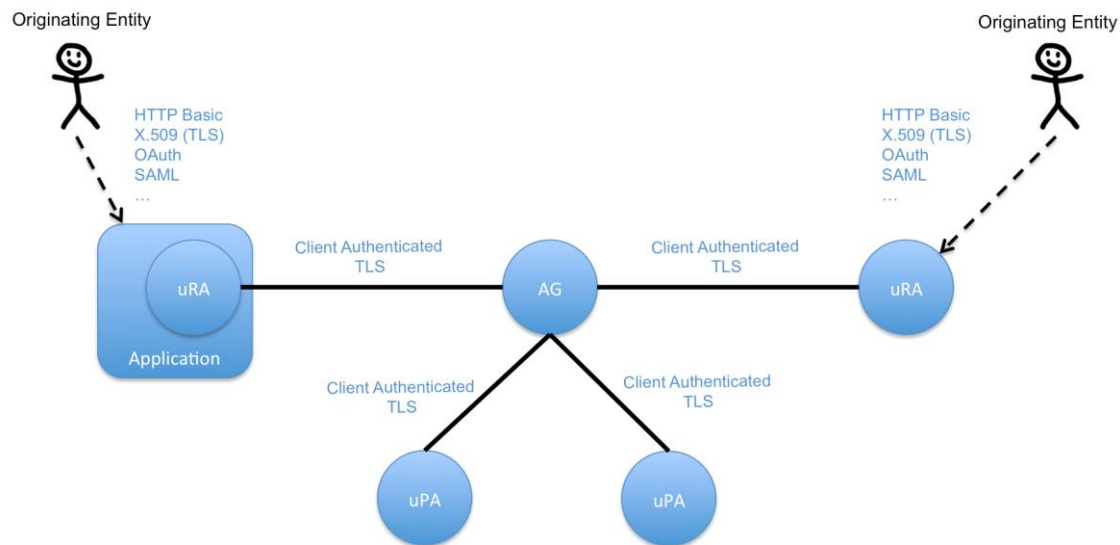


Figure 3 - Authenticated access to Service Plane.

It is not required that every NSA along the Service Plane path of an NSI message flow 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 Service Plane path wants to hide the identity 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. The NSA is allowed to rewrite the NSI message header to make it look like that NSA is the originating NSA.

An intermediate NSA in an NSI Connection Service message flow MAY replace the Originating Entity's identity reference with another identity reference only if this new entity is willing to 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 Policies

Every NSA is allowed to 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 database. Such a policy database can either be local to the NSA or part of an authentication and authorization infrastructure where policies apply to a set of NSA. Depending on the deployment a combination of local and/or, possible multiple, remote policy sources can be used to authorize NSI requests. How authorization policies are administrated is deployment specific.

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 constrain usage.

The PA MUST (subject to policy rules) authorizes all RA to PA Connection Service request messages.

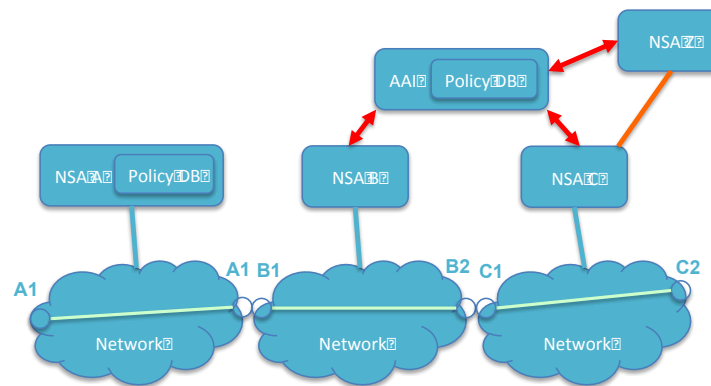


Figure 4. Policy Database Deployment example

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 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 *AttributeStatementType* 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 the Service Plane path can add additional security attributes to a message; these are either new attributes that are deemed useful for NSAs downstream on the Service Plane path, 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 domain that will participate in that specific NSA message workflow.

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 security realm. In this case, the *sessionSecurityAttr* element *type* attribute

identifies the element as domain specific and the name identifies the security realm itself. NSAs that are part of that security domain 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 domains, and more than one NSA can be part of the same authorization domain.

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	M/O	Description
name	Attr	M	The <i>sessionSecurityAttr.name</i> attribute contains the NSA identifier of the uRA issuing the request.
type	Attr	M	The <i>sessionSecurityAttr.type</i> attribute contains the NSI security attribute type identifier of " <i>urn:ogf:nsi:security:attr:originatingId</i> " following the SAML type identifier naming format.
Attribute	Elem	M	The child SAML <i>Attribute</i> 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	M/O	Description
Name	Attr	M	The <i>Attribute.Name</i> attribute contains the MACE identifier " <i>urn:mace:dir:attribute-def:eduPersonTargetedID</i> " indicating this <i>Attribute</i> is modelling a SAML/Shibboleth target identifier value.
NameFormat	Attr	M	The <i>Attribute.NameFormat</i> attribute contains the type identifier of " <i>urn:oasis:names:tc:SAML:2.0:attrname-format:uri</i> " indicating that the <i>Attribute.Name</i> is a proper SAML URI.
AttributeValue	Elem	M	The child SAML <i>AttributeValue</i> element contains the persistent reference to the user identity information as specified on the uRA. This <i>AttributeValue</i> element is populated with a SAML <i>NameID</i> element with the attribute <i>NameID.Format</i> 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 domain.

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

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 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      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:Subject>
  <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:Subject>
</sessionSecurityAttr>

```

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

7.2 Authorization

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.

The goal of the NSI protocol is to transparently pass the required authorization information from the Originating Entity (via a uRA) to the NRM (via uPA). 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. Security related attributes introduced by the uRA are securely transported to all uPA involved in the reservation.

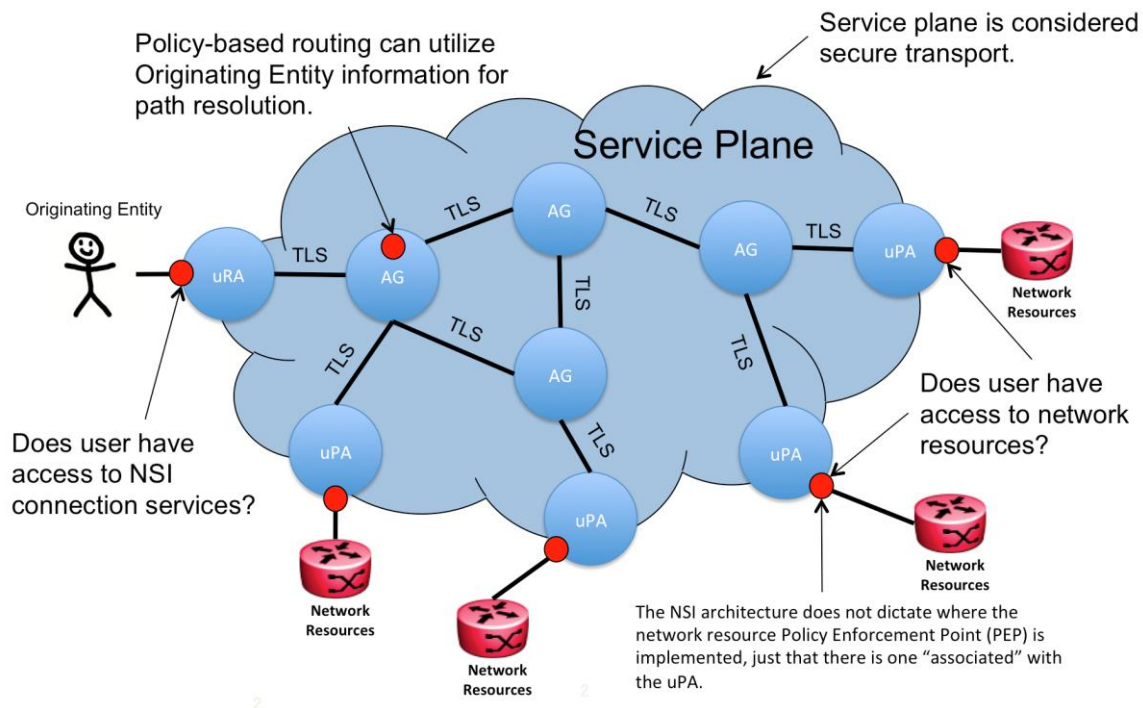


Figure 9 – Service Plane security.

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 DBs, 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 attribute certificates
- Group membership information

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

The *sessionSecurityAttr* element is used to add the additional security attributes to the NSI message header is described in an XML schema, and functions as a container for the individual attributes. The context where the security attribute is to be evaluated is indicated by the *sessionSecurityAttr.type* attribute that dictates the type of security attribute represented by the *sessionSecurityAttr* element. In the previous section the “urn:ogf:nsi:security:attr:originatingId” attribute type was defined with a specific behaviour all NSAs could understand. In this section we define the “urn:ogf:nsi:security:attr:realm” attribute type that allows a *sessionSecurityAttr* element to be scoped to a specific authorization realm. NSAs that are members of the identified authorization realm will understand the contents of the element and use them appropriately, while those that are not a member, need not concern themselves with the content, but still follow the transparency rules if needed. 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.

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

In 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 and then exchange the authorization grant for an access token. This access token represents the grant's scope, duration, and other attributes granted by the authorization grant. A client then accesses the protected resource by presenting the access token to the resource server. See Figure 10 for this OAuth abstract protocol flow, with more details available in [RFC6749].

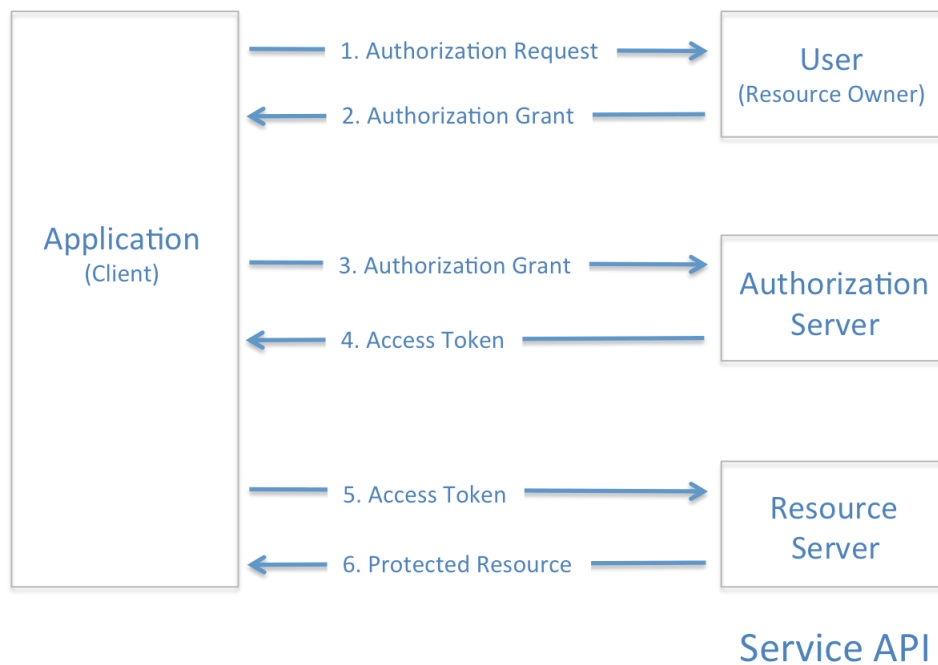


Figure 10 – OAuth 2.0 abstract protocol flow.

In some cases, a client can directly present its own credentials to an authorization server to obtain an access token without having to first obtain an authorization grant from a resource owner. The 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 which 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 the application and the actors within the protocol. The application client uses SSL/TLS for secure communications with the authorization server and the resource server, and is responsible for maintaining the context of the access token (i.e. it must know the resource server corresponding to the access token). The authorization server does have the concept of “realm” so a single access token could grant access to resources on multiple resource servers if they were all part of the same realm.

The 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 Implementing OAuth in NSI

In those NSI deployments using OAuth as an authorization mechanism for granting user/application access to network resources, the NSI Service Plane infrastructure takes on the role of secure transport between the Application actor (user) and the Resource Server actor (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 application client (Originating Entity) interfacing with the uRA is responsible for obtaining any access tokens needed for resources associated with their reservation. This may imply communicating with multiple authorization servers depending on the nature of the reservation (endpoints used), and the number of authorization domains implemented in the network. The Application passes all access tokens associated with the request to the uRA that populates them in *sessionSecurityAttr* elements of the *nsiHeader* element. These access tokens are passed down the reservation tree with the NSI request to the uPA. The uPA extracts the access tokens applicable to the local domain, queries the Authorization server to determine if the token is valid and if the user is granted access to the resources associated with the reservation.

Figure 11 below shows the abstract OAuth protocol flow using NSI as a secure transport between the Application and Resource Server associated with a network's uPA.

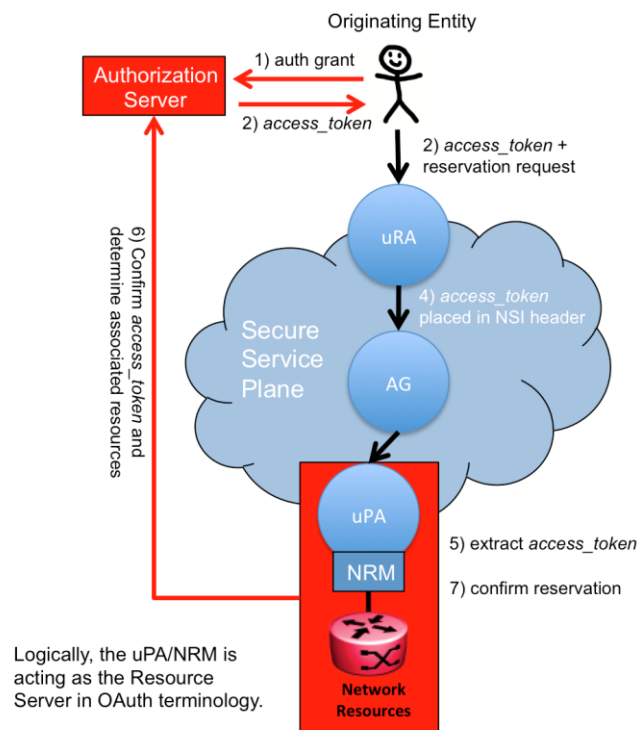


Figure 11 – OAuth protocol flow using NSI.

In an ideal NSI deployment, a single federated authorization solution will have been implemented, requiring allocation of access tokens from only a single authorization domain. However, in most deployment cases this will not happen. As a result, NSI supports the ability to have an Authorization Server per uPA at a minimum, but is flexible enough to have an arbitrary number of Authorization Servers, each uniquely identified by naming scheme.

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

Parameter	Type	M/O	Description
name	Attr	M	The <i>sessionSecurityAttr.name</i> attribute contains the unique OAuth provider “realm” identifier.
type	Attr	M	The <i>sessionSecurityAttr.type</i> attribute contains the NSI security attribute type identifier of “ <i>urn:ogf:nsi:security:attr:realm</i> ” following the SAML type identifier naming format.

Attribute	Elem	M	The child <i>SAML Attribute</i> element contains the OAuth <i>access_token(s)</i> 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 in which the application client utilizes the access token to authenticate with the resource server depends on the type of access token issued by the authorization server. Specifications [RFC 6749] and [RFC 6750] describe this in additional details.

At a minimum the application will need to include the OAuth *access_token* in the *SAML Attribute* element, which can be accomplished the following way:

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

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

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

```
<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">
      2YotnFZFEjrlzCsicMWpAA
    </saml:AttributeValue>
  </saml:Attribute>
</sessionSecurityAttr>
```

Figure 12 – OAuth *access_token* encoding in *sessionSecurityAttr* element.

NSI does not specify the form of the Application/uRA interface, and therefore, cannot specify how these OAuth access tokens specifically get into the uRA. It is therefore left up to uRA specific implementation. The Originating Entity must be able to specify across the Application/uRA interface multiple *realm/access_token* pairs depending on the number required to utilize resources associated with the connection request.

[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, and [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)

- A single ASCII [USASCII] error code from a defined set in [RFC6749]. For example, “invalid_request”, “invalid_token”, and “insufficient_scope” are three used by [RFC6750].

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.

The NSI protocol utilizes the operation specific failed response (i.e. *reserveFailed*) to transport these Resource Server specific error messages back from the uPA through the uRA to the requesting Application via the NSI *ServiceException* element. A standard 00302 *AUTHORIZATION_FAILURE* error code is used for the OAuth type of *ServiceException*. Figure 13 below shows 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>2YotnFZFEjrlzCsicMWpAA</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 13 – OAuth Resource Server error mapped to an NSI *ServiceException*.

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

Parameter	Type	M/O	Description
nsaId	Elem	M	The id of the NSA that generated the OAuth service exception.
connectionId	Elem	M	The <i>connectionId</i> associated with the reservation impacted by this error.
serviceType	Elem	O	The service type identifying the applicable service description in the context of the NSA generating the error.
errorId	Elem	M	The error code "00302" to indicate a security authorization issue.
text	Elem	M	The text error description " <i>AUTHORIZATION_FAILURE</i> " plus any addition descriptive text deemed useful by the generating NSA.
variables	Elem	M	Includes all fields associated with the OAuth error including the original <i>realm</i> and <i>access_token</i> provided in the request to give context to the error.

7.2.2 Authorization using Attribute Certificates

Attribute (or authorization) certificates 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 policy enforcement points using the issuer's public certificate, 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 user/application's (Originating Entity) identity is authenticated (typically using their X.509 certificate) by Authorization Server (*Attribute Authority*).
- The user/application requests an authorization grant to a set of resources, roles, etc. from Authorization Server.
- The Authorization Server validates user/application's access, generates a certificate listing a set of attributes associated with the user (access permissioned modeled in attributes), and returns the generated certificate to user/application.
- The user/application presents this authorization certificate to the Resource Server along with access request.
- The Resource Server uses the Authorization Server's public key to verify the presented authorization certificate was indeed created by the Authorization Server, and utilizes the user/application's public key to validate the certificate corresponds to the requester. Once verified, the Resource Server grants access based on the attributes presented in the authorization certificate.

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

Parameter	Type	M/O	Description
type	Attr	M	The <code>sessionSecurityAttr.type</code> attribute contains the NSI security attribute type identifier of " <code>urn:ogf:nsi:security:attr:realm</code> " following the SAML type identifier naming format.
name	Attr	M	The <code>sessionSecurityAttr.name</code> 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	M	The child <i>SAML Attribute</i> element contains the base64Binary encoded attribute certificate associated with the target resources of the reservation.

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="authorizationCertificate"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
    <saml:AttributeValue xsi:type="xsd:base64Binary">
      MIICiDCCAXACCQDE+9eiWrm62jANBgkqhkiG9w0BAQQFADBFMQswCQYDVQQGEwJV
      UzESMBAGAlUEChMJKNTQs1URVNUMQ0wCwYDVQQLEwRVc2VyMRMwEQYDVQQDEwpt
      UC1TZXJ2aWNlMB4XDTA2MDcxNzIwMjE0MVoXDTA2MDcxODIwMjE0MVoWszELMAkG
      A1UEBhMCMVVMxZjAQBGNVBAOTCU5DU0EtVEVTVDENMAsGA1UECxEVXNlcjEzMBCG
      A1UEAwQdHJzY2F2b0BlaXVjLmVkdTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwYkC
```

```

gYEAv9QMe4lRl3XbWPcflbCjGK9gty6zBJmp+tsaJINM0VaBaZ3t+tSXknelYife
nCc2O3yaX76aq53QMXy+5wKQYe8Rzdw28Nv3a73wfjXJXoUhGkvERcscs9EfiWcC
g2bHOG8uSh+Fbv3lHih4lBJ5MCS2buJfsR7dlr/xsadU2RcCAwEAATANBgkqhkiG
9w0BAQQFAAOCAQEAAdyIcMTob7TVkelfJ7+I1j0LO24U1KvbLzd2OPvcFTcv6fVHx
EjK0QxazXJhrez6+rIdiMXrEz1RdJEsNMxtDW8++sVp6avoB5EX1y3ez+CEAIL4g
cJvKZUR4dMryWshWIBHKFFul+r7urUgvWI12KbMeE9KP+kiiiiTskLcKgFzngw1J
selmHhTcTcCrcDocn5yO2+d3dog52vSotVFDBsBuvDixO2hv679JR6Hlqjtk4GExp
E9iVI0wdPE038uQIJJTXlhsMMLvUGVh/c0ReJBn92Vj4dI/yy6PtY/8ncYLYNkjg
oVN0J/ymOktn9lTlFyTiuY4OuJsZRO1+zWLy9g==
</saml:AttributeValue>
</saml:Attribute>
</sessionSecurityAttr>

```

8 Security Proxy NSA

Any NSA is allowed to take full responsibility for a NSI request message by rewriting the *originatingId* in such a way that it looks like that this NSA is the originating uRA of the reservation request, effectively hiding all NSAs in the workflow up to this point. In addition, the NSA may rewrite any security attributes within the *nsiHeader* if it is aware of the context of these attributes. This may be done to hide the originating user from the network, or perhaps add additional credentials to modify the user's access to network resources.

This Security Proxy NSA may:

- Replace the uRA identifier portion of the *originatingId* from the incoming NSI request message with its own NSA identifier.
- Replace the Originating Entity reference with another valid reference, thereby hiding the Originating Entity.
- Add, replace, or modify authorization attributes as needed, while maintaining the original intent of the transparency rule.

9 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 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.

10 Contributors

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

11 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.

12 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.

13 Full Copyright Notice

Copyright (C) Open Grid Forum (2012-2015). Some Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included as references to the derived portions on all such copies and derivative works. The published OGF document from which such works are derived, however, may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing new or updated OGF documents in conformance with the procedures defined in the OGF Document Process, or as required to translate it into languages other than English. OGF, with the 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.

14 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.

[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.