

Token Validation Service (TVS) Design and Architecture

Version 0.2

October 12, 2007

Editor – Yuri Demchenko

1. Background and general definitions

Token Validation Service (TVS) is a component of the Complex Resource Provisioning (CRP) architecture related to the generic AAA/Authorisation services.

Note. More information on CRP and GAAA-AuthZ will be added later.

The major goal of the TVS is to support token-based policy enforcement mechanisms during user access of the reserved service or network. (It is intended that TVS functionality will also support policy enforcement for the consumable resource).

TVS is used to check if a service/resource requesting subject or other entity, that posses/presents current token, has right/permission to access/use a resource based on advance reservation to which this token refers. During its operation TVS checks if a presented token has reference to a previously reserved resource and a request resource/service confirms to a reservation condition.

In a simple/basic scenario, TVS operates locally and checks local reservation table directly or indirectly using reservation ID (typically, Global Reservation Id - GRI). It is suggested that in multidomain scenario each domain may maintain Local Reservation ID (LRI) and its mapping to the GRI

In more advanced scenario TVS should allow creation of a TVS infrastructure to support tokens and token related keys distribution to support dynamic resource, users or providers federations.

2. TVS functions and general requirements

TVS design approach and architecture should allow/support the following possibility and functionality:

- 1) Simple/basic TVS functionality includes checking validity of a token.
- 2) Extended TVS functionality should allow token re-building when sending dataflow to or requesting service from the next domain.
Note. This functionality requires first developing a dynamic secure association management model.
- 3) Additionally, TVS may be required to support token or token key distribution at the reservation stage or at the stage of the reserved resource deployment/configuration.
- 4) Token building function is an incumbent function of the TVS and consequently it is supported by the Token Builder (TB or TVS-TB) component.

TB function should allow generating token key and token as derivative from the GRI. Additionally TB should allow generating token dynamically using token key and variable dataflow data, e.g. IP packets payload as in case of Token-based Networking (TBN).

5) TVS implementation should support both in-band dataflow token-based signalling and control plane signalling using XML-based tokens.

As a consequence of intended use for in-band token-based signalling, token key and token should be of fixed length.

6) TVS should maintain own run-time table “token – GRI – (LRI) – (token key)”, where LRI and token key are optional. Additionally TVS table may contain a status or validity period of the tuple “token – GRI – (LRI) – (token key)”.

GRI and/or LRI will link to actual local resource reservation table maintained by the resource reservation and management service and contain all necessary details.

This TVS table should be stored in non-volatile memory to keep its state between start-up/initialisation periods.

(Note. Security aspects of maintaining TVS table and TVS initialisation is a topic for further research and development)

7) TVS should allow integration into control plane or data plane using simple API. This should constitute a basic TVS functionality.

8) TVS should allow smooth integration into more general AAA infrastructure and support multidomain resource reservation/authorisation. In this case, TVS may be implemented as a component of more general multidomain complex resource authorisation infrastructure to support authorisation session management.

3. Example of the TVS use cases and operation

Currently 2 use cases provide a range of requirements to TVS functionality and implementation profiles.

3.1. TVS integration into OSCARS/DRAGON infrastructure

Figure 2.1 below illustrates how TVS can be used in OSCARS/DRAGON network provisioning infrastructure.

- 8) When a domain (or InterDomain Controller (IDC)) receives a service request, it extracts the token calls a PEP (additionally providing also any necessary access control related information).
- 9) In a simple scenario, when receiving the token, PEP request the TVS to validate the token. TVS performs this by checking “token – GRI – (LRI) – (token key)” table stored by TVS. TVS replies with the Boolean “Yes/No” answer.
 - This is a responsibility of VLSR and OSCARS to provide a requested service if PEP-TVS response is positive.
- 10) Token validity may expire automatically or token related TVS table entry can be invalidated by IDC. In this case PEP-TVS will return negative answer.

There is some background configuration and setup required but it will be provided as a part of TVS-PEP implementation

3.1. TVS integration into ForCES based Token Based Switch (TBS)

Information is provided in documents posted by Mihai Cristea.

Note. In this case part of TVS functions and components are hardware based but use the same API and messaging. Also the proposed token handling model allow for integration of the circuit provisioning and applications flow provisioning as different layers of the token based enforcement model.

4. TVS implementation

4.1. Initial stage 1

At the initial stage 1 we consider implementation of minimum functionality for the two use case described above.

TVS version 0.1 is implemented as a part of the general GAAAPI Java package and provides required functionality that can be integrated into the target network provisioning systems and applications, in particular OSCARS and DRAGON.

All basic TVS/TB functions are accessible and requested via Java API. This is required to make TVS reaction as quick as possible.

Some off-band TVS functions can be operated via WS/SOAP based interface as for example the TVS table programming.

The TVS API and interfaces implemented in TVSv0.0 are described below (some changes have been done since the initial design document).

4.2. API/internal TVS interface

a) TVS programming interface contain two basic commands:

```
SetEntryTVStable (GRI, (TokenValue | TokenKey)?, NotBefore?,  
NotOnOrAfter?)
```

Status: Not implemented yet

DeleteEntryTVStable (GRI, LRI?) -

b) Token Builder commands

GetToken (GRI, TokenKey?)

Status: Implemented

c) PEP-TVS interface

Boolean ValidateToken (TokenValue, GRI?, TokenKey?)

Boolean ValidateXMLToken (XMLToken, TokenKey?)

Status: Implemented. XML Token format is described below

d) PEP Interface for using with Token based credentials/enforcement

Status: To be defined

4.3. External/WS interface

External TVS interface will allow programming TVS table by sending particular reservation information. The message format may contain the following information:

MessageSetTVS (GRI, ResourceID, TokenValue, LRI?, TokenKey?,
NotBefore, NotOnOrAfter)

Note: In current implementation this information is expressed and communicated in a form of the AuthZ ticket.

4.4. TVS package

Although TVS package is a part of the general GAAAPI package, TVS related functions and classes are provided as a separate “**tvstest01.jar**” file

TVS related classes are organised as a **org.aaaarch.gaaapi.tvstest** package. All interfaces are supported by corresponding method of the TVS.java class.

org.aaaarch.gaaapi.tvstest package relies on a number of supporting classes from the core **org.aaaarch.gaaapi** package which are provided as the following packages:

- org.aaaarch.config**
- org.aaaarch.crypto**
- org.aaaarch.signature**
- org.aaaarch.utils**
- org.aaaarch.gaaapi.common**
- org.aaaarch.gaaapi.ticket**

Token builder functions can be accessed either via TVS.java or directly from **TokenBuilder.java** class

The package is provided with the TestTVS.java test class that allows for interactive test of all available functions.

The package requires a special supporting directories structure for storing keys, schemas and temporal files as explained below.

4.5. Usage examples

1) Generating binary token

To request token generation from the calling application, use these commands/methods:

```
byte[] org.aaaarch.gaaapi.tv.s.TokenKey.generateTokenKey(String gri) throws Exception
byte[] org.aaaarch.gaaapi.tv.s.TokenBuilder.getToken(String gri, byte[] tokenkey) throws
Exception
```

Example:

```
byte[] tokenkey = org.aaaarch.gaaapi.tv.s.TokenKey.generateTokenKey(gri);
byte[] token = org.aaaarch.gaaapi.tv.s.TokenBuilder.getToken(gri, null);
```

Note: GRI can be generated using TVS package as well:

```
String gri = "".concat(org.aaaarch.gaaapi.common.IDgenerator.generateID(20).toString());
```

2) Generating XML token

To request XML token generation from the calling application, use these commands/methods:

```
byte[] org.aaaarch.gaaapi.tv.s.TokenKey.generateTokenKey(String gri) throws Exception --
optionally
public static String org.aaaarch.gaaapi.tv.s.TokenBuilder.getXMLToken(String gri, byte[]
tokenKey) throws Exception
```

Example:

```
String tokenxml = org.aaaarch.gaaapi.tv.s.TokenBuilder.getXMLToken(gri, null);
```

3) Validating binary token

To verify binary token, use these commands/methods:

```
boolean validateToken (String token, String gri, byte[] tokenKey)
throws Exception
```

Example:

```
boolean valid = org.aaaarch.gaaapi.tv.s.TVS.validateToken (token, gri, null);
```

4) Validating XML token

To verify XMLtoken, use these command/methods:

```
boolean validateXMLToken (Document aztdoc, byte[] tokenKey)
throws Exception, MalformedTokenException, NotValidAuthzTokenException
```

```
boolean validateXMLToken (String authzToken, byte[] tokenKey)
    throws Exception, MalformedTokenException, NotValidAuthzTokenException
```

Example:

```
TvsXMLTokenType token = new org.aaaarch.gaaapi.tv.s.TvsXMLTokenType (tokendoc);
boolean timevalid = token.isTimeValid(token);
```

4.6. Configuration

TVS installation requires configuration of a few folder that contain a keystore or used as a temporal directories when processing AuthZ session credentials.

The following directories are used in current implementation but can be configured via the ConfigSecurity.java class (currently hard coded):

```
LOCAL_DIR_ROOT = "" - TVS installation directory

LOCAL_DIR_SECURITYCONFIG = LOCAL_DIR_ROOT + "data/config/"
LOCAL_DIR_KEYSTORE_TRUSTED = LOCAL_DIR_ROOT + "data/keystore/trusted/"
LOCAL_DIR_SYMKEYSTORE = LOCAL_DIR_ROOT + "data/keystore/xmlsec/symkeystore/"
LOCAL_DIR_SCHEMAS = LOCAL_DIR_ROOT + "data/schemas/"
LOCAL_DIR_AAADATA_CACHE_AZTICKETS = LOCAL_DIR_ROOT + "_aaadata/cache/aztickets/"
LOCAL_DIR_AAADATA_TMP = LOCAL_DIR_ROOT + "_aaadata/tmp/"
```

Note. Provided TVS package contains all necessary directories structure and also crypto keys. TVS shared secret is hard coded into the token building classes.

```
<install-root>
+-- data
|   +-- config
|   +-- docs
|   +-- keystore
|       +-- trusted
|       +-- xmlsec
|       +-- symkeystore
+-- tvs-lib
|   +-- endorsed
+-- x-output
+-- _aaadata
|   +-- cache
|       +-- aztickets
+-- tmp
```

4.7. Required external libraries

The list of currently used libraries is as follows (Note: this is the whole set of libraries that is required to support full suggested TVS functionality; however it can be potentially reduced):

```
bcprov-jdk15-130.jar
commons-logging-1.0.3.jar
commons-logging-api.jar
dom3-xercesImpl-2.5.0.jar
dom3-xml-apis-2.5.0.jar
jaxrpc-sec.jar
log4j-1.2.8.jar
resolver.jar
saaj-api.jar
saaj-impl.jar
soapprocessor.jar
xmldsig.jar
xmlsec-1.4.1.jar
xmlsecSamples.jar
```

```
xalan-2.6.jar
xercesImpl.jar
xml-apis.jar
```

5. Token handling model

Token handling model is based on the shared secret HMAC algorithm.

The TokenKey is generated in the following way:

$$\text{TokenKey} = \text{HMAC}(\text{GRI}, \text{tb_secret})$$

where

GRI – global reservation identifier,
tb_secret – shared Token Builder secret.

In current implementation the tb_secret is the secret phrase “** ***** ** *****” 3DES encrypted and hard coded into the Token Builder classes. (For curious people, you can dig into the source code¹.)

Token is created in a similar way but using TokenKey as a HMAC secret:

$$\text{Token} = \text{HMAC}(\text{GRI}, \text{TokenKey})$$

This algorithm allows for chaining token generation and validation process, e.g.:

“GRI-TokenKey-Token=>LRI-l-TokenKey-l-Token”

It is being investigated that current model can be replaced with the IBC (Identity Based Cryptography) that will allow to replace shared secret token handling model.

6. Key management model

Key management model is not discussed at the stage of the project. Token handling model relies on the shared secret that is installed at all participating

7. XML Schemas and examples

7.1. XML Token

XML token is used to provide a reference to already reserved/authorised resource at the access/usage stage of the reserved service or resource. Typically token use is supported by the detailed reservation information stored/cached at the resource.

XML token format uses a special “TVS/TBN” profile of the more general AuthzToken format. Example of the full TVS XML token is shown below:

¹ Or send me email with subject “TokenKey Secret”

```
<AAA:AuthzToken xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
Issuer="urn:aaa:gaaapi:token:TVS" SessionId="a9bcf23e70dc0a0cd992bd24e37404c9e1709afb"
TokenId="d1384ab54bd464d95549ee65cb172eb7">
  <AAA:TokenValue>ebd93120d4337bc3b959b2053e25ca5271a1c17e</AAA:TokenValue>
  <AAA:Conditions NotBefore="2007-08-12T16:00:29.593Z" NotOnOrAfter="2007-08-
13T16:00:29.593Z"/>
</AAA:AuthzToken>
```

where the element <TokenValue> and attributes SessionId and TokenId are mandatory, and the element <Conditions> and attributes Issuer, NotBefore, NotOnOrAfter are optional.

Minimum token format is illustrated in the following example:

```
<AAA:AuthzToken xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
SessionId="a9bcf23e70dc0a0cd992bd24e37404c9e1709afb"
TokenId="d1384ab54bd464d95549ee65cb172eb7">
  <AAA:TokenValue>ebd93120d4337bc3b959b2053e25ca5271a1c17e</AAA:TokenValue>
</AAA:AuthzToken>
```

Attributes “Issuer” allow for distinguishing different AuthzToken profiles. The TVS profile is identified by the URN "urn:aaa:gaaapi:token:TVS".

7.2. AuthzTicket

AuthzTicket is used in the current TVS implementation as a TVS programming message as mentioned in the section 4.3.

Authorisation ticket (AuthzTicket) is a part of GAAA-AuthZ framework functionality and implementation and allows for transferring full AuthZ decision and policy enforcement context between a requestor and a AuthZ service or between different security/AuthZ domains.

AuthzTicket has a complex but flexible format. One of the basic design suggestions is that AuthzTicket is easily mapped to the SAML AuthzDecision Assertion or to XACMLAuthzDecision Assertion defined by the SAML profile of XACML.

In particular use case with Token based network, the AuthzTicket is use for programming TVS and provides both reservation ID/reference and detailed information about programming TBS (token based FORCES switch).

When used together, AuthzTicket and AuthzToken share the SessionId attribute which can be either GRI or LRI (global or local reservation ID) and are cryptographically connected.

AuthzTicket must be digitally signed to keep its integrity.

5.2.1. AuthzTicket data model and schema

Figures below illustrate the AuthzTicket data model core elements and additionally the Resource elements containing all necessary information for the TBS programming. The AuthzTicket schema is available from <http://staff.science.uva.nl/~demch/projects/aaauthreach/draft-gaaa-azticket-03.xsd>

The AuthzTicket contains the following major groups of elements that allow for extended AuthZ session security context management:

- Root element attributes `TicketID`, `SessionID`, and `Issuer` that allows for the ticket unique identification and defines its binding to the session and domains related processes/authorities.
- The `Decisions/Decision` element that holds the PDP AuthZ decision bound to the requested resource or service expressed as the `ResourceID` attribute.
- The `Resources` extendable element that may hold proprietary description of the reserved resource.
- The `Actions/Action` complex element contains actions which are permitted for the Subject or its delegates.
- The `Subject` complex element contains all information related to the authenticated Subject who obtained permission to do the actions, including sub-elements: `Role` (holding subject's capabilities), `SubjectConfirmationData` (typically holding AuthN context), and extendable sub-element `SubjectContext` that may provide additional security or session related information, e.g. Subject's VO, project, or federation.
- The `Delegation` element allows to delegate the capabilities defined by the AuthzTicket to another Subjects or community. The attributes define restriction on type and depth of delegation
- The `Conditions` element specifies the validity constrains for the ticket, including validity time and AuthZ session identification and additionally context. The extensible `ConditionAuthzSession` element provides rich possibilities for AuthZ context expression.
- The `Obligations/Obligation` element can hold obligations that PEP/Resource should perform in conjunction with the current PDP decision.

The semantics of AuthzTicket elements is defined in such a way that allows easy mapping to related elements in SAML and XACML. First three elements the `Decision`, the `Actions/Action`, and the `Subject` have direct mapping to related SAML elements. Other AuthzTicket elements the `Delegation`, the `Conditions`, and the `Obligations/Obligation` element, which is originated from XACML, can be implemented as extensible element of the SAML `Condition` element.

In a typical GAAA-AuthZ operation the AuthzTicket is digitally signed (as shown in example) and cached by the Resource's AuthZ service. To reduce communication overhead when using AzTicket for consecutive requests validation, the associated AuthzToken can be generated of the AuthzTicket, wherein `TokenID` = `TicketID` and `TokenValue` = `SignatureValue`, what is sufficient for identification of the cached AuthzTicket.

Note. For the TBN use case, the `SessionID` attribute can hold GRI, additional session or domains related information can be put into the extensible `ConditionAuthzSession` element or into the proprietary structured the `Resource` element.

Diagrams Fig. 5.1 and 5.2 below illustrate the top and main AuthzTicket elements and the Resource element structure for the FORCES token-based switch. Note, the Resource element is defined as an extendable in the AuthzTicket schema to incorporate different types of target resources (to which AuthZ decision is applied).

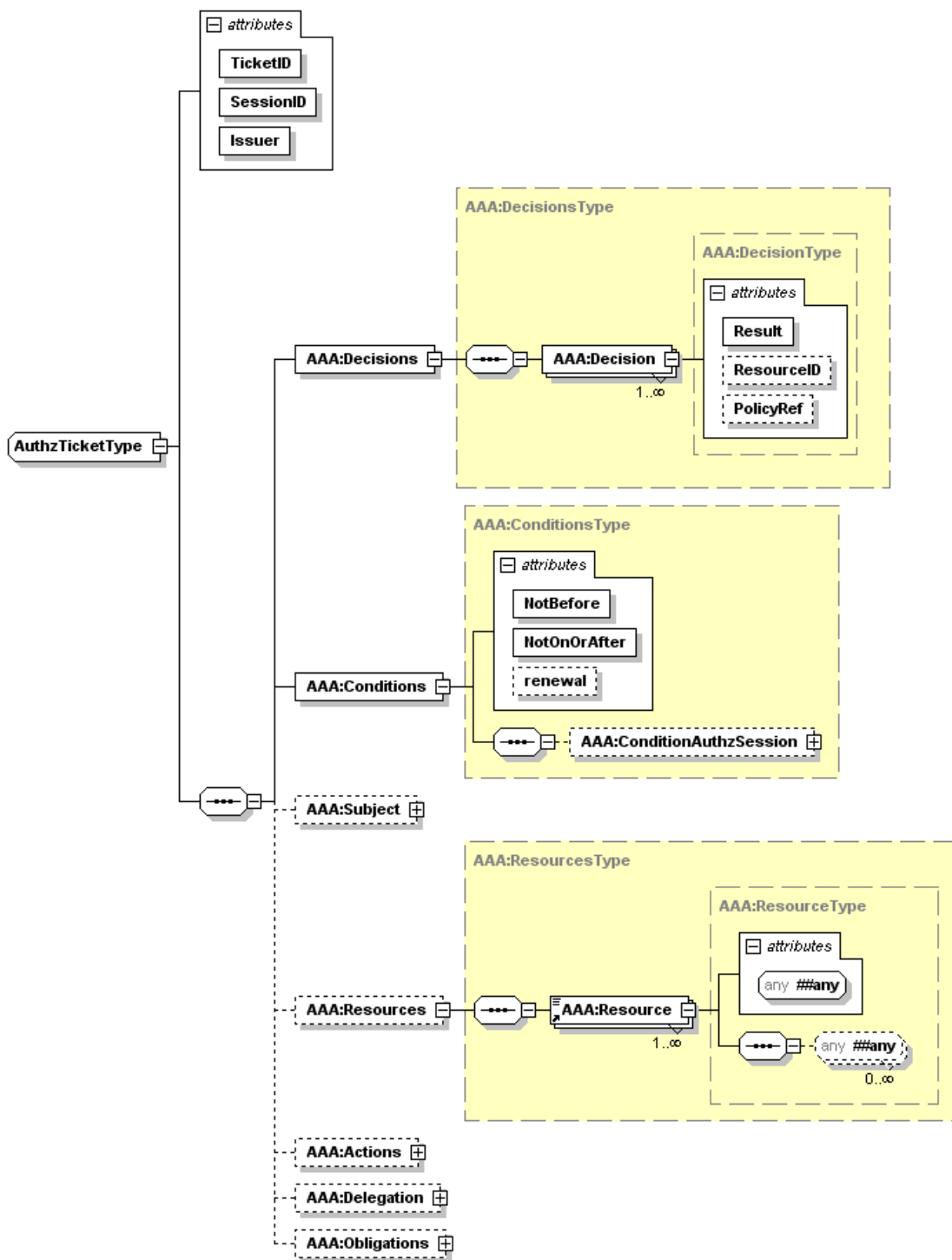


Fig. 5.1 AuthzTicket top and main elements (note, the Resource element is defined as an extendable).

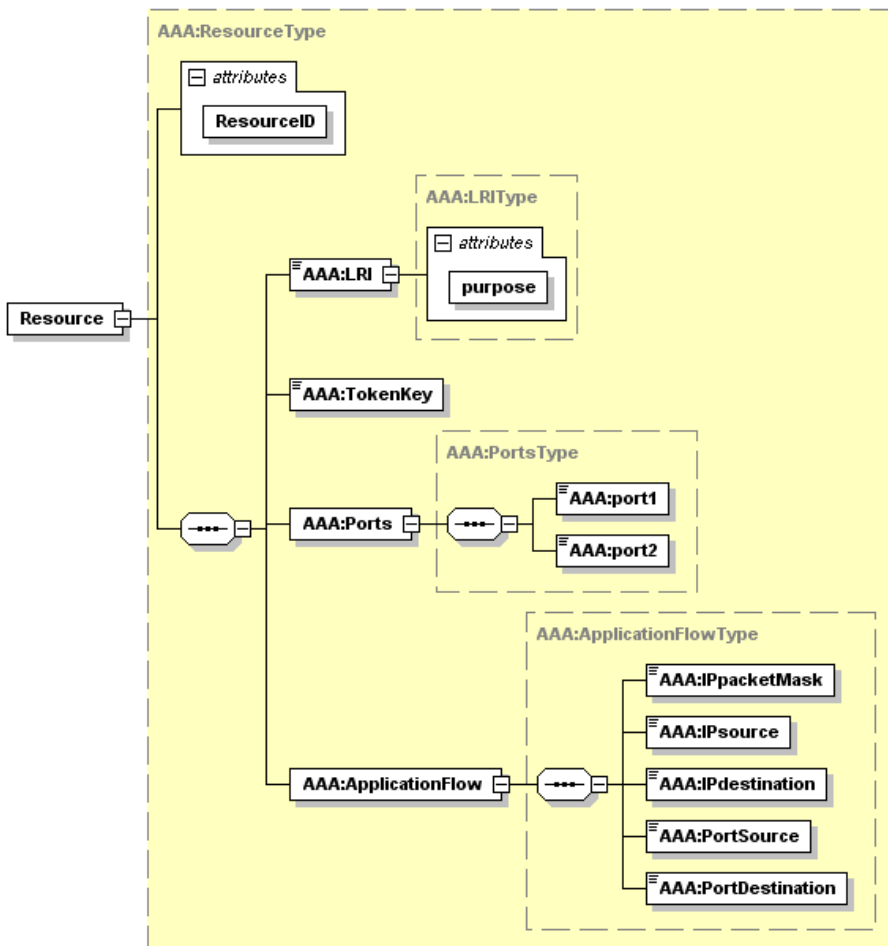


Fig 5.2. The Resource element structure for the FORCES token-based switch (The TBS profile of AuthzTicket schema is available from <http://staff.science.uva.nl/~demch/projects/aaauthreach/draft-gaaa-azticket-03-tbs.xsd>)

5.2.2. AuthzTicket example

The following is an example with the Resource element as required for TBS programming generated with the GAAAPI package.

```

<AAA:AuthzTicket xmlns:AAA="http://www.aaauthreach.org/ns/#AAA" Issuer="urn:cnl:trust:tickauth:pep"
TicketID="cba06d1a9df148cf4200ef8f3e4fd2b3" SessionID=" f8f3e4fd2b3 df148cf4200">
  <AAA:Decisions><AAA:Decision ResourceID="http://resources.collaboratory.nl/Philips_XPS1"
result="Permit"/>
</AAA:Decisions>
<!-- SAML mapping: <AuthorizationDecisionStatement Decision="*" PolicyRef="*" Resource="*"> -->
<AAA:Resources><AAA:Resource ResourceID="http://www.aaaarch.org/TBS/ForceG">
  <AAA:LRI purpose="String">text</AAA:LRI>
  <AAA:TokenKey>String</AAA:TokenKey>
  <AAA:Ports>
    <AAA:port1>String</AAA:port1>
    <AAA:port2>String</AAA:port2>
  </AAA:Ports>
  <AAA:ApplicationFlow>
    <AAA:IPpacketMask>String</AAA:IPpacketMask>
    <AAA:IPsource>String</AAA:IPsource>
    <AAA:IPdestination>String</AAA:IPdestination>
    <AAA:PortSource>String</AAA:PortSource>
    <AAA:PortDestination>String</AAA:PortDestination>
  </AAA:ApplicationFlow>
</AAA:Resource></AAA:Resources>
<AAA:Actions>
  <AAA:Action>cnl:actions:CtrlInstr</AAA:Action>
  <!-- SAML mapping: <Action> -->

```

```

<AAA:Action>cnl:actions:CtrlExper</AAA:Action>
</AAA:Actions>
<AAA:Subject Id="subject">
  <AAA:SubjectID>WHO740@users.collaboratory.nl</AAA:SubjectID>
  <!-- SAML mapping: <Subject>/<NameIdentifier> -->
  <AAA:SubjectConfirmationData>
    IGhAllvwa8YQomTgB9Ege9JRNnld84AggaDkOb5WW4U=</AAA:SubjectConfirmationData>
  <!-- SAML mapping: EXTENDED <SubjectConfirmationData/> -->
  <AAA:Role>analyst</AAA:Role>
  <!-- SAML mapping:
    <Evidence>/<Assertion>/<AttributeStatement>/<Assertion>/<Attribute>/<AttributeValue> -->
  <AAA:SubjectContext>CNL2-XPS1-2005-02-02</AAA:SubjectContext>
  <!-- SAML mapping:
    <Evidence>/<Assertion>/<AttributeStatement>/<Assertion>/<Attribute>/<AttributeValue> -->
</AAA:Subject>
<AAA:Delegation MaxDelegationDepth="3" restriction="subjects">
<!-- SAML mapping: LIMITED <AudienceRestrictionCondition> (SAML1.1),
    or <ProxyRestriction>/<Audience> (SAML2.0) -->
  <AAA:DelegationSubjects>
    <AAA:SubjectID>team-member-2</AAA:SubjectID>
    <AAA:SubjectID>team-member-1</AAA:SubjectID>
  </AAA:DelegationSubjects>
</AAA:Delegation>
<AAA:Conditions NotBefore="2006-06-08T12:59:29.912Z"
  NotOnOrAfter="2006-06-09T12:59:29.912Z" renewal="no">
<!-- SAML mapping: <Conditions NotBefore="*" NotOnOrAfter="*"> -->
  <AAA:ConditionAuthzSession PolicyRef="PolicyRef-GAAA-RBAC-test001" SessionID="JobXPS1-2006-001">
  <!-- SAML mapping: EXTENDED <SAMLConditionAuthzSession PolicyRef="*" SessionID="*"> -->
    <AAA:SessionData>put-session-data-Ctx-here</AAA:SessionData>
  <!-- SAML mapping: EXTENDED <SessionData/> -->
  </AAA:ConditionAuthzSession>
</AAA:Conditions>
<AAA:Obligations>
  <AAA:Obligation>put-policy-obligation(2)-here</AAA:Obligation>
  <!-- SAML mapping: EXTENDED <Advice>/<PolicyObligation> -->
  <AAA:Obligation>put-policy-obligation(1)-here</AAA:Obligation>
</AAA:Obligations>
</AAA:AuthzTicket>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo> ... </ds:SignedInfo>
  <ds:SignatureValue>e4E27kNwEXoVdnXIBpGVjpaBGVY71Nypos...</ds:SignatureValue>
</ds:Signature>

```